D5.2

First Version of SIFIS-Home Security Architecture Implementation

WP5 – Integration, Testing and Demonstration

SIFIS-Home

Secure Interoperable Full-Stack Internet of Things for Smart Home

Due date of deliverable: 30/09/2022
Actual submission date: 30/09/2022

Responsible partner: SEN
Editor: SEN
E-mail address: hakan.lundstrom@sensative.com

30/09/2022
Version 1.0

Project co-funded by the European Commission within the Horizon 2020 Framework Programme		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

The SIFIS-Home Project is supported by funding under the Horizon 2020 Framework Program of the European Commission SU-ICT-02-2020 GA 952652

Authors: Håkan Lundström (SEN), Marco Tiloca (RISE), Luca Barbato (LUM), Otto Waltari (FSEC), Andrea Saracino (CNR), Domenico Deguglielmo (DOMO)

Reviewed by: CNR, Joni Jämsä (Centria), Tom Tuunainen (Centria), Matthias Schunter (INT), Valerio Frascolla (INT)

Revision History

Version	Date	Name	Partner	Section Affected Comments
0.1	13/02/2022	Initial version	SEN	All
0.2	12/04/2022	Restructured version	SEN	All
0.3	27/04/2022	Group OSCORE, ACE, Edhoc	RISE	Implementation
0.4	04/05/2022	Several new chapters added. UX; FIWARE, etc.	SEN	All
0.5	12/05/2022	Several chapters completed	SEN, F-Sec	Validation, Intro, Conclusion, Executive summary
0.9	24/05/22	Ready for review	SEN, CNR, LUM	All
1.0	31/05/2022	Reviewer's comments addressed	SEN	All

Executive Summary

In this deliverable the status and implementation aspects of the first version of the SIFIS-Home security architecture is described based on the latest revised architecture defined in D1.4. It is a very extensive implementation of state-of-the-art technologies done by many different partners across Europe with different type of competences and know-how. Each and every defined SIFIS-Home framework component is walked through in detail and the implementation aspects, as well as status including a GitHub link to the released open-source code, are discussed.

The refined architecture is promising and solves the problems that were discovered during the implementation so far. It is though inevitable that further problems will arise during the integration and final development of all the components and when that happen, we will evaluate the situation and decide on suitable way forward.

Since the implementation and integration of the system, according to plans, will be completed in the next months, the system test activities have not started. Instead, the focus has been on unit testing of the different components one by one to secure they work as planned, so once the integration of all components is complete focus can be put on the SIFIS-Home use cases.

Table of contents

1 Introduction

This deliverable reports on the implementation of the first version of the SIFIS-Home security architecture based on the architecture defined in deliverable D1.4. It starts by a short re-cap of the test bed described in D5.1, then describes how the architecture has evolved from D1.3 until today, how it has affected the implementation, and the main problems that have been solved with the revised architecture.

A key topic during the implementation of the security architecture described in this deliverable is the allocation of the different components to the different partners based on their competence and committed contributions to the project. The report also describes the detailed implementation aspects and status of every component in the security architecture including providing a GitHub link per component to all code that has been released as open source. Special focus is put on the integration of the security solutions developed in WP3 and the analysis methods developed in WP4.

The deliverable then elaborates on the general integration strategy by using Docker containers and the current status of the verification of the SIFIS-Home framework. Since the implementations of framework is still ongoing, and it is not in a system verification testable state, the verification result is focused on the unit testing of some selected components that have reached a sufficiently stable maturity level.

This deliverable finally summarizes our view of the implementation status and the way forward.

2 Test bed used to verify implementation

The implementation of the components is first verified by respective partners and will after integration be verified on the SIFIS-Home official test bed built around the “Panarea” server in CNR’s facilities. The partners will upload their components via Secure shell (SSH) and configure them. In the Panarea server the SIFIS-Home Cloud Framework will execute as well as other SIFIS-Home technologies like simulated SIFIS-Home smart devices, analytics and network security solutions. The test is described in Figure 1.

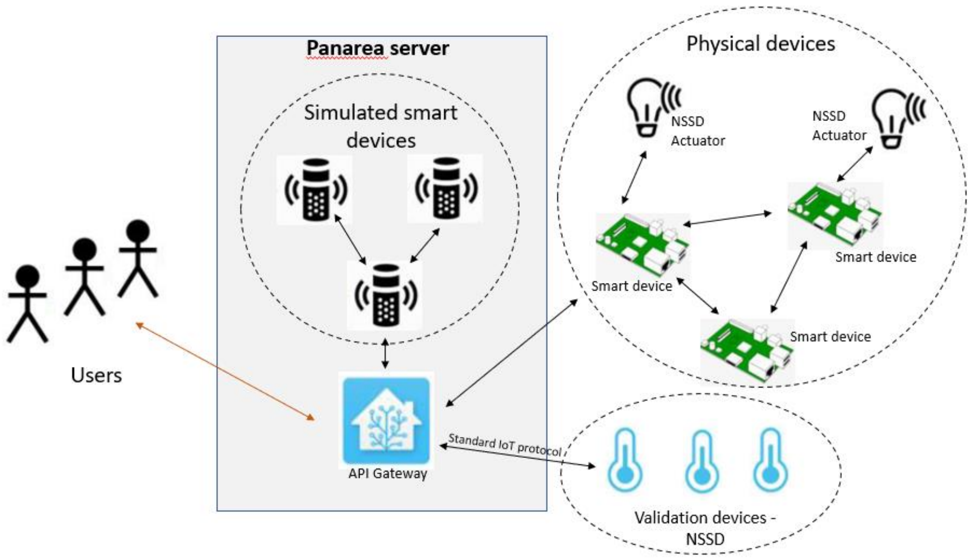


Figure 1: The SIFIS-Home test bed setup

Physical devices, both Not so smart devices (NSSD) and SIFIS-Home Smart devices built upon Raspberry PI, will then be connected to the SIFIS-Home network and all components will be verified.

3 Implementing the security architecture

3.1 *Architecture*

3.1.1 Architecture iterations

The design of the SIFIS-Home framework in WP1 is based on Docker containers with *microservices* design pattern. Each microservice defines a Rest API to interface with it. The high-level architecture of the SIFIS-Home framework was originally defined in D1.3 (see Figure 2 below) and was the basis for the start of the implementation of the security architecture and the test bed design.

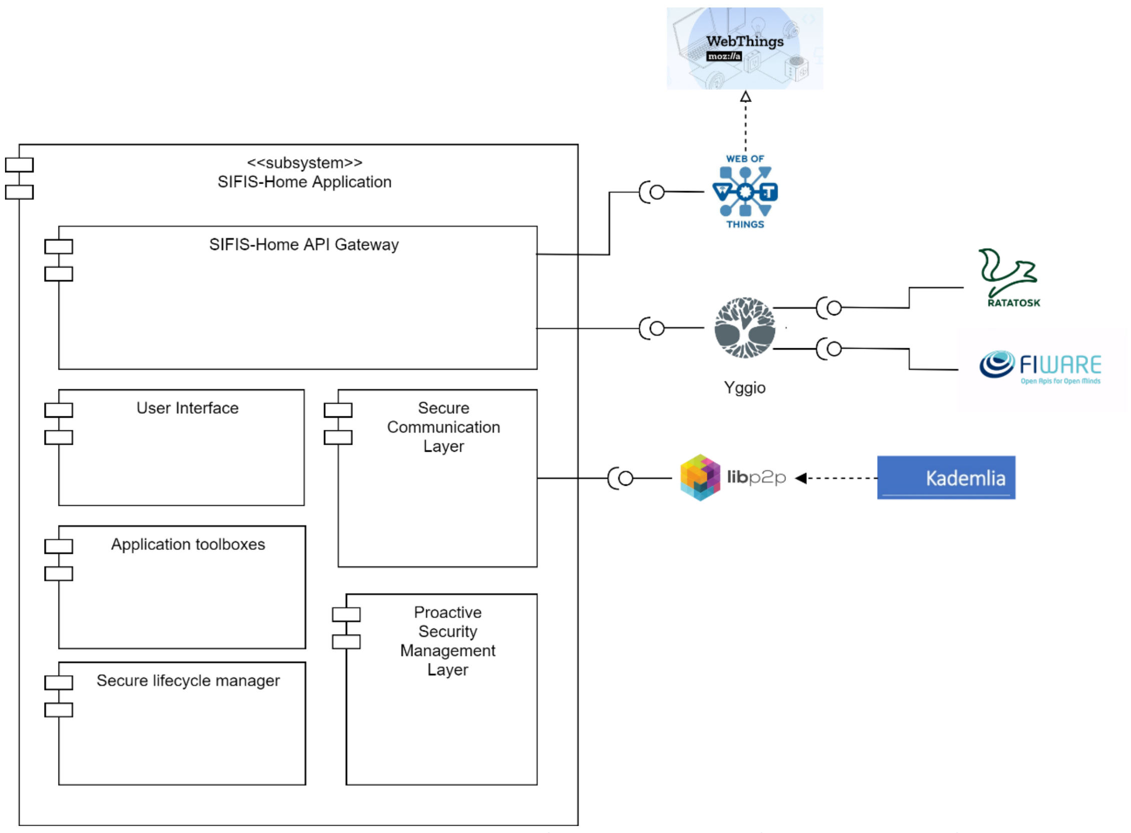


Figure 2 Original high-level architecture of the SIFIS-Home framework as defined in D1.3

As described in D5.1 “First version of the SIFIS-Home Test bed” there were some issues in the architecture related to the API gateway and the Web-of-Things (WoT) that could not be implemented and a revised architecture with some extra components was proposed to WP1 to study. See Figure 3 below.

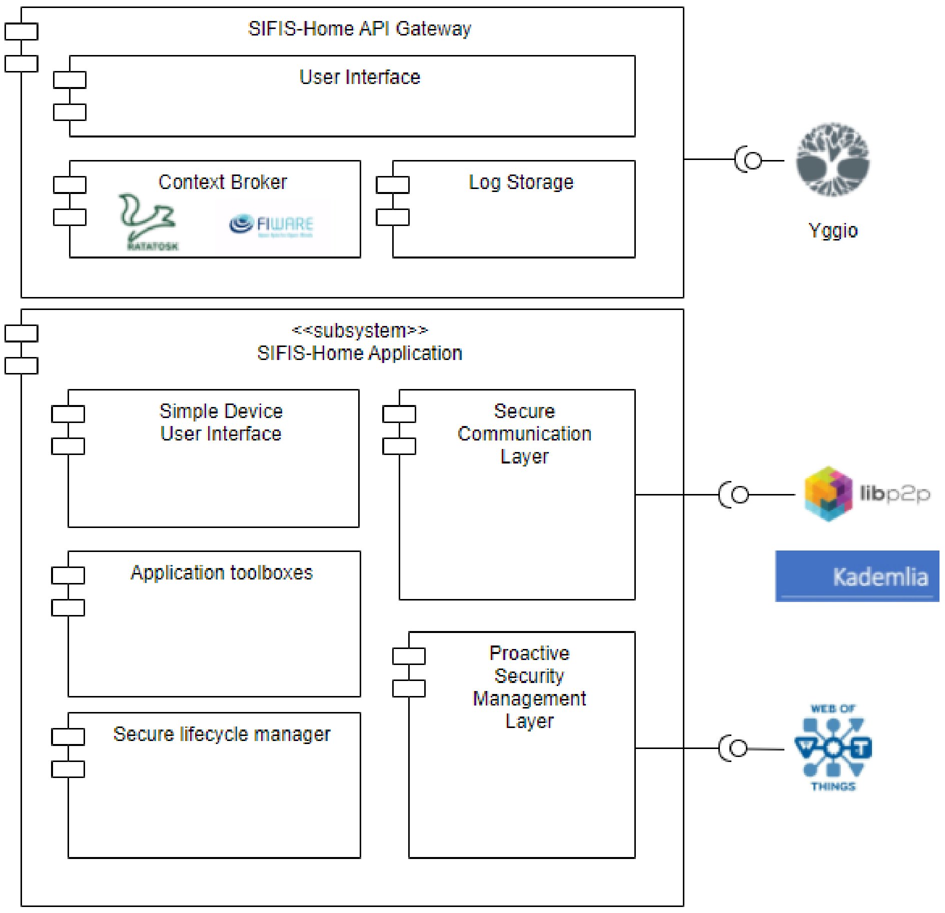


Figure 3: Revised SIFIS-Home architecture with some components moved to the API Gateway

WP1 studied the problems highlighted in D5.1 and revised the security architecture to become modularised in a few different frameworks. The revised architecture was described in detail in deliverable D1.4, where each framework fulfils a specific need:

- **SIFIS-Home Smart Device Framework:** the set of software components that are executed on the Smart Devices (SD).
- **SIFIS-Home Application Framework:** the set of software components that are installed on a mobile device (smartphone).
- **SIFIS-Home NSSD Framework:** the set of software components that are executed on the NSSD (SD).
- **SIFIS-Home Cloud Framework:** the set of software components and applications that reside on the SIFIS-Home cloud that are mainly used to allow a user to control the smart home from a remote side.
- **SIFIS-Home Development Tools:** the set of developer tools that have been developed in the context of WP2. The development tools are out of scope for this document.

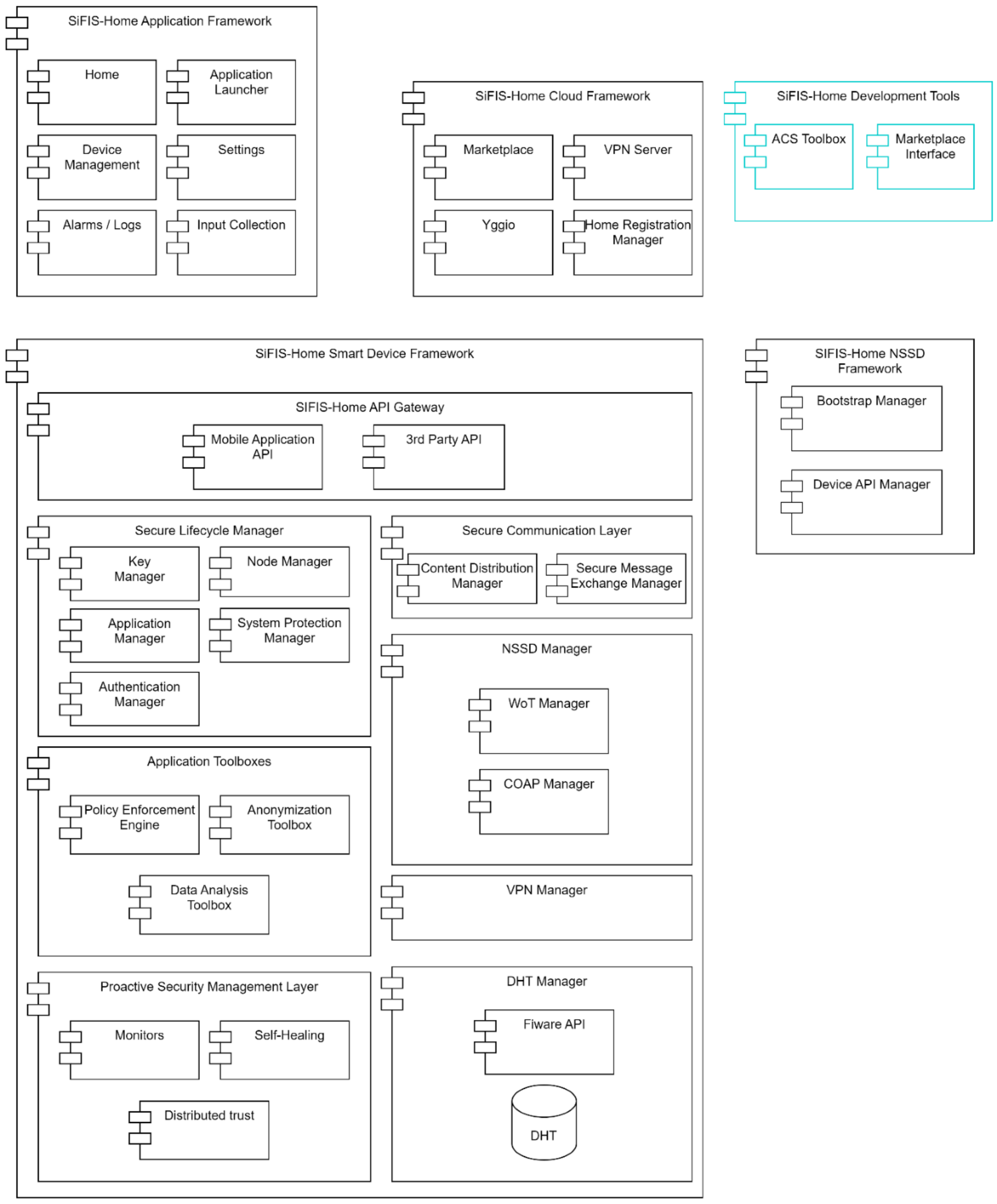


Figure 4: Latest version of the SIFIS-Home architecture as defined in D1.4

The latest version of the security architecture in Figure 4 is currently being implemented and is the main focus of this document.

3.1.2 Allocation of components

The SIFIS-Home architecture is the representation of the devices and actors interacting with the SIFIS-

Home framework as originally defined in D1.3 and then refined in D1.4. To design and later implement the test bed we started by finding out all partners capabilities, experience, commercial interest and availability of legacy code, tools and devices. The combinations of the partners know-how and the architecture and the requirements of the SIFIS-Home framework then led us to identify the optimal allocation of all components to different partners to implement the security architecture.

The optimal partner allocation to implement a test bed of the D1.4 architecture is defined in the Table 1 below except for the User Interface (UI) part that resides on the application framework on mobile phones that is still under technical design phase and marked with To Be Defined (TBD) in the table.

Component / Subcomponent	**Partner**	**Notes**
User Interface	SEN / TBD	Cloud interface / Application Framework
- Home	SEN / TBD	Cloud interface / Application Framework
- Device Management	SEN / TBD	Cloud interface / Application Framework
- Settings	SEN / TBD	Cloud interface / Application Framework
- Alarm / Logs	SEN / TBD	Cloud interface / Application Framework
- Marketplace	SEN	Cloud interface
- Fiware API	SEN	Ratatosk Context Broker on cloud
- Input Collection	DOMO	Component that allows to capture audio and images from the microphones and cameras of the user smartphone.
- Policy Manager	POL	Application Framework
Secure Lifecycle Manager	CNR	
- Application Manager	CNR	Code inlining routines
- Node Manager	CNR	Set of functionalities used by DHT- Distributed hash table
- Authentication Manager	RISE	Set of functionalities and protocols
- Key Manager	RISE	Set of functionalities and protocols
- System Protection Manager	CNR	Set of functionalities
Secure Communication Layer	RISE	
-Secure Message Exchange Manager	RISE	Set of functionalities and protocols
- Content Distribution Manager	RISE	Set of functionalities and protocols
NSSD Manager	DOMO	
- CoAP Manager	RISE	Pub-sub-based command interface
- WoT Manager	LUM/DOMO/RIOTS	It is an application that receives commands from the DHT and forwards them to WoT enabled devices. It also updates the state of WoT enable devices on the DHT.
Proactive Security Management Layer	CNR	
- Monitors	CNR	Set of functionalities
- Distributed Trust	CNR	Set of functionalities
- Self-Healing	CNR	Set of functionalities
Application Toolboxes	CNR	
- Data Analysis Toolbox	CNR	Set of analysis functionalities and tools

		for data format management
--Pre-Processing Layer/Post-Processing	DOMO	Software component that takes care to change the format of data coming from the devices such that it can be easily used by the analytics applications.
-- Behavioral Analysis	CNR	Tools for data analysis
-- Network Analysis	F-Secure	Real time network monitor and analyzer
-- Multimedia Analysis	LUM	Tools for data analysis
-- Application Analysis	CNR	Tools for data analysis
-- Log storage	SEN	Monitoring and storage
-- Physical Analysis	CNR	Monitoring and storage
-- Data Analysis Engine	CNR	Container of analysis functionalities
- Anonymization Toolbox	CNR	Anonymization functionality
- Policy Enforcement Engine	CNR	Enforcement functionalities
SIFIS-Home API Gateway		
Mobile Application API	Centria / TBD	Joining of device, view status, send commands.
3rd party API	LUM / POLITO	Extension of WoT API
Integration components		
SD Web Interface to create users and join an existing house	DOMO	Set of APIs exposed by a Web Service that allow i) adding a new SD to a SIFIS-Home system, ii) adding new users
NSSD interface for joining	DOMO	Set of APIs provided by NSSDs through which it is possible to send them joining information
DHT-MQTT integration component	CNR, DOMO	Software component that allows exchanging messages between the SIFIS-Home DHT and the Yggio MQTT broker.
DHT-KeyCloack integration component	CNR, DOMO	Software component that allows user-related data synchronization between the DHT and Keycloack
WoT Discovery component	CNR, LUM	Software component that allows discovering WoT-enabled devices
Application Manager, Marketplace	RIOTS, CEN,	The Application Manager allows installing third-party SIFIS-Home application on a SD. The Marketplace component is a cloud service providing the list of available third-party applications that can be installed on a SD.
Policy Creation Component	POLITO	Integration

Table 1 Component allocation

3.2 *User Interface - The application and Cloud framework*

This component provides the Graphical User Interfaces to all different typologies of users that will utilize the SIFIS-Home Framework.

The Application Framework that contains the UI that executes on the SIFIS-Home smart devices residing inside the home will contain, for the device, the relevant subset of the web interface. This UI is still under design phase.

The web part of the user interface resides on the Cloud Interface since it needs to be accessible both inside the SIFIS-Home network and externally from internet when the home users are outside of the home. The web UI on the Cloud Interface runs on top of the RATATOSK Context broker, but FIWARE does not implement all the needed APIs required by the cloud UI, so Sensative Yggio's legacy proprietary API's and security APIs are used as well to simplify the implementation of the SIFIS-Home Cloud Interface.

3.2.1 Cloud Framework:

This component, Figure 5, and includes a set of high-level APIs and is used to access SIFIS-Home networks and their UI both internally from inside the network and externally. The web applications are a major development effort and are being developed based on Node.js JavaScript and utilize REACT components with Next.JS framework. For more details one can refer to D5.1.

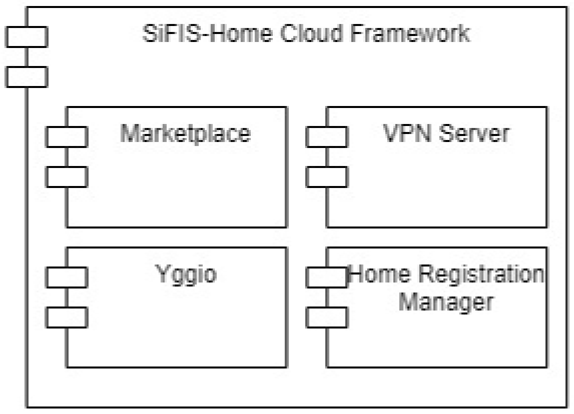


Figure 5: The Cloud Framework

- **FIWARE Context Broker Ratatosk**
  The core part of the API gateway is the FIWARE Context Broker Ratatosk, Figure 6, on top of which its UI is built upon. Ratatosk Context broker contains the full description of every SIFIS-Home network that is connected to the gateway.

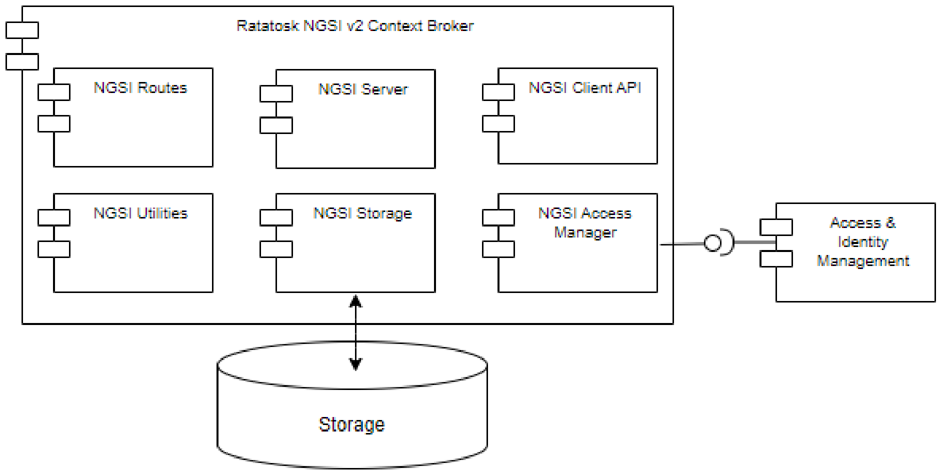


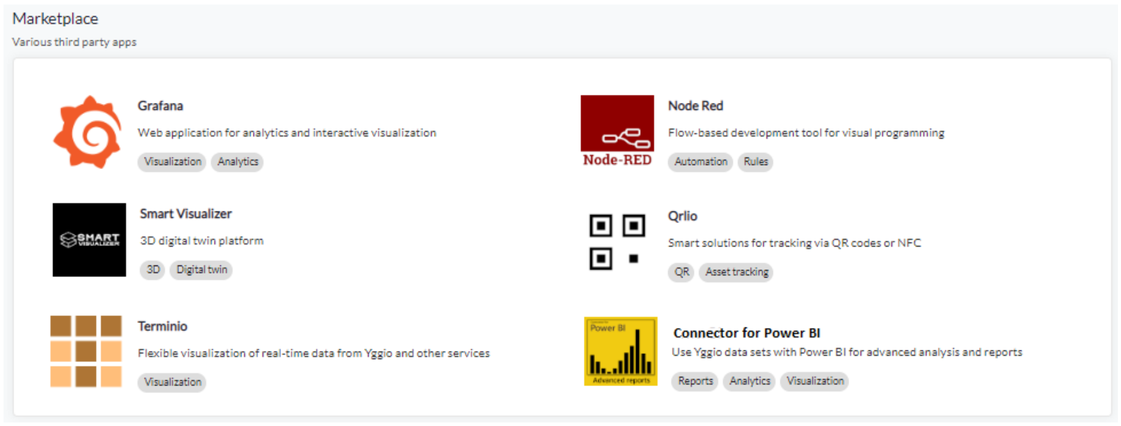
Figure 6: Ratatosk FIWARE Context Broker which is the core of the cloud UI

The Ratatosk implementation is available here: https://github.com/sifis-home/yggio-ratatosk

- **Cloud UI powered by SIFIS Yggio**:
  The Sensative horizontal Internet of Things (IoT) integration platform is used as the backbone of the SIFIS-Home cloud interface. It provides the execution environment that makes Ratatosk possible to execute, and its API makes it possible to implement the SIFIS-Home overall web interface UI. The web UI is feature rich with focus on device and user management and was illustrated in D5.1.

  The Cloud UI implementation is available here: https://github.com/sifis-home/yggio-components

- **Market Place:**
  The Market Place, Figure 7, resides on cloud UI and will allow the end user can download applications and install them in their SIFIS-Home network or directly allow web applications to securely, via OAuth or basic credentials, access the SIFIS-Home network via the cloud interface. Currently the Market Place via its API's fully support OAuth 2.0 security integration to give 3rd party application access to the SIFIS-Home network.


Marketplace
Various third party apps

Grafana
Web application for analytics and interactive visualization
Visualization Analytics

Node Red
Flow-based development tool for visual programming
Automation Rules

Smart Visualizer
3D digital twin platform
3D Digital twin

Qrlio
Smart solutions for tracking via QR codes or NFC
QR Asset tracking

Terminio
Flexible visualization of real-time data from Yggio and other services
Visualization

Connector for Power BI
Use Yggio data sets with Power BI for advanced analysis and reports
Reports Analytics Visualization

Figure 7: The marketplace with some initial applications

The Market Place implementation is available here: https://github.com/sifis-home/yggio-components/tree/master/control-panel-v2/src/pages/apps

- **VPN Manager:**
  This component is responsible for managing the set of VPN servers that allow access to SIFIS-Home enabled houses from a remote side. It provides an HTTP API through which it is possible to create a dedicated VPN instance associated to every SIFIS-Home house. A dedicated DNS record for every VPN instance is created. Our intention is to use Wireguard as VPN server.

- **Home Registration Manager:**

  it provides an interface through which it is possible to create a new SIFIS-Home enabled house. When a new house is created, a VPN server and an Yggio instance dedicated to the house are created.

- **Alarm / Logs:**
  a component including features to show alarms to the user, and to gather logs of the functioning of the SIFIS-Home infrastructure. The log storage component was decided to be based on the well proven standard UNIX SYSLOG format (https://datatracker.ietf.org/doc/html/rfc5424).

  This component is still to be developed and final design and architecture has not been locked down. No code base available yet.

3.2.2 Application Framework

This module handles the standard workflow of the SIFIS-Home framework. It consists of the modules Application Manager, Node Manager, Device Registration Manager and System Protection Manager. The actual UI in the Application Framework is being implemented as a web view application so it can utilize the same applications that are being developed for the cloud interface.

- **Home:** the principal component of the UI, containing commands that lead the resident user of the SIFIS-Home system to the lead features of the infrastructure. From here the user can launch device management, user management and the application launcher**.**
- **Application launcher:** it provides graphical UI from which the user can visualize available applications to be installed on to the SIFIS-Home system, connect, download, install, update them if new versions are available, and launch them.
- **Device management:** this component enables user to do configuration of the available devices in the SIFIS-Home network, each device status can be visualized and if the device supports it also configuration downlinks can get sent. Metadata can be added to each device.
- **Settings:** it provides UI for the configuration of the SIFIS-Home infrastructure. Different interfaces are provided to different actors of the SIFIS-Home system. This is embedded into the device manager.
- **Input collection:** it is in charge of providing the facility of collecting the inputs from the user, in all the forms that are allowed by the system.

3.3 *Smart Device Framework*

3.3.1 Secure Lifecycle Manager

This module, illustrated in Figure 8, handles the standard workflow of the SIFIS-Home framework. It consists of the modules Application Manager, Node Manager, Device Registration Manager and System Protection Manager.

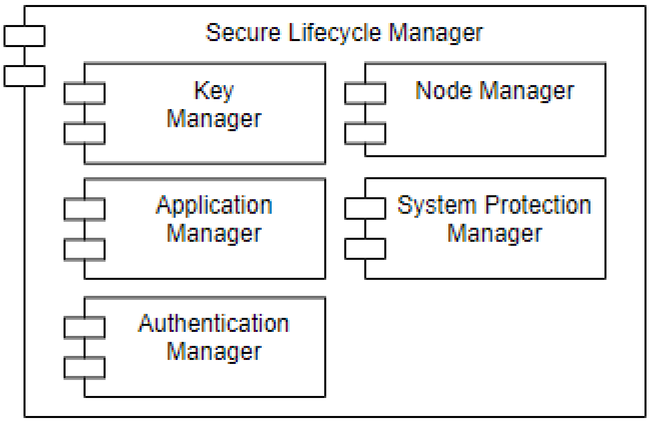


Figure 8: Secure Lifecycle Manager

- **Application Manager**
  It handles installation and removal of SIFIS-Home 3rd party applications. The component is under development and is now available as an interface to enable control of executables in different system, such as operating on Docker Containers, Linux Executables, or Mobile Applications.
- **Node manager**
  It is the component responsible for keeping consistent the list of smart devices in a SIFIS-Home instance, with the participant to the DHT. This component is implemented in the Rust programming language and exploits the LibP2P library as all the other components related to the DHT. The component is fundamental to allow dynamic joining of nodes and also handles the removal of nodes in different circumstances.
- **System Protection manager**
  It is the component that receives inputs from the various monitors and triggers action by communicating with the Application Manager and Node Manager through the DHT.
- **Authentication Manager and Key Manager**
  The following security solutions developed in WP3 pertain to the "Secure Lifecycle Manager" module, and a link to their implementation from RISE is also provided. These implementations build on the open-source Eclipse Californium CoAP framework available at [CALIFORNIUM], which provides the CoAP protocol and the OSCORE security protocol. A single codebase collecting these implementations is accessible at [WP3-CODEBASE], as available for use, integration and testing within the SIFIS-Home project and especially in the interest of the WP5 testbed.

  [CALIFORNIUM] https://github.com/eclipse/californium
  [WP3-CODEBASE] https://github.com/sifis-home/wp3-solutions

  - **OSCORE profile of the ACE framework**, as documented in Section 5.1 of D3.2. This security solution pertains to the "Authentication Manager" and the "Key Manager" components of the "Secure Lifecycle Manager" module. Together with the main ACE framework, the implementation is available at https://bitbucket.org/marco-tiloca-sics/ace-java

- **Key provisioning for Group OSCORE communication using the ACE framework**, as documented in Section 6.1 of D3.2. This security solution pertains to the "Authentication Manager" and the "Key Manager" components of the "Secure Lifecycle Manager" module. Together with the main ACE framework, the implementation is available at https://bitbucket.org/marco-tiloca-sics/ace-java
- **EDHOC key establishment**, as documented in Section 6.3 of D3.2, including specific profiling for CoAP and OSCORE. This security solution pertains to the "Key Manager" component of the "Secure Lifecycle Manager" module. The implementation is available at https://github.com/rikard-sics/californium/tree/edhoc

3.3.2 Secure Communication Layer

This module, shown in Figure 9, handles the standard workflow of the SIFIS-Home framework. It consists of the modules Secure Message Exchange Manager, Content Distribution Manager and Network Protection Manager.

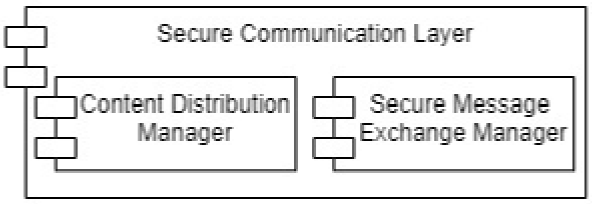

Secure Communication Layer
Content Distribution Manager
Secure Message Exchange Manager

Figure 9: Secure Communication Layer

- **Secure Message Exchange Manager and Content Distribution Manager**
  The following security solutions developed in WP3 pertain to the "Secure Communication Layer" module, and a link to their implementation from RISE is also provided. These implementations build on the open-source Eclipse Californium CoAP framework available at [CALIFORNIUM], which provides the CoAP protocol and the OSCORE security protocol. A single codebase collecting these implementations is accessible at [WP3-CODEBASE], as available for use, integration and testing within the SIFIS-Home project and especially in the interest of the WP5 testbed.

  [CALIFORNIUM] https://github.com/eclipse/californium
  [WP3-CODEBASE] https://github.com/sifis-home/wp3-solutions

- **Group OSCORE**, as documented in Section 4.1 of D3.2. This security solution pertains to the "Secure Message Exchange Manager" and the "Content Distribution Manager" components of the "Secure Communication Layer" module. The implementation is available at https://github.com/rikard-sics/californium/tree/group_oscore.
- **OSCORE profile of the ACE framework,** as documented in Section 5.1 of D3.2. This security solution pertains to the "Secure Message Exchange Manager" and the "Content Distribution Manager" components of the "Secure Communication Layer" module. Together with the main ACE framework, the implementation is available at https://bitbucket.org/marco-tiloca-sics/ace-java.

- **Key provisioning for Group OSCORE communication using the ACE framework**, as documented in Section 6.1 of D3.2. This security solution pertains to the "Secure Message Exchange Manager" and the "Content Distribution Manager" components of the "Secure Communication Layer" module. Together with the main ACE framework, the implementation is available at https://bitbucket.org/marco-tiloca-sics/ace-java.
- **EDHOC key establishment**, as documented in Section 6.3 of D3.2, including specific profiling for CoAP and OSCORE. This security solution pertains to the "Secure Message Exchange Manager" and the "Content Distribution Manager" components of the "Secure Communication Layer" module. The implementation is available at https://github.com/rikard-sics/californium/tree/edhoc.

- **Network Protection manager**
  The Network Protection Manager is a security solution responsible for taking pre-emptive and reactive actions against different security attacks. It takes actions based on commands from other microservices, such as DHT Monitor, Network Monitor, and Analytic Toolbox.

  For external threats to SIFIS-Home, the Network Protection Manager keeps an updated database of known malicious networks from open-source threat intelligence feeds and takes pre-emptive measures by blocking those networks.

  For reactive actions, the Network Protection Manager provides an HTTP REST API that has a function to start an emergency action to isolate the device from the network. For example, suppose a device is detected as part of a DoS attack or can't be trusted for different reasons. In those cases, the Network Protection Manager tries to disable the network from the problematic device.

  The Network Protection Manager is run on all Smart Devices, mainly taking action against the Smart Device where it is run when malicious activities are detected. When possible, the problematic device reports about itself through DHT and is isolated from other Smart Devices by their respective Network Protection Managers. This ensures that if the infected device can't communicate with other devices, in case the malware cancels the action made by Network Protection Manager on that device, the attack still gets isolated.

  The implementation of the Network Protection manager is available at https://github.com/sifis-home/wp5_network_protection_manager.

3.3.3 Proactive Security Management Layer

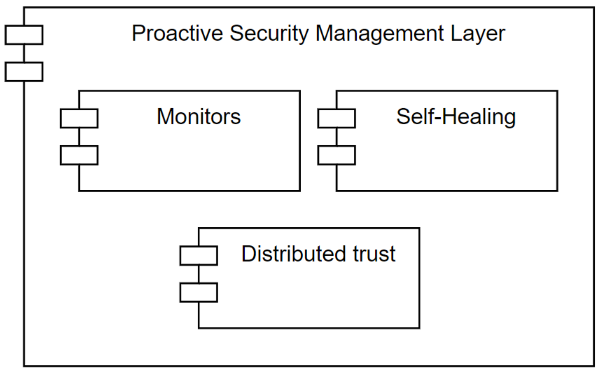


Figure 10: Proactive Security Management Layer

This component, shown in Figure 10, is responsible for maintaining the security aspects of the SIFIS-Home infrastructure in advance. Proactivity implies taking preemptive measures before an incident occurs. The following proactive security measures are being developed as part of SIFIS-Home project, and will be part of the security architecture:

- **Monitors**
  This component is made by a collection of modules used to log specific events at different levels. The levels are the following:
  - **DHT Monitor:** Implemented through the libP2P library, it logs the number and type of operations performed on the DHT.
  - **Application Monitor:** Under development, this component in-lines security critical APIs to log when they are invoked by 3rd party applications. The component is being implemented to support different platforms. We are currently working on three implementations. The first one is designed to be inserted in the SIFIS-Home API. Each API includes a first code line which is used to check authorizations and is able to terminate the execution of the function if authorization is not present. The second one targets Linux applications by inlining specific system calls, based on change of address in the system calls table. This monitor can be installed at runtime through the INSMOD command. Finally, a third implementation is available for Android systems by exploiting the XPosed Framework[1], which is used to inline any API call in the Android system.
  - **Network Monitor:** It acquires traffic through capturing packets via iptables. It is intended to run on central networking devices, such as a router through which devices communicate both inside the SIFIS-Home and to the wider Internet.
  - **SysCall Monitor:** It collects system call events through a REST API and conveys them further to the responsible analytic in the analytic toolbox for assessment.
- **Distributed Trust**
  The distributed trust component is a mechanism that assigns continuously to each smart device a trust score and manages distributed decisions under biased voting. The mechanism ensures that those devices with a trust score below threshold cannot participate in further voting procedures, until their trust level is not re-assessed. The component is implemented in Java 8 and it is based on standard libraries. A porting to Rust is also being performed as well as the integration in the SIFIS-Home platform.

- **Self Healing**
  The self-healing component of SIFIS-Home has two functionalities which are executed once a device has been identified as compromised and made unable to interact with the SIFIS-Home architecture. The first functionality triggers a reconfiguration of the SIFIS-Home architecture to avoid network partitions and reconnect NSSDs which remained isolated after the removal of their responsible device.
  This component is then also used to run a set of routines automatically on smart devices which have been identified as misbehaving. Depending on the identified misbehaviour a number of actions can be taken, e.g., an application deemed as malicious can be removed forcefully, then, after a memory analysis, the device can be re-integrated. As an alternative, the self-healing can

[1] https://xposed-installer.it.uptodown.com/android

trigger a full reset of the smart device. The component is currently under development as a RUST module partially based on the LibP2P library.

- **Manufacturer Usage Description** (MUD)
  MUD [RFC8520] is a proposed standard to define the network behavior of an IoT device. The usage description itself is an access control list defined by the vendor. It explicitly defines protocols, ports, and endpoints with which a particular IoT device is allowed to communicate with. However, as of today, vendors that provide MUD support are very few. Through leveraging the usage description models created by one of the SIFIS-Home analytics, AUD - Aggregated Usage Description manager (described in Section 4.2), we can create access control lists for any local IoT device and mimic the proactive security mechanisms specified in the MUD standard. Through enforcing such an access control measure for a home IoT device, one would block any incoming harmful network intrusion attempts. The code related to MUD adaptation and AUD Manager is available at https://github.com/sifis-home/wp4-aud_manager.

- **Evaluator / Notifier**
  The evaluator component is a software entity responsible for evaluating the relevance of detected anomalies. Since all network anomalies are not inherently harmful, the evaluator decides based on given parameters the severity of an anomaly, and whether it should be reported to end users. Evaluator components are highly specific to the type of anomaly they are expected to evaluate. The evaluator for network anomaly detection is implemented in Python. The notifier component implements the link through which anomaly reports are conveyed to end users.

3.3.4 Application toolboxes:

This component, shown in Figure 11, collects related and interconnected sub-components that are all services inside the SIFIS-Home infrastructure.

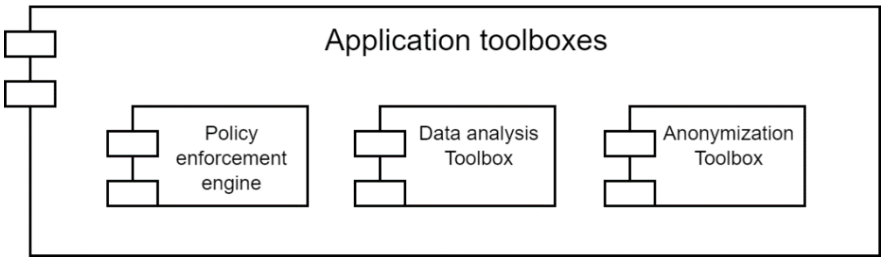
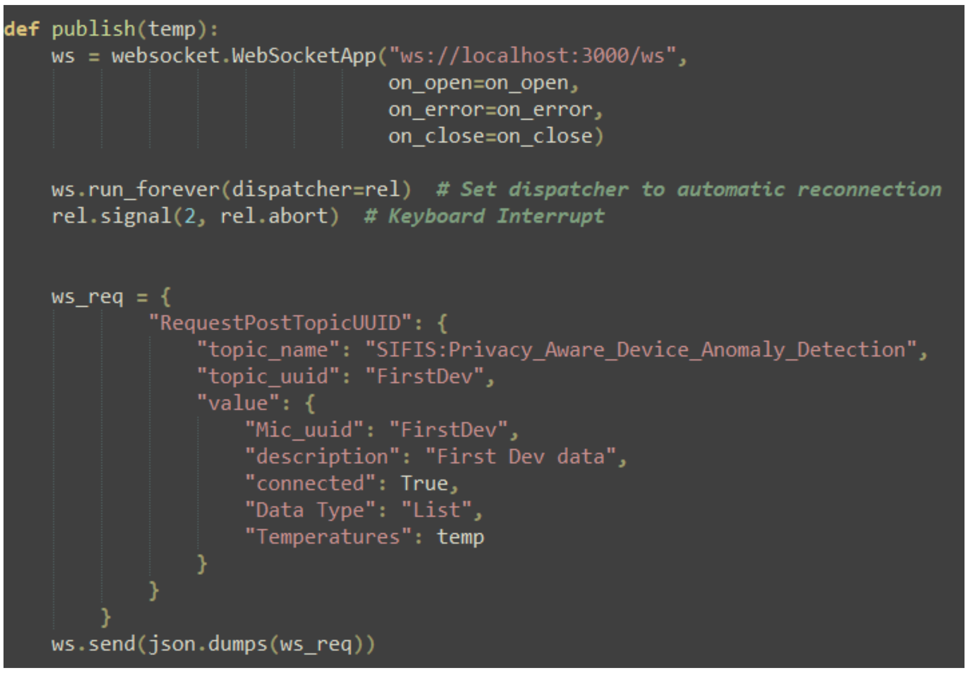


Figure 11: The application toolboxes

- **Data Analysis Toolbox**
  This component of the SIFIS-Home framework is devoted to the execution of the analytics on the data collected from the sensors and smart devices in the smart home, aimed at analyzing voice and gesture commands, providing advanced smart services to the smart home users, detecting misbehavior, intrusions, failures, and so on. This component is activated by the other components of the framework, which request the execution of analytic functions on given sets of data. A request could ask for a single execution of one analytic function on a given set of data, or to repeat the execution of an analytic function several times on distinct sets of data. The interactions between the Data Analysis Toolbox and the other components of the SIFIS-Home Framework occur through the DHT. In particular, the Data Analysis Toolbox creates a topic for each of the analytics it provides and subscribes to the smart home devices to all these topics.

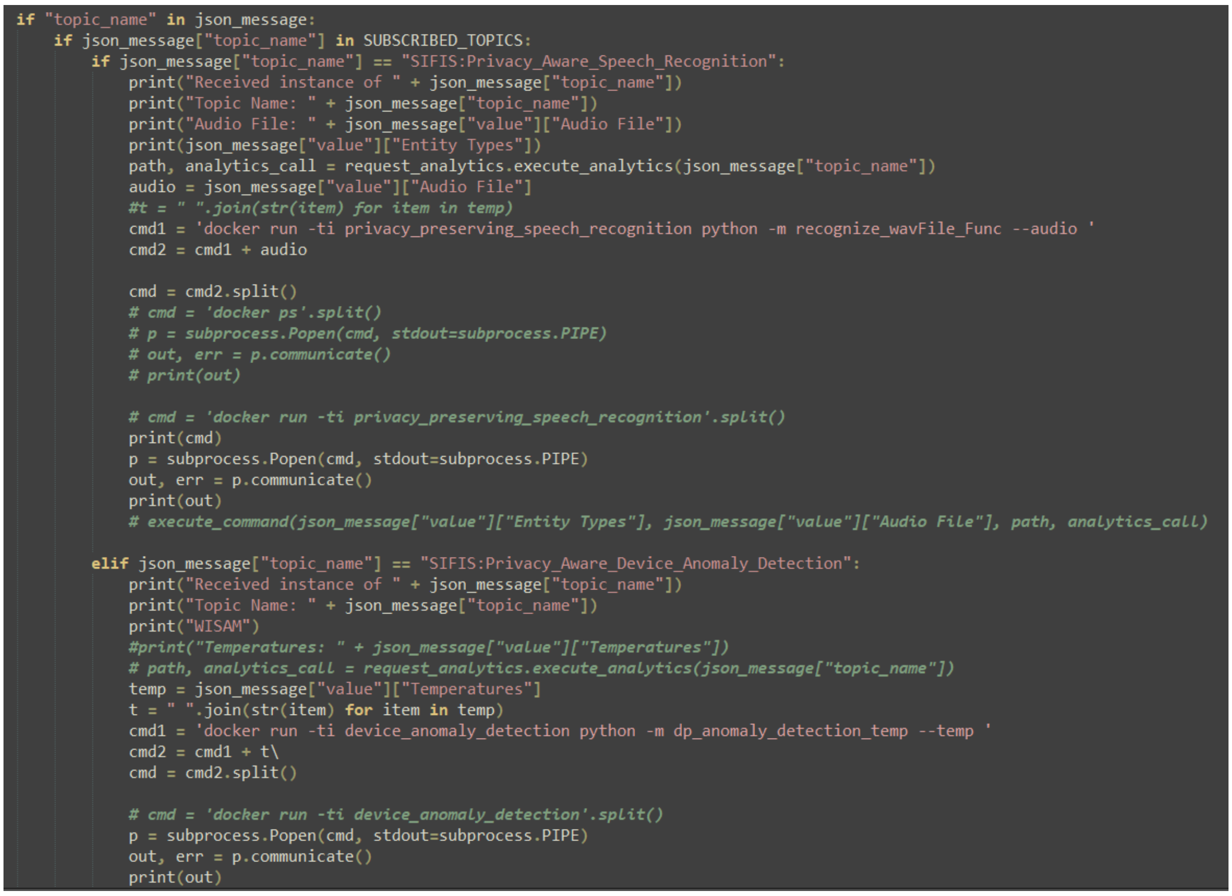
When a component of the SIFIS-Home Framework needs the execution of an analytics, it simply publishes a message on the topic related to these analytics, embedding in the message a JSON string with the details of this invocation (e.g., the link to the set of data to be used, whether the request must be executed once or repeated n times, etc.). The interaction between the Data Analysis toolbox and the DHT starts with a published message as illustrated in Figure 12 to a specific pre-defined topic representing the name of the analytics to be performed on the shared data via JSON format. The published JSON message is composed of the topic name, the topic Id, the ID of the component requesting the data analytics, the description of the requested analytics, and the data inputs needed to perform data analysis, such as an audio file and privacy parameters, and for privacy-aware speech recognition, or a list of temperature values for device anomaly detection analytics. This JSON message is published via an HTTP REST or a WebSocket-based API.



```
def publish(temp):
    ws = websocket.WebSocketApp("ws://localhost:3000/ws",
                                on_open=on_open,
                                on_error=on_error,
                                on_close=on_close)

    ws.run_forever(dispatcher=rel)  # Set dispatcher to automatic reconnection
    rel.signal(2, rel.abort)  # Keyboard Interrupt


    ws_req = {
            "RequestPostTopicUUID": {
                "topic_name": "SIFIS:Privacy_Aware_Device_Anomaly_Detection",
                "topic_uuid": "FirstDev",
                "value": {
                    "Mic_uuid": "FirstDev",
                    "description": "First Dev data",
                    "connected": True,
                    "Data Type": "List",
                    "Temperatures": temp
                }
            }
        }
    ws.send(json.dumps(ws_req))
```

Figure 12: Interaction between the Data Analysis toolbox and the DHT

The Data Analysis Toolbox receives the requests from the DHT, having subscribed to the related topics, and the parameters are included in the messages paired to the requests. Once a published message has been received by the DHT, the data analysis module related to the specified topic is called and executed on the data as in the code presented in Figure 13.



```
if "topic_name" in json_message:
    if json_message["topic_name"] in SUBSCRIBED_TOPICS:
        if json_message["topic_name"] == "SIFIS:Privacy_Aware_Speech_Recognition":
            print("Received instance of " + json_message["topic_name"])
            print("Topic Name: " + json_message["topic_name"])
            print("Audio File: " + json_message["value"]["Audio File"])
            print(json_message["value"]["Entity Types"])
            path, analytics_call = request_analytics.execute_analytics(json_message["topic_name"])
            audio = json_message["value"]["Audio File"]
            #t = " ".join(str(item) for item in temp)
            cmd1 = 'docker run -ti privacy_preserving_speech_recognition python -m recognize_wavFile_Func --audio '
            cmd2 = cmd1 + audio

            cmd = cmd2.split()
            # cmd = 'docker ps'.split()
            # p = subprocess.Popen(cmd, stdout=subprocess.PIPE)
            # out, err = p.communicate()
            # print(out)

            # cmd = 'docker run -ti privacy_preserving_speech_recognition'.split()
            print(cmd)
            p = subprocess.Popen(cmd, stdout=subprocess.PIPE)
            out, err = p.communicate()
            print(out)
            # execute_command(json_message["value"]["Entity Types"], json_message["value"]["Audio File"], path, analytics_call)

        elif json_message["topic_name"] == "SIFIS:Privacy_Aware_Device_Anomaly_Detection":
            print("Received instance of " + json_message["topic_name"])
            print("Topic Name: " + json_message["topic_name"])
            print("WISAM")
            #print("Temperatures: " + json_message["value"]["Temperatures"])
            # path, analytics_call = request_analytics.execute_analytics(json_message["topic_name"])
            temp = json_message["value"]["Temperatures"]
            t = " ".join(str(item) for item in temp)
            cmd1 = 'docker run -ti device_anomaly_detection python -m dp_anomaly_detection_temp --temp '
            cmd2 = cmd1 + t\
            cmd = cmd2.split()

            # cmd = 'docker run -ti device_anomaly_detection'.split()
            p = subprocess.Popen(cmd, stdout=subprocess.PIPE)
            out, err = p.communicate()
            print(out)
```

Figure 13: The data analysis module is called to evaluate a specific topic

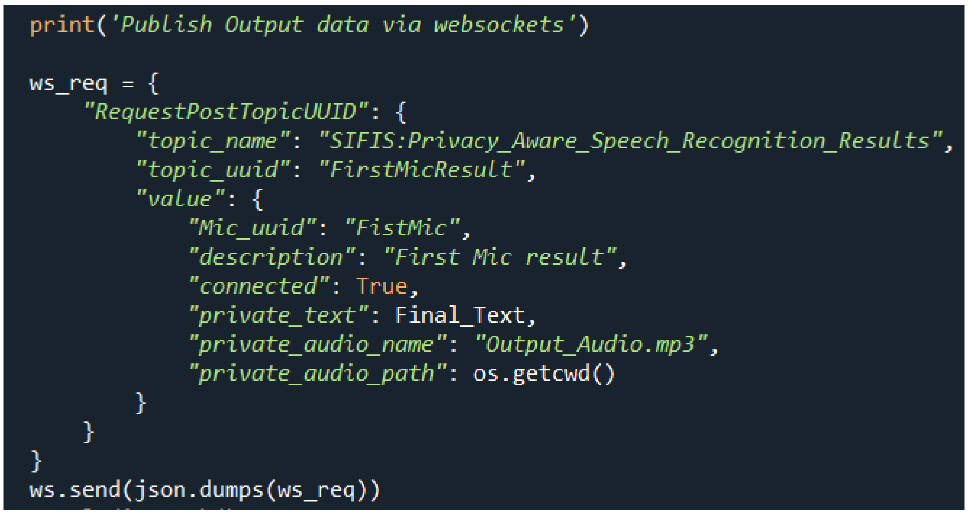
Each analytic has its own execution workflow. Some analytics require the data to be pre-processed before being analyzed. The Data Analysis Toolbox will offer a set of data preparation functions which will involve cleaning missing and noisy data, transforming data into an appropriate and unified format, normalizing data according to a given range of values, and reducing data dimensions by selecting or combining variables into features.

Pre-processing functions will be provided by the Data Analysis Toolbox based on the analytics being invoked. At the time of writing, the analytics integrated within the Data Analysis Toolbox have the required pre-processing steps integrated within the analytics module.

The Data Analysis Toolbox embeds a set of analytics engines, implementing the analytics it provides. Since each analytics engine requires its own execution environment (e.g., a specific library or a specific version of a library, specific or customized utilities, and so on) and the environment of one engine could be not compatible with the environment of one of the others, we have isolated such environments using the container virtualization technology. Hence, each analytics engine is deployed in distinct containers, using the docker technology, and the Data Analysis Toolbox invokes each of such engines through the HTTP REST or the WebSocket-based interface the engine exposes.

Hence, the Data Analysis Toolbox invokes the analytics engine implementing the requested analysis. This engine receives as input the dataset and performs the pre-processing. Some analytics require the results to be post-processed before being returned. Therefore, the post-processing step is also integrated as a function into the data analysis module depending on the analytics being invoked.

Finally, the Data Analysis Toolbox returns the result to the component which has requested the analysis through the DHT as well in a JSON format as shown in Figure 14.



```
print('Publish Output data via websockets')

ws_req = {
    "RequestPostTopicUUID": {
        "topic_name": "SIFIS:Privacy_Aware_Speech_Recognition_Results",
        "topic_uuid": "FirstMicResult",
        "value": {
            "Mic_uuid": "FistMic",
            "description": "First Mic result",
            "connected": True,
            "private_text": Final_Text,
            "private_audio_name": "Output_Audio.mp3",
            "private_audio_path": os.getcwd()
        }
    }
}
ws.send(json.dumps(ws_req))
```

Figure 14: The data analysis result is returned to the requester component

For further details about the analytics toolbox and how each of the analytics is integrated into the security architecture see Section 4.1.

- **Anonymization Toolbox**
  The anonymization toolbox contains software tools to preserve privacy of data before, during, and after analysis. Depending on the data type and the desired level of privacy, the anonymization toolbox can generalize or suppress data, supporting differential privacy for privacy preserving data analysis. At the time of writing, we use several privacy mechanisms according to the data type and the analysis function as below:

  1. **Time Series Data Analysis:**

     Input data: autoencoders are used for anomaly detection and data reconstruction, while the original data are kept private and only the reconstructed data are shared. Moreover, differential privacy is used with autoencoders during the analysis phase to minimize data memorization by the analytics model and protect individual data instances privacy [Dwo08].
     Results: analysis results are classified into categories to prevent information inference of the original dataset.

  2. **Graphical Data Analysis:**

     Input data: the mechanisms of Autoencoders [ZP17] and Differential privacy [Dwo08] are used for privacy-preserving graphical datasets analysis. Autoencoders act as data compressor, reducing the size of data to be analyzed by keeping only the most important features only, thus improving performance and privacy. Instead, Differential Privacy is issued for privacy preservation and minimizing memorization by the learning model. Therefore, the original data are protected, and the results cannot be used to reconstruct the datasets.

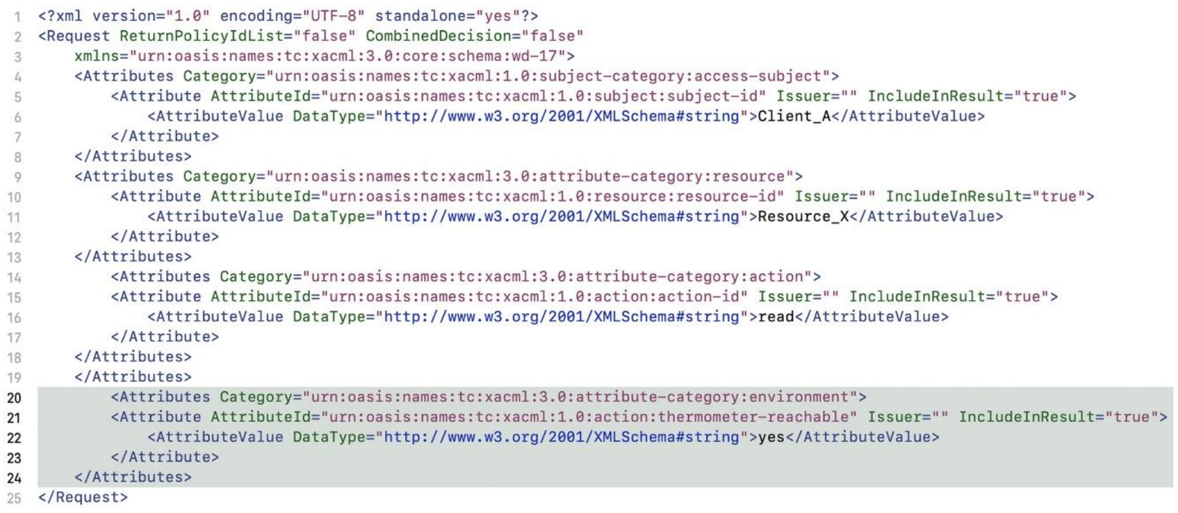
Results: analysis results are classified into categories to prevent information inference of the original dataset.

3. **Audio Data Analysis:**

   Input data: the analytics performs anonymizations for all the sensitive information. **Results**: In the output translation, all the sensitive textual entities detected by the analytic are replaced by default phrases which preserve the essence of the word without revealing the exact word. Also, the audio reconstruction of the anonymized textual translation preserves the privacy of the sensitive entities and replaces the speaker's voice.

- **Policy Enforcement Engine**
  The Policy Enforcement Engine is implemented following the Usage Control (UCON) model [Park et al., 2004]. The UCON model allows dynamic evaluation of access policies through mutable attributes. Attributes are mutable when they change their values over time, due to the normal operation of the system. For instance, the number of people in a room is an example of mutable attribute. An access request is evaluated against a policy, and, if the policy is satisfied, access is granted. Mutable attributes can be part of the policy, which is re-evaluated if some attribute's value change. If the re-evaluation produces a decision of Deny, the access that was previously authorized and, consequently, is currently in progress, is revoked.

  A Java implementation of Usage Control System (UCS) has been exploited to implement the Policy Enforcement Engine. This implementation is an extension of the XACML reference architecture [XACML, 2017] to deal with usage control, it exploits the WSO2 Balana library [Balana, 2021] for implementing the Policy Decision Point (PDP), and it realizes all the other modules of the UCON model, as described in detail in D3.2. The policy is expressed in a language derived from the XACML one, called UPOL, while the communications with the policy enforcement engine occur using the XACML request format. A Policy Enforcement Point (PEP) has been written (in Java) to integrate the UCS within the SIFIS-Home framework. The PEP sends XACML requests to the front end of the UCS, which is the Context Handler (CH). The CH communicates with all the other modules of the UCS. At first, it retrieves all the mutable attributes from the Policy Information Points (PIPs) and "enriches" the XACML request with their values. Figure 15 shows an XACML request coming from the PEP after the manipulation performed by the UCS.



```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Request ReturnPolicyIdList="false" CombinedDecision="false"
    xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
    <Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
        <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" Issuer="" IncludeInResult="true">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Client_A</AttributeValue>
        </Attribute>
    </Attributes>
    <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
        <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" Issuer="" IncludeInResult="true">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Resource_X</AttributeValue>
        </Attribute>
    </Attributes>
        <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
        <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" Issuer="" IncludeInResult="true">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
        </Attribute>
    </Attributes>
    </Attributes>
        <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment">
        <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:thermometer-reachable" Issuer="" IncludeInResult="true">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">yes</AttributeValue>
        </Attribute>
    </Attributes>
</Request>
```

Figure 15: An "enriched" XACML request. The original XACML coming from the PEP has a white background colour, while the mutable attribute added by the UCS is highlighted in grey

Then, the CH retrieves an applicable policy from the Policy Administration Point (PAP) module. Policies stored at the PAP are called Usage Control Policies (UCPs) and are written in UPOL policy language [Di Cerbo et al., 2018], which is XACML based. They are composed of three different sections, i.e., pre-, on- and post- sections, which are evaluated separately and at different times. Figure 16 shows an example of UPOL policy.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Policy
    xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
    PolicyId="policy_0"
    RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit"
    Version="3.0">
   <Description>Policy</Description>
   <Target>
      <AnyOf>
         <AllOf>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
               <AttributeDesignator
                    AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
                    Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
                    DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true" />
               <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Client_A</AttributeValue>
            </Match>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
               <AttributeDesignator
                    AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                    Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
                    DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true" />
               <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Resource_X</AttributeValue>
            </Match>
         </AllOf>
      </AnyOf>
   </Target>
   <Rule Effect="Permit" RuleId="rule-permit">
      <Target />

      <!-- Pre condition -->
      <Condition DecisionTime="pre">
         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
               <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
                  <AttributeDesignator
                        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                        Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
                        DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true" />
               </Apply>
               <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
            </Apply>
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
               <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
                  <AttributeDesignator
                        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                        Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
                        DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true" />
               </Apply>
               <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Resource_X</AttributeValue>
            </Apply>
         </Apply>
      </Condition>

      <!-- On going condition -->
      <Condition DecisionTime="ongoing">
         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
               <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
                  <AttributeDesignator
                        AttributeId="urn:oasis:names:tc:xacml:3.0:environment:thermometer-reachable"
                        Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment"
                        DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true" />
               </Apply>
               <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">yes</AttributeValue>
            </Apply>
         </Apply>
      </Condition>

      <!-- Post condition -->
      <Condition DecisionTime="post">
         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
               <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
                  <AttributeDesignator
                        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                        Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
                        DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true" />
               </Apply>
               <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
            </Apply>
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
               <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
                  <AttributeDesignator
                        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                        Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
                        DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true" />
               </Apply>
               <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Resource_X</AttributeValue>
            </Apply>
         </Apply>
      </Condition>
   </Rule>
   <Rule Effect="Deny" RuleId="urn:oasis:names:tc:xacml:3.0:defdeny">
      <Description>DefaultDeny</Description>
      <Target />
   </Rule>
</Policy>
```

Figure 16: Example of UCP showing the pre-, on-, and post- sections of the policy. The mutable attribute "thermometer-reachable" must match "yes" for the on- section to be satisfied against a request

Then, the CH queries the PDP module to obtain an access decision. The PDP uses the XACML-based WSO2 Balana decisional engine for policy evaluation. However, being Balana a pure-XACML decisional engine, the CH extracts either the pre-, on-, or post- section from the UCP and creates an XACML policy before feeding the PDP. If the access decision produced by the PDP is Permit, the CH sends the original XACML request and the UCP to the Session Manager (SM) module, which selects a new session identifier and creates a new entry in the database (an in-memory JDBC database) with the received information. Finally, the CH sends back a message to the PEP communicating the access decision, and the session identifier if the decision was Permit.

A PIP monitors a mutable attribute, which is stored at an Attribute Manager (AM). Currently, in our implementation, AMs can be a database or a file, so PIPs implement the logic to retrieve the attribute values, either by querying the database or by retrieving the file. When a PIP is monitoring a mutable attribute, it periodically polls the AM to retrieve the current value and compares it with the value obtained during the previous poll. If these values differ, the CH starts a routine to re-evaluate all the sessions in the SM's database that are currently using that attribute in their on- section of the UCP. If a policy is not satisfied anymore, the corresponding access must be revoked. So, the CH sends a revocation message to the PEP, which interrupts the access to the resource and notifies the CH.

3.3.5 API Gateway

This component, Figure 17, provides the API's used by applications executed on the SIFIS-Home smart devices.

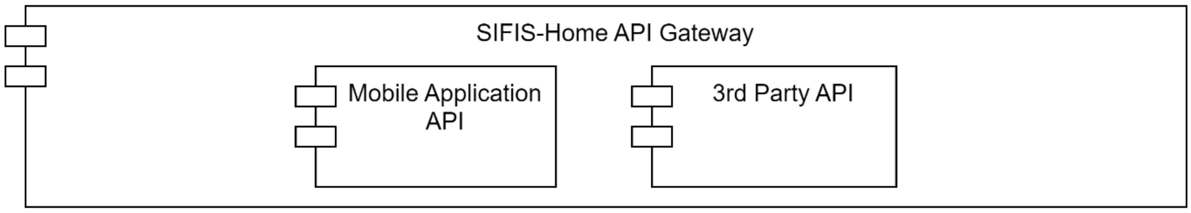


Figure 17: The API Gateway with the mobile and 3rd party API

- **Mobile Application API**
  This component allows the SIFIS mobile application to interact with the smart home and initialize new SIFIS-Home-compliant Smart Devices.

  When the Smart Device is started for the first time, it starts in the initialization mode. In the initialization mode, the device presents a Wi-Fi access point allowing the mobile application to join the device's local configuration network and set up the device.

  The mobile application provides all information needed for the new Smart Device to join the SIFIS-Home network. This information includes network settings, DHT credentials, and the device's information, such as its name in the SIFIS-Home network. After configuration, the device is restarted, and all SIFIS-Home services are started on the next boot.

  The mobile application requires an access token to use Mobile Application API securely. This token is delivered as a QR code with the Smart Device.

- **3rd party API**
  This component keeps the API for that allows downloaded 3rd part applications to interact with the smart home. This module is still under definition.

3.3.6 NSSD Manager

The NSSD Manager is the component that allows the management of the NSSDs that are part of the smart home.

- **CoAP Manager**
  On a device acting as CoAP client, the CoAP Manager receives commands and retrieves information from the DHT manager, and then takes care to accordingly execute the requested operations, by interacting with the targeted CoAP server device(s). The communication with such CoAP server device(s) is protected and established by using the advanced security protocols and solutions that have been designed and developed in the context of WP3.

  This enables a more convenient, remote provisioning of commands to devices acting as CoAP client, which becomes especially relevant when: i) a CoAP client does not provide a direct or convenient input interface; or ii) the user prefers to remotely instruct the CoAP client or simply has to, e.g., as currently not present in the Smart Home and relying on a remote access over the Internet.
  To this end, a CoAP client can act as pub-sub client in the pub-sub system enforced in the Smart Home (e.g., by interacting with a DHT-based pub-sub broker) in order to receive published commands, then take an appropriate course of action and accordingly communicate with the target CoAP server(s), and finally publish corresponding results to be eventually displayed to a user.
  Further details about the implementation strategy for this component are provided in Section 4.3.
- **WoT Manager**
  WoT enables direct control of smart home devices over the web by giving them URLs, making them discoverable and linkable, also defining a standard data model and APIs to make the devices interoperable and to exchange data between devices and systems. The flow is visualized in Figure 18.

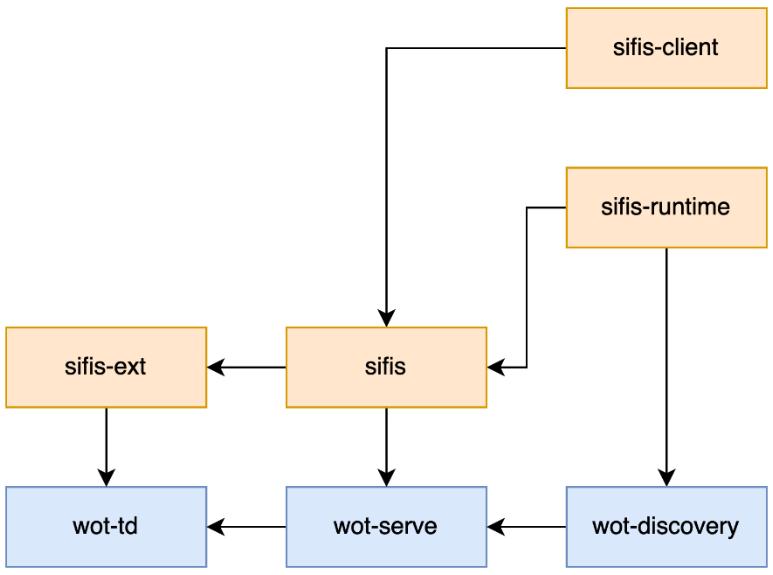


Figure 18: The WoT manager interaction flow

The WoT manager will interact with SIFIS-Home DHT using the **sifis-client** and the **sifis-runtime** components that act as translation layer between the SIFIS-Home concepts and the broader WoT ones.

The implementation keeps a separation between the WoT general components listed in Table 2, that can be reused by a larger public, and the SIFIS-Home-specific ones augmented using the Hazard Ontology.

Crates	Description	Status
wot-td	Produce and consume Thing Description	Release 0.2
wot-serve	Serve Things using HTTP and mDNS-SD	Release 0.2
wot-discovery	Discover Things in the network	Work in Progress
libsifis	Initial Proof of Concept	Being replaced by **sifis-ext** and **sifis** crates
sifis-ext	Thing Description extension	Work in Progress
sifis-rust	SIFIS-Home rust data structures and Servient	Work in Progress

Table 2 WoT components

3.3.7 DHT Manager

The DHT Manager (Figure 19) is composed of two different software components, the DHT and the Fiware API component. The DHT component manages the DHT and allows access to it from other applications and services. Instead, the Fiware API component interacts with Yggio in order to offer a FIWARE compatible API to a SIFIS-Home enabled smart home.

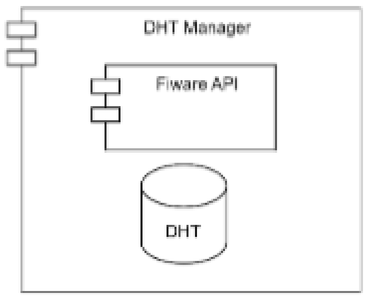


Figure 19: The DHT manager components

- **FIWARE API**
  This component forwards the persistent and volatile messages published through the DHT to the Yggio instance residing on the SIFIS-Home cloud. Also, it takes care to forward configurations and messages sent to Yggio to the DHT. The Fiware API component uses the HTTP interface offered by the DHT to store and retrieve messages inside/from the DHT. Also, it uses the MQTT protocol to connect to the MQTT broker used by Yggio and, hence, receive/publish messages from/to Yggio.

- **DHT**
  The DHT allows applications installed on different SDs to communicate by using a completely distributed publish/subscribe mechanism (i.e., without a central broker). Also, through the DHT, it is possible to store a certain set of messages in a persistent way to make them available in case of node failures/reboots (e.g., settings and policies should be stored in a persistent way). The DHT has been developed using Rust. The DHT can be embedded into native Rust applications as a library. Non-native Rust applications can interact with the DHT using its HTTP interface. In particular, the DHT offers both an HTTP REST and a WebSocket-based API to access the messages stored inside it. The current version of the SIFIS-Home DHT code can be found at https://github.com/sifis-home/libp2p-rust-dht.

3.4 *NSSD Framework*

The SIFIS-Home NSSD Framework (Figure 20) is the set of components that are expected to be present on every NSSD device that should be part of a SIFIS-Home enabled smart home. In detail, the NSSD Framework is composed of the Bootstrap Manager component and the Device API Manager.

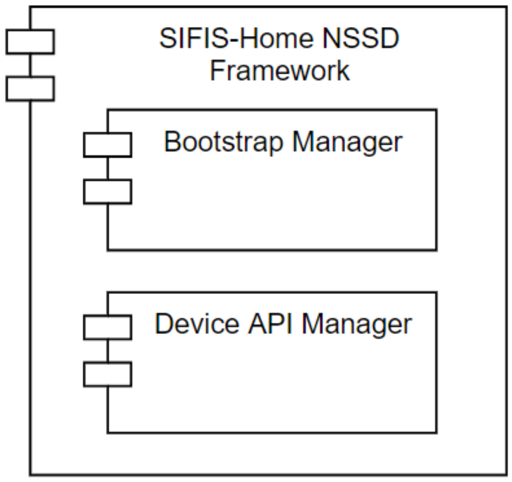


Figure 20: The NSSD Framework components

3.4.1 Bootstrap Manager

This component allows a NSSD device to receive all the information needed to allow it to join a SIFIS-Home network. In detail, in order for a NSSD to join a SIFIS-Home network, it should be provided with i) the Wi-Fi credentials (i.e., SSID and Password) of the network to which the device should connect to and ii) the shared key/authorization token that should be provided by any SD wanting to interact with the NSSD during its lifetime. In detail, the Bootstrap Manager is going to provide a number of HTTP API endpoints through which it is possible to set the Wi-Fi credentials and the key/authorization token that the NDDS should use. When the NSSD receives the Wi-Fi credentials and the token, it performs a reboot operation and connects to the configured Wi-Fi network. Also, it will not be able to control the NSSD if not provided with the correct token.

3.4.2 Device API Manager

This component offers the API through which a NSSD can be operated. In the case of a WebThings enabled NSSD, all the functionalities of the NSSD are offered by means of a set of properties, actions, events. WebThings makes it possible to execute an action, receive an event and set a property using the HTTP protocol. The WebThings functionalities on microcontroller-based devices will be implemented using the WebThings Arduino Library (https://github.com/WebThingsIO/webthing-arduino).

4 Integration

4.1 *Integration strategy*

The design of the SIFIS-Home framework in WP1 is based on Docker containers with *microservices* design pattern. Each microservice defines an API, usually a Rest API, for interfacing with it. To add new modules to the system one just needs to add new Docker containers that hold the new modules. The new modules can then directly start interacting with the existing modules in the system based on their provided API's. This makes it fast and easy to add new functionality.

4.2 *Analytics integration (WP4)*

The integration of the code implementing the analytics designed and implemented in WP4 within the Data Analysis Toolbox is performed through a common procedure:

1. The analytics designed for SIFIS-Home require different (and sometimes even conflicting) execution environments, depending on the software libraries they use (sometimes even on the version of them). For this reason, we decided to use virtualization technology and to deploy each analytics in its customized container, leveraging on the docker technology. Hence, each analytics environment is structured and created within a specific docker image.
2. Docker images are used to create docker containers on the SIFIS-Home devices for analytics deployment. A repository of analytics docker container images will be created to easily store and retrieve available images.
3. The analytics running within the docker containers resulting from the previous step are being invoked by the Analytics APIs. Each analytics provided by the Data Analysis toolbox is paired to a topic. Moreover, each analytics has a separate topic for the results obtained by running it on some data.
4. The interactions between the Data Analysis Toolbox and the other components of the SIFIS-Home Framework (which need to ask for analytics executions) occur through the DHT, which provides a publish/subscribe mechanism.
5. Interactions start from analytics consumer side requesting the execution of an analytics. This request is represented by a message in JSON format published on the topic related to analytics. The JSON message contains invocation details in addition to a set of data to be used and privacy parameters to be used if any.
6. On message publish action, the Analytics APIs subsystem invokes the analytics of the related topic. As previously mentioned, each analytics is deployed in its own container. The invocation of an analytics by the Analytics APIs subsystem occurs exploiting the HTTP REST or the WebSocket protocol, depending on the analytics deployment on the related container. As a matter of fact, each analytics has its own protocol, and needs different parameters to be passed.
7. Once the analytics has been executed and the result is available, the Data Analysis Toolbox returns the result to the component which has requested the analysis through the DHT using the HTTP REST or the WebSocket-based API in a JSON format.

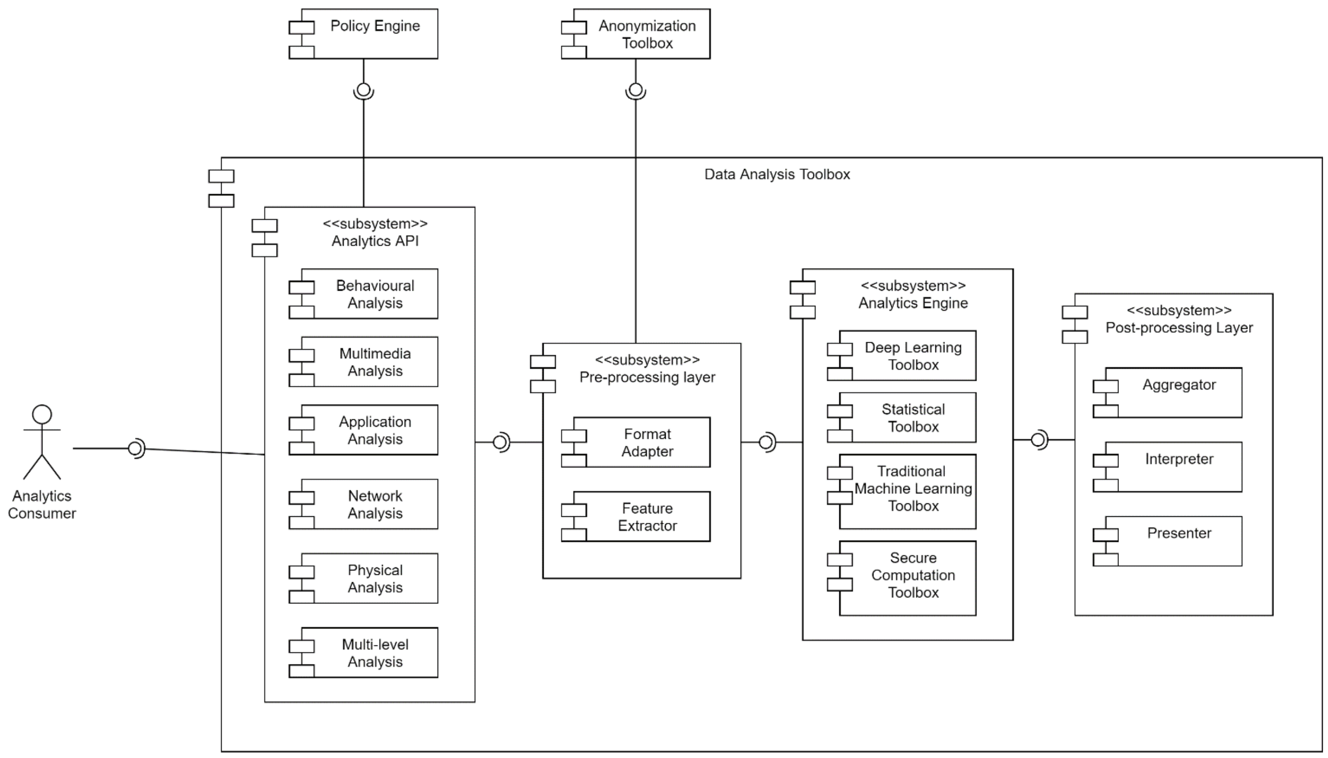


Figure 21: Analytics components

- **Pre-Processing Layer/Post Processing**
  This component takes care to pre-process data (adjust format and remove unused information) coming from different sensors and devices before sending it to the different SIFIS-Home analytics. Also, it takes care to collect the information produced by the different analytics and prepare it for use by part of the frontend applications.

  On the one hand, data Pre-processing for privacy protection purposes is performed locally on the device requesting analytics before data publishing, such as using autoencoders or Gaussian Blurring on graphical datasets. However, data pre-processing for data cleaning and preparation is implemented before the analytics module implementation on the device performing data analysis. On the other hand, data post-processing is implemented directly after the implementation of the analytics method on the same device analysing the data for aggregation and categorization purposes of the results as an example.

- **Analytics API**
  As previously explained, the Analytics API subsystem is in charge to interact with the DHT to subscribe the topics representing all the analytics provided by the Data Analysis Toolbox and to retrieve the requests sent by the other components of the SIFIS-Home framework by publishing on these topics. Depending on the topic published by the component requesting the execution of an analytics, the right component of the Analytics API subsystem is invoked. Such component knows how to interact with the requested analytics which is running in its docker container, and if a pre-processing operation is required, the right function of the pre-processing layer subsystem is also invoked. At the time of writing, we have the following component of the Analytics API subsystem:

  - **Behavioural Analysis**

This API uses WebSocket to invoke analytics toolboxes responsible for the analysis of behavioural and environmental data collected from IoT devices like sensors, smart watches, smart thermostats, smart cameras to derive insights, track user behaviour, predict user behaviour, identify anomalous actions, or for activity classification. Behavioural and environmental data are captured by SIFIS-Home sensors and devices and published with the device ID to the required analytics topic. The analytics topic is used to invoke the WebSocket-based API that the analytics expose, in order to execute the responsible docker container with the shared data and parameters. The results are then published by the Analytics Engine to the related topic.

- **Network Analysis**
  Network analysis is concerned with performing network traffic analysis to identify hidden and complex patterns and anomalous behaviours or security threats using stream data and packet data. Common functions carried out by this WebSocket-based API include continuous monitoring of network traffic, malware detection, and network abnormal behaviour detection and troubleshooting. Stream data and packet data are captured and published with the device ID to the required analytics topic. The analytics topic is used to invoke the related WebSocket-based API for docker container execution with the shared data and parameters. The results are then published by the Analytics Engine to the desired topic. Centria's analytics will provide REST API that provides service status(start/stop/restart/is_running), available interfaces, configuration, data and alarms. Optionally webhooks for data and alarms are also available.
  The network anomaly detection analytic called Aggregated Usage Description (AUD), developed by FSEC is an analytic that is implemented as a background service. The analytic is executed in a container labelled "aud_manager", and it is intended to continuously analyze network traffic. AUD manager operates directly with network and transport layer (L3 & L4) information, which reside below session and application layers through which vulnerabilities such as session hijacking and man-in-the-middle attacks are often exploited. The analytic developed in WP4 strives to spot the anomalous network events before the data packets reach their destinations, i.e., target devices. AUD manager provides a REST API for starting and stopping the analytic. The REST API also provides methods to report current state of the analytic, as well as runtime diagnostics, such as network statistics and information about recent anomalies. Upon detecting an anomaly, the analytic calls the Evaluator/Notifier in the Proactive Security Management Layer.

- **Multimedia Analysis**
  This WebSocket-based API is invoked after messages with recorded and captured multimedia data are published on the DHT to topics concerned with such data analytics. The docker container related to the topic is triggered to execute the analysis on the captured data and parameters and the results obtained are published by the Analytics Engine to the required topics.

- **Application Analysis**
  Application data are captured by SIFIS-Home components and published with the component ID to the wanted analytics topic. The analytics topic is used to invoke the WebSocket-based API which the analytics expose, in order to execute the responsible docker container with the shared data and parameters. The results are then published by the Analytics Engine to the related topic.

- **Physical Analysis**
  Physical data are collected by SIFIS-Home components and published with the

component ID to the wanted analytics topic. The analytics topic is used to invoke the WebSocket-based API the analytics expose, in order to execute the responsible docker container with the shared data and parameters. The results are then published by the Analytics Engine to the related topic.

- **Analysis Engine**
  This component handles the execution of SIFIS-Home analytics, providing statistical-based, machine learning, and deep learning tools for the analysis of data collected by sensors. The analysis engine processes collected data and makes logical predictions based on the provided data input and instructions it has been configured on. Finally, it outputs the data and shares it back with the analytics consumer.
  The Data Analysis Engine is implemented as a collection of docker containers, integrated via a nested if statement for the received message topic from components publishing data to be processed. Depending on the topic specified in the published message, the right docker container is selected to be executed with the data and parameters included in the message. After the docker execution is finished, the results obtained are also published back by the Data Analysis Engine to the analytics result topic. The details concerning the implementation of each single analytics have been given in D4.2.

4.3 *Network and security solution integrations (WP3)*

The security solutions developed in WP3 build on the CoAP web-transfer protocol and the OSCORE security protocol for CoAP, and provide related security services to enforce secure end-to-end (group) communication, management of keying material, and fine-grained enforcement of access and usage control. These security solutions are presented in D3.2, which will be updated and obsoleted by D3.3.

Regardless the specifically considered, possibly combined, security solutions, interacting parties are devices acting as CoAP client and/or CoAP server. In stand-alone proof-of-concept implementations used for focused demonstrations, the user could interact with a CoAP client, in order to provide a command available from a basic set, and thus make the CoAP client take a corresponding course of action in communicating with the intended CoAP server(s).

Such a user interaction was simply based on a command line interface. That is, the user could provide a command via keyboard to the CoAP client, which in turn provided any relevant output (e.g., responses obtained from the CoAP servers) on a display.

While this approach made it possible to effectively run stand-alone demonstrations, it is inconvenient in a broader, more realistic setup where the user wants to remotely provide the CoAP client with a command to process, e.g., by using an application running the SIFIS-Home framework. In addition to that, the user might not even be presently at home, hence requiring to reach out the CoAP client through the Internet. In either case, rather than relying on a physical interaction with a keyboard to feed a command line interface, the user would better remotely provide CoAP clients with commands to process.

To this end, the following strategy has been devised, and it will be refined and followed in the coming months during the WP5 activities.

From a logical point of view, the CoAP clients expecting commands from a user additionally rely on the availability of a publish-subscribe system in order to: i) obtain user commands as content published on a related, device-specific topic, rather than input provided on a terminal; and ii) relay back any relevant output following the course of action triggered by the command, as content published on a related, device-specific topic rather than as output on a display.

Practically, this can be enforced by means of two different approaches, both relying on the publish-subscribe communication model.

- An MQTT publish-subscribe system, with the MQTT broker locally deployed in the Smart Home network. Practically, a device acting as CoAP client would also act as MQTT client, by subscribing to a topic on which it receives user commands, and by publishing on a related topic where it provides the output from the actions performed following those commands. The user application takes the reversed pub-sub roles, in order to publish commands and subscribe to outputs from the CoAP client. The two topics are specific for the CoAP client in question.

- A distributed publish-subscribe system, based on a Distributed Hash Table (DHT) deployed within the Smart Home network. Practically, the DHT enforces a distributed pub-sub broker, exposing an interface based on WebSockets for publishing on and subscribing to its topics. Then, a device acting as a CoAP client would also act as a pub-sub client, by bi-directionally interacting with the DHT-based pub-sub broker through WebSockets. Similarly, to the first approach discussed above, the CoAP client would subscribe to a topic on which it receives user commands and would publish on a related topic in order to provide the output from the actions performed following those commands. The user application takes the reversed pub-sub roles, in order to publish commands and subscribe to outputs from the CoAP client.

In either case, an "outer" user can also interact with the CoAP clients through the Internet, when not presently in the Smart Home. This requires an additional step, also based on a pub-sub model and relying on the Sensative MQTT broker deployed in the cloud. The outer user would leverage the same model discussed above, although having a direct interaction specifically MQTT-based and specifically with the Sensative MQTT broker. Then, the Sensative MQTT broker will interact with the publish-subscribe system deployed in the Smart Home, just like the user would do for the case discussed above where the user is presently at home.

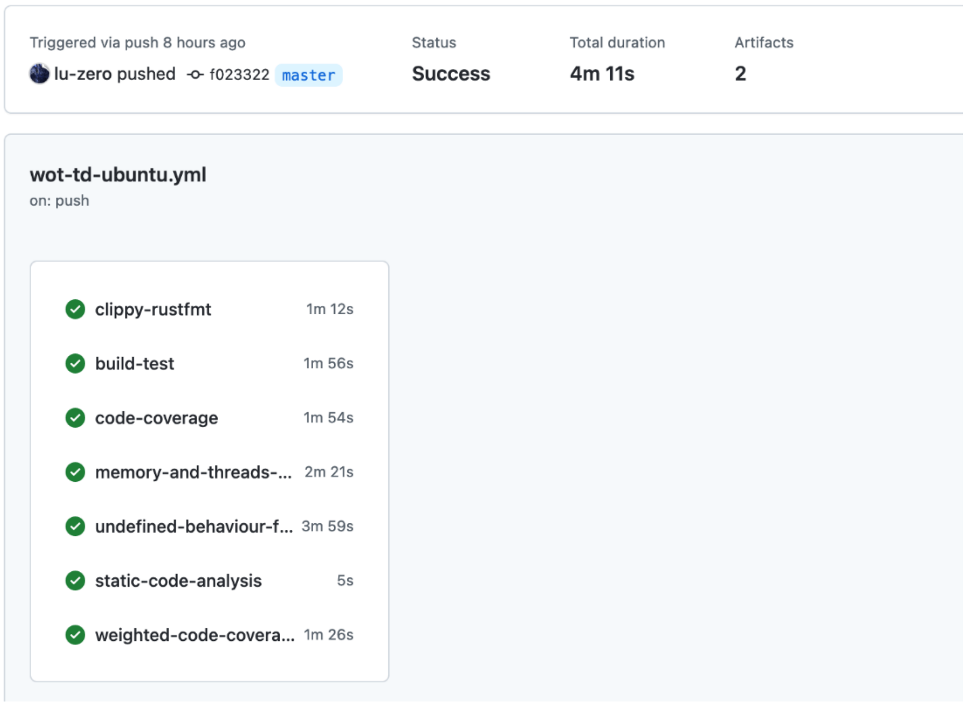
When setting up such a pub-sub route, care must be taken when setting the "outer" pub-sub topics at the Sensative MQTT broker to be consistent with the "inner" pub-sub topics of the pub-sub system deployed in the Smart Home. For example, given an inner topic "COMMAND_DEV1" used to publish commands intended to "Device_1" acting as a CoAP client, the corresponding outer topic can be "HOME_42_COMMAND_DEV_1", where "HOME_42_" is a prefix used to uniquely identify the specific Smart Home and thus enforce per-smart-home topic namespaces.

5 Verification status and results

Since implementations and integration for many of the SIFIS-Home modules are still ongoing, project partners have made their own unit test to secure the code quality of the modules and functionality they provide. Once the system is more completely integrated, the official system test of the full SIFIS-Home framework will start and will be reported in next deliverable.
Some of the mentioned unit tests executed by partners are described below.

5.1 *wot-rust crates*

All the crates use the continuous integration setup for GitHub-Actions provided by the **sifis-generate** (Figure 22).



Triggered via push 8 hours ago
lu-zero pushed -o- f023322 master

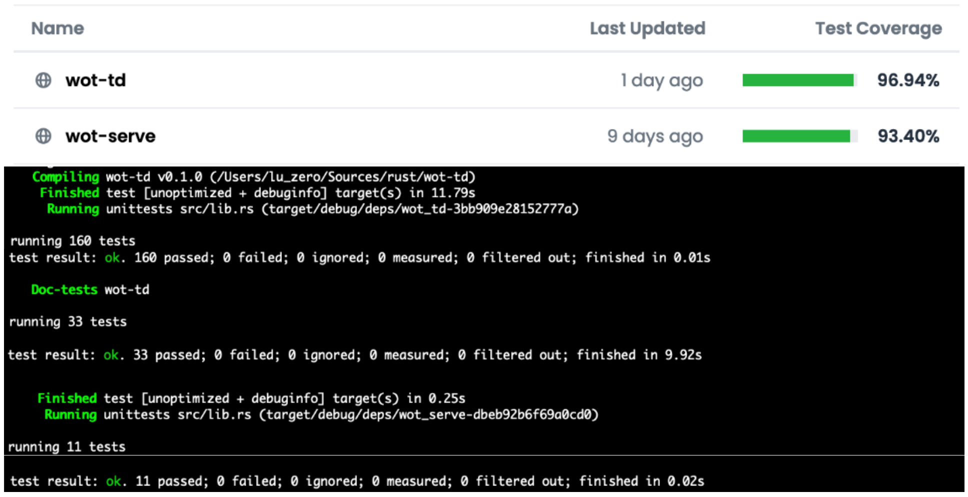
Status	Total duration	Artifacts
Success	4m 11s	2

wot-td-ubuntu.yml
on: push

✓ clippy-rustfmt	1m 12s
✓ build-test	1m 56s
✓ code-coverage	1m 54s
✓ memory-and-threads-...	2m 21s
✓ undefined-behaviour-f...	3m 59s
✓ static-code-analysis	5s
✓ weighted-code-covera...	1m 26s

Figure 22: Continuous integration setup

The crates try to stay above the 90% coverage every commit (Figure 23).



Name	Last Updated	Test Coverage
wot-td	1 day ago	96.94%
wot-serve	9 days ago	93.40%

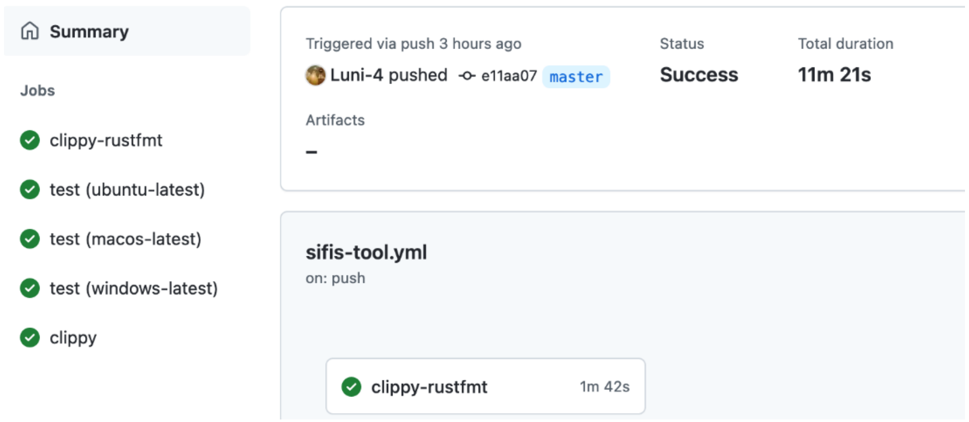
```
   Compiling wot-td v0.1.0 (/Users/lu_zero/Sources/rust/wot-td)
    Finished test [unoptimized + debuginfo] target(s) in 11.79s
     Running unittests src/lib.rs (target/debug/deps/wot_td-3bb909e28152777a)

running 160 tests
test result: ok. 160 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.01s

   Doc-tests wot-td

running 33 tests

test result: ok. 33 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 9.92s


    Finished test [unoptimized + debuginfo] target(s) in 0.25s
     Running unittests src/lib.rs (target/debug/deps/wot_serve-dbeb92b6f69a0cd0)

running 11 tests

test result: ok. 11 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.02s
```

Figure 23: Automated tests of every commit

The built-in **cargo test** ensures that both the code and the examples in the documentation are tested.

5.2 *sifis-generate*

The **sifis-generate** crate is also using its own continuous integration template as visualised in Figure 24.


Summary

Jobs

✓ clippy-rustfmt

✓ test (ubuntu-latest)

✓ test (macos-latest)

✓ test (windows-latest)

✓ clippy

Triggered via push 3 hours ago

Luni-4 pushed e11aa07 master

Status: Success

Total duration: 11m 21s

Artifacts: –

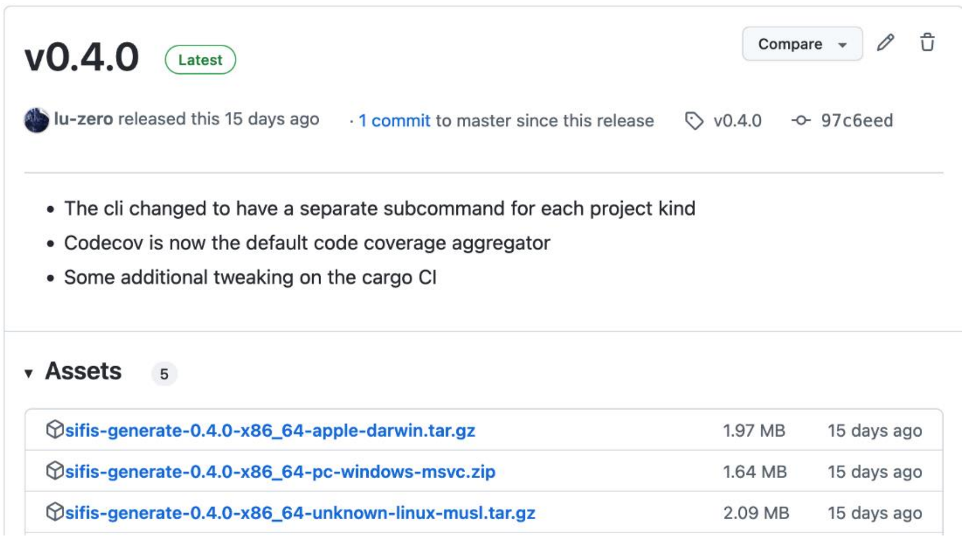
sifis-tool.yml

on: push

✓ clippy-rustfmt 1m 42s

Figure 24: Continuous integration template for sifis-generate

Being an executable, it also peruses the deploy component to provide prebuilt binaries (Figure 25) for Linux, Windows and macOS (every release).


v0.4.0 Latest

Compare

lu-zero released this 15 days ago · 1 commit to master since this release v0.4.0 97c6eed

- The cli changed to have a separate subcommand for each project kind
- Codecov is now the default code coverage aggregator
- Some additional tweaking on the cargo CI

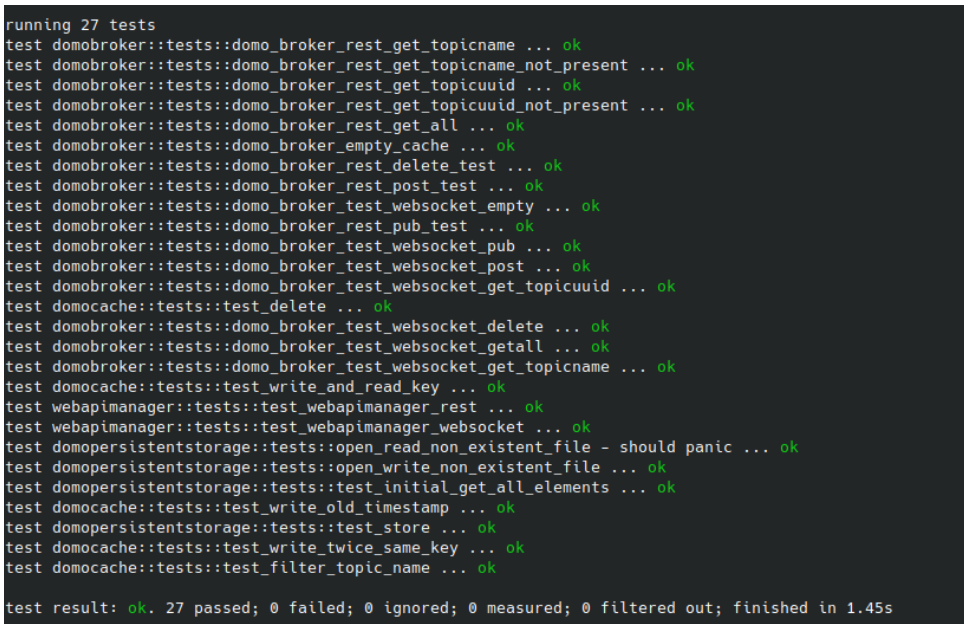
Assets 5

sifis-generate-0.4.0-x86_64-apple-darwin.tar.gz	1.97 MB	15 days ago
sifis-generate-0.4.0-x86_64-pc-windows-msvc.zip	1.64 MB	15 days ago
sifis-generate-0.4.0-x86_64-unknown-linux-musl.tar.gz	2.09 MB	15 days ago

Figure 25: Prebuilt binaries

5.3 *DHT*

The current implementation of the SIFIS-Home DHT has been tested using standard unit tests. The following Figure 26 reports the current test results.



```
running 27 tests
test domobroker::tests::domo_broker_rest_get_topicname ... ok
test domobroker::tests::domo_broker_rest_get_topicname_not_present ... ok
test domobroker::tests::domo_broker_rest_get_topicuuid ... ok
test domobroker::tests::domo_broker_rest_get_topicuuid_not_present ... ok
test domobroker::tests::domo_broker_rest_get_all ... ok
test domobroker::tests::domo_broker_empty_cache ... ok
test domobroker::tests::domo_broker_rest_delete_test ... ok
test domobroker::tests::domo_broker_rest_post_test ... ok
test domobroker::tests::domo_broker_test_websocket_empty ... ok
test domobroker::tests::domo_broker_rest_pub_test ... ok
test domobroker::tests::domo_broker_test_websocket_pub ... ok
test domobroker::tests::domo_broker_test_websocket_post ... ok
test domobroker::tests::domo_broker_test_websocket_get_topicuuid ... ok
test domocache::tests::test_delete ... ok
test domobroker::tests::domo_broker_test_websocket_delete ... ok
test domobroker::tests::domo_broker_test_websocket_getall ... ok
test domobroker::tests::domo_broker_test_websocket_get_topicname ... ok
test domocache::tests::test_write_and_read_key ... ok
test webapimanager::tests::test_webapimanager_rest ... ok
test webapimanager::tests::test_webapimanager_websocket ... ok
test domopersistentstorage::tests::open_read_non_existent_file - should panic ... ok
test domopersistentstorage::tests::open_write_non_existent_file ... ok
test domopersistentstorage::tests::test_initial_get_all_elements ... ok
test domocache::tests::test_write_old_timestamp ... ok
test domopersistentstorage::tests::test_store ... ok
test domocache::tests::test_write_twice_same_key ... ok
test domocache::tests::test_filter_topic_name ... ok

test result: ok. 27 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 1.45s
```

Figure 26: The DHT verification results

5.4 *Security solutions (WP3)*

The security solutions developed in WP3 and indicated in Sections 3.4.1 and 3.4.2 have been successfully tested through focused demonstrators, by relying on the corresponding implementations and real hardware platforms. In particular:

- The first demonstrator considered a CoAP group communication scenario, where: i) devices rely on the ACE Framework and its OSCORE profile, in order to securely interact with an OSCORE Group Manager and obtain keying material for Group OSCORE; ii) as members of the security group, such devices securely communicate with other group members using CoAP and Group OSCORE, protecting group messages with its group mode or pairwise mode.

- The second demonstrator considered a CoAP client and a CoAP server that first establish an OSCORE Security Context through the EDHOC key establishment protocol, and then securely communicate with one another using the established OSCORE Security Context. The EDHOC key establishment considered: i) peer authentication based on either signatures through private signing keys, or MACs through static-static Diffie-Hellman keys; and ii) the original EDHOC workflow or an optimized, shortened EDHOC workflow which requires one round-trip less by combining the last EDHOC message with the first OSCORE-protected message.

5.5 *Cloud UI components*

The cloud UI components that reside on top of the FIWARE Ratatosk Context Broker are major implementations that include thousands of lines of code written in REACT and Node.JS. These UI components have been up and running on the SIFIS-Home Panarea server with live NSSD data since January 2022 and are also commercially released since May 2022 in Sensative Yggio horizontal IoT integration platform. The UI components are extensively verified from all aspects for commercial distribution in every release of a new set of functionalities.

5.6 *Network Anomaly Detection / AUD Manager*

The basic functionalities of AUD manager have been tested with virtual mock devices. Further testing and validation of AUD manager is still in progress.

6 Conclusion

In this deliverable we have presented the status of the current implementation of the SIFIS-Home security framework and the latest test results. It is a very extensive implementation of state-of-the-art technologies done by many different partners across Europe with access to different type of competence, tools, legacy code and commercial interest to consider. A key step to make it happen was when the implementation allocation of the SIFIS-Home framework components were done in 2021 and it was vitally important to have the right partners for every component. We do believe we made the best possible allocation of all components and partners have been working very hard together to get the implementations done and sort out every problem once they were discovered. We now are on the 3rd iteration of the security architecture defined in D1.4 that has been refined step by step from the original architecture described in D1.3 and we have also added some missing integration modules to provide the glue layers to make all work together.

Going forward we will define, allocate and complete the missing implementation while also aim to get the system verification up and running as soon as possible, so to be able to verify all the defined use cases.
In summary, our self-assessment of the work done so far is that we are overall on-track and aligned with the planned set of action in the project.

7 References

[Maymounkov et al. 2002] P. Maymounkov, D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric", Peer-to-Peer Systems. IPTPS 2002. Lecture Notes in Computer Science, vol 2429. Springer, Berlin, Heidelberg

[Faiella et. al 2016] Mario Faiella, Fabio Martinelli, Paolo Mori, Andrea Saracino, Mina Sheikhalishahi: Collaborative Attribute Retrieval in Environment with Faulty Attribute Managers. ARES 2016: 296-303

[WoT, 2020] Web Of Things (WoT) Architecture, W3C recommendation 9 April 2020, https://www.w3.org/TR/wot-architecture/

[FIWARE, 2021] What is FIWARE?, https://www.fiware.org/developers/

[YGGIO, 2021] Yggio DiMS, Digitalization infrastructure Management System, https://sensative.com/yggio/

[La Marra et al, 2017] Antonio La Marra, Fabio Martinelli, Paolo Mori, Andrea Saracino: Implementing Usage Control in Internet of Things: A Smart Home Use Case. TrustCom/BigDataSE/ICESS 2017: 1056-1063

[Facchini et al, 2020] Simone Facchini, Giacomo Giorgi, Andrea Saracino, Gianluca Dini: Multi-level Distributed Intrusion Detection System for an IoT based Smart Home Environment. ICISSP 2020: 705-712

[Saracino et al, 2021] M Sheikhalishahi, A Saracino, F Martinelli, A La Marra, Privacy preserving data sharing and analysis for edge-based architectures, International Journal of Information Security, 1-23

[XACML, 2017] eXtensible Access Control Markup Language (XACML) version 3.0 plus errata 01 (2017). URL: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.html

[Perkins, 1999] C.E. Perkins, E.M. Royer, Ad-hoc on-demand distance vector routing, Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications

[Dwo08] C. Dwork. Differential privacy: A survey of results. In International conference on theory and applications of models of computation, pages 1–19. Springer Berlin Heidelberg, 2008.

[ZP17] Zhou, C., & Paffenroth, R. C. (2017, August). Anomaly detection with robust deep autoencoders. In Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 665-674).

[Park et al., 2004] Park, J., & Sandhu, R. (2004). The UCONABC usage control model. ACM transactions on information and system security (TISSEC), 7(1), 128-174.

[Di Cerbo et al., 2018] Di Cerbo, F., Lunardelli, A., Matteucci, I., Martinelli, F., & Mori, P. (2018, September). A declarative data protection approach: from human-readable policies to automatic enforcement. In International Conference on Web Information Systems and Technologies (pp. 78-98). Springer, Cham.

[Balana, 2021] WSO2 Balana implementation. URL: https://github.com/wso2/balana

[RFC8520] Lear, E., Droms, R., and D. Romascanu, "Manufacturer Usage Description Specification", RFC 8520, DOI 10.17487/RFC8520, March 2019. URL: https://www.rfc-editor.org/info/rfc8520

Glossary

Acronym	Definition
ACE	Authentication and Authorization for Constrained Environments
AM	Attribute Manager
AODV	Ad-hoc On-demand Distance Vector
API	Application Programming Interface
CoAP	Constrained Application Protocol
CH	Context Handler
DHT	Distributed Hash Table
EDHOC	Ephemeral Diffie-Hellman Over COSE
FR	Functional Requirements
HTTP	Hyper Text Transfer Protocol
JSON	JavaScript Object Notation
MQTT	MQ Telemetry Transport
MUD	Manufacturer Usage Description
NFR	Non-functional requirement
NSSD	Not So Smart Device
OS	Operative System
OSCORE	Object Security for Constrained RESTful Environments
PAP	Policy Administration Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
PTP	Policy Translation Point
P2P	Peer to Peer
REST	Representational State Transfer
SD	Smart Device
SIFIS-Home	Secure Interoperable Full Stack Internet of Things for Smart Home
SSH	Secure Shell
SM	Session Manager
TBD	To Be Defined
UC	Use case
UCON	Usage Control
UCP	Usage Control Policy
UCS	Usage Control System
UI	User Interface
US	User story
WoT	Web of Things
WP	Work Package
XACML	eXtensible Access Control Markup Language