D2.2

REWIRE Operational Landscape, Requirements and Reference Architecture - Final version

Project number:	101070627
Project acronym:	**REWIRE**
Project title:	REWiring the ComposItional Security VeRification and AssurancE of Systems of Systems Lifecycle
Project Start Date:	1st October, 2022
Duration:	36 months
Programme:	HORIZON-CL3-2021-CS-01
Deliverable Type:	Report
Reference Number:	HORIZON-CL3-2021-CS-01-101070627/ D2.2 / v1.0
Workpackage:	WP2
Actual Submission Date:	27th May, 2025
Responsible Organisation:	UCL
Editor:	Corentin Verhamme, Francois Koeune
Dissemination Level:	Public
Revision:	v1.0
Abstract:	Deliverable 2.2 provides the final version of the REWIRE Reference Architecture comprising an open, modular, highly portable and RoT vendor agnostic security stack that exposes advanced trust, attestation and formal verification services for more robust security guarantees on the trust state of a service-graph-chain. Details on the novel composition of design-time and runtime assurances for the secure lifecycle management of Systems-of-Systems (SoS) are provided. We also define the policy expression language that is expressive enough to be able to cope with the different type of operations that need to be enforced (during runtime) as a reaction to a change in the trust level of a device.
Keywords:	Attestation, Formal Verification, Evidence-based Trust Assessment, Trusted Computing Base, Threats

The The project REWIRE has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101070627.

Editor

Corentin Verhamme, Francois Koeune(UCL)

Contributors (ordered according to beneficiary numbers)

Stylianos Basagiannis, Simone Fulvio Rollini (UTRCI)
Dimitris Karras, Thanasis Giannetsos, Sofianna Menesidou, Panagiotis Banavos (UBITECH)
Corentin Verhamme, Francois Koeune (UCL)
Sjors Van den Elzen (SECURA)
Samira Briongos, Javier de Vicente Gutierrez (NEC)
Annika Wilde, Ghassan Karame (Bochum University)
Spiros Kousouris, Erifili Ichtiaroglou (S5)
Konstantinos Lykos, Ioannis Tsesmelis, Manolis Sordakis, Ilias Aliferis (UNIS)
Jesus Sanchez, Francisco Javier Romero Martìnez (ODINS)
Christiana Kyperounta (8BELSS)
Kaitai Liang, Zeshun Shi (TUD)
Yalan Wang, Liqun Chen (SURREY)

Disclaimer

Executive Summary

A core offering of REWIRE entails the provision of the necessary *security and trust enablers for enhancing the secure lifecycle management of embedded systems considering the trend towards the adoption of OpenHW architectures, such as RISC-V*. Thus, REWIRE aims to address the key challenge of ensuring the correctness of software and hardware artefacts for achieving high security and trustworthiness guarantees for embedded systems. To this end, REWIRE aims to provide a novel composition of **design-time** and **runtime** assurances to achieve the requirements characterizing such systems in an efficient manner.

In this deliverable, we first provide the updated and finalized definition of the REWIRE reference architecture and functional components, including the final categorization of actions into the **Design-time phase** and **Runtime phase**. A central part of the Design-time phase is the **Compositional Verification and Validation** component, whose target is the definition of the **trust boundary** of the system, i.e., the identification and formal verification of the security processes and enablers that can be performed during design-time, and the definition of the properties that need to be attested dynamically during runtime, considering the requirements provided by the Manufacturer, security administrator, and use case providers. In addition, a core output of this phase is the construction of **security and operational policies** specifying the actions that need to be performed by the devices for providing lifetime assurance.

After the Design-time phase is complete, the **Runtime phase** of the REWIRE framework is initiated. First, the **Zero-Touch Onboarding (ZTO)** scheme is performed for the generation of the cryptographic material for the execution of the security enablers of REWIRE, leveraging the **Attribute-Based Signcryption (ABSC)** scheme for enabling devices to provide verifiable proof of ownership of their attributes. Then, the **Runtime phase** entails the execution of the security enablers specified by the policies, specifically the **attestation enablers**, the **secure software update process**, and the **live migration process**. For the management of keys required by these enablers, REWIRE implements a **Key Management System** as a Secure Monitor Extension in the context of Keystone. In addition, the **Blockchain Infrastructure** is used for the data management of attestation- and application-related data, and the **Secure Oracle Layer** is responsible for ensuring the veracity of the data recorded on the ledger. Note that this is instantiated over the **HyperLedger Besu** technology paired with the SGX-based **Town Crier (TC)** as an authenticated bridge for the execution of smart contracts, as well as **Fabric Private Chaincode (FPC)** for the execution of chaincode within an SGX enclave. In this deliverable, we also provide a detailed overview of the most prominent threats targeting Blockchain Infrastructures, as well as the associated mitigation measures. Finally, the **AI-based Misbehaviour Detection Engine** of REWIRE is implemented as an additional Trust Source for assisting a potential trust assessment of the system, and has been deployed across the **Smart Cities** and **Smart Satellites** use cases. In this deliverable, we focus on the former, and we detail the manner with which this component aims to identify and address the core challenges arising within urban environments, with a focus on architectural research infrastructures.

In addition, for the better orchestration of the novel security and trust extensions offered by REWIRE, as part of this deliverable, REWIRE is *one of the first projects of its kind to provide a concrete mapping of the trust extensions and enablers with the evidence that can be monitored and attested during runtime*, in order to achieve the required level of assurance. Towards this direction, an exhaustive threat profiling is put forth, considering threats and vulnerabilities at various layers of the compute continuum. Then, taking into consideration the **likelihood** and **impact** of these threats, REWIRE documents the list of evidence that, if monitored, can provide guarantees on the security monitors deployed and can act as *security dots*

that can be connected in order to complete the assurance puzzle. For performing the threat profiling, we consider an adversarial model that categorises threats depending on the approach of the potential adversary into **host-based attacks**, **network-based attacks**, and **physical/algorithmic-based attacks** (including consideration of **Post-Quantum** threats). Then, we provide a detailed lists of threats for each core component of REWIRE, including the properties affected by each threat and the security measures that can be used for mitigating those threats. We also perform a correspondence between the types of available evidence, the mitigation measures, and the associated threats.

Another core focus of this deliverable is the definition of the **policy expression language** of REWIRE, which is used for the definition of the security and operational policies, and is expressive and granular enough to capture both the **operational policies** for performing reactive actions such as secure software updates or live migration, and **security policies** to dictate the triggering of attestation services. We provide the motivation behind the selection of the **Medium-level Security Policy Language (MSPL)** with regards to its capability to fulfil the expressiveness requirements of REWIRE. Then, we provide detailed definitions of the policy structure in the context of REWIRE, their positioning within the overall framework, and we demonstrate their construction through practical examples instantiated in the REWIRE action workflow.

Contents

List of Figures

List of Tables

Versioning and contribution history

Version	Date	Author	Notes
v0.1	22.11.2024	Table of Contents & Allocation of tasks to the partners	Francois Koeune (UCL), Thanassis Giannetsos (UBI)
v0.15	27.11.2024	Listing of the updates regarding the operation of hooks (Chapter 2)	Sjors Van den Elzen (SECURA)
v0.2	05.12.2024	Listing of the updates on the Compositional Verification and Validation component (Chapter 2)	Stylianos Basagiannis, Simone Fulvio Rollini (UTRCI)
v0.25	11.12.2024	Listing of the updates on the Policy-Compliant Blockchain Infrastructure (Chapter 2)	Kaitai Liang, Zeshun Shi (TUD), Dimitris Karras, Panagiotis Banavos (UBI)
v0.3	19.12.2024	Description in the state-of-the-art in policy enforcement, MSPL Vocabulary, policy positioning in REWIRE (Chapter 6)	Dimitris Karras, Thanassis Giannetsos (UBI) Stylianos Basagiannis, Simone Fulvio Rollini (UTRCI)
v0.35	09.01.2025	Description of REWIRE policy structure using MSPL, definition of policy models for attestation and security update policies (Chapter 7)	Dimitris Karras, Thanassis Giannetsos (UBI) Stylianos Basagiannis, Simone Fulvio Rollini (UTRCI)
v0.37	17.01.2025	Definition of adversarial model and assumptions, threats against Zero-Touch Onboarding (Chapter 4)	Jodie Knapp, Liqun Chen (SURREY)
v0.4	23.01.2025	Definition of threat landscape for Blockchain Infrastructure considering external operations (Chapter 4)	Kaitai Liang, Zeshun Shi (TUD), Dimitris Karras, Panagiotis Banavos (UBI), Samira Briongos, Javier de Vicente Gutierrez (NEC)
v0.45	31.01.2025	Description of Secure Oracle Layer, expansion of threat landscape considering internal operations, mitigation measures (Chapter 5)	Kaitai Liang, Zeshun Shi (TUD), Dimitris Karras, Panagiotis Banavos (UBI)
v0.5	10.02.2025	Compilation of list of updated REWIRE framework requirements (Chapter 3)	Sofianna Menesidou (Chapter 3)
v0.53	17.02.2025	Definition of threats against the attestation schemes of REWIRE (Chapter 4)	Dimitris Karras, Thanassis Giannetsos (UBI), Samira Briongos, Javier de Vicente Gutierrez (NEC)
v0.55	25.02.2025	Definition of the threat landscape against the AI-based Misbehaviour Detection Engine (Chapter 4)	Spiros Kousouris, Erifili Ichtiaroglou (S5)
v0.6	18.03.2025	Listing of threat landscape against the SW update protocol and the live migration protocol (Chapter 4)	Corentin Verhamme, Francois Koeune (UCL), Samira Briongos, Javier de Vicente Gutierrez (NEC)
v0.65	26.03.2025	Analysis based on classes of attacks and categorization based on types of evidence collected (Chapter 4)	Samira Briongos (NEC), Dimitris Karras, Thanassis Giannetsos (UBI)
v0.7	02.04.2025	Description of the instantiation of the AI-based Misbehaviour Detection Engine in the context of the REWIRE Smart Cities UC (Chapter 8)	Spiros Kousouris, Erifili Ichtiaroglou (S5), Jesus Sanchez (ODINS)
v0.75	15.04.2025	Construction of the plausibility checks for the Smart Cities UC (Chapter 8)	Spiros Kousouris, Erifili Ichtiaroglou (S5), Jesus Sanchez (ODINS)
v0.8	23.04.2025	Finalization of the Executive Summary, Introduction, and Conclusions (Chapters 1, 9)	Dimitris Karras (UBI)
v0.85	01.05.2025	Details on the REWIRE approach on HW-based extensions	Samira Briongos, Javier de Vicente Gutierrez (NEC)
v0.9	07.05.2025	Internal Review	Corentin Verhamme, Francois Koeune (UCL)

v0.95	12.05.2025	Changes based on Internal Review	Dimitris Karras, Thanassis Giannetsos (UBI)
v1.0	27.05.2025	Final submission of deliverable	Dimitris Karras, Thanassis Giannetsos (UBI)

List of Abbreviations

Abbreviation	Translation
ABE	Attribute-Based Encryption
ABS	Attribute-Based Access
ACC	Assisted Cruise Control
AE	Authenticated Encryption
AIC	Attestation Integrity Verification
AIK	Attestation Identity Key
API	Application Programming Interface
ATL	Actual Trust Level
BBL	Berkeley Boot Loader
BIOS	Basic Input/Output System
CDI	Compound Device Identifier
CIV	Configuration Integrity Verification
CPU	Central Processing Unit
CRL	Certificate Revocation List
CRTM	Core Root of Trust for Measurement
DDA	Direct Anonymous Attestation
DICE	Device Identifier Composition Engine
DID	Decentralised Identity Documents
DMA	Direct Memory Access
EAP	Enhanced Authentication Protocol
ECU	Electronic Control Units
EK	Endorsement Key
EPC	Enclave Page Cache
FPGA	Field Programmable Gate Arrays
FSM	Finite State Machine
HSM	Hardware Security Module
HW	Hardware
IDM	Identity Management
IOMMU	Input/Output Memory Management Unit
ISA	Instruction Set Architecture
ISO	International Organization for Standardization
KMS	Key management system
MUD	Manufacturer Usage Description
OEM	Original Equipment Manufacturer
PCR	Platform Configuration Register
PMP	Physical Memory Protection
PPI	Physical Presence Interface
PSK	Pre-Shared Key
PUF	Physically Unclonable Function

RA	Rich Application
RATS	Remote ATtestation procedureS
REE	Rich Execution Environment
RoT	Root of Trust
RT	Runtime
SBI	Supervisor Binary Interface
SC	Side-Channel
SDLC	Software Development Lifecycle
SGX	Software Guard Extensions
SM	Security Monitor
SoC	System-on-Chip
SP	Service Provider
SRK	Storage Root Key
SUIT	Software Updates for Internet of Things
SSA	Security-Sensitive Application
SSI	Self-Sovereign Identity
SW	Software
TA	Trusted Application
TCB	Trusted Computing Base
TCG	Trusted Computing Group
TDISP	TEE Device Interface Protocol
TEE	Trusted Execution Environment
TMFS	TEE Management Framework Specification
TOCTOU	Time of Check Time of Use
TPM	Trusted Platform Module
TTP	Trusted Third Parties
UDS	Unique Device Secret
VC	Verifiable Credentials
VM	Virtual Machine
VP	Verifiable Presentations
vTPM	Virtual Trusted Platform Module
ZTO	Zero-touch onboarding

Chapter 1

Introduction

One of the main targets of REWIRE is to *enhance the secure lifecycle management and guide the calculation of the trustworthiness level of embedded systems featuring RISC-V architectures*, operating in the context of large-scale **Systems-of-Systems (SoS)**, which can be seen as a type of **Multi-Agent Systems (MAS)**. In this context, trust and trustworthiness are key constructs underpinning interactions among entities that often operate without a central authority. While trust is instrumental towards decision-making in such environments, helping entities (agents) select collaborators, assess the trustworthiness of incoming information, and mitigate threats from untrustworthy or hostile entities, it is usually confined to system integrity and dependability aspects, and excludes other important dimensions, such as privacy, authenticity, availability, and robustness of the participating agents and their actions.

In general, the notion of trust is expressed as a function of a triplet of metrics, namely (i) **belief** referring to the degree of confidence that an entity (device) will act as expected, (ii) **disbelief**, meaning the degree of confidence that the entity will act contrary to expectations, and (iii) **uncertainty**, reflecting the degree of lack of knowledge or evidence for the determination of the trustworthiness level of the device. Determining the level of trust in a device based on this approach requires the collection of **trustworthiness evidence** from the available set of **Trust Sources**. However, as aforementioned, trust management in the aforementioned types of SoS and MAS needs to consider the fulfilment of various types of security properties, depending on the needs and requirements of the respective domain.

Considering the above, REWIRE aims to address the key challenge of ensuring the correctness of all software and hardware artefacts and achieve high security and trustworthiness profiles for embedded systems, by providing a set of **security enablers and trust extensions that are able to provide trustworthiness evidence in a verifiable manner**, in order to unlock the trust characterization of the system. Another core innovation supporting this goal is the **symbiosis of design-time with runtime assurances** as part of a hybrid methodology, by demonstrating how these technologies can work in symbiosis with runtime enablers to achieve the stringent **security and privacy requirements** needed for achieving the extended notion of trust in REWIRE, which goes beyond integrity and extends to properties such as correctness, robustness, certifiability, auditability, etc.

In this regard, a core innovation of REWIRE entails the provision of a **Compositional Verification and Validation** methodology and the definition of actions that need to be performed during the design-time phase pertaining to the **formal Verification of protocols and schemes**, before they are synthesized into the hardware configuration to be deployed to the device. This includes the identification of the **trust boundary** of the device during the design phase, i.e., the set of properties of the device that can be formally verified during design time and do not need to be attested during runtime. Conversely, the properties that need to be attested and verified during runtime are **abstracted**. This means that, in order to reduce the complexity and enable the convergence of the employed Formal Verification protocols, the properties that change during the execution of software processes on the device are excluded from the Formal Verification process, and instead need to be dynamically attested during runtime using the security and attestation enablers of REWIRE.

However, when attesting these properties during runtime, serious efficiency challenges may be faced by the devices that may need to verify a large part of the software stack, particularly with regard to **tracing**, as the employed attestation schemes and protocols may require the collection of a large amount of traces as attestation evidence. In this deliverable, building upon what has been documented in D2.1 [21], we **confine the design space of system properties that need to be attested and verified during runtime**, thus alleviating such efficiency challenges and enabling the timely execution of such attestation operations given the available resources on such devices.

To summarize, the core motivation and one of the core driving pillars behind the REWIRE architecture is *how to overcome the challenges of different runtime remote attestation and verification capabilities to achieve the fulfilment of the zero-trust paradigm*. The basic challenge is that, for these mechanisms, there is not yet a well-defined methodology for identifying what is the **minimum set of runtime device properties** that, if verified, can provide guarantees for the functional correctness of the system. Thus, REWIRE aims to achieve the symbiosis and convergence of **design-time** Compositional Verification and Validation capabilities with **an extended set of runtime Trusted Computing (TC) security enablers** required for achieving trustworthiness in an efficient manner.

1.1 Scope and Purpose

Realizing the ultimate goal of REWIRE pertaining to the holistic protection of embedded systems in line with the notion of zero-trust, in this deliverable, we provide the final version of the REWIRE architecture, security framework, and trust extensions. In this regard, this deliverable provides a **final and clear distinction between the Design-time and the Runtime phase**, in order to clarify the symbiosis of the compositional Design-time verification and verification with runtime attestation, i.e., the unique feature offered by REWIRE.

To this end, for the Design phase, we have captured the output that can be converted to **security policies** to be enforced for the runtime attestation of the devices, as well as the device configurations to be synthesized to the devices. In the Runtime phase, we have provided the final versions of all the protocols that prepare a device to **establish its trust**, to **calculate its trust level**, and to **re-establish its trust**. Finally, we capture the common denominator with the core innovation of REWIRE, which is the description of the design of the latest version of the **REWIRE Trusted Computing Base (TCB)**, including all extensions to support the secure lifecycle management (e.g., attestation, secure software update, live migration). The aforementioned processes function in tandem with the **Blockchain Infrastructure** for the construction of a **security hygiene marketplace** that is able to store attestation evidence in an auditable and certifiable manner.

Note that all the aforementioned are aligned with the design boundary of REWIRE regarding the degree that it is based on the underlying hardware extensions to support its ownn Trusted Computing (TC) enablers. This means that, as we show and elaborate in Section 2.7, even though the REWIRE implementation instantiates and evaluates the applicability of the provided TC enablers in **OpenHW** and **RISC-V environments**, *it is only based on the HW-rooted security capabilities that are exposed at a hypervisor security level*. Thus, REWIRE does not require any updates or refinements in the underlying instruction sets for supporting its runtime trust assessment and verification capabilities, but it uses HW-rooted security capabilities that are already exposed and provided by the Security Monitor.

In this regard, REWIRE is one of the first project of its kind to investigate the feasibility of **DICE-like process verification capabilities in the context of RISC-V architectures**, such as operations related to device identity management (e.g., support of runtime verifications, software updates). To this end, REWIRE introduces a new SBI call, while still leveraging the information and functionalities already exposed by the Security Monitor, and does not go beyond this level and does not require any updates or refinements in the ISA extensions or reconfigurations of the underlying HW-based structures. This

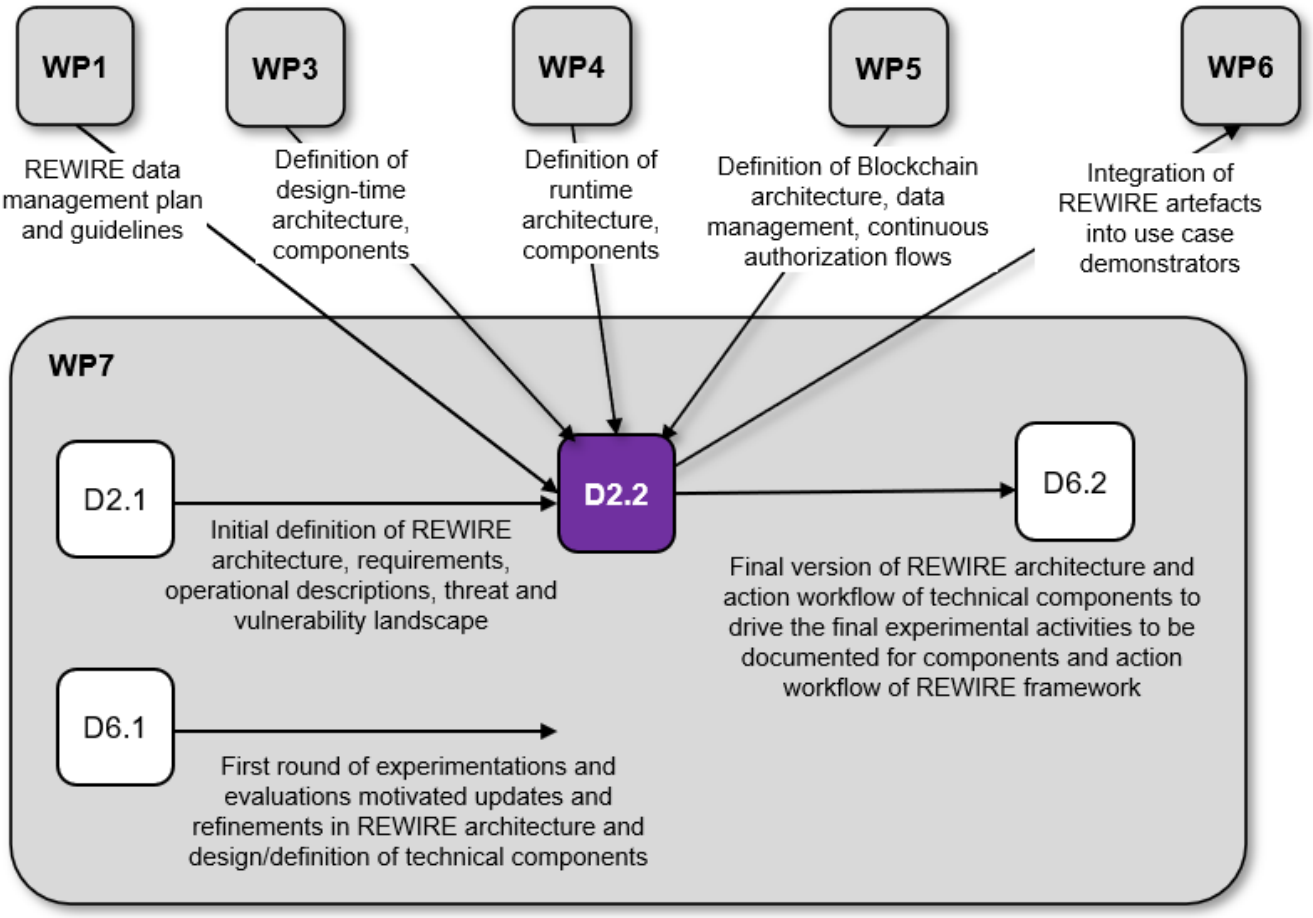

Figure 1.1: Relation of D2.2 with other WPs and Deliverables

provides evidence for the general applicability of REWIRE that overcomes the fragmentation challenges posed by the discrete security elements.

1.2 Relation to other WPs and Deliverables

Figure 1.1 depicts the relationships of this deliverable with other Work Packages (WPs), as well as other tasks in the same WP(2). As aforementioned, the main purpose of this deliverable is to expand upon the initial reference architecture defined in the previous version of this deliverable, D2.1, and document the final version of the reference architecture of REWIRE, taking into account all updates in the core technical artefacts of ENTRUST, considering also the outcomes of the experimental activities carried out in the context of D6.1. Thus, this deliverable receives input from the technical deliverables where these artefacts were documented, in the context of WP3 for the Design-time phase and WP4 for the runtime phase, and WP5 for processes centered around the REWIRE Blockchain Infrastructure.

In addition, D2.2 expands upon the definition of the threat model of D2.1, including specific threats affecting the core technical artefacts of REWIRE in both core architectural phases. D2.2 also provides the foundation against which the experimental activities of D6.2 will be performed, considering the final versions of the REWIRE technical components and action workflow.

1.3 Deliverable Structure

This deliverable is structured as follows: **Chapter 2** is dedicated to the final definition of the REWIRE conceptual architecture, as well as the core functional components, their positioning within the architec-

ture, and the corresponding action workflows. **Chapter 3** outlines the updates in the requirements of the REWIRE framework, corresponding to the practical needs of organizations and environments targeted by REWIRE. **Chapter 4** provides a comprehensive list of the most prominent threats per component, as well as the types of evidence and mitigation measures corresponding to each threat. **Chapter 5** details the Secure Oracle layer of REWIRE, focused on security concerns to be addressed, and the threat analysis of the Secure Oracle and data veracity. **Chapter 6** describes the policy expression language of REWIRE, and the motivation behind the selection of MSPL. **Chapter 7** provides details on the policy modelling process of REWIRE, including detailed descriptions of the policy structure and examples of policy definitions. **Chapter 8** is focused on the AI-based Misbehaviour Detection Engine of REWIRE, focused on its application in the Smart Cities use case. **Chapter 9** concludes the deliverable.

Chapter 2

REWIRE Conceptual Architecture and Functional Components

Here, we provide a detailed description of the **final version of the REWIRE architectural flow**, featuring the **finalized distinction between the operations included in the Design-time and Runtime phases**. In this regard, we also provide completed descriptions of the features that had not yet been finalized in D2.1 [21], as well as updates and refinements in the experimentation flows given and put forth in D6.1 [20]. The main example in this context is the updated version of the REWIRE TCB pertaining to the execution of the software update process, due to the engineering capabilities and intricacies of the OpenHW devices. In other words, REWIRE had to adopt updates in the **communication between the trusted and untrusted world** and the number of internal components to be able to operate in RISC-V environments, considering the hardware limitations of such boards.

Considering all the above, in order to better address the aforementioned challenges, in Table 2.1 we provide the list of updates on the initial conceptual architecture and functional components defined in D2.1 [21]. Throughout this section, we provide a detailed description of the updated conceptual architecture of REWIRE, and we expand upon and provide detailed descriptions of all the updates listed in the Table.

Update	Description
Definition of the Policy Orchestrator	In D2.1 [21], particular focus had been placed on the formal verification of the LRBC scheme. In the updated architecture documented in this deliverable, we provide a more concrete definition of the Policy Orchestrator, which captures the security policies as constructed and outputted by the Compositional Verification and Validation component. These are expressed using the MSPL policy language, and capture aspects such as the properties to be attested, as well as the frequency of actions to be performed (e.g., attestation, secure update, live migration). Note that further details on the selection of MSPL as the policy language of REWIRE are provided in Chapter 6.
Enrichment of Formal Verification capabilities	In D2.1 [21], we had captured the models for the verification of the functional correctness of a protocol before it is synthesized into the hardware configuration of a device. In this deliverable, as will be shown in Section 2.1.2, we extend this and provide enriched formal verification capabilities and models to capture the verification of the security properties that the trust enablers of REWIRE can offer during runtime, particularly with regard to the Configuration Integrity Verification (CIV) scheme of REWIRE. In this regard, the core innovation of REWIRE is that it offers one of the first attempts in the literature to formally verify the concept of implicit attestation with local key restriction usage policies, without the need for the presence of a remote Verifier.
Refinement of runtime Trusted Computing Base (TCB)	In this deliverable, as will be shown in Section 2.3.3, we provide refined and updated sequence diagrams capturing the actions for enabling secure software updates, based on some bugs that were identified during the implementation process. This is related to enclave management when updating the software stack of the enclave, and is related to the intrinsic characteristics of Keystone, as documented in D6.1 [20].

Live Migration	In this deliverable, we put forth the concrete set of actions needed for the live migration process, which refers to the verifiable state management and transition of an enclavized application from one device to a different one. In general, trust establishment in a device can be categorized into three core phases, namely the establishment, maintenance, and re-establishment of trust. Live migration refers to the latter category, as it is related to the re-establishment of trust following issues that may have been identified by the security and attestation enablers of REWIRE During runtime, which have also been finalized and integrated into the REWIRE architecture.
Key Management Layer of REWIRE TCB	In D2.1 [21], it had been documented that the cryptographic keys required for the execution of various security enablers were stored in their respective enclaves. Here, we provide an enhancement on the REWIRE TCB with the concept of a harmonized key management layer, exposing interfaces that allow the enclaves to manage their keys as part of the Security Monitor (SM). Thus, in Section 2.6.1, we provide details on the design and implementation of the key management layer, as well as the functions exposed for enabling the execution of security enablers.
Policy-compliant Blockchain Infrastructure	In D2.1 [21], we had documented the use of Town Crier (TC) for supporting data veracity of application-related data, while we had also documented the use of the Fabric Private Chaincode (FPC) technology for capturing the trustworthiness of evidence. Here, in order to create a harmonized Secure Oracle layer, we activate the communication between these two aspects, and we investigate the enrichment of this layer in order to enable the veracity of both attestation- and application-related data. In this regard, for facilitating this communication and improving the integration of TC with a TEE-based Blockchain infrastructure, we are currently investigating the Phala Network as a possible alternative to FPC, as it is an EVM-compatible Blockchain technology that executes smart contracts within a Trusted Execution Environment (TEE), similarly to FPC. The motivation behind the selection of Phala is the capability to integrate various cryptographic enablers, which may have been challenging to perform in FPC.

Table 2.1: Updates on the REWIRE components and conceptual architecture

2.1 Design Phase Architecture

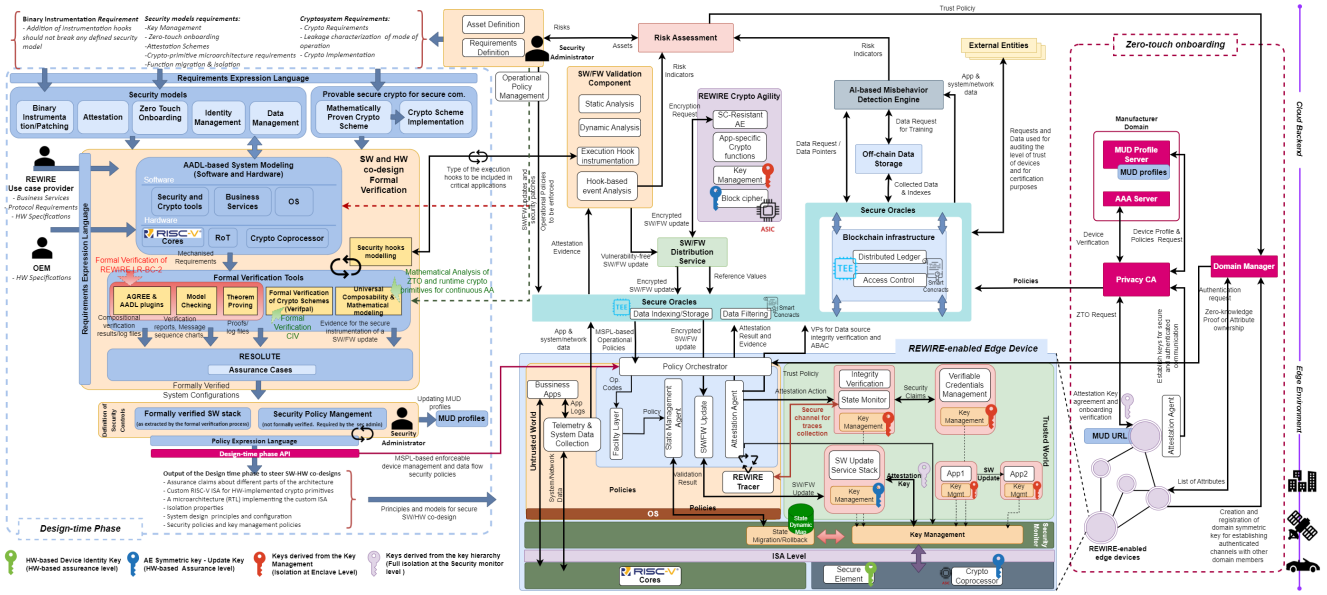

Figure 2.1: REWIRE Conceptual Architecture

The design phase of the REWIRE architecture entails the initial instantiation of the REWIRE framework during the initial phase of its deployment, as well as the expression of the overarching requirements of the system by the **Security Administrator**, the **Use Case Providers**, and the **OEM**. The goal of the design phase of the REWIRE framework is essentially the identification of the **trust boundary** of the system, i.e., the part of the system that can be considered trustworthy, for which the Security Administrator can

have the necessary guarantees that it will behave as expected. Parts of the system that remain outside this trust boundary cannot be formally verified during design-time, and need to be attested during runtime based on the attestation of the respective properties linked to the correct and expected behaviour of the device. The actions comprising the Design-time phase of REWIRE are depicted in the sequence diagram of Figure 2.2 can be summarized as follows:

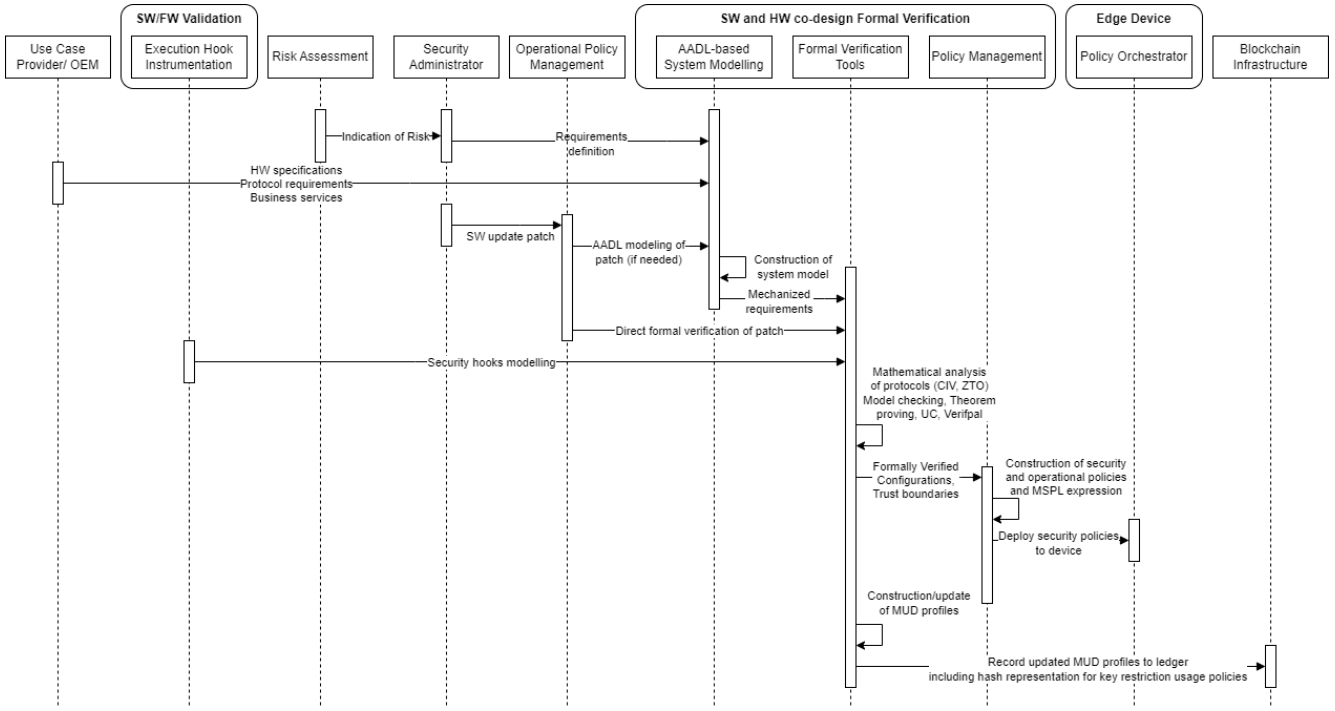

Figure 2.2: Sequence diagram of the design phase of REWIRE

2.1.1 Risk Assessment and Extraction of Security and Privacy Requirements for Data Transactions

At the beginning of the Design phase, the **Service or Application Provider** and the **Security Administrator** provide information on the assets belonging to the target domain infrastructure and the interconnectivity between them, and expresses the security and trustworthiness requirements of their application service to be deployed over a specific set of devices comprising the domain. Security requirements in REWIRE are captured through an **expression language** that ensures expressiveness in capturing the security and privacy requirements that the device needs to be able to abide to during runtime, as well as the requirements on the interactions and communication between devices, both when they are part of the **same service graph chain in a domain**, or between **different domains**. Thus, the expressed requirements capture a range starting from the device per se (e.g., security, unforgeability, untraceability, runtime configuration), up to the requirements of devices stemming from their interconnectivity within the same or different service graph chains. Note that, while it is not within the scope of REWIRE to provide such a language, it is able to support *any language that enables the representation of requirements given by a security service provider, including obligations to be consumed by the formal verification models, in a machine-readable format*. One such example is the **ALFA policy language** [19], which provides the appropriate encoding for expression and enforcement. However, REWIRE is agnostic to the type of language, as long as it is able to support the aforementioned requirements.

It is also important to note that **the REWIRE Risk Assessment component participates in both the Design-time and the Runtime phases of REWIRE**. In the Design-time phase, after the extraction of the security and privacy requirements based on the information provided by the manufacturers of the devices comprising the service graph chain, the first step is the **risk quantification** process based on the most

prominent types of threats for assets deployed in the type of the target domain.

During runtime, the Risk Assessment component captures all asset topologies and relationships, including **SW-SW relationships** (e.g., process A receives input from process B), **HW-HW relationships** (e.g., device A sends a packet to device B), and **SW-HW relationships** (e.g. operating system A runs on device B), and is responsible for enabling any trust assessment framework to have an up-to-date view of the **Required Trust Level (RTL)** of the device, which translates to the types of evidence or security controls that can be deployed during runtime. For instance, if a the secure boot control is considered, the appropriate policies need to be constructed for collecting the evidence that demonstrates the correct operation of the secure boot process. This process happens constantly and allows capturing the risks identified by the **AI-based Misbehaviour Detection Engine** and the **Hook-based Event Analysis** components, or changes in the domain infrastructure. This information is used by the Security Administrator in order to guide the formulation of the system model to be used in the formal verification process.

After the appropriate system configurations have been specified as part of the design phase, the first step in order to enable the device to leverage the secure lifecycle management capabilities of REWIRE is to be equipped with the knowledge and attributes needed for enabling the device to **prove the correctness of its configuration and behavioural integrity during runtime**. The first step of this process entails the **REWIRE Risk Assessment (RA)** component. Note that the REWIRE RA features the appropriate interfaces for capturing the requirements of each device and asset, considering the relationships between assets. Note that the REWIRE RA engine, based on the OLISTIC engine developed by UBITECH, leverages the CVSS scoring system, but provides the capability for using different scoring systems more tailored to specific domains (e.g. TARA for automotive). Thus, *REWIRE innovates by merging different risk quantification processes tailored to the domain of interest*, and features a high degree of modularity that enables it to integrate different methodologies for capturing additional properties (e.g., privacy, which is not a core property of REWIRE, but it can be monitored with the appropriate methodologies). The process followed by REWIRE also features the degree of modularity required to capture it, as well as cascading effects and dependencies around the devices, *REWIRE envisions equipping the device with the knowledge of what attributes it should focus on in case of control flow integrity violations*.

The operation of the REWIRE RA engine is summarized as follows:

1. Through REST API, the RA component allows the Security Administrator to obtain the definition of all assets belonging to the system and capture the relationships between all assets.

2. Considering the identified threats and vulnerabilities, by employing a methodology that takes into consideration the **likelihood** and **impact** of the risks, the RA component generates the **risk quantification** graph of the system. Note that there is the capability for sharing this graph with authorized components and users through Kafka.

3. After the risk quantification process is complete, it can be provided via Kafka to the Security Administrator, so that they can manually identify in a supervised manner the security controls that can be deployed to minimize the impact of the risks, which in turn leads to the definition of the types of trustworthiness evidence (e.g., attestation evidence) that needs to be deployed during runtime to know that the security controls are deployed correctly.

4. In case of an indication of risk during runtime (e.g., failed attestation), or a risk indicator representing an attack or an anomaly, the **Security Context Broker (SCB)** pushes through Kafka the associated data to the Risk Assessment component, including indicators for a risk event (e.g., from the AI-based Misbehaviour Detection component). This triggers an execution of the risk quantification process and the recalculation of the risk graph.

5. All raw traces are shared through HTTP in an authenticated manner through the **Town Crier (TC)**, which acts as the secure oracle in the context of REWIRE. As these traces are associated through crypto structures, the TC verifies that they originate from an authenticated device, as well as the

integrity and correctness with the associated signatures. This is performed using the **Attribute-Based Signcryption (ABS)** scheme, based on which the device provides a **Verifiable Presentation (VP)** that is used to verify that it possesses the required attributes to provide those traces.

6. Afterwards, the TC sends the raw traces to be stored off-chain, and then the associated pointers to the location of the data on-chain on the **Besu** infrastructure.

Note that the risk quantification graph can be elevated to a **Required Trust Level (RTL)**, which describes the baseline level of trustworthiness that needs to be exhibited by a device so that it can be considered in a correct state. This means that a higher RTL corresponds to a need of stronger trustworthiness guarantees, which translates to the need for the correct enforcement of security controls, such as secure boot, runtime control flow integrity, etc. Thus, the REWIRE Risk Assessment produces the necessary output that enables devices to prove the ownership of such capabilities and attributes. In addition, it is possible for this process to be integrated into an automated trust assessment process, which falls outside the scope of REWIRE.

2.1.2 Compositional Verification and Validation Pipeline

The **Compositional Verification and Validation** component is a core aspect of the REWIRE framework, whose purpose is, in general, to use a variety of **formal verification tools** in order to **verify the protocols and schemes that will be employed by the devices, before they are synthesized and deployed to the actual devices prior to their execution**. In addition, this process entails the identification of the properties of the device that cannot be verified during design time, and need to be abstracted in order to be verified during runtime using the security enablers (e.g., attestation) of REWIRE, specified in the form of **operational and security policies**.

In this process, a core innovation of REWIRE is *the identification of boundaries of the protocols and schemes to be synthesized prior to their deployment to their respective devices, in order to narrow down the set of properties to be defined during runtime*. This approach serves to both facilitate the convergence of the formal verification processes during design-time, as well as increase the efficiency of the attestation process during design-time.

Considerig the above, the core dimensions of the REWIRE Compositional Verification and Validation approach can be summarised as follows:

- The compositional formal verification approach followed by REWIRE is a scalable solution to the security verification tasks we've seen as part of WP3. The biggest challenge is to address effectively the state-space explosion problem inherent in monolithic analysis of large complex security-oriented models. Rather than verifying the entire system as a whole, we have followed a compositional method decomposing REWIRE's main architecture components into smaller, more manageable modules. For instance, the LR-BC model verification paradigm using NUSMV is one of the core innovations of REWIRE. This first mode of operation was presented in D3.1, while the formal verification of the attestation toolkit itself is provided in D3.3 [22]. In the REWIRE approach, each component is verified independently against a local specification, often under assumptions about its environment (typically formalized via assume-guarantee reasoning). The global correctness of the system is then inferred from the correctness of the components and the validity of the composition rules, driven by the use-cases, as depicted in the architecture diagrams in AADL. This paradigm is particularly powerful for complex multi-layered cyber-security solutions, where modularity is a natural design principle. In this way we facilitate reusability of proofs, supporting incremental verification during system evolution. However, ensuring sound and complete composition rules remains a key research challenge, especially in the presence of rich interaction semantics, such as shared variables or asynchronous communication.

- While compositional formal verification is effective for ensuring design-time correctness, it presents limitations in terms of providing the same level of guarantees in evolving threats during the operation of the device, and in validating cyber-security components that face unpredictable and evolving threats during runtime. These components may operate in open environments with uncertain assumptions, making formal verification insufficient. In such cases, runtime validation techniques (such as the deployment of runtime monitors) are crucial. However, the question to be answered in this regard is *what part of the runtime system behaviour must be monitored for ensuring runtime trustworthiness?* In this regard, REWIRE allows the security administrator to identify the runtime properties of the system and provides the required tracing capabilities for the collection of the corresponding evidence, as well as the mechanism to deploy MSPL-based policies through the **Policy Orchestrator**, as will be detailed in Chapter 6. The monitors offered by REWIRE continuously observe system behaviour during execution, checking compliance with security policies or detecting deviations indicative of attacks, thus complementing formal verification with dynamic assurance in operational contexts. An effective approach to enforcing operational constraints in cyber-physical or security-critical systems involves the formulation of executable policies, such as those inspired by the MSPL paradigm. These policies are formally specified yet directly interpretable by enforcement mechanisms embedded in the system, such as runtime monitors or Real-Time Operating System (RTOS) configurations. By encoding constraints—such as access control rules, resource usage bounds, or timing guarantees—into executable form, the system can dynamically enforce compliance during execution. This enables adaptive enforcement strategies that respond to changes in the operational context, while ensuring that safety, security, or performance constraints are maintained. Integration with RTOS primitives further allows fine-grained control over scheduling, task isolation, and privilege levels, thereby bridging the gap between high-level policy specification and low-level enforcement.

- Through MSPL, REWIRE's compositional verification and validation framework that integrates offline realizable formal verification techniques—such as theorem proving and model checking—with structured formal validation methods to address the correctness and assurance needs of complex cyber-physical and security-critical systems. Thus, the verification layer follows a compositional approach, where system components are analyzed individually against local specifications, using assume-guarantee reasoning (AADL) and formal refinement techniques (NUSMV, CBMC, Coq) to infer global system correctness. This is complemented by a validation layer that leverages formally defined architecture models to establish traceable assurance claims, capturing structural, behavioral, and security properties such as data flow correctness and isolation constraints. Where compositional verification proves insufficient—particularly for dynamic security components—the framework supports executable policy enforcement through runtime mechanisms (e.g., monitors or RTOS-based enforcement), ensuring operational compliance with high-level constraints. Together, this integrated approach provides a scalable and trustworthy methodology for the design-time and runtime assurance of safety and security properties in heterogeneous and evolving systems.

Considering all the above, the Compositional Verification and Validation pipeline of REWIRE combines three core dimensions:

- The first dimension is the **formal verification of individual component operational profiles** mapped to the security requirements of the operation of the devices, as analysed in D3.2 and will be expanded in D3.3 [22], meaning components and devices in isolation, without considering them as part of a service graph chain. This dimension includes the use of **theorem proving tools** for formally verifying one of the core crypto schemes leveraged, which is the use of **LRBC** for enabling the establishment of software updates through an authenticated and encrypted communication channel. LRBC is essentially a lightweight block cipher design which has been proposed for use in resource-constrained IoT devices in order to provide data security at the sensing level. As

part of this process, during the following period, the mathematical analysis of the REWIRE **Zero-Touch Onboarding (ZTO)** scheme and the runtime crypto primitives for continuous authentication and authorization will be performed using the **Universal Composability (UC)** method. Note that, as will be described in D6.2 [24] and further detailed in D3.3 [22], focus will be placed on the formal verification of the **REWIRE Configuration Integrity Verification (CIV) scheme** as the main attestation extension that enables the device to establish its level of trustworthiness during runtime.

- Building upon this formal verification model, the second dimension is the **evaluation of design-time assurances for validating the global correctness of the interactions between individual REWIRE components**. Note that this is a new feature of the component, which is analyzed in detail in D3.3 [22]. In this process, a model-based design framework for the verification and validation of the system architecture is required in order to model the mapping of the previously identified requirements to specific assets and artefacts, as well as verify the functional and requirements-driven correctness of the system. To this end, REWIRE employs **AADL-based System Modelling** for specifying both software and hardware configurations and building a representation of the system to be validated, which captures the interactions between components to be verified as part of the use cases. AADL is a standardized architecture description language for hierarchical structures and connections between software and hardware components, and the approach followed by REWIRE entails creating a virtual representation of the assets and their interconnectivity, as well as associated asset properties and requirements defined by the Security Administrator or Use Case provider. Afterwards, based on the AADL-defined architectural model, the **Resolute** language/tool is used for **ratifying and validating the correctness and the achievement of the requirements** through architectural **assurance cases**. Note that an assurance case is a structured argument that presents and supports claims about a system's behaviour, and will guide the formulation of the security policies to be deployed to the devices.

- The third dimension concerns the **verification of the correctness of the data sources to narrow down the runtime verification**, which has been analysed in detail in D4.3 [25]. This entails the delineation of the expected nominal behaviour of the devices, i.e., **the identification of the trust boundaries of the system** by leveraging different types of formal verification mechanisms. Thus, whatever is included in the trust boundary can be considered by the Security Administrator to have the necessary guarantees to behave as expected during the operation of the devices. Conversely, protocols or applications that are too complex to be formally verified need to be abstracted, so that they can be attested during runtime, through the attestation of the respective **system properties** linked to the appropriate behaviour of the device. Thus, *the protocols that will be synthesized as part of the device hardware are evaluated, so that we can formally verify if and to what extent these protocols and schemes can capture the requirements of the device, and for the requirements that cannot be verified in design time, attestation policies for runtime will be created*. In this regard, a core innovation of REWIRE is that, when it comes to device interactions, we also consider the types of input/output, but depending on the complexity of the model, *some operations are abstracted in order to enable the convergence of the verification process*. Thus, during runtime, there is no need to attest the entire software stack of the device (which may be quite computationally complex), but it is sufficient to attest to the **the veracity of the data itself used as input during the operation of the device**, thus narrowing down the design space of the runtime process. Using this method, the efficiency of the novel CIV scheme offered by REWIRE has been evaluated, through the verification of a smaller set of runtime system properties.

2.1.2.1 Formal Verification of Runtime (Implicit) Attestation based on key Restriction Usage Policy Management

As described in D2.1 [21] and expanded in D3.3 [22], the **Formal Verification** component of REWIRE is used in order to formally verify the functional correctness of a configuration before it is synthesized in a

RISC-V implementation. Specifically, we have performed the formal verification of the **LRBC encryption scheme**, which has been proposed as a block cipher for use in IoT devices with guarantees against leakage. The next step of this process is the formal verification of the **security enablers** of REWIRE, i.e., the attestation schemes used for verifying the configuration correctness of a device, which are supported by the underlying **Root-of-Trust (RoT)** through the appropriate cryptographic keys.

While the formal verification of protocols and interfaces has gained increasing prominence in the literature as a fundamental process for the identification of bugs or misuse cases affecting complex computational systems, the formal verification process presents various challenges, such as the fact that protocol verification is an inherently undecidable problem for an unbounded number of protocol executions and size of terms. Thus, existing state-of-the-art methods to conduct formal verification of attestation schemes are typically limited in some manner, as they are focused on either (i) only the **cryptographic mechanisms supported by the underlying RoT**, or (ii) remote attestation services considering a **Prover-Verifier model**. However, these approaches present drawbacks that limit the reasoning that can be derived about the level of assurance of the overall system and the scope of the analysis that can be achieved, as there is *no distinction between the cryptography used for self-consumption of the RoT (e.g., secure storage), and the cryptography specifically provided for the overall application*.

In this regard, a core innovation of the formal verification approach of REWIRE is to provide a methodology that alleviates these drawbacks, by considering that *RoTs are considered inherently secure and trusted-by-default elements that implement hardened protections (e.g. tamper-proof or side-channel resistance)*, thus **all internal operations, including handling of cryptographic data, can be idealized**. Thus, the formal verification approach of REWIRE aims to leverage this property and translate it to the overall application security based on the cryptographic capabilities provided by such RoTs. In this context, REWIRE is the first project of its kind to perform formal verification of the REWIRE Configuration Integrity Verification (CIV) scheme based on **local attestation**, which utilizes **key restriction usage policies** and does not require an external Verifier.

Specifically, as all security enablers provided by REWIRE are **agnostic to the underlying RoT**, the formal verification methodology of REWIRE entails the consideration of **perfectly secure crypto** provided by the RoT. This enables us to follow a two-layer methodology for formal verification: (i) At the first layer, we formally verify the attestation scheme excluding the threat model for the traces collected as attestation evidence (e.g., leakage of traces) by considering them trusted, and (ii) at the second layer, we weaken this assumption by considering the possibility to verify the traces themselves. Note that this highlights an innovation of REWIRE, as existing approaches typically consider the traces trusted as part of the TCB, while we also consider the capability for the verification of the traces themselves. In this context, REWIRE is also the first project of its kind to formally verify key policy management.

The adoption of formal methods in the specification and verification of security protocols yields multiple benefits. First, by encoding the protocol in a formal language, a precise documentation of the protocol behavior is obtained, that prevents ambiguities, contributes to detecting logical issues, and serves as implementation guidance. Second, applying formal verification techniques ensures mathematical rigor in proving security properties, such as authenticity and confidentiality, identifying potential flaws, already in the specification phase, that might not be apparent through conventional testing during development.

The CIV attestation protocol will undergo the following activities. First, the protocol specification, currently expressed in a semi-formal notation, will be encoded in the formal language defined by the Verifpal tool[1], highlighting the correspondences between the communication steps in the original notation and in its encoding. The reference framework consists of a public channel, where two or more agents, according to the protocol (multiple sessions are allowed to run in parallel), perform computations and exchange messages, which can be intercepted and possibly modified or blocked by a malicious attacker. Second, the protocol formalization will be subject to property checking analysis, focusing on some of the standard security properties supported by Verifpal, such as confidentiality (the attacker cannot obtain a certain

[1] https://verifpal.com/

piece of information), authentication (a recipient can successfully make use of a message, apparently originating from an agent, only if that agent is indeed the sender), and equality.

Verifpal accounts for two types of attackers, a “passive” one that can observe, but not alter communications, and an “active” one, that can freely modify and inject messages; the active attacker is constrainable by using “guarded” messages, readable, but unmodifiable. This flexibility, together with the possibility of explicitly modeling leakage of agents’ local data, allows for a multi-layered security analysis as described above.

2.1.3 Security Policy Construction and Management

After the formal verification of the processes and the definition of the trust boundaries of the system, the next step is the appropriate **definition of the security controls**. These are defined by the Security Administrator as mitigation actions that can safeguard the operation of the system during runtime, corresponding to the properties that have been identified as needing to be attested dynamically during runtime. This guides the creation of the policies to be deployed, expressed in the selected policy language, which has been specified as the **Medium Security Policy Language (MSPL)** in the context of REWIRE. Note that the selection of MSPL is a design choice of REWIRE, but it is possible to select any policy language with the level of expressiveness required for capturing different types of requirements (e.g., the ALFA policy language [19] emerges as an interesting alternative). However, MSPL has been selected since it uses the same encoding as the Yang data model, which is the structure adopted for secure data management and capturing attestation evidence as proposed by the Internet Engineering Task Force (IETF).

The policies constructed in the context of REWIRE specify information including the *conditions for triggering the execution of the policy*, the *types of evidence that need to be collected* in order to verify the correctness of the identify properties, and the *types of security controls* to be applied. Note that, in the context of REWIRE, two types of policies are considered:

1. **Operational policies:** Recall that it is possible that the Security Administrator needs to deploy a patch or update following an indication of risk, going either through the AADL modelling process or directly through the Formal Verification process. Note that indications of risk may originate from the security controls of REWIRE, such as the attestation enablers or the AI-based Misbehaviour Detection component. These policies essentially dictate the software update tasks that need to be performed by the devices for mitigating the identified threats or vulnerabilities. Any new patch needs to be validated by the AADL, as well as the Formal Verification component of REWIRE.

2. **Security policies:** These refer to the policies that dictate or predicate the execution of a security enabler (e.g., attestation process) for verifying the correctness of the identified security properties that need to be attested during runtime. Note that the output of these security enablers constitutes the trustworthiness evidence based on which any Trust Assessment Framework (TAF) can characterize and evaluate the Actual Trust Level (ATL) against the Required Trust Level (RTL) of the system, as defined by the Risk Assessment component.

The output of the Compositional Verification and Validation component also includes the **SW/HW co-design system configuration** of the software stack to be deployed, considering also the expected configuration that is used as a basis for the construction of the policy hash upon which the key restriction usage policy is constructed. In the context of REWIRE, the deployment and enforcement of this configuration is assumed to be performed through well-established concepts, such as the use of **Manufacturer Usage Descriptions (MUDs)**. Specifically, REWIRE assumes the existence of MUD profiles which are always synchronized with the latest formally verified system verification of the device, including also the policy hash representation of the expected configuration of the device. The latter is later going to be used for the establishment of the **key restriction usage policies governing the correct usage of the Attestation Key (AK)**. Note that, in this regard, *REWIRE is the first project of its kind to demonstrate the*

use of key restriction usage policies in a manner that guarantees that only the latest policy can be used so that attempts by attackers to use deprecated policies are mitigated, thus elevating them to verifiable key restriction usage policies.

In parallel with the two types of policies specified in the previous step, the actions pertaining to the MUD profile management are as follows:

1. When a fundamental update on a device needs to be performed, the deployment is performed through the MUD profile of the device. Specifically, when an update is made to the MUD profile of the device on the Blockchain Infrastructure of REWIRE, an automatic notification is issued to the concerned devices regarding the new configuration of the device, thus triggering an **update of the key restriction usage policies** considering the new correct configuration state of the device.

2. If there is a reaction to an **indication of risk** resulting in the enforcement of a software update, or a refinement of a security policy dictating the types of attestation tasks to be executed for the specified set of system properties to be attested dynamically during runtime, this is enforced through the construction and deployment of **MSPL-based enforceable device management and data flow security policies**. Thus, the output of the AADL is pushed through the MUD profile of the device, while lighter updates to security policies (e.g., types of attestation tasks) are expressed in MSPL format and pushed to the MUD profile of the device.

2.2 REWIRE Zero Touch Onboarding Extensions

One core offering of REWIRE is the provision of mechanisms for *equipping the device with the required cryptographic material for achieving runtime trust establishment*. This is achieved through a **Zero-Touch Onboarding (ZTO)** mechanism that is performed when a device enters the domain, and is based on a novel and efficient **Attribute-Based Signcryption (ABSC)** scheme, which combines the benefits of **Attribute Based Encryption (ABE)** and **Attribute-Based Signatures (ABS)**, for enabling the device to verifiably demonstrate its attributes prior to its enrolment. Note that the inclusion of the latter is a novelty of REWIRE.

Specifically, the enrolment of the device into the service graph chain (SGC) of the target domain in the context of REWIRE is performed after the finalization of the Design Phase and the software/hardware configuration has been synthesized and before proceeding to the runtime phase, through the execution of the **Zero-Touch Onboarding (ZTO)** scheme. In this context, REWIRE extends the **Extensible Authentication Protocol (EAP)** by providing new cryptographic schemes and protocols to authenticate and enrol the device, based on its capability to prove the ownership of specific attributes. Note that REWIRE supports two variants of the ZTO scheme with different privacy flavors:

1. The first variant, which has been described in D4.2 [23], is based on **Verifiable Credentials (VCs)** issued for the device in order to depict its attributes in a verifiable manner. This approach provides strong privacy protection in the form of **selective disclosure**, as it allows the creation of **Verifiable Presentations (VPs)** that contain only the subset of attributes required for onboarding a device. This approach leverages the **Attribute-Based Signcryption (ABSC)** scheme, which enables the creation of attribute-based signatures. This has been described in D4.2 [23], and its final version will be presented in D4.3 [25], along with the security analysis of the scheme.

2. The second flavor, whose definition and implementation are documented in D4.3 [25], has fewer privacy restrictions while not breaching the core privacy profile of the device. Instead of employing selective disclosure, with this approach, the device provides **zero-knowledge proof** of all the attributes disclosed in the Verifiable Credential.

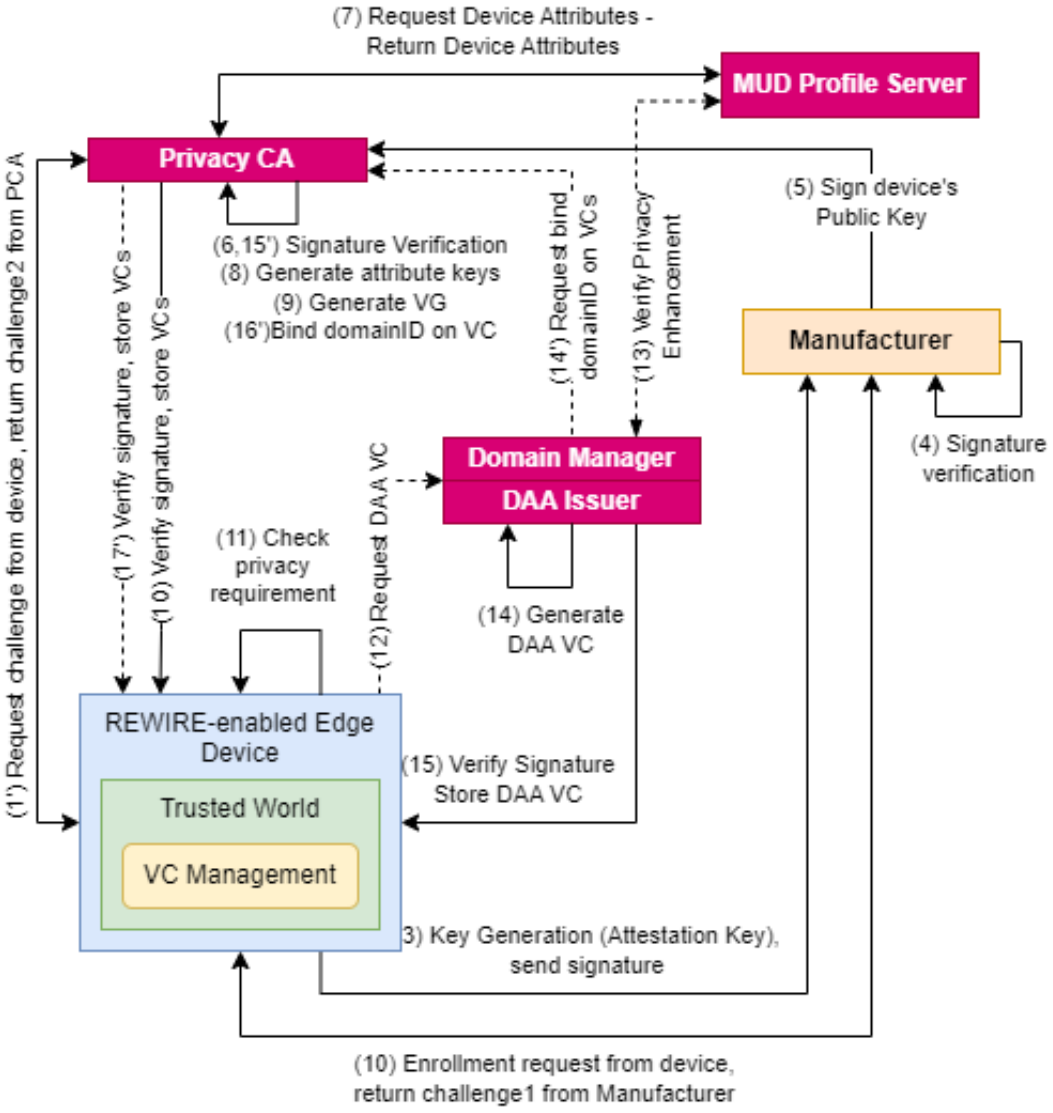

Figure 2.3: REWIRE Zero-Touch Onboarding

In the following, we provide a description of both aforementioned flavours of the ZTO scheme. Specifically, while the initial part is common between both flows, they diverge at a point that will be specified during the description of the flow detailed below.

Figure 2.3 illustrates the zero-touch onboarding process of an edge device into the Rewire infrastructure. When the device is initialized, it first generates its attestation key pair, a private-public key pair used to prove the device's identity within the system. This key pair is created using a predefined pairing-friendly elliptic curve group, which enables advanced cryptographic operations required by the system. Once the attestation key is generated, the device asynchronously requests two challenges, also referred to as nonces: one from the Privacy Certificate Authority (Privacy CA) and one from the Manufacturer.

Upon receiving these two challenges, the device computes two separate cryptographic digests. The first digest, intended for the Privacy CA, is generated by hashing the concatenation of the two challenges. The second digest, dedicated to the Manufacturer, is produced by hashing the concatenation of the two challenges along with the device's public key. To prove authenticity, the device signs the Manufacturer-specific digest using its Root ID Key, a secret key known only to the device and the Manufacturer. At the same time, the device signs the Privacy CA-specific digest using its attestation private key. In addition to sending the signed digest, the device also forwards the Privacy CA-issued challenge to the Manufacturer.

A trusted and authenticated communication channel is established between the Manufacturer and the Privacy CA, ensuring the integrity and authenticity of all messages exchanged. Through this secure

channel, once the Manufacturer verifies the HMAC using the Root ID Key and confirms the authenticity of the device's public key, it signs the public key and forwards it, together with the original Privacy CA challenge, to the Privacy CA.

Upon receipt, the Privacy CA verifies the Manufacturer's signature on the device's public key. If the verification succeeds, the Privacy CA uses the validated public key to verify the device's signature over the hash of the two challenges. Successful completion of these verifications confirms the device's authenticity. Before generating attribute keys, the Privacy CA extracts the device's attributes from the MUD, requested and downloaded from the MUD Profile Server for the id of the device of the specific authenticated public key.

These attributes are used to derive the attribute keys, which in turn are used to construct Verifiable Credentials (VCs) that are cryptographically bound to the device through the device's attestation public key. The resulting credentials, along with the attribute keys, are then securely transmitted back to the device through a secure and authenticated channel for storage and future use. Depending on domain-specific privacy policies, additional steps may follow.

After initial onboarding, the device submits its Domain ID to the Domain Manager. The Domain Manager first verifies the Domain ID and generates a domain-specific attribute key, which is shared among all devices participating in the same domain, thereby establishing a secure communication channel. Here, the following two alternatives are available, representing the two aforementioned privacy flavours:

1. In the first flavour, the device creates a **proof of knowledge using the Verifiable Credential acquired from the Privacy CA**, disclosing **all attributes defined in the device's MUD profile**. This proof is constructed using the device's attestation key and includes a signature over the Domain ID.

2. In the second flavour, the device creates a **Verifiable Presentation (VP)** containing only the subset of attributes required for the verification of the device. Note that this approach is in line with the notion of **selective disclosure**, that enables the device to only share the required attributes, and not the entirety of available device attributes.

Upon successful verification of the proof or VP, the Domain Manager consults the device's Manufacturer Usage Description (MUD) profile to determine whether enhanced privacy measures are necessary. If enhanced privacy is required, the Domain Manager will forward the device's proof of knowledge to the Domain Attestation Authority Issuer to produce a domain-specific, randomized DAA Verifiable Credential (DAA VC) binded to the public part of the device's attribute keys issued by the Privacy CA. Once verified by the device, this DAA VC is stored and used for privacy-preserving authentication within the respective domain. If no enhanced privacy is required, the Domain Manager requests the Privacy CA to bind the device's Domain ID to its existing Verifiable Credential (VC). The updated VC is then transmitted back to the device for secure storage.

2.3 Runtime Architecture

As detailed in the previous section, the core outcomes of the design phase of the REWIRE framework are the **security and operational policies** dictating the types of actions needed for verifying the correctness of the state of the attributes that have been identified as needing to be attested dynamically during runtime, as well as the reference values representing the correct system configurations against which the device state needs to be verified. The runtime phase receives the outcomes of the design phase and prepares the device so that it can utilize the secure lifecycle management capabilities of REWIRE. Thus, the sequence of actions of the runtime phase of REWIRE is summarized in the following sections.

2.3.1 REWIRE Policy Orchestrator

REWIRE, in its final version, implements a **Policy Orchestrator**, which acts as a bridge to mediate the interactions between the device and the outside world. Thus, the Policy Orchestrator is responsible for mediating the execution of the security and operational policies originating from the **Compositional Verification and Validation** component and sending them to the **Facility Layer** of the device, which can translate then to **execution logic**. This logic an dictate the **Attestation Agent** to perform the attestation tasks described in Section 2.3.2, or the execution of the enablers described in Sections 2.3.3 and 2.3.4.

2.3.2 REWIRE Trusted Computing Enablers

As aforementioned, in order to attest to the correctness of the properties that have been determined to require runtime verification based on the output of the **Compositional Verification and Validation** pipeline, REWIRE offers a set of security enablers to serve this purpose. With regard to the security enablers offered based on the use of Trusted Computing, REWIRE is the first project of its kind to offer **implicit attestation** based on the **key restriction usage policies**, alleviating the need for a remote Verifier. In addition, REWIRE innovates by offering a **process state verification** enabler, which provides the capability for **verifiable enclave and process management**, ranging from the verifiable launch of the enclave, to the runtime self-issuance of certificates with guarantees on the correct state of the device.

2.3.2.1 Implicit Attestation through Configuration Integrity Verification (CIV)

Here, we provide further details on the former of the aforementioned type of security enabler, namely **Configuration Integrity Verification (CIV)**, which is able to attest to the correctness of the configuration state of a device. In general, as part of this process, the **Attestation Agent**, which is running on a secure enclave as part of the REWIRE TCB, receives traces from the REWIRE Tracer, and it attempts to match them with the reference values contained in the key restriction usage policies. The novelty of REWIRE in this regard is the use of a **Daemon-based tracer**, which operates in the background for enabling the collection of traces in a non-intrusive manner during the operation of the target binary, without impeding its operation.

Thus, in case a policy dictates the execution of such attestation action, the action workflow can be summarized as follows:

1. The **Attestation Agent**, which runs in a secure enclave in the trusted world, communicates with the corresponding enclave in the **trusted world** of the device. Note that the **Attestation Key** has been created during the onboarding process, and is stored in the **Key Management** module of the enclave.

2. The **REWIRE Tracer** collects the required attestation evidence for proving the correctness of the configuration state of the device. Note that the Tracer application has counterparts in both the trusted and the untrusted world of the device, and there is a secure communication channel established between them.

3. The collected evidence is used in order to verify the key restriction usage policy. If successful, the Attestation Key can be used in order to sign the response to the attestation challenge, thus verifying the correctness of the device configuration.

4. In case of a failed attestation, the process outlined starting from step 1d of this action workflow is performed for the storage of the raw traces and the re-quantification of risk.

In case of a failed attestation, mitigation measures need to be taken in order to ensure the correct operation of the device. Specifically, the device state management capabilities of REWIRE include resilience mechanisms, ensuring service continuity even in case of device compromise or misconfiguration. Note

that these features are enforced by the **Policy Orchestrator**, as their execution is dictated by policies, as previously described in the Design phase. In addition, following the execution of such mitigation measures, raw traces are also sent to the **Software/Firmware Vulnerability Analysis tool** of REWIRE, which was first described in D2.1 [21] and is further elaborated and benchmarked in D3.3 [22]. This component employs a mixture of well-established and new fuzzing techniques capable of coping with the internal interdependencies of RISC-V architectures, in order to provide more information that allows the **Software Service Provider** to push a software update. Any specific risks and vulnerabilities identified are pushed to the Risk Assessment component, in order to update the risk graph, thus resulting in pushing new security controls in the form of policies.

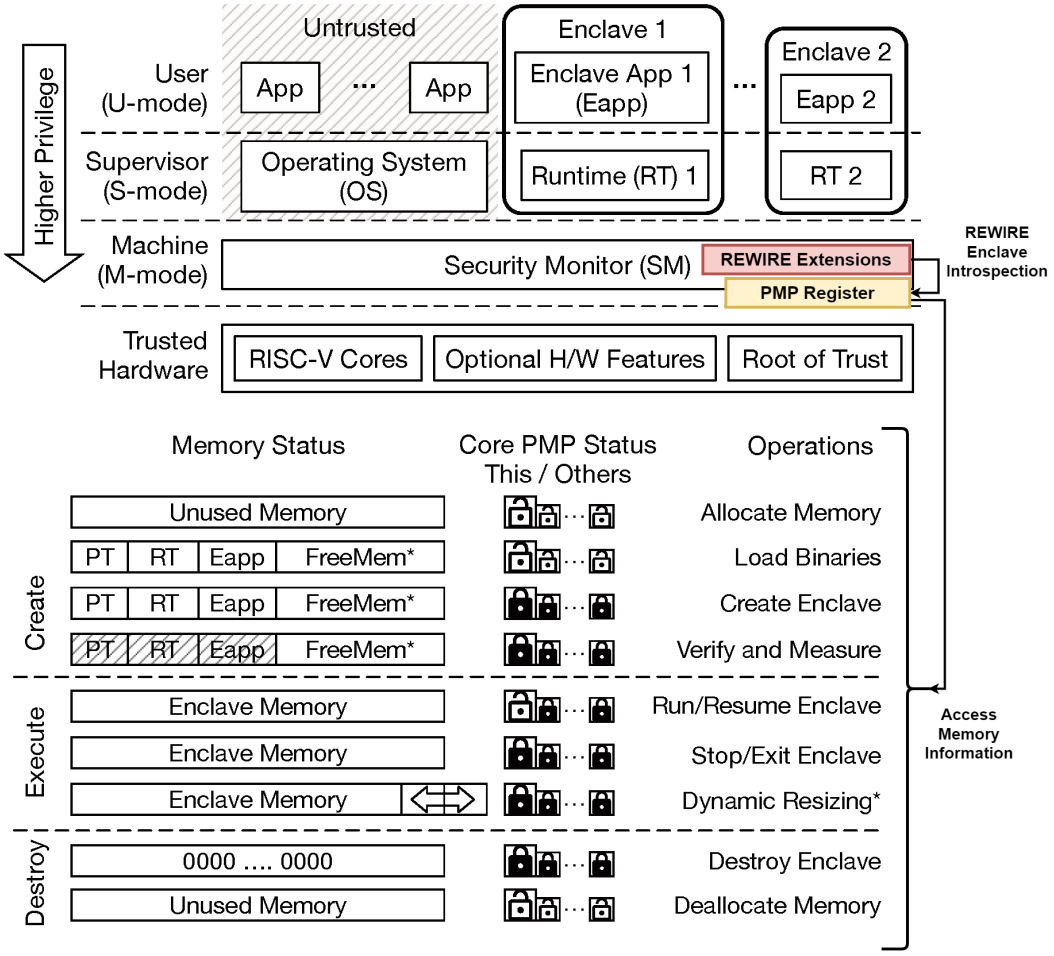

Figure 2.4: REWIRE Process State Verification

2.3.2.2 Process State Verification

The **Process State Verification** process is part of the runtime security guarantees offered by REWIRE. Specifically, in the final version of the REWIRE architecture, we provide the necessary functionalities to be able to *translate the output of attestation processes into certificates*, which any remote Verifier can access in order to audit the correctness of the device state. In addition, in contrast to existing device state verification capabilities in the literature, REWIRE is the first project of its kind to provide these mechanisms within the realm of the device itself.

Thus, REWIRE aims to provide the capability for the device to self-issue such **DICE-like certificates** that a remote Verifier can use to verify the entire lifecycle of an enclave, from its secure launch and including its entire lifecycle to be able to provide runtime guarantees that the enclave behaves as expected. Compounding this issue, as will be detailed in D4.3 [25], REWIRE introduces new SBI Calls for being able to access the aforementioned information. Specifically, in the case of **Keystone architectures**, we are

able to receive information about the memory status of the enclave regarding runtime execution, memory allocation, etc., from the **PMP Register** of the device.

Figure 2.4 depicts the positioning of the Process State Verification process in the context of a device, as well as the types of information that can be monitored. Further information regarding the PMP register and the structure of memory-related information can be found in [13].

2.3.3 SW Update Process

One mechanism for addressing Indications of Risk is the **Software Update** capability of REWIRE, with verifiable guarantees on the correctness of the process. The first version was provided in D4.2 [23], but refinements were made and documented in D6.1 [20] with regards to the integration of this process so that it can be supported in Keystone. Specifically, these refinements entailed *fine-tuning for race conditions that may be present when the enclaves are trying to access the same set of data as part of the Security Monitor*.

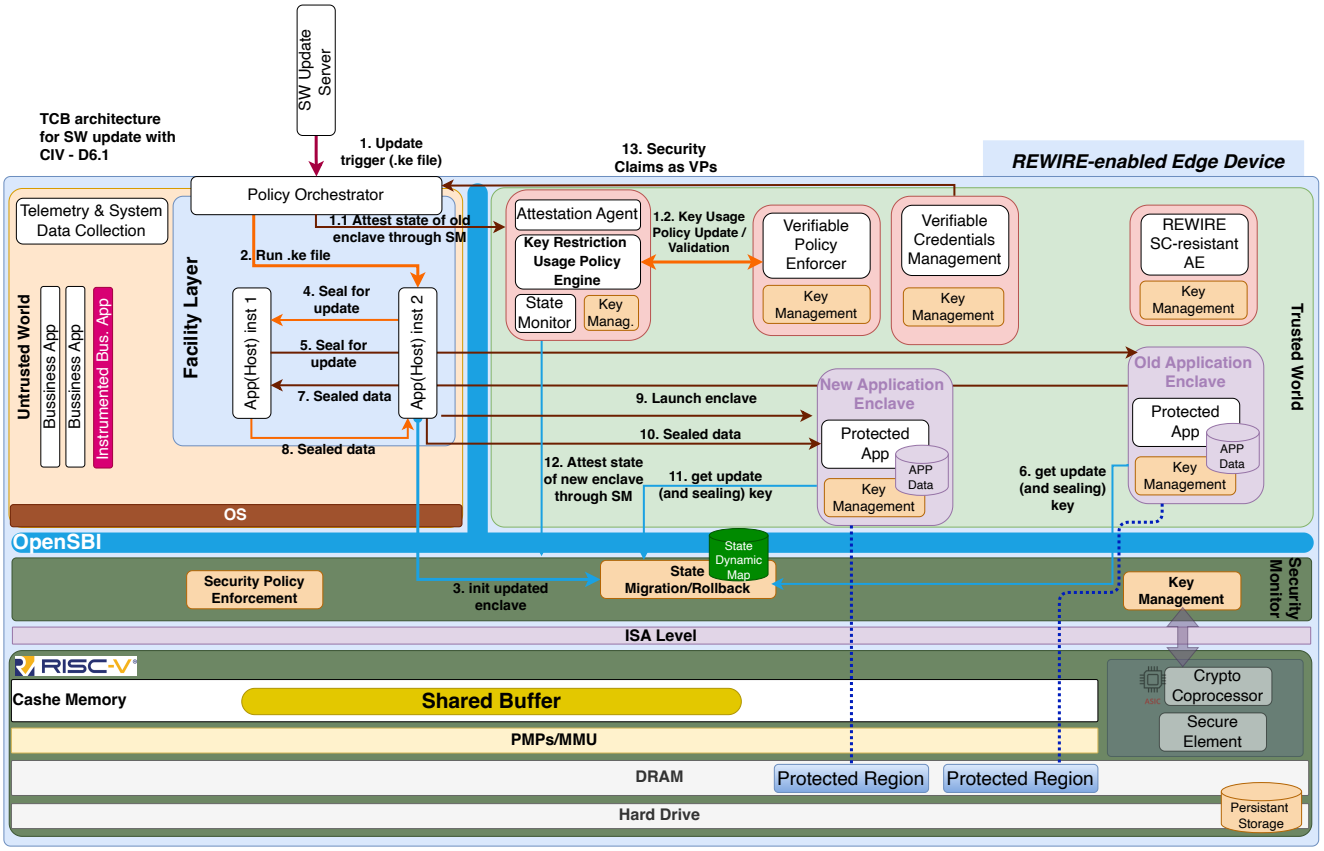

Figure 2.5: REWIRE TCB Architecture

Figure 2.12 represents the latest update of the REWIRE TCB. We now describe our updates to the TCB regarding the Software update process. We consider an enclave application E_1, orchestrated by a host application H_1, that is currently running on the device. Throughout the update process, it is replaced with an updated enclave application E_2, orchestrated by a new host application H_2. On a high-level, the SW update securely launches an updated enclave, transfers the state of the old enclave to the new one, and then terminates the old enclave. The detailed flow is comprised of 13 steps, which we describe in the following:

1. **Update Trigger:** An update is triggered in response to an event, such as a detected security bug or a feature extension. In such a scenario, the policy orchestrator receives an updated `.ke` file (including the new host and enclave) to the policy orchestrator (cf. step 1). The policy orchestrator

then attests the enclave to be updated using the REWIRE enhanced CIV-attestation scheme (cf. step 1.1). Moreover, the policy orchestrator verifies the update policy of the old enclave, validating that an update is permitted (cf. step 1.2).

2. **Start Updated Binary:** If the attestation and verification of the old enclave and the update are successful, the policy orchestrator runs the updated file. This starts the host H_2 of the updated enclave E_2, which then makes a series of SBI calls to the SM to securely install E_2, as described in the next steps.

3. **Enclave Initialization:** H_2's first SBI call triggers an existing function in the SM that initializes the updated enclave. Specifically, a new enclave E_2 is instantiated and initialized with the updated enclave code to register it in the Security Monitor and assign an enclave ID that can be used in the process of the enclave update. At the end of this step, the E_2 is initialized and registered but not yet running.

4. **Request State Export from Enclave Host:** If the enclave that is to be updated is stateful, it is essential to transfer the state from E_1 to E_2. In that case, the H_2 signals H_1 to export its enclave's transferrable state.

5. **Request State Export from Old Enclave:** The host cannot access the encrypted enclave state. Thus, H_1 signals its corresponding enclave E_1 to seal its state for an update.

6. **Request Update Sealing Key:** The enclave state is encrypted with a key that is bound to the hash of the specific enclave binary. That is, E_2 cannot access this encryption key, nor should it be able to do so to guarantee backward secrecy. Thus, E_1 queries a dedicated transport key from the Security Monitor. To realize this step, we implemented a new SBI call `get_transport_sealing_key` that derives an encryption key based on the enclave IDs of E_1 and E_2.

7. **Transfer Encrypted State to Enclave Host:** E_1 uses the received update transport key to encrypt its current state and return the encrypted data to H_1.

8. **Transfer Encrypted State to Updated Enclave Host:** Next, H_1 transfers the sealed state to H_2.

9. **Updated Enclave Launch:** The updated enclave E_2 is launched so that code in the enclave can be executed, e.g., import sealed state from the old enclave instance. As part of the enclave launch, the update is verified in terms of SW version and authenticity through a signature. Moreover, the SM marks E_1 as an enclave that is in the process of being updated and does not forward any further ecalls to this enclave. This is important to ensure that (i) no untracked state upates are performed during the update process and (ii) the two enclave cannot execute in parallel. We implement this functionality in a new SBI call `update_enclave`.

10. **State Import:** H_2 then starts the execution of E_2 using a new `runUpdate` SBI call.Next, H_2 forwards the sealed data received from H_1 to E_2.

11. **Request Update Unsealing Key:** H_2 queries the corresponding unsealing key from the Security Monitor to decrypt the sealed data. To realize this step, we implemented a new SBI call `get_transport_unsealing_key` that derives a decryption key based on the enclave IDs of E_1 and E_2, corresponding to the encryption key derived in step 6.

12. **Attestation:** At this stage, the update is finalized. In a last step, the Attestation Agent attests the updated enclave through the security monitor to ensure that it is running correctly.

13. **Security Claims:** Finally, the Attestation Agent returns the security claims in the form of VPs to the Policy Orchestrator.

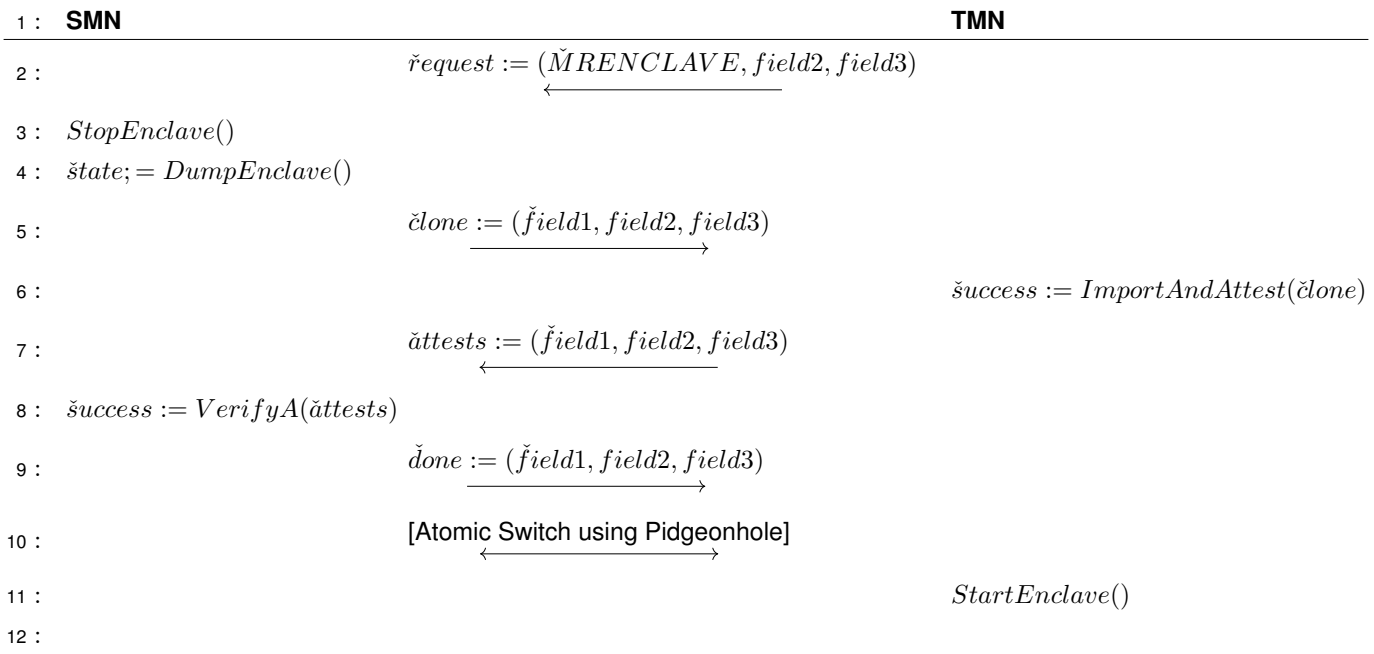

Figure 2.6: REWIRE Migration Protocol Overview between Source Migration Node (SMN) and Target migration Node (TMN)

The flow described above is our realization of the SW update protocol described in deliverable D4.2 [23], including several modifications to adapt to the environment provided by Keystone. A detailed description of the challenges and the concrete modifications associated with each of these challenges is provided in deliverable D6.1 [20].

2.3.4 Secure Live Migration Reinforcement

REWIRE provides the capability for *migrating an application state running in an enclave of a device whose own trust level has been deemed compromised or untrustworthy (Source Migration Node), to a different device with the required trust level (Target Migration Node)*. This translates to securely importing the state to the different device, so that it can continue its operation on the healthy device. In this regard, the core innovation of REWIRE is **allowing the migration to be taken through a verifiable state management process**. Thus, it goes beyond migrating an entire enclave and enables managing enclave states with proof of execution that the migratable state has been loaded and reinstated correctly in the target migration node (Figure 2.6).

By using the advanced introspection capabilities of REWIRE, it is able to extract data that captures the state of an enclave (both **volatile** and **persistent** states), which is afterwards imported by using a mix of HW-based sealing keys provided by the underlying **Root-of-Trust (RoT)**, as well as additional keys (e.g., Diffie-Hellmann). Note that the live migration process of REWIRE is agnostic to the type of TEE used, but the first implementation has been performed using Keystone while the new tracing capabilities on the internal enclave memory targeted th intel SGX TEE and specifically the Gramine version by extending the *proc* tracing system for monitoring ELF mapping (mor detailed can be found in deliverable D4.3 [25]).

At a high level, the live migration process of REWIRE consists of three core phases, namely **Enrolment**, **Migration Affinity**, and **Verifiable Import**. In the following, we provide a description of the action workflow of this process, elaborating on all aforementioned phases.

1. **Enrolment Phase:** This phase is responsible for installing migration services and credentials on the device, and provisioning the migration service of both devices participating in the migration, namely the **Source Migration Node (SMN)** and **Target Migration Node (TMN)**, with the necessary credentials and access control policies. Each node generates a cryptographic key pair, and its public

key is registered to the Domain Manager, which issues a digital certificate attesting to its authenticity. This lightweight, one-time setup enables mutual authentication between nodes without requiring a persistent trusted third party during live migration. In addition, each device is provisioned with a **reference system call graph** specific to each trusted application it hosts and can be subject to migration. This graph captures the correct and expected flow of the system calls during relaunching of the migrated application while loading its volatile state, thus verifying that the application continues executing correctly after migration. In addition, each Migration Service initializes a **secure migration policy** during the Enrolment Phase, which governs the application instantiation and includes a mapping of unique enclave identities to their operational status, ensuring that only one active instance of each trusted application can run at any given time. The initialization of this mapping can be either static, provisioned directly by the Orchestrator -— ideal for resource-constrained or highly regulated environments —- or dynamic, maintained autonomously by the Migration Service to support more flexible deployment scenarios. This proactive policy enforces single-instance execution by design, offering strong protection against enclave duplication, fork bombs, and cloning attacks without relying on reactive detection mechanisms.

2. **Migration Affinity Phase:** At a high level, this phase initiates the migration process, verifies the target TCB, and exports the enclave state. This phase is initiated when the TMN initiates the process to migrate an application originating from the SMN. This process an be outlined as follows:

   (a) The migration service of the TMN sends a migration sallence to the SMN, including a **digital signature over the enclave to be migrated**, its **Public Key Certificate (PKC)**, and its **Public Key (PK)**. Upon reception, the SMN verifies the authenticity of the migration challenge and verifies the existence and operating status of the application to be migrating through a secure enclave registry service.

   (b) After successful validation, the SMN and TMN establish a secure communication channel. This is done using an **Elliptic Curve Diffie-Hellman (ECDH)** key exchange, followed by a secure **Key Derivation Function (KDF)** to generate a symmetric encryption key, which ensures confidentiality and forward secrecy for the transferred state data.

   (c) The Migration Service of the SMN instructs its application to export its current volatile state, which is then written into a file. This state includes the necessary metadata to resume correct execution, and is sealed using the enclave-based sealing mechanism of the platform (e.g., Keystone) for protecting this information.

   (d) The sealed file is handed off to the Migration Service of the SMN so that it can be unsealed.

   (e) The volatile steate is then encrypted and authenticated using the shared secret between the SMN and TMN.

   (f) In parallel, a reference hash is computed over the persistent state of the migrated application, incorporating the mapping of program symbols, relocation segments, and the expected system call graph that was distributed during the enrollment phase. This process ensures that the migrated state corresponds to a valid and untampered execution flow.

   (g) The encrypted state, authentication digest, public key of the SMN, the corresponding certificate, and the fresh nonce are sent to the TMN for validation.

3. **Verifiable Import Phase:** After the encrypted volatile state to be migrated is received to the TMN, this phase is dedicated to taking the necessary actions for ensuring that the execution of the application is securely resumed in the trusted environment of the new device. This is achieved though replaying the state, generating the System Call Control Flow Graph (SC-CFG), and attesting to its correctness. Specifically, this process consists of the following steps:

   (a) To decrypt the received volatile state, the Migration Service derives the same shared secret used by the source by performing the ECDH operation with its own private key and the public

key of the source migration node. This shared secret is then passed through the agreed key derivation function, resulting in a symmetric encryption key.

(b) With this key, the Migration Service of the TMN decrypts the volatile state and recomputes the HMAC-SHA256 digest, comparing it to the received one. A match confirms both the integrity and authenticity of the transferred state.

(c) Following decryption, the volatile state is sealed again into a file using the signing key of the TMN in order to bind it to the enclave's trusted identity.

(d) The migrated trusted application is then launched, as a child enclave of the Migration Service of the TMN, unsealing its volatile state, in a manner that this does not incur any additional overhead for safety-critical applications from the normal enclave launch.

(e) From this point, the Migration Service of the TMN performs runtime introspection of the enclave to validate the success and integrity of the migration. This introspection observes the memory region occupied by the migrated trusted application to extract its program symbols, relocation segments, and other aspects of its persistent state, while also capturing the control flow of system calls made during the application's re-launch. These observations ensure that (i) the enclave is correctly positioned in the memory, that (ii) no unauthorized modifications have occurred, and that (ii) the application has successfully loaded the migrated volatile state, allowing it to continue from where it was stopped on the SMN. In particular, the observed sequence of system calls serves as implicit proof that the application correctly restored the received volatile state, thereby ensuring execution continuity across devices and mitigating potential attacks.

(f) A secure hash is computed over the introspected data along with the received nonce, producing an updated authentication digest that reflects a successful and trustworthy migration.

(g) To complete the phase, the Migration Service updates the pigeonhole mapping with the measurement of the newly launched enclave, marking it as an active and validated instance within the system.

(h) The final step in the Verifiable Import phase is updating the pigeonhole mapping with the newly launched enclave's measurement. This ensures that the enclave is registered as an active and validated execution environment.

2.3.5 REWIRE Tracing Capabilities

As part of the REWIRE architecture, the core orthogonal technology that supports the execution of runtime attestation processes is the novel **efficient tracing mechanism** of REWIRE. In this context, as documented in D2.1 [21] and further explained in D4.2 [23], *REWIRE is one of the first projects of its kind to envision the provision of tracing capabilities capturing operational and configuration details of binaries, compiled for RISC-V architectures*. Note that, while the initially proposed methodology entailed the integration of monitoring hooks for enabling the runtime tracing of the execution flow of the binary, this approach requires *access to the source code of the traced binary* for the installation of these monitoring hooks, which is considered as a strong prerequisite that may significantly limit the feasibility and applicability of the provided solution in real-world scenarios where software is IP protected, thus solutions requiring access to the source code may be considered intrusive.

Therefore, we opted for a less intrusive tracing methodology which alleviates this assumption, namely a **tracing Daemon** which operates in the background and establishes a secure communication channel with the **Attestation Agent (AA)**, so that the required traces are collected in a non-intrusive manner during the operation of the target application, and afterwards signed and forwarded to the AA. In this regard, we consider that the REWIRE Tracer is capable of monitoring two types of binaries:

- **Stateful binaries:** This case refers to the execution of a binary with a finite execution flow, that may receive one or more input, and terminates after the execution of the process. In this case, the Daemon receives the process ID as input, and monitors and extracts the configuration hash.
- **Stateless binaries:** This refers to the case of a binary that runs continuously. In this case, every time a loop instance of the binary is executed, the traces corresponding to the configuration hash are extracted.

Note that the configuration hash is treated as a **configuration profile**, meaning the metadata of a binary that capture or depict its operational boundaries, captured using commands such as *ld_preload* that enable accessing and reading such metadata from the binaries. Thus, the program symbols to be used during runtime (e.g., list of libraries or system calls that the binary is allowed or expected to run during runtime) are the baseline of the configuration profile to be extracted. As further research to be performed, we will investigate the feasibility of a further extended Tracer that is able to identify the range of values of the memory stack can be accessed by a binary during runtime by calculating the offsets that capture and represent the region of the stack space that can be accessed or processed, as this is currently considered in the state-of-the-art as a particularly challenging issue.

Throughout this Section, we will provide further details on the operation of this tracing approach, as well as its positioning within the overall REWIRE framework.

2.3.5.1 Tracing in Tandem with REWIRE Trusted Computing Base (TCB)

In order to better illustrate the architectural approach of the REWIRE tracing capabilities, here we provide some information on its use in tandem with the **Trusted Computing Base (TCB)** of the device. Note that the TCB refers to a set of **hardware, software, and firmware components** which are supposed to be resistant to any type of attacks specified by the adversarial model, and are assumed secure by default. The TCB is required for implementing primitives such as attestation, which provide security guarantees to the device. In general, the following types of TCB can be defined:

- **Hardware-based TCB:** In this case, the System-on-Chip (SoC) hardware component, including the processor, system buses, and peripherals, are part of the TCB. Moreover, the existence of a HW-based trusted component (e.g., RISC-V TEE) is assumed, which provides trustworthy and correct outputs. Such environments possess processors which enable the creation of hardware-assisted trusted execution environments, such as Keystone in RISC-V environments.
- **Software-based TCB:** In this case, the firmware of the device is assumed to be part of its TCB. The firmware is responsible for booting up the device, before loading both the secure world operating system (OS) and the normal world boot loader and OS. Furthermore, it is assumed that each boot stage in the firmware measures and validates the next boot stage, and extends the loaded image hashes into the secure element for later providing an attestation report on the loaded software, including the secure world OS image. Note that the firmware implements the *security monitor*, which provides a secure context switch between normal and secure world, and prevents leaking the internal states of the secure world to the normal world.

In general, we consider that a device consists of two distinct parts: (i) the **trusted world**, which constitutes the TCB of the device and consists of the aforementioned types of components which are resistant to the specified attack surface, and (ii) the **untrusted world**, which refers to the part of the device that cannot be considered trusted. In the case of RISC-V architectures considered in REWIRE, when running a trusted application in the trusted world as an enclave, an untrusted counterpart is also created in the host. In this case, constant communication is performed between the trusted and untrusted counterpart of the application for orchestration purposes, as well as the exchange of various types of application and operational data. In general, untrusted-trusted world communication is facilitated through **ocalls**.

In REWIRE, the Attestation Agent runs on the **trusted world** of the host device (while also having a counterpart in the untrusted world), and acts as the bridge for the correct orchestration of the Tracer, based on the attestation tasks and security enablers that need to be executed. Conversely, we consider that the Tracer is a software component which belongs to the **trusted world** and is part of the TCB of the edge device. In addition, the Tracer has its own **Tracer Key (TK)** that enables it to sign the collected traces. As aforementioned, the tracer Daemon continuously performs introspection of the target binary and is able to monitor its **configuration hash**. Thus, following the collection of these traces, communication between the trusted and untrusted world occurs between the Tracer (trusted) and the Attestation Agent (untrusted) for the exchange of traces, signed with the TK, used in order to perform attestation actions.

2.3.5.2 Operational Mode of the REWIRE Tracing Flow

Here, we provide a detailed action workflow of the REWIRE Tracer in the context of an attestation process, including the issuance of an attestation challenge, the collection and signing of the required traces as attestation evidence, and the verification of the Traces by the Verifier entity. Here, we provide detailed descriptions of the action workflows for the two modalities of the Tracer outlined in the previous Section, namely the tracing of **stateful binaries** and the tracing of **stateful binaries**, as also depicted in Figure 2.7.

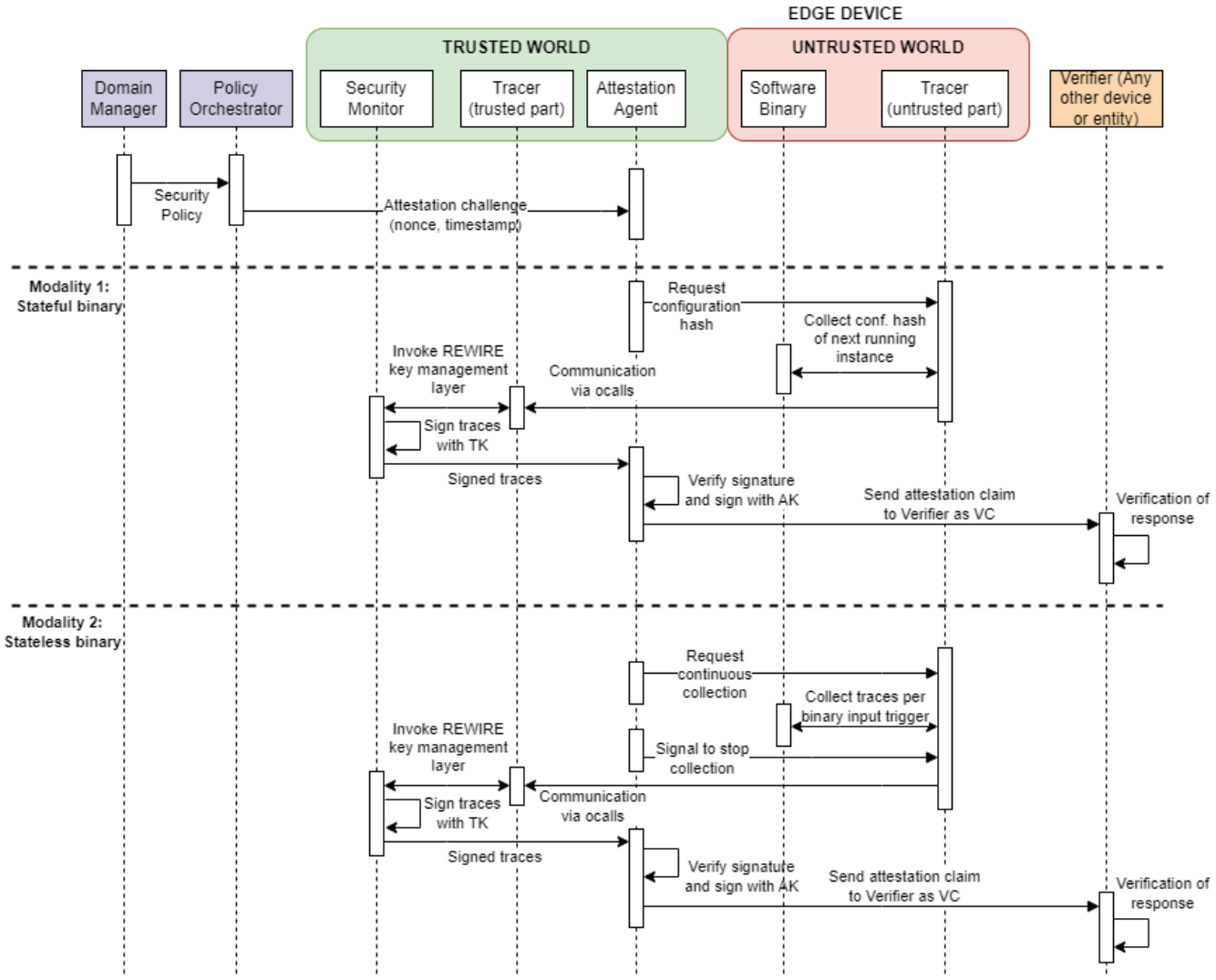

Figure 2.7: REWIRE Tracer sequence diagram for stateful and stateless binary modalities

The action workflow of the **tracing of a Stateful Binary (Modality 1)** consists of the following steps:

1. The **Domain Manager** sends the security policy to the **Policy Orchestrator**, so that the target device can attest to the correctness of its configuration state through an attestation process by providing the required traces as attestation evidence.
2. The Policy Orchestrator forwards the attestation challenge to the **Attestation Agent** of the device to be attested, consisting of a random *nonce* and a *timestamp* that will eventually be used in order to verify the validity of the attestation response.
3. The Attestation Agent requests the configuration hash from the part of the Tracer in the untrusted world, in order to initiate the collection of the appropriate attestation evidence.
4. The Tracer collects the configuration hash from *the next running instance of the binary to be traced*.
5. Via an *ocall*, the untrusted part of the Tracer communicates with its counterpart in the trusted world, so that the **REWIRE Key Management Layer** of the **Security Monitor** is invoked.
6. The traces are signed with the **Tracer Key** at the side of the Security Monitor, and the signed traces are sent to the Attestation Agent.
7. The Attestation Agent verifies the traces, and signs them with the **Attestation Key**.
8. Item the attestation claim is sent from the Attestation Agent to the Verifier (which may be any other device or entity) in the form of a Verifiable Credential, including the nonce and the timestamp.
9. The Verifier verifies the response to the attestation challenge. If successful, the device is considered to be in a correct configuration state.

The action workflow of the **tracing of a Stateless Binary (Modality 2)** consists of the following steps:

1. The **Domain Manager** sends the security policy to the **Policy Orchestrator**, so that the target device can attest to the correctness of its configuration state through an attestation process by providing the required traces as attestation evidence.
2. The Policy Orchestrator forwards the attestation challenge to the **Attestation Agent** of the device to be attested, consisting of a random *nonce* and a *timestamp* that will eventually be used in order to verify the validity of the attestation response.
3. The Attestation Agent requests the configuration hash from the part of the Tracer in the untrusted world, in order to initiate the collection of the appropriate attestation evidence.
4. The Attestation Agent requests from the Tracer the initiation of the collection of traces from the target software binary. Note that the collection of traces continues until the Attestation Agent requests the collection to stop, based on the periodicity specified in the security policy.
5. The Tracer instance residing in the untrusted world collects traces from the target binary per input trigger, as the provision of input signifies the execution of the binary.
6. The Attestation Agent signals the Tracer to stop the collection of attestation evidence, as aforementioned.
7. Via an *ocall*, the untrusted part of the Tracer communicates with its counterpart in the trusted world, so that the **REWIRE Key Management Layer** of the **Security Monitor** is invoked.
8. The traces are signed with the **Tracer Key** at the side of the Security Monitor, and the signed traces are sent to the Attestation Agent.
9. The Attestation Agent verifies the traces, and signs them with the **Attestation Key**.

10. Item the attestation claim is sent from the Attestation Agent to the Verifier (which may be any other device or entity) in the form of a Verifiable Credential, including the nonce and the timestamp.

11. The Verifier verifies the response to the attestation challenge. If successful, the device is considered to be in a correct configuration state.

2.4 REWIRE Blockchain Infrastructure for Auditable Data Transactions

REWIRE uses a **policy-compliant Blockchain infrastructure** in order to achieve the auditability and certifiability of data transactions, and provides the capability for secure policy enforcement. In this regard, a core innovation of REWIRE as part of its extended trust considerations is the veracity of data transactions, when they are recorded on-chain. To this end, REWIRE introduces the notion of the **Secure Oracle Layer** depicted in Figure 2.11, and innovates by demonstrating how this can operate in symbiosis with different technologies, such as **Trusted Computing (TC)-based enablers for data transactions**, and **Fabric Private Chaincode (FPC) for managing attestation-related applications**. The REWIRE Blockchain Infrastructure provides the capability for managing both **application-** and **attestation-related data**, and in the following we elaborate on each of the corresponding data flows.

The **application-related data flow** in the REWIRE secure oracle system ensures the secure handling of sensitive data from its generation to storage and retrieval, maintaining privacy, auditability, and compliance with blockchain policies. For example, in the smart satellite use case, the data provider generates satellite-related application data, which is then processed through the Town Crier secure oracle. Within the SGX enclave, the data undergoes integrity and authenticity checks using cryptographic methods. Once verified, the data is stored either directly on the Besu Blockchain (if ≤ 20 KB) or in off-chain storage with a reference pointer recorded on-chain for larger datasets. This hybrid approach optimizes storage efficiency while ensuring data integrity and accessibility. By leveraging the secure oracle and blockchain infrastructure, the system bridges the gap between application data and privacy-preserving, scalable storage.

Authorized parties can query the stored data by sending a request to the Besu Blockchain. The blockchain processes the query to locate the required data. If the data size is within the on-chain storage limit (≤ 20 KB), it is retrieved directly from the blockchain. For larger datasets, only a reference to off-chain storage is kept on-chain, and the blockchain retrieves this reference instead. The queried data or its reference is then returned to the application layer, ensuring secure and seamless data access. This flow guarantees efficient handling of both on-chain and off-chain data, maintaining scalability while preserving the integrity and accessibility of application-related data.

The **attestation-related data flow** in the REWIRE secure oracle system ensures the secure handling of attestation reports, verifying the integrity and authenticity of devices and their execution environments while maintaining compliance with blockchain policies. For example, when a device generates an attestation report, it notifies the TownCrier-Besu component through the Security Context Broker (SCB). The Town Crier secure oracle retrieves the report and processes it within the SGX enclave, where the integrity of the report and the authenticity of the generating device are verified using cryptographic methods. Once verified, the attestation report is stored on the Besu Blockchain. By leveraging the secure oracle and blockchain infrastructure, this flow enables a seamless bridge between device attestation and scalable, privacy-preserving storage. When the attestation report is successfully recorded on the blockchain (Besu), the FPC component is notified via WebSockets that a new report is available. Through the SCB, the FPC component retrieves the newly posted report from Besu and writes it to the FPC ledger. This process integrates the attestation report securely into the chaincode execution layer operating within a Trusted Execution Environment (TEE), ensuring trust and privacy at every stage.

Authorized parties can then query the latest attestation report of a specific device or retrieve all attestation reports of a specific device through the SCB on FPC. This query functionality ensures seamless access to attestation data while preserving the system's scalability and security. In this initial implementation, the user's signature verification occurs in the Town Crier's SGX enclave, while the user attribute verification is performed by the SCB before the attestation report is posted to the FPC ledger. This ensures robust and secure management of attestation data within a trusted, decentralized ecosystem.

One core design choice of REWIRE, which was introduced in D5.1 and elaborated in D5.2, was **the selection of FPC for providing the capability of the execution of smart contracts inside an enclave**. While FPC is a strong technology to provide guarantees even in on-chain execution, it poses challenges regarding **flexibility in the introduction of additional security mechanisms**. Specifically, the construction of FPC entails launching ane executing smart contracts in an enclave wrapped inside a container, which makes it difficult to support TC-based and crypto extensions without changing the inherent structure of the FPC. Thus, as part of the vision of REWIRE to be able to add strong crypto extensions to the Blockchain infrastructure, such as the compositional Attribute-Based Access Control (ABAC) and the ability to use VCs for access control with proof of knowledge on the attributes, REWIRE *investigates equivalent technologies with FPC, namely Phala, that provides additional levels of flexibility with regards to its capability for integration of crypto schemes*.

In this context, and considering the ongoing research efforts of REWIRE into improving the integration of TownCrier with a TEE-based blockchain infrastructure, we are currently exploring the Phala Network as a possible alternative to FPC. Phala is an EVM-compatible blockchain that also executes smart contracts within a Trusted Execution Environment (TEE), similar to FPC. This shared characteristic makes it an interesting candidate for addressing some of the challenges we face in integrating TownCrier with a non-EVM-based system like FPC.

One of the key motivations for investigating Phala is its event-driven execution model, which may offer a way to facilitate direct communication between TownCrier and a blockchain-based TEE environment. Given that TownCrier is designed to operate with EVM-compatible systems, Phala's compatibility with Ethereum-based tools and oracles could help mitigate the existing integration constraints. However, further research is required to determine whether Phala's execution model and TEE-based smart contract framework can fully support our requirements for secure, verifiable, and privacy-preserving attestation data flows.

Another factor under consideration is Phala's interoperability with secure oracles. Since it is designed to handle privacy-preserving smart contract execution, it could potentially enable a more flexible and efficient approach to attestation data storage and verification. However, before any conclusions can be drawn, we need to further evaluate how Phala's architecture interacts with existing components in the REWIRE framework, particularly in relation to data verification, policy enforcement, and real-time event processing.

At this stage, our research into Phala is exploratory, and no final decisions have been made regarding its adoption. While it presents certain advantages in terms of its TEE-based execution model and EVM compatibility, further analysis is required to assess whether it aligns with our long-term objectives for attestation data management and secure oracle integration. Our goal is to determine whether Phala can provide a viable alternative to FPC in supporting efficient, secure, and scalable attestation flows, but this remains an open research topic that requires additional evaluation and testing.

2.4.1 Application-related Data Management

Figure 2.8 illustrates the application-related data flow within the REWIRE framework, specifically detailing how an external entity(eg. the AI-based misbehaviour detection component) requests satellite application data through the secure blockchain-based infrastructure.

The process begins with an external entity, such as an AI-based Misbehavior Detection Engine, submitting a request for application data. This request is generated from the entity's wallet and includes a Verifiable Presentation (VP), which serves as cryptographic proof of the requestor's credentials.

Upon receiving the request, the Security Context Broker (SCB) acts as the authorization mediator, ensuring that the requestor possesses the required attributes to interact with the blockchain infrastructure. The SCB validates the VP attributes against predefined access policies. If the validation fails, the request is rejected. If the validation succeeds, the SCB forwards the request to the Besu-aligned distributed ledger for further processing.

The Besu blockchain records the request as an event, ensuring traceability and auditability. At this point, the TownCrier Secure Oracle, which continuously listens for new events on the Besu blockchain, detects the newly logged event. TownCrier then triggers a query to an External Trusted Data Source to retrieve the requested application data.

Once the External Trusted Data Source processes the request, it returns the requested application data to the TownCrier Secure Oracle. Before relaying this data back into the blockchain, TownCrier verifies the requestor's signature from the VP to ensure that the data request was legitimate and authorized. If the signature verification fails, the process is halted, and the data is discarded. If the verification succeeds, the verified application data is submitted to the Besu blockchain for storage.

Upon successful storage of the application data, a confirmation message is sent from the Besu blockchain back to the Security Context Broker (SCB). The SCB then notifies the external entity that the requested application data is now available, completing the flow.

This process ensures secure, verifiable, and decentralized access to application-related data while integrating blockchain infrastructure with trusted oracles and external data sources.

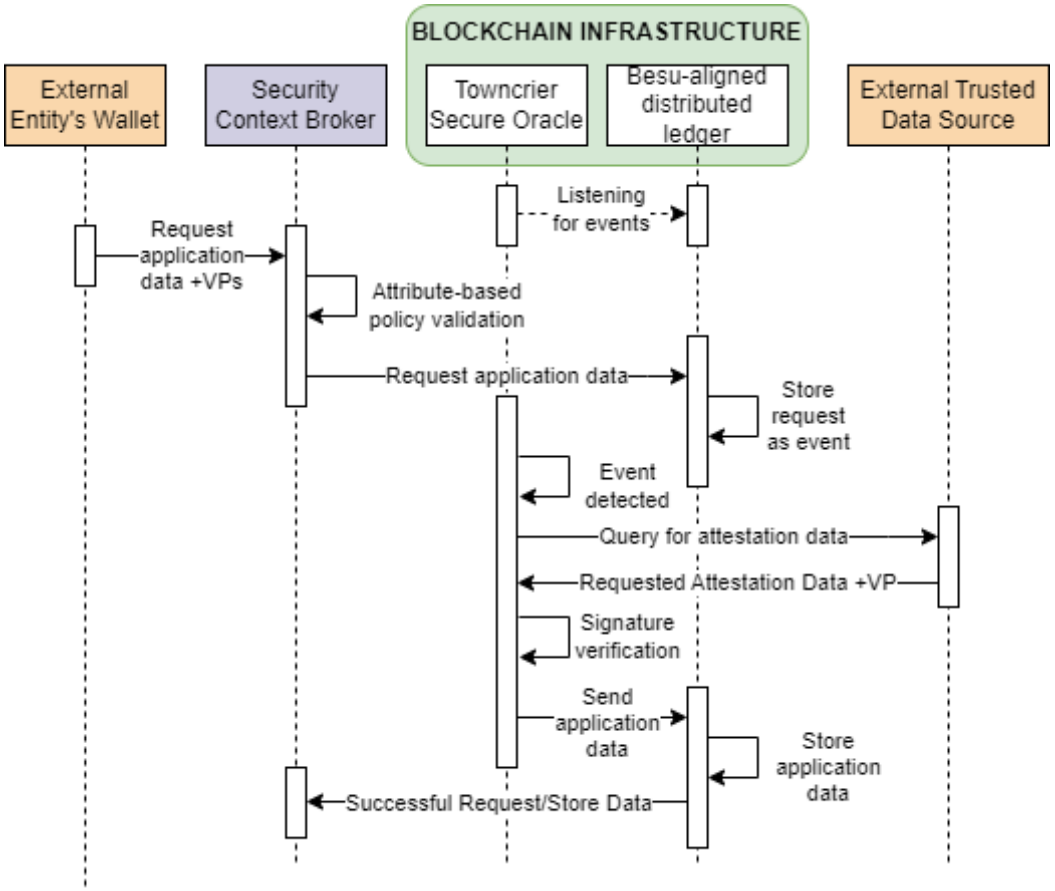

Figure 2.8: REWIRE Application-related Data Management Sequence Diagram

2.4.2 Attestation-related Data Management

The attestation-related data flow in the REWIRE framework, depicted in Figure 2.9 ensures the secure and verifiable handling of attestation reports from devices to the blockchain infrastructure. This process integrates SCB, TownCrier, Besu, and FPC, ensuring that only valid, authenticated attestation data is stored in the system.

When a device finalizes an attestation report, the SCB, which continuously monitors device attestation events, detects this action. Upon detection, SCB triggers a request for the attestation data, ensuring that the finalized report can be securely stored in the blockchain infrastructure.

Upon receiving this request, SCB forwards it to the Besu blockchain, where it is logged as an event. Since the TownCrier Secure Oracle constantly listens for new events in Besu, it detects the new request event and initiates the next step of the process. TownCrier then queries the specific device that generated the finalized attestation report, requesting the attestation data for further verification and storage.

The device responds with the requested attestation data, along with its Verifiable Presentation (VP), which serves as cryptographic proof of the device's identity and authorization. Before proceeding further, TownCrier first verifies the VP's signature to ensure the authenticity and correctness of the attestation data. If verification fails, the data is discarded. If verification succeeds, TownCrier forwards the verified attestation data to the Besu blockchain for storage.

At this stage, SCB listens for new events on the Besu blockchain through WebSockets. Upon detecting that attestation data has been successfully stored in Besu, SCB triggers the next step in the process: a request to retrieve the attestation data from Besu. This request is accompanied by the device's VP, ensuring that the retrieved data is correctly linked to the original attestation request.

Once the attestation data is successfully retrieved from Besu, SCB forwards it to the FPC ledger for final storage. Before the data is stored, FPC performs another round of signature verification to confirm its authenticity and ensure that no tampering has occurred. If the verification fails, the data is discarded. If verification succeeds, the attestation data is securely stored in FPC.

Finally, the SCB registers a successful request message, signaling that the attestation data has been securely verified, stored, and is now available for trusted queries. This ensures a complete, auditable, and decentralized process for attestation management in the REWIRE framework.

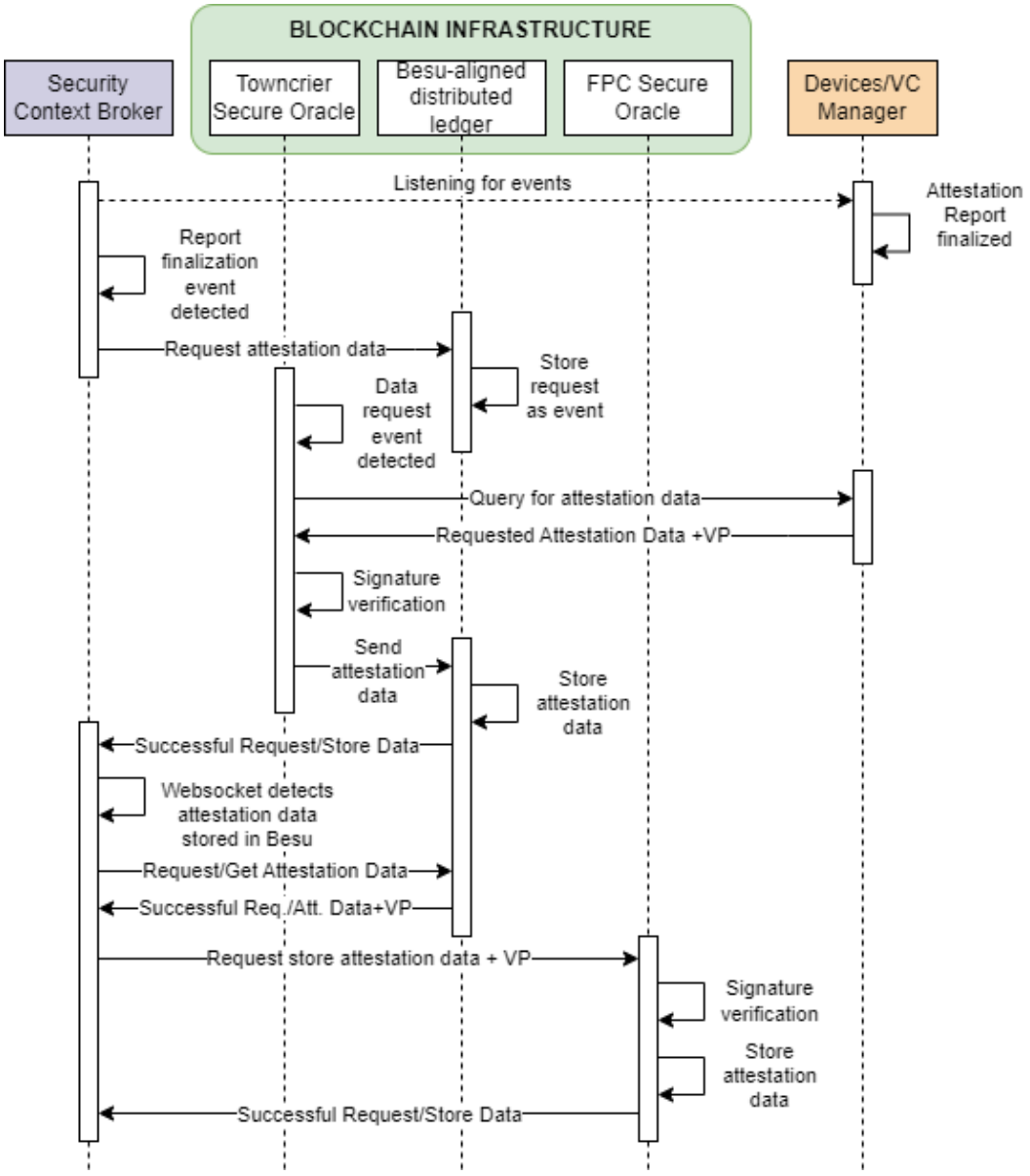

Figure 2.9: REWIRE Attestation-related Data Management Sequence Diagram

2.4.3 Attestation-related Data Management with Direct TC and FPC Integration

The endmost attestation-related data flow depicted in Figure 2.10 envisions a direct integration between the TownCrier Secure Oracle and Fabric Private Chaincode (FPC), eliminating the need for Besu blockchain as a mediator. This approach seeks to create a more efficient and streamlined attestation pipeline, ensuring that attestation data remains secure, verifiable, and privacy-preserving while reducing redundant steps.

The process begins when a device (VC Manager) finalizes an attestation report. The SCB continuously listens for device-generated attestation events, and upon detecting such an event, it triggers a request for attestation data from the corresponding device.

Once the request is triggered, SCB forwards it directly to TownCrier, instead of passing it through Besu, as in the current implementation. TownCrier then logs this request as an event and prepares to fetch the required attestation data.

Upon detecting the request, TownCrier queries the specific device that generated the attestation report, requesting the attestation data. The device responds with the requested attestation data along with

its Verifiable Presentation (VP), ensuring cryptographic proof of the data's authenticity and the device's identity.

Before submitting this data for final storage, TownCrier performs an initial signature verification to validate the authenticity of the attestation data and the legitimacy of the requesting device. If the verification fails, the attestation data is discarded. If the verification succeeds, TownCrier forwards the attestation data directly to FPC for storage, ensuring that the process remains fully decentralized and privacy-preserving.

Within FPC, the attestation data undergoes a final round of verification before being securely stored in the TEE-based ledger. Upon successful storage, FPC generates a confirmation message, which is sent to SCB. The SCB then notifies the external requesting entity, confirming that the attestation data has been successfully processed and stored in the ledger.

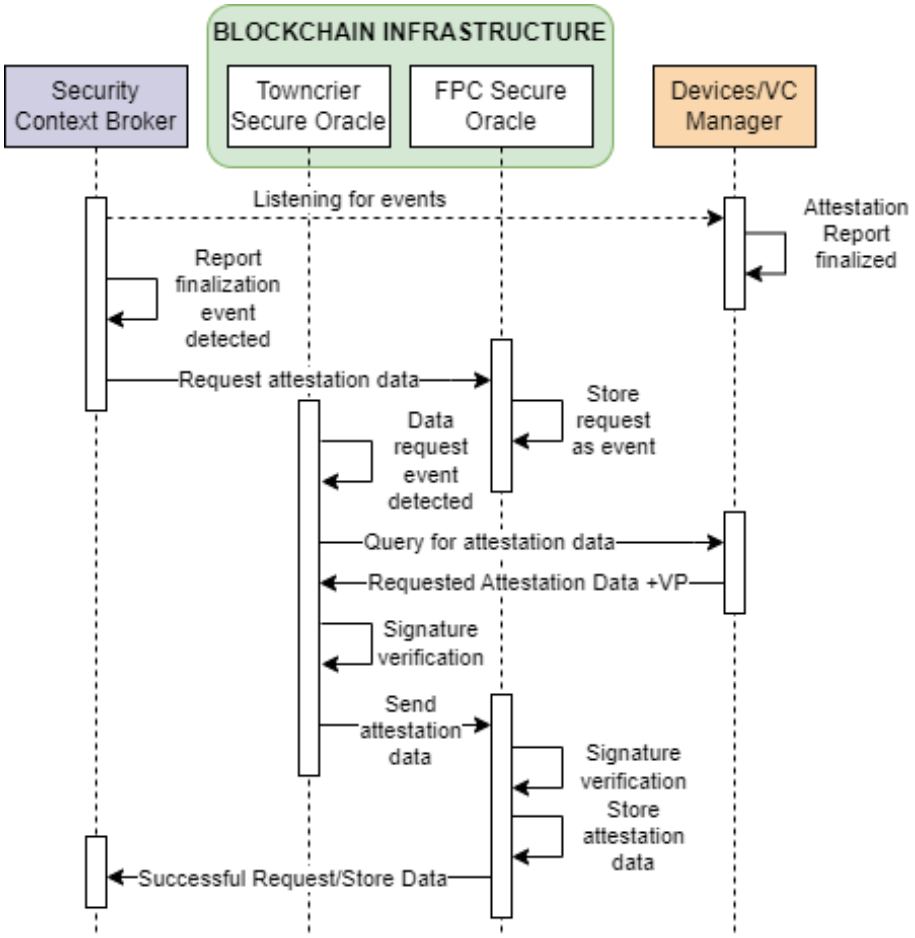

Figure 2.10: REWIRE Attestation-related Data Management Sequence Diagram with direct Town Crier and Fabric Private Chaincode Integration

2.5 REWIRE Secure Oracle Layer

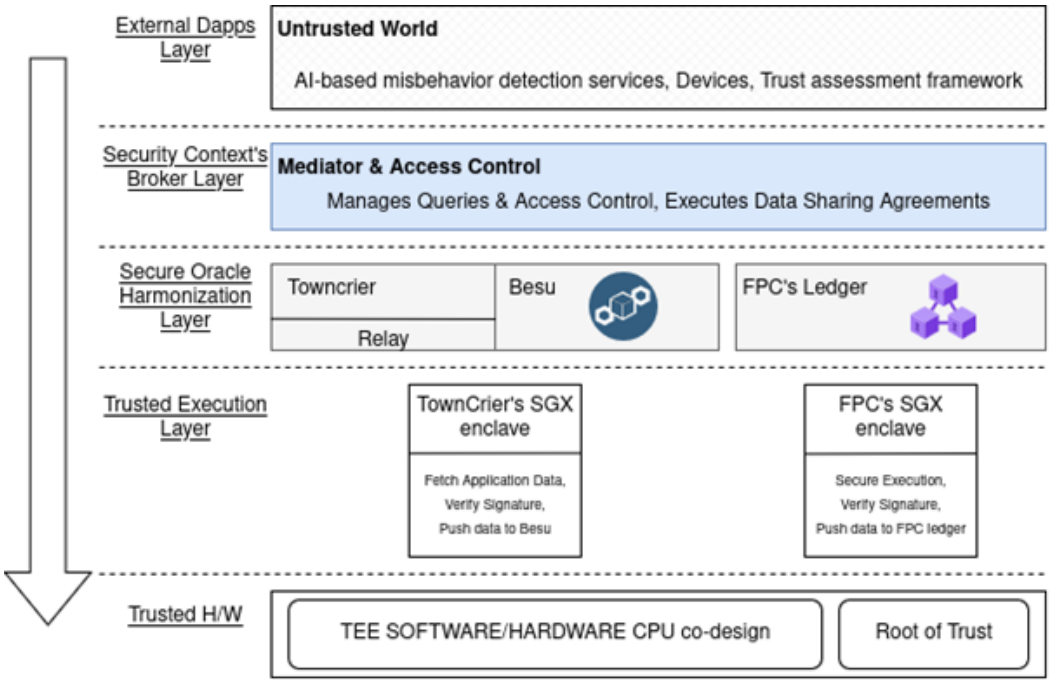

Figure 2.11: Multi-layer data management architecture of REWIRE

The REWIRE framework follows a multi-layered architecture, depicted at a high level in Figure 2.11, designed to ensure secure, verifiable, and privacy-preserving data management by integrating secure oracles, blockchain infrastructures, and trusted execution environments (TEE). This architecture enables external applications and services to interact with REWIRE's blockchain components while enforcing strict security policies, access controls, and data-sharing agreements.

At the top, the External DApps Layer represents the untrusted world, where applications such as AI-based misbehavior detection services, trust assessment frameworks, and devices operate. These external applications generate requests for application or attestation data but cannot directly access REWIRE's blockchain components. Instead, they must go through the Security Context Broker (SCB) Layer, which acts as a bridge and access control mechanism.

The Security Context Broker (SCB) Layer plays a crucial role in managing access control, query validation, and policy execution. Acting as a trusted intermediary, it ensures that all incoming queries are authenticated and authorized before being forwarded to the blockchain infrastructure. Additionally, the SCB executes data-sharing agreements, ensuring that only authorized entities can access and retrieve data from the system. This layer abstracts the complexity of interacting with REWIRE's secure blockchain components and exposes a controlled interface for external applications.

While SCB regulates access, the Secure Oracle Harmonization Layer provides a standardized communication framework between SCB and the blockchain infrastructures. This layer, introduced by REWIRE, serves as a shadow layer that enables seamless interaction with various secure oracle technologies. By exposing a harmonized API, this layer ensures that external applications can communicate uniformly with different blockchain infrastructures, regardless of their underlying architecture.

The Secure Oracle Harmonization Layer includes:

- TownCrier Secure Oracle, which is responsible for fetching external application data, verifying signatures, and pushing data securely to the blockchain.
- enabled distributed ledger handling application-related data.

- contracts for attestation data storage and verification.

By placing Besu and FPC within the Secure Oracle Harmonization Layer, REWIRE ensures that all blockchain-based operations follow a unified and verifiable approach to secure data management, regardless of the underlying oracle mechanism.

Beneath this layer, the Trusted Execution Layer ensures that all cryptographic verifications, chaincode execution, and blockchain interactions occur within hardware-protected environments. This layer consists of:

- TownCrier's SGX Enclave, which fetches, verifies, and securely submits external data to the blockchain.
- FPC's SGX Enclave, which is responsible for secure execution of attestation data, cryptographic verifications, and private storage within the FPC ledger.

By leveraging Intel SGX enclaves, REWIRE guarantees that even if an attacker compromises the host system, they cannot access or tamper with sensitive data being processed within the enclave.

At the lowest level, the Trusted Hardware Layer forms the foundation of REWIRE's security architecture. It includes:

- TEE Software/Hardware CPU Co-Design, ensuring that all sensitive computations take place within a trusted execution environment.
- Root of Trust, providing hardware-based assurances that only authenticated and verified code can execute within the system.

This multi-layered approach guarantees security, privacy, and verifiability throughout the data lifecycle, from the moment an external application submits a request to the final secure storage of data in the blockchain. The SCB Layer acts as the mediator, managing access control and query handling, while the Secure Oracle Harmonization Layer ensures standardized blockchain interactions. Through the integration of TEE-based execution, secure oracles, and harmonized blockchain interfaces, REWIRE provides a scalable, future-proof solution for privacy-preserving, verifiable, and trusted data exchange.

2.5.1 Key Constraint

A major challenge in implementing the ideal flow described in Section 2.4.3 lies in TownCrier's inability to directly listen for events in FPC. This limitation is represented by the red arrow in the diagram, which highlights the missing functionality that prevents TownCrier from interacting natively with FPC. Currently, TownCrier is designed to work exclusively with EVM-based blockchains (such as Ethereum, Besu, and GoQuorum), where it can listen for on-chain events and react accordingly. However, FPC does not follow the EVM model, and its event structure is fundamentally different from what TownCrier is designed to process. This discrepancy prevents TownCrier from directly fetching attestation events from FPC, forcing the use of Besu as an intermediary in the current implementation. Due to this constraint, we are currently unable to fully eliminate Besu from the attestation flow, leading to increased complexity, redundant event logging, and additional processing delays. Our end goal is to enable TownCrier to directly integrate with FPC, eliminating the dependency on Besu and achieving a more optimized, decentralized, and efficient attestation pipeline.

2.6 REWIRE Key Management System

Crypto operations lie in the heart of REWIRE and are part of core security enablers and functionalities, such as the establishment of authenticated channels for performing secure software updates. Such operations revolve around the **Key Management System (KMS)** of REWIRE, which provides the capability for management of the various cryptographic keys used as part of these processes. More specifically, encryption is one of the most effective measures to secure sensitive data, prevent unauthorized access, and guarantee data integrity. Encryption keys are the cornerstone of this process. Anyone who possesses these keys can decrypt protected data or impersonate key owners. If encryption keys are stolen or mishandled, unauthorized users may gain access to sensitive information. Conversely, if keys are lost, even authorized users may be permanently locked out of their data. It is therefore crucial to understand that the effectiveness of encryption relies not only on strong algorithms but also on the secure management of the encryption keys that protect the data.

A Key Management System (KMS) is a framework for generating, exchanging, storing, and managing cryptographic materials to ensure their security. A KMS centralizes control over keys and manages them throughout their lifecycle according to best practices. Systems may use different types of keys (e.g., symmetric or asymmetric) for various purposes (e.g., attestation or data sealing), and designing a comprehensive strategy to manage all of them—while accounting for their differing requirements—is a non-trivial task.

In REWIRE, we assume the absence of a cryptographic co-processor capable of generating keys, managing them, and providing cryptographic services to enclaves or applications. Therefore, we have chosen to implement the key management system inside the Security Monitor (SM), making it part of the Trusted Computing Base (TCB), the trusted core of our system. However, if a system running REWIRE includes such a hardware module, the KMS would act as a wrapper and mediator for access to that component.

2.6.1 Harmonized Key Management System as an SM Extension

As mentioned earlier, a KMS must keep encryption keys secure throughout their entire lifecycle—from creation to eventual destruction. This lifecycle includes key generation, storage, distribution, usage, rotation, and revocation or destruction. In REWIRE, the KMS is part of the SM, which gives it full control over the keys. This approach increases the TCB size, but ensures that the most critical assets are managed by the most trusted software component.

To implement the Harmonized Key Management (HKM) system, REWIRE draws inspiration from OP-TEE [15], an open-source Trusted Execution Environment (TEE) primarily designed for ARM TrustZone and compliant with GlobalPlatform specifications. Specifically, we reference the GlobalPlatform TEE Internal Core API specification [8], which defines the API exposed to Trusted Applications. However, we do not strictly adhere to these specifications.

Most cryptographic services in REWIRE are provided by a cryptographic library integrated into the SM. Thus, operations such as key generation are executed in the most privileged mode, where interrupts are not allowed. It's important to note that widely used cryptographic libraries have undergone extensive testing and analysis, making them more secure than ad-hoc implementations.

REWIRE unifies all interfaces for key generation and management, as well as for obtaining the results of basic cryptographic operations. Requests are made with various arguments—some mandatory, others dependent on the specific operation. The SM performs validation and consistency checks before forwarding requests to the cryptographic library. Supported key types include elliptic curve keys for attestation, symmetric keys for sealing, Tracer Keys (as previously discussed), update keys (LRBC keys), and migration keys.

All keys generated for a particular enclave are securely stored by the SM in a protected area, which also contains additional enclave-related metadata. When a request is made to retrieve a previously generated

key, it is fetched from this secure storage. Note that any keys not hardcoded or derived from hardware components may be lost if the system loses power before the keys are persisted. These keys are isolated from other enclaves and from the operating system and are never exposed to the untrusted world.

The REWIRE SM extension to manage the keys and its interfaced will be fully described in Deliverable 4.3. However, we briefly hint at how this Key Management System has been designed. In Table 2.2, we provide descriptions of the core functionalities performed by the REWIRE KMS, as well as the inputs and outputs of each function. We also provide a mapping of each function with its corresponding GlobalPlatofrm API functionalities. Note that detailed descriptions of these APIs are provided in D4.2 [23].

Functionality	Description	Input	Output	GlobalPlatform API
Key Generation	While some keys for the enclaves are generated when the enclave is created (e.g. attestation keys), some others are created upon the request of the enclave. This function generates cryptographic keys tailored for secure enclaves and applications, using information from a provided structure. The keys adhere to specified cryptographic standards and are securely managed within the Security Monitor, which stores them in a reserved area allocated for that specific enclave. All the inputs are validated to ensure that all the fields are properly set.	Struct with all the required parameters and information to generate the keys. Such information includes the expected utilization of the key (e.g. tracer) or the name of the cryptographic algorithm, the enclave identifier, and key length.	Returns a 0 when successful or an error code on failure (e.g. invalid input, unsupported key type, or internal error during generation.)	TEE_GenerateKey, TEE_InitValueAttribute, TEE_DeriveKey
Key Retrieval	Most of the keys never leave the Security Monitor, however, for public key cryptographic schemes, the public key should be shared. Therefore, we provide an interface to retrieve such keys. Note that not only can the enclaves make such requests, but any other application that needs to validate some data coming from the enclave might make use of such an interface.	Struct with all the required parameters and information about the expected keys. Such information includes the expected key with its codename and the enclave identifier.	Returns the requested key when successful or an error code on failure (e.g. invalid input, not enough privileges, or internal error during retrieval.)	TEE_SetOperationKey, TEE_DeriveKey, TEE_GetObjectValueAttribute, TEE_CopyObjectAttributes1
Cryptographic Operations	This function performs cryptographic operations such as encryption, decryption, signing, and verification. It supports various cryptographic algorithms and ensures that all operations are securely executed within the Security Monitor. It is mainly intended to create hashes or digests of data, its validation, and signature verification, although encryption and decryption are also supported. Note that, as in previous cases, the inputs are validated before executing any operation.	Struct with all the required parameters to execute the requested operation. Such information includes the inputs (plaintext or ciphertext), the desired operation (e.g. AES encryption), or the name of the cryptographic operation that has to be executed, or the enclave identifier.)	Returns the result when successful (or a pointer to the result altogether with its length) or an error code on failure (e.g. invalid input, unsupported algorithm, not enough privileges, or internal error during generation.)	TEE_SetOperationKey, TEE_AllocateOperation, TEE_OperationMode, TEE_CipherInit, TEE_AE_Encrypt, TEE_AE_Decrypt, TEE_AE_Sign, TEE_AE_Verify

Sealing and Unsealing Operations	Although this function also performs cryptographic operations (encryption, decryption, signing, and verification) it is slightly different than the previous ones as the result (data encrypted with a symmetric key) can be directly saved to disc and it is partially binded to the device and identifier of the enclave. Keystone also includes sealing and unsealing functions, and this functionality is partially a wrapper over the functions implemented in Keystone	Struct with all the required parameters to execute the requested operation. Such information includes the inputs (plaintext or ciphertext), the desired operation (e.g. sealing, unsealing), and the enclave identifier.)	Returns the result when successful (or a pointer to the result altogether with its length) or an error code on failure (e.g. invalid input, not enough privileges, or internal error during generation.)	TEE_SetOperationKey, TEE_DeriveKey, TEE_OpenPersistentObject, TEE_CreatePersistentObject

Table 2.2: Functionalities of the REWIRE Key Management System

2.6.2 Key Hierarchy

As detailed throughout this and all technical deliverables, REWIRE offers a variety of security enablers used for the secure lifecycle management of devices. These are supported by a set of keys, which are supported by the Key Management Layer of REWIRE under a key hierarchy, which will be analyzed in detail in D4.3 [25]. Here, in Table 2.3, we provide a brief summary of the keys used in the context of REWIRE, as well as the security enablers where there are used.

Key	Description and Need
Root ID Key	An hardware-based asymmetric key in the underlying Root-of-Trust (RoT), embedded at manufacturing time. The Root ID Key (RK) is unique for every RoT acting as the device identifier. Thus, it cannot be changed, removed, or used outside of the device, and is considered as the identity key of the device.
Attestation Key	Attestation Keys (AKs) are asymmetric identity keys binded to the Root ID Key of the device, and must be certified by the Domain Manager (DM) before being used in the context of an attestation enabler. In case privacy-preserving capabilities need to be added to the attestation process, the AK can be elevated to a Direct Anonymous Attestation (DAA) Key.
Attribute-based Signing and Encryption Keys	These keys are used as part of the Attribute-Based SignCryption (ABSC) protocol. Specifically, for each attribute defined in an attribute-based policy that a device must adhere to in order to be onboarded or interact with the Blockchain infrastructure, each device should be able to ask the Domain Manager (or the appropriate Certification Authority) for the appropriate keys, which enable it to exhibit proof of ownership of the required attributes.
LRBC Key	A hardware-based key for the establishment of an encryption communication channel, over which software updates are securely transmitted.
Sealing Keys of the TEE	In general, in Keystone, sealing keys are derived for enabling storage of data in untrusted, non-volatile memory outside an enclave. Sealing keys are bound to the identity of the processor, the Security Monitor (SM), and the enclave, thus allowing the enclave to derive the same key after an enclave restart so that in can retrieve the data from the untrusted storage and decrypt it using the derived key. In the context of REWIRE, sealing keys are used for ensuring the integrity of enclavized applications during update and migration processes.
DH-based Symmetric Keys	When performing a live migration process from one device to a different one, symmetric Diffie-Hellman (DH) keys are generated in order to establish a secure communication channel between the devices.

BBS+ Keys	BBS+ based keys in the context of REWIRE are used to establish authentic and anonymous (if needed) channels between devices belonging either to the same domain, or to different domains.

Table 2.3: Keys used in REWIRE Key Management Layer

2.7 REWIRE Approach on HW-based extensions

An instruction set architecture (ISA) is an abstract model that generally defines how software controls the CPU in a computer or a family of computers. In other words, an ISA refers to the set of instructions that a computer processor can understand and execute, e.g. arithmetic operations or data movement. Sometimes, the ISA is referred to just as architecture. Historically, each CPU has been produced by a single company that controls the ISA that is implemented and the workings of the CPU. For example, Intel (x86) decides which instructions and features to implement, manufactures the processors and then sells them. Other companies such as ARM sell licenses and a basic design of their CPU (usually as an RTL file) that can be extended and modified according to the buyer's needs (e.g. Apple M1 and M2 processors). Each of them publishes a manual for each computer they sell, where they explain and describe the instructions carried out by the machine so programmers can make use of them. Once programmers build some Software for a certain ISA, such Software works on any CPU that implements that ISA, regardless of how the ISA is actually implemented.

RISC-V is a new approach based on open-source principles. Different working groups are involved in the writing and ratification of the specifications of the architecture [35, 36]. While all RISC-V specifications and software repositories are generally accessible, contributions may be constrained to members only. Anyone can implement a RISC-V CPU using a standardized instruction set without requiring any special licensing or royalty payments. As a result, RISC-V CPUs can be at a much lower cost than proprietary CPUs and have exploded in the low-cost microcontroller space and more powerful RISC-V CPUs are appearing, leading to Single Board Computers (SBCs) similar to the Raspberry Pi, along with inexpensive low-end laptops and tablets. Further, manufacturers or CPU designers are allowed to add customized instructions to their processors in order to build other functionality, and are free to implement some of the modules defined in the standard or not (on top of the base integer ISA, which must be present in any implementation). The modules that have been fully implemented are reflected in the processor codename.

Initially, a number of soft cores with different licensing models were listed on the RISC-V organization webpage, so software developers could just download them, synthesize them in an FPGA, and try out their software. At the time of writing, that section has disappeared, but some of the cores (along with other resources) are still accessible in [31]. Some of these are compatible with Keystone (some of the Keystone components have been adapted to the processors), and instructions on how to run Keystone on them is included in the Keystone repository. For example, Rewire initially used the CVA6 (a 6-stage, single-issue, in-order CPU that implements the 64-bit RISC-V instruction set.) as Keystone was recently ported so it could run on a CVA6 Platform on the Genesys 2 Board.

We emphasize that REWIRE follows the same philosophy as Keystone [14]. It builds Trusted Execution Environments for RISC-V purely on Software but relies on the Hardware components that are part of the Specification and are therefore found on any CPU that implements the ISA. In other words, the trusted extensions designed for REWIRE are agnostic to any other instructions the underlying ISA might be implementing (with the exception of the basic ones of the specification). We stay at the SM level (Machine Mode in Software), which has full control over the hardware, where the guarantees are for the trusted computing base. Figure 2.12 shows the REWIRE secure modules on top of a RISC-V core where they run (highlighted in grey). These represent the core functionalities of REWIRE as trust extensions, namely (i) the self-issuance and management of **Device Certificates** for binding the device identity to the underlying

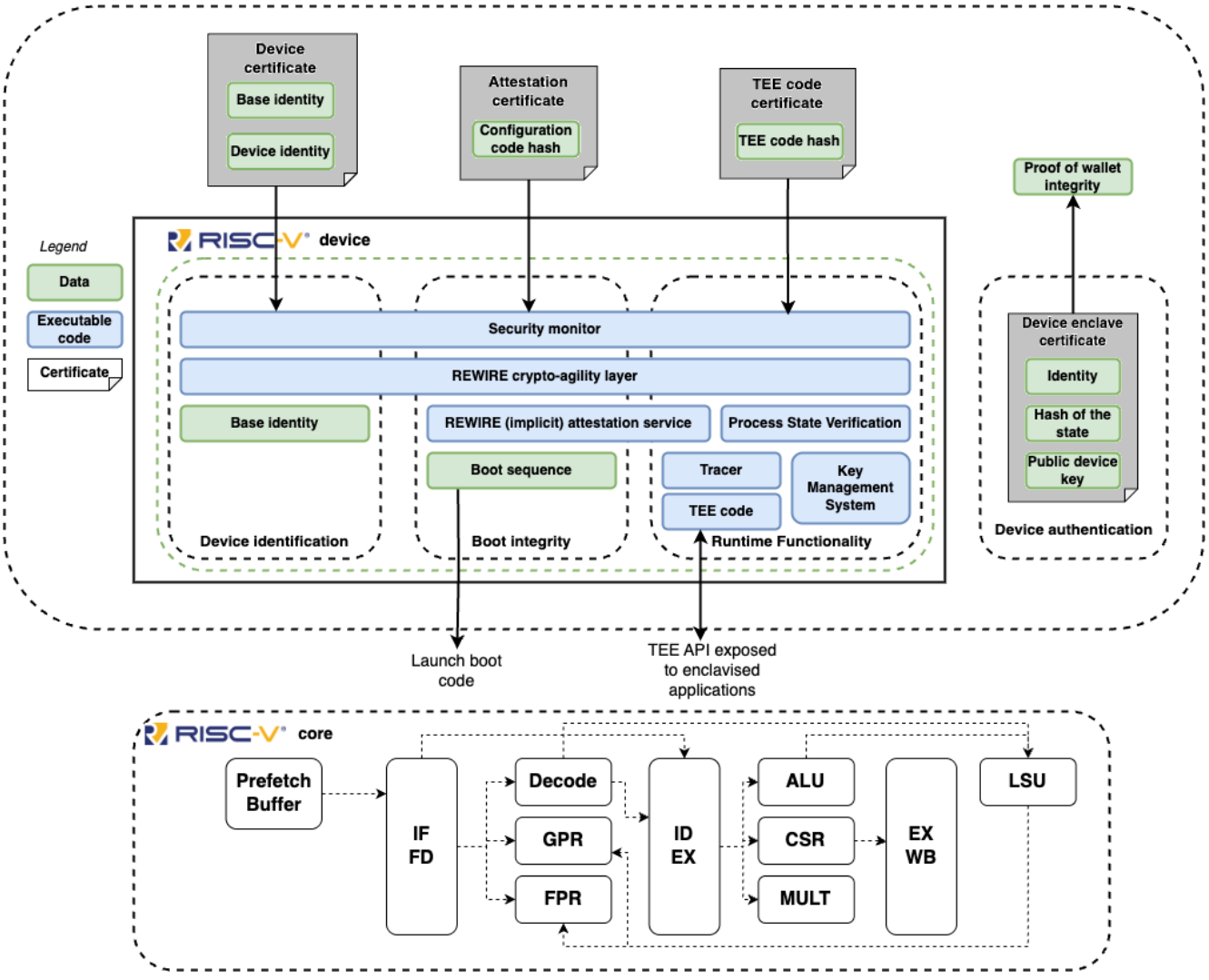

Figure 2.12: REWIRE Secure components altogether with a sample Risc-V core

HW protected by Keystone, (ii) the self-issuance and management of **Attestation Certificates**, not only to capture the verifiable launch of an enclave but also at a proecss level during runtime (see Process State Verification, Section 2.3.2.2), and (iii) the capability for management of **Verifiable Credentials (VCs)** for providing proof of integrity, ensuring that the credentials have not been tampered with.

Note that although the SM is depicted on top, some of the modules are either part of the SM (Key Management System) or accessible exclusively by the SM (Base Identity). One of the main goals of such a picture is to show that there are no direct dependencies between the Hardware components and the Rewire solution. The only assumption is that the RISC-V core executes the instructions as defined in the specification and implemented by that core. The device should get some identifiers (base identity) during manufacturing. These identifiers might be used to derive information for the enclaves, or for the attestation processes, also the SM may get its identifying information derived from them. Finally, the SM includes both data and executable code. The OS, enclaves, and user-mode applications running in the machine are not shown in Figure 2.12, it shows the libraries onboarded to support the creation and management of enclaves.

Consequently, REWIRE is not planning to introduce any modification to the Hardware neither to design Hardware specifically for the RISC-V ISA, as no modifications to the hardware layer are required to support the creation and execution of the enclaves. Adding some other instructions to the ISA is generally useful to provide additional features or to speed up some operations. We note that such an approach will limit the amount of devices where Rewire can run. Nevertheless, such instructions or crypto co-processors might be available, and in that scenario, Rewire can take advantage of them. To sum up,

Rewire will work on the software layer and extend the SBI [26] to support all the functionality that it is building, continuing with the extensions Keystone already added and following the same structure and approach.

An initial version of the Rewire Security Monitor was tested and evaluated on the CVA6 platform on the Genesys 2 Board (see deliverable 6.1 for further details [20]). However, we found several limitations for the different use cases and services we were building. Namely, there was a single core available, and therefore it was not possible to execute different tasks in parallel. We had issues with some of the peripherals as there were no drivers for them, the working frequency of the board was in the order of Megahertz, and the Linux OS we could run on it was quite limited.

We found an alternative to showcase our system in the StarFive VisionFive 2 board. As aforementioned, Rewire **does not require a specific Hardware to provide security guarantees to the enclaves**, therefore it was possible to run the Rewire SM on the VisionFive board. The VisionFive is designed to run Linux, it is based on the RISC-V architecture featuring a StarFive JH-7110 with a RISC-V quad-core CPU with 2 MB L2 cache and a monitor core, supporting RV64GC ISA, working up to 1.5 GHz, and providing the system memory of 2 GB, 4 GB, or 8 GB LPDDR4 SDRAM up to 2,800 Mbps. It is possible to think of this board as a Raspberry Pi offering enough peripherals and capabilities for the Rewire use cases.

Chapter 3

REWIRE Framework Requirement updates

In the context of this deliverable, we have ratified and validated the need for all requirements defined in D2.1, and evaluated their correspondence to the practical needs of organizations and environments targeted by REWIRE. This exercise also gives us the capability to check the security capabilities of REWIRE with regards to their capability to fulfil the aforementioned requirements. Thus, in the tables provided throughout this Chapter, we provide an **updated and refined version of the set of framework requirements that need to be fulfilled by REWIRE**, as well as the description of the **Key Performance Indicators (KPIs)** that will be used for evaluating the fulfilment of each requirement in the context of D6.2, as well as the target values for each of the KPIs.

	FR.FR.1 (Mandatory)	
Title	**Dynamic awareness of potential vulnerabilities and threats and complete overview of the deployed environment**	
Actors Involved	Security Administrator, Risk Assessment, AIMDE, SW/FW Validation Component, Attestation Agent, Tracer, Blockchain Infrastructure.	
Description	**Background**: The REWIRE framework should be able to collect and report information regarding vulnerable or being under attack assets. To assess the risks and the security posture of a highly distributed environment with several interconnected heterogeneous devices, there is a need to identify risks so that to proceed with the enforcement of new operational and security policies for regulating the operation of the environment or even to proceed to updates on the HW and SW co-design on systems (following the design-time phase of REWIRE). **Description**: REWIRE needs to provide the administrator with the necessary tools to perform dynamic and (semi-)automated risk assessment. To do so, visualization technologies that can offer a graphical representation of the monitored environment and ease the administrator to keep track of the existing interconnections among the numerous and heterogenous devices is necessary. When new vulnerabilities or threats are detected by the REWIRE artifacts (e.g., AIMDE) or reported by the community, this graphical representation will assist the administrator to evaluate how the interdependencies of the assets could trigger cascading effects and increase the cyber risk for critical services. REWIRE needs to provide solutions which are aligned with the good practices and standards to enable the dynamic risk assessment of IoT environments and safety critical applications. That is, based on evidence, the risk assessment tool will perform an informed analysis over the IoT deployment being monitored and will inform the system administrator about its security posture. This functional requirement could enhance the runtime monitoring capabilities of REWIRE and assist the administrator to take informed decisions for mitigation actions or attestation policies. **Remarks**: • Risk assessment needs to provide an output and highlight the identified risks and feed the HW/SW co-design formal verification processes. • The term dynamicity refers to the ability of the framework to acquire and process events from a monitored topology in near real-time manner.	
Connected to other requirements	• **FR.FR.4: Application and security data event sharing:** The risk assessment tool needs be aware on identified misbehaviour incidents.	
KPIs	**Description**	**Value**
	Elapsed time to complete a risk assessment execution for different specific scenarios.	- time for a complete risk assessment execution based on the identification of new threats $<=$ 2sec

	Elapsed time to collect all the necessary evidence including attestation and misbehavior-related evidence.	- time to collect evidence $<=$ 1 sec

Table 3.1: FR.FR.1 Dynamic awareness of potential vulnerabilities and threats and complete overview of the deployed environment

	FR.FR.2 (Mandatory)
Title	**Secure remote asset management and reconfiguration effectiveness**
Actors Involved	Security Administrator, Blockchain Infrastructure, SW/FW Validation Component, SW/FW Distribution Service, Secure Oracles, SW/FW update (on the device), Attestation Agent, SC-Resistant AE
Description	**Background**: The capability to remotely update IoT devices is a critical aspect of ensuring the long-term security of these assets. Devices should be able to receive SW/FW updates and configuration data in a secure manner, utilising crypto protocols resistant against side-channel attacks. Network administrators require a human-friendly summary whenever a device finished successfully and securely the SW/FW update. Thus, continuous monitoring and patch management to increase cyber resilience in a secure way is of paramount importance in REWIRE. End-devices in Smart Cities scenarios are deployed in vast geographical areas and operate without human supervision. They even lack keypads or displays. Thus, to ensure the proper operation of the network, remote management and reconfiguration are essential. Not only these must be implemented, but also, they require a high level of security since these devices might manage critical assets — fire detection and suppression systems in public areas, access control to public premises, waste management, plague control, street lightning, traffic control. Currently, most automotive SW updates are performed in an authorized dealership for SW updates regarding both new functionalities and bug fixes. At the same time, new automotive technologies in the field of autonomous driving and V2X connectivity require constant SW updates throughout the product lifecycle as well as day-to-day operation. Therefore, it emerges as a necessity to remotely perform SW updates (OTA) and this needs to happen in a secure manner as it can affect road and traffic safety. The same applies also for the Spacecraft applications and services. In fact, updating the SW/FW of a satellite which is in orbit around Earth is a rather challenging task, as the update procedure takes place in segments, utilising multiple ground stations communicating with satellites for only limited time frames. Secure and reliable SW/FW update processes and patch management are crucial for the domain. **Description**: REWIRE must allow the Security Administrator (e.g., network owner, manufacturer / supplier) to perform remote distribution of SW/FW update supported by a Blockchain Infrastructure for auditing purposes — i.e., selecting a FW binary or SW configuration files and the target devices from a list that must receive this information in a timely manner. Also, SW/FW data must be validated before the deployment. There is a risk of attackers triggering rogue SW/FW updates — e.g., malware or tampered SW/FW or even legitimate developers may unintentionally create vulnerable binaries. REWIRE will ensure security during SW/FW remote transmission and validation upon reception capitalizing on openly standardized protocols and data formats to guarantee the seamless integration of several types of devices within the domain. In Smart City scenarios, due to their heterogeneous nature, target devices will implement resource constrained MCUs that encourage employing lightweight security and communication protocols and mechanisms. In contrast, for vehicular scenarios, due to the typically high processing power of ADAS ECUs, more computational resources will be available, and thus, stronger security protocols can be considered. This can significantly improve the secure transmission of SW packages to the targeted vehicle(s), preserving security and integrity of the SW update package. A generalized SW update or bug fix can be globally deployed on control units of all cars of a given model, simplifying the update procedure, and reducing the attack surface. Therefore, REWIRE can provide the OEM / Authorized supplier with the ability to select a specific SW/FW package and automatically roll it out to specific vehicles or the whole OEM fleet (in case of a general update). Hence, remote SW/FW update will facilitate proper operation of autonomous driving and V2X connectivity (e.g., ensure that the vehicle is always up to date regarding HD road maps). In the context of smart satellites, secure SW/SW update processes will be developed to ensure that not only the update can take place utilising multiple ground stations, but also the update operation will be fail-safe and ensure that, even in the case of an update fault, the satellite will remain operational through a rollback mechanism. This SW/FW update must be transferred privately and without tampers. In addition, in the case of an already successful attack, a SW/FW update will offer a chance to regain full or partial control of the already compromised, safety relevant or not, components of the vehicle's system. Moreover, OTA updates can immediately deploy SW enhancements or new functionalities (e.g., improved perception algorithms with edge-case training), allowing continuous improvement throughout the product lifecycle. However, due to the nature of massive deployments, this operation should also enable the concurrent transmission of the SW/FW configuration data. Since this is not scalable in terms of time, computational resources or radio bandwidth, multi-cast secure communication technologies should be available instead. Finally, after a successful or failed SW/SW update procedure takes place, this should be reported back, in a confidential way, to the administrator through the secure remote asset management tools.

	Remarks: • The tools must receive SW/SW configuration data package from an authorized user as input, which will be in turn transmitted as an output through the infrastructure to the end devices. • The SW/FW package should meet the REWIRE framework security standards. • Despite the "one-to-one" nature of automotive SW/FW updates that will be considered on REWIRE, one-to-many case can optionally also be taken into consideration, for the case of Smart Cities scenarios or updating entire (or large groups of) vehicle OEM fleets. Therefore, both scenarios can fall under security clearance. • Devices with heterogeneous computational or bandwidth requirements must be supported — openly standardized protocols and data formats. • A report of the success or failure of the procedure must be presented to the administrator, in a confidential way, through the remote asset management tools. • User should be informed, in a confidential way, whether the SW/FW update has been successfully deployed. • SW/FW update will be triggered by the administrator based on the risk assessment output.
Connected to other requirements	• **FR.FR.1: Dynamic awareness of potential vulnerabilities and threats and complete overview of the deployed environment:** The Risk Assessment needs to consider potential identified vulnerabilities and threats. • **FR.FR.5: SW/FW unpacking and vulnerability analysis**: The SW/FW binaries need to be validated and be sanitised from known vulnerabilities before the SW/FW update process takes place. • **FR.FR.7: Zero Touch device Onboarding**: The secure device onboarding is a perquisite to achieve secure remote asset management and reconfiguration.
KPIs	**Description** / **Value** Elapsed time for an end-to-end SW/FW update in one-to-one mode. This includes establishment of an authenticated encryption channel with LRBC and the execution of the attestation inside the TEE. The SW/FW update should be linear to the size of the update. — - end-to-end time for SW/FW update (one-to-one) $<=$ 2 mins Elapsed time for an end-to-end SW/FW update in one-to-many mode. This includes the time to get the update from the BC and the necessary operation (e.g., ABAC etc.) — - end-to-end time for SW/FW update (one-to-many) $<=$ 3 mins

Table 3.2: FR.FR.2 Secure remote asset management and reconfiguration effectiveness

	FR.FR.3 (Mandatory)
Title	**Device status auditing**
Actors Involved	Blockchain Infrastructure, Secure Oracles, Off-chain Data Storage, Attestation agent
Description	**Background**: The need for data auditing is not new, while there is an exhaustive literature with works for blockchain-based data auditing for different domains and applications. Regular audits can foster the identification of potential security risks and device status auditing can assist to maintain a high level of security. Thus, the continuous and automated auditing, including the device statuses, in a secure and trusted way to support the demanding business needs is a necessary REWIRE feature. **Description**: Blockchain technology is necessary to maintain the collected evidence, while smart oracles can be leveraged to support data collection (e.g., device statuses). Prior to any action, there is a need for established authentication between the device and the oracle. The smart oracles can filter, format, and perform a veracity check on the submitted data before storing them on-chain. On top of that, oracles are necessary to be TEE-enabled in order to overcome the data veracity issue and assure that the audit mechanism is trustworthy itself. Thus, all the stored information can be used as traceable and non-tampered evidence.

	Device status auditing is an essential aspect of the REWIRE project as it enables continuous monitoring of the security status of IoT devices. This function helps in identifying potential vulnerabilities in the system, thus reducing the risk of cyber-attacks. More specifically, the ”health” status of a device or critical system should be auditable (also for certification purposes). Thus, results from the attestation of devices or from the SW/FW validation process should be available to other actors through the blockchain infrastructure. Collected data should reflect the trustworthiness of the systems. For example, when setting up communication with another device or a device wants to enter the overall infrastructure network (secure device onboarding), the status of that device before proceeding should be in a trusted state. By regularly auditing the device status, the project can ensure that the hardware and software components are functioning as intended, and no malicious activity is occurring. This can also result in a form of certification, demonstrating compliance with industry standards and regulations. **Remarks**: • The decentralised smart oracles enable data collection through smart contracts in a secure and trustworthy manner so that the security audit itself is trustworthy. • The oracles will be supported and embedded with a TEE so that it can run secure operations within a trusted and safe environment.
Connected to other requirements	• **FR.FR.4: Application and security data event sharing** : Auditing data are transmitted through the oracles in a secure way. • **FR.FR.7: Zero Touch device Onboarding**: A device status auditing mechanism mandates secure device onboarding. • **FR.FR.11: Trust Aware Continuous Authentication and Authorisation**: A secure and trusted device status auditing mechanism mandates a continuous authentication and authorisation of the device. • **FR.FR.13: Data veracity management**: A core research challenge that needs to be addressed is the veracity of the collected data since it is necessary for valid auditing. • **FR.FR.16: Device provenance and device status**: Device provenance and status is necessary for the corresponding auditing.
KPIs	**Description** / **Value** Elapsed time to create the conformity certificate inside the TCB for all the enclaves. The auditing should be linear to the number of the created certificates (one per device or per enclave). — - time to create the conformity certificate <= 10sec Elapsed time to write a Certificate (e.g., VC) for the device status per transaction (for both non TEE-enabled and TEE-enabled peers). This includes the time for IDM & access control. — - time to write a VC for the device status <= 3sec Elapsed time to read a Certificate for the device status per transaction (for both non TEE-enabled and TEE-enabled peers). This includes the time for IDM & access control. — - time to read a VC for the device status <= 1sec

Table 3.3: FR.FR.3 Device status auditing

	FR.FR.4 (Mandatory)
Title	**Application and security data event sharing**
Actors Involved	Blockchain Infrastructure, Secure Oracles, Off-chain Data Storage, AIMDE
Description	**Background**: Sharing application and security events enables quick and crucial decision making about the security level and potential countermeasures against cyberattacks. However, the current event sharing solutions do not allow easy communication and knowledge sharing among detection systems exploiting AI-assisted detection techniques. This type of data is highly sensitive and must be protected. Thus, both confidentiality and integrity are mandatory requirements. However, due to the dynamic nature and the amount of event data a centralized way is not an option, while decentralised solutions (e.g., Blockchain) are not always scalable.

Description	**Description**: REWIRE framework should be able to identify misbehaviour of devices and report threat intelligence data generated, in the form of events. Data generated in the context of threat Intelligence actions (e.g., attestation, SW/FW validation, AI-based misbehaviour detection) shall be made available to the administrator of the monitored environment, (acting as the security service operator), to be able to dynamically assess its security posture and take mitigation actions. In the context of REWIRE, these application and security event data are stored on a Blockchain platform for security, transparency, immutability, and decentralization qualities. However, application and security data may be of high volume and/or velocity and stem from different layers of a system architecture (application, network, etc.). That is, a Blockchain infrastructure needs to operate in tandem with external (off-chain) data storage solutions to maintain bulky data and have a synchronised operation for data indexing with the Blockchain per se. On top of that, this functional requirement could enhance the runtime monitoring capabilities of REWIRE by a) assisting the administrator to take informed decisions on misbehaving devices and act, b) sharing of threat Intelligence data within REWIRE distributed environment will allow other entities (e.g., devices) operating in the same use case to understand the security status of other devices they need to interact with. **Remarks**: • The AIMDE needs to provide an output that will enable the system administrator to promptly identify threats/attacks and utilise the results for risk assessment and further mitigation/corrective actions. • Blockchain provides secure, transparent, immutable, and decentralised storage for threat intelligence data; nevertheless, due to high data volume and velocity, a combination of blockchain and off-chain storage solutions is required for efficient data management and synchronisation.
Connected to other requirements	• **FR.FR.1: Dynamic awareness of potential vulnerabilities and threats and complete overview of the deployed environment**: Application and security data events needs be considered on the risk assessment methodology of REWIRE.
KPIs	**Description** / **Value** Average elapsed time for training for point anomalies. — - average training time for point anomalies $<$ 5min Average elapsed time for training for contextual anomalies. — - average training time for contextual anomalies $<$ 1h Average elapsed time for training for south atlantic anomalies. — - average training time for south atlantic anomalies $<$ 10min Elapsed time to apply. — - time to to apply $<$ 1 min

Table 3.4: FR.FR.4 Application and security data event sharing

	FR.FR.5 (Mandatory)
Title	**SW/FW unpacking and vulnerability analysis**
Actors Involved	SW/FW Validation Component
Description	**Background**: SW//FW verification assesses whether a product is built correctly. Thus, a FW/FW shall be validated before the deployment on a device, either during the design phase or during the runtime phase of the framework and before the over the air SW/FW update. Detecting SW/FW vulnerabilities before the deployment avoids unnecessary risks during runtime and is of paramount importance in REWIRE. **Description**: In the context of REWIRE, the validation is part of an end-to-end chain covering device integrity and trust. Guided fuzzing and symbolic execution techniques are combined to verify whether SW/FW contains any vulnerabilities. In REWIRE, using a combination of static symbolic execution and fuzzing this analysis aims to find crashing inputs for the targeted FW/SW. The SW/FW can be emulated on a system dedicated to the software analysis, in order to easily scale it to many different devices and firmware versions. Through the use of root cause analysis, the information on these crashing inputs can be used to determine the existence of bugs in the firmware. When feasible, these bugs will be analysed further to determine whether they constitute a vulnerability. The information on identified vulnerabilities can then be communicated to other components in the REWIRE project, such as the risk assessment tool for increasing the awareness of the security operator, or, in the context of the Design-time phase of the REWIRE framework, this information can be given as input to the formal verification tools in order to analyse the overall security of the HW/SW co-design. **Remarks**: • The SW/FW package must be accessible by a third party, i.e., it cannot be encrypted or packaged using a proprietary format that is not publicly known. • The firmware has to be based on a Unix-like operating system and compiled for a RISC-V architecture. • The analysis combines dynamic and static analysis techniques to combine their strengths and compensate for their weaknesses

Connected to other requirements	• **FR.FR.1: Dynamic awareness of potential vulnerabilities and threats and complete overview of the deployed environment**: The Risk Assessment needs to take as an input the vulnerability analysis performed during SW/FW unpacking. • **FR.FR.2: Secure remote asset management and reconfiguration effectiveness**: FR.FR.2 refers to the need for over the air updates. Thus, before any deployment of a new FW/SW, the SW/FW unpacking, and vulnerability analysis needs to take place to guarantee that the SW/FW is vulnerability-free.
KPIs	**Description** / **Value** Detection accuracy of SW/FW Validation Component. / detection accuracy $>= 80\%$ Elapsed detection time of SW/FW Validation component. / detection time $<$ 1sec

Table 3.5: FR.FR.5 SW/FW unpacking and vulnerability analysis

	FR.FR.6 (Mandatory)
Title	**Secure Measurement and Attribute Extraction**
Actors Involved	Tracer, SW/FW Validation Component (adding specific hooks for the introspection of applications), Attestation Agent
Description	**Background**: The increasing threat surface of IoT devices mandates to consider the underline security risks and to ensure that those devices are secure and trusted. REWIRE mechanisms should be able to identify critical states of a systems' operation, (e.g., specific signals in ECUs) in order to check for the correct operational behaviour of devices. Traditional static analysis techniques cannot capture the dynamic threats manipulating dynamically the execution flow of a code base. Thus, more sophisticated, and dynamic techniques are need, while legacy dynamic tracing techniques are not scalable due to the state explosion problem. Remote Attestation (RA) techniques are an essential part of trusted computing. Current RA techniques should provide proofs of static and dynamic system properties to offer holistic security, through verification of system's integrity. These dynamic properties are used as evidence to verify the system integrity during runtime. To do so, runtime monitoring of the system data and its execution flows enables the collection of useful information about the behaviour of the system. Thus, potential anomalies or deviations from intended behaviour can be identified. However, dynamic attestation has some challenges from the dynamic nature of the attributes itself making it difficult to identify and deduce their good state to scalability and efficiency issues. **Description**: REWIRE requires a non-intrusive runtime validation process of the runtime data and execution stream. The dynamically collected execution traces on the devices (e.g., REWIRE evidence collectors), upon a runtime attestation fail (i.e., attack indication), will be further analysed dynamically to emulate the problematic execution path and potentially uncover a previously unseen malicious execution behaviour. The REWIRE Tracer should be able to extract the system state to be validated. Apart from binary and configuration files signatures, the tracer needs to be able to perform non-intrusive memory introspection to enable control flow attestation during run-time. The addition of monitoring hooks will be performed using the SW/FW validation component. SW/FW can be unpacked, resulting in at least a file system. The software contained in this file system can be disassembled and lifted to an Intermediate Language, on which it is more feasible to reason about the software and adjust it. In this representation, monitoring hooks will be added, after which it can be compiled back to its original architecture. The software in the unpacked firmware will be overwritten using this modified version, after it will be repacked. This modified version of the firmware can then be installed on a device, resulting in live monitoring of its behaviour. Note that installing modified SW/FW is not possible on a device where creating legitimate firmware is intentionally made more challenging, such as with proprietary packaging formats, encrypted firmware, or signed firmware. After the deployment of a SW/FW, the injected monitoring hooks will enable the REWIRE tracer to identify efficiently critical states of a systems' operation, in order to check for the correct operational behaviour of devices in the context of the REWIRE attestation schemes. **Remarks**: • The firmware package must be accessible by a third party, i.e., it cannot be encrypted or packaged using a proprietary format that is not publicly known. • Monitoring hooks are applied statically, meaning the firmware image itself is changed. • REWIRE, in the case where the tracer is not positioned in the trusted world (e.g., enclave), will assure that the tracer is in a secure state and the captured traces are valid and not tampered with (i.e., Integrity of Trust Measurements). • REWIRE tracing is aimed to be efficient since only a fraction of the execution needs to be monitored, due the minimisation of the stack that needs to be introspected as a result of the formal verification of the SW/HW co-design, and as a result of the use of monitoring hooks.
Connected to other requirements	• **FR.FR.3: Device status auditing**: REWIRE mechanisms should be able to identify critical devices' statuses for checking for the correct operational behaviour of devices. • **FR.FR.16: Operational assurance and configuration integrity**: The secure measurement and attribute extraction is the baseline to provide operational assurance through attestation.

KPIs	Description	Value
	Overhead to the executable that is being traced.	overhead <=25%

Table 3.6: FR.FR.6 Secure Measurement and Attribute Extraction

	FR.FR.7 (Mandatory)
Title	**Zero Touch device Onboarding**
Actors Involved	Keystone TEE, Blockchain Infrastructure, Secure Oracles, MUD Profile Server, Privacy CA, Domain Manager, Attestation Agent
Description	**Background**: Faced with the rapid increase in IoT inter-connected devices and the high demand for new services, the management of such devices is getting complex. These devices are subject to an expanding list of attacks that exploit both software vulnerabilities and design choices, highlighting the importance of management cryptographic keys. In addition, traditional onboarding approaches cannot keep up with rapidly evolving application requirements. The limitations of traditional onboarding models, such as security and time-consuming manual steps, especially in the IoT domain led towards developing zero-touch onboarding mechanisms (also known as zero touch provisioning). **Description**: REWIRE needs an efficient and scalable zero-touch onboarding solution to configure the IoT devices onto a network as well as a minimized attack surface. Also, an access control mechanism is needed to assist in the onboarding and managing of devices into the network ensuring secure (authenticated) enrolment and consistent standardisation. More specifically, in REWIRE an Attribute-based Access Control (ABAC) along with the trust-related attributes in the form of VPs is considered a natural fit to adopt. The cryptographic key used to generate the VP is held in the TEE (utilising the REWIRE key management mechanisms), with key usage policies restricting its use, resulting in control over whether the device can get access to the network, given that it exhibits specific attributes advocating its operational correctness. Also, a proposed solution that will be investigated further in REWIRE is the authentication protocol using DAA (instead of "group" keys to preserve the privacy of authenticators). Recall, DAA is a cryptographic scheme used to ensure security and privacy guarantees and it has been adopted by TGC TPM2.0 for Elliptic Curve based DAA (ECDAA). REWIRE will focus on DAA with attributes (DAA-A) protocol, in which the platform can select which attributes (from VPs) are shown/hidden to the verifier. The authenticity of the hidden attributes will be proved by a zero-knowledge protocol and usefully DAA-A provides user-controlled linkability. **Remarks**: • Currently, the DAA protocol is not included in the W3C description of VCs • The two most prominent EC based DAA protocols for TPM 2.0 can be extended to DAA-A protocols.
Connected to other requirements	• **FR.FR.8: Dynamic Credential Management** : Dynamic credential management is necessary for the authorisation during onboarding. • **FR.FR.10: Flexible and reliable key management**: Key management is necessary for providing the necessary crypto primitives to perform the secure onboarding solution and the management of the cryptographic keys. • **FR.FR.15: Common Trusted Computing Base**: A common TCB is necessary to be adopted in order to fulfil the correct device state and hence the secure onboarding. • **FR.FR.16: Device provenance and device status**: Zero-touch onboarding is tightly coupled with the device provenance and status. • **FR.FR.18: Policy-based device state configuration**: During onboarding, policies regarding the device state are necessary to fulfil the security requirement. • **FR.FR.21: Provably secure crypto protocols and algorithms**: All exchanges of sensitive data such as access control and DAA solutions should be protected through cryptographic means.
KPIs	**Description**: Elapsed time for a complete secure device onboarding, including the issuance of VCs through ABS. — **Value**: - time for a complete secure device onboarding < 10 sec	

Table 3.7: FR.FR.7 Zero Touch device Onboarding

	FR.FR.8 (Mandatory)
Title	**Dynamic Credential Management**

Actors Involved	TEE, Attestation Agent, Blockchain Infrastructure.
Description	**Background**: Most existing credential management solutions require a centralized authority, either for attribute registration or credential verification. On the contrary, SSI enables entities to have ownership and control of their unique identity data without involving any third party. Currently, several works are exploring how to leverage blockchain to build SSI solutions, however, there is a lack of systematic architecture design for blockchain-based SSI systems. On top of that, current solutions are considered coarse-grained and may underperform or lack on data security (e.g., inadequate access control for credentials). In addition, highly distributed environments mandate the continuous authorisation and authentication of devices as well as secure device onboarding. Apart from the certificates used to verify the identity, certificates are also employed to verify the data provenance; a fact that adds further complexity to the system. This highlights the need in REWIRE of credential management (including revoking credentials in case of misbehaviors) in a dynamic way. **Description**: REWIRE will generate credentials for ensuring efficient, trustworthy, and secure communication between entities. Trust claims are made using VPs, while a selective disclosure of attributes will be used in order to minimise the information which needs to be exposed by the attributes to the verifying entities. More specifically, VPs are comprised of self-issued verifiable credentials and the key used to generate the VP is stored in the TC wallet. In the context of REWIRE, dynamic credentials are necessary as they can be generated on-demand, they are unique to a client (instead of static credentials which are pre-defined and shared) and they can change over time - a credential is associated with some policy encompassing the expiration date of when the credential needs destroying. Crucial to security, the dynamic functionality also mitigates security risks if a credential is leaked. In this case, the credential is revoked and remediated. To do so, REWIRE necessitates dynamic attestation which is reflected in the dynamic nature of credentials. Last but not least, blockchain technology will be also used to facilitate sharing and auditing of verifiable credentials between entities that want to establish a trust relationship in a zero-trust manner. **Remarks**: • VPs issued by parties have the following advantages: interoperability (DIDs are “chain-agnostic” so they’re not permanently bound to a single BC); privacy (communicating parties can issue verifiable statements with the selective disclosure of attributes (DAA-A)); scalability (maintaining a subset of credentials off-chain can reduce the costs). • Credentials used for static attestation would only demonstrate the state of a device at the time of executable code, therefore, it would not provide a view of program behaviour at run-time.
Connected to other requirements	• **FR.FR.7: Zero Touch device Onboarding**: Dynamic credential management is necessary to achieve zero touch device onboarding. • **FR.FR.10: Flexible and reliable key management**: Flexible and reliable key management • **FR.FR.11: Trust Aware Continuous Authentication and Authorisation**: The dynamic credential management is the baseline for device authentication and authorisation. • **FR.FR.14: Confidentiality and integrity in data processing**: Dynamic credential management is of paramount importance to ensure the data integrity and confidentiality during processing (e.g., by utilising ABE). • **FR.FR.15: Common Trusted Computing Base**: A common TCB is the baseline as the trust anchor for credential management. • **FR.FR.16: Device provenance and device status**: evice status corresponds to the overall device trustworthiness and is reflected in the device credentials (e.g., attributes). • **FR.FR.18: Policy-based device state configuration**: Policies are based on the credentials of the device (e.g., attributes). • **FR.FR.21: Provably secure crypto protocols and algorithms**: Dynamic credential management is necessary to ensure the security of crypto protocols and algorithms.
KPIs	**Description** / **Value** Elapsed time for certificate creation per device. We consider linear of number of attributes inside the credentials. / - time for certificate creation per device approx. $<=$ 3 sec Elapsed time for certificate creation per enclave. We consider linear of number of attributes inside the credentials. / - time for certificate creation per enclave approx. $<=$ 3 sec

Table 3.8: FR.FR.8 Dynamic Credential Management

	FR.FR.9 (Mandatory)
Title	**Establishment of Secure and Authenticated Communication Channels**

Actors Involved	SW/FW Validation Component
Description	**Background**: The establishment of secure and authenticated channel is imperative in REWIRE with the huge amount of heterogeneous interconnected devices. It is vital to protect the critical process of secure SW/FW update and protect this process by deploying secure and authenticated communication channels. **Description**: REWIRE will consider the critical character of SW updates, as they represent the first trust level of the system, and the fact that REWIRE devices often operate in unmonitored environment, the channel should be secure even in the presence of physical attackers capable of performing side-channel attacks. Moreover, being able to achieve this security level based on off-the-shelf components and without the need for the implementer to have a deep expertise in the field of side-channel countermeasures would allow swift and easy development of REWIRE devices. Towards this direction, REWIRE needs to design an Authenticated Encryption (AE) algorithm. This algorithm ensures both the confidentiality and the authenticity of exchanges between two actors sharing a symmetric key, meaning that attackers cannot obtain any information on the transmitted payload, nor modify this payload in any way without being detected. This algorithm should also meet a state-of-the-art definition of security in adversarial physical conditions, such as CIML, precisely defining which security properties hold even if the attacker can monitor side-channel behaviour, tamper with nonces. The algorithm will be implemented as a leakage-resilient mode of operation based on a standard block cipher such as the AES. In as much as possible, leakage resilience will be enforced by the mode of operation itself, without relying on side-channel countermeasures in the implementation of the underlying block cipher. **Remarks**: • The shared symmetric key used for SW updates must be a long-term key that can of course not be exchanged nor derived using methods that are not protected against side-channels, or these methods must be applied before deployment, in a secure environment. • The resulting cryptographic algorithm will consist in: (a) A SW implementation of the mode of operation, relying on a block cipher implemented in HW. (b) A characterization of the minimal expected properties of the HW block cipher, together with an example of off-the-shelf device meeting these requirements.
Connected to other requirements	• **FR.FR.2: Secure remote asset management and reconfiguration effectiveness**: The leakage-resilient mode of operation of the AES will support the SW/FW update processes of REWIRE • **FR.FR.5: SW/FW unpacking and vulnerability analysis**: The SW/FW unpacking, and vulnerability analysis should be performed in a secure and authenticated channel.
KPIs	**Description**: Time elapsed for a SW-based establishment of a secure and authenticated channel including the TEE overhead. **Value**: - time for a secure and authenticated channel $<=$ 700ms

Table 3.9: FR.FR.9 Establishment of Secure and Authenticated Communication Channels

	FR.FR.10 (Mandatory)
Title	**Keystone TEE, Security-by-design Monitors, Attestation Agent**
Actors Involved	Flexible and reliable key management
Description	**Background**: The key management process includes the secure generation, distribution, operation, storage and deletion of cryptographic keys, while the huge number of heterogeneous IoT devices, implies complex key management for different involved stakeholders, types of devices, and key lifecycles. Encrypting data to protect the confidentiality of the underlying information is a necessity. When it comes specifically to the SW update key, once the corresponding encryption (symmetric) key has been generated and used for its purpose, it must be stored for later use when the corresponding ciphertext needs to be decrypted. Storage of cryptographic keys is challenging, and solutions based on key hierarchies are necessary. **Description**: Intuitively, this design in the REWIRE architecture will organise cryptographic keys so that a root (master) key encrypts so-called leaf (children) keys, with the leaf keys encrypting the data in question. One difficulty is to ensure that the master key is simultaneously secure and accessible to decrypt leaf keys. Benefits to REWIRE of utilising a key hierarchy include minimising the amount of plaintext key material requiring protection and storage – plus, storing a master key alone restricts access to just one key. Moreover, key hierarchies enable segmentation of encrypted data. That is, data can be encrypted using independent cryptographic keys which minimises the impact of potential key corruption/loss. Furthermore, the REWIRE KMS manages keys in the Security Monitor since critical services, like key derivation, must run in isolation from other SW running on the same platform. The TEE must manage keys for remote attestation, sealing, migration etc. **Remarks**: • Key hierarchies can be comprised of several layers of keys, such that a parent key provides protection to their corresponding leaf keys, all the up to the master root key which encrypts all keys irrespective of what layer they belong to.

Connected to other requirements	• **FR.FR.7: Zero Touch device Onboarding**: Key management is necessary to achieve zero touch device onboarding. • **FR.FR.8: Dynamic Credential Management**: Key management is the baseline for providing the necessary crypto primitives (e.g., digital signatures) for dynamic credential management. • **FR.FR.15: Common Trusted Computing Base**: A common TCB is the baseline as the trust anchor for key management. • **FR.FR.18: Policy-based device state configuration**: Key-restriction policies binded to the device state are used allow or not the key usage. • **FR.FR.21: Provably secure crypto protocols and algorithms**: Proper key management is a prerequisite to ensure the security of crypto protocols and algorithms.	
KPIs	**Description**	**Value**
	Elapsed time for the execution of the key restriction usage policy check.	- key restriction usage policy check $<$ 100ms
	Different types of keys to be supported and maintained (i.e., identity key, integration key, authentication key, attestation key, etc.).	- types of keys $>$ 4 keys
	Elapsed time for accessing the keys from the enclave through the security monitor.	- time for accessing the keys $<=$ 500ms

Table 3.10: FR.FR.10 Flexible and reliable key management

	FR.FR.11 (Mandatory)
Title	**Trust Aware Continuous Authentication and Authorisation**
Actors Involved	Blockchain Infrastructure, Attestation Agent, TEE
Description	**Background**: Several well-established access control mechanisms exist in the literature. The adoption of an access control mechanism such as ABAC or RBAC for enabling authenticated and authorised resource and data accessing is necessary in the IoT domain. It must be noted also here the need for external, but authenticated and authorized, entities to retrieve data, mainly for certification purposes. The access control needs to consider the use of trust-related attributes (claims) as VPs. This requirement is needed in the REWIRE project to provide advanced security measures for the data on Blockchain Infrastructure, with the help of advanced technology enablers, such as wallet, smart contract, and TEE technologies. **Description**: In REWIRE it is necessary to ensure that the security measures implemented are not only robust but also efficient, as inefficient security measures may lead to practical difficulties and a lack of adoption. Thus, REWIRE needs to consider the use of crypto that can offer both security and efficiency. By incorporating such technology, the REWIRE project can ensure that sensitive data is protected without sacrificing usability and convenience. Besides, it's necessary to have access to reliable and trustworthy data for security assessment. This data is mainly related to the devices' statuses but also those hard copies of the data stored in an off-chain and cloud server backend. Recall that it may not be practical to store all data on-chain and it is necessary to intake updated statuses to the ledger. To ensure authenticity and integrity, a robust mechanism for authenticated off-chain data retrieval is essential. By doing so, the REWIRE project can access dependable data for security assessment. In addition, the above provided solutions will be designed in an efficient and cost-effective access control and TEE protection. This functional requirement enhances the capabilities of REWIRE by providing an advanced and efficient security mechanism that continuously monitors user behaviour and detects potential security risks in real-time. By adopting Trust Aware Continuous Authentication and Authorization, the REWIRE project can ensure that only authorized and authenticated resources can access the sensitive data. This technology also helps prevent unauthorized access and cyber-attacks, contributing to the overall security of the REWIRE project. Besides, the external data retrieval will be considered for the authentication so that the income and outcome data sources are trusted and verified. It has to be noted that the authentication and authorization mechanisms of REWIRE will not only be focusing on the devices belonging in the monitored ecosystem, but the same principles will be applied and need to be followed by external entities that need to access to the collected data (mainly for certification purposes). **Remarks**: • Flexibility related to who can generate these claims either by a trusted issuer or self-made attestation evidence is also important.

Connected to other requirements	• **FR.FR.3: Device status auditing**: Authentication and authorisation of the device is necessary for the device status auditing. • **FR.FR.13: Data veracity management**: Authentication and authorisation of the device is necessary for veracity of the collected data. • **FR.FR.15: Common Trusted Computing Base**: Authentication and authorisation of the device is based on the underline TCB.	
KPIs	**Description**	**Value**
	Number of crypto primitives that can be supported (crypto agility).	- number of crypto primitives $>= 3$
	On average end-to-end (including communication time between trusted and untrusted world) execution of crypto operations either signature or encryption/decryption.	- average communication time of crypto operations $<= 2$ sec
	Overhead introduced by the size of the credential	- linear to the number of attributes disclosed to the credential

Table 3.11: FR.FR.11 Trust Aware Continuous Authentication and Authorisation

	FR.FR.12 (Mandatory)	
Title	**Data provenance**	
Actors Involved	TEE	
Description	**Background**: The term data provenance also called data lineage, refers to a record trail that accounts for the origin of a piece of data together with an explanation of how and why it got to the present place. REWIRE devices must be able to establish and verify the origin of sensitive data they receive in a privacy aware manner, and potentially record this proof of origin together with data at rest, as they run in a fully decentralized environment. **Description**: Sensitive data exchanged in the framework of REWIRE must be protected by cryptographic means, to allow verifying its provenance and the fact that it has not been modified. Depending on the context, these means can consist in: • Asymmetric cryptographic primitives, such as digital signature. Digital signatures provide the advantage of being transferrable to a third party, meaning that it is possible for an entity to prove, in an undeniable way, that a piece of data was signed by another entity. Signatures can also be chained if multiple provenances (e.g., along a transmission path) must be established. However, digital signatures are more expensive than symmetric primitives, and signature generation is more difficult to protect against side-channel attacks (signature verification, on the other hand, does not suffer from this drawback, as it involves no secret parameter). • Symmetric cryptographic primitives, such as authenticated encryption. Symmetric primitives allow verifying that a piece of data originates from an entity knowing a specific secret key and has not been modified. However, this verification is not transferable to a third party (no distinction can be made between the data sender and receiver) and can typically not be chained. Less costly and easier to protect against side-channel attacks, they remain nonetheless useful in cases where such transfer is not required. **Remarks**: • Data provenance (ability to check the integrity of data and the identity of its emitter) must not be confused with verifiability that the data is harmless, which is addressed by misbehaviour detection.	
Connected to other requirements	• **FR.FR.10: Flexible and reliable key management**: Key management is the baseline for providing the necessary crypto primitives (e.g., digital signatures) for assuring the integrity and data provenance. • **FR.FR.15: Common Trusted Computing Base**: Common Trusted Computing Base	
KPIs	**Description**	**Value**
	Elapsed time for data provenance (execution of DAA without any key restriction usage policies)	- controlled linkability $<$ 2 sec

Table 3.12: FR.FR.12 Data provenance

	FR.FR.13 (Mandatory)
Title	**Data veracity management**

Actors Involved	Blockchain Infrastructure, Secure Oracle, Attestation Agent, Keystone TEE
Description	**Background**: Data veracity management is a critical aspect of ensuring the integrity of data within the IoT domain. By establishing a trustworthy and auditable data provenance mechanism, the system can ensure the veracity and traceability of vital data, thereby augmenting the security and operational assurance of the IoT ecosystem. A blockchain-based oracle can provide data veracity validation by verifying the accuracy and integrity of data before it is entered into the blockchain. On top of that, the execution of data transaction needs to take place through these secure oracles and with the use of smart contracts supported by root of trust. **Description**: To provide trusted and accurate data, the use of a blockchain-based oracle is essential. In the context of REWIRE, a secure oracle acts as a trusted intermediary between external data sources and the blockchain infrastructure. The secure oracle is responsible for verifying the authenticity and veracity of data before it is entered into the blockchain. The use of a blockchain oracle enhances the veracity of the data management process and helps to prevent the introduction of fraudulent or inaccurate data into the system. This functional requirement enhances the capabilities of REWIRE by providing a reliable method for ensuring the veracity of data within the system. By utilizing a TEE based blockchain oracle, the REWIRE project can ensure that only trusted and accurate data is utilized for security assessment and certification. This helps to improve the overall security posture of IoT devices and ensures that they meet the necessary cybersecurity standards. **Remarks**: • Secure oracles are enabled by the REWIRE customizable TEE to offer the necessary crypto primitives and ensure the secure management of data sharing.
Connected to other requirements	• **FR.FR.3: Device status auditing**: REWIRE mechanisms should be able to identify critical devices statuses for checking the correct operational behaviour of devices which also affects the data veracity. • **FR.FR.11: Trust Aware Continuous Authentication and Authorisation:**: Continuous authentication and authorisation of the device is a prerequisite for data veracity • **FR.FR.15: Common Trusted Computing Base**: Common Trusted Computing Base
KPIs	**Description**: Elpased time for data veracity checks to be executed by the Secure Oracle through the VCs. **Value**: - data veracity time $<$ 20 ms

Table 3.13: FR.FR.13 Data veracity management

	FR.FR.14 (Mandatory)
Title	**Confidentiality and integrity in data processing**
Actors Involved	TEE
Description	**Background**: REWIRE needs to ensure the data integrity and confidentiality of critical functions running on the IoT devices so that to guarantee the operational assurance of the device and the critical services offered by both the device and the REWIRE platform. Such critical functions can include key management, attestation or SW update validation. This must be performed in a way that an application built with trusted and untrusted parts is able to call one of such critical functions from the untrusted part, then through a high-privileged component transition data and execution to the trusted part and remain unable to access the trusted part. This guarantees confidentiality and integrity. On the other hand, the trusted part must be able to execute code and manipulate data and return results back to the untrusted world through the high-privileged component. It is also important to be able to seal and unseal data, while having a way, such as monotonic counters, which prevents rollback attacks and bad data updates. **Description**: The key component to achieve this is the REWIRE TEE (for which, namely, Keystone has been proposed). By means of the capabilities normally issued in Keystone, REWIRE must ensure the integrity and confidentiality of data passing through the untrusted world to the trusted world of the device. This requirement needs to be considered also on the level of confidentiality of data between different TEEs, between trusted portions of memory (i.e., enclaves), between the OS and the enclaves, between user applications and the enclaves, plus in terms of the necessity of being able to successfully migrate the execution (together with data) of critical functions between different TEEs. The REWIRE TEE must protect the confidentiality and integrity of all the enclave code and data at all points during execution. Ideally, it must protect against a physical adversary even though it has access to DRAM or the OS. The high-privileged component must be fully trustable, and it must be verified by the root of trust in hardware during boot time. Such high-privileged component (i.e., the Security Monitor if referring to Keystone) must perform memory isolation (e.g., by means of leveraging PMP in the case of the RISC-V ISA), while being able to receive calls from the OS and such isolated memory regions. **Remarks**: • A trusted high-privileged component must specify physical memory regions to isolate and control their memory access permissions. • Complete isolation must be achieved from all other elements except the high-privileged component and the own isolated memory region.

Connected to other requirements	• **FR.FR.2: Secure remote asset management and reconfiguration effectiveness**: The SW/FW update provenance is computed inside an isolated memory region with the support of the TEE. • **FR.FR.3: Device status auditing**: Confidentiality and integrity is crucial for auditing which also includes attestation results. • **FR.FR.10: Flexible and reliable key management**: Key management is the baseline for providing the necessary crypto primitives for assuring confidentiality and integrity, while the TEE can allow the key generation take place in enclaves. • **FR.FR.12: Data provenance**: Confidentiality and integrity is crucial for data provenance. Also, the TEE can allow intra-enclave enhancement of data provenance.

KPIs

Description	Value
Elapsed time for a complete secure device onboarding, including the issuance of VCs through ABS.	- time for a complete secure device onboarding $<$ 10 sec
Elapsed time for certificate creation per device. We consider linear of number of attributes inside the credentials.	- time for certificate creation per device approx. $<=$ 3 sec
Elapsed time for certificate creation per enclave. We consider linear of number of attributes inside the credentials.	- time for certificate creation per enclave approx. $<=$ 3 sec
Number of crypto primitives that can be supported (crypto agility).	- number of crypto primitives $>=$ 3
On average end-to-end (including communication time between trusted and untrusted world) execution of crypto operations either signature or encryption/decryption.	- average communication time of crypto operations $<=$ 2 sec
Overhead introduced by the size of the credential	- linear to the number of attributes disclosed to the credential
Number of operations supported by REWIRE TCB (i.e., secure storage, sw update, attestation etc.).	- operations $>=$ 3
Elapsed time for certification of secure launch of enclaves as part of REWIRE TCB	- time for secure enclave launch $<=$ 10 sec
Elapsed time of the execution of CIV considering key restriction usage policies and excluding network latency.	- CIV execution time $<=$ 1.5 sec
Elapsed time of checks for one key restriction usage policy.	- key restriction usage policy checks time $<=$ 200ms
Overhead imposed when the attestation process is instantiated and executed inside a REWIRE TEE.	- REWIRE TEE overhead $<$ 20%
Elapsed time for the actual platform measurement and sharing with the attestation process inside the trusted world.	- platform measurment $<$ 500ms

Table 3.14: FR.FR.14 Confidentiality and integrity in data processing

	FR.FR.15 (Mandatory)
Title	**Common Trusted Computing Base**
Actors Involved	TEE
Description	**Background**: The security of modern computing systems has come under scrutiny due to the abundance of vulnerabilities related to the high complexity of OSs and hypervisors. Due to this, it has become more attractive to rely on smaller and lower layers, i.e., FW or even immutable HW, to enforce security and to reduce the underlying TCB. Also, due to the various and diverse systems, a common TCB integrated in all the different use cases is necessary. Briefly, the TCB of a device is the software stack and hardware components that are required for it to function correctly and guarantee the security and operational requirements. A common TCB ground can ensure the applicability of the formal verification and the security-by-design principle of REWIRE and thus its crucial for the project. In addition, a common base enables the integration to the various endpoints in REWIRE (e.g., oracles, end user device and back-end). **Description**: In REWIRE, the TCB solution should be used in all the diverse and safety-critical systems of the project. A common TCB will foster the evaluation the defined theorem proofs and the crypto implementations. Such a requirement leads us to the adoption of an open-source framework for RISC-V for all the systems that are going to be used in the REWIRE use cases. On top of that, by being agnostic of the underline TCB, REWIRE seeks to foster collaboration and innovation within the IoT ecosystem, allowing more flexible and adaptable solutions as the technology evolves. Thus, it is a strategic decision that ensures the future-proof of REWIRE advancements in the safety-critical IoT systems.

	Remarks: • REWIRE should aim to create a TCB that can be used in all the diverse systems of the project. • A common TCB will reinforce the assurance level as the formally verified underline technology among the different systems.
Connected to other requirements	• **FR.FR.7: Zero Touch device Onboarding**: A common TCB is necessary to achieve zero touch device onboarding. • **FR.FR.8: Dynamic Credential Management**: A common TCB is the necessary trust anchor for credential management. • **FR.FR.10: Flexible and reliable key management**: A common TCB is the necessary trust anchor for key management. • **FR.FR.11: Trust Aware Continuous Authentication and Authorisation**: A common TCB is the necessary trust anchor for continuous authentication and authorisation. • **FR.FR.12: Data Provenance**: A common TCB is the necessary trust anchor for data provenance. • **FR.FR.13: Data veracity management**: A common TCB is crucial for the data veracity of the collected data. • **FR.FR.16: Operational assurance and configuration integrity**: A TCB acts as the trust anchor for assuring correct operation and configuration integrity of a device. • **FR.FR.17: Chain of trust creation**: A TCB acts as the trust anchor for trust establishment.
KPIs	**Description** / **Value** Number of operations supported by REWIRE TCB (i.e., secure storage, sw update, attestation etc.). / - operations $>= 3$ Elapsed time for certification of secure launch of enclaves as part of REWIRE TCB / - time for secure enclave launch $<= 10$ sec

Table 3.15: FR.FR.15 Common Trusted Computing Base

	FR.FR.16 (Mandatory)
Title	**Operational assurance and configuration integrity**
Actors Involved	Attestation Agent, TEE, BC Infrastructure, Risk Assessment
Description	**Background**: To ensure the overall trustworthiness of the IoT ecosystem, data are collected and analysed in order to detect potential misbehaviours. This involves management of provenance information especially in distributed environments such as IoTs. Decentralised environments must establish the origin of data (data provenance) and verify it to take a proactive approach to security rather than a reactive approach post-attack. Further, device provenance provides guarantees that a device has been onboarded correctly in the system. Moreover, the health (status) of a device is equally important in the context of REWIRE. Device status corresponds also to the results from device attestation and/or SW/FW update validation, considering corresponding policy regulations, and it is used to determine the device and the systems overall trustworthiness. Several attestation schemes exist in the literature covering all phases of a device's execution; from the trusted boot and integrity measurement of an edge device, enabling the generation of static, boot-time or load-time evidence of the system's components correct configuration (e.g., Configuration Integrity Verification), to the runtime behavioural attestation of safety-critical components of a system providing strong guarantees on the correctness of the control flow (Control-Flow Attestation). However, due to scalability and performance issues of these schemes, there is a need to increase the efficiency necessary. Assurance of the operation and configuration integrity are crucial; thus the SW/FW update needs to be attested in an efficient way. **Description**: It is difficult to have a high level of certainty about where sensitive data might be located and the origin of data in the fully decentralised environment that is REWIRE. Thus, we need to differentiate between the integrity and attribution of a message from the verifiability of the message, which is captured for the requirement of misbehaviour detection, to determine data provenance. Tracking the origin of data may cause privacy issues, thus, where appropriate we should consider using digital signatures and DAA-A in REWIRE (see the zero-touch onboarding requirement). In doing so, we can enable the verification of authorship of verifiable statements to belong to a device without breaching the privacy of the device. With respect to device status in REWIRE, the health of a device is useful information to actors in the BC infrastructure to determine the devices' trustworthiness when attempting to onboard into the network or set up communication with other devices in the network. Device status also aids in REWIRE identifying security risks/vulnerabilities to reduce the chance of attacks.

	REWIRE requires a new breed of efficient attestation mechanisms that can provide the same level of security guarantees, while increasing the efficiency due to the limited vector of properties that need to be attested. Based on this attestation assessment the state of the component could be derived and classified into compromised, under-attack, out-dated, normal operation, and others. On top of that, REWIRE's TEE provides the necessary mechanisms to enable the isolated execution of sensitive functions in enclaves. This can be achieved by utilising Keystone's Security Monitor (SM), that ensures physical memory isolation for said enclaves by manipulating RISC-V PMP registers. Critical services, including cryptographic services like key derivation must run in isolation from any other software, including privileged one, are running on the same platform. **Remarks**: • The TEE (Keystone) can allow intra-enclave enhancement of data provenance. • REWIRE state and/or integrity system verification acts as an ex-ante requisite in many situations, e.g., prior to a TEE Migration but also as an ex-post requisite for others, e.g., after an OTA FW update. • The REWIRE TEE must ensure that the memory assigned to a critical service is inaccessible from other services and even the OS. • REWIRE attestation is efficient since it focuses only on the sequence of assembly commands from the ISA instruction set of the RISC-V.
Connected to other requirements	• **FR.FR.3: Device status auditing**: Device provenance and status is necessary for the device status auditing. • **FR.FR.6: Secure Measurement and Attribute Extraction**: Secure measurement and attribute extraction is of paramount importance to verify the device status. • **FR.FR.7: Zero-touch onboarding**: Zero-touch onboarding is a perquisite for the device provenance and status. • **FR.FR.15: Common Trusted Computing Base**: A common TCB is the cornerstone for assuring correct operation and configuration integrity of a device.
KPIs	**Description** / **Value** Elapsed time of the execution of CIV considering key restriction usage policies and excluding network latency. — - CIV execution time $<=$ 1.5 sec Elapsed time of checks for one key restriction usage policy. — - key restriction usage policy checks time $<=$ 200ms Overhead imposed when the attestation process is instantiated and executed inside a REWIRE TEE. — - REWIRE TEE overhead $<$ 20%

Table 3.16: FR.FR.16 Operational assurance and configuration integrity

	FR.FR.17 (Mandatory)
Title	**Chain of trust creation**
Actors Involved	TEE, Blockchain Infrastructure
Description	**Background**: A major challenge is to guarantee the operational assurance and trust establishment among different and heterogeneous devices in a distributed environment to ensure its trustworthy operation in the context of a collaborative infrastructure. The chain of trust concept is a hierarchical series of trusted components within a system that ensures the security (and the integrity) of its software and hardware components, while part of establishing the chain of trust is the creation of a secure environment where only trusted and verified components may operate and interact with each other. However, restricted and distributed IoT devices does not allow the deployment of more complex protection schemes such as TPMs to support the trust concept. Thus, there is an urgent need for more lightweight software-based solutions that can act as the RoT, the baseline of the trust chain. **Description**: RoT is the cornerstone to facilitate remote attestation protocols by offering the necessary crypto functions that verify the trustworthiness of the underline device. In the context of REWIRE, TEE functionalities will be utilised, as software and more lightweight solution of RoT, to manage the device's keys. More specifically, REWIRE TEE should provide the necessary mechanisms for trust management and for creating a chain of trust among heterogeneous and limited IoT devices. In essence, this trust should be verifiable and based on reliable evidence, allowing for transparency and accountability within the IoT ecosystem. This approach will assist on the delegation of critical tasks (e.g., in the automotive domain) in devices proved their trustworthiness. **Remarks**: • Attestation protocols will guaranty the trustworthiness of devices in such a distributed environment

Connected to other requirements	• **FR.FR.15: Common Trusted Computing Base**: A common TCB is the baseline as the trust anchor for the creation of chain of trust.	
KPIs	**Description**	**Value**
	Time for storing trust-related information to the Blockchain for auditability and certifiability.	- storage time $<=$ 5sec, since this is not a real-time operation

Table 3.17: FR.FR.17 Chain of trust creation

	FR.FR.18 (Mandatory)	
Title	**Policy-based device state configuration**	
Actors Involved	TEE	
Description	**Background**: Complex distributed systems require policies that must be set during design time and implemented during runtime. REWIRE is no exception to this. All use case scenarios will make use of policies for their correct performance. Examples can include (1) the onboarding of a new device, which can include policy-regulated accesses to the network (2) performing SW/FW updates, that must be policy-controlled and attested (3) migrating data and execution to a more secure remote enclave, or (4) revocation of credentials for a suspicious device. **Description**: The REWIRE TEE must express the configuration of the design time phase and must be able to measure itself and report back its configuration. This expression of the design time phase shall allow for policy enforcement and policy-based access to different keys for different uses. Ways to do this must be explored (i.e., will policy enforcement happen at the Security Monitor level? Will some of it run inside enclaves?). Nevertheless, the Security Monitor should be modular to adapt to different devices and different policies. In this regard, the TEE provider could configure, build and deploy the security monitor to suit different needs. The REWIRE system requires that different applications in the untrusted world that correspond to basic functions, common to all use cases, communicate either to the Security Monitor or to/from an enclave using the Security Monitor as a means to protect the enclave. This must be performed according to policies provided by the use cases (security models requirements) and using the Security Monitor as the central point for arbitration. The TEE requires that: • The Security Monitor must have the highest privileges, be trustable and have access to the HW root of trust and any crypto co-processor at the HW level, if applicable. Generally speaking, the platform must support storage of a device root key accessible only to the bootloader / Security Monitor.- The Security Monitor must have the highest privileges, be trustable and have access to the HW root of trust and any crypto co-processor at the HW level, if applicable. Generally speaking, the platform must support storage of a device root key accessible only to the bootloader / Security Monitor. • The Security Monitor must be able to arbitrate communication between the untrusted host and the machine (i.e., allow or deny the protection and execution of an enclave that was initially allocated by the untrusted host). This must be performed following the aforementioned policies, which should cover: Device Secure Enrolment, Attestation, Function migration and Isolation. • Beyond arbitration, the Security Monitor must provide keys for securing critical enclave functions such as enclave migration or local and remote attestation. **Remarks**: • The Security Monitor (trusted and highest-privileged component) must express design time phase policies by being able to allow or restrict access of untrusted hosts to the trusted world.	
Connected to other requirements	• **FR.FR.2: Secure remote asset management and reconfiguration effectiveness**: The verification of the SW/FW update provenance determines whether such update can be run inside an isolated memory region. • **FR.FR.3: Device status auditing**: Attestation must be policy-regulated • **FR.FR.10: Flexible and reliable key management**: The TEE can allow that key generation takes place in enclaves, while some of it can take place in the Security Monitor	
KPIs	**Description**	**Value**
	Efficiency of controlling the granularity of the level of tracing from the policy orchestrator and the enforcement of MSPL policies	- time to control and enforce a policy $<=$ 200 ms

Table 3.18: FR.FR.18 Policy-based device state configuration

	FR.FR.19 (Mandatory)	
Title	**Formally Verifiable Security Modules**	
Actors Involved	The hardware-layer of the REWIRE architecture. Notably, the RISC-V core and an AES co-processor.	
Description	**Background**: The hardware design should be formally verified in order to provide guarantees about the required level of trustworthiness when such devices are placed in their operational environments. **Description**: The design phase of REWIRE aims at using formal methods to assert security properties of the proposed hardware architecture, therefore reducing the number of threats that need to be covered/detected by the run-time attestation. Hence, the hardware architecture used in REWIRE needs to enjoy a high-level of trustworthiness regarding its security claims. Examples of such security claims could be protection against side-channel attacks (e.g., Spectre and Meltdown) at the RISC-V core micro-architectural level, for instance. REWIRE should also focus on functional correctness of hardware IPs, resulting in an architecture that is robust against vulnerabilities and exploits stemming from implementation bugs. Regarding the latter, components that are formally verified against a higher-level specification shall be used: one example of such could be a verified functionally correct instance of an AES co-processor. Such AES-co-processor (either tightly or loosely coupled) shall be accessed with a RISC-V custom instruction, which will be part of the REWIRE extended RISC-V ISA. The formal verification tools and languages for achieving this requirement can vary. REWIRE should explore the use of different techniques, including model checking (both directly at the model-level and through external tools) and theorem proving (likely with the Coq proof assistant). **Remarks**: • REWIRE should explore the use of different techniques such as model checking and theorem proving.	
Connected to other requirements	• **FR.FR.20: Expression of requirements and assumptions using formal specification languages**: Expression of requirements and assumptions using formal specification is necessary to formally verifiable HW-security designs.	
KPIs	**Description**	**Value**
	Number of faults identified.	- number of faults > 5

Table 3.19: FR.FR.19 Formally Verifiable Security Modules

	FR.FR.20 (Mandatory)
Title	**Expression of requirements and assumptions using formal specification languages**
Actors Involved	REWIRE's Security Requirements, AADL representation of the REWIRE architecture, Formal Verification Tools, Provable secure crypto for secure communication, Security models
Description	**Background**: From the perspective of the design stage, each IoT Service Provider needs to set its overarching requirements with formal descriptions using specification languages. To do so, requirements stemming from the use cases and the REWIRE framework itself need to be expressed using specification languages so that to be given as inputs in the Formal verification framework of REWIRE. The process of designing, implementing, and deploying cryptographic mechanisms is notoriously difficult due to the prevalence of design flaws, implementation bugs, and side-channel vulnerabilities, even in widely deployed mechanisms. Each step of the process is highly complex and filled with potential pitfalls. At the design level, cryptographic mechanisms must meet specific security objectives against a defined class of attackers. This often involves composing intricate building blocks, with abstract constructions making up primitives, primitives forming protocols, and protocols comprising systems. At the implementation level, high-level designs must be translated into concrete functional details such as data formats, session state, and programming interfaces, while prioritizing interoperability and performance. At the deployment level, the implementation must also consider low-level threats, such as side-channel attacks, that were not considered at the design level. All the above strengthens the case for a multi-phase design toolchain supported by automated processes and tools that will support the verification and validation of safety and security profiles for embedded systems. **Description**: REWIRE's Security Requirements shall be expressed in formal languages that can be automatically given as inputs for tools such as theorem provers and model checkers. To achieve that, REWIRE shall use AADL plugins for automatic reasoning about requirements. Requirements shall finally be mapped to corresponding Assurance Cases using Resolute, an AADL plugin for formal reasoning. A level of automation in the verification processes should be achieved in REWIRE. This may refer only to a smaller part of the verification process. In addition, it depends on the methodologies that will be used (e.g., Theorem proofs or model checking). A toolchain specific to aid the security and safety designs will be based on a mature model-based system engineering approach that will enable both verification of designs as well as, traceability of requirements from systems level (overarching requirements) till low level embedded specifications.

	Remarks: • By connecting security requirements to a formal compelling argument, trust that every single security requirement is satisfied by the architecture is achieved. • REWIRE will offer a 4-layered security sandbox comprising a toolchain that can efficiently safeguard and assess the trustworthiness level of an edge device throughout its entire lifecycle and application stack.
Connected to other requirements	• **FR.FR.19: Formally Verifiable Security Modules**: Expression of requirements and assumptions using formal specification is necessary to formally verifiable HW-security designs.

KPIs	**Description**	**Value**
	Number of use cases for successful requirement traceability (i.e., leveraging resolute).	- number of use cases = 3

Table 3.20: FR.FR.20 Expression of requirements and assumptions using formal specification languages

	FR.FR.21 (Mandatory)
Title	**Provably secure crypto protocols and algorithms**
Actors Involved	REWIRE Design-time architecture, Formal Verification Tools, Provable secure crypto for secure communication, Security models
Description	**Background**: Cryptographic protocols cannot just be based on ad hoc constructions evaluated by attempting attacks against them, as this approach leaves the door open to unexpected attacks or loosely defined security properties. Instead, state-of-the-art protocols must come with precise definitions on the properties they should achieve, and a mathematical proof (reduction) that these properties are indeed enforced based on some well-defined assumptions. In addition, implementations of these protocols should undergo formal verification ensuring that they do meet the protocol's specifications. **Description**: REWIRE will use several cryptographic protocols, including: • A (symmetric) authenticated encryption algorithm, specifically designed for the critical operation of securely distributing SW updates. Depending on the context, this scheme could also be used for transmission of other critical data. • Various symmetric and asymmetric protocols and algorithms allowing operations such as key exchange, digital signature and encryption of regular data exchanges, etc. All exchanges of sensitive data between REWIRE devices should be protected (authenticated and/or encrypted) through cryptographic means. In addition, the authenticated encryption protocol used for SW update distribution should provide protection against side-channel attacks. This authenticated encryption scheme, specifically designed for REWIRE, should be provably secure, and its security proof should take into account the threat of side-channel attacks, to the extent of the current state of the art in this field. Implementations of this scheme designed for REWIRE should be formally verified to ensure it meets its specifications. Other cryptographic protocols used in REWIRE should correspond to widely used good practices, such as industrial or de facto standards. They should ideally be provably secure, unless other constraints (such as interoperability with standards) prevent it. REWIRE will investigate on mathematical and modelling process aiming to provide proofs on the correctness of the crypto implementations and on the security of their primitives. **Remarks**: Provably secure schemes will take the form of: • A specification of the scheme, together with a security proof, in the form of an academic paper. • A specification of the scheme, together with a security proof, in the form of an academic paper.
Connected to other requirements	• **FR.FR.8: Dynamic credential management**: Dynamic credential management is imperative in order to ensure the security of crypto protocols and algorithms. • **FR.FR.10: Flexible and reliable key management**: Key management is the baseline to ensure the security of crypto protocols and algorithms.

KPIs	**Description**	**Value**
	Number of security protocols (e.g., attestation, LRBC, AES) to be formally verified.	- protocols to be formally verified $>= 3$
	Number of crypto operations (e.g., LRBC, ABSC, SW update, live migration and zero-touch onboarding) to be security analysed.	- crypto operations to security analysed $>= 5$

Table 3.21: FR.FR.21 Provably secure crypto protocols and algorithms

	FR.FR.22 (Mandatory)	
Title	**Trusted Root-of-Trust and secure boot**	
Actors Involved	TEE	
Description	**Background**: Many of REWIRE's data and applications rely on the use of TEEs in the ecosystem's devices / components, but for the former to be trustworthy they need to incorporate a secure-by-design trusted RoT. The RoT, for all intents and purposes, must be (as) inaccessible (as possible) outside the device, and in this way the device can trust the keys and other cryptographic information it receives from the RoT module and other components in the ecosystem can trust the device. The RoT must remain the foundation on which other secure operations depend and thus must contain protected keys for cryptographic functions to enable a secure bootstrapping process – this including a secure SM boot process. Every device / component in the REWIRE ecosystem must be able to boot using exclusively trusted and authenticated firmware / software. Otherwise, the device might reboot at some point and run malicious code, rendering the device compromised. **Description**: The RoT (e.g., a hardware based crypto engine) must cryptographically measure the bootloader software and all software loaded by the bootloader, and then use the processor's private key to sign the measurement, producing a certificate. The public key associated with the processor's private key is signed by the manufacturer and it is known by others. The privacy of the secret key must be guaranteed (several methods can be explored, like using TRNG or PUFs). As part of the fully secure boot process, the RoT must also authenticate the SM. Specifically, at each CPU reset, the RoT must measure the SM image, then generate a fresh attestation key from a secure source of randomness, store it to the SM private memory, and signs the measurement and the public key with the hardware-visible secret key mentioned earlier. As seen, the RoT protects important assets such as keys of the device (which are then used for other security-critical functions like attestation) on shielded or protected locations in the device. The integrity of these locations should be preserved, and they should be prevented from unauthorized modification otherwise the identity of the device might get compromised and/or the RoT might not be able to verify different pieces of software (e.g., the SM) or compute valid signatures. During the operation of the device, the SM can provide security services to the TEEs or to other applications. If these services are anchored on the RoT, the SM should be able to attribute the services to the owning entity. **Remarks**: • A secure-by-design RoT (e.g., a hardware based crypto engine) must be able to cryptographically measure the bootloader software, the Security Monitor and any other software at boot-time. • The TEE RoT in REWIRE can act as the foundation for critical security services / concepts such as ownership, integrity and authenticity. • (Relations of trust) The Keystone user trusts the SM only after verifying if the SM measurement is correct, signed by trusted hardware (RoT), and has the expected version. The SM only trusts the hardware, and the host trusts the SM.	
Connected to other requirements	• **FR.FR.3: Device status auditing**: Secure measurements and attribute extraction are necessary for auditing of the state of the device. • **FR.FR.15: Common Trusted Computing Base**: At the heart of the TCB we have the underline RoT. • **FR.FR.16: Operational assurance and configuration integrity**: The first step to assure correct operation and configuration integrity is the trusted boot. • **FR.FR.17: Chain of trust creation**: The main idea behind the chain of trust is to have a trusted RoT. • **FR.FR.24: Platform Measurement Service**: In order for a TCB to provide valid measurements should be trusted.	
KPIs	**Description**	**Value**
	Elapsed time to complete a trusted boot in a device.	- time for a trusted boot $<$ 5 sec

Table 3.22: FR.FR.22 Trusted Root-of-Trust and secure boot

	FR.FR.23 (Mandatory)
Title	**Immunity to Software Attacks and Integrity protection**
Actors Involved	TEE

Description	**Background**: The increasing number of online IoT and edge devices exposes them to several software attacks. RoT is the baseline to provide immunity to these remote software attacks and assuring the integrity of the device. Ideally, immunity to software attacks and integrity protection should also protect against physical attacks, however in reality physical attack protection is difficult. Software RoT is mainly designed to resist software-based attacks, however if correctly designed it can also protect from hardware-based attacks. **Description**: As the components of the RoT are inherently trusted and are supposed to be secure by design, it should be guaranteed that the security critical functions that they perform cannot be compromised via software attacks. These functions can be implemented in hardware and/or protected firmware, but in any case, it should be guaranteed that they cannot be modified via a software attack. Additionally, the RoT protects important assets such as keys of the device (e.g., ownership, attestation, device keys) on shielded or protected locations in the device. The integrity of these locations should be preserved, and they should be prevented from unauthorized modification otherwise the Identity of the device might get compromised and/or the RoT might not be able to verify different pieces of software neither to compute valid signatures. **Remarks**: • REWIRE's customizable TEE as the RoT should provide security form software attacks and integrity protection to all the critical functions.
Connected to other requirements	• **FR.FR.22: Trusted Root-of-Trust and secure boot**: A trusted RoT will be in position to provide immunity to software attacks and integrity protection. • **FR.FR.24: Platform Measurement Service**: The RoT should provide an assurance that that measurements are correctly captured thus is necessary to be immune to software attacks and have integrity protection.
KPIs	**Description**: Percentage of SW-based attacks that can be successfully identified. — **Value**: - Identified SW-based attacks approx. 95%

Table 3.23: FR.FR.23 Immunity to Software Attacks and Integrity Protection

	FR.FR.24 (Mandatory)
Title	**Platform Measurement Service**
Actors Involved	TEE
Description	**Background**: Endpoint devices contain hardware, firmware, drivers, OS, and software that affect the integrity and security of the devices and the network within which they reside. Thus, a service providing knowledge on the current posture of these endpoint devices is of paramount importance. Platform integrity measurement is a core service of attestation mechanisms based on which can be decided whether the platform is in a correct state or not. Platform integrity measurement is important for building up a trusted environment. This service provides the ability to reliably create characteristics of the platform by calculating hashes of core and/or data. Current measurement methods suffer from high computational complexity and heavy data processing, thus they consume a lot of resources and spending too much time. **Description**: One of the security services that the SM offers is a measurement service of the platform characteristics, in such a way platform users can rely on a report that describes the current state of the platform and is emitted by this platform and signed by the SM and/or by the device identifier. The SM can prove to a remote client that some enclave contains the program expected, and is running on hardware that is trusted. This report might be a cryptographic hash with a list of features or services provided by the RoT. **Remarks**: • Integrity measurements may be in the form of cryptographic hash. • The underline RoT must act in concert to enable reliable and trustworthy measurements.
Connected to other requirements	• **FR.FR.22: Trusted Root-of-Trust and secure boot**: A trusted RoT will be in position to provide valid platform measurement. • **FR.FR.23: Immunity to Software Attacks and Integrity protection**: The RoT should have immunity to software attacks and integrity protection to provide a valid platform measurement.
KPIs	**Description**: Elapsed time for the actual platform measurement and sharing with the attestation process inside the trusted world. — **Value**: - platform measurment < 500ms

Table 3.24: FR.FR.24 Platform Measurement Service

Chapter 4

Threat Landscape Analysis

Recall that the motivation behind the REWIRE Risk Assessment pipeline is to *maintain an up-to-date view of the risk quantification graph of the system*, based on which we are able to calculate the **Required Trust Level (RTL)** of the target service-graph-chain. The RTL, beyond being a measure of the minimum quantifiable trust value of a device needed so that a device can be accepted into the SoS, it also defines the *necessary parameters to be able to characterize and quantify the trust level of the device*. This is achieved by identifying the risks with the highest **likelihood** and **impact**, and delineating the security controls that can protect against those risks. Then, from these security controls, we are able to map the evidence that needs to be monitored during runtime in order to obtain **Proof of Execution** of these security controls. Thus, *through exhausting threat profiling, REWIRE is one of the first projects of its kind to provide a* ***detailed mapping of the types of evidence that we need to monitor in a verifiable manner during runtime****, based on which the trust characterization of the device will take place*.

Comparison Between 2016-2024		
OWASP-2016	**OWASP-2024-Release**	**Comparison Between 2016-2024**
M1: Improper Platform Usage	M1: Improper Credential Usage	New
M2: Insecure Data Storage	M2: Inadequate Supply Chain Security	New
M3: Insecure Communication	M3: Insecure Authentication / Authorization	Merged M4&M6 to M3
M4: Insecure Authentication	M4: Insufficient Input/Output Validation	New
M5: Insufficient Cryptography	M5: Insecure Communication	Moved from M3 to M5
M6: Insecure Authorization	M6: Inadequate Privacy Controls	New
M7: Client Code Quality	M7: Insufficient Binary Protections	Merged M8&M9 to M7
M8: Code Tampering	M8: Security Misconfiguration	Rewording [M10]
M9: Reverse Engineering	M9: Insecure Data Storage	Moved from M2 to M9
M10: Extraneous Functionality	M10: Insufficient Cryptography	Moved from M5 to M10

Figure 4.1: OWASP Mobile Top 10 risks comparison between 2016 and 2024

In this context, it is important to note that REWIRE follows an extended threat model that does not remain static, but may **dynamically shift and expand throughout the lifecycle of the system**, considering the rapidly evolving threat landscape affecting IoT devices. For instance, consider the Top 10 Mobile Risks of 2024 as identified by the **Open Worldwide Application Security Project (OWASP)** [18] shown in Figure 4.1, where we observe that comparing the ranking of threats between 2016 and 2024 yields interesting results with the emergence of new threats, while other threats may rise or decrease in prominence. There is a shift towards vulnerabilities that stem from insecure components that may attempt to become part of a secure system. This is fully aligned with the vision of REWIRE regarding **"Secure Systems of Systems" (SoS)**, requiring the type of composite attestation processes that we are considering in REWIRE based on a multitude of runtime monitors. Thus, securing systems throughout their operational lifecycle necessitates the consideration of the potential for such changes in the threat landscape.

Then, based on this threat model, REWIRE provides a rich set of **security mechanisms** and **Trusted Computing extensions** considering an extended set of trust properties, which go beyond integrity and consider a wide set of properties, such as confidentiality, correctness, authenticity, auditability, availability, etc., as will be expanded in Section 4.1. However, it is also important to consider the **types of evidence** that need to be monitored in order to evaluate the fulfilment of the aforementioned types of properties. In this context, we need to consider the adoption of the **zero-trust** principle by REWIRE, meaning that *we make no assumptions on the trustworthiness level of devices, and their trustworthiness level needs to be evaluated dynamically considering a set of trust properties*. As detailed in Section 2.1.2, the **Compositional Verification and Validation** component helps us to identify the properties that need to be attested during runtime in this regard, as well as the corresponding types of evidence that need to monitored.

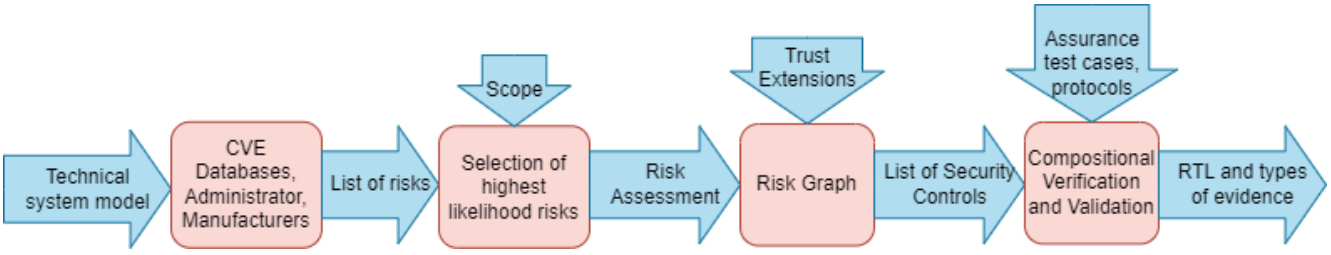

Figure 4.2: REWIRE threat modelling pipeline

The aforementioned process is compounded in the diagram of Figure 4.2, which provides a high-level overview of the process followed in REWIRE in this regard. Specifically, after the definition of the technical system model including the types of devices and their interconnectivity, the list of risks affecting the devices and system is compounded based on input by **CVE databases**, the **domain administrator**, and **device manufacturers**. Afterwards, the most relevant risks are selected, focusing on the highest likelihood risks based on the scope of the application. Then, the **Risk Assessment** component calculates the **risk graph**, which then provides the basis for the identification of **security controls** for the mitigation of risks. Then, the **Compositional Verification and Validation** component, considering the available protocols and assurance test cases, leads to the determination of the **Required Trust Level (RTL)** of the device, as well as the **types of evidence** that need to be monitored in order to verify the achievement of the RTL. It is important to note that, while the output of the verification pipeline of REWIRE cannot capture real-time attacks, REWIRE employs runtime monitors that can collect the required trustworthiness evidence during runtime.

In this Chapter, we provide an **exhaustive listing of the threats affecting the different phases of the lifecycle management of a device**, as well as the corresponding **trust extensions that can be used for the mitigation of each threat**. This approach is also combined with the **mapping of the types of trustworthiness evidence** that the REWIRE attestation and tracing mechanisms can monitor for providing the required runtime guarantees with the aforementioned threats and trust extensions. This exercise culminates in a detailed mapping between threats, trust extensions, and evidence for the mitigation of the identified threat landscape.

4.1 System Model & Security Requirements

With regards to the system model considered in the formulation of the threat landscape, we assume the existence of a **Domain**, whose infrastructure consists of multiple interconnected heterogeneous devices. Also, it is possible for devices to exhibit two types of communication: (i) **Inter-domain communication**, i.e., communication between devices belonging to the same domain, or (ii) **Intra-domain communication**, i.e., communication between devices belonging to different domains.

In Table 4.1, we provide the set of security requirements considered in the formulation of the threat landscape of REWIRE, which may be affected by the identified threats. Based on these, throughout the

remainder of this Chapter, we perform a mapping of the threats identified for each component, as well as the security properties affected by each threat.

Requirement	Description
Integrity	Ensures that data is accurate and has not been altered without authorization, either by a malicious party, or as a result of some error or misconfiguration.
Confidentiality	Ensures that information is not accessed by or disclosed to parties that do not have the appropriate permissions to do so.
Authenticity	Verifies that users or data sources are who or what they claim to be.
Authorization	Grants or denies access to resources, services, or sets of data, based on the appropriate permissions (e.g., possession of a specified set of attributes).
Correctness	Ensures that systems and data behave and function as intended.
Auditability	Enables actions and events to be recorded (e.g., the Blockchain Infrastructure) and reviewed for accountability by external certification or auditing entities.
Availability	Ensures that systems and data are accessible when needed by authorized users and devices.
Secure Enrolment	During the secure enrolment of a device, it should be verified that its local Attestation Key (AK) only ever obeys policies generated for the specific device by the Compositional Verification and Validation component, and distributed by the Policy Orchestrator.
Policy Authenticity	An adversary should not be able to use policies meant for one node to unlock the AK of a different node. By initially embedding the identifier of the device in the authorization policy of its Attestation Key, we ensure that policies approved for the device must bear the device identifier to be satisfiable.
Measurement Authenticity	When attempting to use the AK of a device predicated under a key restriction usage policy, an adversary should not be able to supply inauthentic measurements in order to unlock the AK. In addition, it should not be possible to perform path spoofing or unauthorized configuration modification that may have occurred since the last measurement, e.g., as a result of transient malware.
Policy Freshness	A device should not be abe to satisfy approved policies beyond their intended timeframes, in order to prevent adversaries from satisfying policies approved for old configuration states in an attempt to evade the remeasurement process of the current configuration of a node.
Implicit Revocation	Continuous use of a local AK required continued authorization, otherwise the device should not be able to attest to the correctness of its state.
Zero-Knowledge Verification	Verifiers should not need reference materials to determine the correctness of a Prover node, and should not be able to infer anything about its configuration. This is achieved by the Prover solely presenting a correct signature over a fresh challenge.

Table 4.1: Security Requirements in REWIRE

4.2 Adversarial model and Assumptions

Same as the Chapter 7 in D2.1, we categorise attacks in order to enable capturing the entire **compute continuum** where the target ”Systems-of-Systems” are deployed. Specifically, this includes (i) **Host attacks** which mainly target far-edge devices, (ii) **Network attacks** affecting the communication with edge devices or cloud-based servers, and **Physical/algorithmic-based attacks**, which are attacks that rely on exploiting the physical properties of the devices or the communication medium, or the algorithmic structure of the processes running on the device.

- **Host-based attacks:** These refer to attacks that aim to disrupt the normal operation of the device. In other words, by targeting individual computer systems, an attacker exploits vulnerabilities in local software. A software attacker can try to control applications installed in host or untrusted OS. Thus, a malicious party may be able to launch a wide variety of attacks, such as introducing malicious information in network communications, maliciously affecting enclave management (launching adversarial enclaves or performing forking or cloning attacks for manipulating the state of the enclave), maliciously modifying any memory outside of TEE to add, dropping or replaying

enclave messages. These attacks can lead to authenticity, integrity and confidentiality violations regarding the data handled by the device.

- **Network-based attacks:** These refer to attacks that intercept information transmitted through the network infrastructure or disrupt the operation or communication devices. Specifically, by targeting a communication infrastructure, an attacker can impact the integrity of data in transit by intercepting, disrupting or manipulating network traffic. Here, we list some examples of network-based attacks. (i) An attacker can eavesdrop the communication channel or try to inject fake messages (interception attack). Hence, an adversary can access information without authorisation. This kind of attacks can pose threats to privacy, confidentiality and integrity violations. (ii) Sybil attacks enable a malicious party to add multiple ghost nodes in a network or system to operate many false identities simultaneously (e.g., in the Blockchain), which will pose threats to authorisation and confidentiality violations. (iii) An impersonation attacker can pretend to be someone else to steal sensitive data from an entity in the system. This kind of attack can lead to confidentiality violations. (iv) An attacker can launch man-in-the-middle attack, in which the attacker put himself/herself in the middle of two entities (normally a user and an application), to intercept their communications and data exchanges and use them for malicious, unauthorised actions. This attack can pose threats to authorisation and authentication, leading to privacy, confidentiality and integrity violations. Finally, an important network-based attack is denial-of-service attack, which can disrupt the operation of enclavized applications. Keystone also does not provide any protection mechanism against cloning attacks that can lead to multiple enclaves with the same IDs, thus leading to potential availability violations through DoS attacks for depleting the available resources.

- **Physical/algorithmic-based attacks:** These attacks, also referred to as **Side-Channel Attacks (SCA)**, aim to compromise the physical or underlying algorithmic structures of the devices, and mainly the mathematical structure of the cryptographic enablers. Thus, by targeting at underlying system design/implementation, an attacker exploits hardware or computational algorithm weakness. Specifically, an attacker can run SCA against specific device to recover the long-term key, which posts threats to confidentiality and integrity violation. Moreover, an attacker can run SCA on a device in order to inject a single fake message (without recovering the long-term secret), which leads to threats to integrity violation. Finally, an attacker can run SCA on a device in order to read a single message (without recovering the long-term secret), which can pose threats to confidentiality violation. Also, it is important to note that *SCA are the prime types of vulnerabilities and attacks affecting quantum crypto schemes*. Considering the rapid advancements in the field of quantum computing in recent years, there is a need to prepare for the **Post-Quantum (PQ)** transition of infrastructure that employs classical cryptography. Thus, while this does not fall under the core research activities of REWIRE, we also need to consider **attacks/threats the quantum computer poses to cryptographic primitives based on traditional mathematical hard problems**, such as the ones highlighted in Table 4.2. At a high level, PQ crypto schemes can be categorized in various PQC families, namely: (i) **Lattice-based**, which are based on the mathematically hard structure of lattices, such as Learning-With-Errors (LWE), and have been standardized by NIST with the Module-Lattice-Based Key Encapsulation Mechanism (ML-KEM) and Module-Lattice-Based Digital Signature Algorithm (ML-DSA) schemes. (ii) **Code-based**, which are based on decoding problems in error-correcting codes. (iii) **Multivariate**, based on solving systems of multivariate quadratic equations. (iv) **Hash-based**, which use hash functions for security, typically in signatures, NIST-standardized with the Stateless Hash-Based Digital Signature Algorithm (ML-DSA). (v) **Isogeny-based**, which are based on problems involving isogenies between elliptic curves.

PQC Family	Benefits	Limitations	Applications

Lattice-based	• Well-defined mathematical constructions to guarantee long-term security. • Mature security analysis with strong resistance against quantum attacks. • Generally efficient for key generation and encryption. • Well-studied with numerous practical implementations and optimization research. • Flexible with a range of cryptographic primitives (e.g., encryption, signatures). • Standardized as part of NIST initiatives	• Key sizes can be relatively large. • Some schemes can have high memory usage - drawback for constrained devices. • Vulnerable to certain side-channel attacks if not CMs are in place • Lack of validation of CMs, especially in the case of composable cryptosystems (e.g. Kyber-Dilithium as a single cryptosystem)	• Communications Security (PQTLS, IPsec) • Personal data exchanges security and privacy
Code-based	• Well-suited for use in encryption and digital signatures. • Good for long-term security, withstanding both classical and quantum attacks.	• Extremely large public key sizes • Less efficient due to increased computational complexity. • Limited range of applications	• Protection of intellectual property • Voting systems • Blockchain
Multivariate	• High efficiency and fast signature generation and verification. • Good candidate for digital signatures, especially in low-resource environments. • In standards-track	• Limited cryptanalysis and need for further investigation on weaknesses and countermeasures • Not as widely adopted or understood as other PQC families. • Large key and signature sizes • Some multivariate schemes have been broken, leading to trust issues and uncertainty about long-term security	• Public key cryptosystems • Applications requiring fast authentication, such as smart cards and RFID
Hash-based	• Strongly based on the security of hash functions, which are well-studied and trusted. • Relatively simple and secure designs with rigorous security proofs. • XMSS and LMS already provided in commodity secure elements. • Standardised in Trusted Computing • SPHINCS+ standardized in NIST, FAEST is in NIST's consideration for the extra call	• Limited to digital signatures and not suitable for encryption or key exchange. • Signature schemes often have a large size • Stateful • Stateless variants are computationally more intensive, affecting performance.	• Digital signatures • Blockchain

Isogeny-based	• Small key sizes and signatures, making it ideal for environments with strict memory constraints. • Based on the hard problem of finding isogenies between elliptic curves, offering strong quantum resistance • can benefit from existing acceleration approaches. • Good potential for key exchange applications and good candidate for drop-in replacement as it is based on ECC. • In standards-track	• Computationally intensive, resulting in slower performance. • Lack of constant time optimizations • Immature cryptanalysis and need for further investigation on weaknesses and countermeasures. • Still open on how to design securely and accelerate	• Practical benefits for small scale devices (as in the context of the envisioned Drone use case)

Table 4.2: Benefits and limitations of Post-Quantum Crypto (PQC) families

Considering the above, in the following, we provide a list of threats and attacks against the various REWIRE components, including property violations and security measures that can be used for their mitigation. However, especially for the verification process, formal verification techniques aim to mathematically prove correctness and security of systems and software. Thus, in the verification process, we list challenges in the following section, instead of attacks/threats.

4.3 Attacks/Threats against the REWIRE Operational Phases

In Section 4.2, we defined the adversarial model of REWIRE, followed by different types of attacks and threats affecting the devices considered in REWIRE. Based on the (revised) adversarial model and assumptions, in this section, we list all possible attacks/threats against all core REWIRE phases. As detailed in Chapter 2, the core phases of the REWIRE action workflow, consist of various sub-steps which may be affected by different types of threats and vulnerabilities that will be detailed throughout the remainder of this Section. Considering the above, the core phases of REWIRE can be summarized as follows:

1. ***Design-time Phase:*** This phase entails the initial instantiation of the REWIRE framework, starting with the initial definition and expression of the overarching requirements by the System Administrator, the use case providers, and the device manufacturers. Then, the Risk Assessment component is responsible for performing an initial risk calculation for the device, both individually and as part of a service graph chain. The core element of the Design Phase is the **Compositional Verification and Validation component**, which is responsible for the **formal verification** of security schemes and processes, and the definition of the trust boundary of the device, i.e., the set of trust properties that cannot be verified during design-time and need to be verified at runtime. The Formal Verification process may pose various challenges, whether they originate from malicious parties or not, that affect not only the correctness of the process, but may also have adverse effects on the execution of the subsequent steps of the REWIRE pipeline in the Runtime phase. These challenges are listed in Table 4.3. Finally, the output of the design-time phase also includes the construction of the security and operational policies that dictate the execution of security enablers for verifying the trustworthiness level of the runtime properties.

2. ***Runtime Phase:*** The Runtime Phase includes operations which may be subject to various types of threats and vulnerabilities, as will be detailed throughout the remainder of this Section. After the Design phase is complete, the **Zero-Touch Onboarding (ZTO)** scheme of REWIRE (Section 4.3.1) is responsible for the enrolment of the device into the domain and the construction of all cryptographic primitives and keys needed for the security enablers of REWIRE. Then, the Policy Orchestrator is

responsible for communicating with the Facility Layer of the device governing the execution of the actions defined by the policies. These may include **attestation enablers** (Section 4.3.3), such as the Configuration Integrity Verification (CIV) scheme for verifying the correctness of the configuration state of the device based on key restriction usage policies. They may also include the execution of a **software update process** (Section 4.3.2) for installing a patch to mitigate identified vulnerabilities, and the live migration process for moving the execution of a process on an enclave of a compromised device to an enclave of a different device. These operations are supported by the tracing capabilities of REWIRE, which are responsible for collecting the required trustworthiness evidence. This evidence may also be utilized by the **AI-based Misbehaviour Detection Engine (AIMBDE)** (Section 4.3.4) for the identification of threats following a failed attestation

Challenges to Verification Process	
Title	Modeling issues
Challenge	Formal verification requires a formal model of the system under analysis. If the model does not accurately reflect the actual system, due to assumptions, modelling simplifications, or errors, verification results might overlook vulnerabilities that are exploitable by attackers.
Approach	An iterative process could be adopted for model development, starting from a more abstract model to then refine it, incorporating details. The use of an executable model would be beneficial, enabling validation by means e.g. of simulation of various scenarios.
Title	Specifications issues
Challenge	Formal verification analyses a system implementation for compliance with its specification. If the specification incorrectly or incompletely describes the system requirements, security flaws might escape the analysis in the first place.
Approach	A thorough collaboration among all relevant stakeholders, from security experts to end users, could decrease the chance of overlooking critical requirements. A preliminary application of formal analysis, to a formalized specification and expected system properties, could help identifying discrepancies with the system requirements.
Title	Toolchain issues
Challenge	Formal verification is carried out by specialized tools that, being pieces of software themselves, are subject to bugs and vulnerabilities. An attacker, that gains access to a machine hosting the analysis, might tamper with the code, changing the tools behavior and compromising the verification process. In a setting, where verification is a step of an automated flow, attackers might also attempt to introduce malicious inputs to sway the analysis, or to modify the environment in which the analysis is performed. If evidence of correctness or validity is generated e.g. in form of an independently checkable proof, the proof generation engine or the checking mechanism could be altered, causing insecure systems to be recognized as secure.
Approach	Measures could be enforced to make sure the formal verification tools are run in a controlled and secure environment, also regulating access controls. Syntactic and semantic input validation methods could be envisioned, to prevent malicious inputs from affecting the analysis. Formally verified and monitored implementations of verification tools themselves could be employed, to guarantee correct behavior.

Title	Scope limitation
Challenge	Formal verification typically focuses on logical correctness and security properties. Side-channel vulnerabilities, that might leak sensitive information, have been a less common research subject; this is also due to the physical aspects involved, such as propagation delays, power consumption, electromagnetic radiations, which are difficult to model accurately.
Approach	Formal verification could be complemented with a simulation framework that, taking advantage of statistical models for side-channel parameters, could effectively combine logical and physical domain aspects.
Title	Theoretical issues
Challenge	The complexity of the models and of the properties to be proven might make problems undecidable or semi-decidable; the verification process might not be guaranteed to terminate, and, even if it is, it might cause false positives (spurious Property violations) or false negatives (wrong claims of validity).
Approach	Modeling languages and verification methods could be tailored to specific types of systems and properties: the use of a less expressive and more constrained language could in fact turn an undecidable verification problem into a decidable one.
Title	Scalability issues
Challenge	Formal verification techniques, e.g. model checking, face the "state space explosion" problem, where the number of model states to be analyzed grows exponentially with the size of the system; the analysis becomes intractable in practice, leading to an incomplete verification, or forcing the use of approximations or heuristics.
Approach	Methods such as abstraction and modular verification could be beneficial against the state space explosion problem: on one side, the model is simplified while retaining the characteristics that are relevant to the property being proved; on the other side, the verification problem is more effectively tackled by reducing it to a combination a sub-problems, whenever it is possible to decompose the model into sub-components and the property into sub-properties, to then merge the partial verification results.

Table 4.3: Challenges to Verification Process

4.3.1 Zero-Touch Onboarding (ZTO)

As mentioned in D2.1 [21], the zero-touch onboarding (ZTO) process consists of two phases. One is **verifiable credential issuance**, in which the device will authenticate with the privacy CA and retrieve a set of attribute keys and a verifiable credential over its attributes. The other one is the **domain enrolment**, in which the device request to enrol to a domain. In the following table, we list possible threats in ZTO process. In cases characterized by privacy requirements, we make use of a DAA scheme to achieve the verifiable credential issuance. The security of a DAA scheme is mainly captured by the unforgeability and anonymity of a device's identity and attributes. The unforgeability means that an invalid device, who does not possess a valid identity key, cannot get the credential from the privacy CA. In this process, the device needs to prove the identity to the privacy CA using a zero-knowledge proof technique. Due to the property of simulation-soundness (or the simulation-sound extractability), anyone except the valid device cannot get to know the identity, which means the invalid device, who does not possess the identity key, cannot

generate a valid proof to request a verifiable credential from the privacy CA. Due to the zero-knowledge property of a zero-knowledge proof technique, the anonymity of a valid device's identity can be ensured.

After the credential issuance, any outside attacker who can get the verifiable credential from the privacy CA, without knowing the identity key or attributes, cannot generate a valid verifiable presentation associated with the verifiable credential due to the unforgeability of the signatures (e.g., can be binded to BBS or CL signatures). The anonymity of the device's identity and attributes can also be ensured by the zero-knowledge of the zero-knowledge proof technique used in the BBS signature.

In Table 4.4, we provide a comprehensive list of attacks and threats against the ZTO process.

Threats on ZTO	
Title	Single-point-of-failure
Description	One core aspect of the ZTO process is the issuance of Verifiable Credentials (VCs) comprising the attributes that can be used for characterizing the trust level of the device. In this regard, the credential issuance, key revocation are controlled by one single entity. It follows that, if the entity is corrupted, it will lead to the single-point-of failure problem.
Property violation	Integrity, key binding, anonymity, robustness
Security Measures	Let multiple entities take responsibility for key update, revocation and review together. Or use more secure methods to realize the function of credential issuance and revocation. The issuer is also equipped with a TEE key that enables to attesting the correctness of the credential.
Title	Impersonation
Description	If a malicious entity aims to obtain illicit access to data stored on the Blockchain Infrastructure, it may attempt to impersonate a legitimate device that belongs to the target domain. Thus, an adversary may pretend to be legitimate user that possesses the required attributes to get valid credentials and onboarding successfully, so that it can present a valid credential in the context of an attribute-based scheme (DAA, ABS, ABSC, ABE).
Property violation	Authentication, authorization, confidentiality, anonymity, unforgeability
Security Measures	Ensure the security of communication to prevent the adversary from gaining valid information for pretending to be a legitimate user. Bind the device's certificates with their public/private/identity keys and corresponding domain ID; Zero-knowledge proof can be used in important steps during communication to protect sensitive personal information.
Title	Forgery attack
Description	When using a signature-based scheme of REWIRE, such as the ABSC scheme for verifiably proving the correctness of the device attributes, a malicious entity may attempt to create a signature that appears legitimate. Thus, an adversary tries to forge a valid signature assigned to a legitimate user without knowing the corresponding signing key, in order to obtain illicit access to a service or piece of information.
Property violation	Unforgeability, integrity, anonymity, authentication, authorization, linkability

Security Measures	The signature algorithm (DAA, ABS, ABSC) achieves the property of unforgeability through the use of hardware-based guarantees for proving ownership of the specified attributes, and the zero-knowledge proof techniques are simulation-sound extractable.
Title	Quantum attacks
Description	Considering the recent research advancements in the field of quantum computing, solving the underlying mathematical problems based on which most classical cryptographic schemes are based on will become trivial, as the computational hardness assumptions will not be valid. Thus, if quantum computers comes into being and become commercially available, the existing cryptographic algorithms based on traditional mathematical hard problems will be broken (DAA, ABS, ABSC, ABE). This necessitates the development of equivalent, quantum-resistant schemes.
Property violation	Authentication, authorization, integrity, unforgeability, anonymity, confidentiality, linkability
Security Measures	Replace the existing scheme (DAA, ABS, ABSC, ABE) based on traditional mathematical hard problems with any post-quantum schemes, such as based on lattice, hash functions, code-based schemes, etc. Utilize the NIST-approved quantum-resistant schemes for replacing existing ones (e.g., ML-KEM, ML-DSA, FN-DSA), with the appropriate architectural adjustments.

Table 4.4: Threats on ZTO

4.3.2 Operational and Side-Channel Attacks (SCA) Against the SW Update Process

In general, **Side-Channel Attacks (SCA)** aim to exploit physical characteristics of the channel over which communication is performed, or the device itself, or they may target the fundamental implementation characteristics of a protocol or algorithm. In the context of secure updates, SCA can be used for compromising the update process, or for a malicious party to install an illegitimate update.

Throughout this section, we provide information on the types of attacks that may affect the software update process. Specifically, in Table 4.5, we provide physical attacks and SCA that target the integrity of the SW update process. In Table 4.6, we provide attacks pertaining to the state management of the enclaves, which may affect the software update process or the live migration process.

Attacks on the integrity of the SW update process	
Title	Software/firmware update key extraction
Description	It is possible that a malicious entity may aim to compromise the secure software update process, for instance to obtain illicit control of the device, or tamper with the update patch in order to perform a malicious update. Thus, the adversary could try recovering the update key by observing and/or tampering with the SW/FW update messages exchanged with the embedded device during an update process. Note that these are the messages exchanged within the device, and not data in transit (which are protected through the cryptographic schemes offered by REWIRE).
Property violation	Confidentiality, integrity, authentication

Security Measures	Both the symmetric and asymmetric cryptographic schemes are based on state of the art processes that offer hardware-based security guarantees for ensuring the security of the utilized keys pertaining to the SW update process, and ensure resistance against chosen message attack, hence preventing this threat.
Title	Software/firmware update key extraction through a side-channel attack
Description	In order to compromise the security update process, such as injecting the update patch with a malicious piece of code, an adversary could try recovering the update key by performing a side-channel attack against the embedded device during an update process. Such attacks aim to exploit the physical attributes of the device or the communication channel.
Property violation	Integrity, authentication
Security Measures	The symmetric cryptography algorithm (namely LRBC2) offers protection against side-channel attacks trying to recover the long-term key or forging a fresh message pretending to be authentic. The asymmetric cryptography algorithm (signature) does not manipulate the private key at all on embedded device's side, hence totally preventing the attack.
Title	Attack on the software/firmware update *confidentiality*
Description	The software update mechanism as defined ensures both confidentiality and integrity against a classical adversary, as well as integrity, but not confidentiality, against a side-channel adversary. It is therefore possible for an adversary with side-channel capabilities to threaten the confidentiality of the software update (i.e., discover the content of the update upon which a side-channel attack is being applied, but not that of subsequent updates).
Property violation	Confidentiality.
Security Measures	A stronger cryptographic primitive could be used instead of LRBC. However, ensuring confidentiality in the presence of side-channels is a difficult task, and the overhead is expected to be very high. Considering the limited extent of the threat, leaving it as such is probably a better option.
Title	Fault induction attack
Description	Active physical attacks are those where deliberate manipulations of the physical device are performed in order to induce errors during its normal operation, especially during crytpographic operations. With these attacks, the adversary induces processing errors, could be used to recover long-term keys or forge messages.
Property violation	Confidentiality, authentication, integrity
Security Measures	Stronger cryptographic algorithms would be needed to defend against this additional threat. In the current state of the art, efficient achievement of this goal is still an open question. Despite the fact the fact that LR-BC2 is not designed for protection against fault attacks, it is likely that it is hardened against it due to its structure. This is not a formal claim and its fault resilience also remain an open question.

Title	Quantum attacks
Description	It is conjectured that a new generation of computers, based on quantum computing, could revolutionise the global field or cryptography, by being able to break many of the existing algorithms. As aforementioned, a quantum computer will be able to trivially solve the underlying mathematical problems of classical cryptographic algorithms, thus necessitating the adoption of quantum-resistant cryptographic schemes, which are not subject to computational hardness assumptions. Attacks against quantum schemes may include **timing attacks** which exploit variations in the time it takes a system to execute operations, **fault injection attacks (FIA)** for disrupting normal operation and extracting secrets or bypassing security measurements, and **Side-Channel Attacks(SCA)** which are based on physical characteristics of the system (e.g., power consumption or electromagnetic leaks).
Property violation	Confidentiality, integrity, authentication
Security Measures	Current algorithms should be replaced by quantum-proof ones (currently still in design/standardisation phase). It is expected that symmetric algorithms (among which LRBC) will be more resistant to quantum attacks, and could be corrected by a mere doubling of their keys and internal state sizes. On the other hand, asymmetric algorithms, such as the digital signature used in the one-to-many use case, would have to be replaced by totally different schemes.
Title	Bug in the code
Description	An unexpected programming bug in the code of LRBC or some underlying library could introduce unexpected weaknesses.
Property violation	Confidentiality, integrity, authentication
Security Measures	A reactive approach must be adopted, whereby bug discovery triggers a swift patch deployment for future equipment. In the most damaging cases, replacing existing equipment might prove necessary. To this end, REWIRE offers formal verification capabilities for the functional correctness of such operations within the Compositional Verification and Validation component, in order to mitigate such bugs.

Table 4.5: Physical attacks against the integrity of the SW update process.

Threats on state management of the enclaves and apps to be updated and/or migrated	
Title	Unauthorized update/migration trigger
Description	Unless desiring an auto-update model, each SW update and migration must be authorized by the device owner. However, in this case, it is possible for a malicious entity to attempt to initiate an update process, in order to install a malicious update or to obtain access to the device itself. If an untrusted entity successfully triggers an update on a device, the device owner is not in control of the device state.
Property violation	Authentication.

Security Measures	The REWIRE SW update protocol verifies that each update is signed by the device-owning entity and provides hardware-based guarantees for ensuring that the device is the one that it claims to be. A similar mechanism is planned for the REWIRE SW migration protocol.
Title	State manipulation
Description	As part of the REWIRE SW update and migration protocols, it is possible to optionally transfer the enclave state from the source device to the target enclave. An adversary might recover or/and manipulate the state during transmission to gain information about the enclave state or influence the enclave's execution.
Property violation	State confidentiality and integrity.
Security Measures	The REWIRE SW update protocol enforces the encryption of the state with a transport key during transmission to ensure confidentiality. Further, a MAC is attached for integrity. The REWIRE SW migration protocol will include a similar mechanism to ensure confidentiality and integrity.
Title	Migration to an insecure target
Description	Each enclave comes with a manifest defining security properties that a host device must meet. In general, a manifest file is an application-specific configuration text file that specifies the environment and resources for running an application inside an enclave. If an adversary successfully initiates a migration process to a target device that does not meet the security requirements, the security of the enclave application may be violated, leading to further vulnerabilities.
Property violation	Secure migration of a TEE from Source to Target.
Security Measures	The REWIRE SW migration protocol could enforce an attestation of the target device via the REWIRE enhanced CIV attestation to ensure that the target device meets the security requirements of an enclave before proceeding with the migration. Specifically, the CIV scheme is able to verify if the properties specified in the manifest file are present and correct in the target device.

Table 4.6: Threats against the state management of enclaves in the the SW update or live migration process

4.3.3 Attestation

Attestation schemes can be used for a Prover device to prove to a Verifier about the correctness of one or more trustworthiness attributes, to prove the correctness of its configuration state, or to attest to the correctness of the execution of a process. However, it is possible for a malicious party to compromise an attestation process, for example by causing false positive attestation results. Throughout this section, we provide details on such types of attacks.

Threats on Attestation	
Title	Spoofing of attestation result

Description	A software adversary may attempt, on a Prover device performing an attestation process, to exploit a software vulnerability that allows it to modify that node's critical configurations that can be attested by the CIV enabler. Thus, the adversary may attempt to affect the result of the attestation leading to an incorrect outcome (e.g., a false positive result in case the configuration has been compromised).
Property violation	Measurement authenticity, integrity
Security Measures	By predicating all policies for a device on the requirement of explicit TEE authorization for the extracted measurements, adversaries are prevented from supplying inauthentic measurements to unlock the key restriction usage policy. Thus, spoofing is prevented, and the detection of unauthorized configuration modifications is ensured.
Title	Policy Misconfiguration
Description	It is possible for an adversary to have access to the required information to fulfil the local attestation policy on a device, e.g., by causing an erroneous attestation result. Thus, an adversary may attempt to validate multiple policies at the same host by exploiting policy session management vulnerabilities. A malicious entity may also attempt to utilize a previous or deprecated policy to perform the attestation process, which does not apply to the current configuration of the device. This, it is possible that the adversary compromises the attestation process by fulfilling a policy predicated on an incorrect device configuration.
Property violation	Policy authenticity, integrity, authorization, freshness
Security Measures	By initially embedding the identifier of a device in its local attestation key's authentication policy, we ensure that policies approved for a node's local AK must bear that node's identifier to be satisfiable, thus preventing any adversary from using policies meant for one node to unlock the AK on a different node. Regarding threats against policy fresnhess, by predicating all policies approved for a node's local AK on additional authorization with explicit mention of that node and the approved policy it applies to (through the configuration identifier) and having the orchestrator include expiration on such authorizations, we secure a leasing mechanism to prevent nodes from satisfying approved policies beyond their intended timeframes, thus preventing adversaries from satisfying policies approved for old configuration states in an attempt to evade the remeasurement process of the node's current configuration.
Title	Implementation disclosure attack
Description	It is possible that a dishonest verifiers may attempt to infer information about the Prover device's configuration. This is referred to as an implementation disclosure attack, and aims to obtain further information about the inner workings of a device, in order to uncover other potential vulnerabilities that may be used to launch future attacks.
Property violation	Privacy, confidentiality, zero-knowledge verification

Security Measures	The CIV scheme of REWIRE is based on local attestation leveraging key restriction usage policies, meaning that the the use of the Attestation Key is permitted only in case the policy is fulfilled. Thus, there is no need to send any information about the device configuration to the Verifier, as the construction of the signature is proof itself that the device is in a correct state, thus preventing implementation disclosure attacks.
Title	Control-flow attacks
Description	It is possible for a strong software adversary, on the prover, to exploits a severe software vulnerability to mount control-flow attacks to divert the attested program's execution. Thus, a dishonest entity may attempt to convince the Verifier that an illegal (forged) attestation path (e.g., one that reveals an attack against the attested program) is legal. For instance, Return-Oriented Programming (ROP) or Jump-Oriented Programming (JOP) attacks may be used for constructing exploits.
Property violation	Unforgeability
Security Measures	It is possible to utilize a Control-Flow Attestation (CFA) scheme utilizing zero-knowledge signatures in order to precent such attacks. Specifically, these schemes may require jumps and calls to target valid neighbors, and returns to target a contextually valid node. If these conditions are not fulfilled, the presence of such attacks is detected.

Table 4.7: Threats on Attestation

4.3.4 AI-based Misbehaviour Detection Engine

Up until now, all threats we considered were inherent to the operation of the device itself. However, it is possible that an attacker may also target the application of the device, i.e., **application data**, which may be the target of the attack. In this context, the AI-based Misbehaviour Detection Engine (AIMDE) of REWIRE has a critical role in identifying threats and anomalies in devices' operation, and functions as an additional **Trust Source** that may be used as input in a trust assessment methodology. However, their heavy reliance on input data makes them susceptible to threats/attacks like data poisoning and evasion, which can undermine their accuracy, integrity, and robustness. Therefore, it is crucial to identify and address these threats to ensure such systems reliability and secure operation. This also necessitates the consideration of approaches based on **robust and trustworthy AI**, which possess the required guarantees in order to create trust and confidence that the defined security properties are fulfilled.

Threats on AI-based Misbehavior Detection Engine	
Title	Data Poisoning Attacks
Description	It is possible that a malicious actor may aim to compromise the model of the AI-based Misbehaviour Detection engine, in order to steer the model behaviour in an adversarial manner, or even to implamt a backdoor in the model to be triggered by specific input patterns. In this regard, data poisoning attacks are a type of adversarial attack where an attacker deliberately inserts corrupted, false, misleading samples into the training datasets towards introducing biases and reduce model's performance.

Property violation	System's Integrity and Robustness
Security Measures	To prevent such attacks, it is possible to perform data validation using robust training methods and monitor data provenance. In addition, it is possible to perform data validation and sanitization for using clean, trusted, and verified data sources, or implementing a degree of supervision over the training process.
Title	Evasion Attacks
Description	Such attacks aim to make a model into making incorrect predictions, by generating small intentional perturbations (changes) to the input data used for training of the model. Thus, the attacker may attempt to influence the model in order to provide the desired (malicious) output.
Property violation	System's Integrity and Robustness
Security Measures	It is possible to prevent evasion attacks by using adversarial training (retraining the model using adversarial examples), pre-processing for washing out adversarial noise, data cleaning of input data, and useing explainability models to validate the results.
Title	Resources Exhaustion
Description	These types of attacks aim to overload the system in order to reduce its performance or cause denial of service. Thus, by targeting the training or inference phases, it is possible to make the model slow, unresponsive, or unusable.
Property violation	System's Robustness
Security Measures	It is possible to prevent resource exhaustion attacks by limiting the number of requests, using anomaly detection systems, and implementing efficient resource management strategies. In addition, it is possible to perform input size validation by rejecting inputs that exceed expected size, complexity, or computation cost.

Table 4.8: Threats on AI-based Misbehaviour Detection Engine

4.3.5 Blockchain

The **Blockchain Infrastructure** is a core component of REWIRE, responsible for application- and attestation-related data management. Also, recall that a core innovation of REWIRE is the **Secure Oracle Layer**, for ensuring the veracity of the data to be recorded on the Blockchain. While Blockchain infrastructures offer strong security guarantees, it is possible for malicious entities to launch attacks on such infrastructures, for example to compromise data, or obtain unauthorized access to stored data, despite not possessing the required attributes.

Throughout this section, we provide further information on attacks that may target Blockchain infrastructures. Note that, in this Section, we consider the Secure Oracle Layer as a **blackbox**, and we do not consider its internal operations that can be targeted. However, because of the importance of such attacks that can target internal operations, in Chapter 5 we expand upon the threat model and take a closer look into the underpinnings of the infrastructure, and we provide detailed information on the vulnerabilities regarding its internal operations.

Threats on Fabric Private Chaincode	
Title	MSP Manipulation
Description	Malicious manipulation of the Membership Service Provider (MSP) configuration involves tampering with identity and membership settings in a blockchain network. This can allow attackers to spoof identities or gain unauthorized access by altering roles or policies. Such actions compromise network trust, enabling false endorsements, unauthorized transactions, or disruption of consensus.
Property violation	**Authentication** – Unauthorized access due to tampered identity.
Security Measures	- Robust MSP configuration validation - Identity management and audits - Access control policies for MSP changes
Title	Smart Contract Exploitation & Data Veracity
Description	Exploiting vulnerabilities in chaincode involves identifying and taking advantage of flaws in the smart contract logic deployed on a blockchain network. Attackers may manipulate the chaincode to bypass access controls, alter business logic, or trigger unintended behaviors that benefit them. In more severe cases, these vulnerabilities can be used to extract sensitive or private data stored on the ledger or passed through transactions. Such exploits can compromise data integrity, leak confidential information, and disrupt the reliability of the blockchain application if not properly secured through thorough testing and auditing.
Property violation	**Integrity** – Unauthorized modification or exploitation of smart contract logic.
Security Measures	- Code audits - Formal verification of chaincode logic - Bug bounty programs - Automated testing pipelines
Title	Data Veracity Attacks
Description	Manipulating or falsifying data within Fabric Private Chaincode (FPC)'s private collections involves unauthorized modification of confidential information that is meant to be securely shared only among a defined subset of organizations. An attacker who gains access or privileges, either through misconfiguration or exploitation, can alter or inject false data into these collections, undermining the trust and accuracy of off-chain or sensitive business processes. Such tampering can lead to incorrect decision-making, fraudulent transactions, or violations of compliance requirements, especially in systems that rely heavily on the integrity of private data.
Property violation	**Integrity** – Falsification of private transaction data.
Security Measures	- Transaction validation - Zero-knowledge proofs (ZKPs) for verifying data without revealing it - Tamper-evident logging
Title	Man-in-the-Middle (MITM)

Description	Interception and potential alteration of communications between nodes in Fabric Private Chaincode (FPC) may compromise the confidentiality and integrity of data exchanged within the network. If an attacker is able to eavesdrop on or tamper with messages transmitted between enclaves or peer nodes, especially during transaction endorsement or data exchange, they could manipulate sensitive information, replay messages, or inject malicious data. This could compromise secure computation, leak private data, or disrupt consensus. Ensuring secure communication channels, such as using end-to-end encryption and mutual authentication, is critical to defending against such attacks.
Property violation	**Confidentiality** – Exposure of sensitive data during transmission.
Security Measures	- End-to-end encryption using TLS/SSL - Key exchange protocols - Channel encryption
Title	Private Data Exposure
Description	Unauthorized access to sensitive private data in private data collections within Fabric Private Chaincode (FPC) represents a significant security risk, as these collections are designed to store confidential information. If an attacker gains improper access—whether through exploiting weaknesses in access control mechanisms, misconfigurations, or privilege escalation, they can retrieve or manipulate private data, which could include financial records, personal information, or proprietary business processes. Such breaches not only undermine the integrity of the system but also expose critical data to misuse.
Property violation	**Confidentiality** – Exposure of private data.
Security Measures	- Access control policies - Data partitioning - Auditing access to private data - Secure enclave (e.g., Intel SGX) for private data processing
Title	Chaincode Vulnerabilities
Description	Bugs or flaws in chaincode can create significant vulnerabilities that attackers can exploit to gain unauthorized access or manipulate the system's logic. These issues may arise from coding errors, poor input validation, or overlooked edge cases in the contract's functionality, allowing attackers to bypass security checks, modify data, or trigger unintended behaviors. Such flaws can lead to data breaches, where sensitive information is exposed or altered, or financial losses, where transactions are fraudulently executed. In addition, flaws in the chaincode logic may undermine the integrity of the blockchain, disrupting trust between participants.
Property violation	**Integrity** – Exploitation of chaincode vulnerabilities.
Security Measures	- Regular code audits and testing - Formal verification - Security-focused development lifecycle
Title	Re-Entrancy Attacks

Description	Exploiting chaincode vulnerabilities that allow re-entrant calls can lead to significant security risks in blockchain applications. A re-entrant call occurs when a smart contract or chaincode invokes itself recursively, often in an unexpected or uncontrolled manner. This can be particularly dangerous if the chaincode doesn't properly handle state changes or transaction sequencing, leading to unexpected behaviors such as double-spending, incorrect data updates, or manipulation of transaction results. Attackers can exploit such vulnerabilities to override critical operations, bypass security mechanisms, or obtain illicit access to private data.
Property violation	**Robustness** – Execution vulnerabilities that lead to inconsistencies.
Security Measures	- Re-entrancy protection within chaincode logic - Formal verification of execution flow - Static and dynamic analysis tools
Title	Denial of Service (DoS)
Description	Overloading the Fabric Private Chaincode (FPC) network or smart contract can cause severe disruptions by overwhelming the system's resources, leading to slowdowns, crashes, or denial-of-service (DoS) conditions. Attackers may flood the network with excessive requests, create computationally expensive transactions, or exploit inefficient smart contract logic to consume disproportionate amounts of processing power, memory, or bandwidth. This can degrade the performance of the blockchain network, making it unresponsive to legitimate users or causing delays in transaction processing.
Property violation	**Availability** – Disruption of service due to resource exhaustion.
Security Measures	- Load balancing and rate limiting - Peer redundancy - Resource monitoring and dynamic scaling
Title	Key Compromise
Description	Compromise of private keys in Fabric Private Chaincode (FPC) represents a critical security threat, as it may grant unauthorized acces to chaincode or private data to malicious parties. If an attacker gains access to a private key, they can impersonate legitimate participants in the network, enabling them to invoke chaincode, alter transaction outcomes, or retrieve confidential data without authorization. This breach can lead to data manipulation, unauthorized transactions, or even full control over the blockchain's operations, depending on the role of the compromised key.
Property violation	**Authentication** – Loss of key security, allowing unauthorized transactions or data access.
Security Measures	- Cryptographic consensus algorithms - Network diversity - Peer verification of transaction logs
Title	Revealing Private Information in Private Data Collections

Description	Unauthorized decryption or inference of private information from the private data collections in Fabric Private Chaincode (FPC) poses a significant threat to the confidentiality of sensitive data stored within the blockchain. Private data collections are designed to ensure that certain data is only accessible to authorized participants, but if an attacker gains access to the encryption keys or exploits vulnerabilities in the system, they could decrypt or infer such confidential information, thus leading to the exposure of sensitive data.
Property violation	**Confidentiality** – Exposure of sensitive information in FPC's privacy-preserving features.
Security Measures	- Access control lists (ACLs) for data sharing - Zero-knowledge proofs (ZKPs) for verifying transaction information without revealing it
Title	Replay Attacks
Description	Replaying valid transactions on the network can lead to significant security issues, when the transactions are replayed with malicious intent or to manipulate outcomes. In a replay attack, an attacker captures and retransmits a legitimate transaction to the network, causing it to be processed again. This can result in unintended consequences such as double-spending, repeated data updates, or the execution of contracts under false pretenses.
Property violation	**Integrity** – Invalidating transactions by re-executing them.
Security Measures	- Timestamping and nonce validation - Transaction IDs with replay protection - Double-spending detection

Table 4.9: Threats on Fabric Private Chaincode

Threats on Besu Blockchain and Secure Oracle	
Title	Consensus Manipulation Attack
Description	An adversary attempts to manipulate the consensus mechanism by compromising one or more validators in the IBFT 2.0 network. The attacker may submit malicious proposals or disrupt the voting process, leading to potential forks or invalid blocks being added to the blockchain. In the context of Besu, this can exploit validator misbehavior or insufficient monitoring.
Property violation	**Integrity**, **Robustness**, and **Consensus Finality**. The attack could violate the integrity of the blockchain by introducing invalid blocks or disrupting the consensus, preventing the network from reaching agreement.
Security Measures	- Strong validator authentication mechanisms using PKI-based identities to ensure only authorized nodes participate in consensus. - Continuous monitoring and logging of validator behaviour to detect and respond to anomalies quickly. - Employ a distributed set of validators to avoid regional failures or collusion. - Regularly update and audit validator software to address known vulnerabilities and improve resilience.
Title	Single Point of Failure in Town Crier Oracle

Description	A single Town Crier oracle node is responsible for fetching and verifying external data. If this node fails or is compromised, the system's ability to provide accurate and timely attestation is disrupted, leading to potential delays or propagation of false data to the blockchain.
Property violation	**Availability**, **Integrity**, and **Robustness**. This failure can disrupt the application's ability to function, introduce incorrect data, or halt processing.
Security Measures	- Deploy multiple redundant Town Crier nodes to ensure data availability even if one node fails. - Use a distributed oracle architecture to aggregate and cross-verify data from multiple independent sources. - Encrypt and sign all data fetched by the oracle to ensure tamper-proof data transmission. - Periodically audit oracle behavior to ensure compliance with system requirements.

Table 4.10: Threats on Blockchain

4.4 Trustworthiness Evidence Mapping for Proof-of-Execution of Required Security Controls

Having detailed the endmost goal and vision of REWIRE throughout the previous sections, including the security measures and controls considered as part of the REWIRE framework, REWIRE is the first project of its kind to provide a mapping of the types of evidence which can be monitored during runtime, and enable us to verify and attest to the trustworthiness level of a device against the **Required Trust Level (RTL)** which may have been identified. This mapping of evidence is categorized as (i) **static evidence** representing the persistent state of a device (e.g., secure boot, presence of a valid TCB) (ii) **runtime evidence** representing the volatile state of a device (e.g., control flow graph execution, list of active processes), and (iii) **crypto evidence**, representing evidence pertaining to the correctness of the underlying crypto structures. This categorization of evidence and corresponding attacks is provided in Table 4.11.

Evidence or Asset	Description	Type	Motivation	Threat or Violation
Static Evidence				
secure or measured boot	"status of whole platform (e.g., server with vRouters, or hardware router); enabled: y/n? measurement values?"	boot-time integrity	detect compromise of platform code/TCB (BIOS, OS, HV...)	kernel code compromise
XRd container	hash + signature of container image	container (boot-time) integrity	detect compromise of the XRd container image / configuration	XRd container image/configuration violation
vendor + version of router	e.g., "Cisco XRd vRouter, version 24.4.1". Could include additional version information, e.g., of the (N)OS, kernel, or certain software packages	boot-/load-time with runtime checks	periodically match against CVE database to identify new known vulnerabilities	Impersonation, property violation

network interfaces	list of network interfaces (in each namespace!), their metadata (e.g., link status, MAC/IP address, subnetwork, type), maybe their network statistics (e.g., num. of in/out/drop packets)	runtime	detect misconfigurations that might tamper with routing (e.g., address spoofing), interface DoS/downtime, suspicious virtual interfaces (e.g., TAP/TUN) usable for local sniffing, DoS attack detection (e.g., high drop/error rate, unbalanced in-to-out packet rate)	packet drop, forwarding DoS, packet redirection via header manipulation, packet content manipulation, packet sniffing, address spoofing, address hijacking (e.g., BGP hijacking)
open connections / sockets	list of network connections and open sockets (e.g., listening) and their metadata / types (e.g., UDP, raw)	runtime	detect suspicious SSH/-management connections (infiltration), raw sockets (packet sniffing/tampering), other unexpected in/out traffic	packet drop, forwarding DoS, packet redirection via header manipulation, packet content manipulation, packet sniffing, address spoofing, address hijacking
firewall / nat tables	cf. routing table	runtime	detect changes / check entries; e.g.: packet redirection, or DoS	packet drop, packet redirection via header manipulation, address spoofing, address hijacking
packet processing	if router actually uses the above network tables (firewall, nat, fib) for packet processing	runtime	detect if packets are wrongly processes / forwarded / manipulated; e.g., packet manipulation, redirection	packet content manipulation, packet sniffing
network-related configuration files	e.g., SSH server config, BGP, MPLS, tunnels (IPsec/MACsec? VLAN?), maybe DNS	load/runtime	detect vulnerabilities due to misconfiguration, e.g., insecure SSH/BGP configs, disabled or insecurely configured tunnel protocols (e.g., insecure algorithms, static keys, short key length), DNS rerouting/misresolution	packet redirection via header manipulation, packet content manipulation, packet sniffing, address spoofing, address hijacking
cryptographic keys / certificates	CA certificates, service certificates (SSH, IPsec, ...), platform identity certificates (e.g., DICE), and associated cryptgraphic keys	load/runtime	check for: rogue CAs, revoked or deprecated certificates, insecure key length / algorithms (e.g., for SSH or IPsec), unknown SSH keys for remote access, etc.	downgrade of security protocols, impersonation, trace data tampering/spoofing, Single-point-of failure
Dynamic Evidence				
process measurements	measurement of all loaded processes, maybe including CLI arguments. simple: hash of binaries, complex: hash of virtual memory space (e.g., static code/data pages)	load time integrity	detect corrupted or unknown programs (e.g., malware)	trace data tampering / spoofing, remote compromise of edge nodes, user malware

active processes	list of active processes and their metadata (e.g., paused, num. of threads, parent ID), maybe system telemetry (e.g., CPU and memory usage)	runtime	detect unknown processes or threads, or DoS against benign processes (e.g., killed process, unexpected CPU/memory util)	remote compromise of edge nodes, user malware
routing table (RIB, LIB)	routing tables (protocol-specific RIBs, LIB of MPLS), changes/access attempts to them	runtime	detect changes to routing table (might affect trusted path), identify caller context of table change, e.g.: packet redirection, or DoS	packet drop, packet redirection via header manipulation, address spoofing, address hijacking
forwarding table (FIB)	routing tables (protocol-specific RIBs, LIB of MPLS), changes/access attempts to them	runtime	detect changes to routing table (might affect trusted path), identify caller context of table change, e.g.: packet redirection, or DoS	packet drop, packet redirection via header manipulation, address spoofing, address hijacking
additional system / network telemetry	e.g., NIC interface hardware counters (packet statistics), end-to-end latency/bandwidth with neighbouring routers, CPU performance counters	runtime	FSM-/statistic-based anomaly detection	packet drop, forwarding DoS, packet redirection via header manipulation, address spoofing, address hijacking
open files / files on disk	open file descriptors, critical files on disk (e.g., binaries and shared libraries)	load/runtime	Check for tampering with system libraries? Anomaly detection on file I/O behaviour (using an FSM)?	downgrade of security protocols, device identity spoofing, trace data tampering / spoofing, Single-point-of failure
system calls	system call tracing of a small set of important processes, e.g., rib/fib/if managers, main DPDK-facing process	load/runtime	anomaly detection using a FSM capturing the ”normal” syscall behaviour?	downgrade of security protocols, trace data tampering / spoofing, Single-point-of failure , kernel code compromise
loaded kernel modules	list of active (i.e., loaded) kernel modules	load/runtime	detect unknown or corrupted kernel modules that can tamper with router; can prioritize networking-related kernel modules (e.g., NIC driver, netfilter / iptable, DPDK ”vfio_pci”?)	kernel code compromise
Multiple privacy CAs	The credential issuance and revocation are controlled by one single entity, i.e., the privacy CA. If the privacy CA is corrupted, the single-point-of failure problem happens. Multiple CAs can solve this single-point-of failure problem	integrity, key binding, anonymity, robustness	try to prevent the single-point-of-failure problem	Single-point-of-failure

A secure channel and NIZK during Verifiable credential issuance (Impersonation)	An adversary try to get a valid credential from the issuer without proving the possession of valid identity keys and attributes keys to the issuer or by getting a valid user's private keys during communications. Therefore, a secure channel and NIZK proof are necessary	authentication, authorization, anonymity	To pretend to be a legitimate user	Spoofing, impersonation
A secure channel, NIZK and secure signature algorithm for Enrolment (impersonation)	An adversary try to forge a valid signature (verifiable presentation) without a valid identity key and attribute keys. A secure channel, NIZK and secure signature algorithm for Enrollment are necessary	unforgeability, confidentiality	To enroll a domain to access corresponding service	Spoofing, impersonation, downgrade of security protocols
Crypto Evidence				
Post-quantum cryptographic primitives (Quantum computer)	A quantum computer comes into being, then all cryptographic primitives to protect all assets will not be secure any more	authentication, authorization, integrity, unforgeability, anonymity, confidentiality, linkability	To protect all assets from attacks launched by the quantum computer	Post-quantum attacks

Table 4.11: Mapping of types of evidence with threats or violations

Chapter 5

REWIRE Secure Oracle Variants and Security Considerations

Recall that one core innovation of REWIRE is the provision of a **Secure Oracle Layer**, which enables the privacy-preserving execution of chaincode and smart contracts within a **Trusted Execution Environment (TEE)** in order to provide **data veracity** on data to be recorded on the **Blockchain Infrastructure**. This Chapter is dedicated to the analysis of the threat landscape against the Secure Oracle Layer, as well as the security measures that can be enforced on their mitigation. Note that, while in Section 4.3.5 we treated it as a blackbox, here we also focus on its internal operations that can be targeted by malicious parties.

The policy-compliant Blockchain Infrastructure of REWIRE is instantiated over the **HyperLedger Besu** technology, paired with the SGX-based **Town Crier (TC)** as a credible and authenticated bridge for the execution of smart contracts. REWIRE also employs **Fabric Private Chaincode (FPC)**, which is responsible for the execution of chaincode within an SGX enclave. The use of Blockchain in REWIRE enables the *secure collaboration between multiple organizations or entities in zero-trust environments*, which is a core benefit considering the zero-trust nature of environments where the REWIRE framework is expected to operate. Specifically, we are able to consider environments of interoperable components supplied by multiple vendors with no single tenant or provider. Such environments are characterized by dynamic topologies with real-time constraints, which may also exhibit different levels of security, privacy, trust, and operational assurance goals. In this regard, *the Secure Oracle Layer of REWIRE provides data veracity in the enforcement and execution of data-sharing agreements in a trustworthy manner, thus protecting against attacks aiming to exploit weaknesses in the overall data management process*.

Throughout this Chapter, we provide a detailed overview of the most prominent types of threats targeting Blockchain Infrastructures, as well as the capability of the **Secure Oracle Layer** of REWIRE to enhance the security profile of all data management interactions. Note that *Secure Oracles refer to trusted systems or services that provide external data to smart contracts in a secure, reliable, verifiable, and tamper-resistant manner, thus acting as a bridge between off-chain sources and on-chain smart contracts*. For instance, the aforementioned Town Crier utilizes Software Guard Extensions (SGX) for allowing the secure execution of code in an isolated, tamper-resistant enclave. In addition, it is also possible to leverage SGX for the secure execution of smart contracts, in a manner that ensures the security and integrity of the smart contract logic and exchanged data. Thus, as will be reflected in the threat landscape described throughout this Section, *this approach provides guarantees on the integrity of the smart contract execution, as well as the data provided as input*.

5.1 Adaptive Security

As REWIRE is based on the use of permissioned ledgers, the threat landscape differs from permissionless chains, in the sense that various types of attacks (e.g., network partitioning attacks) are not as

significant a threat on permissioned networks because users and device IDs are known, their activities are monitored, and access is managed by access control policies and data sharing agreements. Thus, the most impactful attacks in the context of REWIRE are (i) attacks caused by the integration of **weak protocols**, (ii) **malicious acts** that may arise through the interaction of external entities with the Blockchain, and (iii) **application bugs** that try to exploit possible vulnerabilities in the smart contract chaincode. The latter category also includes (untrustworthy) external data sources that communicate with the Blockchain through a Secure Oracle, and may provide data as input to a smart contract. Note that, *in the context of REWIRE, we do not make any trust assumptions on the data sources, which is a core motivation behind the use of Secure Oracles*.

In Table 5.1, we expand upon this classification and we provide further details on each aforementioned type of attack.

Attack Type	Description
Weak Protocols	A key aspect of Blockchain infrastructures is the use of a **consensus protocol**, which is responsible for governing the transactions of data to be recorded on the ledger, and the employed consensus protocol may vary depending on the Blockchain platform adopted. While attacks against weak consensus protocols are increasingly prevalent, such threats based on the exploitation of weaknesses in such protocols can often be resource-intensive. However, successful attacks may be able to **remove blocks from the chain**, **fully destroy a Blockchain**, or **acquire full control over the stored data**. In this regard, there are usually assumptions on the capabilities of the attacker (due to the resource-intensive nature of this type of attacks) that are required for compromising and controlling the majority of miners in order to control the consensus process. However, there is also the commonly used and industry-adopted **Raft consensus mechanism**, which builds upon a leader-elected model for managing the cluster of nodes that need to validate a transaction prior to its recording on the ledger. The minimul number of nodes required to perform an operation in a Raft cluster is $(N/2 + 1)$, where N is the total number of members in the cluster. While there is extensive literature on majority-based consensus protocols, they have typically not been the focus of Raft, as such clusters are usually assumed to be trusted. However, in REWIRE, where the **zero-trust paradigm** is a core consideration, we have considered additional security/safety controls for safeguarding the consensus process, as outlined in Section 5.1.4.
Malicious Acts	Malicious acts may refer to attacks that involve **spreading malware to deceive users**, or **compromising user identity** in an attempt to conduct impersonation and obtain unauthorized access to recorded data. This category also includes **impersonation attacks**, where a malicious entity impersonates a valid user or device and provides fake attestation-related data (acting as a Verifier) in order to accuse a Prover device of having failed an attestation task. This may, in turn, lead to the isolation of the Prover, thus disrupting the operation of the entire service graph chain. Such attacks typically target the **exploitation of the mechanisms employed by the device for securely managing Verifiable Credentials** and/or digital currency. Malicious acts may also include crypt-jacking, slack, and forum attacks for causing miners to log in through corrupted links, and Glupteba is a type of malware that utilizes the Bitcoin Blockchain for its update. In Section 5.1.3, we provide further details on the protection against this type of attacks through the use of **HW-based keys**, so that **no attacker is able to steal VCs or use VCs in case of leakage**.
Application Bugs	Exploitation of application bugs refers to cases where vulnerabilities have been identified in the smart contract code. This may occur in cases where developers do not perform secure code practices and (unintentionally) introduce SW-based vulnerabilities that can be exploited by attackers through providing malicious input. The exploitation of such types of vulnerabilities may have serious adverse effects, such as monetary fraud, or impeding the ability of an application or security process to function as intended. For instance, in REWIRE, smart contracts are utilized for the secure and auditable deployment and enforcement of security policies. Thus, **compromising this type of smart contract functionality may result in backdoors and Trojans** that can be exploited by attackers for gaining access to the system and taking control of the entire smart contract functionality and decision logic. The aforementioned **secure oracles**, which are also employed by REWIRE, are one of the most prominent solutions against these types of attacks, as they are able to ensure the integrity and security of the data to be recorded on the Blockchain infrastructure.

Table 5.1: Classification of attacks against REWIRE Blockchain Infrastructure

In the following, we provide further details on prominent types of attacks and risks that fall under the categories outlined in Table 5.1, and we perform a detailed analysis on the profiles of such impactful

attack vectors and the possible mitigation measures that can protect against them. WE also highlight the security mechanisms employed by REWIRE towards safeguarding the execution of data sharing agreements (i.e. providing the necessary assurance claims that the required trustworthiness level has been achieved).

5.1.1 Denial of Service (DoS) and Distributed DoS (DDoS)

The purpose of **Denial-of-Service (DoS)** attacks is to disrupt the availability of the Blockchain network, and can be a threat to any distributed system. Note that DoS attacks refer to those originating from a single source, while **Distributed Denial-of-Service (DDoS)** attacks originate from multiple sources providing overwhelming traffic to the network in order to hinder its capability to provide its intended services. As many different types of attacks can result in denial of service, this makes this types of threats difficult to proactively prevent. This risk can be mitigated by collecting performance metrics, such as **transaction throughput and latency**, to detect compromised availability early on.

In the context of REWIRE, as aforementioned, we leverage the **Hyperledger Besu** platform for the implementation of the Blockchain Infrastructure. This technology, as detailed in D6.1, leverages the **Quorum Byzantine Fault Tolerance (QBFT)** consensus mechanism which, in tandem with the employed permissioning model, ensures that only authorized nodes can participate, thus enhancing the security and trustworthiness of the data flow. It also contains a built-in **Ordering Service**, as it ensures a single, agreed-upon ordering for transactions in the Blockchain. In addition, Besu also provides final confirmation of data delivery, recording an immutable entry that confirms the integrity and availability of the data to be recorded. These features provide protection against DDoS attacks, as only permissioned nodes are allowed to interact with the Blockchain infrastructure, thus preventing illicit traffic from flooding the network.

Recal that, in the context of REWIRE, we leverage the notion of **Secure Oracles** through the SGX-based **Town Crier (TC)** paired with Hyperledger Besu. This can also used for the mitigation of DDoS attacks, as it can securely mediate the interactions of external data sources (users and/or devices) with the distributed ledgers. This approach, as aforementioned, is able to ensure the reliability and trustworthiness of data recorded on the Blockchain, thus providing protection against DDoS attacks. Specifically, this is achieved through (i) **decentralization**, as spreading the workload against multiple nodes makes DDoS attacks harder, (ii) **redundancy**, as the use of muiltiple data sources and fallback nodes ensure service continuity, and (iii) **authentication**, as the authenticated communication and cryptographic proofs employed by secure oracles can ensure that the recorded data is legitimate, even when the system is under attack.

Thus, this decentralized approach not only removes the need for additional resources at the edge, but also provides **auditability** and **compliance** through the **continuous monitoring and verification of the integrity of external data sources**. This is enhanced through the **secure onboarding** of the devices into the REWIRE network, as well as the use of the **Configuration Integrity (CIV)** scheme for verifying that the devices interacting with the Blockchain are in a correct configuration state. Thus, **only devices that have not been compromised** (or there is no indication of having been compromised) **are allowed to send data to be recorded to the Blockchain**, thus limiting the surface of nodes that may be willing to launch a DDoS attack.

5.1.2 Consensus Manipulation

As aforementioned, the Hyperledger Besu technology employed in REWIRE leverages the QBFT consensus mechanism, while it is also possible to employ different consensus mechanisms, such as **IBFT 2.0**, **Clique**, **Proof of Stake (PoS)**, and **Ethash**. The consensus mechanism, in general, is responsible for governing the process of **synchronizing channel states** for making a collective decision on the recording of data transactions on the ledger. The main objective is to **distribute a state machine across**

a cluster of computing systems, thus ensuring that **each node in the cluster agrees upon the same series of state transactions**.

Specifically, Hyperledger Besu implements the **QBFT proof of authority (PoA)** consensus protocol, which is the recommended enterprise-grade consensus protocol for private networks. In QBFT networks, approved accounts, referred to as **validators**, validate transactions and blocks, and validators take turns to create the next block. Before inserting the block onto the chain, a super-majority (greater than or equal to 2/3) of validators must first sign the block. Also, note that it is possible for existing validators propose and vote to add or remove validators. Considering all the above, here we provide some types of attacks that may target the consensus mechanism:

- ***Leader Manipulation:*** In QBFT, a designated proposer (leader) is responsible for proposing new blocks. Thus, if an attacker is able to disrupt the operations of the leader, either via overwhelming it with requests (i.e., DDoS) or exploiting vulnerabilities to crash it, this may stall the consensus process and cause delays in block finalization. This has been documented in the context of Byzantine Fault-Tolerant Consensus Algorithms [37], which is the general category under which the QBFT algorithm is classified.
- ***Malicious Validator Collusion:*** In the QBFT consensus mechanism, if more thatn 1/3 of the validators of the network collude maliciously, it is possible that they may disrupt the network by censoring transactions, creating conflicting blocks, or preventing consensus by not participating in the voting process. This may halt the consensus process, or cause potential forks or violations of network safety and liveness properties, as discussed in [37].
- ***View-Change Flooding:*** In BFT-based schemes, in case the leader that coordinates the consensus procedure is determined to be faulty, it is possible to trigger a **view-change protocol** to select a new leader in order to maintain safety even during periods of timing violations, since progress only depends on the leader [34]. However, in case a malicious validator is present in the network, it may repeatedly trigger view changes by falsely claiming that the leader is unresponsive, thus causing the network to consume resources on unnecessary leader elections. This leads to increases in the latency of the consensus process, reduced throughput, and potential destabilization of the network.
- ***Equivocation (Double Voting):*** In case a malicious validator is participating in the consensus process, it may send conflicting votes to different nodes in order to confuse the consensus process, thus causing the consensus mechanism to fail if the equivocation is not detected and mitigated [38].
- ***Network Partitioning Attacks:*** During the execution of a consensus process, it is possible that an attacker may cause a network partition, thus isolating subsets of validators from each other. Thus, the victim node(s) have a different view of the Blockchain that the adversary controls. This may lead to inconsistent views of the Blockchain or stop the consensus process until the partition is resolved. It may also cause potential forks or inconsistencies in the state of the Blockchain [32].

In addition, attacks on the network consensus include **DDoS** and **transaction reordering attacks** [28]. The Ordering Service of the Hyperledger Besu Blockchain infrastructure currently only utilizes **Crash Fault Tolerance (CFT)** consensus algorithms, meaning that no malicious Orderer node can be tolerated [3]. According to [4], the following attacks can be considered when the Hyperledger Fabric Ordering Service contains a malicious ordering node:

- ***Sabotage Attack***: **The Ordering Service is responsible for collecting the transactions and putting them into data blocks**. In this type of attacks, a malicious or compromised Orderer does not add transactions stemming from a given Peer node (or a set of Peers). This behaviour can negatively impact the given Peer nodes (e.g., a Peer wants to add new logs, but its logs will not be included in a block) which, in turn, affects the immutability and auditability of the entire Blockchain since this might lead to data transactions not recorded.

- ***Block Size Attack and Batch Time Attack***: The Ordering Service is also responsible for setting the **Channel Configuration**. **If a malicious entity significantly modifies the block size, it may be possible that the block (consisting of a large number of transactions) will never be published**. Conversely, if the **block size is set to an exceedingly small size, then a large number of blocks will be committed to the ledger, which can have a negative effect on the performance of the Blockchain**. The **height of a ledger is the one factor that affects mostly the time of a data transaction** - both as it pertains to data querying and data recording since there is a higher number of occupied block positions in the ledger that need to be traversed. Furthermore, having a higher number of blocks been published on a ledger, affects the efficiency of the consensus mechanism since there are more remote calls between the Ordering Peers for synchronizing the state of the channel after the recording of a new block. If a batch time attack is executed with an extremely large amount of time, then the Ordering Service will have to wait for an extremely long time before cutting a block. Thus, the manipulation of the batch time (that can be initiated by a malicious ordering node) can negatively impact the Blockchain's performance (i.e., transaction validation throughput and latency).

- ***Transaction Reordering Attack***: In this attack, the **order of the transactions can be placed differently from their arrival**. For example, consider two Peers, Peer A and Peer B, where Peer A's transaction arrives before Peer B's transaction. If the Orderer places Peer B's transaction as arriving first, this can cause an issue in case the two Peers compete for which Peer can resolve a task faster, and which Peer sent the result in a transaction first (these checks are usually performed during a *mining* process that may also include the solving of a *hash puzzle* [12]). *In REWIRE, for instance, we use the ID of the last block as the seed for calculating the nonce to be used as the SW Update challenge for a Verifier to initiate the attestation process with a Prover* [9, 10]. Such re-ordering attacks can actually have an impact on this functionality. Furthermore, based on the time of issuance of a transaction, its auditability might be affected if a re-ordering attack takes place, which in turn will affect the certification of the entire service graph chain. Another possible attack that falls under this umbrella of **Transaction Ordering Dependence vulnerabilities, is when a newly generated block contains 2 transactions enforcing the same contract**. Such plots do not provide enough information to users to determine the state of the contract or when the individual invocation is initiated. Therefore, **when the output of both transactions is dependent on the order, the contract might result in an inconsistent state**. Think, for instance, a Verifier device pushing to the ledger two attestation results: (i) one representing the result of an attestation task executed with a Prover device as dictated through a security policy (smart contract), and (ii) another one been the result of its local attestation for producing those evidence needed for the Blockchain Peer to attest to the correct state of the Verifier itself. Thus, these two transactions are dependent to each other and if the ordering is intentionally altered then this might lead to the rejection of the attestation report as possibly been falsified since the integrity of the Verifier could not be attested first.

The Ordering Service of the Hyperledger Fabric is based on the **Raft Crash-Fault-Tolerant (CFT)** protocol, which is based on such a **Leader Election Process**. The main principle of the CFT is that the Orderer nodes follow the Leader. Therefore, it is possible that a malicious Leader can cause the service to behave maliciously. Mitigation measures against such attacks are further discussed in Section 5.1.4.

5.1.3 Verifiable Credential Manager Compromise

In the context of REWIRE, as previously analysed, devices have the capability to manage their identity through the use of **Verifiable Credentials (VCs)**. As shown in Figure 5.1, these are issued during the secure enrolment of the device and contain a set of device attributes, which may afterwards be used in the context of **Verifiable Presentations (VPs)** that can verifiably prove ownership of these attributes in a

secure and verifiable manner. As showcased in Chapter 2, VCs are stored and managed by the **Verifiable Credential (VC) Manager** in the Trusted World of the device, which also includes key management capabilities.

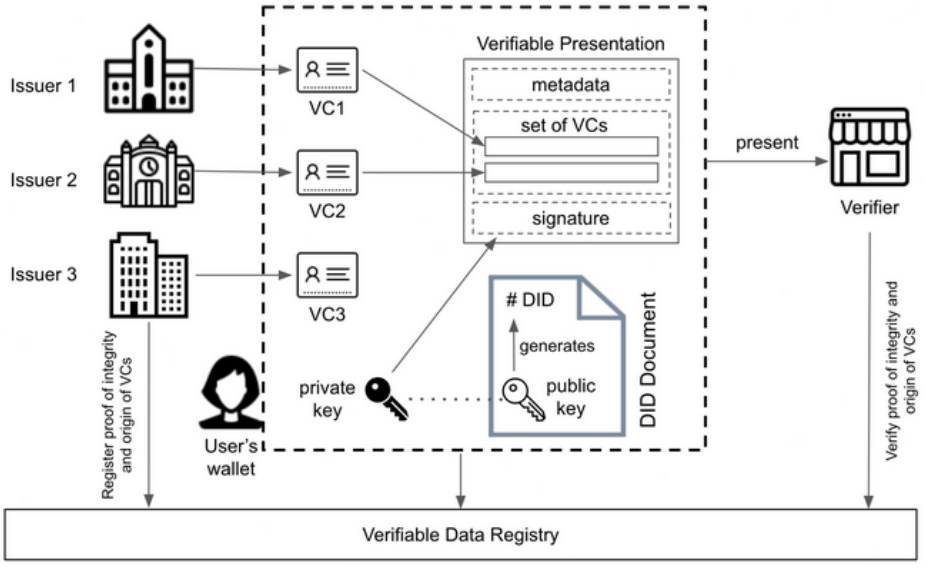

Figure 5.1: Issuance of Verifiable Credentials (VCs)

As VCs essentially contain information on the identity of the device, they can be examined in the context of **identity management**, which entails the **storage**, **management**, and **sharing** of identity-related data. This is in line with the notion of **selective disclosure**, based on which users and devices should be able to manage and control their identity-related data, by only sharing the subset of data that is needed in order to access a particular set of data stored on the Blockchain.

In general, the **VC Issuer** is responsible for the issuance of those VCs, and is responsible for signing the VC that contains a set of assertions about the Holder device, using its public/private key pair. The signing of credentials makes them tamper-proof and cryptographically verifiable regarding their origin. The signed VCs are then issued to the Holder, who stores them in its VC Manager component before being able to create derivations of it using their private/public key pair included in the credential. These derivations are essentially the aforementioned **Verifiable Presentations (VPs)**, which are cryptographically verifiable proof of ownership of these credentials. A VP may be presented to any entity that is responsible for verifying the possession of the specified attributes by the device, and incudes **a digital signature with the Holder's "binding key" as tamper-evident proof that authorship of the data can be trusted after a process of cryptographic verification**. This means that *it is essential that the Holder never shares its private binding key and that it is protected properly*.

Considering the above, the main challenge regarding threats against VC management lies in *the potential for attacks against SW-based VC management variants targeting these binding keys*, as the use of a SW-based keystore introduces several security risks and raises trustworthiness issues [27]. Specifically, it has been well established that memory-related vulnerabilities can be exploited in order to compromise the target device, obtain illicit access to its memory layout, and try to extract cryptographic material. In this case, a malicious party may attempt to impersonate a legitimate user or device (that has been enrolled into the system) by stealing the stored VCs and constructing VPs for obtaining unauthorized access to various data resources. Thus, the challenge is formulated as, *how can a Verifier entity be sure that the key of the Holder presenting a VP remains under its control, and cannot be leveraged by an unauthorized entity?*

In the context of REWIRE, as previously analysed, the secure management of keys is enabled in Keystone through the use of **secure enclaves** with minimal trusted hardware. Specifically, Keystone builds enclaves through the **Security Monitor (SM)**, which is responsible for setting up enclave memory regions (enclave page tables), protecting memory isolation, and handling cryptographic operations. Enclave memory is isolated and protected by hardware mechanisms, such as **Physical Memory Protection (PMP)**, so that only the enclave itself (and the SM) are able to access key material stored in the enclave memory. In addition, it is possible to create **session keys** or **intermediate signing keys** within the enclave during runtime, or sealed to local storage through a sealing key.

According to the eIDAS regulation of the European Union pertaining to identity management [6], it is specified that a certain **Level of Assurance (LoA)** is defined regarding credential management, characterizing the degree of confidence in the employed electronic identification means, thus providing assurance that the entity claiming a specific identity is the intended one to which that identity is assigned. However, based on the eIDAS regulation, *bare proof-of-possession of a SW-based (private) key does not even achieve the lowest LoA, as it only involves a single authentication factor*. In order to achieve the ”high” LoA as defined by eIDAS, a set of requirements need to be fulfilled, such as **the isolation of keys from the Holder while still being stored in the user's domain**, and **the binding of identity data to the Holder (Holder Binding) based on a unique identifier representing the holder** (i.e., a secret key). Thus, from all the above, it follows that REWIRE offers the hardware-based capabilities that provide the foundation for achieving the aforementioned requirements, while also *empowering users to control their level of privacy by selectively disclosing only the attributes needed for accessing a service in a verifiable manner*.

5.1.4 REWIRE Consensus Mechanisms

As previously outlined, REWIRE employs the **Quorum Byzantine Fault Tolerance (QBFT)** consensus mechanism within the Hyperledger Besu platform. It is a robust, deterministic consensus protocol for permissioned networks, as it provides **fast finality**, **high resilience**, and **secure validator management**. Note that QBFT was adopted by Hyperledger Besu as a more robust alternative to Istanbul BFT (IBFT), in order to address limitations in terms of scalability, performance, and configurability (with regard to block times, validator, rotation, etc.). It ensures consensus finality, meaning that once a block is committed it is final and cannot be reverted (conversely to Proof-of-Work or Proof-of-Stake systems), thus making it ideal for the types of organizations and environments targeted by REWIRE. Here, we provide a comprehensive overview of the QBFT scheme, and the process followed for consensus achievement.

In the QBFT scheme, two types of roles are defined for the participating nodes: (i)

- **Validators**, which refers to all nodes participating in voting to reach consensus. Note that QBFT supports two types of validators: (i) **Static**, which are configured in the genesis block and remain unchanged, and (ii) **Dynamic**, which can be added or removed through special consensus transactions, thus allowing for a more flexible and adaptive network.
- The **Proposer (Leader)**, who is selected in a round-robin manner among validators to propose a block.

QBFT works in **rounds** for the validation and addition of a block to the chain, each one of which consists of the following steps, considering a fault tolerance level f, referring to the maximum number of faulty nodes that can be tolerated:

1. **Propose:** The proposer creates and broadcasts a block proposal to all other validators. Validators verify the proposal's validity and sequence number before entering the pre-prepared state.
2. **Prepare:** Validators receive the proposal and, if valid, broadcast a **prepare** message to indicate their agreement on the proposed block. Validators wait to receive $2f + 1$ prepare messages from other validators before progressing to the next phase.

3. **Commit:** After receiving enough *prepare* messages, validators broadcast a *commit* message.

4. When a validator receives enough *commit* messages ($\geq 2f + 1$), it finalizes the block and adds it to the chain. Note that this process ensures that consensus is reached, even in the presence of up to f malicious or non-responding validators.

From all the above, it follows that QBFT provides a variety of countermeasures against consensus manipulation attacks. Specifically, QBFT follows the classic BFT assumption of tolerating up to f malicious nodes in a total of $n = 3f + 1$ validators, meaning that as long as at least two thirds of validators are honest, the protocol ensures **safety** (no forks or conflicting blocks) and **liveness** (network progression). In addition, the aforementioned $2f+1$ voting threshold regarding *prepare* and *commit* messages ensures that no single malicious actor (or group of $\leq f$ malicious actors) can forge a block or create inconsistencies, and manipulation is impossible without corrupting at least $f+1$ nodes. Another feature of QBFT that enables the mitigation of consensus manipulation attacks is **leader rotation**, which entails a proposer selection process, based on which proposers (leaders) are rotated **deterministically** among validators in a round-robin manner, and if a leader is malicious or faulty (e..g, proposes invalid blocks or remains silent), a timeout triggers a new round with the next validator as leader. This process both *prevents long-term leader-based manipulation* (e.g., spamming the chain with bead proposals) and *limits the power of a single node to delay or manipulate consensus*.

5.2 Threat Analysis of Secure Oracle and Data Veracity

Secure oracles serve as critical bridges between off-chain data sources and on-chain smart contracts. In REWIRE, the Town Crier-based secure oracle is responsible for querying data from external sources, verifying its authenticity inside a Trusted Execution Environment (SGX enclave), and delivering the processed output to the Besu blockchain. This process ensures that data critical to REWIRE's use cases remains verifiable, tamper-proof, and privacy-protected.
By design, the secure oracle provides confidentiality and integrity during the data-fetching process. However, its integration into a distributed blockchain workflow—and its reliance on external databases and trusted hardware—expose it to a variety of threat vectors. These must be carefully analyzed and mitigated to protect REWIRE's trust and resilience.

5.2.1 Identified Threats

Despite its strong guarantees, the secure oracle remains a potential target for various attacks. These range from communication-level threats to hardware-level side channels and smart contract exploitation. Below we provide a more detailed analysis of the main threats identified in our implementation:

- **Man-in-the-Middle (MitM) Attacks**: During the data fetching phase, the enclave communicates with external databases such as InfluxDB. An attacker positioned between these two components may intercept or manipulate the data exchange. Even though HTTPS is used for secure communication, misconfigured TLS or weak certificate validation can expose the oracle to MitM risks. *Impact*: Compromised data integrity or confidentiality may lead to propagation of false information to the blockchain.

- **Data Source Pollution and Veracity Attacks**: The oracle depends on external data sources to retrieve facts and metrics. If a data source is compromised—e.g., the InfluxDB instance is infiltrated—an attacker can inject incorrect or malicious data that appears valid. Without additional source validation or attestation mechanisms, the enclave would unknowingly process this data. *Impact*: Can severely compromise decision-making processes and trust in system outputs.

- **Replay Attacks and Timestamp Spoofing**: Replay attacks involve resending previously valid queries, while timestamp spoofing manipulates time-related metadata. Without nonce and timestamp verification, the enclave may process outdated or duplicate data, assuming it to be fresh. *Impact*: System state can be reverted or inconsistently updated, harming reliability and freshness guarantees.
- **Re-Entrancy and Smart Contract Exploits**: Smart contracts that interface with the oracle may contain logic flaws that allow recursive calls (re-entrancy) or conditional logic manipulation. These vulnerabilities have been historically exploited (e.g., DAO hack). In REWIRE, this type of exploit could lead to repetitive calls to oracle functions or manipulation of their outputs. *Impact*: May result in denial of service, unintended behavior, or unauthorized state changes.
- **Side-Channel Attacks on SGX**: While SGX enclaves provide strong execution isolation, they are not immune to physical or microarchitectural leakage. Attackers with access to shared hardware resources (like cache) may extract internal data by observing timing behavior or speculative instruction patterns. *Impact*: Leaks secrets such as cryptographic keys or internal logic, weakening the oracle's core security guarantees.
- **Flooding and Denial of Service (DoS)**: The relay or enclave may be subjected to high-volume or malformed queries that exceed their processing capabilities. Attackers may try to overwhelm system resources, disrupt normal queries, or degrade performance. *Impact*: Affects availability of oracle services and creates bottlenecks in data processing.
- **Single Point of Failure**: The current design assumes a single Town Crier enclave and relay instance for communication and verification. This architectural choice introduces a centralization risk—if either the enclave or relay crashes or is compromised, the oracle pipeline halts. *Impact*: Leads to disruption of all dependent workflows and potential loss of system availability.

A summary of the identified threats and corresponding mitigation measures is shown in Table 5.2.

5.2.2 Security Measures and Future Directions

To mitigate these risks, REWIRE implements several protections across the secure oracle pipeline:

- **Secure Communication via gRPC + TLS**: Ensures encrypted and authenticated data transmission between relay and enclave, preventing unauthorized data injection and eavesdropping.
- **Nonce-based Freshness Validation**: Each request includes a nonce and timestamp, preventing reuse or replay attacks.
- **Verifiable Proof Validation (VPs)**: Oracle input is bundled with cryptographic proofs validated inside the enclave, ensuring authenticity and integrity.
- **Data Source Attestation**: Incorporating trust attestations from the data source helps mitigate data pollution and verify origin authenticity.
- **Off-Chain Storage with On-Chain Anchoring**: Large data payloads are stored off-chain with cryptographic hashes recorded on-chain for tamper detection.
- **Benchmark Logging and Auditability**: Oracle operations are timestamped and logged to support traceability, anomaly detection, and incident analysis.
- **Redundancy and Load Mitigation**: Plans include deploying multiple relay/enclave nodes to avoid single points of failure and absorb abnormal traffic loads.

- **Secure Smart Contract Practices**: Formal verification and testing pipelines are applied to mitigate logic flaws, including re-entrancy and business logic bugs.

Threat	Mitigation	Status
Man-in-the-middle (MitM) during HTTPS retrieval	Mutual TLS (gRPC) and input integrity validation in enclave	Implemented
Data source pollution and veracity attacks	Verifiable proofs (VPs) validation and data source attestation	Implemented
Replay attacks and timestamp spoofing	Nonce and timestamp freshness validation	Implemented
Re-entrancy and smart contract exploits	Formal verification, secure coding practices	Partial
Side-channel leakage from enclave execution	TEE hardening and research ongoing	Open
Flooding and denial of service (DoS)	Rate limiting, redundancy, load monitoring	Planned
Single point of failure (relay or enclave crash)	Redundant and distributed oracle architecture	Planned

Table 5.2: Key Threats and Mitigation Measures in Secure Oracle

Chapter 6

Policy Expression Languages and Policy Enforcement

One core aspect of the design-time phase of the REWIRE architecture entails the formal verification of the system configurations of the target device based on the requirements and specifications given by the OEM and the use case provider. The output of this process is intended to steer the SW-HW co-design process, as well as the enforceable application-specific configurations to be provided to the Policy Orchestrator (been responsible for enforcing any security actions been manifested from the design phase). The Policy Orchestrator is then responsible to convert "*actions to tasks*" that will be enforced by the Facility Layer - tasks/actions are usually operations that guarantee device security continuity and lifecycle management; i.e., it can be the list and order of execution of attestation tasks or policies regarding SW/FW updates and any migration as a result to the change of the trust level of the host device. The core innovation of REWIRE in this regard is twofold: (i) the identification of **which properties can be formally verified at design-time and which need to be attested during runtime**, thus defining the **trust boundary** of the operations of the device, and (ii) the execution of the security policies in a **semi-automated manner depending on the need for the periodicity of the defined tasks**. Such a solution bridges the gap in REWIRE between design time assurances against use-case requirements-properties offered by formal verification, and run-time protection through policy enforcement with regard to properties where formal verification was infeasible.

The main challenge to consider with regard to the expression of such security policies is to determine *what* is the necessary information that should be included within a policy and *how* it can be conveyed. The most prominent method to achieve this is the **Common Information Model (CIM)** [5], which is the main DMTF standard (Distributed Management Trask Force) that provides a common definition of security management functions and security information, independent of the type of device or the types of security mechanisms defined by the security policies. The CIM defines concepts such as authorization, authentication, filtering and security, as well as the notion of obligation and delegation policies. However, CIM models cannot be directly used as a policy language, due to the large number of classes they contain. Considering also the aforementioned aspects of the policy modelling approach of REWIRE, the issue of policy modelling becomes increasingly complex and necessitates the identification of a language that is able to appropriately express the information that needs to be conveyed by the REWIRE security policies.

Considering the above, a *policy language needs to be selected in order to compliment the Formal Verification component of REWIRE, enforcing properties by the Policy Orchestrator* that offers both the required level of **granularity e.g. for the expression of the types of attributes to be attested during runtime**, but also the **periodicity in the execution of the tasks defined by the policies** (e.g., attestation, device migration, or software update). Therefore, in REWIRE, we need to be able to consider a policy language that can express **abstractions when formulating such security policies**, used for expressing specific configurations with regards to the attestation mechanisms to be executed, in a **device independent format**. To this end, as it will be demonstrated throughout this Chapter, in REWIRE we adopt the integration

of the **Medium-Level Security Policy Language (MSPL)**, first introduced and defined in the European project SECURED [29].

As a first step in identifying the abstractions to be modelled, we first need to define the **security objectives** for the devices targeted by the REWIRE solution. These can essentially be extracted by answering the following questions:

- *What has to be done about security?* This entails the identification of the **security properties** of the device, such as confidentiality, integrity, availability, non-repudiation, authentication, authorization, or audit, as well as the required **security services**. The security services in REWIRE also cover requirements such as *operational assurance and static configuration assurance*, which are achieved through **attestation schemes** such as Configuration Integrity Verification (CIV) and Control Flow Attestation (CFA).
- *Where is the security service required?* The entities and devices where the security process is being executed needs to be identified, considering that the entities referenced by an objective are a subset of the asset cartography of the target domain, which dictates the IDs (or types) of devices that need to execute an security service (e.g., attestation or software update) towards creating trust-aware service graph chains.
- *Between which nodes in the chain does the security service need to be executed?* Considering the case of remote attestation, this process requires a **Prover** to be attested, and a **Verifier** that verifies the correctness of the Prover's state. This enables interacting devices belonging to application domain, to exchange data in a trustworthy manner. Thus, to address the need for establishing federated trust between services and devices, we need to identify which devices need to fulfil these roles, i.e., the set of devices acting as Provers and Verifiers.
- *When is the security service required?* This refers to the temporal relation of the service with the occurrence of the corresponding security event, i.e., before, during, or after. In REWIRE, recall that the **periodicity** of an action is of particular importance, as it needs to be specified whether, for instance, a security action needs to be executed periodically at regular time intervals, or as a response to a security event (e.g., secure update as response to a failed attestation).
- *What additional elements to complement the specification of a security objective are needed?* This refers to any additional information that may be required in order to execute a security service (e.g., attestation or software update) for achieving a security objective. This is related to the information provided by the Formal Verification component of REWIRE, which dictates which attributes can be verified during the design-time phase, and which attributes need to be attested dynamically during the runtime phase.

When considering the security of systems consisting of multiple interconnected assets, the interpretation of the above questions requires the modelling of two different types of security objectives:

- *Direct objectives*, initially defined by the domain administrator, as the main goals of the security processes of the domain infrastructure.
- *Indirect objectives*, which are indirectly derived from the security relationships and interactions between assets, thus creating security chains to express the security posture of the entire ecosystem.

Considering all the above, it is necessary to identify the most appropriate policy language that is able to answer all the aforementioned security objectives, considering also the architectural approach of REWIRE, as well as the requirements that need to be fulfilled by the selected language. In the following, we will provide information on available policy languages in the literature, as well as the motivation behind the eventual selection of the MSPL language, and details on the structure and MSPL and its use for formulating policies.

6.1 Policy Position in REWIRE through an Example

In Chapter 2, we provided a detailed overview of the REWIRE framework, centred around the use of the **Formal Verification** component for the identification of the properties that need to be assessed through attestation schemes for ensuring that the trustworthiness level of the system remains at an acceptable level, and the actions that need to be taken for lifecycle management of a device (secure updates and device migration). Such attestation actions are enforced through the use of **security policies**, which are formulated based on the output of the Formal Verification component and dictate the processes/binaries that need to be verified prior to their execution.

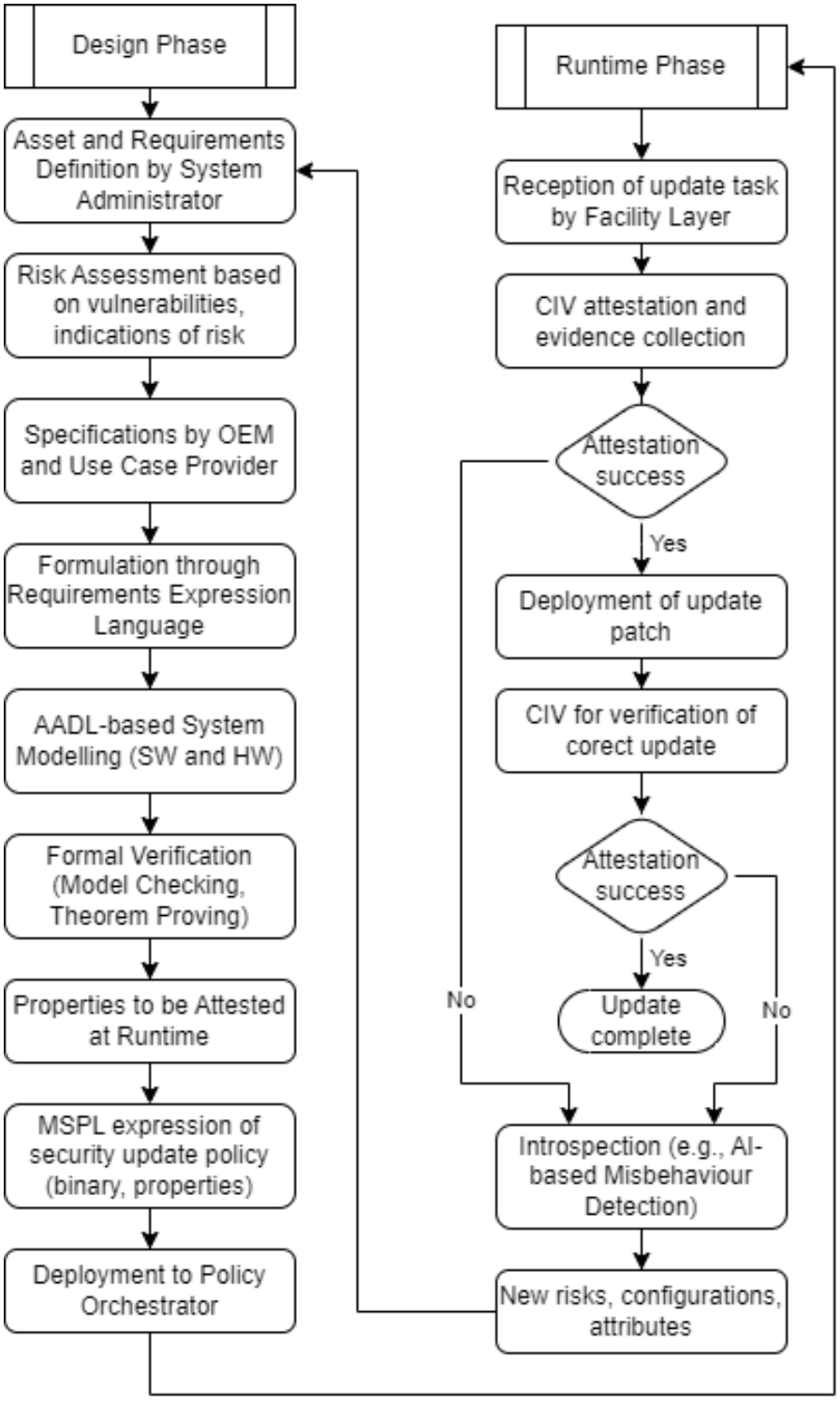

Figure 6.1: Secure Software Update example of design-time and runtime policy positioning in REWIRE.

Recall that the REWIRE architecture functions under two core phases, i.e., the **Design phase** and the **Runtime phase**. Here, to better illustrate the reasoning and motivation behind the selection of the REWIRE security language, as well as the expression of the security policies, as depicted in Figure 6.1, we provide details on the positioning of these policies within the REWIRE architecture throughout these two core phases through the example of a **secure software update process**, which is a core security enabler of REWIRE, described in detail in Section 2.3.3. This process is outlined as follows:

6.1.1 Design-time Phase:

1. The **Compositional Verification and Validation** component receives information regarding system requirements and asset definition from the **System Administrator**, as well as hardware specifications from the **OEM** and requirements related to protocols and business services from the **Use Case provider**. Then, a model of the system is created using **Architecture Analysis and Design Language (AADL)**, which allows for hierarchical modelling of system components and their interactions.

2. Once the AADL model is created, the Formal Verification component employs a variety of reasoning techniques in order to validate the correctness of the software stack of the device, including security and cryptographic processes running on the device. The core outcome of this process is the identification of the **properties that can be formally verified during design-time**, and those that cannot be verified, due to state-space explosion (exhaustion of computational resources when analyzing the formal model) and **need to be dynamically attested during runtime** using the available attestation enablers. The **Manufacturer Usage Description (MUD)** profile for each device is also created, specifying essential information about device characteristics and establishing a baseline for its behavioural profile (e.g., policy hashes, type of risks, Required Trust Level (RTL), types of available security controls, types of evidence needed for the execution of the security controls).

3. Using the selected *Policy Expression Language* (MSPL), the set of security policies is formulated, specifying the **properties to be attested**, as well as the **actions to be performed** in the context of the secure lifecycle management of the device. These may include **attestation processes** for attesting the correctness of the defined properties, as well as **secure updates** or **device migration** actions for ensuring that any identified vulnerabilities are mitigated. In the Secure Software Update example depicted in Figure 6.1, the policy specifies the **binary that needs to be updated**, as well as the **properties to be attested** for verifying the correctness of the device, both before and after the update is performed. It is important to note that *the Compositional Verification and Validation process can verify that the authenticity of the update process has not been compromised, but cannot attest to the type of update or the correctness of the update itself*. This necessitates the execution of a **Configuration Integrity Verification (CIV)** process both before and after the update is performed through a runtime policy. *This also highlights the need for the expressiveness of the employed policy language, in order to capture different types of security controls.*

4. The policy is sent to the **Policy Orchestrator** of the device, which is responsible for orchestrating its execution, considering the **periodicity** properties defined in the policy. For instance, a policy may need to be executed periodically at regular time intervals, or as a response to a security incident (e.g., deployment of a secure update as a mitigation action for patching an identified vulnerability).

5. The **Facility Layer** of the device receives the tasks dictated by the policy (e.g., attestation, update, migration), and is responsible for orchestrating its execution by the appropriate devices or entities.

6.1.2 Runtime Phase:

1. In order to enable and support the execution of the security enablers specified by the security policies of REWIRE, it is necessary to enrol the device into the REWIRE framework through the

Zero-Touch Onboarding (ZTO) process. This process entails the creation and activation of the required cryptographic material for utilizing the available attestation enablers, such as the **Attestation Key (AK)** for enabling the execution of attestation enablers and secure communications within the REWIRE infrastructure. Afterwards, the device communicates with the **MUD Profile Server**, which receives the aforementioned MUD profile of the device and translates it into context-specific policies, stored in the REWIRE Blockchain Infrastructure.

2. Through the **Facility Layer** of the device, the actions specified by the security policies received through the Policy Orchestrator are performed. In the case of the Secure Software Update depicted in Figure 6.1, the actions specified are as follows:
   - CIV attestation on the properties determined during design-time, to verify the correctness of the device before the update (e.g., the correct software version is installed). This requires the collection of evidence through the **REWIRE Tracer** running at the Untrusted World of the device and the communication with the **Attestation Agent** of the device for the execution of the attestation logic, and the verification of the correctness of the device state by the **Verifiable Policy Enforcer (VPE)** through the validation of the **Key Restriction Usage Policies**.
   - The software patch itself is deployed and installed on the device.
   - After the installation is complete, an additional CIV process is performed in order to verify that the update has been installed correctly.
3. In case one of the aforementioned CIV processes fails, this indicates the need for further investigation. In order to detect anomalies in the operational behaviour of devices, REWIRE also offers **AI-based Misbehaviour Detection** capabilities that process system and network data collected during runtime. To this end, the attestation evidence is send by the devices to the AI-based Misbehaviour Detection component, which then generate **risk indicators**, in case of the identification of a device behaviour that diverges from the known and expected behaviour of the device. Risk indicators may also originate from hook-based event analysis performed by the **SW/FW Validation Component**.
4. The identified risk indicators are forwarded to the **Risk Assessment** engine for performing risk re-evaluation and updating the risk graph. Such risk indicators may be the result of failed attestation actions that demonstrate that the device is not in an expected configuration state, or outcomes of the aforementioned AI-based Threat Intelligence or SW/FW Validation actions.
5. Based on the outcomes of the updated Risk Assessment, a **SW/FW Update** or **security patch** may need to be deployed via the **SW/FW Validation Component**. Then, the update is sent to the **SW/FW Distribution Service**, which is responsible for distributing the vulnerability-free SW/FW update to the device.
6. The outcomes of the risk assessment process may include the need for the definition of additional actions that need to be performed for the mitigation of newly identified threats and vulnerabilities, such as the execution of attestation enablers. It may also include changes in the attributes that need to be verified during runtime. This leads to **updating the security policies** accordingly and their distribution to the **Policy Orchestrator** of the device, thus leading to the continued runtime evaluation of the device.

6.2 REWIRE Compositional Validation and Verification solution

Throughout this chapter, we provide details on the purpose and motivation behind the use of policies within the REWIRE framework, as well as information on their positioning within the action workflow of REWIRE. Here, considering all aforementioned information, we present the characteristics that need to

be fulfilled by the selected policy language in order to support the execution of these policies. Thus, the remainder of this section is dedicated to the justification of the selection of MSPL as the policy language employed by REWIRE.

The main vision of REWIRE consists of two core pillars: (i) the deployment of **adaptable policies for the execution of attestation tasks** with the capability to account for the required information, e.g., the properties to be attested, the periodicity of the attestation task, the type of evidence to be collected, and (ii) the deployment of the appropriate **policies for lifecycle management of the devices through secure updates and device migration**. Thus, we need to consider that policies in REWIRE do not only refer to authorization and authentication decisions, but also policies that make sound statements on the **correctness of the state of the device**, which extends to the **overall composition of the system**. Thus, edge devices and their software components must be able to prove statements about the state, so that other components can align their actions appropriately, and a policy language is required so that the correctness of specific properties of the device can be verified, based on the output of the formal verification process.

Determining what policies need to be expressed and enforced within the REWIRE architecture entails modelling and specifying the necessary policies for enforcing authorisation decisions based on properties attested to by the assets (which are composed into broader systems/SoS), as defined by the Formal Verification component. To this end, we should be able to express policies that: (i) **mitigate the risks** of the safety and security critical systems when enforced, (ii) consider **resource constraints** imposed by components that need to attest the correctness of their properties, (iii) specify the **type of evidence** to be collected in the context of attestation processes, as well as to perform a more in-depth investigation of the system's behaviour towards detecting new threats and zero-day vulnerabilities, and (iv) define the **periodicity of the actions to be performed**. The policies must be **expressive**, **deployable**, and **enforceable** and may be updated during runtime if the risk graph is updated to incorporate new types of vulnerabilities. Through the definition of such policies, the device can attest to the correctness of the defined behavioural and execution properties, periodically or on-demand. These may include execution paths to specific memory regions, as a result of the invocation of the functions of interest.

Taking into consideration all the above, the high-level security requirements that should be fulfilled by the policy language in the context of REWIRE are as follows [33]:

- **Simplicity:** The definition of a security policy must be tailored to the available security enablers, and must be written in a way that is simple enough to be understandable and eventually converted into smart contracts, but also have the required level of granularity to express the properties to be attested and the periodicity of the attestation tasks. This is accomplished by structuring the policy with predefined statements regarding tasks that must be executed by the target device and what properties need to be attested.

- **Expressiveness:** The language employed by REWIRE should be expressive and flexible in order to support the conditions regarding the periodicity of the execution of the specified tasks, such as the order of execution tasks, the type of security enabler to be executed, where the task is executed, the type of keys to be used, etc.

- **Extensibility:** The policies must be extensible so that, in the case of a newly identified vulnerability or a zero-day threat, the policy language should allow for the update of the relevant policies with additional tasks, as well as devices to execute these tasks.

- **Abstraction:** The selected language must contain abstract security-related configurations, independent from the specific characteristics of the device where the policy is applicable, since the representation of a policy should not depend on the semantic details regarding the configuration of the device.

- **Diversity:** The REWIRE framework must take into consideration the trustworthiness level of all applicable assets, and the selected policy language must support the configuration of such capabilities, which can then be applied to different types of attestation actions.
- **Continuity:** The policy language should ensure the continuity of the policy chain, meaning that it should be possible to track the tasks performed as part of the enforced policies and the entities they are executed by.

Regarding the actual content of the policies, we need to consider the **constraints** imposed by the target system. For instance, the **available resources** need to be considered, including the enablers that can be executed by the target device. For instance, consider that the Risk Assessment engine has identified a number of vulnerabilities for a particular device. However, if this device has limited computational resources, it may not be able to support the execution of a the desired set of actions. In this case, a constraint to be considered would be the selection of the optimal attestation tasks that can be executed in tandem with the operational tasks of the device, thus minimizing the impact on the normal operation of the device.

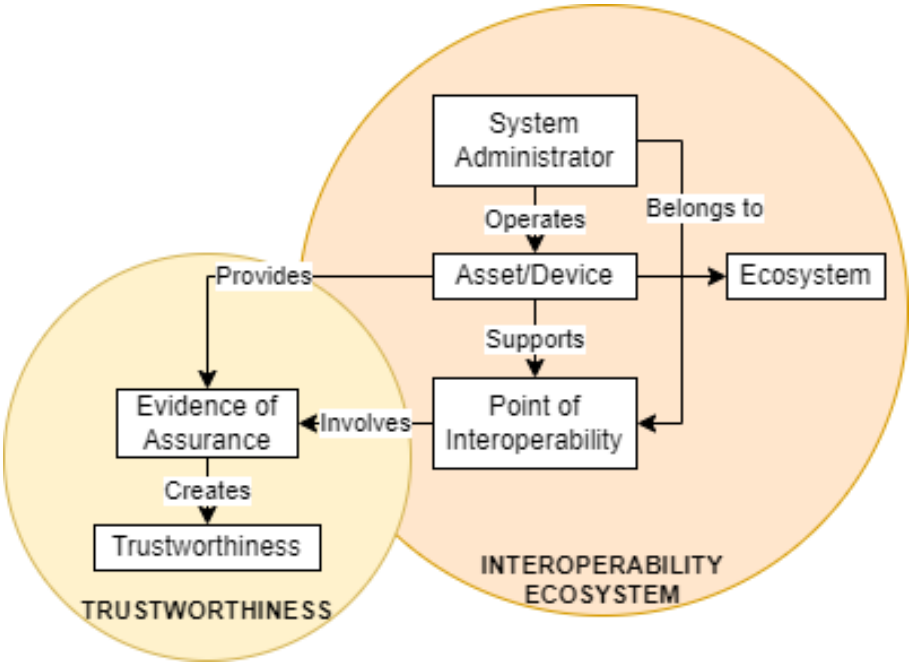

Figure 6.2: Desired REWIRE policy output.

As it will also become evident throughout the remainder of this Chapter, MSPL is the policy language that is able to fulfil all policy language requirements of REWIRE. Specifcally, the MSPL language [33] is able to describe the configuration settings for a class of security controls. It is simple and flexible, offering a level of abstraction that supports statements related to typical actions performed by various security controls. It also supports the execution of an ordered sequence of actions related to matching packets or payloads, which is appropriate for addressing the periodicity needs of REWIRE. In the following, we will provide details on other existing languages in the literature, and we will demonstrate that MSPL is the most appropriate language in the context of REWIRE.

Considering all the above, the **desired output of a policy within REWIRE** is presented in 6.2, and is expressed as follows: Consider a system operated under a **System Administrator**, and consists of various **Assets** and belongs to an **Ecosystem**. In order to achieve **Interoperability**, i.e., interaction with other devices, with the required level of **Trustworthiness**, this needs to be performed by involving specific types of **Evidence of Assurance**. This structure supports the level of abstraction required for policy modelling in REWIRE, and is a high-level depiction of the policies that will be expressed using MSPL, and will be described in further detail in the following.

6.3 State of the Art

As detailed in Chapter 2, the architectural approach of REWIRE entails the formal verification of the available security controls and processes by the **SW and HW co-design Formal Verification** component, considering the requirements of use case providers and OEM requirements (e.g., business services, protocol requirements, and HW specifications), in order to output a set of **security policies** that need to be received and enforced by the **Policy Orchestrator**. Thus, it is important to select the appropriate **policy expression language** in order to be able to express these policy in a manner that is readable by the Policy Orchestrator as **enforceable application-specific configurations**. In this regard, several policy expression languages have been proposed in the literature, suitable for various different types of applications depending on their needs. *Here, we provide a brief overview of the features and shortcomings of the most prominent languages, in order to determine the most appropriate language to be used in the context of REWIRE*:

- **Common Information Model (CIM) [5]:** This is the main DMTF standard that provides a common specification-agnostic definition of management-related information for systems and services with several customization capabilities. The CIM defines three components/fields: (i) **Specification**, which contains the information needed for integration with other management models, (ii) **Schema**, which contains the actual model descriptions, and (iii) **Metamodel**, which defines the semantics for the construction and representation of new models.

- For enabling the medium- and low-level abstraction in the representation of security policies, the **xCIM Security Policy Language (SPL)** has also been defined, allowing the representation of policies as structured XML documents. However, since the creation of such XML documents may be exceedingly verbose and prone to errors, two higher-level languages have been developed to address this issue [2], whose representations can be translated into xCIM models:

  - ✓ **Security Policy Language (SPL)**, which allows the representation of policies in a user-friendly language close to spoken English.
  - ✓ **System Description Language (SDL)**, which is defines an XML-based schema that simplifies the representation offered by xCIM.

- **eXtensible Access Control Markup Language (XCAML) [1]**: A standard specification for an attribute-based access control policy language, which defines three core element categories, i.e., **PolicySet**, **Policy**, and **Rule**, and allows any combination of such elements that enables the definition of the conditions that need to be fulfilled for the execution of the policy, the entities the policies apply to, and the actions defined for the execution of the policy or policy set. Thus, a policy request is followed by the evaluation of its rules and the appropriate access decision and response.

- **High-level Security Policy Language (HSPL) [29, 30]**: HSPL hides low-level configuration details of the defined security policy and provides the capability to express general protection requirements, in a manner that is readable and understandable by non-technical personnel and users. Specifically, an HSPL statement is structured as:

  $$[sbj]\ act\ obj\ [\ (field\_type,\ value)\ ...\ (field\_type,\ value)\ ]$$

  where sbj refers to the entity authorized to perform the action act on the object obj. It is also possible to optionally include additional constraints on the types of objects and scope of actions by adding $(field\_type, value)$ tuples.

- **Medium Security Policy Language (MSPL) [11, 17]:** MSPL uses an XML structure for representing statements, and contains a **meta-model** in the form of an **XML schema** specifying the elements

of a statement, including **rules**, **conditions**, **actions**, and **policies**. It is organized based on **capabilities**, which are defined as basic features that can be configured in order to enforce a security policy.

In order to select the most appropriate policy language to be employed by the REWIRE Formal Verification component for the expression of the policies so that they can be applied by the Policy Orchestrator, the selected language needs to (i) have **unambiguous structure**, (ii) be **robust enough to describe the security requirements and constraints**, and (iii) support **translation of policies into the actions to be enforced by the Policy Orchestrator**, and (iv) be **generic enough to support various types of actions defined by the available security controls**.

Considering the above, languages such as XCAML, CIM, SDL, and SPL are not appropriate for REWIRE, as they ar related to specific contexts (e.g., access control), and would require significant effort to be adapted to the REWIRE framework in order to achieve the required level of expressiveness. Conversely, HSPL provides a high-level representation which, while providing a readable and understandable representation, does not achieve the level of granularity in policy expression required by REWIRE. Thus, **MSPL is selected as the REWIRE policy language**, as it fulfils the aforementioned requirements pertaining to the structural specification of policies for the Policy Orchestrator.

6.4 MSPL Vocabulary

MSPL is a policy expression language with a **medium level of abstraction**, meaning that while it remains independent from the specifications and technologies employed, it also provides a high level of granularity in policy expression compared to languages such as HSPL. Here, we provide details on the vocabulary associated with MSPL, which will guide the expression of policies in a manner that can be processed by the Policy Orchestrator.

6.4.1 MSPL Structure

In general, a policy expressed in MSPL consists of two core components, namely **ITResource** and **Configuration**, and the latter may contain **Capability** and **Configuration Rule** subcomponents. Here, we provide definitions and descriptions of each of these components.

- **ITResource:** This is the central element of MSPL which represents the configuration of a software or hardware asset, and contains a definition of *configurations, dependencies, priorities*, and *security enabler candidates*. It essentially contains a structured description of the operational landscape of the asset, including dependencies with other assets or processes, as well as the list of security enablers (e.g., attestation tasks, software updates) that can be deployed in order to protect an asset against identified vulnerabilities.
- **Configuration:** This element contains the abstractions that need to be provided to the REWIRE Policy Orchestrator, so that the final security enablers (e.g., attestation tasks, software updates) can be provided as output. Specifically, it contains a set of *capabilities* and *configuration rules*, which are used in order to describe the attributes and constraints on the software or hardware assets as follows:
  - ✓ **Capability:** Capabilities in the context of MSPL refer to the ability of an asset to perform specific operations. These, in turn, are linked to specific actions that need to be performed in order to express these capabilities. For instance, if an asset has the capability to perform an operation, this may be expressed and represented through the implementation of a software function. Also, note that *capabilities are connected with the identified threats and vulnerabilities* that need to be addressed. For instance, if a vulnerability of a device can be addressed via

an attestation action, this is connected to the capability of the device to perform that attestation action.

✓ **Configuration Rule:** These refer to an *abstract set of configuration settings for a specific capability*, which consist of: (i) **Conditions**. These describe predicates that, when satisfied, trigger a set of corresponding actions. For example, if a software process is installed on a target asset, this refers to the conditions that need to be fulfilled for the action to be executed. (ii) **Actions**. These refer to the actions that need to be executed when the corresponding Conditions are satisfied, such as attestation actions in response to an identified vulnerability. These actions also include attributes, such as *whether the task is an edge or infrastructure service*, the **execution time** of the action or any time restrictions, any **privacy requirements**, etc.

6.4.2 MSPL Component Relationships

Having defined the structure and components participating in the MSPL expression language, we can now provide a high-level representation of the MSPL schema employed in REWIRE considering the relationships between components, attributes, and fields. This structure, depicted in Figure 6.3, can be summarized as follows:

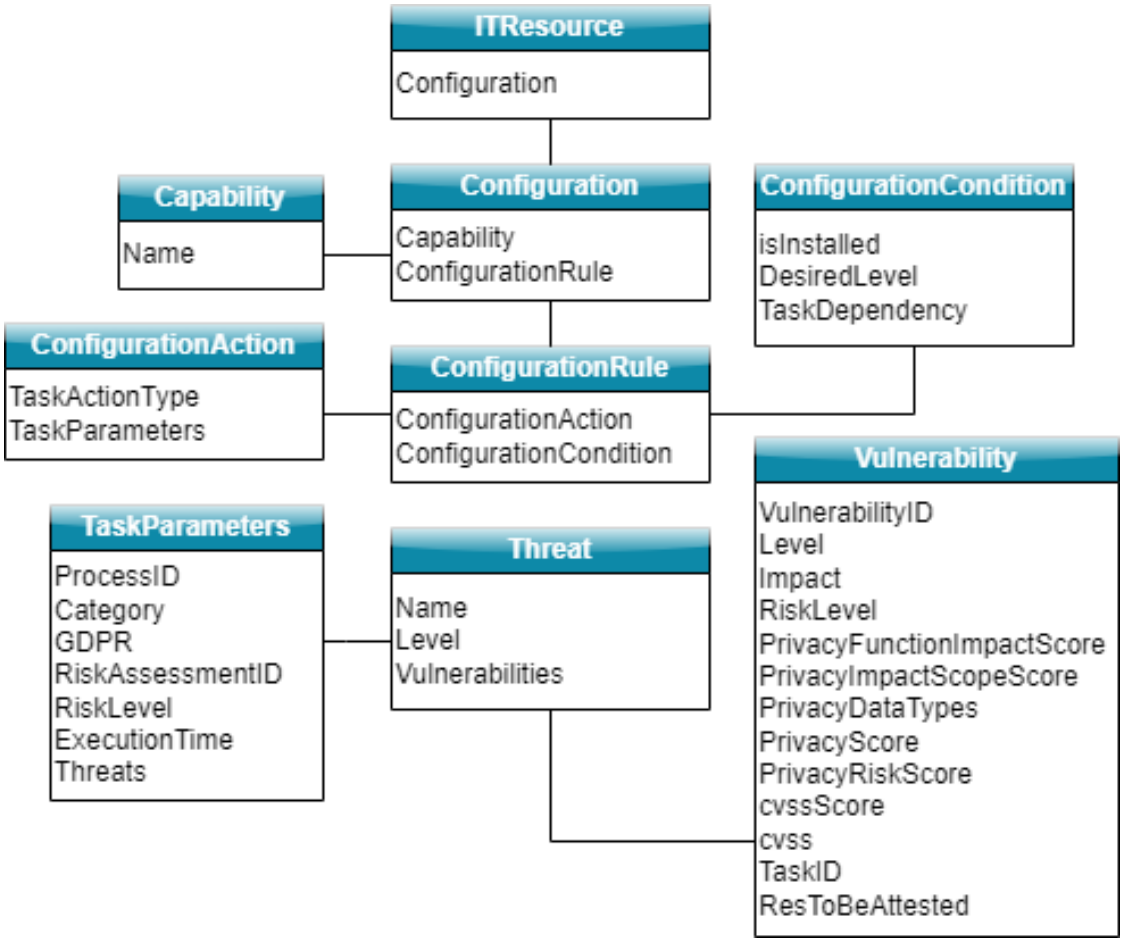

Figure 6.3: REWIRE MSPL Schema.

- The main building block of the MSPL schema is the **ITResource**, which as aforementioned, can represent a hardware or a software asset. Each asset is defined by its **Configuration**, containing all information regarding the asset, the actions it can perform, its properties, etc.

- The **Configuration** block contains the **Capability** field, speficying the types of capabilities characterizing the asset, which may be a **TaskCapability** or **DeviceCapability** depending on whether it refers to a task or device capability, respectively. The Configuration block also contains the **ConfigurationRule**, containing the policy abstractions that need to be defined within the REWIRE framework in order to generate the actual security policies.
- The **ConfigurationRule** block contains two sub-blocks, corresponding to the Condition and Action properties for the definition of the actions that need to be taken as part of the defined policy:
- ✓ **ConfigurationCondition:** Contains the conditions or prerequisites for the execution of a task, as well as a depiction of interdependencies with other tasks, and other tasks that need to be executed before it.
  - ✓ **ConfigurationAction:** Contains the type of action that needs to be executed, including the parameters needed for the execution of this action. The parameters are defined in the **TaskParameters** field, and contain information on the threats affecting the asset and the risk level associated with them.
- The **Threat** field contains the threats identified for the particular asset. Also, the **Vulnerability** field within the **Threat** field defines the risk and impact level of the vulnerability associated with the threat, as well as privacy and CVSS scores. It also contains the **TaskID** and **ResToBeAttested**, specifying the action to be taken for mitigating the vulnerability, as well as the resources that need to be attested. Note that, in the context of REWIRE, the contents of the TaskID field may refer to various types of actions, such as **attestation tasks**, **software updates**, or **device migration**, which may be used in order to mitigate vulnerabilities. Types of actions offered by the REWIRE framework are given in Table 7.3, and in Chapter 7 examples of MSPL policy models for actions offered by the REWIRE framework are provided.

Chapter 7

Policy Modelling in REWIRE

In this Chapter, we go into further detail regarding the construction of policies using the **Medium Security Policy Language (MSPL)** in the context of REWIRE, by expanding upon the fields that may be contained in the MSPL structures outlined in the previous Chapter. Then, we provide details on the types of policies that may be constructed in REWIRE, considering the types of actions that need to be executed. In Table 7.1, we provide a high-level overview of such actions that may be specified by the policies, as well as the information that needs to be dictated by the policies.

Action	Description and Parameters
Attestation task	Specifies the types of attestation tasks to be executed, such as Configuration Integrity Verification (CIV) and Control Flow Attestation (CFA). The corresponding policy specifies the device it refers to and the runtime properties to be attested defined by the Formal Verification process, and can also specify the periodicity of the attestation action, which may be synchronous (e.g., execution at regular time intervals), or asynchronous (e.g., attestation before the execution of a binary to verify its correctness, or to verify the correctness of a software update patch after it has been deployed).
Traceability through hooks	REWIRE employs monitoring hooks deployed at the binaries of a device in order to control the level of tracing, so that the activation and deactivation of hooks may be used for the adjustment of the level of tracing. For instance, in case it is needed to identify the execution path for the identification of a malicious attack, it is possible to activate additional tracing hooks to increase the level of granularity in the tracing process.
Application State Migration	Enclave Migration in the context of REWIRE allows compromised services to be migrated securely to a new enclave, thus ensuring uninterrupted execution. Thus, a security policy may specify that, if a device has been deemed compromised, a device state that is known to be correct and trustworthy can be migrated to a new device.
Software Update	In case a vulnerability has been identified, there may need to be a software update deployed to the device in order to mitigate and address the identified vulnerability. Thus, the corresponding policy may specify the current software version, as well as the target software version that needs to be installed through the SW/FW Verification component.
Privacy Level	This type of policy specifies the properties of keys that need to be instantiated during runtime, controlled via the Security Management Layer. For instance, it may need to be specified whether ephemeral keys are needed, whether the same domain is considered in all communications, as well as privacy considerations (i.e., if the Attestation Key needs to be elevated to a DAA Key for granting privacy preservation capabilities).

Table 7.1: Actions that can be enforced to support secure lifecycle management in REWIRE

7.1 MSPL Policy Modelling

As aforementioned, **MSPL is a security policy language with a medium level of abstraction**. It provides a high level of granularity in the expression of actions, resources, and properties to be attested, while simultaneously is still independent on the specifications and technologies employed. In the following, we focus more on the details regarding the structure of policies using MSPL, focusing on the most pertinent fields regarding its application in the context of REWIRE.

As outlined in the previous Chapter, an MSPL policy model consists of two main components: **ITResource** and **Configuration**, the latter of which can contain **Capability** and **Configuration Rule** components. Here, we provide detailed definitions for these components.

7.1.1 ITResource

ITResource is the central element of MSPL and represents a configuration for a *hardware* or *software* asset. An ITResource contains **configurations, dependencies, priorities,** and **security enabler candidates**, define in a structured format the operational landscape of the asset in relation to its configuration, its own software process, or its dependencies with other assets or processes. It also defines the security enablers (e.g., attestation tasks, software update, software igration) that can be deployed for the protection of the asset against the identified vulnerabilities. A complete description of the ITResource element and its hierarchical structure has been provided in the SECURED project [29].

As it will be shown in the following, in the context of REWIRE, the ITResource field is used for each software and hardware asset participating in the Formal Verification process. The goal of this definition is to provide the Policy Orchestrator with the required information for the deployment of the appropriate actions through the Facility Layer of the device, including information regarding the capabilities of each device, such as the identification number of the device, the software executed on the device, and the periodicity of the defined tasks.

7.1.2 Configuration

The **Configuration** element contains the **abstractions** that need to be defined in the policy as provided by the Formal Verification component, so that it can specify the final security enablers (attestation tasks, SW update, migration) to be dictated by the policy. This essentially constitutes the answer to the security objective questions, which have been posed an the beginning of Chapter 6.

The Configuration field contains a set of **capabilities** and **configuration rules**. In the context of REWIRE, this field represents the capabilities of the devices or assets that have been defined as an ITResource. For instance, the capability of a software asset, referred to as **TaskCapability**, may contain a short description of what the specific software function does. Conversely, for hardware assets, the **DeviceCapability** contains information such as a unique device identifier, or the positioning of the asset within the domain infrastructure.

Configuration rules internally contain two elements: **ConfigurationRuleAction** and **ConfigurationCondition**, used in order to describe the attributes of the assets that may be attested during runtime.

7.1.2.1 Capability

This field refers to the **Capabilities** of a software or hardware asset, i.e., the ability of the asset to perform specific operations. These capabilities are linked to specific actions that should be implemented in order to express these capabilities that the asset needs to perform, as dictated by the policies. To this end, in order to express a capability, a corresponding **software function should be implemented**, that represents the capability of the device to perform the specified action.

Capabilities can either refer to Tasks or Devices, respectively, as shown in Table 7.2. Specifically, **Tasks** refer the to the actions to be executed (e.g., attestation, software update), or the attestation tasks to be executed for protecting against identified threats, as well as the conditions placed on these actions. On the other hand, **Devices** are considered as a statically configured resource, and each edge device is characterized by a set of configuration attributes, including a **unique device identifier**, and the **current security level** of the device based on the vulnerabilities that have been identified by the Risk Assessment Engine.

Also, note that the capabilities correspond to the **threats and vulnerabilities** identified for each asset by the Risk Assessment Engine. Thus, the vulnerabilities of each asset are linked to the capabilities that a particular asset has in order to address these vulnerabilities. For example, if a vulnerability can be addressed by executing an attestation task, the capability of the device refers to the operation that can be performed by this asset, i.e., the corresponding attestation scheme (CFA, CIV) and the parameters that it should be configured with (e.g., level of granularity in tracing), in order to address this vulnerability and protect the asset.

Capability	Description	Use case example
Task	Attestation tasks	Control Flow Attestation task, Configuration Integrity Verification task
Device	Edge devices	Raspberry pi 3 model B

Table 7.2: Capabilities in REWIRE.

7.1.2.2 Configuration Rule

Configuration rules are objects that may contain an **abstract set** of configuration settings for a specific capability, as previously defined. Specifically, a configuration rule consists of **conditions** and **actions**, defined as follows:

- **Conditions**: These describe predicates that trigger a set of actions when they are fulfilled. For example, if a software process is installed on a hardware asset, this contains conditions conditions that should be valid so that the software can be executed.
- **Actions**: Contains specific actions that should be executed for the validation of the attributes to be attested during runtime as defined by the Formal Verification Component, along with the asset where the task needs to be execution, the **periodicity** of the task (synchronous or asynchronous), or any other information, such as time restrictions (important for safety-critical operations where a decision needs to be made in near real-time), **Privacy** Requirements, etc. These actions are executed when the conditions are satisfied.

In the context of REWIRE, in case the Risk Assessment component has identified a zero-day software vulnerability, the configuration **Conditions** may include the specific device where the software is installed. The **Actions** would refer to the specific functions that have been implemented in order to mitigate this vulnerability, such as a software update or a migration action to a different device. In addition, while capabilities require distinct configurations, the MSPL model provides a hierarchical approach that structures actions and conditions placed on assets.

In REWIRE, we consider two main actions to be carried out over capabilities that are aligned with the two main flow of actions for which MSPL policy modelling is used; (i) **for depicting the inputs given to the Policy Orchestrator**, and (ii) **for expressing the operations that need to be carried out through the Facility layer**:

- **CONSTRAINT_SOLVE.** This action denotes a list of assets and their attributes, as well as conditions that must be met in order to ensure the safety of the relevant assets in a SoS ecosystem.
- **EXECUTE.** This action denotes execution of tasks on devices, and provides a list of tasks and their attributes and conditions on devices where tasks should be executed, as well as information on the periodicity of the speficied tasks.

Finally, to represent tasks in MSPL and all of the identified risk information, we consider the following attributes based the information provided by the Risk Assessment component of REWIRE:

- **Task Identifier.** A unique identifier for the software asset;
- **Category.** Indicates if the task should be executed on an edge device or at the backend infrastructure of REWIRE;
- **GDPR.** Flag indicating whether the General Data Protection Regulation (GDPR) should be considered for this asset, specifically referring to operational data shared by the devices, extending to properties such as *anonymity, unlinkability, untraceability or unobservability*. For instance, this may refer to if an Attestation Key (AK) needs to be elevated to a Direct Anonymous Attestation (DAA) key for privacy purposes;
- **Risk Assessment Identifier.** An identifier provided by the risk assessment component in order to link the policy applied for the task to a specific risk indication;
- **Risk Level.** The current risk level of the task as calculated by the Risk Assessment component, based on the identified vulnerabilities and the risk calculation process;
- **Execution Time.** The expected latency of the execution of the task;
- **Threats.** List of the vulnerabilities identified by the Risk Assessment component. Table 7.4 depicts an indicative list of vulnerabilities considered in REWIRE.

Representing risk attributes for hardware assets (devices) in MSPL is similar with the representation of software assets (tasks), and includes attributes for **Risk Assessment Identifier**, **Risk Level** and **Threats**. Furthermore, similarly to tasks, devices are represented using a unique **Device Identifier** that is used to refer to a specific device.

Task ID	Task Description
0	Control Flow Attestation (CFA)
1	Configuration Integrity Verification (CIV)
2	Swarm Attestation
3	Direct Anonymous Attestation
4	Static attestation
5	Software Update
6	Device Migration
7	General computation task

Table 7.3: Task types and their respective identifiers in REWIRE.

Vulnerability ID	Description
CAPEC-242	Code injection
CAPEC-175	Code Inclusion
CAPEC-29	Leveraging Time-of-Check and Time-of-Use
CAPEC-186	Malicious Software Update
CAPEC-63	Cross-Site Scripting
CAPEC-151	Identity Spoofing
CAPEC-1000	Mechanisms of Attack

Table 7.4: Indicative list of vulnerabilities considered in REWIRE.

The **ConfigurationCondition** element dictates the conditions that should be fulfilled in order to execute the actions specified by the policy, such as requiring that a task should be executed on a specific device. In REWIRE we consider the following conditions, which can be placed on software assets:

- **DeviceCondition:** A device identifier that specifies which device the task should be executed on.
- **DesiredLevel:** The required security level that must be achieved prior to executing the specified tasks (e.g., attestation, software update, migration) on the device.
- **TaskDependency:** A list of tasks that should be executed before the task specified by the policy.
- **isInstalled:** A list of device identifiers that indicates the devices that this task should be executed on.

Similarly to software assets, in REWIRE we also consider conditions for hardware assets. Specifically, conditions for devices only include the **DesiredLevel**, which refers to the desired risk level of the device after it has executed the specified tasks.

7.2 Policy Models Definition

Here, we combine all the information presented in the previous sections, in order to formulate and structure the policy models based on the MSPL syntax presented in Section 6.4, so that all necessary information is included. The methodology presented is used both to represent policies that are provided as output of the Formal Verification component to the Policy Orchestrator, so that the Facility Layer can handle the specified tasks, considering any imposed constraints or periodicity requirements. Note that the fields included in this representation have been selected, so that all the required policy abstractions that were presented in Chapter 7.1 are achieved.

- **ITResource**: This is essentially the central element of MSPL, which specifies whether the listing corresponds to a **hardware** or a **software** asset, and represents a configuration of the specified asset. It contains a definition of **configurations**, **dependencies**, **priorities** and **enabler candidates**.
- **configuration**: In this field, we define an abstract representation of configuration settings which must be valid, in order to enable the execution of the required actions. The configuration field contains the **capability** and **configurationRule** fields.
  - ✓ **capability**: This field specifies what is the capability of the corresponding ITResource. Specifically, the capability of software assets refers to the tasks that can be executed by this particular software, and we use the *TaskConfiguration* label. Conversely, for hardware assets, we refer to the capabilities of the particular device, therefore the *DeviceCapability* label is sued.
  - ✓ **configurationRule**: Recall the policy **abstractions** for the conditions and constraints that need to be defined within REWIRE and should be considered for the formulation of the policies as output of the Formal Verification component. These are the abstractions which were referred to and analyzed throughout Section 7.1, and answer the questions posed in regards to the security objectives. This field contains the abstractions that need to be contained within a policy listing. There are two types of *configurationRule*: *configurationRuleAction* and *configurationCondition*.

Next, we refer to the fields contained within the **configurationRuleAction** tag, which contains information regarding the action that should be executed, as dictated by the particular policy, as well as the threats and vulnerabilities that this action is associated with. Note that the namespace field contained within this XML tag is intended to show if this action applies to a task (*TaskAction*) or a device (*DeviceAction*).

- **TaskActionType** or **DeviceActionType**: This field specifies the type of action that is specified by the policy. Here, as we refer to policies that are provided as input to the Policy Orchestrator for the execution of specific actions, we use the *EXECUTE* attribute for signifying that there are actions that should be taken by the particular asset.

- **TaskParameters** or **DeviceParameters**: The elements contained within this XML tag denote all parameters specified by this policy.
- **ProcessID** or **DeviceID**: An identifier for the process or device targeted by the policy, and can be a numerical value or a name.
- **category**: Shows if this particular process runs on the edge (*Edge Service*) or the infrastructure (*Infrastructure Service*).
- **GDPR**: This tag contains a boolean value (true or false) specifying whether there are privacy requirements or not.
- **RiskAssessmentID**: Refers to the identification number of the particular task or device in the Risk Assessment graph.
- **RiskLevel**: Contains the Risk Level of the specified software or hardware asset, and is provided as output of the Risk Assessment process. This can take any of the values VL (Very Low), L (Low), M (Moderate), H (High) or VH (Very High).
- **ExecutionTime**: Refers to software assets, and specifies the execution time of the asset.
- **Threats**: This XML tag contains all threats identified for the particular software or hardware asset. Further information regarding this field will be provided below.

The elements contained within the **Threat** tag denote the various aspects and characteristics of each threat identified for the particular asset, and exists for both software and hardware assets. *Note that the information in this field has been given as an output from the Risk Assessment Engine.* These elements are as follows:

- **Name**: A name label, which refers to a categorization of the threat given by the Risk Assessment Engine.
- **Level**: Each threat is characterized by its level according to its severity, and can take the values VL, L, M, H, VH.
- **VulnerabilitiesList**: This tag specifies the mapping of the threats to a list of vulnerabilities. Each vulnerability is denoted by the **Vulnerability** tag.
- **VulnerabilityID**: Each vulnerability is denoted by its CAPEC identifier. This is an open source threat database, where various known vulnerabilities are listed. Some examples of CAPEC vulnerabilities considered in REWIRE are listed in Table 7.4.
- **Level**, **Impact**, **RiskLevel**: These fields can range from VL to VH, and contain the Level, Impact and Risk Levels, respectively, as identified by the Risk Assessment component for the particular vulnerability.
- **PrivacyFuncImpactScore** and **PrivacyImpactScopeScore**: Specify the level and scope of the impact of the corresponding vulnerability on the privacy aspects of the system.
- **PrivacyDataTypes**: Denotes the data structure outputs that should have privacy preserving properties.
- **PrivacyScore** and **PrivacyRiskScore**: Specify the risk level to privacy associated with this particular vulnerability.
- **cvssScore**, **cvss**: Denote the **Common Vulnerability Scoring System (CVSS)** score and level for the specified vulnerability, which has been identified by the Risk Assessment component.

- **TaskID**: A vulnerability may be associated with a task responsible for its mitigation (e.g., attestation, software update, device migration). Each task that is employed within REWIRE is characterized by an **Task ID**. A list of these tasks and their corresponding IDs is given in Table 7.3.
- **AttestationTaskParameter**: Denotes the parameters that are required for identifying the attestation task that should be executed for the mitigation of the particular vulnerability.
- **ResToBeAttested**: Refers to the resources that should be attested in order to address the particular vulnerability. In this field, we may provide the **trusted reference values** for the attestation, meaning the correct paths that are expected to be executed, so that it can be verified that the actual execution paths correspond to the given trusted reference values.
- **VerifierID:** Refers to the ID of the device (thus, it exists only for the policies depicting information for hardware assets) that will be acting as the Verifier of an attestation task. Recall that an attestation task is a remote process between a Prover and a Verifier, who aims to verify the correctness of the Prover specified by its Device ID. Essentially, devices communicate with each other in the context of the envisioned service graph chain, which also specifies the mapping between Provers and Verifiers.

Next, we refer to the contents of the **configurationCondition** tag. For software assets, this tag contains the conditions that should be fulfilled so that the particular software can run. However, this field is also applicable to hardware assets, and contains the desired conditions of the particular asset.

- **isInstalled**: Only applicable to software assets. Denotes the hardware assets on which the software asset is installed and executed.
- **DesiredLevel**: Applicable to software and hardware assets from the software or device. Determines the desired level of safety and security that is required, and is represented by using the VL to VH scale that was also used in the aforementioned fields. The Risk Assessment engine should take into consideration this value in tandem with the vulnerability levels in order to generate the risk graph.
- **DeviceCondition**: Provides any other complementary information that may be required to determine the condition on the device where the software is executed.
- **TaskDependencyList** and **TaskDependencies**: This field specifies any dependencies with other tasks. This means that the software processes contained within this list should be executed before the task specified by this policy.

Note that, while one listing is created for each hardware and software asset, it is possible for one hardware asset to execute multiple software assets. This is reflected in software asset listings by using the **isInstalled** field, which depicts the connections between them. This representation also depicts the threats and vulnerabilities for **all layers of the application stack**, from low-level memory-related vulnerabilities to network threats.

In the following, we provide examples of policy models used in the context of REWIRE for executing actions at assets specified as output of the Formal Verification engine, namely **attestation policies** and **software update policies**.

7.2.1 Attestation Policy

Next, we provide the MSPL representation of policies depicting attestation tasks to be executed by the specified devices, as well as parameters corresponding to the different attestation types, which will be passed the Facility Layer for executing the attestation task.

7.2.1.0.1 Control Flow Attestation (CFA): This type of attestation is used for verifying the correctness of the control flow of a software process by collecting the required control flow traces, and comparing them against a list of valid control-flows (*trusted reference values*) to be validated against, representing the expected system behaviour when executing the process of interest. These traces represent the valid control flows of the program, including the order of execution of functions and other basic blocks during the implementation of the target process.

A CFA attestation policy example is given in Listing 7.1 in the example, the task attributes metadata used to refer to it such as a unique identifier 0 denoted by **ProcessID**, and other metadata attributes such as **category** and **GDPR**. Furthermore, the **TaskParameters** field contains a **RiskLevel** attached to it which has been marked as H (High), due to the vulnerability *CAPEC-242* that has been detected for it, as shown in the **VulnerabilitiesList** defined within the **TaskParameters** field of the policy. Each Vulnerability of this list has a risk **Level** associated with it, and various privacy attributes attached to it. In this example, the vulnerability *CAPEC-242* relates to code injection, which can be exploited by an adversary to execute malicious code on the edge device.

As the CFA attestation enabler has been identified for lowering the risk associated with this vulnerability, the **TaskID** has a value of 0, which matches the CFA attestation task, as denoted in Table 7.3. The attestation task parameters in this case also include the software for which to track the traced control flows. and the process name is given in the **AttestationTaskParameter**, which in this example refers to **SecureUpdate**, referring to a secure update process. Also, the input to the Verifier is given by the **ResToBeAttested** element, which in this example is a pointer to pre-computed secure control flow graph matching this **SecureUpdate** process. In the *ControlFlowParameters* field, the valid control flows are specified, meaning the $main$ function of the $software\_process$ binary is executed first, and then a $printf$ function is invoked. Then, an additional permitted control flow is defined for the execution of the same binary, specifying the execution of the $main$ function first, but afterwards specifying the execution of a $write$ operation to a buffer as part of the $libc$ binary.

The policy also includes a set of constraints within the **configurationCondition** tag. In this example, the constraints include lowering the risk level for executing the task to Low, denoted as L in the **DesiredLevel** tag. The constraints also include the **TaskDependencyList**, which contains a list of prerequisite process identifiers, specifying tasks that should be executed before the CFA attestation. This identifier is given in the **PolicyID** field, and the policy in this example has a value of 0. This means that process 0 cannot be scheduled until processes 5 and 15 have been executed. Finally, the constraints can include which hardware assets the task can be executed on. In this example, this device has an identifier of 0. The last constraint is **isInstalled**, which shows on which devices the process must be installed (device 0 in this example). This constraint is expected and must be set in the operational policy model, as a process needs to be installed on a device in order to be executed.

Listing 7.1: Example of a CFA attestation policy expressed in MSPL

```
<ITResource>
    <configuration xsi:type="RuleSetConfiguration">
        <capability xsi:type="TaskCapability">
            <Name>TaskCapability</Name>
        </capability>
        <configurationRule>
            <configurationRuleAction xsi:type="TaskAction">
                <TaskActionType>EXECUTE</TaskActionType>
                <TaskActionParameters>
                <TaskParameters xsi:type="TaskParameters" />
                    <ProcessID>0</ProcessID>
                    <category>Infrastructure Service/Edge Service</category>
                    <GDPR>false</GDPR>
```

```
            <RiskAssessmentID>1714</RiskAssessmentID>
            <RiskLevel>H</RiskLevel>
            <ExecutionTime>15</ExecutionTime>
            <Threats xsi:type="TaskThreats">
                <Threat>
                    <Name>DEFAULT-I</Name>
                    <Level>VH</Level>
                    <VulnerabilitiesList>
                        <Vulnerability>
                         <VulnerabilityID>CAPEC-242</VulnerabilityID>
                         <Level>VL</Level>
                         <Impact>VH</Impact>
                         <RiskLevel>H</RiskLevel>
                         <PrivacyFuncImpactScore>0
                                            </PrivacyFuncalImpactScore>
                            <PrivacyImpactScopeScore>0
                                             </PrivacyImpactScopeScore>
                            <PrivacyDataTypes>[dat1, dat2]
                                             </PrivacyDataTypes>
                            <PrivacyScore>0</PrivacyScore>
                            <PrivacyRiskScore>0</PrivacyRiskScore>
                            <cvssScore>6</cvssScore>
                            <cvss>H</cvss>
                            <TaskID>0</TaskID>
                            <AttestationTaskParameter>
                                SecureUpdate
                            </AttestationTaskParameter>
                            <ResToBeAttested>
                                ValidControlFlowGraphsPointer
                            </ResToBeAttested>
                        </Vulnerability>
                    </VulnerabilitiesList>
                </Threat>
            </Threats>
            <ControlFlowParameters xsi:type="ControlFlowParameters">
                <ProcessID>software_process</ProcessID>
                <ValidControlFlows>
                    <ControlFlowList>
                        <Block>main,software_process</Block>
                        <Block>printf,libc-2.27.so</Block>
                    </ControlFlowList>
                    <ControlFlowList>
                        <Block>main,software_process</Block>
                        <Block>write,libc-2.27.so</Block>
                    </ControlFlowList>
                </ValidControlFlows>
            </ControlFlowParameters>
        </TaskActionParameters>
    </configurationRuleAction>
    <configurationCondition xsi:type="TaskCondition">
        <isInstalled>[0,1]</isInstalled>
```

```
                <DesiredLevel>L</DesiredLevel>
                <DeviceCondition>0/<DeviceCondition>
                <TaskDepndencyList>
                    <TaskDepndency>5</TaskDepndency>
                    <TaskDepndency>15</TaskDepndency>
                </TaskDepndencyList>
            </configurationCondition>
            <Name>"TaskRule"</Name>
        </configurationRule>
        <Name>"TaskConfiguration"</Name>
    </configuration>
</ITResource>
```

7.2.1.0.2 Configuration Integrity Verification (CIV): The CIV attestation process entails the verification of the correctness of an attribute of the specified binary, specified by the Formal Verification component to require runtime attestation (and cannot be verified statically during design-time).

In Listing 7.2, we provide an example of a policy specifying the execution of a CIV action, considering the possibility that a CIV attestation enabler is used instead for the mitigation of the same vulnerability defined in the CFA example of Listing 7.1. The expression of this policy is similar, with some key differences. Specifically, the **TaskID** field is set to 0, corresponding to the ID of the CIV enabler, as specified in Table 7.3. In addition, **ResToBeAttested** field specifies the attribute to be attested, as specified by the Formal Verification component as an attribute that requires runtime attestation. Finally, the configuration parameters of the CIV attestation enabler specified in the **ConfigurationIntegrityParameters** field within the **TaskActionParameters** is a list of expected hash values for code pages in libraries and binaries loaded in the device memory. Each hash is characterized by the corresponding source (library or binary identifier), the page index, and the corresponding hash value as computed with the SHA-256 cryptographic hash algorithm.

Listing 7.2: Example of a CIV attestation policy expressed in MSPL

```
<ITResource>
    <configuration xsi:type="RuleSetConfiguration">
        <capability xsi:type="TaskCapability">
            <Name>TaskCapability</Name>
        </capability>
        <configurationRule>
            <configurationRuleAction xsi:type="TaskAction">
                <TaskActionType>EXECUTE</TaskActionType>
                <TaskActionParameters>
                <TaskParameters xsi:type="TaskParameters" />
                    <ProcessID>0</ProcessID>
                    <category>Infrastructure Service/Edge Service</category>
                    <GDPR>false</GDPR>
                    <RiskAssessmentID>1714</RiskAssessmentID>
                    <RiskLevel>H</RiskLevel>
                    <ExecutionTime>15</ExecutionTime>
                    <Threats xsi:type="TaskThreats">
                        <Threat>
                            <Name>DEFAULT-I</Name>
                            <Level>VH</Level>
                            <VulnerabilitiesList>
```

```
                        <Vulnerability>
                         <VulnerabilityID>CAPEC-242</VulnerabilityID>
                         <Level>VL</Level>
                         <Impact>VH</Impact>
                         <RiskLevel>H</RiskLevel>
                         <PrivacyFuncImpactScore>0
                                          </PrivacyFuncalImpactScore>
                            <PrivacyImpactScopeScore>0
                                           </PrivacyImpactScopeScore>
                            <PrivacyDataTypes>[dat1, dat2]
                                           </PrivacyDataTypes>
                            <PrivacyScore>0</PrivacyScore>
                            <PrivacyRiskScore>0</PrivacyRiskScore>
                            <cvssScore>6</cvssScore>
                            <cvss>H</cvss>
                            <TaskID>1</TaskID>
                            <ResToBeAttested>
                                ConfigurationIntegrityPointer
                            </ResToBeAttested>
                        </Vulnerability>
                    </VulnerabilitiesList>
                </Threat>
            </Threats>
            <ConfigurationIntegrityParameters
                xsi:type="ConfigurationIntegrity">
                <ValidHashes>
                    <Hash>libc-2.27.so,0,
                        9ac411bef0479d2ebd6ee2126243b8b4a5d7c3d4ed5f2899
                    </Hash>
                    <Hash>software_process,
                        0721718c63188fe6ed74462222b4af4177e518e3ac2
                    </Hash>
                </ValidHashes>
            </ConfigurationIntegrityParameters>
        </TaskActionParameters>
    </configurationRuleAction>
    <configurationCondition xsi:type="TaskCondition">
        <isInstalled>[0,1]</isInstalled>
         <DesiredLevel>L</DesiredLevel>
         <DeviceCondition>0/<DeviceCondition>
        <TaskDepndencyList>
            <TaskDepndency>5</TaskDepndency>
            <TaskDepndency>15</TaskDepndency>
        </TaskDepndencyList>
    </configurationCondition>
    <Name>"TaskRule"</Name>
  </configurationRule>
  <Name>"TaskConfiguration"</Name>
 </configuration>
</ITResource>
```

7.2.2 Software Update Policy

Here, we provide an example for a policy dictating a **software update** process, including the execution of a CIV task in order to verify the correctness of the software update patch after it has been installed. Thus, this example also demonstrates a policy with dependencies between specified tasks, as the installation of the software update is a prerequisite for the execution of the CIV process, as the installation process needs to be complete in order to verify whether it has been successfully installed. Listing 7.3 depicts an MSPL example representing the aforementioned actions and includes two rules, which correspond to the Software Update and CIV tasks. It also includes information regarding the actions to be executed and the conditions placed on executing these actions on the appropriate edge device.

The first task has an identifier of 0, given in the **ProcessID** field. Also, it refers to a **Software Update** task, which is denoted by the value 5 in the **TaskType** attribute. The full task type representation is available in Table 7.3. Also, the **ExecutionTime** of the task has been specified as 15. Regarding the conditions for its execution, shown in the **configurationConditions** tag, this task should be executed on the edge device with a **DeviceID** of 5, as denoted by the **DeviceCondition** tag.

The second task has a **ProcessID** of 5 and is a Configuration Integrity Verification task, since its **TaskType** attribute has a value of 1 and should be verified by the edge devices with ID 5 as denoted by the **VerifierID** tag. Also, the task with a **ProcessID** of 0 (i.e., the Software Update task) should first be executed on the same device as a prerequisite, as shown in the **TaskDependency** field. The **isCNF** attribute is set to false, which means all conditions must be satisfied for the constraint to be resolved. In addition, the **ExecutionTime** for this task has been specified as 50. Also, the resource to be attested in this task has been specified as **ValidControlFlowPointer_Task0**, as noted in the **ResToBeAttested** tag.

Listing 7.3: Software update policy example

```
<ITResource>
    <configuration xsi:type="RuleSetConfiguration">
        <Name>"TaskConfiguration"</Name>
        <capability xsi:type="TaskCapability">
            <Name>TaskCapability</Name>
        </capability>
        <configurationRule>
            <configurationRuleAction xsi:type="TaskAction">
                <TaskActionType>EXECUTE</TaskActionType>
                <TaskActionParameters>
                <TaskParameters xsi:type="TaskParameters" />
                    <ProcessID>0</ProcessID>
                    <TaskType>5</TaskType>
                    <ExecutionTime>15</ExecutionTime>
                </TaskActionParameters>
            </configurationRuleAction>
            <configurationCondition xsi:type="TaskCondition">
                <isCNF>false</isCNF>
                <DeviceCondition>5</DeviceCondition>
            </configurationCondition>
            <Name>"Task0Rule"</Name>
        </configurationRule>
        <configurationRule>
            <configurationRuleAction xsi:type="AttestationAction">
                <TaskActionType>EXECUTE</TaskActionType>
                <TaskActionParameters>
                <TaskParameters xsi:type="TaskParameters" />
```

```
                <ProcessID>5</ProcessID>
                <TaskType>1</TaskType>
                <ExecutionTime>50</ExecutionTime>
                <VerifierID>5</VerifierID>
                <ResToBeAttested>ValidConfigurationIntegrityPointer_Task0
                                                </ResToBeAttested>
            </TaskActionParameters>
        </configurationRuleAction>
        <configurationCondition xsi:type="TaskCondition">
            <isCNF>false</isCNF>
            <DeviceCondition>5</DeviceCondition>
            <TaskDepndencyList>
                <TaskDepndency>0</TaskDepndency>
            </TaskDepndencyList>
        </configurationCondition>
        <Name>"Task5Rule"</Name>
    </configurationRule>
</configuration>
</ITResource>
```

Chapter 8

Instantiation of AI-based Misbehaviour Detection for Reinforcing Evidence-based Trust Level Evaluation

As detailed in Chapter 2, one of the core functionalities envisioned in REWIRE is the **embodiment of trustworthiness management as a functional component of any safety-critical application:** *Provide the necessary trust extensions based on which dynamic trust relationships can be assessed and established as part of a trustworthy service-graph-chain* [7, 16]. In such complex systems, trust decisions are usually based on incomplete, conflicting, or uncertain information. This is mainly due to the **zero-trust** principle where no inherent trust or trustworthiness can underpin the interactions between any entities. Therefore, helping devices to assess the trustworthiness of incoming information and mitigate threats from untrustworthy or hostile entities is pivotal. While desired properties are usually confined to system integrity and dependability, these might also cover other crucial aspects such as privacy, robustness, availability of data, service continuity, etc. **This is where REWIRE trust extensions come to fill in the gap by putting forth an open, modular, highly portable, and RoT vendor independent security stack that exposes advanced trust, attestation and tracing services allowing for more robust security guarantees on the correct state of the target device, thus, enabling the construction of a trustworthy compute continuum**.

In this context, trustworthiness can be broadly conceived as a measure of the trustee's ability to achieve a specific task, as part of a trust relationship. In other words, the entrusted task here can be understood in terms of the expectations a trustor has from the trustee[1]. Given this, trustworthiness is defined as **the likelihood of the trustee to fulfil trustor's expectations in a given context.** Here, expectations could relate to the corrctness of data, as well as the assessor's ability to assess the correctness of data. However, expectations can also relate to the behaviour of the trustee (e.g., if the trustee is a node then in relation to the functionality of that node), the process through which the entrusted task was carried out by the trustee, and the purpose for which the task was chosen. In that case, we need to bridge the trustworthiness of data sources with expected behaviour, since there is nothing in the trustworthiness of data sources and data that would entail consistent behaviour. Different assessors (i.e., trustors) might have different rules on how to translate this to expected behaviour. So, we can do this bridging based on decision making, for example, rules or policies that could support making decisions or calculations about expected behaviour related to the data that we have collected by different data sources.

Such data sources provide the type of evidence based on which the trust characterization of the target system can occur - thus, it is imperative to be **monitored continuously and in a verifiable manner**. In the context of security-related trust properties, trust sources may include detection mechanisms, e.g., Intrusion Detection Systems, Misbehaviour Detection Systems, or mechanisms to evaluate the existence and/or enforcement of security claims such as secure boot, configuration integrity etc. In principle, the

[1] Please check APPENDINX XX for a detailed vocabulary on trust-related aspects of a "Systems-of-Systems".

received evidence can be distinguished into two main categories depending on the inferred claims. On the one hand, there is binary evidence that allows the verifier to compute whether the claim exists or not. For instance, this is the case of the secure boot certificate chain that allows a verifier to make an informed decision that the prover has executed its secure boot process. This is part of REWIRE's TCB capabilities as detailed in Section 2.3.2 and further documented in D4.3 [25]. On the other side, there is evidence that results in claims expressed in the form of a range. This is the case of some Misbehaviour Detection systems that provide a score - e.g., confidence level - about abnormal behaviour of a trustee. This diversity of the evidence outcome introduces the need for robust quantification mechanisms to enable the quantification of trustworthiness in a trust model while using heterogeneous evidence and trust sources.

This the core functionality of the **AI-based Misbehaviour Detection Engine (AIMDE)** focusing on the processing of (application-related) data towards the detection of any anomalies or inconsistencies in the bahvrioural pattern on the target device that can be used as an additional source of **trustworthiness evidence** - complementary to the attestation evidence that primarily allow for the state monitoring and management of the appliction stack running in the device and not the application data. The AIMDE is intended to function as a **Trust Source**, evaluating the benefits it can bring to the accuracy of such evidence-based trust assessment mechanisms. Compunding this issue, the AI-based Misbehaviour Detection Engine (AIMDE) is designed for deployment across two key use cases featuring mixed-criticality applications: (i) the **Smart Cities** use case and b) the **Smart Satellites** use case (which has been thoroughly described in D5.1).

Considering the above, for the evaluation of the AIMDE in terms of its applicability and effectiveness as a Trust Source, we capture two types of events:

1. **Direct identification:** This case refers to the identification of anomalies or misbehaviours, based on behaviours of the device that are known to be correct and expected. These are defined in terms of **plausibility checks**, which essentially entail the definition of expected patterns or behaviours and the evaluation of the plausibility of an event against the expected **ground truth**. This has been evaluated in D6.1 [20] in the context of the Smart Satellites use case, with the identification of anomalies in application data.

2. **Identification based on data correlations:** In this case, we aim to detect anomalies considering correlations between data. Specifically, it is possible that a set of data may not exhibit discernible patterns as inherent structures of the data (e.g., the battery or CPU usage may not conform to a specific pattern), but correlations with other types of data may be considered for identifying such patterns. For instance, in an agricultural environment in the context of the Smart Cities use case, it is possible that the temperature of the soil is correlated to its humidity level.

As aforementioned, the purpose of our evaluation activities is the positioning of the AIMDE as an additional Trust Source for assisting the trust characterization of the target system. Up until now, the evaluation activities carried out in the context of D6.1 [20] were focused on measuring the accuracy of the employed classification models through plausibility checks for datasets that possess some inherent structures, and the degree to which we can identify anomalies by translating these structures into some ground truth. This was performed in the context of the **Smart Satellites** use case, where we demonstrated a detailed evaluation of both supervised and unsupervised data for point anomaly detection (e.g., anomalies in satellite movement). Nevertheless, it cannot be ensured that such inherent structures will exist in the available data sets, thus necessitating the investigation of the correlation-based approach. In this regard, as highlighted above, *one core characteristic of the Smart Cities Use Case is the existence of such correlated types of data.*

Thus, considering all the above, the next step in the AIMDE evaluation strategy is to consider test cases with a varying level of data correlation so as to measure its impact on the confidence level of any misbehaviour detection. In the context of this use case, the AIMDE aims to proactively identify and address the diverse challenges and complexities that arise within urban environments, with a focus on agricultural

research infrastructures. Specifically, the primary end goal of implementing the AIMDE within the Smart Cities Use Case focusing on agricultural research infrastructures, is to enhance the reliability and security of the collected agro-climatic data. By proactively identifying anomalies that could indicate sensor malfunctions, environmental threats or even sophisticated cyber-attacks, the AIMDE aims to provide an early warning system for operators, enabling timely interventions, thus ensuring the continued integrity and efficient operation of critical monitoring infrastructure. Thus, this UC offers unique opportunities to leverage AI for detecting potential misbehaviour, thereby strengthening the operational integrity and bolstering security measures in such infrastructures.

8.1 Application Domain & Type of Data

As part of the Smart Cities use case, the consortium has identified test cases related to the **agricultural application domain**: Particularly focused on the agricultural investigation regarding environmental conditions of infrastructures situated near urban areas. Thus, the use case explored here fits within the broader framework of Smart Cities, where the integration of advanced technologies supports more sustainable and efficient urban systems. In this context, we investigate the application of the AIMDE in agricultural research infrastructures, which act as testbeds for environmental monitoring, precision farming, and climate-resilient crop management. They are equipped with embedded sensing platforms capable of autonomous operation in open-field environments, continuously collecting real-time data to support both research and practical decision-making.

One core property of this domain is that it provides us with the capability to leverage realistic data with a high degree of correlation, such as soil humidity, air humidity, and soil electric conductivity. As many environmental variables naturally influence each other, it is crucial to determine how effectively the AIMDE can distinguish genuine misbehaviour from normal, even though interconnected variations. Successfully detecting anomalies in in this context demonstrates the AIMDE's ability to learn and model such complex data relationships, making it more reliable against sophisticated attacks which can be designed to look like normal environmental changes.

The application domain in this instance involves the continuous monitoring of agro-environmental conditions within research infrastructures that are often located in or near urban areas. These environments are characterized by complex interactions between natural and human-influenced systems, where maintaining accurate and reliable data is essential. Monitoring efforts focus on key variables such as soil quality, microclimate, water availability, and solar exposure, all of which are critical for both experimental analysis and operational control. The integration of these infrastructures within the urban ecosystem also supports the development of sustainable agriculture models, allowing cities to adapt to food production challenges under real-world conditions.

In terms of the information gathered, the system collects data that is primarily environmental, agronomic, and infrastructural. It captures key parameters for assessing soil health—such as moisture content, temperature, and electrical conductivity—across different depths and spatial locations. The system also keeps track of atmospheric variables like external air temperature, relative humidity, wind speed and direction, and solar radiation. Light availability is measured using Photosynthetically Active Radiation (PAR), both in open-air and underwater conditions, depending on the type of experimental setup. Additional sensor outputs, including voltage levels and probe inclination, help assess the operational and physical condition of the hardware itself.

Altogether, this broad spectrum of data is essential for agricultural practices that rely on a deep understanding of environmental dynamics. It supports continuous monitoring of crop and soil conditions, enables early detection of stress factors, and allows for optimized management of irrigation and nutrient delivery. By incorporating these sensor systems into Smart City infrastructures, the use case illustrates how urban and peri-urban agriculture can benefit from intelligent, interconnected monitoring technologies that enhance resilience, efficiency, and sustainability.

8.1.1 Use of Correlated Data in AI-based Misbehaviour Detection

As aforementioned, correlated data provides an alternative to time-series-based analysis, particularly in domains where explicit temporal patterns are weak or non-existent. For instance, some application domains involve complex systems where behaviours or states are governed not by a clear sequence of events over time, but by interdependencies between measured variables. In such cases, misbehaviours or anomalous activity may not be evident from a chronological standpoint, but can emerge when analysing how variables co-vary or deviate from expected relationships. For instance, a system might function within acceptable thresholds for individual variables, but deviations in the correlation structure among variables may signal a breakdown or manipulation that would otherwise remain undetected.

These implicit dependencies can be leveraged to construct models that learn the interaction patterns among different features that may be exhibited in such domains, enabling the system to detect misalignments that suggest misbehaviour. Thus, modelling joint distributions or correlations across features, the AIMBDE can identify discrepancies that violate the learned dependencies, even in the absence of repeatable or time-based patterns. This correlation-aware approach is particularly valuable for datasets that lack temporal continuity or are too sparse to support effective time-series modelling. As a result, correlated data acts as a bridge, closing the detection gap in scenarios where direct identification of anomalies is insufficient or impossible.

Compounding all the above, we can highlight the core differences between the Direct Identification and Correlation-based Identification approaches. In the direct approach, considering that there are **inherent structures** in the dataset, we are able to define the **ground truth** against which the data will be evaluated, in order to identify any deviations from the expected behaviour of the data. However, when such structures are absent, we need to employ a correlation-based approach, wherein we try to **transfer learning from one data field from another**, so that when combined and fused, we are able to identify **anomalies of a specific type**. However, one core challenge in this regard is that **the ground truth cannot be defined in a generic/universal manner, but needs to be identified against specific threats**. Thus, while the benefit of the correlation-based approach is overcoming the gap introduced by the absence of data patterns, the drawback is the **fine-graining of the ground truth against specific vulnerabilities**.

Considering the agricultural domain in the context of the Smart Cities use case, a significant challenge in this domain lies in the difficulty of establishing definitive ground truth for certain types of anomalies. While sensors' failures/malfunctions can be clearly detectable, identifying tampered data that does not look obviously wrong compared to real data, or unusual environmental events can be complex and requires domain experts' knowledge. The AIMDE's ability to provide evaluation metrics for its anomaly detection methods becomes particularly valuable in these scenarios, allowing operators to prioritise investigations based on the likelihood of actual misbehaviour. The evaluation of the AIMDE in this UC will therefore not only focus on detection accuracy, but also on its ability to provide interpretable and reliable indicators in situations where absolute ground truth is hard to define.

8.2 Threat Model

In open-field agricultural environments, sensor networks are commonly used to monitor key environmental and soil conditions. These systems often operate without constant supervision and play a critical role in supporting decisions related to crop management, irrigation, and resource optimization. However, their exposure and reliance on wireless communication make them vulnerable to a range of potential threats.

Various types of cyber-physical threats may impact the accuracy and availability of these variables. For instance, data manipulation attacks may introduce falsified readings that simulate abnormally dry or overly optimal soil conditions. This can lead to misguided irrigation actions—either triggering unnecessary water usage or delaying interventions during actual stress periods. Likewise, incorrect values for exterior

humidity can influence how soil moisture dynamics are interpreted, especially in relation to evaporation and transpiration rates.

In addition to direct manipulation, spoofing attacks further complicate monitoring reliability. By impersonating legitimate sensor nodes and injecting synthetic yet statistically plausible data, attackers can mask anomalies within normal-looking trends. This is particularly challenging when the forged values fall within expected seasonal or diurnal patterns, making traditional rule-based detection mechanisms insufficient. However, anomaly detection models such as AIMDE, which leverage the natural correlations among these variables, are better suited to identifying such inconsistencies.

Another significant risk arises from replay attacks, which involve injecting previously recorded, contextually outdated sensor values into the system. These attacks can delay detection of actual environmental changes, skew model predictions, and potentially conceal additional malicious activities. The ability to identify temporal misalignments in correlated data streams is therefore crucial to maintaining system integrity.

Beyond immediate operational impact, these threats can have longer-term consequences. Manipulated or spoofed sensor data may influence historical datasets used for planning and modeling, distorting the perceived trends in environmental behaviour. Replay attacks can also act as a smokescreen for physical intrusions or other coordinated exploits, further eroding trust in the system's data. In resource-constrained environments—such as remote or solar-powered sensor deployments—even seemingly minor data manipulations can go undetected for extended periods, especially if the attack mimics plausible environmental variation.

Additionally, sustained or repeated attacks targeting specific variables like soil moisture or conductivity may introduce gradual shifts in automated decision-making processes. For example, consistently forged dryness values could cause overwatering that depletes aquifers or alters soil structure over time. These gradual effects make detection even more challenging, as they may initially be attributed to environmental anomalies rather than system compromise. Identifying such patterns requires monitoring not only the values themselves but also their contextual coherence over time.

8.3 Identification of Correlations in Smart Cities Dataset

For the Smart Cities UC, agroclimatic data, such as soil conditions (including humidity, radiation, electric conductivity, etc), indoor/outdoor air temperature, wind speed and direction, solar panel battery levels, panels inclination, are available from IoT sensors of an agricultural research centre in Murcia, Spain. These diverse datasets can serve as the basis for efficient monitoring and managing of urban agricultural and environmental systems in smart city environments.

Under this context, the AIMDE is leveraged to analyse patterns in this dataset and identify potential anomalies that could signify the system's (or sensors') misbehaviour. These misbehaviours focus on detecting and flagging abnormal changes in soil parameters, or unexpected shifts in indoor climate patterns. By detecting these anomalies, the AIMDE can promptly alert the operators to intervene, thus ensuring the continued integrity and efficient operation of the agricultural infrastructure. Misbehaviour in this context refers to significant deviations from standard agroclimatic behaviour, operational failures in the monitoring equipment (sensors), which could indicate malfunctions or environmental threats, or even cybersecurity attacks. To verify these potential misbehaviours, the AIMDE utilises the available dataset for anomaly detection, allowing for the precise identification of irregularities, thus ensuring the efficient operation of the agricultural infrastructure.

It shall be noted that at the time of writing this deliverable, hourly data covering a period of 8 months was available (originating from sensors measurements). However, it originally lacked the contextual information needed to label or clearly identify anomalous behaviour. To overcome this, the analysis first focused on identifying potential correlations among the provided data attributes; an aspect not evident

from the data itself. Upon their analysis, the data variables identified to have a weak or inconsistent degree of correlation include **Soil Electric Conductivity, Soil Humidity, and Exterior Air Humidity**. These findings highlighted the need for expert knowledge and further analysis to accurately interpret potential anomalies based on these less strongly correlated data points.

The motivation behind this correlation analysis is two-fold: first, to establish a baseline understanding of the expected interrelationships between the different agroclimatic variables, thus defining ”normal” system behaviour, (i.e., identify genuine anomalies that can serve as ground truth for training a supervised machine learning model); and second, to subsequently detect deviations from these established correlations, which can serve as strong indicators of potential anomalies such as sensor malfunctions or environmental irregularities. Experts' knowledge was then sought to validate the findings, who confirmed that the strongest correlation existed between Soil Electric Conductivity, Soil Humidity, and Exterior Humidity, reinforcing the validity of this initial analytical step in the absence of labelled anomaly data.

Following this, a preliminary round of point anomaly detections was undertaken, using the Isolation Forest algorithm capable of identifying anomalies, based on how many splits are needed to isolate a data point (within the overall timeseries provided), with fewer splits indicating a higher likelihood of anomaly. As the behavior of the Isolation Forest is influenced by the contamination parameter (which defines the expected proportion of anomalies) several tests were run with varying contamination values to align the results as closely as possible with experts-reviewed anomalies (from prior discussions). Despite this, the model still may flag some normal values as anomalies and in order to refine the results additional Experts input was sought. To support this process an updated dataset has been provided back to the Experts, including an “Anomaly score” column (with negative values indicating potential anomalies and lower scores representing a higher model prediction confidence) for their review.

Plausibility Check	Description	Indicators
Soil Humidity Fluctuations	Abnormal soil humidity readings indicating potential hardware malfunction or environmental influences. Such uncontrolled or illegitimate soil humidity fluctuations may affect the quality of agricultural production.	Sudden spike or decline in soil humidity readings, or stationary measurements unchanged for prolonged periods.

Table 8.1: Plausibility check for REWIRE AI-based Misbehaviour Detection Engine (AIMDE) in Smart Cities Use Case

Note that, due to the absence of labelled data, it was not yet possible to fully validate the model's results at this point, and the interpretation of the model's prediction and the refineent of the detection accuracy is ongoing. However, the data provided by the municipality served towards providing the **ground truth**, i.e., the expected behaviours against which the AIMDE will evaluate the correctness of the measurement data. Considering this, in Table 8.1, we provide the **plausibility check** defined for this use case, pertaining to the **fluctuations in soil humidity**. Based on the aforementioned findings and the available data, and considering aforementioned the plausibility check, the anomaly detection ”strategy” moving forward will include the following approaches:

- **Unsupervised Anomaly Detection: Detecting Point Anomalies**

This refers to the case where a **ground truth is not available**, i.e., we need to rely on correlations between the measured data. Thus, building on the identified correlation, the analysis will continue focusing on point anomaly detection on Soil Humidity, utilising features such as the Exterior Air Temperature and Soil Humidity sensors' readings. Methods such as Elliptic Envelope or Local Outlier Factor should be suitable for identifying any anomalies.

- **Supervised Anomaly Detection: Detecting Point Anomalies**

This refers to the case where a ground truth is available, thus enabling the possibility for **direct identification** of anomalies (i.e., deviations from the ground truth). Specifically, based on the experts' feedback from the validation of the preliminary point anomaly results, we can use these outcomes as ground truth to train supervised models for anomaly detection on Soil Humidity and Exterior Temperature. Depending on the dataset size, models such as XGBoost classifiers or neural networks can be employed.

- **Supervised/Unsupervised Anomaly Detection: Detecting Contextual Anomalies**

This refers to the case where a ground truth may or may not be available, thus necessitating a **hybrid approach including both direct identification, and identification based on data correlations**. In this regard, we can also explore the possibility of identifying anomalous events by detecting patterns in the available agricultural data within specific contexts, such as time of day or season. To achieve this, calendar-based features combined with soil humidity and/or temperature can be used to train models. Depending on the availability of labeled anomalies, either unsupervised approaches like Isolation Forest or supervised models such as XGBoost classifiers can be applied. Neural networks may also be considered, provided the dataset size is sufficient to support their use.

Chapter 9

Conclusions

Throughout this Deliverable, we have expanded upon the core target of REWIRE of providing the necessary security and truste enablers in order to enhance the secure lifecycle management of embedded systems featuring RISC-V architectures. In this regard, we provided a detailed description of the final reference architecture of REWIRE, featuring the distinct categorization of actions into the **Design-time** and **Runtime Phases** of the action workflow. The REWIRE approach is centred around the **Compositional Verification and Validation Component**, whose purpose is the Formal Verification of the security processes and enablers of REWIRE. The core tenet of this approach is the definition of the **trust boundary** of a device, which entails the identification of the properties that can be verified during design-time, and those that need to be attested dynamically during runtime.

First, we provided a detailed definition of the REWIRE reference architecture and action workflow, focusing on the updates compared to those documented in D2.1. Specifically, one such update pertains to the definition of the **Policy Orchestrator**, which is responsible for the distribution and enforcement of the security and operational policies provided as output of the Design-time phase, in order to ensure the execution of the security enablers specified therein. In addition, we expanded upon the **Formal Verification** capabilities of REWIRE by providing enhanced capabilities for the verification of security properties of REWIRE, especially the **Configuration Integrity Verification (CIV)** scheme. In this regard, REWIRE provides one of the first attempts in the literature for the formal verification of the local attestation concept with key restriction usage policies. We also expanded upon the **Live Migration** feature, which entails the transition of an enclavized application in a device that may be deemed compromised to a different one, which exhibits the required level of trust. The **Key Management Layer** is also documented, for allowing the enclaves to manage their respective cryptographic keys, instantiated in the **Security Monitor (SM)** in Keystone architectures. Finally, we provided further details on the **Policy-compliant Blockchain Infrastructure**, and specifically the use of **Town Crier (TC)** as a Secure Oracle for supporting the data veracity of application-related data, as well as the use of **Fabric Private Chaincode (FPC)** for capturing the trustworthiness of evidence.

Next, we provided an updated and comprehensive list of **Functional Specifications and Requirements** of the REWIRE framework, building upon those introduced in D2.1, considering also the aforementioned framework and component updates, as well as their functionality in the context of the types of systems and environments targeted by REWIRE. These were assicuated with the corresponding list of **Key Performance Indicators (KPIs)** for evaluating their fulfilment, which will guide the evaluation activities to be performed and documented in the context of D6.2. The activities carried out in this deliverable also included the comprehensive definition and documentation of the **Threat Landscape** of REWIRE, considering the types of threats affecting each component, and the need to dynamically adapt to the continuously evolving types of threats affecting IoT devices. This exercise culminated in the correspondence between threats and the respective **security enablers** offered by REWIRE which can be used for their mitigation, the **properties** associated with the threats, as well as the types of **trustworthiness evidence** that need to be collected for evaluating their correctness.

Particular focus was placed on the **REWIRE Blockchain Infrastructure** and the **Secure Oracle Layer**. Specifically, REWIRE instantiates the Blockchain Infrastructure over the **HyperLedger Besu** technology, paired with the SGX-based **Town Crier (TC)** acting as a secure oracle and serving as a credible and authenticated bridge for the execution of smart contracts. **Fabric Private Chaincode (FPC)** is also employed for ensuring veracity of data recorded to the Blockchain. Next, a detailed Blockchain-related threat analysis was performed, containing detailed descriptions of attacks such as **(Distributed) Denial of Service (DoS and DDoS)**, **Consensus Manipulation**, and attacks related to the **compromise of Verifiable Credential management mechanisms.** We also provided a threat analysis of Secure Oracle approaches, as well as associated mitigation measures and potential future research directions.

Another core focus of this Deliverable was the definition of the **Policy Expression Language** of REWIRE, focused on capturing the required **expressiveness level** for depicting the security and operational policies provided as output of the Design-time phase. To this end, we described the motivation behind the selection of the **Medium Security Policy Language (MSPL)**, compared to other available languages in the literature, and we provided details on the structure and fields included in a REWIRE policy, the positioning of policies in the REWIRE framework (demonstrated through the example of secure software updates), and examples of MSPL policy constructions for various REWIRE security enablers.

Finally, we documented information on the instantiation of the **AI-based Misbehaviour Detection Engine (AIMDE)** of REWIRE in the context of the **Smart Cities** use case. This analysis was focused on the specific needs of this application domain, namely the diverse challenges and complexities of urban environments with focus on agricultural research infrastructures, in order to strengthen their operational integrity. Thus, a threat analysis for this domain was performed, as well as a round of anomaly detections for identifying anomalies most prominent in such environments.

Bibliography

[1] Anne Anderson, Anthony Nadalin, B Parducci, D Engovatov, H Lockhart, M Kudo, P Humenn, S Godik, S Anderson, S Crocker, et al. eXtensible Access Control Markup Language (XCAML) Version 1.0. *OASIS*, 2003.

[2] Jorge Bernal Bernabé, Juan M Marín Pérez, Jose M Alcaraz Calero, Jesús D Jiménez Re, Félix JG Clemente, Gregorio Martínez Pérez, and Antonio FG Skarmeta. Security Policy Specification. In *Network and Traffic Engineering in Emerging Distributed Computing Applications*, pages 66–93. IGI Global, 2013.

[3] Sotirios Brotsis, Nicholas Kolokotronis, Konstantinos Limniotis, Gueltoum Bendiab, and Stavros Shiaeles. On the security and privacy of hyperledger fabric: Challenges and open issues. In *2020 IEEE World Congress on Services (SERVICES)*, pages 197–204, 2020.

[4] Ahaan Dabholkar and Vishal Saraswat. Ripping the fabric: Attacks and mitigations on hyperledger fabric. In V. S. Shankar Sriram, V. Subramaniyaswamy, N. Sasikaladevi, Leo Zhang, Lynn Batten, and Gang Li, editors, *Applications and Techniques in Information Security*, pages 300–311, Singapore, 2019. Springer Singapore.

[5] DMTF Standard. Common Information Model. https://www.dmtf.org/standards/cim [Online; accessed November-2021].

[6] EUR-Lex. Regulation (eu) no 910/2014 of the european parliament and of the council of 23 july 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing directive 1999/93/ec. Official Journal of the European Union, 2014.

[7] Athanasios Giannetsos, Dimitris Papamartzivanos, Sofia Anna Menesidou, and Sophia Karagiorgou. Towards 5g embedded trust: Integrating attestation extensions in vertical industries. In *2021 European Conference on Networks and Communications*, 2022.

[8] GlobalPlatform, Inc. Globalplatform technology tee internal core api specification version 1.3.1. https://globalplatform.org/wp-content/uploads/2021/03/GPD_TEE_Internal_Core_API_Specification_v1.3.1_PublicRelease_CC.pdf, 2021.

[9] Daan Gordijn, Roland Kromes, Thanassis Giannetsos, and Kaitai Liang. Combining id's, attributes, and policies in hyperledger fabric. In Weizhi Meng and Wenjuan Li, editors, *Blockchain Technology and Emerging Technologies*, pages 32–48, Cham, 2023. Springer Nature Switzerland.

[10] Ilya Grishkov, Roland Kromes, Thanassis Giannetsos, and Kaitai Liang. Id-based self-encryption via hyperledger fabric based smart contract. In Weizhi Meng and Wenjuan Li, editors, *Blockchain Technology and Emerging Technologies*, pages 3–18, Cham, 2023. Springer Nature Switzerland.

[11] Ana Hermosilla, Alejandro Molina Zarca, Jorge Bernal Bernabé, Jordi Ortiz, and Antonio F. Skarmeta. Security orchestration and enforcement in nfv/sdn-aware uav deployments. *IEEE Access*, 8:131779–131795, 2020.

[12] Nikolay Ivanov, Chenning Li, Qiben Yan, Zhiyuan Sun, Zhichao Cao, and Xiapu Luo. Security threat mitigation for smart contracts: A comprehensive survey. *ACM Comput. Surv.*, 55(14s), jul 2023.

[13] Keystone. Keystone security monitor. https://docs.keystone-enclave.org/en/latest/Security-Monitor/index.html [Online; accessed May-2025].

[14] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. Keystone: An open framework for architecting trusted execution environments. In *Proceedings of the Fifteenth European Conference on Computer Systems*, pages 1–16, 2020.

[15] Linaro and the TrustedFirmware.org project. Op-tee. https://www.trustedfirmware.org/projects/op-tee/, 2019.

[16] Dimosthenis Masouros, Dimitrios Soudris, Georgios Gardikis, Victoria Katsarou, Maria Christopoulou, George Xilouris, Hugo Ramón, Antonio Pastor, Fabrizio Scaglione, Cristian Petrollini, António Pinto, João P. Vilela, Antonia Karamatskou, Nikolaos Papadakis, Anna Angelogianni, Thanassis Giannetsos, Luis Javier García Villalba, Jesús A. Alonso-López, Martin Strand, Gudmund Grov, Anastasios N. Bikos, Kostas Ramantas, Ricardo Santos, Fábio Silva, and Nikolaos Tsampieris. Towards privacy-first security enablers for 6g networks: The privateer approach. In Cristina Silvano, Christian Pilato, and Marc Reichenbach, editors, *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 379–391, Cham, 2023. Springer Nature Switzerland.

[17] Sara N. Matheu, Alberto Robles Enciso, Alejandro Molina Zarca, Dan Garcia-Carrillo, José Luis Hernández-Ramos, Jorge Bernal Bernabe, and Antonio F. Skarmeta. Security architecture for defining and enforcing security profiles in dlt/sdn-based iot systems. *Sensors*, 20(7), 2020.

[18] OWASP. Owasp mobile top 10. https://owasp.org/www-project-mobile-top-10/ [Online; accessed May-2025].

[19] David Brossard Pablo Giambiagi. Alfa language basics. https://alfa.guide/alfa-authorization-language/ [Online; accessed May-2025].

[20] REWIRE. Rewire integrated framework (1st release) and use case analysis. Deliverable D6.1, The REWIRE Consortium, 21 2023.

[21] REWIRE. Rewire operational landscape, requirements, and reference architecture - initial version. Deliverable D2.1, The REWIRE Consortium, 12 2023.

[22] REWIRE. Rewire design time secure operational framework - final version. Deliverable D3.3, The REWIRE Consortium, 30 2024.

[23] REWIRE. Rewire runtime assurance framework - initial version. Deliverable D4.2, The REWIRE Consortium, 18 2024.

[24] REWIRE. Rewire integrated framework (final release), use case evaluation and project impact assessment. Deliverable D6.2, The REWIRE Consortium, 36 2025.

[25] REWIRE. Rewire runtime assurance framework - final version. Deliverable D4.3, The REWIRE Consortium, 30 2025.

[26] RISC-V Platform Specification Task Group. Risc-v supervisor binary interface specification. https://www.scs.stanford.edu/~zyedidia/docs/riscv/riscv-sbi.pdf, March 2022.

[27] Ahmad Sabouri, Ioannis Krontiris, and Kai Rannenberg. Attribute-based credentials for trust (abc4trust). In Simone Fischer-Hübner, Sokratis Katsikas, and Gerald Quirchmayr, editors, *Trust, Privacy and Security in Digital Business*, pages 218–219, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[28] Sarwar Sayeed, Hector Marco-Gisbert, and Tom Caira. Smart contract: Attacks and protections. *IEEE Access*, 8:24416–24427, 2020.

[29] SECURED EU FP7 project. Deliverable D4.1: Policy specification. https://www.secured-fp7.eu/files/secured_d41_policy_spec_v0100.pdf [Online; accessed November-2021].

[30] SECURED EU FP7 project. Deliverable D4.2: Policy transformation and optimization techniques. https://www.secured-fp7.eu/files/secured_d42_policy_refinement_v0103.pdf [Online; accessed November-2021].

[31] suryakantamangaraj. Awesome risc-v resources. https://github.com/suryakantamangaraj/AwesomeRISC-VResources?tab=readme-ov-file, 2022.

[32] Yining Tang, Qihang Luo, Runchao Han, Jianyu Niu, Chen Feng, and Yinqian Zhang. Unraveling responsiveness of chained bft consensus with network delay. 01 2025.

[33] Fulvio Valenza and Antonio Lioy. User-oriented network security policy specification. *J. Internet Serv. Inf. Secur.*, 8:33–47, 2018.

[34] Gang Wang. SoK: Understanding BFT consensus in the age of blockchains. Cryptology ePrint Archive, Paper 2021/911, 2021.

[35] Waterman, Andrew and Asanovic, Krste and Hauser, John and SiFive Inc. The risc-v instruction set manual. volume ii: Privileged architecture. https://github.com/riscv/riscv-isa-manual/releases/download/Priv-v1.12/riscv-privileged-20211203.pdf, December 2021.

[36] Waterman, Andrew and Asanovic, Krste and SiFive Inc. The risc-v instruction set manual. volume i: Unprivileged isa. https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf, December 2019.

[37] Gengrui Zhang, Fei Pan, Yunhao Mao, Sofia Tijanic, Michael Dang'ana, Shashank Motepalli, Shiquan Zhang, and Hans-Arno Jacobsen. Reaching consensus in the byzantine empire: A comprehensive review of bft consensus algorithms. *ACM Comput. Surv.*, 56(5), January 2024.

[38] Weiyu Zhong, Ce Yang, Wei Liang, Jiahong Cai, Lin Chen, Jing Liao, and Naixue Xiong. Byzantine fault-tolerant consensus algorithms: A survey. *Electronics*, 12(18), 2023.

ANNEX

Trust Management Terms and Definitions

This chapter summarizes terms and definitions of those terms related to trust management, evidence-based trust assessment mechanisms and its associated functionalities along with a selected set of its characteristics capturing those important specifications that enable the **characterization and quantification of trust in complex environments and multi-agent systems with potentially high level of uncertainty in the trustworthiness of exchanged data**. This vocabulary acts as a complementary source to the attestation and other trust extensions been defined in the context of REWIRE. *This vocabulary it is intended for use horizontally in the information technology domain and all domains where Subjective Logic is used as the foundation of trust reasoning.*

ATL (Actual Trustworthiness Level) The ATL reflects the result of an evaluation of a specific (atomic or complex) proposition for a specific scope provided by the TLEE. It quantifies the extent to which a certain node or data can be considered trustworthy based on the available evidence.

ATO (Atomic Trust Opinion) An ATO is a subjective logic opinion created by the TSM when quantifying one specific type of a Trust Source based on trustworthiness evidence. An ATO is formed by a Trust Source in the context of a single trust relationship.

Component Diagram A component diagram is a graph whose vertices represent components necessary to realize a concrete system function, and whose edges represent communication links between these components. A component diagram is used to design function-specific trust models.

Data-centric Trust Data-centric trust is evaluated in the context of a data-centric trust relationship. In this trust relationship, trust is evaluated from one node to data, where the trustor (the one who trusts) is a node and the trustee (the one who is trusted) is data.

Functional (direct) Trust Functional (also called direct) trust is evaluated in the context of a functional trust relationship. In this trust relationship, trust is evaluated between two trust objects have a direct relation, i. e., the trustor has a direct observation of the trustee and where the trustor engages in a functional relationship with the trustee within the system model like receiving a data item from the trustee.

Node-centric Trust Node-centric trust is evaluated in the context of a node-centric trust relationship. In this trust relationship, trust is evaluated from one node to another node so that the trustor (the one who trusts) and the trustee (the one who is trusted) are both nodes.

PROP (Proposition) A *proposition* is a logic statement about some phenomenon of interest whose level of trustworthiness we are interested in assessing. A proposition could be 1) atomic—a proposition whose truth or trustworthiness can be directly assessed, or 2) composite, comprising of multiple atomic propositions. The proposition describes the fulfillment of the properties in relation to *data* or *nodes*.

Referral Trust Referral trust is evaluated in the context of a referral trust relationship. In this trust relationship, trust is evaluated between two trust objects that do not have a direct relation but the trustor has a direct relationship with another intermediate node(s) that have a direct observation of the trustee.

RTL (Required Trustworthiness Level) The RTL reflects the amount of trustworthiness of a node or data that an application considers required in order to characterize this object as trusted and rely on its output during its execution.

TA (Trust Assessment) The TA is a component inside the TAF which orchestrates the overall process of trustworthiness level evaluation and trust decision taking.

TAF (Trust Assessment Framework) A software framework which, given a trust model for a specific function running inside a CCAM system, is able to evaluate trust sources for trustworthiness evidence and evaluate propositions within the trust model to obtain their ATLs. Optionally, also an RTL can be evaluated and trust decisions can be taken and communicated to the application.

TDE (Trust Decision Engine) The TDE is a component inside the TAF which performs the last step before an output is provided to the application that requested trustworthiness assessment. The TDE either forwards the Actual Trustworthiness Level (ATL) calculated by the TLEE along to the application or outputs a Trust Decision (TD). A TD is created after comparing the ATL to the Required Trust Level (RTL) in a predetermined manner. Whether the output of the TAF is an ATL or a TD depends on the needs of the application requesting trustworthiness assessment.

TLEE (Trustworthiness Level Expression Engine) The TLEE is a component inside the TAF that calculates the level of trustworthiness for a concrete trust model and the proposition that needs to be evaluated. The TLEE uses the numerical values of the Atomic Trust Opinions computed based on the trust sources collected in the TSM. Based on these inputs, the TLEE calculates an ATL and provides it to the TA. The TLEE encapsulates most of the Subjective Logic formalism.

TM (Trust Model) Trust model is a graph-based model which is built on top of a system model which represents all components and data needed to perform a certain function. Components represented either create, transmit, process, relay, and receive the data used as input to a function. The vertices in a trust model correspond to an abstraction called trust objects, and the edges in a trust model correspond to trust relationships between a pair of trust objects. The trust model also encompasses a list of trust sources used to build up / quantify trust relationships by providing atomic trust opinions. The trust model is a main input to the TMM and the TLEE. Since trust is a directional relationship between two trust objects and it is always in relation to a concrete property or scope, then as part of the trust model, there can be multiple trust relationships between the same two trust objects, depending on different properties of the trust relationship, or the scope of the trust relationship.

TMM (Trust Model Manager) The TMM is a component inside the TAF responsible for storing trust models and making them accessible for TLEE and other purposes. In particular, it is able to provide TMs for specific functions running in a multi-agent system, also considering different scopes that TMs may cover.

Trust Objects Trust objects are core building blocks of a trust model. They represent entities that assess trust or for which trust is assessed. The trust objects are identified 1) based on the *components* from the component diagram and 2) the *atomic propositions* (i.e., the properties about data or nodes for which trust assessment is conducted).

Trust Relationships A trust relationship is a directional relationship between two trust objects that are called trustor (i. e., the “thinking entity”, the assessor) and a trustee (one who is trusted). The trust relationship is always in relation to a concrete property and a certain scope.

Trustworthiness Tier Trustworthiness Tier is a categorization of the levels of trustworthiness which may be assigned by the TAF (or another Verifier that appraises the attestation results of a TC and communicates them to the TAF) to a specific Trustworthiness Claim.

Trustworthiness Claims A Trustworthiness Claim (TC) is a form of node-centric ATO provided by a TS and contains a specific data quote used for conveying the information needed by the TAF to make a decision on the trust level of an object. The TC is usually produced (by the Attester) so as to provide trustworthiness evidence (cf. "Trust Source") that can be used for appraising the trustworthiness level of the Attester in a *measurable* and *verifiable* manner. Measurable reflects the ability of the TAF to assess an attribute of the Attester against a pre-defined metric while verifiability highlights the need for all claims to have integrity, freshness and to be provably & non-reputably bound to the identity of the original Attester. Examples sets of TCs might include (among other attributes) evidence on system properties including: (i) **integrity** in the context that all transited devices (e.g., ECUs) have booted with known hardware and firmware; (ii) **safety** meaning that all transited devices are from a set of vendors and are running certified software applications containing the latest patches and (iii) **communication integrity**. For a more detailed list of possible system (behavioural) evidence that can be appraised as part of TCs, please refer to Section 4.4.

TS (Trust Source) A TS manages one or multiple trustworthiness evidence inside the TAF. On request of the TA, it quantifies the trustworthiness of a trustee based on a specific type of evidence in form of an atomic trust opinion.

TSM (Trust Sources Manager) The TSM is a component inside the TAF responsible for handling all available TS inside a TAF and to establish and integrate new TSs dynamically through a plugin interface.