
Co-funded by the Horizon 2020
Framework Programme of the European Union

# Practical Autonomous Cyberhealth for resilient SMEs & Microenterprises

Grant Agreement No. 883335
Innovation Action (IA)

# D4.2 Trust Attestation and Verification Framework – First Release

| **Document Identification** | | | |
|---|---|---|---|
| **Status** | Final | **Due Date** | 31/12/2021 |
| **Version** | 1.0 | **Submission Date** | 24/01/2022 |

| | | | |
|---|---|---|---|
| **Related WP** | WP4 | **Document Reference** | 1.0 |
| **Related Deliverable(s)** | D2.1 | **Dissemination Level (*)** | PU |
| **Lead Participant** | HPELB | **Lead Author** | Supreshna Gurung and Ludovic Jacquin (HPELB) |
| **Contributors** | HPELB<br>POLITO<br>SFERA | **Reviewers** | Davide Sanvito (NEC)<br>Diego R. Lopez (TID) |

| **Keywords:** |
|---|
| Trust, Attestation, Recovery, Fault management, Breach management, TPM, Finite State Machine, Deterministic Finite Automata. |

# Document Information

| List of Contributors | |
|---|---|
| Name | Partner |
| Supreshna Gurung, Ludovic Jacquin | HPELB |
| Silvia Sisinni, Ignazio Pedone and Antonio Lioy | POLITO |
| Izidor Mlakar, Valentino Šafran, Renato Pulko and Primož Jeran | SFERA |

| Document History | | | |
|---|---|---|---|
| Version | Date | Change editors | Changes |
| 0.6 | 15/12/2021 | HPELB, POLITO, SFERA | Final draft for internal review. |
| 0.11 | 05/01/2022 | HPELB, SFERA | Revised draft for second internal review. |
| 1.0 | 21/02/2022 | HPELB | Final version for submission |

| Quality Control | | |
|---|---|---|
| Role | Who (Partner short name) | Approval Date |
| Deliverable leader | HPELB | 21/01/2022 |
| Quality manager | INFILI | 24/01/2022 |
| Project Coordinator | DBC | 24/01/2022 |

# Executive Summary

Following the work presented in Deliverable 2.1, where the requirements of the PALANTIR solution are elicited and the high-level design and architecture of the platform is presented, a detailed study of the different components has been done in order to obtain the low-level architecture and design (up to subcomponent granularity), the specifications (by means of the transformation of the user requirements into technical requirements/specifications) and the implementation guide (describing the technologies to use) for each component of the PALANTIR solution. This work has been divided into the three technical development work packages of PALANTIR, namely Work Packages (WP) 3, 4 and 5. This deliverable covers the Trust, Attestation and Recovery component developed in WP4. The target audience for this document is the technical communities that are interested in understanding, developing or deploying the Trust, Attestation and Recovery component of the project PALANTIR's solution.

# Table of Contents

# List of Tables

# List of Figures

# List of Acronyms

| Abbreviation / acronym | Description |
|---|---|
| AE | Attestation Engine |
| AMD SEV | AMD Secure Encrypted Virtualization |
| DFA | Deterministic Finite Automata |
| DIME | Distributed Intrusion Monitoring Engine |
| Dx.y | Deliverable number y, belonging to WP number x |
| ETSI | European Telecommunications Standards Institute |
| FBM | Fault & Breach Management |
| FSM | Finite State Machine |
| GDPR | General Data Protection Regulation |
| IAK | Initial Attestation Key |
| IDevID | Initial Device IDentity |
| IMA | Integrity Measurement Architecture |
| Intel SGX | Intel Software Guard Extensions |
| IR | Incident Response |
| JSON | JavaScript Object Notation |
| MSPL | Medium-level Security Policy Language |
| NFV | Network functions virtualization |
| OCI | Open Container Initiative |
| PCR | Platform Configuration Register |
| RoT | Root of Trust |
| RR | Remediation and Recommendation |
| RS | Recovery Service |
| SC | Security Capability |
| SCHI | Security Capabilities Hosting Infrastructure |
| SCO | Security Capability Orchestrator |
| TAR | Trust, Attestation and Recovery |
| TEE | Trusted Execution Environment |
| TI | Threat Intelligence |
| TPM | Trusted Platform Module |
| UEFI | Unified Extensible Firmware Interface |
| UML | Unified Modeling Language |
| URL | Uniform Resource Locator |
| VE | Virtual Environment |
| VM | Virtual Machine |
| WP | Work Package |
| XML | Extensible Markup Language |

# 1. Introduction

This "D4.2 Trust Attestation and Verification Framework –First Release" deliverable presents the architecture, specifications and implementation of the different entities that compose the Trust, Attestation and Recovery (TAR) component of the PALANTIR architecture presented in D2.1; it relates to tasks T4.2 and T4.4. This document provides the design and implementation guidelines for the work in WP4 and in related aspects in other WPs, such as WP3 and WP5.
This deliverable is the first iteration of the Trust, Attestation and Recovery component specifications. The next version of this deliverable is due at month 31 of the PALANTIR project, where the final design and implementations is provided, together with other technical details on implementation and integration with other components of the PALANTIR platform.
This deliverable starts with the design and architecture of each sub-component of the TAR, followed by their technical specification and the implementation guidelines.

# 2. Design and Architecture

Figure 1 represents the high-level architecture introduced in Deliverable 2.1 (D2.1) of the project. This section details the internal architecture of the TAR component, specifically the Attestation Engine (AE), the Fault and Breach Management (FBM) and the Recovery Service (RS).

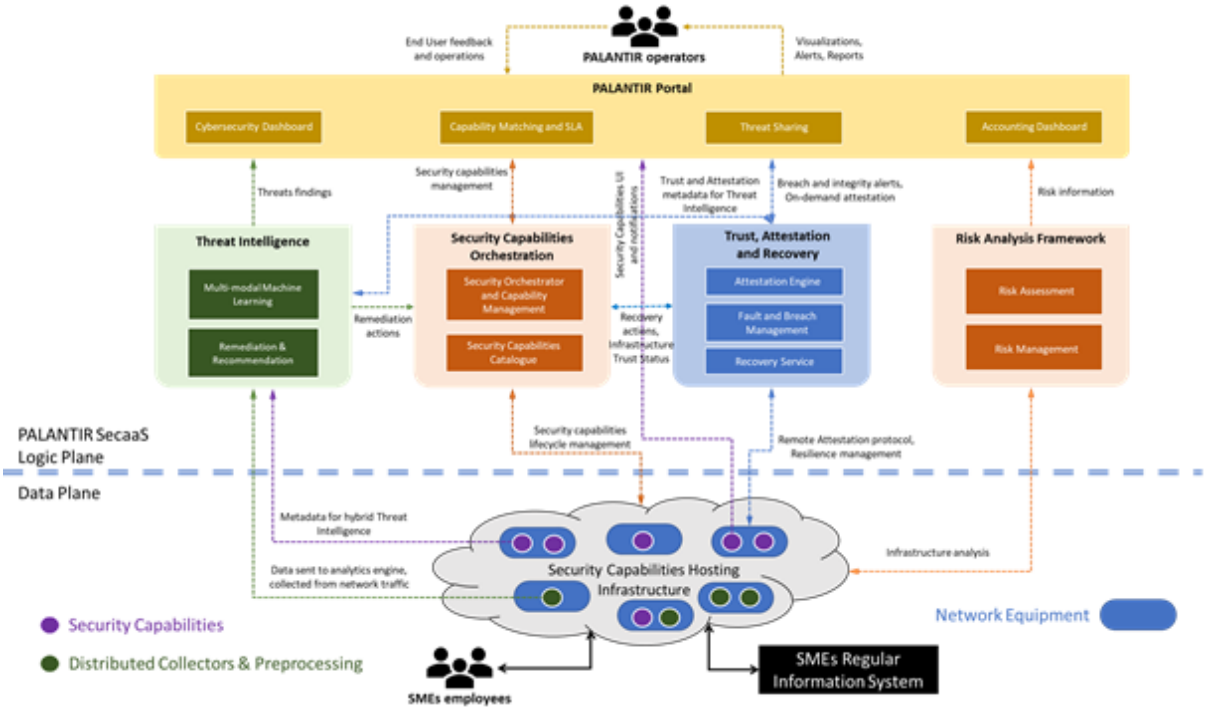


Figure 1: High level PALANTIR architecture

## 2.1. Attestation Engine

The Attestation Engine component inside the Trust, Attestation and Recovery framework implements the remote attestation solution of Project PALANTIR. The Attestation Engine reaches this goal by remotely establishing trustworthiness of a platform or service by exploiting Trusted Computing technologies such as Trusted Platform Module (TPM) [1] chips as the Root of Trust (RoT) and Measured Boot feature of Unified Extensible Firmware Interface (UEFI) along with leveraging memory inspection capability to perform runtime [2] verification of the platform.

The Attestation Engine integrates with the Security Capabilities Hosting Infrastructure (SCHI) to perform remote attestation procedure and forwards attestation results to the Recovery Service, another component of TAR that assists with resilience management and incident response of the Security Capabilities (SC). The attestation results are also forwarded to Threat Intelligence (TI) for threat classification using technologies like machine learning and deep learning as well as the PALANTIR Portal for visualisation.

### 2.1.1. Subcomponents

The Attestation Engine is a distributed system that has subcomponents distributed throughout the PALANTIR infrastructure in order to maintain trustworthiness of the system.

- **Attestation Agent:** Each monitored node of SCHI hosts an Attestation Agent that is responsible for forwarding attestation information and alerts used by the AE. For example, it extracts the measurement stored in the TPM, signed with a TPM specific key, and sends them to the AE.

- **Reference Measurement plugin:** The AE needs a baseline set of measurement, known to be correct, to compare the SCHI measurement against. For SC, those baseline sets are generated by a Reference Measurement plugin that generates the known-good measurement for the new SC images onboarded in PALANTIR; this plugin is deployed in the Security Capability Orchestrator (SCO).

### 2.1.2. General Workflow

The Attestation Engine is responsible for establishing trustworthiness of the infrastructure and security services. This is obtained by applying trusted computing attestation to the platforms and Security Capabilities starting at the initial deployment through operation. The following figures describe the workflow of each phase of the attestation procedures performed by the Attestation Engine.

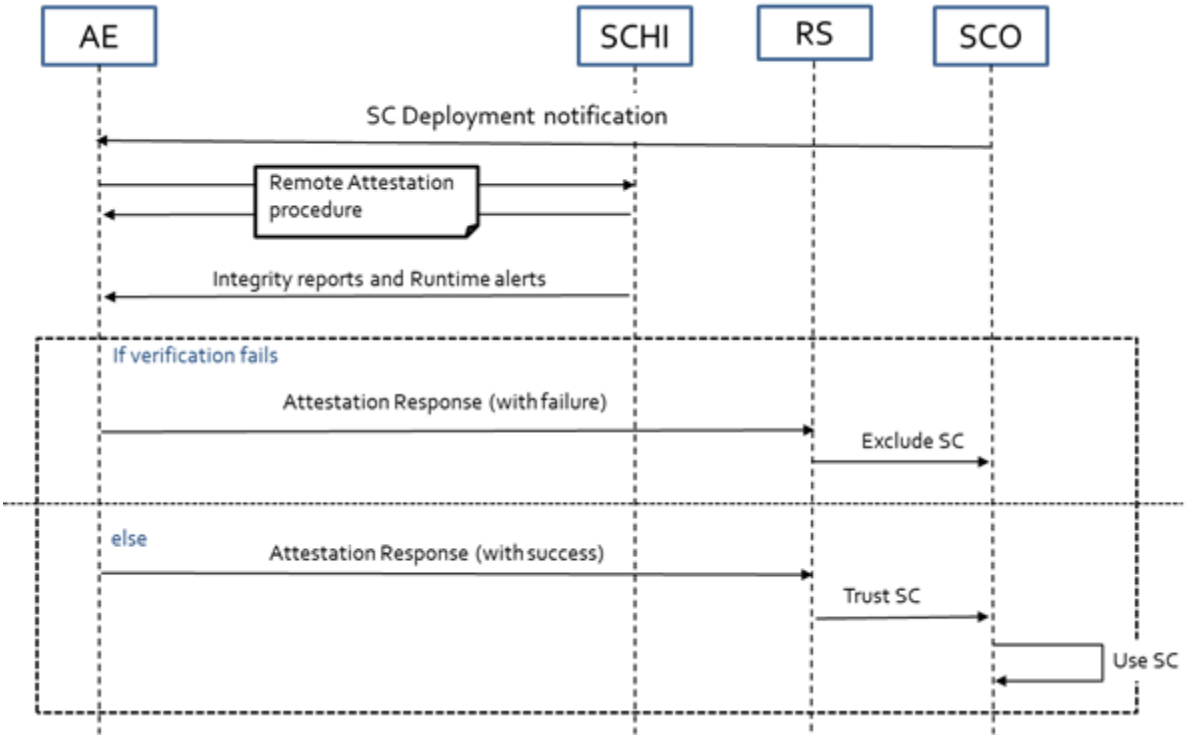


Figure 2: Initial Attestation Workflow

The initial attestation occurs each time a new instance of a given Security Capability is deployed. Figure 2 depicts the workflow for initial attestation of every new instance. The Security Capabilities Orchestration, which is in charge of the overall management of the security capabilities, notifies the Attestation Engine when a new instance of the Security Capability is deployed or when a new platform is introduced in the SCHI. The Attestation Engine performs remote attestation procedures on the SC to validate its state. The SCHI also assists with the integrity reports from the AE agent hosted in SCHI for verification. If the verification fails, the AE sends a failure response to the Recovery Service (RS) suggesting the SCO not to use that SC and to exclude it from the infrastructure.

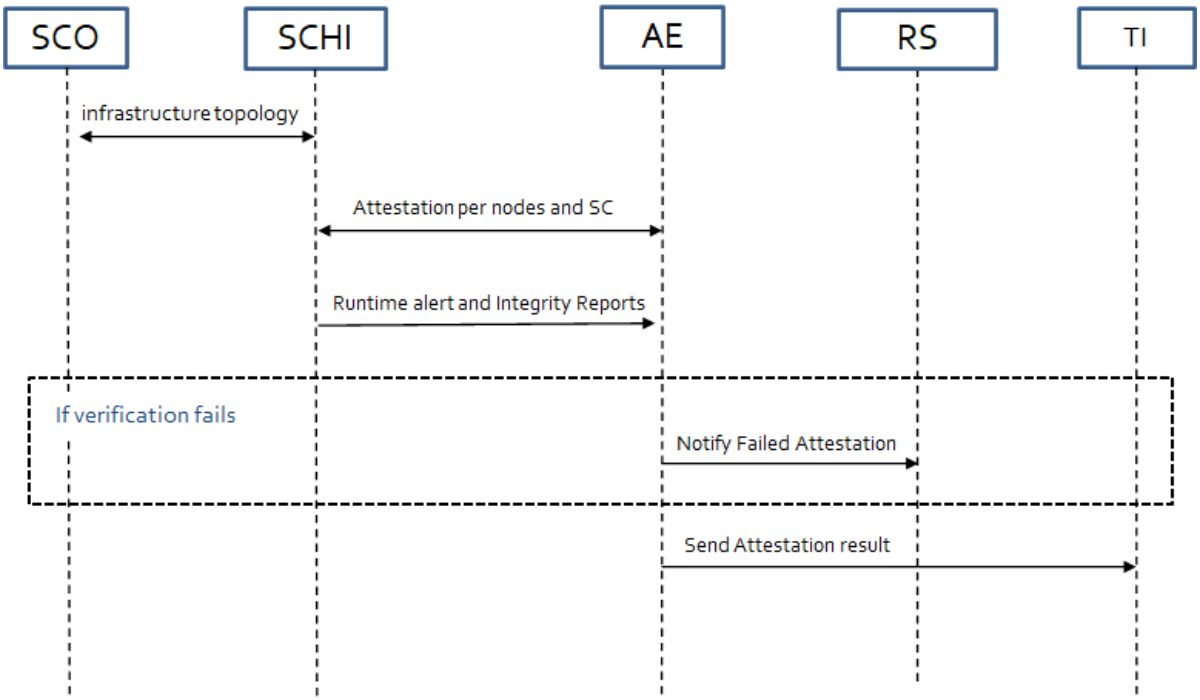


Figure 3: Periodic attestation Workflow

In order to maintain the trustworthiness of the infrastructure and services, the Attestation Engine performs periodic runtime attestations after the initial attestation as interpreted in Figure 3. The SCHI periodically notifies infrastructure topology updates to the Attestation Engine to keep the AE aware of the latest topology of the infrastructure. The infrastructure topology is iterated periodically per node and SC. If the verification fails at any time, the Attestation Engine sends a failed attestation notification to the RS. The failed attestation notification is also forwarded to Threat Intelligence for further analysis.

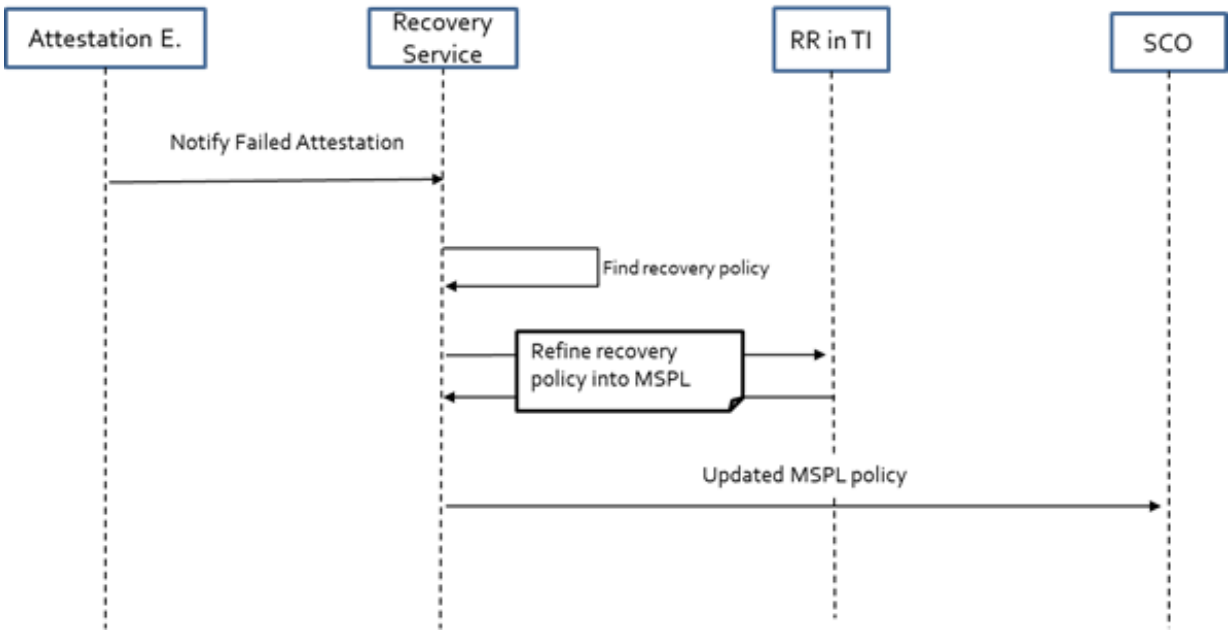


Figure 4: Remediation for failed attestation Workflow

In case of failed attestation, the Attestation Engine sends a failed attestation notification to the Recovery Service. The RS is responsible for applying remediation processes in case of failed attestation; therefore, it finds a suitable recovery policy based on the type of attestation failure. The RS then interacts with the Remediation and Recommendation (RR) component within TI to refine the recovery policy into a MSPL. The Recovery Service forwards that MSPL to the SCO and performs the remediation process accordingly as mentioned in Figure 4.

### 2.1.3. Interactions between the Attestation Engine and the other components

Based on the workflows defined previously, the interaction between the AE and the other components of PALANTIR is represented in Figure 5. The interaction with each component is described below.

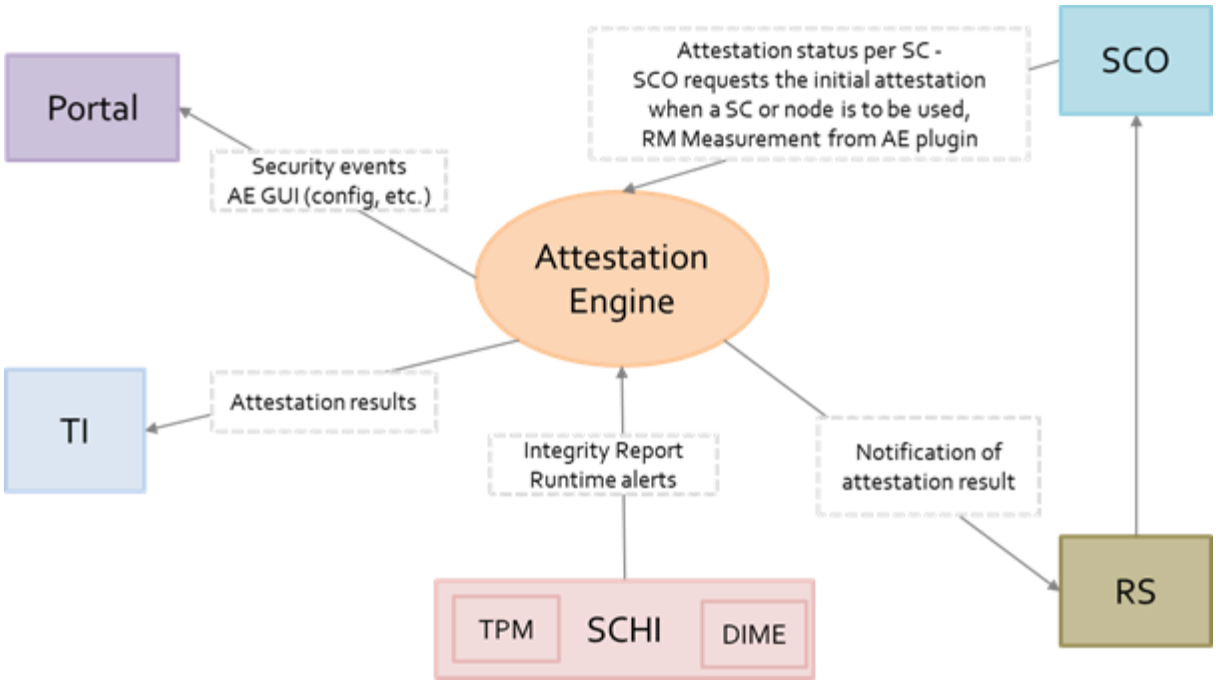


Figure 5: Inter-components interaction of Attestation Engine

#### Security Capability Orchestrator – Attestation Engine interactions

The SCO interacts with the Attestation Engine to notify deployment of a new node and Security Capability. The SCO also hosts an Attestation Engine plugin that interacts with the AE through a Kafka topic to update Reference Measurement to AE.

#### Security Capability Hosting Infrastructure – Attestation Engine interactions

The other communication is from SCHI to Attestation engine in order to retrieve attestation proof from TPM measurements, which is fetched from REST API defined in the Attestation Engine. The integrity report and runtime alerts are sent from the SCHI to the Attestation Engine through an agent in SCHI. This is done by the Redfish API defined in the Attestation Engine agent (DIME) in SCHI.

#### Recovery Service – Attestation Engine interactions

After the node verification, the attestation result is forwarded to the Recovery Service. The Recovery Service suggests the SCO to exclude the node/SC in case of failed attestation and to trust the node/SC in case of successful attestation. This is done by a Kafka topic defined by the Attestation Engine

#### Threat Intelligence – Attestation Engine interactions

The periodic attestation results and runtime alerts are sent to the Threat Intelligence by the Attestation Engine through a Kafka topic defined by the Threat Intelligence for further analysis.

#### Portal – Attestation Engine interactions

The security events related to the attestation results are also sent to the Portal through Kafka topic defined by the Portal.

### 2.1.4. Internal Operation

The Attestation Engine keeps an updated list of known-good reference measurements about platform nodes, software packages and valid configurations to assess the trust of the infrastructure and Security Capabilities. It interacts with the SCO to retrieve the information needed, per node, for performing attestation of SCs. The Attestation Engine maintains an updated view of infrastructure topology from the SCO. This information is used by the Attestation Engine to detect any compromised node by periodic attestation.

The Attestation Engine also performs runtime verification by continually monitoring the SCHI memory, logging violations such as kernel memory modification, and sending alerts to the AE.

## 2.2. Fault & Breach Management

The Fault & Breach Management (FBM) component aims to enable the design and deployment of procedures and strategies with regard to fault and breach management and remediation by unifying, correlating and automating event handling across the end-to-end physical and virtual infrastructure.

The FBM utilizes inputs from the RR and the AE in a management and orchestration 'platform' for semi-automated design and deployment of notification, management and remediation policies. These policies are personalized by network/service operators and are modelled to their specific requirements like specific mechanisms to be triggered after a security issue is detected.

The goal of this component is to personalize the triage, reduce triage and mean-time-to-repair as well as to minimize exposure, while improving communication and collaboration. To this end, the FBM interacts with the TI, the SCO and the Portal Dashboard using Kafka protocol and topics.

### 2.2.1. Subcomponents

The FBM (Figure 6) is a fully virtualized system that consists of the Incidence Response (IR) and Recovery Service (RS) subcomponents. Both subcomponents are designed to carry out personalized intervention defined as a Finite State Machine (FSM).

In more detail, the purpose of the FBM component is as follows:

- A Finite State Machine (FSM) is a computing model that may be used to mimic sequential logic or to describe and regulate an execution flow. A state machine is a behaviour model with a finite number of states. Based on the current state and a provided input, the machine performs state transitions and creates outputs. A state is a description of a system's current state as it awaits the execution of a transition.
- IR engine: a PALANTIR-protected infrastructure hosts one IR instance that is responsible for triggering mitigation policies when a threat/attack related to data breach is detected by the TI.
- RS Engine: a PALANTIR-protected infrastructure hosts one RS instance that is responsible for triggering recovery policies when attestation faults are detected. In addition to automation and mitigation services offered by the SCO, the FSMs also include alerts/messages and interventions that require human intervention. In such cases, notifications or requests can be sent to the Dashboard based on the FSM result. The notification or request can then be handled by the PALANTIR operator.

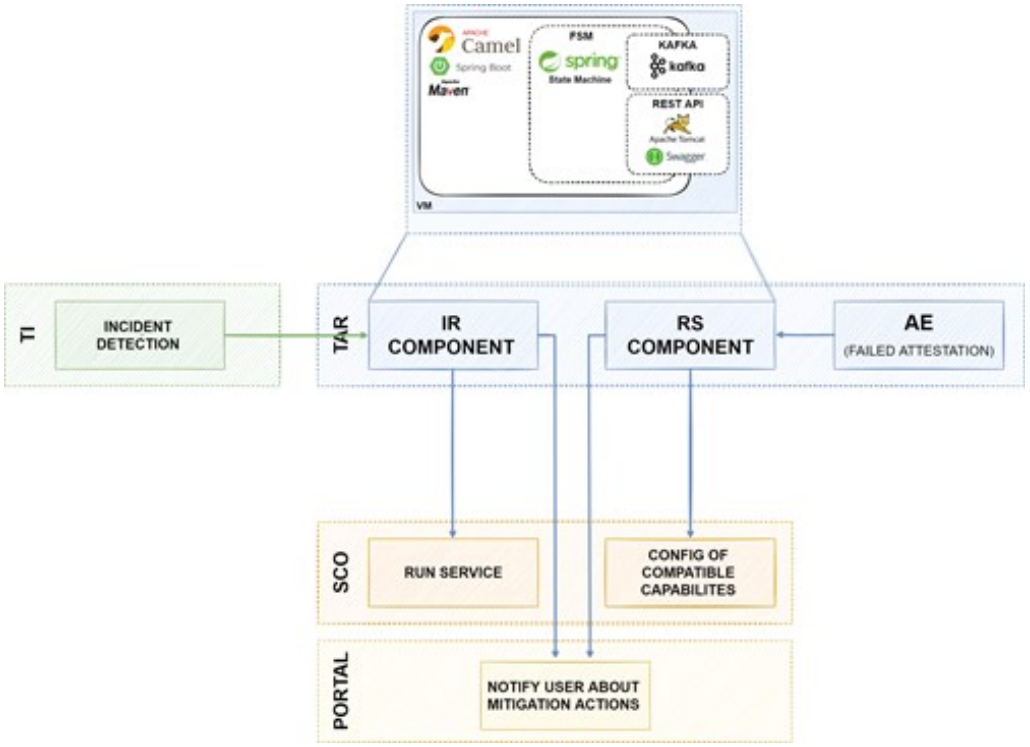


Figure 6: Conceptual workflow of the FBM.

Figure 6 represents the components and conceptual workflow of the PALANTIR FBM component. The objective of the IR is to handle remediation policies for threat mitigation, especially those that cannot be handled (for instance) by the cross-system or stakeholder notifications which may be specific for an individual entity (ME/SME). Figure 7 outlines a conceptual, functional diagram representing such a complex policy. This diagram is used as a baseline (generic) workflow when considering the FSM implementation options to be supported and the design of remediation actions personalized to the requirements of the ME/SME. As an example, a server is infected with malware and this is detected by TI. The goal would be to isolate infected server and create local copies to start a "chain of custody". In PALANTIR there is an FBM policy created to isolate and, if needed, to shut-down the infected server. The execution of this policy within the IR subcomponent can be triggered automatically via the TI.

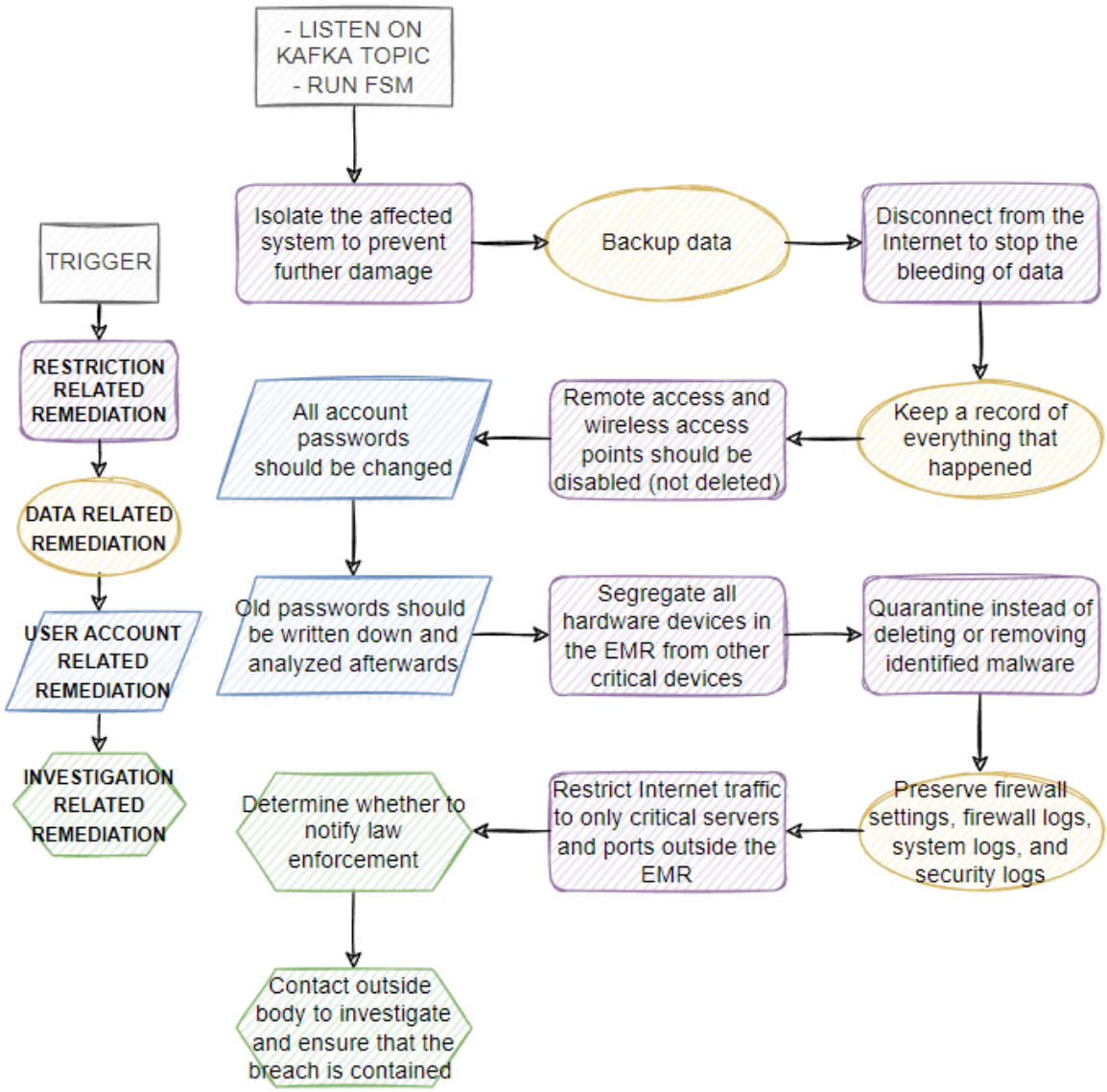


Figure 7: Data Breach and Remediation workflow diagram.

The goal of the RS component is to periodically check the status of the PALANTIR platform and to respond to possible faults or malfunctions. Similarly to the IR, a RS policy is defined with SME/ME specific recovery goals. Below is an initial set of possible recovery services functionalities:

- Verify whether services are running.
- Compare what is supposed to run against what is actually running.
- Check whether services are responding.
- Check whether services are working correctly.

Figure 8 illustrates the design of PALANTIR's FBM component to support the IR and the RS components. The FBM supports two different implementations of FSMs.

- FSM1 can be triggered through Kafka or REST interactions, or be pre-started. The FSM1 implements a REST API to communicate with the REST endpoint/s while FSM1 is executed. It

is best suited for ''simple'' scenarios, i.e. messaging and alerts, or where services triggered require manual intervention (i.e. password changes, user policy management changes, etc.).

- FSM2 is better suited for complex operations, for example where interactions with the PALANTIR components (e.g. the SCO) are foreseen. To this end, FSM2 implements an internal Kafka Producer which can interact with multiple PALANTIR components.

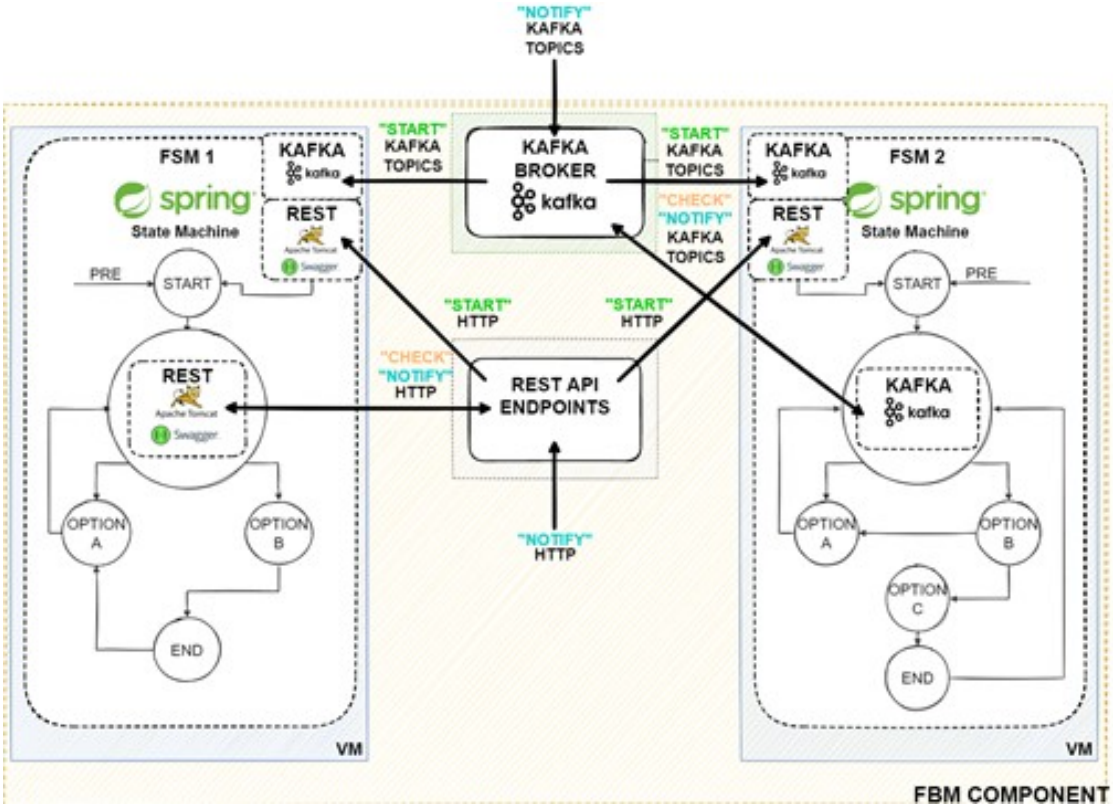


Figure 8: PALANTIR FBM with 2 FSMs variants.

### 2.2.2. General Workflow

The general workflow of the IR component is presented in Figure 9, where two different scenarios and two differently modelled FSMs are triggered by the TI.

- In the first scenario S1, a data breach detected by TI is triggering the FSM1, which alerts the user in the PALANTIR Portal about the detected data breach. Shortly after, the IR starts running the service defined for the detected incident (e.g. Data Backup) and notifies the user again about the action performed.
- In the second scenario S2, a data breach detected by TI is triggering the FSM2, which asks for user intervention through service reconfiguration and awaits user response. Once the confirmation is received from the user about the reconfiguration, the IR component refines the response policy with the RR in the TI. Then the IR deploys the recommended service, gets the information back from the SCO and finally can send the notification to the Portal.

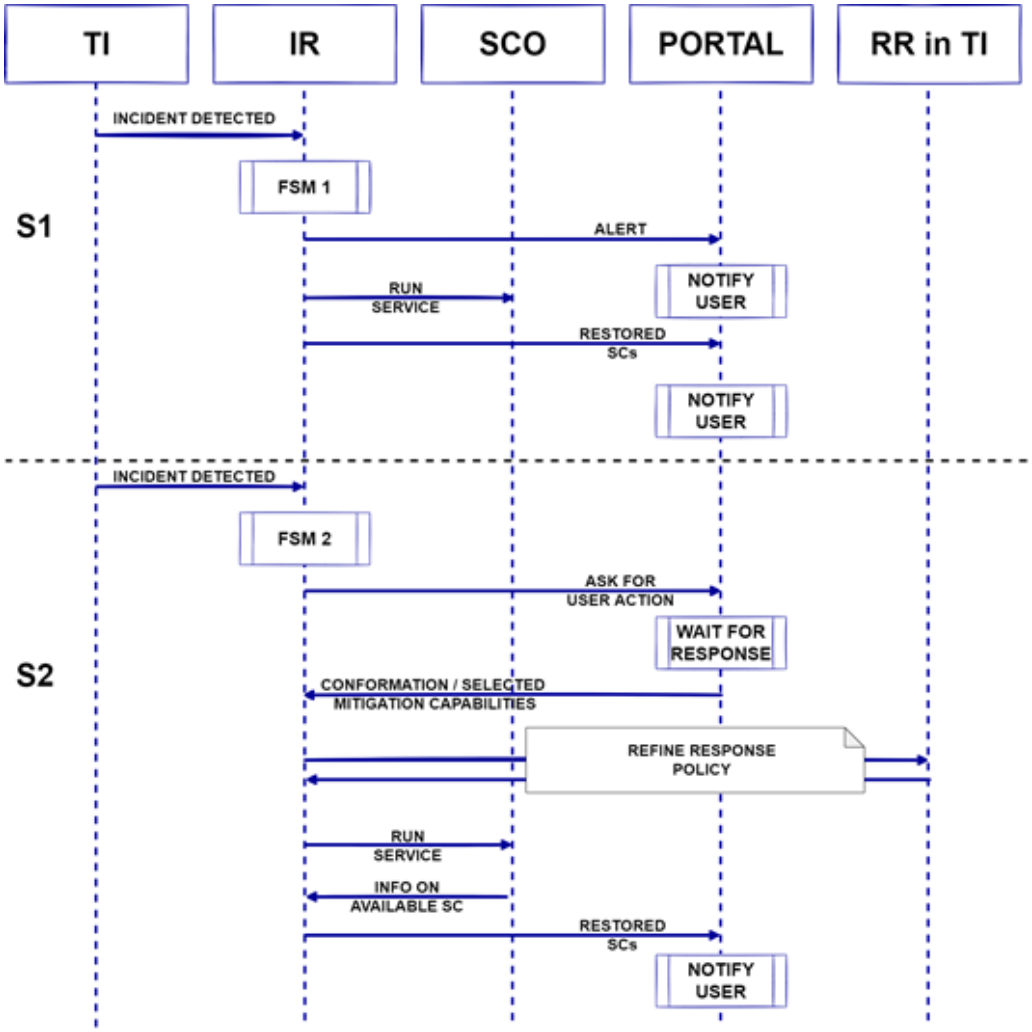


Figure 9: Two Workflow scenarios handling the data breaches.

The workflow of the RS component is shown in Figure 10. Again, two possible scenarios with different FSMs are presented here.

- In scenario S1, the AE notifies the RS about a failed attestation that triggers FSM1, which requests the compatible capabilities to the SCO, then decides the recovery policy to be applied and sends a notification to the user via the Portal.
- Scenario S2 triggers FSM2 upon notification from the AE and, based on the payload of the message, asks for user intervention regarding the configuration. It waits for confirmation from the user and retrieves the configuration of the compatible capabilities from the SCO. The recovery policy is then refined and sent to the RR in the TI. The updated policy is forwarded to the SCO, and a notification can be displayed on the Portal.

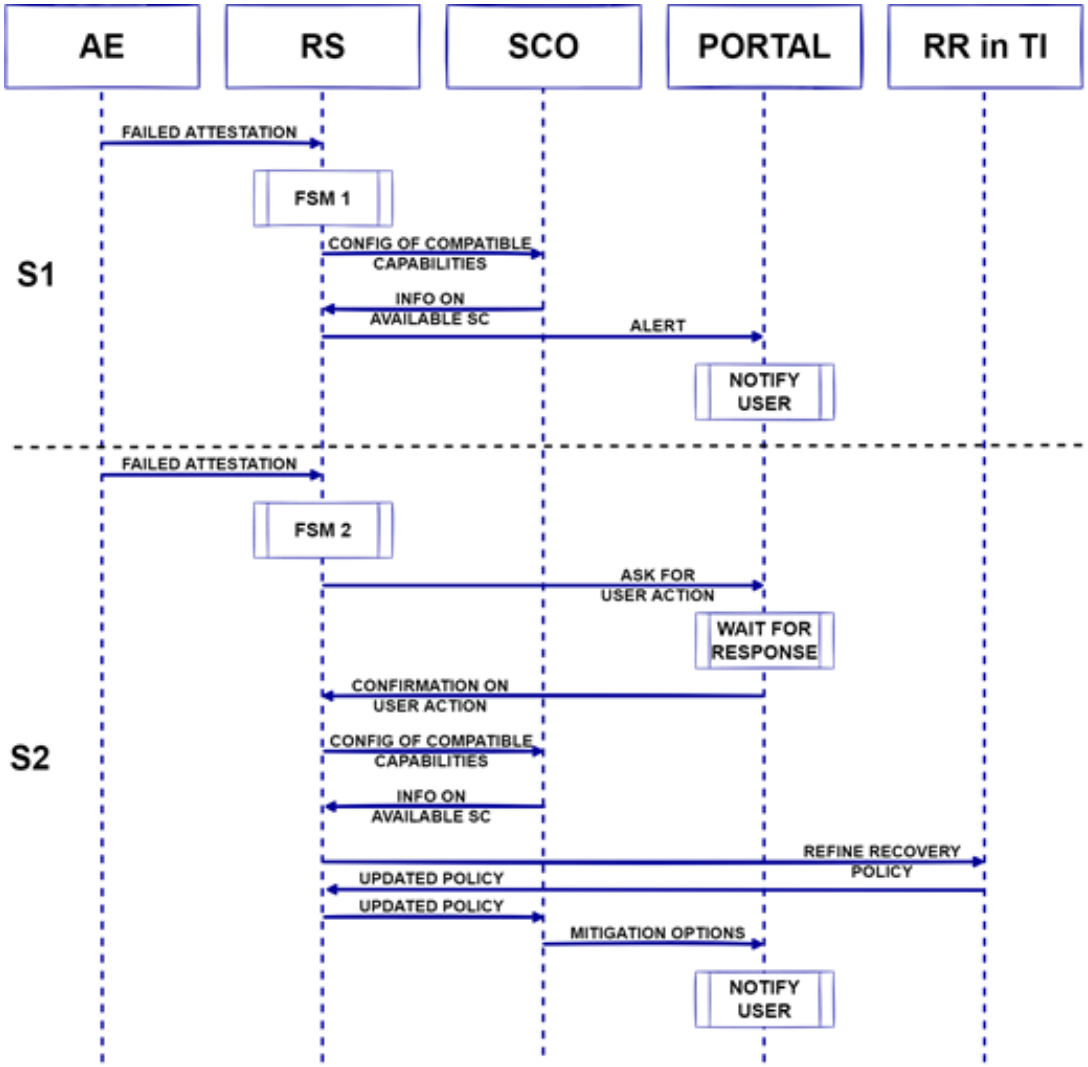


Figure 10: Two Workflow scenarios for triggering the Recovery Policies.

# 3. Specifications

This section details how the PALANTIR requirements, provided in D2.1, are addressed by each component of the Trust and Attestation Framework.

## 3.1. Attestation Engine

Before going into the implementation section, the AE requirements from D2.1 are reminded to the reader in Table 2, which also explains the choices to fulfil the requirements related to the AE.

### 3.1.1. Attestation Engine requirements

D2.1 specifies the generic requirements for most PALANTIR components and then specific requirements for each component. This section describes how the AE meets those requirements.

| Req. ID | Requirement description |
|---|---|
| R1.1.4 | The platform SHOULD implement communication between PALANTIR components with a lightweight message queue |
| *The AE relies on the Apache Kafka [3] software bus, which is the preferred choice within the PALANTIR project. For compatibility with existing open-source project, the AE uses the Redfish and Keylime APIs between the AE and the SCHI nodes.* | |
| R1.2.5 | The PALANTIR-introduced security mechanisms should be transparent to the operation of vertical applications. |
| *The technologies, methods and protocols used by the AE do not impact the operation of the PALANTIR's end user applications.* | |
| R1.2.8 | Technologies used by PALANTIR SHOULD be trustable. |
| *The AE leverages Reference Measurements that enable the stakeholders to ensure that the running software is the one they wanted.* | |
| R1.2.9 | The PALANTIR system MUST provide security mechanisms to ensure that user (and endpoints) data are securely processed and stored wherever it is processed or stored. |
| *By implementing attestation and runtime monitoring, the AE provides PALANTIR with a way to ensure that the appropriate hardware, firmware, software and configuration are used to process data.* | |
| R1.3.30 | The platform SHOULD provide network isolation for compromised systems. |
| *By implementing attestation and runtime monitoring, the AE provides PALANTIR with a way to detect compromised systems and then isolate them through the appropriate recovery policy in the RS.* | |
| R1.4.1 | PALANTIR SHOULD deploy mechanisms for the periodic attestation of the platform and the running applications', services' and configurations' integrity. |
| *The AE deploys both initial and periodic attestation of the hardware, firmware and software running on the SCHI.* | |
| R1.4.2 | PALANTIR SHOULD recover from threats on the Security Capability Hosting Infrastructure. |
| *Upon failure of an attestation, or reception of a security alert, from the SCHI, the AE notifies the RS, which in turn applies the appropriate recovery policy.* | |
| R1.4.3 | The platform SHOULD be able to identify and isolate network segments, data or equipment at risk and enable automatic redundancy and (offline) data backup service to prevent corruption or loss of data. The risks are recognised complex reflected primarily in unexpected/unusual behaviour. |

| Req. ID | Requirement description |
|---|---|
| *By implementing attestation and runtime monitoring, the AE provides PALANTIR with a way to detect compromised systems and then isolate them through the appropriate recovery policy in the RS.* | |
| R1.4.4 | The platform SHOULD be able to collect and analyse the status and health of the underlying infrastructure, including components at risk due to improper communication security (i.e., no or weak encryption), weak passwords, or irregular updates. |
| *By implementing attestation and runtime monitoring, the AE provides PALANTIR with a way to detect incorrect configuration or missing updates in the SCHI.* | |

Table 1: Requirements related to Attestation Engine

## 3.2. Fault & Breach Management

D2.1 specifies the generic requirements for most PALANTIR components and then specific requirements for each component. This section describes how the FBM meets those requirements in Table 2.

| Req. ID | Requirement description |
|---|---|
| R1.1.4 | The platform SHOULD implement communication between PALANTIR components with a lightweight message queue |
| *The FBM supports the Apache Kafka [3] software bus, which is the preferred choice within the PALANTIR project. For compatibility with existing open-source projects, the FBM relies on a SpringBoot implementation.* | |
| R1.2.5 | The PALANTIR-introduced security mechanisms should be transparent to the operation of vertical applications. |
| *The FSMs represent a convenient and user-friendly way of specifying security functionalities and system responses in a user-understandable way. This means that end-users can easily foresee and model the impact of the PALANTIR security mechanism on day-to-day operations. However, the FBM has no impact on the operation of the PALANTIR's end user applications.* | |
| R1.3.3 | The platform SHALL provide a variety of SecaaS packages on the Catalogue. |
| *The PALANTIR FSM enables a unique way of integrating automated and supervised actions into holistic management and remediation policies, personalized to the ME/SME requirements. The FBM implements the link between the services (including isolation) and the users. Within the PALANTIR Project, generic FSMs are offered, which can be further modelled by PALANTIR Operators.* | |
| R1.3.9 | The platform SHOULD be able to monitor the deployed security capabilities and expose such data through programming interfaces for other internal components. |
| *The RS component enables the implementation of periodic checks of the status of PALANTIR architecture. The FBM implements the link between the services (including isolation) and the PORTAL and can report on the status. Using FSMs and a Kafka Software bus, the PALANTIR operators can also design more complex, automatic or hybrid (with manual interventions), workflows* | |
| R1.3.12 | The platform SHALL deploy in cloud/hosted and edge SecaaS delivery modes. |
| *The FBM is deployed as a cloud service that supports REST API and Apache Kafka software buses to interact with the rest of the PALANTIR services. Thus it can be deployed virtually anywhere; however, close to edge deployments are preferred. The FBM components can be deployed on a VM or as a Docker solution.* | |

| Req. ID | Requirement description |
|---|---|
| R1.3.13 | The platform SHOULD deploy in lightweight SecaaS delivery mode with minimal computational resources. |
| *The Spring Boot Framework used to implement FBM has a low memory footprint. It is coupled with Maven—a project-management tool that is based on a POM (project object model) representation. The selected technologies enable full customisation of the end-user applications and required libraries through a single XML file. With Spring Boot Thin Launcher we can also create 'thin' Jar deployments especially relevant for microservice development.* | |
| R1.3.14 | The platform SHALL be able to retrieve the basic status for the security capabilities instantiated or available (in the Catalogue). |
| *The RS component enables the implementation of periodic checks of the status of PALANTIR architecture and respond to possible faults or malfunctions.* | |
| R1.3.30 | The platform SHOULD provide network isolation for compromised systems. |
| *By implementing attestation and runtime monitoring, the AE provides PALANTIR with a way to detect compromised systems and then isolate them through the appropriate recovery policy in the RS.* | |
| R1.4.1 | PALANTIR SHOULD deploy mechanisms for the periodic attestation of the platform and the running applications', services' and configurations' integrity. |
| *RS enables the definition of personalized policies that is executed as FSMs to provide the needed attestation, configuration and remediation.* | |
| R1.4.2 | PALANTIR SHOULD recover from threats on the Security Capability Hosting Infrastructure. |
| *IR component is designed to complete this requirement by implementing FSMs with steps for recovering from threats. Upon threat detection a specific policy is triggered within* | |
| R1.4.3 | The platform SHOULD be able to identify and isolate network segments, data or equipment at risk and enable automatic redundancy and (offline) data backup service to prevent corruption or loss of data. The risks are recognised complex reflected primarily in unexpected/unusual behaviour. |
| *Similar to R1.3.30, another pre-modelled FSM or even multiple FSMs are executed upon receiving TI threat findings (JSON message) to satisfy this requirement.* | |
| R1.4.4 | The platform SHOULD be able to collect and analyse the status and health of the underlying infrastructure, including components at risk due to improper communication security (i.e. no or weak encryption), weak passwords, or irregular updates. |
| *By implementing the Kafka broker and clients, the platform is able to collect the infrastructure information.* | |

Table 2: Requirements related to the FBM.

Table 3 outlines the Remediation Actions for Data Breaches associated with the D2.1 requirements that are used to create the FSMs. This way we can form the FSMs to help in mitigation of specific threats.

| Remediation Action description | Req. ID |
|---|---|
| Isolate the affected system to prevent further damage | R1.3.30 |

| Remediation Action description | Req. ID |
|---|---|
| Disconnect from the Internet by pulling the network cable from the firewall/router to stop the bleeding of data | R1.3.30 |
| Keep a record of everything that happened with next information:<br>*How did you find out about the alleged breach*<br>*When and how you were notified, as well as when and how you were notified*<br>*What the notification said to you*<br>*All actions conducted between the time of notification and the incident's conclusion*<br>*The date and time when systems were disconnected from the Internet*<br>*When you disable remote access (and when you don't)*<br>*When and if you updated your account passwords/credentials*<br>*All other system hardening or remediation steps taken* | R1.4.3,<br>R1.4.4 |
| Remote access and wireless access points should be disabled (not deleted) | R1.4.3 |
| All account passwords should be changed, and non-critical accounts should be disabled (not deleted) | R1.4.4 |
| Old passwords should be written down and analysed afterwards | R1.3.3 |
| Change access control credentials (usernames and passwords) and implement highly complex passwords, or even better, use passphrases | R1.4.3 |
| Segregate all hardware devices from other critical devices. Relocate these devices to a separate network subnet and keep them powered on to preserve volatile data | R1.4.3 |
| Quarantine instead of deleting or removing identified malware found by your antivirus scanner for later analysis and evidence | R1.3.30 |
| Preserve firewall settings, firewall logs, system logs, and security logs. Take screenshots if necessary | R1.4.3 |
| Restrict Internet traffic to only critical servers and ports outside the database. If you must reconnect to the Internet before an investigator arrives, remove your database from any devices that must have Internet connectivity until you consult with your forensic investigator | R1.4.3 |
| Determine whether to notify law enforcement | R1.3.3,<br>R1.3.9 |
| Contact an outside body experienced in managing data breaches to immediately investigate and ensure you've properly contained the breach | R1.3.3,<br>R1.3.9 |

Table 3: Data Breach and Remediation steps.

# 4. Implementation

This section explains the implementation details of the different subcomponents within the Trust, Attestation and Recovery component. Information including specific technologies and techniques and their benefits is provided.

## 4.1. Attestation Engine

The Attestation Engine for PALANTIR is implemented in two different flavours, one by HPELB and another by POLITO. Both implementations are interoperable as they comply with the architecture presented above in Section 2.1.

### 4.1.1. Design and technologies of the POLITO AE

The POLITO AE is a monitoring entity of an NFV platform intending to provide periodic remote attestation of both the NFV Infrastructure and SCs running inside containers, ensuring that the deployed SCs are trustworthy. The AE has been developed with a modular architecture, whose subcomponents are represented in Figure 11.

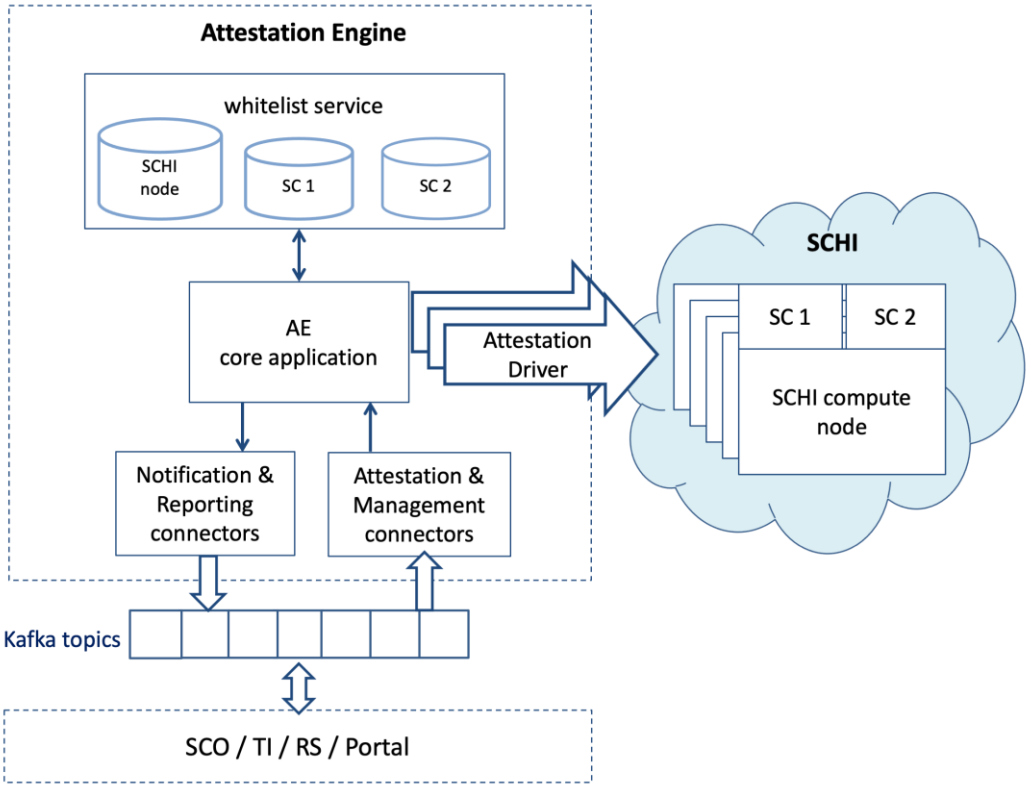


Figure 11: POLITO AE in-depth architecture

These subcomponents are described below:

- The *AE core application* receives from the SCO requests for registering new compute nodes and SCs, to be deployed in the SCHI through the *Attestation & Management Connectors*. Moreover, it provides the *integrity verification* functionality by means of *Attestation Drivers*, developed as plugins, that allow instantiating different remote attestation workflows, depending on the type of host, each one with a specific verification logic. This makes the AE able to attest nodes with different architectures (e.g., x86, ARM) and with RoTs based on different

technologies (e.g., hardware TPM, ARM TrustZone [15], Intel SGX [16] or AMD SEV [17], but also virtualised Trusted Execution Environments (TEE), although these offer lesser assurance than hardware-based ones), avoiding the lock-in with a particular vendor.

- The POLITO AE supports the *Notification and Reporting* functionality of integrity status information to external entities by means of pluggable connectors that allow the interaction of the AE with the other PALANTIR components, in particular the RS, the TI and the Portal.
- The POLITO AE relies on the *Whitelist Service* component for the creation of the whitelists for both the host systems and the containers, by relying on the information retrieved from the SCO and the SCC.

The operational workflow of the POLITO AE is compliant with the one described in section 2.1.2.

The POLITO AE has been implemented as a set of microservices, where the AE core application holds a central role. Each of these components is developed using the Python language, with a Docker [18] file that allows its deployment through the Docker container engine; in addition, the Docker Compose [19] tool has been used to quickly and efficiently instantiate all the AE sub-components and easily enable network interaction between them. The AE core application leverages Django REST web framework [20] and exposes APIs to register the computational nodes and SCs deployed in the SCHI. Other APIs are exposed to attest SCs and to verify the correct configuration of all the AE's subcomponents. The Whitelists Service is in charge of centralizing the creation and management of the whitelists for physical hosts and containers. This component communicates with the other POLITO AE sub-components through REST APIs. The Attestation & Management and Notification & Reporting Connectors have not yet been implemented. The idea is to develop them as Kafka consumers and producers, providing connectivity between the POLITO AE and the other PALANTIR components (e.g., SCO, RS, TI, and the Portal).

The integrity verification functionality provided by the POLITO AE relies on the Keylime framework [5], which supports periodic remote attestation of compute nodes based on TPM 2.0 as hardware RoT. The POLITO AE does not use the basic version of the Keylime framework but an extended version. This version supports a custom attestation solution, developed by POLITO, which allows attesting separately the host system and each SC running on it. This solution defines a new Linux *Integrity Measurement Architecture* (IMA) template, which associates an entry of the IMA Measurement Log (ML) with the host system or a specific container by relying on properties owned by the containerized processes, valid for most of the Open Container Initiative (OCI) Container Runtimes currently in use. The remote attestation of OCI-containers based on this solution relies on a hardware RoT since it leverages IMA and TPM 2.0 technologies. Moreover, unlike the basic version of Keylime, this extended version can perform the integrity verification of the IMA ML by relying on any Platform Configuration Register (PCR) bank present in the TPM 2.0, not just the SHA-1 bank. Further details on the Keylime framework and the attestation driver developed to integrate it within the POLITO AE are in turn in the following sections.

### Keylime Architecture

In this section we briefly introduce the Keylime overall architecture and its main components.

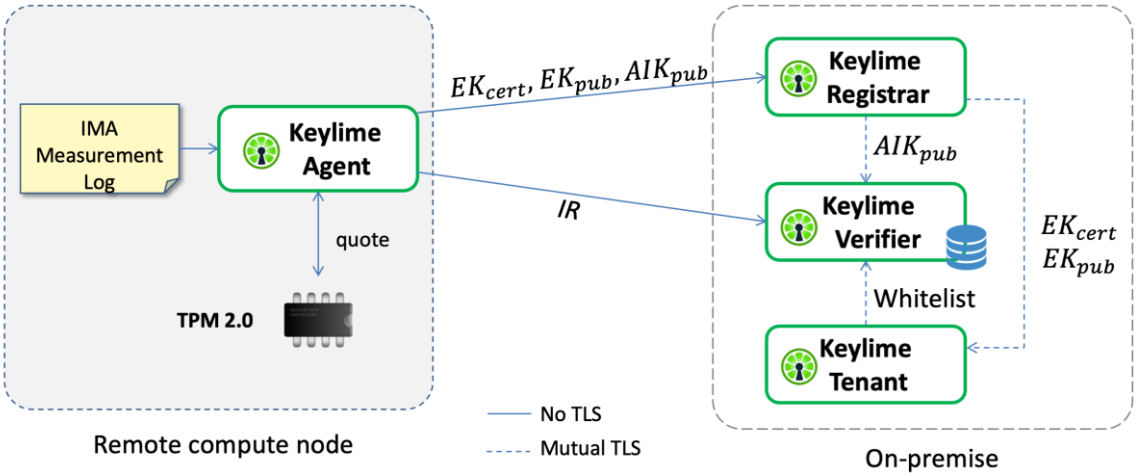


Figure 12: Keylime architecture

The Keylime architecture is based on the interaction of the four main modules depicted in Figure 12:

- the *Keylime Tenant* is the module that kicks off the framework; it registers the *Keylime Agent* with the *Keylime Verifier*, sending it all the information (whitelist, exclude list and TPM policy) necessary to start the periodic remote attestation on the node, and verifies the authenticity of the TPM of the remote platform, by checking the validity of the certificate of the TPM's Endorsement Key ($EK_{cert}$);
- the *Keylime Agent* is a service running on the remote compute node to be attested and is in charge of sending the Integrity Reports (IRs), constituted by a TPM 2.0 quote and the IMA ML, to the *Keylime Verifier*;
- the *Keylime Registrar* receives the TPM credentials from the Agent and sends them to the Verifier and Tenant to allow them to verify the TPM quotes and the $EK_{cert}$, respectively;
- the *Keylime Verifier* is responsible for evaluating the integrity level of the remote platform, verifying the Integrity Reports based on the whitelist received from the Tenant and of the $AIK_{pub}$ key received from the *Keylime Registrar*.

### Keylime Attestation Driver

An Attestation Driver is a specific implementation of a generic integrity verification interface, which exports six methods:

- **registerNode()** for registering a new compute node in a given attestation framework;
- **registerSC()** for registering a new SC deployed on a compute node;
- **pollHost()** for checking the integrity state of a compute node and all the SCs running on it through the attestation framework;
- **getStatus()** for getting the status of the attestation framework, that is, if it has been properly configured and is active;
- **deleteNode()** for removing a compute node previously registered in the attestation framework;
- **deleteSC()** for removing a SC associated to a compute node.

The POLITO AE supports an Attestation Driver for the Keylime framework; the workflows for the registration of compute nodes or SCs and the integrity check of them are described below.

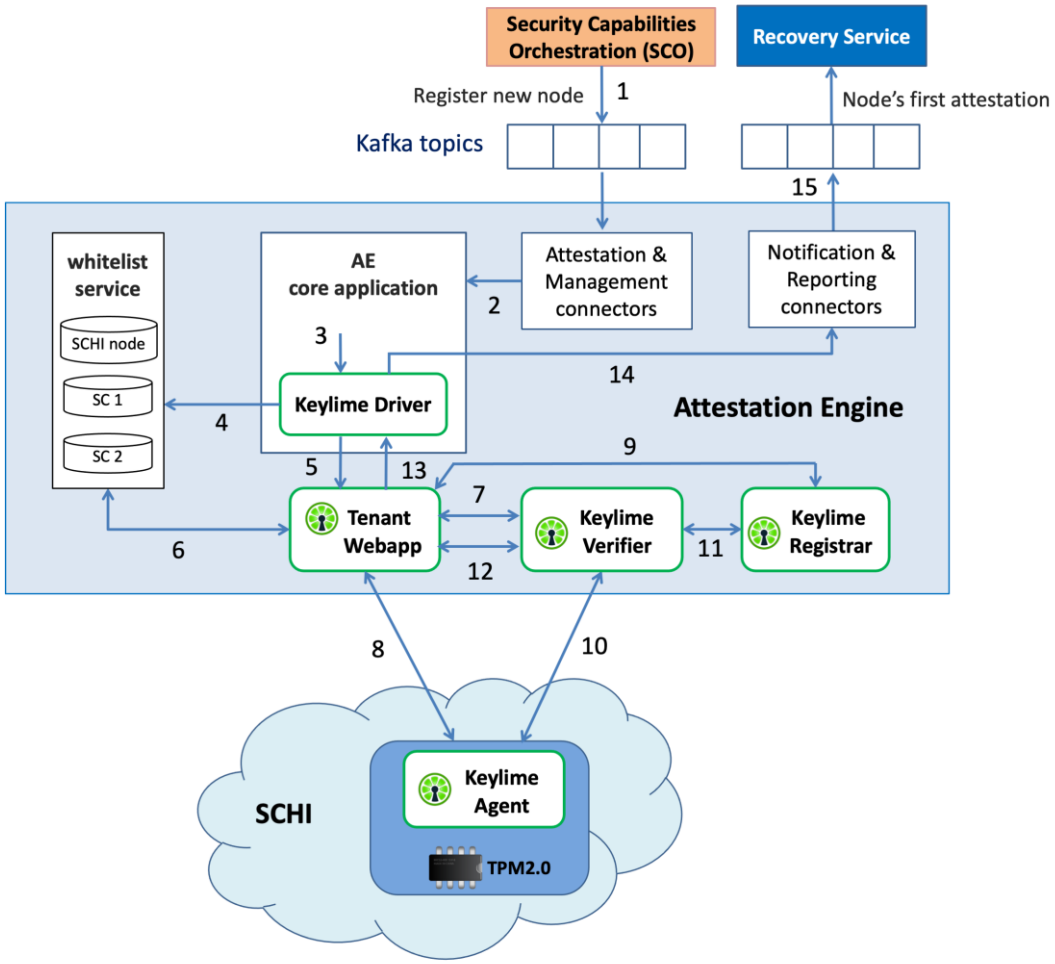


Figure 13: New compute node registration workflow with Keylime Driver

Figure 13 depicts the interactions triggered when the SCO adds a new compute node in the SCHI:

1. the SCO posts, in a specific Kafka topic, a message which is pulled by an *Attestation & Management connector*;
2. the *connector* notifies the *AE core application* about the presence of a new computational node to be periodically attested;
3. the *AE core application* invokes the **registerNode**() method exposed by the *Keylime Driver*;
4. the *Keylime Driver* requests the *Whitelist Service* to create the whitelist corresponding to the new compute node;
5. the *Keylime Driver* registers the new compute node with the Keylime framework by sending a REST request to the *Tenant Webapp*; the request body specifies the URL through which the node's whitelist can be downloaded, the exclude list and the TPM policy;
6. the *Keylime Tenant Webapp* downloads the whitelist specific for the compute node from the *Whitelist Service*.

Steps 7, 8, 9, 10 and 11 concern the "*Three Party Bootstrap Key Derivation Protocol*", described in the Keylime whitepaper [21], at the end of which the *Keylime Verifier* begins the periodic remote attestation on the compute node:

12. the *Keylime Tenant Webapp* retrieves the integrity status of the new compute node from the Verifier;
13. the *Keylime Tenant Webapp* sends the registration response to the *Keylime Driver*, specifying the integrity status of the new compute node;

14. the *AE core application* creates a JSON object describing the integrity status of the new compute node and sends it to a *Notification & Reporting* connector;
15. the *connector* posts the received message to a specific Kafka topic, from which the RS can retrieve the information of the first attestation of the computational node.

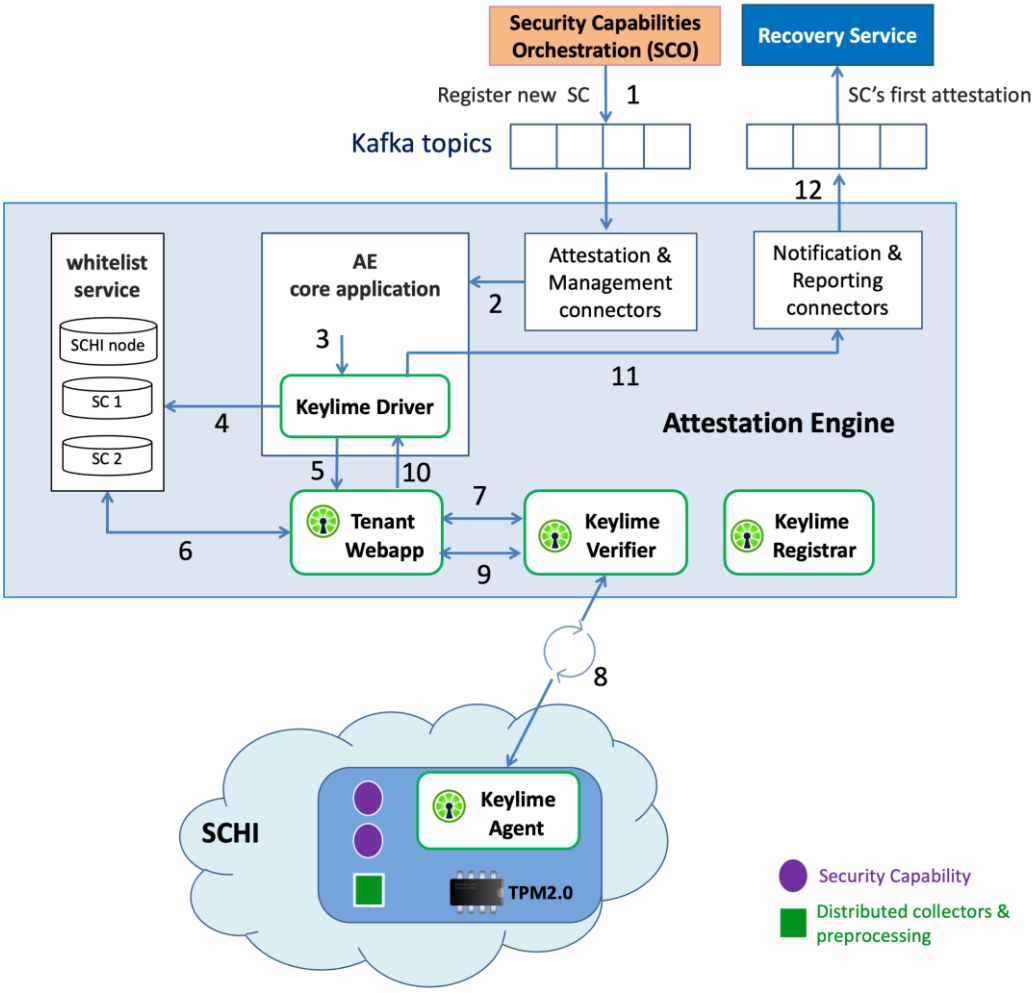


Figure 14: New SC registration workflow with Keylime Driver

Figure 14 depicts the workflow that takes place when the SCO deploys a new SC on a compute node previously registered:

1. the SCO posts a message in a specific Kafka topic in order to notify this event; the message is then pulled from an *Attestation & Management* connector;
2. the *connector* notifies the *AE core application* about the presence of a new SC deployed on a given computational node;
3. the *AE core application* invokes the **registerSC**() method exposed by the *Keylime Driver*;
4. the *Keylime Driver* requests the *Whitelist Service* to create the whitelist for the new SC;
5. the *Keylime Driver* registers the new SC with the Keylime framework; the request body specifies the compute node on which the SC is deployed, the URL through which the SC's whitelist can be downloaded and the exclude list;
6. the *Keylime Tenant Webapp* downloads the SC's whitelist from the *Whitelist Service*;
7. the *Keylime Tenant Webapp* registers the new SC in the Keylime Verifier, associating it with a previously registered *Keylime Agent*;
8. during the periodic remote attestation on the compute node, the *Keylime Verifier* attests the new SC on the basis of the whitelist and exclude list received from the *Tenant Webapp*;

9. the *Keylime Verifier* sends to the *Tenant Webapp* the integrity status of the new SC;
10. the *Keylime Tenant Webapp* sends the registration response to the *Keylime Driver*, specifying the SC's integrity status;
11. the *AE core application* creates a JSON object describing the SC's integrity status and sends it to a *Notification & Reporting* connector;
12. the connector posts the received message to a specific Kafka topic, from which the RS can retrieve the SC's first attestation information.

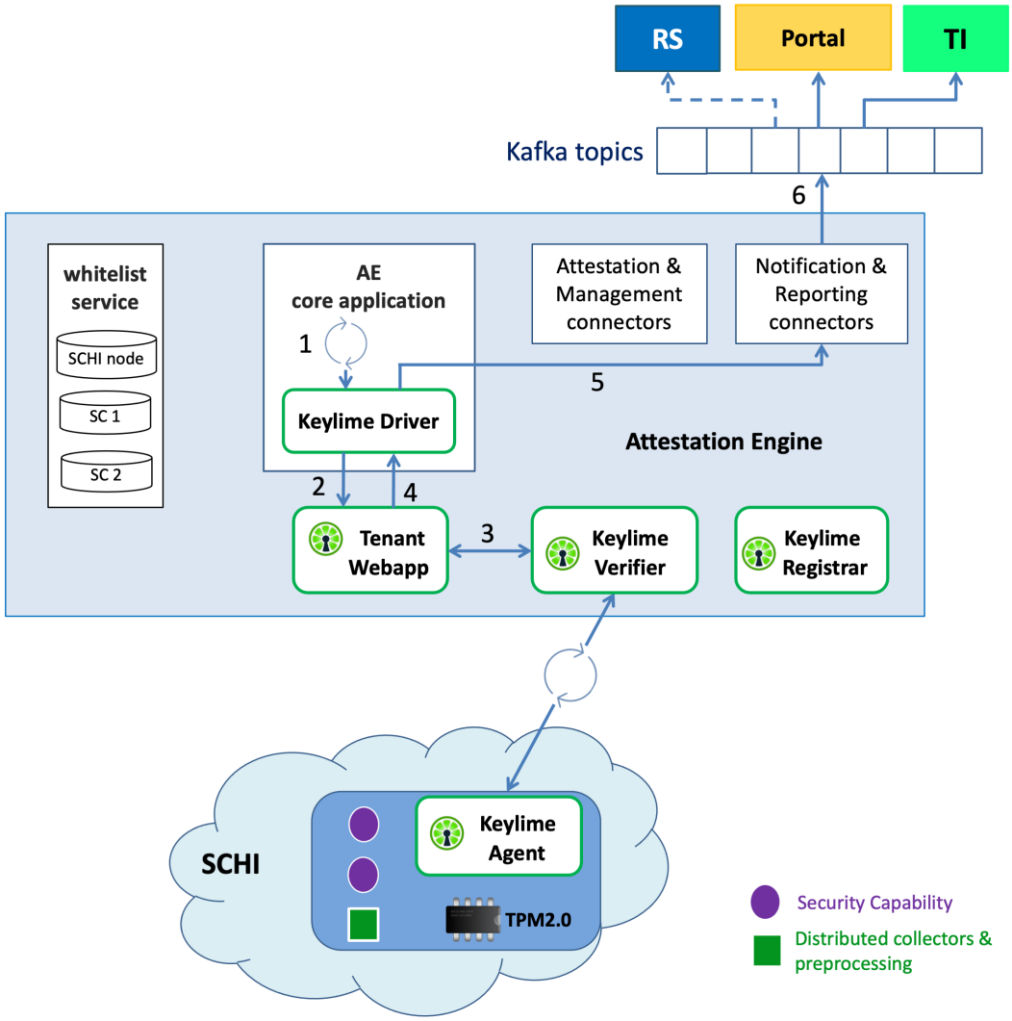


Figure 15: Workflow for the integrity check of a compute node with deployed SCs

Figure 15 depicts the workflow for checking the integrity status of all the SCHI's compute nodes and of the SC's running on them:

1. the *AE core application* periodically invokes the **pollHost**() method exposed by the *Keylime Driver* for each compute node previously registered;
2. the *Keylime Driver* asks the *Tenant Webapp* for the integrity status of a given node;
3. the *Keylime Tenant Webapp* retrieves the current integrity status of the compute node and the SCs running on it from the *Keylime Verifier*;
4. the *Keylime Tenant Webapp* sends the response to the *Keylime Driver*;
5. the *AE core application* creates a JSON object describing the current integrity status of the node and the SCs and sends it to a *Notification & Reporting* connector;

6. the *connector* posts the received message to a specific Kafka topic, from which Portal and TI can retrieve the integrity status information; if the message describes an integrity failure, the connector also posts it to a Kafka topic from which the RS retrieves information.

### 4.1.2. Design and technologies of the HPELB AE

The Attestation Engine from HPELB is based on an internal project called Project Aurora [4], which enables zero-trust security architectures from edge-to-cloud to transform security from a barrier to an innovation accelerator. The Project Aurora software is extended to support the additional PALANTIR requirements that are not currently addressed.

#### HPELB AE extensions

Beyond what is supported in Keylime, the HPELB AE implements the following capabilities.

##### HPELB DIME

The HPELB AE benefits from DIME [2], which is a HPELB solution for memory inspection to provide continuous runtime verification of critical memory regions such as the kernel space. DIME is implemented as a firmware agent, coupled with a kernel module, in the SCHI that sends runtime alerts to the AE. The HPE server deployed in the PALANTIR testbed embeds the DIME firmware agent, which is available to the PALANTIR consortium, particularly POLITO and its AE.

##### Hardware attestation

The HPELB AE leverages the TCG Platform Certificate Profile standard [6] to bootstrap its hardware attestation feature. After the initial attestation of the component of a server, the HPELB AE can track hardware changes throughout the lifecycle of the server to ensure that no unauthorised change happens.

##### TCG IDevID and IAK

The HPELB AE leverages the TPM 2.0 Keys for Device Identity and Attestation standard [7] when creating the cryptographic keys and certificates used to perform attestation (Initial Attestation Key – IAK) or authentication (Initial Device IDentity – IDevID). The TCG DevID specification implements the IEEE 802.1AR Secure Device Identity standard [8].

##### Appraisal policies

The HPELB AE implements an appraisal – or verification – policy engine that allows users to tailor the verification to their need. For example, the HPELB AE can track hardware changes, verify that UEFI Secure Boot is in place, monitor if firmware or software updates matches the reference measurement (i.e., is it an authorised update).

## 4.2. Fault & Breach Management

The initial FBM infrastructure is hosted on the Proxmox Virtual Environment (VE) [9], where each component is deployed as a Virtual Machine (VM). As depicted Figure 16, there are 3 VMs (components) building the architecture of the FBM.

Figure 16 outlines the experimental and development infrastructure established for the PALANTIR project. After the technologies are developed, the component is pushed to the common PALANTIR testbed. The Spring Boot application include and runs the Apache Camel 3.9.0 [10], Swagger (webjars) 3.51.2 [11] and Spring Statemachine 2.0.0 [12] dependencies. Apache Camel provides easy routing between synchronous (HTTP - REST) and asynchronous (Kafka) protocols. It also allows integration with other Java libraries. Swagger dependency prepared for Camel REST Domain Specific Language (DSL) provides simple setup and configuring of the REST endpoints for REST API. Once the Swagger is built and configured, it provides a Swagger UI interactive documentation accessible over the embedded Tomcat HTTP web server. Spring Statemachine is a framework for application developers to use state machine concepts with the Spring Boot application. Spring Statemachine framework runs the

predefined Finite State Machine (FSM) from the UML – XML file. FSMs are defined graphically inside the Eclipse Papyrus [13] modelling environment.

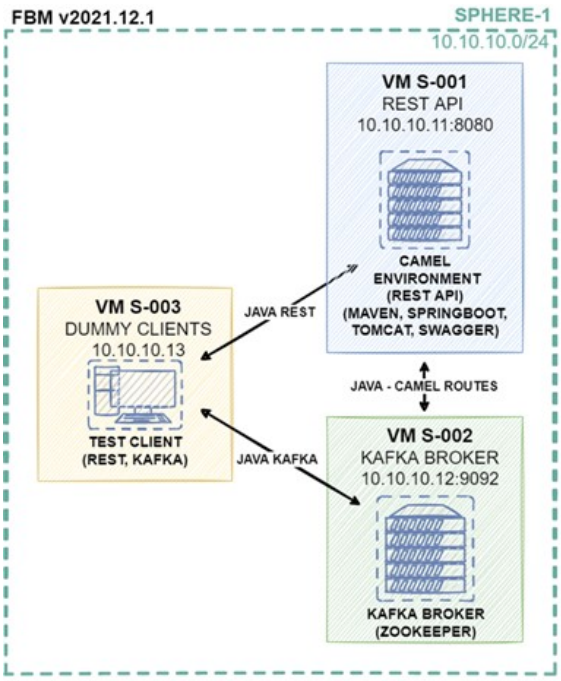


Figure 16: First version of the FBM architecture.

VM S-001 is hosting the Spring Boot application that runs defined FSMs based on KAFKA topics message or REST HTTP request message. This is outlined in Figure 17. Each FSM is built separately as a Java JAR file. Then it can be triggered from the aforementioned Spring Boot application on an appropriate JSON-based request. Each FSM goes through its states and end once every state is completed.

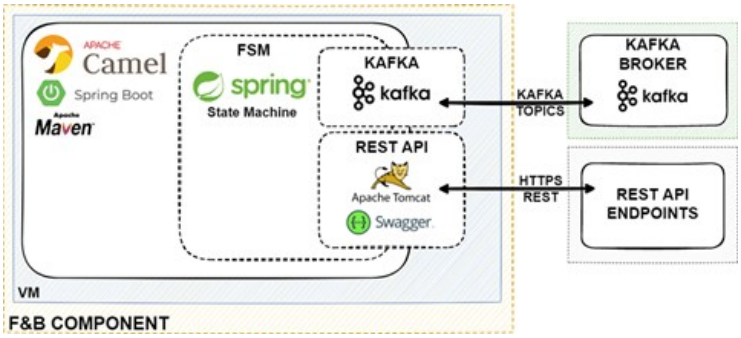


Figure 17: Structure of the Spring Boot app which runs FSMs.

VM S-002 is running Kafka Broker with its Zookeeper listening for incoming messages and serving outgoing messages on specific topics. Apache Kafka is a community distributed event streaming platform capable of handling large number of events a day.

VM S-003 hosts dummy Kafka and REST clients. The same version of Kafka is installed on that VM providing the ability to open Kafka Producer and Kafka Consumer as services for communication with the Kafka Broker for testing purposes. The VM also provide the Postman [14] platform. Postman is an

API platform that allows to create and use APIs. Postman streamlines collaboration and simplifies each phase of the API lifecycle, allowing to design better APIs quicker. We combine the use of Postman and Swagger UI when building and testing REST endpoints.

The Swagger Specification is a REST API description format. API specifications are available in YAML and JSON formats. Figure 18 outlines the FBM Swagger UI for PALANTIR project.

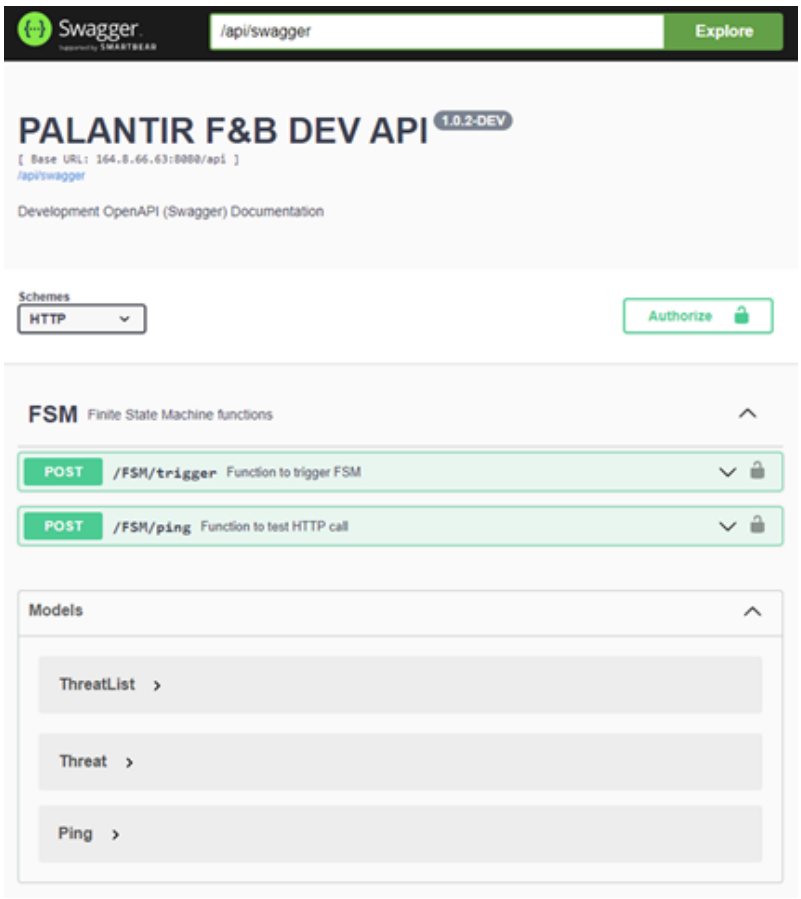


Figure 18: FBM Swagger UI used in the implementation.

### 4.2.1. PALANTIR interface to design FSMs

The environment used for modelling and the creation of the FSMs is Eclipse Papyrus. Eclipse Papyrus provides a simple "drag & drop" workflow which exports the final FSM model to the UML (XML) document containing the FSM configuration. This document is then used to build the Spring State Machine inside the Spring Boot Java application.

Eclipse Papyrus, shown in Figure 19, is a UML tool with many features enabling complex customisations. The Papyrus workspace contains different nodes which are used to build the UML state machine. Each model should start with an "initial state" and end with a "final state". What is in the middle of those two states is up to the modeller of the state machine. Each transition between the nodes can be connected to (call) program code written in different programming languages. This option is used in the FSMs prepared for PALANTIR for communication with Kafka and REST.

Eclipse Papyrus' modelling elements are all meant to be customized and reused as much as possible. As a result, if we wish to customize the modelling environment to meet our needs, we may alter the basic setup for a certain domain, notation, or modelling practice using Eclipse Papyrus' strong modification features. Because many configurations in Eclipse Papyrus are model-based, live modification is possible. These are some of the possibilities that Eclipse Papyrus offer:

- Create own notation, whether graphical, textual, or tabular.
- Filter current palettes or create own with a model-based setup.
- Create dedicated property views to display only the features that are essential.
- Read models with specialized model explorer structure and rendering.
- Reuse common languages or create own modelling languages using the UML profile editor.

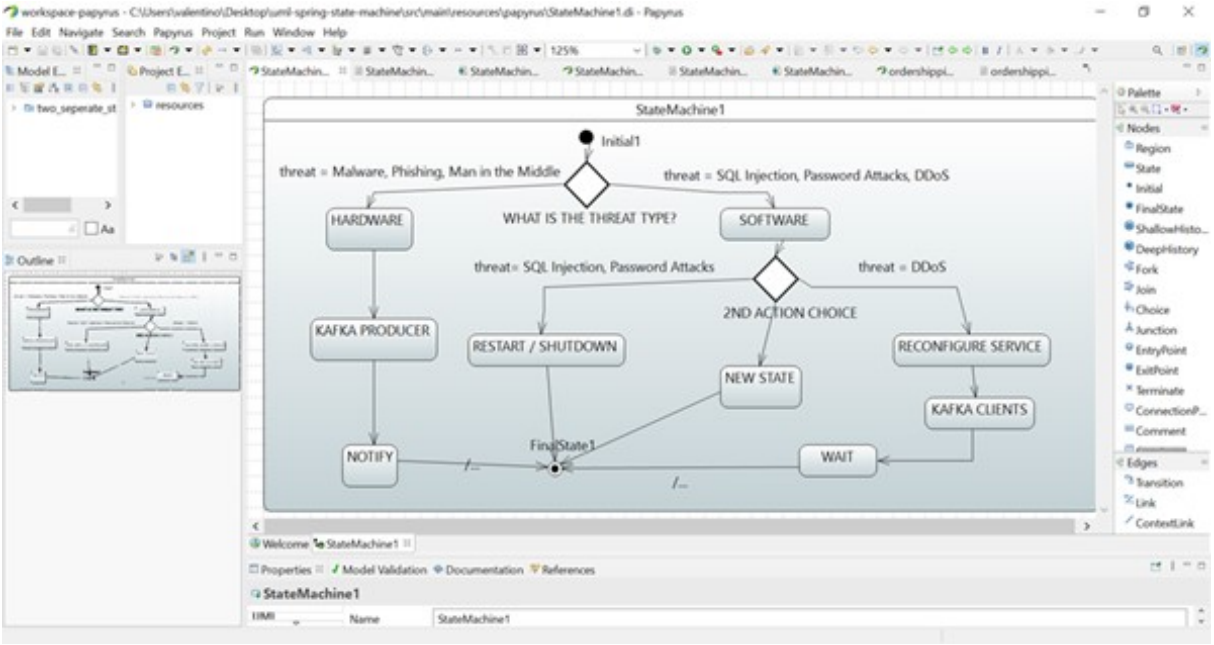


Figure 19: Designing the FSM for December PALANTIR GA inside the Papyrus.

## 4.3.Recovery Service

As was defined in D2.1, the Recovery Service assures the SCHI's resiliency by organizing the recovery activities needed when a platform or capability is deemed untrustworthy or when a defect or breach is discovered. The recovery options are many and varied in nature: a platform may be rebooted or isolated by rerouting network traffic around it, a Security Capability can be re-configured or implemented with a new solution, and so on. Such a wide range of recovery activities necessitates a flexible method of picking the best recovery approach for each case, such as changeable playbooks.

The RS implementation also uses the FSMs similarly to the Data Breach implementation. The structure of the RS component consists of the same building blocks: Spring Boot application, Spring Statemachine, Kafka and Camel with Swagger UI. Figure 20 depicts the diagram and flow of the RS component.

The following example was presented on the PALANTIR December 2021 GA meeting. A Kafka producer (e.g. the AE) is sending to the topic “palantir_demo_1” while on the Spring Boot application Kafka consumer is subscribed to the same topic and retrieves the incoming JSON based message. Here JSON contained the “ddos” threat name. Based on received threat name FSM RS1 started its execution. Because threat was categorized as a “ddos” FSM chooses its right branch and first state “SOFTWARE” and state “SERVICE RECONFIGURE”. From there, the next state was called as a Java Bean. This Java Bean is a Kafka client (i.e. a Kafka producer and consumer in the same time) listening on different topics. First, the Kafka producer sends a message to topic “palantir_demo_2” and another external Kafka consumer waits for a message on that topic. Once the message is received, it is up to operator to conclude the FSM by sending a confirmation message. However, this confirmation message is sent to a different

topic “palantir_demo_3”, which the Kafka consumer inside the FSM is waiting to reach the final “END” state.

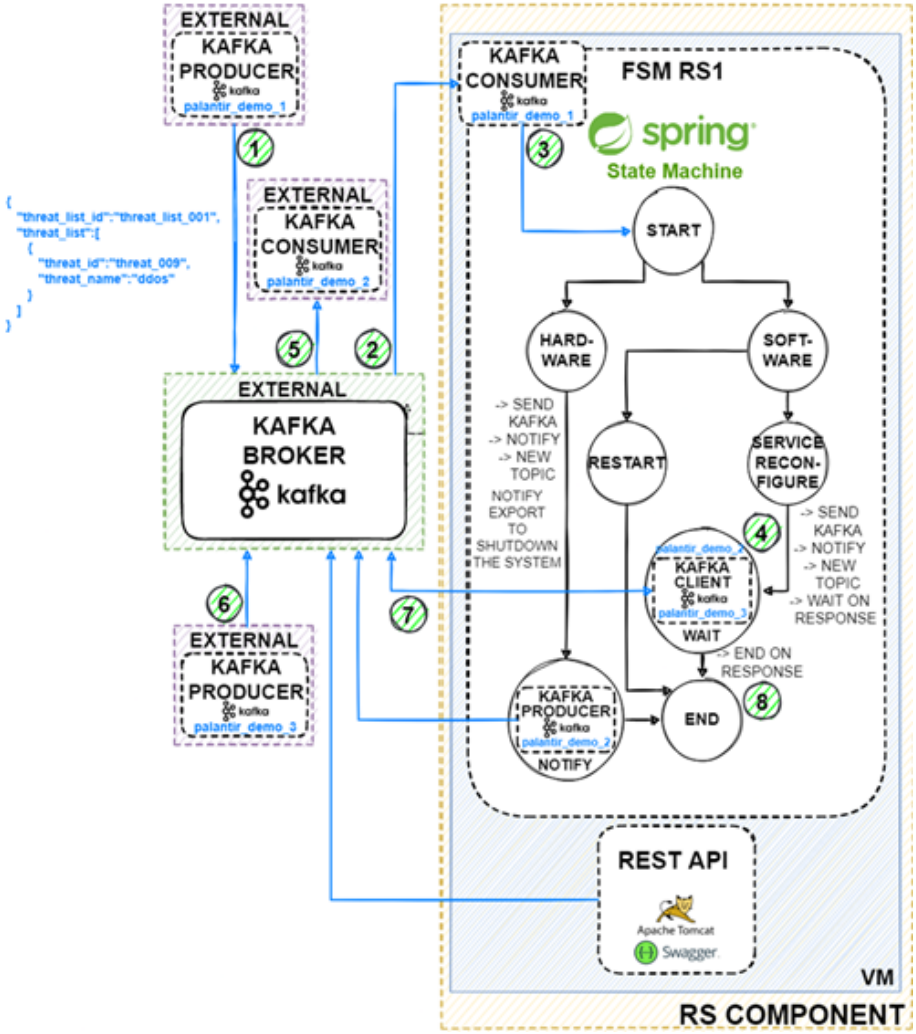


Figure 20: RS Component Structure proposition with FSM model.

# 5. Conclusions

This deliverable provides the initial details on the architecture, design, and specification of components of the Trust, Attestation and Recovery framework.

The document provides the current, up-to-date architecture of the TAR components, refined from the requirements presented in D2.1, the specification of the interactions among the other components of PALANTIR and the implementation and integration.

This document is followed by a second iteration, which is "D4.4 Trust, Attestation and Verification Framework – Specifications of second release", later in the project with finalized technical details on specification and implementation of the components comprising the TAR.

# 6. References


[1] Trusted Platform Module Working Group, TCG: TCG Trusted Platform Module Library Specification, Family "2.0"

[2] HPE Distributed Intrusion Monitoring Engine (DIME): https://community.hpe.com/t5/Advancing-Life-Work/Quis-custodiet-ipsos-custodes-HPE-next-generation-intrusion/ba-p/7042089#.YbM9nr3P1PY

[3] Apache Kafka: https://kafka.apache.org/

[4] HPE Project Aurora: https://www.hpe.com/us/en/security/project-aurora.html

[5] Keylime project: https://www.keylime.dev

[6] TCG Platform Certificate Profile: https://trustedcomputinggroup.org/resource/tcg-platform-certificate-profile/

[7] TCG TPM 2.0 Keys for Device Identity and Attestation: https://trustedcomputinggroup.org/resource/tpm-2-0-keys-for-device-identity-and-attestation/

[8] IEEE 802.1AR: Secure Device Identity: https://1.ieee802.org/security/802-1ar/

[9] Proxmox Virtual Environment: https://www.proxmox.com/en/proxmox-ve

[10] Apache Camel: https://camel.apache.org/

[11] Swagger UI: https://swagger.io/tools/swagger-ui/

[12] Spring Statemachine: https://projects.spring.io/spring-statemachine/

[13] Eclipse Papyrus: https://www.eclipse.org/papyrus/

[14] Postman: https://www.postman.com/

[15] ARM TrustZone: https://developer.arm.com/ip-products/security-ip/trustzone

[16] Intel SGX: https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/attestation-services.html

[17] AMD Secure Encrypted Virtualization (SEV): https://developer.amd.com/sev/

[18] Docker: https://www.docker.com/

[19] Docker Compose: https://docs.docker.com/compose/

[20] Django: https://www.djangoproject.com/

[21] N. Schear, P. T. Cable, T. M. Moyer, B. Richard, and R. Rudd, "Bootstrapping and maintaining trust in the cloud", Proceedings of the 32nd Annual Conference on Computer Security Applications, New York, NY, USA, December 5 - 8, 2016, pp. 65–77, DOI 10.1145/2991079.2991104