D5.4

Final 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: 30/06/2023
Actual submission date: 30/06/2023

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

30/06/2023
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), Otto Waltari (FSEC), Andrea Saracino (CNR), Marco Rasori (CNR), Domenico De Guglielmo (DOMO), Luca Barbato (LUM)

Reviewed by: Marco Tiloca (RISE), Rikard Höglund (RISE)

Revision History

Version	Date	Name	Partner	Section Affected Comments
0.1	15/04/2023	ToC Defined	SEN	All
0.2	30/04/2023	Integration from D5.2	All	All
0.3	06/05/2023	Continuous Integration	LUM, DOMO, SEN	6
0.4	19/05/2023	Validation	LUM, POL, CNR, SEN	7
0.5	10/06/2023	Ready for Review	All	All
0.6	25/06/2023	Reviews Addressed	All	All
1.0	30/06/2023	Ready to Submit	All	All

Executive Summary

In this deliverable, which deprecates deliverable D5.2, the status and implementation aspects of the final version of the SIFIS-Home security architecture is described, together with a brief overview of the test beds and the successful validation results of the system. Since the overall security architecture is the same as in deliverable D5.2 and still consistent with what was defined in deliverable D1.4, the main differences with respect to deliverable D5.2 are primarily in terms of implementation details. The SIFIS-Home security architecture represents a comprehensive implementation of state-of-the-art technologies, developed by multiple partners across Europe with diverse competences and know-how. Every component of the SIFIS-Home framework is examined in detail, discussing the final implementation aspects and integration status, and providing a GitHub link to access the released open-source code.

The final version of the SIFIS-Home testbeds includes emulated and simulated devices for development and testing of the SIFIS-Home solution. Emulated devices are virtual and based on x86 hardware, while simulated devices are Raspberry Pis with an ARM architecture. The testbed integrates a cloud interface enabling external access from outside the Smart Home, and includes live NSSD – Not so smart devices that have been proven convenient to expedite cloud interface development and validation.

The testbed has served as the platform for partners to develop, deploy, and test their applications for the defined SIFIS-Home use cases. The integration and deployment tools employed in this endeavour are cutting-edge, as utilizing Docker for containerization; GitHub as the code repository for version control and collaborative development, build, and integration; and Github ghcr.io in conjunction with Docker Watchtower for automated software deployment.

These tools, combined with the static analysis tools developed by WP2, have allowed for the establishment of a comprehensive CI (Continuous Integration) and CD (Continuous Deployment) process. Furthermore, ghcr.io serves as a repository for third-party applications within the SIFIS-Home Marketplace. As a result, the development of third-party applications can follow and benefit from the same processes as the core SIFIS-Home framework components and modules.

The final SIFIS-Home architecture is now confirmed, it does fulfil the promises of the SIFIS-Home framework as per the project description of work. In particular, all the problems encountered during the design and implementation phase have been properly addressed and overcome. It is rewarding to see real-time data flowing through the deployed testbeds as well as through the WP6 Pilot, while also being able to continuously interact with sensors and actuators and to install applications, and further knowing that applicable analytics suitable to the hardware architecture x86 or ARM are executing on the SIFIS-Home smart devices to keep the system secure.

Table of contents

1 Introduction

This deliverable report provides a comprehensive overview of the implementation of the final version of the SIFIS-Home security architecture, as based on the architecture defined in deliverable D1.4. The report presents the evolution of the architecture throughout the SIFIS-Home project, its impact on the implementation process, and the resolution of key problems that have been encountered. Additionally, it focuses on the architectural components developed by different partners, based on their expertise and contributions in the other Work Packages. Detailed implementation aspects and status updates for each component are discussed, together with links to open-source code available on GitHub. The integration of security solutions from WP3 and analysis methods from WP4 is explored. The report further highlights the general integration strategy using Docker containers and presents the results of the verification and validation of the SIFIS-Home framework.

Overall, the report provides a valuable insight into the implementation of the SIFIS-Home security architecture, offering a detailed account of the architecture's evolution, the individual architectural components, the specific integration efforts, and the overall progress towards the final, integrated security architecture. The reported findings contribute to the field of secure Smart Home systems and serve as a foundation for future research and development in this domain.

2 Actors, devices and the Cyber-Perimeter

2.1 *Analysis of Components and Actors of the SIFIS-Home architecture*

The main components of the SIFIS-Home security architecture are the following:

- *Smart Devices:* These are devices that implement a SIFIS-Home distributed hash table (DHT) that enables a client to set up a Peer-to-Peer (P2P) logical model. General examples of Smart Devices are Smart TVs, Smart Refrigerators, Laptops/Desktops, Family Hubs.
    - *Internet Connected Smart Devices*: This is a subset of the Smart Devices without a SIFIS-Home client App, like smart phones and tablets, but located inside the Smart Home Cyber-Perimeter and equipped with a network interface.
- *Not So Smart Devices (NSSD):* These are constrained devices that cannot be customized by installing third party software or applications. In the SIFIS-Home testbeds, we are using several of these, such as smart sensors and smart lights. The NSSD devices are all connected to the SIFIS-Home network via Smart Devices that implements the DHT.

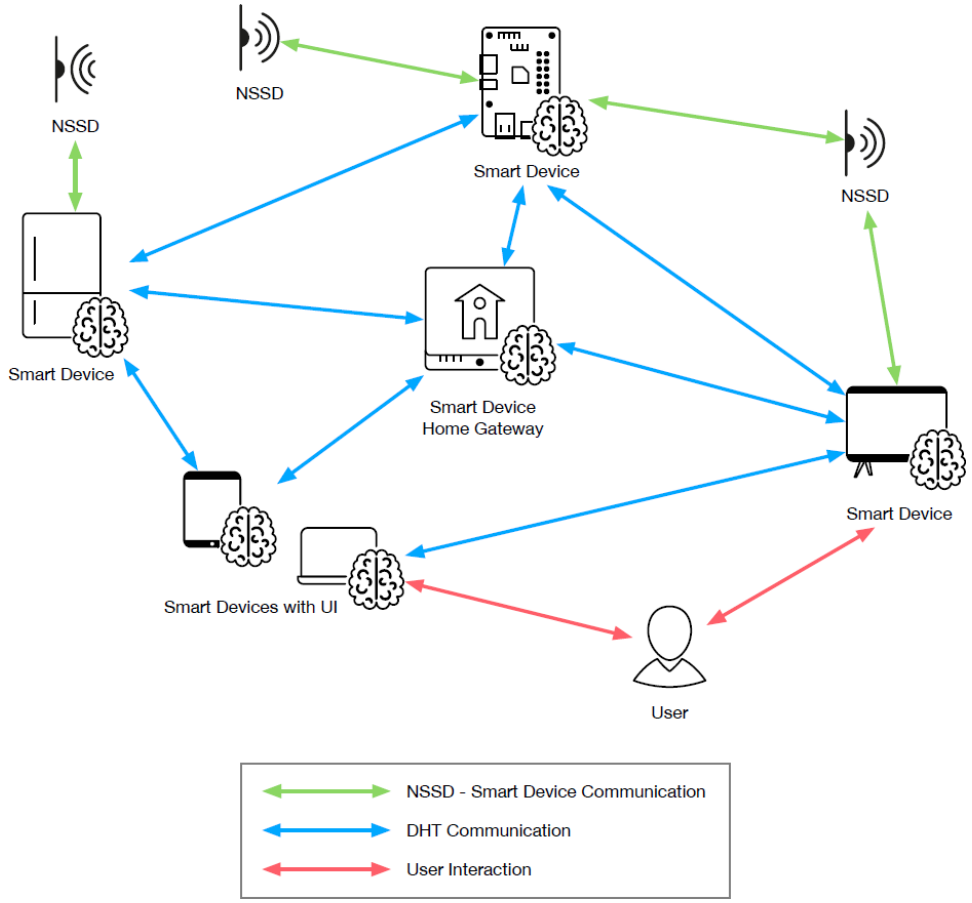


Figure 1: Communication and interaction between components

The actors we have defined for the SIFIS-Home architecture are the following:

- *SIFIS-Home Administrator*: The administrator is a user who is the owner of an instance of the SIFIS-Home architecture.
- *SIFIS-Home Tenant*: The SIFIS-Home tenant is the standard user of the Smart Home system.
- *SIFIS-Home Maintainer:* The maintainer is an entity external to the Smart Home and trusted by the administrator to correctly configure the Smart Home security, privacy and safety policies.
- *SIFIS-Home Tenant with restrictions*: This user is a Smart Home tenant with restrictions on the functionalities they can access.
- *Guest*: A guest is a Smart Home user who is not resident in the Smart Home but is allowed to access and use the premises and some functionalities for a limited amount of time, upon authorization from the administrator or another tenant.
- *External Operator*: The external operator could be a technician, a plumber, a gardener, or a house cleaner, as allowed to access the Smart Home premises for a limited amount of time, with the authorization from a tenant.

In the testbeds, all these roles are represented and have their distinct access rights to the system.

2.2 *Analysis of The Smart Home Cyber-Perimeter*

In order to protect the Smart Home and its users from unintended disclosure of sensitive information,

in WP1 we defined a logical distinction between the outside and inside of the Smart Home, based on what we called the Smart Home *Cyber-Perimeter*.

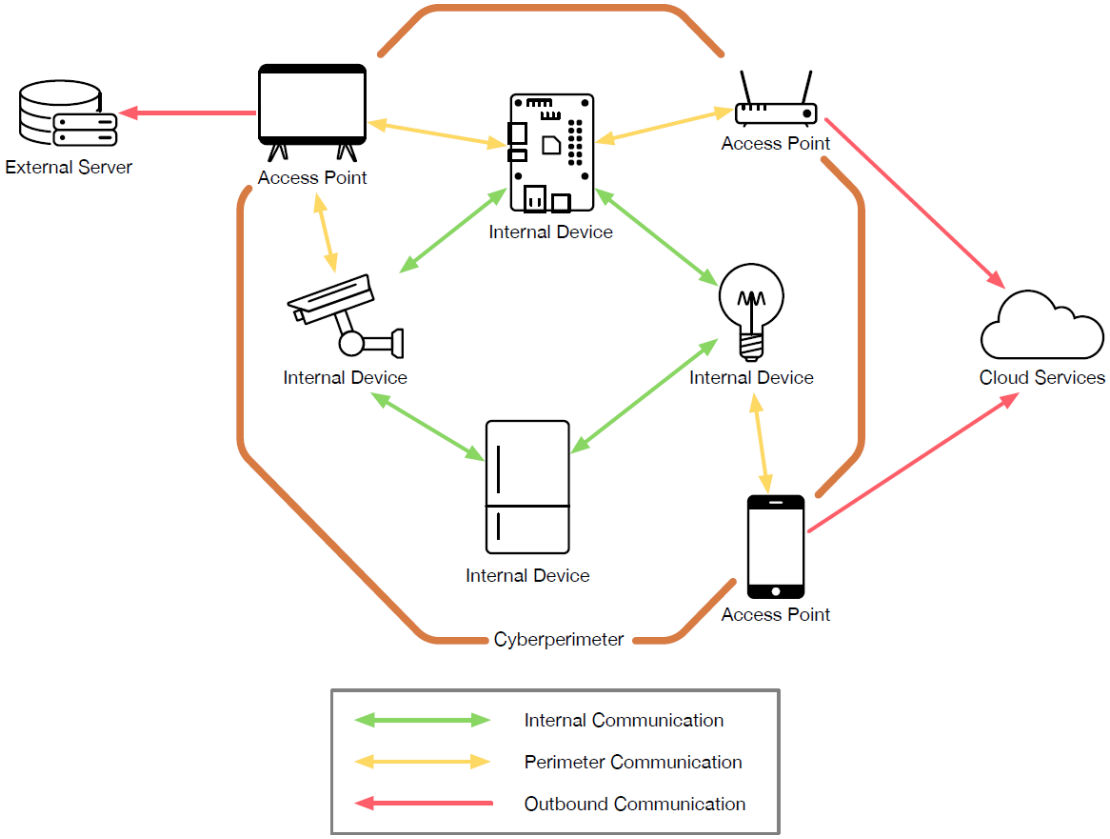


Figure 2: Concept of Smart Home Cyber-Perimeter

The Smart Home Cyber-Perimeter is secured by a firewall that isolates the SIFIS-Home network from the Internet on the outside. Inside the Smart-Home, the SIFIS-Home network analytics constantly monitor the activities of the network to ensure that no malicious actor has managed to penetrate the firewall.

Since a user should also be able to access the SIFIS-Home network from outside the Smart Home, a cloud service is required as well, as part of the SIFIS-Home architecture. The SIFIS-Home network subscribes to a publish / subscribe broker in the cloud service, in order to share the status of the network with the cloud service and to receive commands from users outside the Smart Home via the cloud service. By employing a publish/subscribe communication model, the Smart Home guarantees that all communication sessions originate internally. This approach eliminates the requirement to open any incoming ports on the firewall, thereby ensuring no compromise related to security.

3 Test beds used to verify implementation

The testbeds are built on and around a server, called “Panarea”, which is located at the CNR facilities. To upload new code to the test beds the Continous Integration and Continous Deployment process (chapter 6 Continuous integration and deployment) are used with Ansible [ANSIBLE] deployment scripts are used. Further, the testbeds can be reached directly by the partners via SSH - Secure Shell access – in order to debug their code.

In the Panarea server, the SIFIS-Home cloud interface based on Sensative Yggio is running, as well as several virtual devices each serving as a SIFIS-Home Smart Device based on x86 hardware. These emulated devices comprise the full SIFIS-Home software stack including applicable analytics and network security solutions.

The testbeds also include physical SIFIS-Home Smart Devices based on Raspberry Pi based on ARM hardware. Furthermore, some standard validation NSSD - IoT products based on the LoRaWAN protocol are used with the testbed, in order to collect data for analytics as well as to perform basic software quality control of the cloud interface.

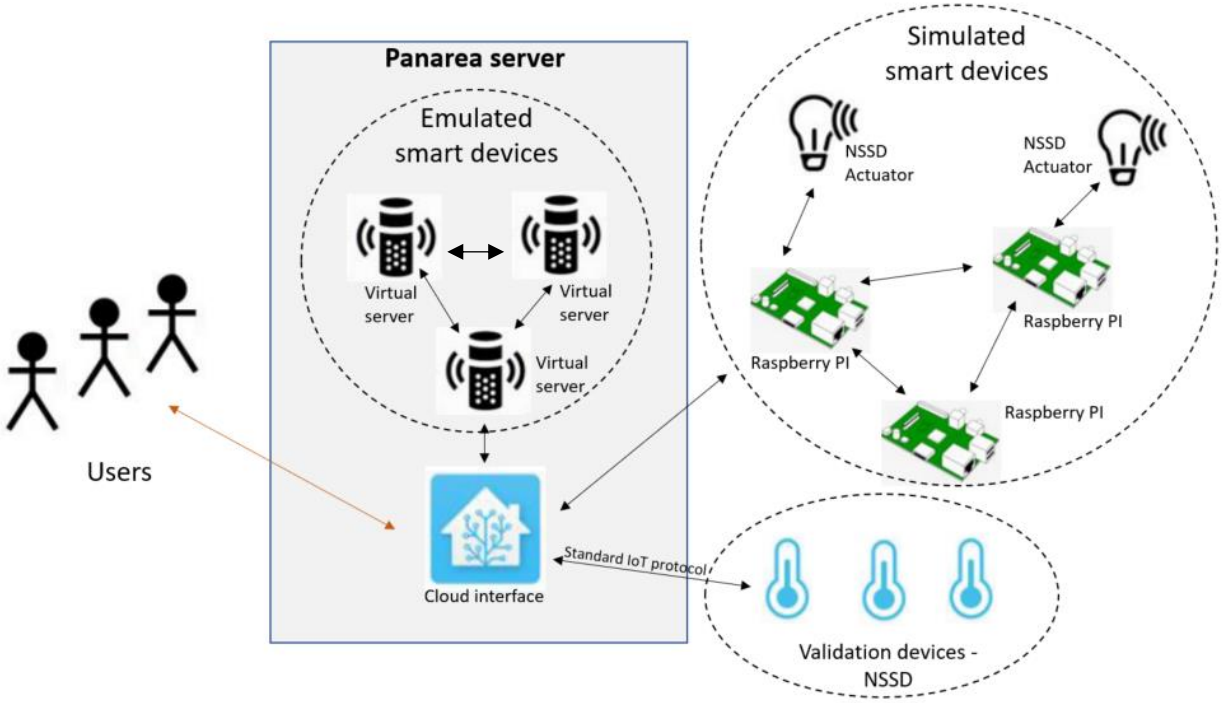


Figure 3: The SIFIS-Home testbed setup

All Smart Devices are connected via the core technology used in SIFIS-Home, namely the DHT (Distributed Hash Table), and relying on a publish/subscribe communication model. The SIFIS-Home devices authenticate and register with the cloud interface, thus creating the SIFIS-Home network. A mobile application installed in an end user's mobile phone is used to directly interface with the SIFIS-Home network from inside the Smart Home, for example to turn on or turn off items without going via the cloud interface. End users outside the Smart Home can access the SIFIS-Home network via the cloud interface from anywhere, for example to check the system status of the Smart Home.

4 Implementation of the security architecture

4.1 *Architecture*

4.1.1 Architecture iterations

The design of the SIFIS-Home framework in WP1 is based on Docker containers with *microservices* design pattern. Each microservice subscribes to the DHT and, depending on its components' type, may

also define a REST API to interface with it. The high-level architecture of the SIFIS-Home framework was originally defined in deliverable D1.3 (see Figure 4 below), and was the basis for the start of the implementation of the security architecture and the test bed design.

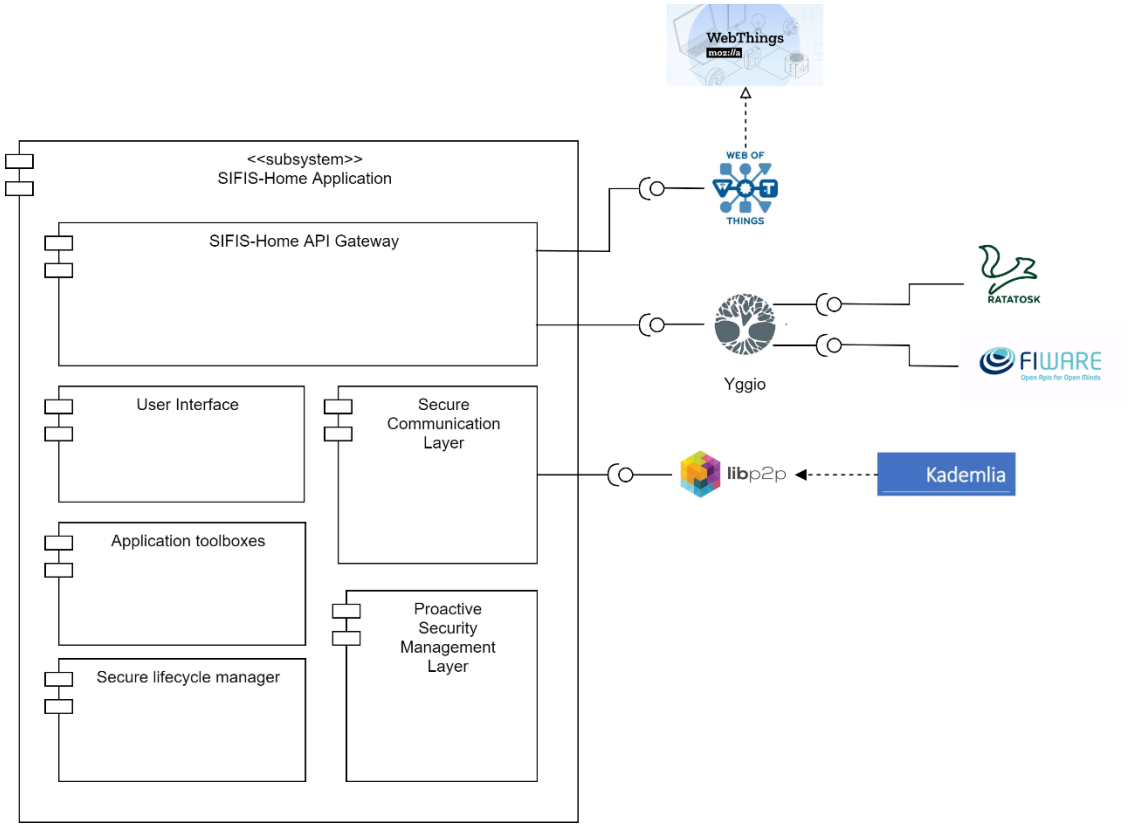


Figure 4 Original high-level architecture of the SIFIS-Home framework as defined in D1.3

The original architecture had some issues and could not be implemented as is. WP1 studied the highlighted problems, and revised the security architecture to become modularized as composed of a few different frameworks. The revised architecture was described in detail in deliverable D1.4, where each framework fulfils a specific need, as summarized below:

- **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 Not So Smart Devices (NSSD).
- **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 site.
- **SIFIS-Home Development Tools:** the set of developer tools that have been developed in the context of WP2. The development tools, that are described in D2.4, are out of scope for this document.

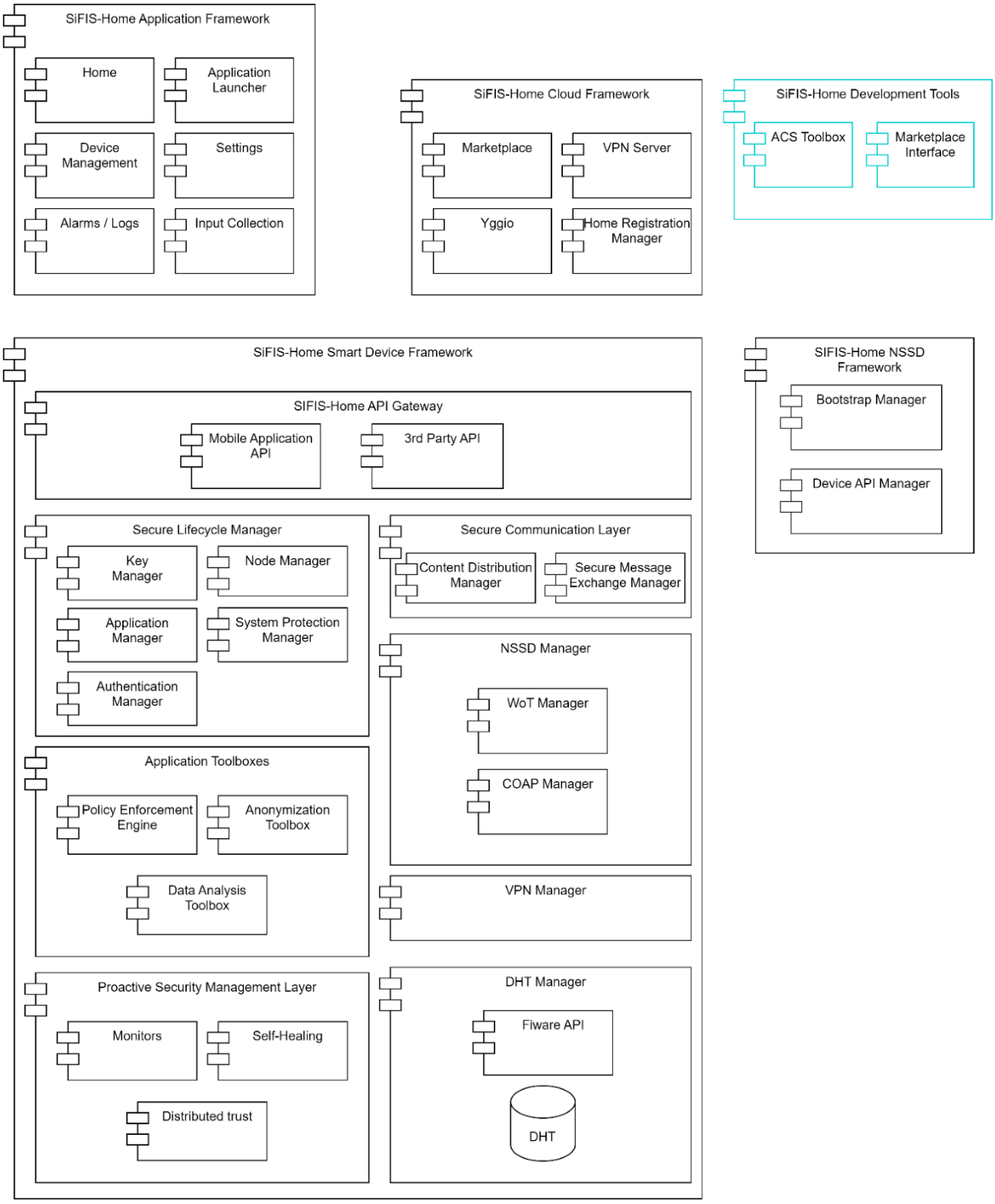


Figure 5: Latest version of the SIFIS-Home architecture as defined in D1.4

The high-level architecture of the SIFIS-Home framework defined in deliverable D1.4 is the basis for the final version of the Security Architecture implementation and the final testbed design. The SIFIS-Home API gateway hosts the Mobile Application API and the API for 3rd party applications, and it is only used by the SIFIS-Home mobile application and 3rd party applications inside the Smart Home. In

the SIFIS-Home architecture, the cloud interface interacts with the DHT via the standard protocol MQTT secured by TLS 1.3. Using MQTT makes it possible to simply yet securely communicate through a firewall, by taking advantage of its publish/subscribe model while employing standard security settings that protect the Cyber Security perimeter.

4.1.2 Authorization and Access management integration

The integration challenges in authorization and access management for the SIFIS-Home framework involve handling devices, users, and access rights within the Smart Home and through the Yggio cloud interface. The main issue is synchronizing user and device states to enable the policy enforcement engine in the Smart Home to regulate user activities on each device.

To address this, the cloud interface utilizes KeyCloak, a trusted open-source component for access and authorization management that meets security requirements. Within the SIFIS-Home network, a Distributed Hash Table (DHT) is employed for device and user management. The crucial integration activity involves synchronizing users by enabling the SIFIS-Home DHT to utilize the authentication mechanism of KeyCloak to obtain a valid JSON web token (JWT). This JWT grants the SIFIS-Home framework access to the cloud interface, facilitating synchronization of users, access rights, and devices.

4.2 *The Application Framework and the Cloud Framework*

The SIFIS-Home Framework and Cloud Framework parts of the architecture incorporate user interface components that cater to the diverse needs of different user types. It consists of two user interface implementations: a mobile phone application, and a comprehensive web-based cloud interface.
The mobile phone application is installed as a dedicated app on users' mobile devices, providing a convenient and user-friendly experience while they are within the Smart Home and have direct access to the SIFIS-Home network. On the other hand, the web-based interface, accessible through the Cloud Interface, allows easy interaction to the SIFIS-Home network both from within the Smart Home and remotely via the Internet.

4.2.1 Application Framework (Mobile Application)

This module handles the standard workflow of the SIFIS-Home framework. The actual UI of the Application Framework is implemented in the mobile application.

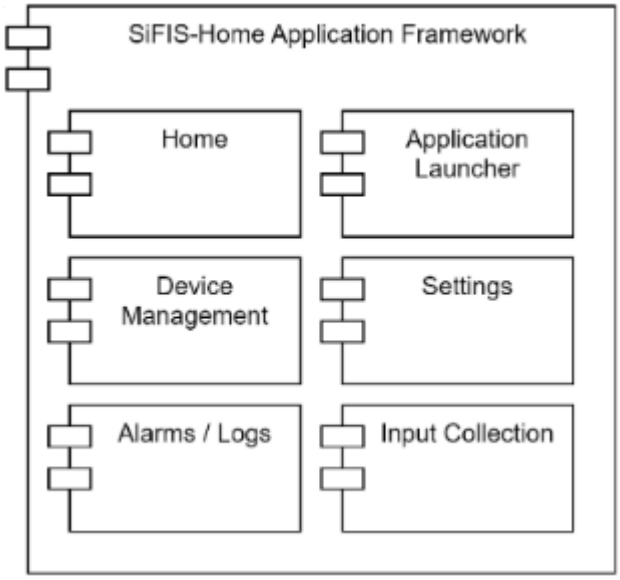


Figure 6 Components of the Application Framework module as defined in deliverable D1.4

- **Home:** the home screen of the UI that the user will see after logging into to the home. From here the users a home in the SIFIS-Home system can access different applications that is used to interact with the SIFIS Home system like the device management, change settings, the application launcher and settings**.**
- **Application launcher:** it provides a graphical UI from which the user can visualize the available applications in the SIFIS-Home system, as well as view the Market Place applications download them, install them, update them if new versions are available, and launch them.
- **Device management:** this component enables the user to configure the devices in the SIFIS-Home network. Each device status can be visualized and if the device is an actuator one can also send commands to it, for example turn on and off a light.
- **Settings:** it provides a 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 management.
- **Alarms / Log:** this component enables the user to extract and view the log stored in the cloud interface by the DHT.
- **Input collection:** it collects input from the user in all the forms allowed by the system, such as voice commands.

Mobile Application UI

The Mobile Application within the SIFIS-Home system serves as a user-friendly interface for end users to interact with the SIFIS-Home framework. It utilizes the "API gateway" in the device framework part of the architecture to establish a direct connection with the SIFIS-Home framework residing within the Smart Home.

Through the Mobile Application, users can easily manage the SIFIS-Home network basic functionalities. This includes listing the installed devices within the home, enabling users to perform various actions on these devices, such as collecting environment measure readings, or controlling actuators, or turning devices on or off.

Additionally, the Mobile Application provides access to system logs for monitoring purposes, as well as the ability to install third-party applications directly into the SIFIS-Home framework, in order to extend the capabilities of the system.

The Mobile Application was written in Javascript and Vue utilizing NativeScript framework and Stackblitz development environment. Mobile Application can run both on Android and iOS with the NativeScript preview application available at https://preview.nativescript.org/.

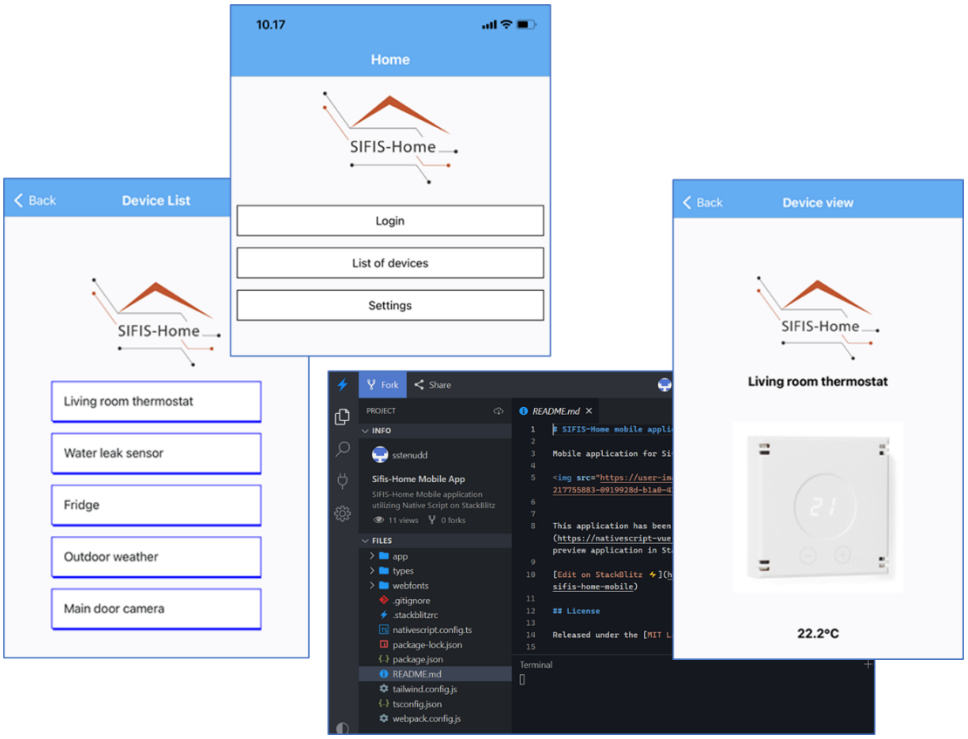


Figure 7 Mobile Application UX

4.2.2 Cloud Framework

This module, see **Errore. L'origine riferimento non è stata trovata.**, includes a set of high-level APIs and is used to access the SIFIS-Home networks and their UI both internally from inside the Smart Home network and externally. The web applications are a major development effort, are based on Node.js JavaScript, and utilize REACT components with Next.JS framework. The web applications execute on top the FIWARE Context Broker Ratatosk that resides inside Yggio from an architectural point of view,

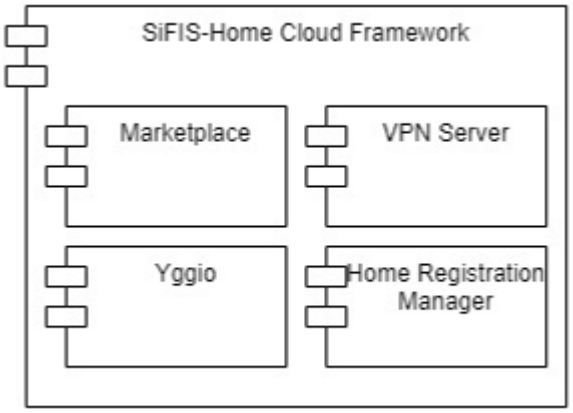


Figure 8: The Cloud Framework

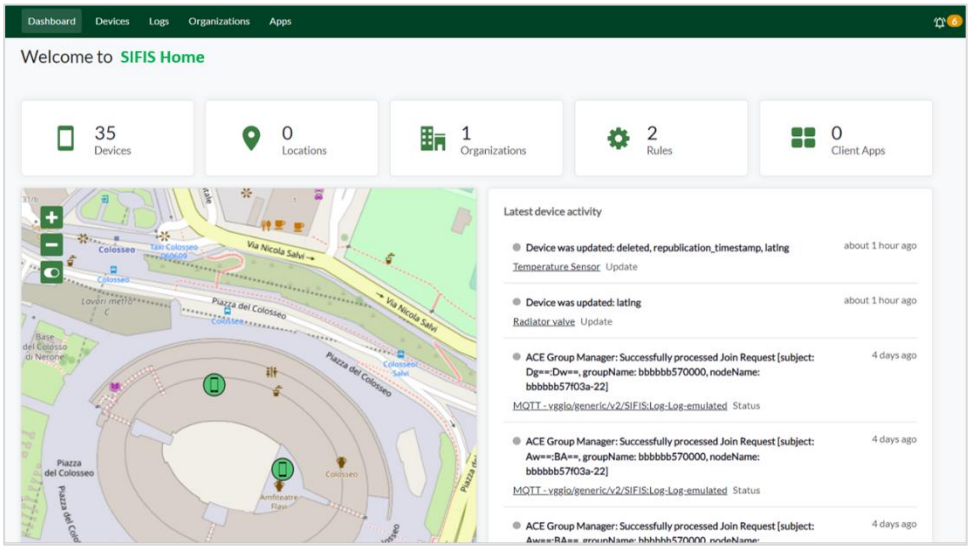


Figure 9: The start page of the Cloud Interface

- **Yggio / Cloud UI** :
  The Sensative horizontal Internet of Things (IoT) integration platform Yggio is used as the backbone of the SIFIS-Home cloud interface. It provides the execution environment that makes Ratatosk FIWARE Context Broker possible to execute, and its API makes it possible to implement the SIFIS-Home overall web interface UI. Ratatosk in turn uses the open source KeyCloak component as a security plug-in for access and authentication that provides a JWT – JSON Webtoken. The JWT is used by the DHT and the Mobile Application to authenticate users and interact with the Ratatosk Rest API.

  The cloud interface can be used to interact with the SIFIS-Home framework both within the Smart Home and outside the Smart Home via a feature-rich web UI.

  FIWARE Context Broker Ratatosk: FIWARE NGSI v2 Ratatosk is a publish/subscribe Context Broker that holds the representation of a system state via FIWARE entities. The Context Broker implements [FIWARE NGSI v2 API] FIWARE NGSI v2 APIs. The FIWARE Context broker API is exclusively used to power the cloud user interface of the SIFIS-Home and is not involved in the actual secure SIFIS-Home network inside the Smart Home, which instead relies on a DHT to create a distributed and robust network without a single point of failure.

Each of the FIWARE entities is described in JSON via a data model. FIWARE defines recommended data models to simplify interoperability between systems at https://github.com/smart-data-models but, if none fits, it is also possible to define one's own data models or use a subset of an existing data model to represent the type of object one wants to describe.

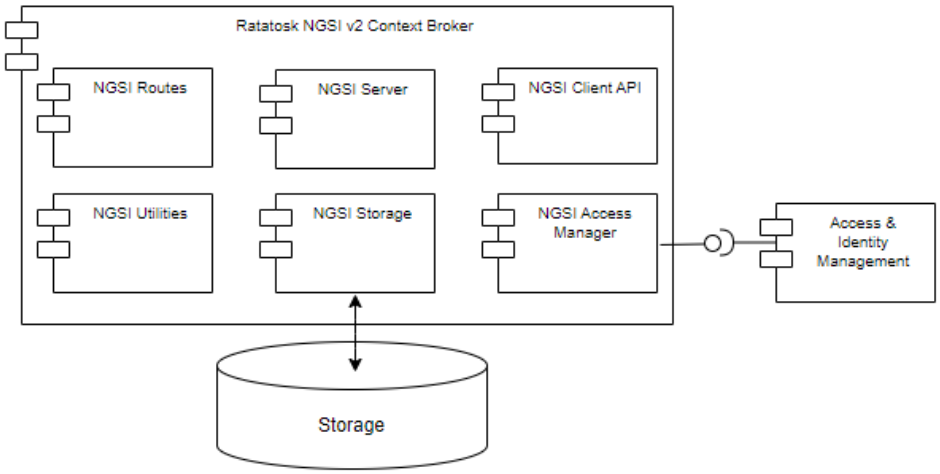


Figure 10 Ratatosk FIWARE Context Broker architecture

An important distinction with Ratatosk is that it is a secure FIWARE Context Broker relying on the KeyCloak open-source component as a security plug-in, and that it always requires a valid authentication token to accept a command. This will be used to make sure that a user in the Smart Home who attempts to perform some actions has the required authorization to do so.

A design aspect of SIFIS-Home is that the cloud interface that implements the FIWARE Context Broker must be able to send a command through a firewall to the DHT based network inside a Smart Home, and then execute some command, like turning on a lamp. The natural FIWARE solution to use NGSI subscriptions would unfortunately not work, since subscription requests are required to be IP addressable and will get blocked by the Smart Home firewall. The solution that we identified was to develop an MQTT-to-DHT bridge, i.e., the devices and analytics in the Smart Home both publish events and subscribe to the FIWARE Context Broker events not via the API and NGSI subscriptions, but rather via a standard, open-source MQTT broker that integrates with Ratatosk. The Ratatosk events will then be triggered either by the user via the UI or by other SIFIS-Home devices. There are more details about the MQTT-to-DHT bridge in chapter 4.3.7 DHT Manager.

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

Cloud Home: This is the main component of the User Interface and is used to launch other applications installed in the Smart Home system. The dashboard in the figure below shows some key metrics of the system, such as the number of devices and installed third party applications.

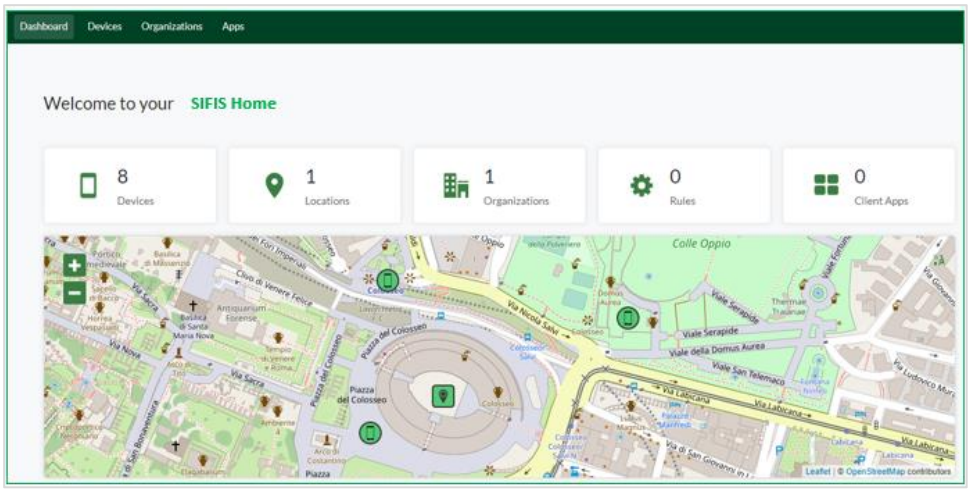


Figure 11 Home screen with a simple dashboard and a map

Cloud Device management: This component enables the configuration of the devices in the SIFIS-Home network. This is a core system component that manages onboarding, configuration, displaying of device status, and other functionalities related to the devices added to the Smart Home system. Depending on what role the logged in user has, different activities like read or write data are allowed.

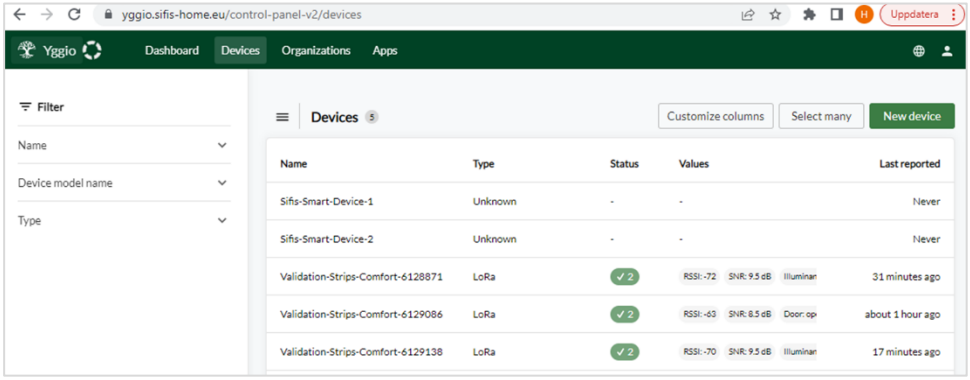


Figure 12 Device Manager with a list of devices

Cloud Settings: This component provides user interfaces for the configuration of the SIFIS-Home infrastructure and most items in the cloud interface, such as devices and analytics. Each visible item is represented by FIWARE entities in the Ratatosk Context broker. This component allows the user to view and edit the values of FIWARE entities.

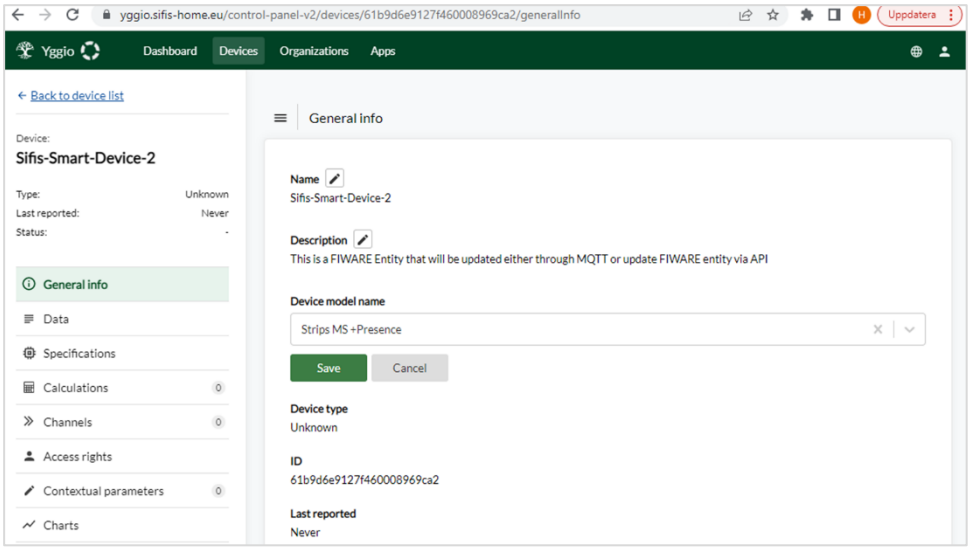


Figure 13 View status and edit settings for a SIFIS-Home Smart Device

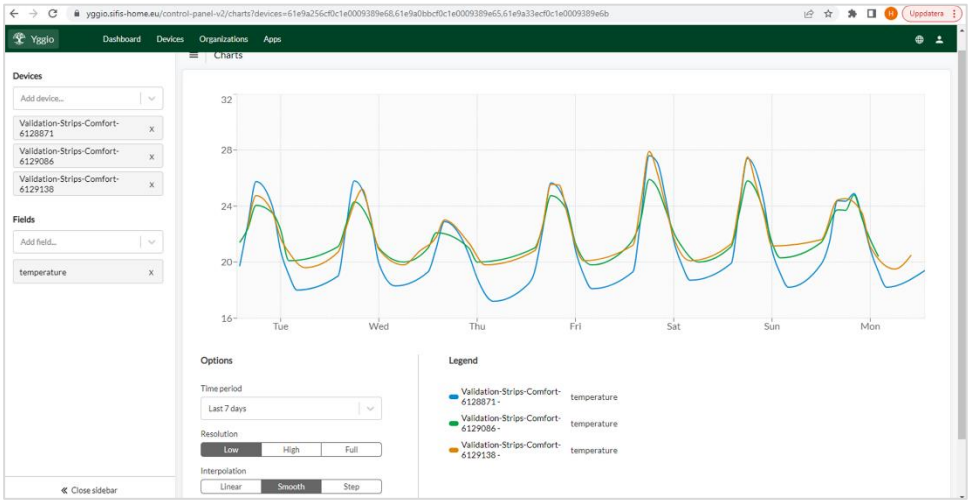


Figure 14 Compare time series data of 3 devices used for validation.

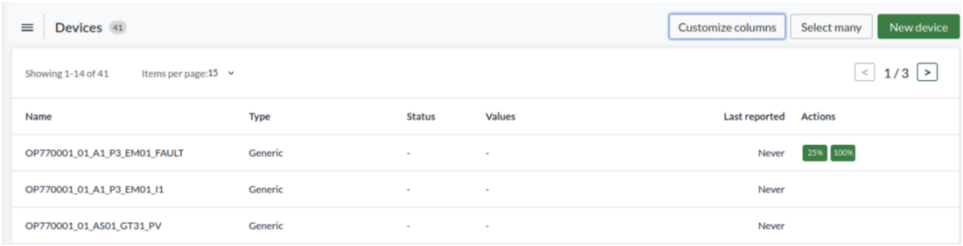


Figure 15 Control actuators via command buttons in the cloud interface

Figure 16 Create custom command buttons

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

- **Cloud Market Place:**
  The Market Place, as depicted in **Errore. L'origine riferimento non è stata trovata.**, is accessible through both the cloud UI and the Mobile Phone Application within the SIFIS-Home system. It enables end users to download applications and install them either directly on their SIFIS-Home network or grant secure access to web applications to the SIFIS Home network via OAuth through the cloud interface. Presently, the Market Place API fully supports OAuth 2.0 security integration, allowing third-party web applications to access the SIFIS-Home network. Installed web applications are regulated by the enforcement of access rights based on their intended users, ensuring secure sharing among household users, thus promoting a safe and controlled environment within the SIFIS-Home ecosystem.

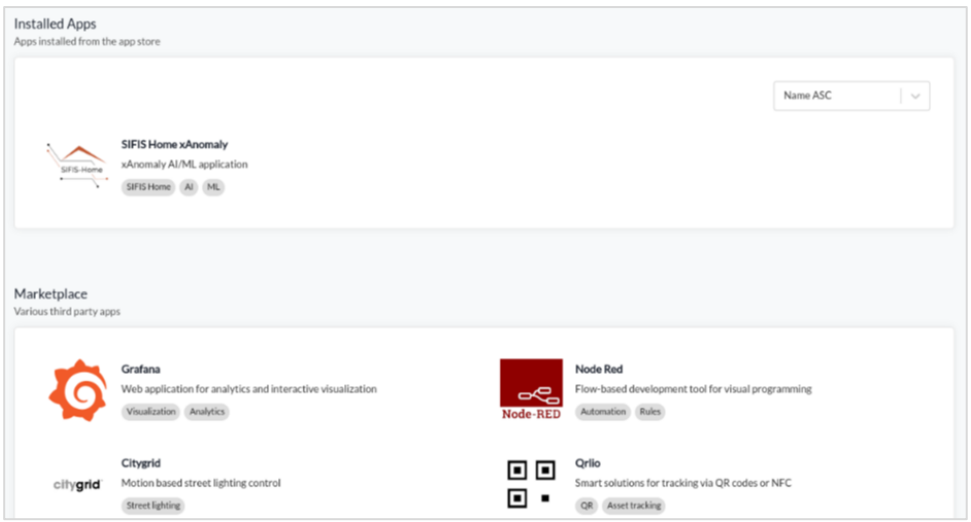


Figure 17 The Market Place

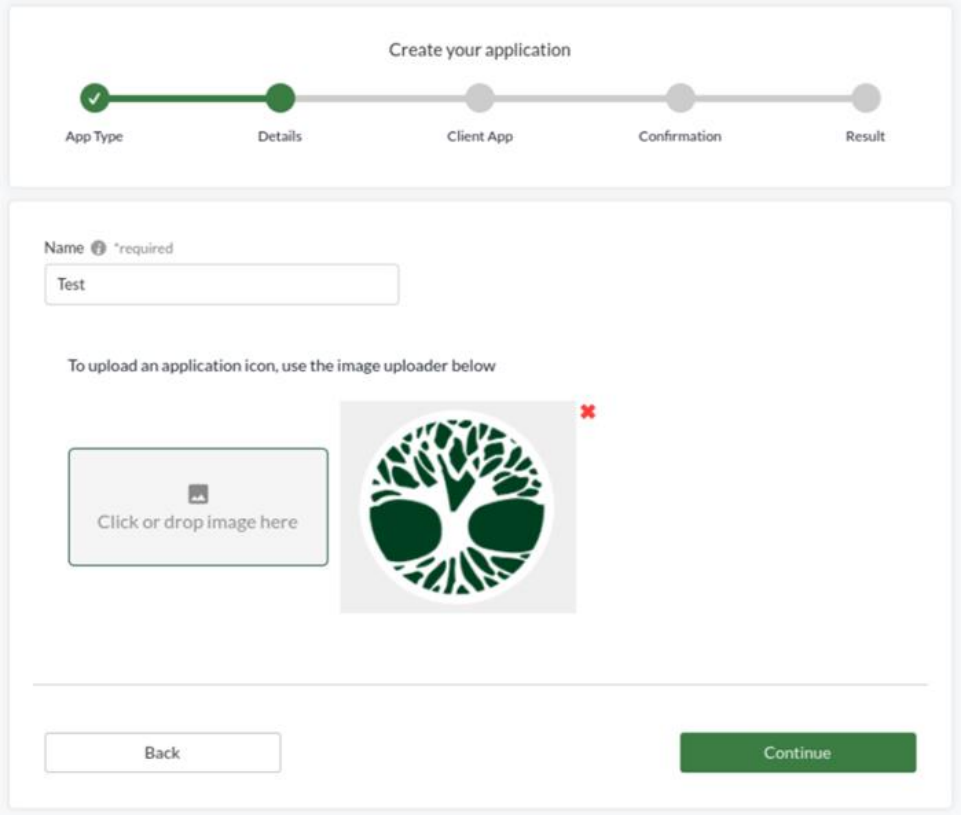


Figure 18 Register an OAuth application in the Market Place

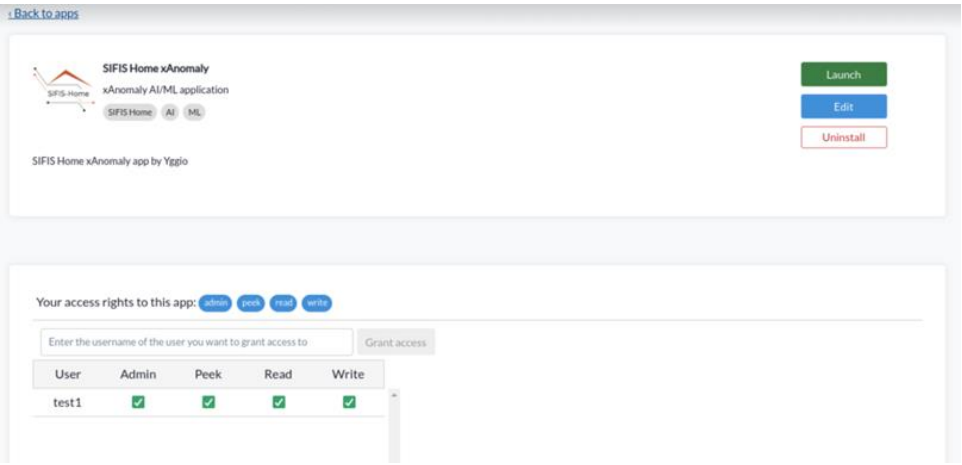


Figure 19 Share installed application with other users

For seamless integration into the SIFIS-Home ecosystem, third-party Market Place applications are required to be uploaded to the SIFIS Home Docker repository. Subsequently, these applications undergo an automatic security scanning process utilizing tools developed by WP2. Once the applications pass the security scanning successfully, they are appropriately labeled and made available in the Market Place. This meticulous procedure ensures that only trusted and secure applications are accessible to users, in the interest of preserving the integrity and safety of the SIFIS-Home environment.

The Market Place implementation is available here as an embedded application inside the

control panel: https://github.com/sifis-home/yggio-components/tree/master/control-panel-v2/src/pages/apps

- **VPN Server:**
  This component was responsible for managing the set of VPN servers that would allow access to SIFIS-Home enabled Smart Homes from a remote site. Since we decided to use a publish/subscribe mechanism via MQTT to access the DHT from the cloud backend this component was no longer needed and will not get implemented.
- **Home Registration Manager:**
  This component provides an interface through which it is possible to create a new SIFIS-Home enabled home. When a new home is created, a household organization with default users is created in the cloud interface dedicated to the house.
- **Alarm / Logs:**
  The functionality of this component encompasses two key features: displaying alarms in the cloud interface and the mobile application and gathering logs pertaining to the operation of the SIFIS-Home infrastructure. Regarding to alarms, it provides a means to highlight any critical issues or events that require attention. Additionally, it facilitates the collection of logs, capturing essential information about the overall functioning of the SIFIS-Home system for analysis and monitoring purposes.

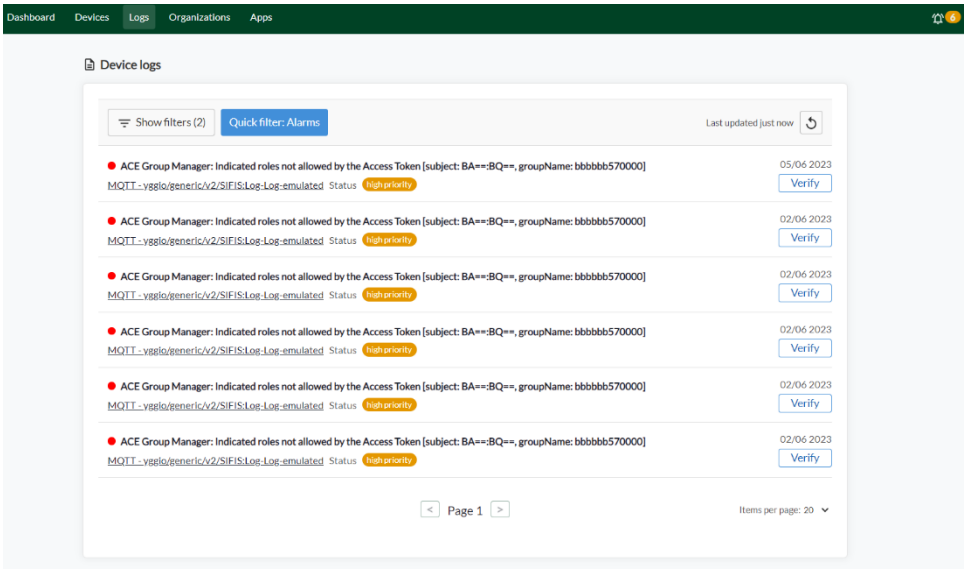


Figure 20 The log in the cloud UI with quick filter on alarms activated.

The alarm and log component receives logs from the Distributed Hash Table (DHT) via MQTT, these logs are received under a JSON key named "log" located at the top level of each message. The "log" key contains crucial information such as type, priority, category, and message, which collectively determine how the log message should be handled and displayed in the user interface (UI).
When incoming logs have a priority level of "severe" or "high," they are categorized as alarms. As a result, the user will be notified via the cloud UI and the mobile application of these alarms and provided with the ability to dismiss them.

{
- Some key values…
- Log : {
  - **type** The type of log. Must be 'info', 'warning', or 'error'.
  - **priority** Must be 'low', 'medium', 'high', 'severe'. Medium, high and severe priority logs can be `verified` by the user. High and severe priority logs are shown as alarms in the control panel.
  - **category** Must be 'status', 'update', 'access', 'command', 'uplink', 'rule' or 'analytics'
  - **message** The log message. Maximum 200 characters.

  }

}

Figure 21 The log JSON structure

Each log within the SIFIS-Home system is equipped with access rights, ensuring that only end users who have access to the device that generated the log can view and dismiss alarms related to that device. This ensures that the responsibility for handling and acknowledging device-specific alarms lies solely with the users who have direct access to the respective device.

4.3 *Smart Device Framework*

4.3.1 Secure Lifecycle Manager

This module, illustrated in **Errore. L'origine riferimento non è stata trovata.**, handles the standard workflow of the SIFIS-Home framework. It consists of the components Key Manager, Application Manager, Authentication Manager, Node Manager, and System Protection Manager.

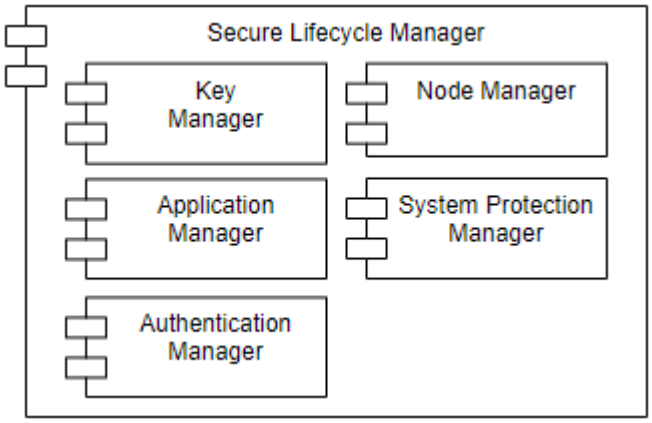


Figure 22 Secure Lifecycle Manager

- **Application Manager**
  The application manager is the component responsible for controlling the deployment, execution and removal of the SIFIS-Home third party applications. This component directly interacts with the SIFIS-Home marketplace triggering the download of applications and managing their deployment on one or more smart devices. The application manager has been implemented as a Python application which interacts with the Docker of the smart device where it is deployed. Third party applications, in fact, come as Docker containers, which will be deployed and run automatically.

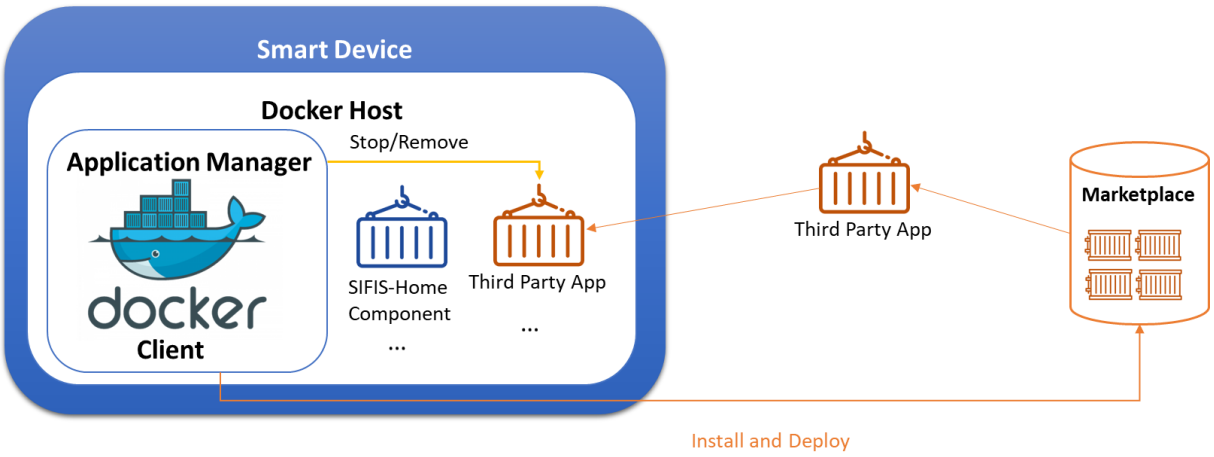


Figure 23 Logical Schema of the Application Manager

Thus, the application manager offers an API to download the Docker from a trusted and authenticated URL belonging to the marketplace. Another API is used to run an application container, which can be triggered automatically after installation or at user's request. Furthermore, the application manager act as controller of the running applications. In fact, when instructed by the System Protection Manager, the Application Manager can stop a running container and eventually remove it from the smart device, if deemed as misbehaving.

- **Node Manager**
  It is the component responsible for ensuring consistency of the list of smart devices in a SIFIS-Home instance, with respect to those participating in the DHT. This component is implemented in the Rust programming language and exploits the LibP2P library as all the other DHT-based components related to the DHT. The component is fundamental for allowing the dynamic joining of nodes and also handles the removal of nodes in different circumstances. The Node Manager implements a majority trust-weighted mechanism to take and enforce decision on events such as joining of new nodes, or removal of existing nodes. Such a voting is needed, since a compromised node might autonomously perform attacks relevant for a DHT system, which endanger the whole SIFIS-Home architecture. In particular, a compromised node might add to the architecture malicious or even fictional nodes in an attempt to perform a sybil attack. Moreover, a compromised node might try to remove a node in an attempt to create a partition, or to damage the system availability. Furthermore, a malicious node might communicate false or wrong readings from the connected NSSDs to trigger unwanted reactions. To avoid such issues, with the lack of a root of trust, voting is used to find agreement on these topics and to filter malicious interactions coming from compromised nodes. The node manager enforces the decisions of voting procedures by acting directly on the DHT, to enable a new node to be part of the network, or by triggering a rekeying to forcefully remove compromised nodes from any interaction. As anticipated, the voting procedure is weighted on a trust score, which is defined by the Trust Manager.

- **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. The component also interacts with the Alarm and Log component to provide information on identified threats or to notify alarms. The system protection manager includes a set of rules that define the behavior to be applied on the response of each analytic which is relevant for intrusion

detection tasks. In particular, all analytics related to anomaly detection, multi-level intrusion detection and network intrusion detection, will report either the identifier of a smart device or NSSD, or the identifier of an application, which is misbehaving. The System Protection Manager will either raise an alarm, communicating with the Notification Manager, or it shall command the force stop of an application through the application manager, or it will force the removal of a device from the system through the Node Manager.

- **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 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 codebase collecting these implementations is accessible at [WP3-CODEBASE][ACE-UCS][ACE-ENTITIES], as available for use, integration, and testing within the SIFIS-Home project, and especially used in the WP5 testbed.

  [CALIFORNIUM] https://github.com/eclipse/californium
  [WP3-CODEBASE] https://github.com/sifis-home/wp3-solutions
  [ACE-UCS] https://github.com/sifis-home/ace-ucs
  [ACE-ENTITIES] https://github.com/sifis-home/ace-entities

  - **OSCORE profile of the ACE framework**, as documented in Section 5.1 of deliverable D3.3. 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 deliverable D3.3. 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 6.2 of deliverable D3.3, including the specific workflow optimization 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
  - **Usage Control integration within the ACE AS and notification of revoked access credentials**, as documented in Sections 5.2, 5.3, and 5.4 of deliverable D3.3. This security solution pertains to the "Authentication Manager" component of the "Secure Lifecycle Manager" module. The implementation is available at https://bitbucket.org/marco-rasori-iit/ace-java/src/sifis-home/, and is also mirrored at https://github.com/sifis-home/ace-ucs. A related codebase consisting of an ACE AS, an RS, and a Client is available at https://github.com/sifis-home/ace-entities

4.3.2 Secure Communication Layer

This module, shown in **Errore. L'origine riferimento non è stata trovata.**, handles the standard workflow of the SIFIS-Home framework. It consists of the components Secure Message Exchange Manager and Content Distribution Manager.

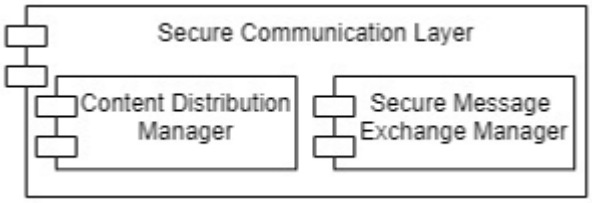


Figure 24 Secure Communication Layer

- **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 codebase collecting these implementations is accessible at [WP3-CODEBASE][ACE-UCS][ACE-ENTITIES], as available for use, integration and testing within the SIFIS-Home project and especially used in the WP5 testbed.

  [CALIFORNIUM] https://github.com/eclipse/californium
  [WP3-CODEBASE] https://github.com/sifis-home/wp3-solutions
  [ACE-UCS] https://github.com/sifis-home/ace-ucs
  [ACE-ENTITIES] https://github.com/sifis-home/ace-entities

  - **Group OSCORE**, as documented in Section 4.1 of deliverable D3.3. 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 deliverable D3.3. 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 deliverable D3.3. 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 6.2 of deliverable D3.3, including the specific workflow optimization 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
  - **Usage Control integration within the ACE AS and notification of revoked access credentials**, as documented in Sections 5.2, 5.3, and 5.4 of deliverable D3.3. 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://bitbucket.org/marco-rasori-iit/ace-java/src/sifis-home/, and is also mirrored at https://github.com/sifis-home/ace-ucs. A related codebase consisting of an ACE AS, an RS, and a Client is available at https://github.com/sifis-home/ace-entities

4.3.3 Proactive Security Management Layer

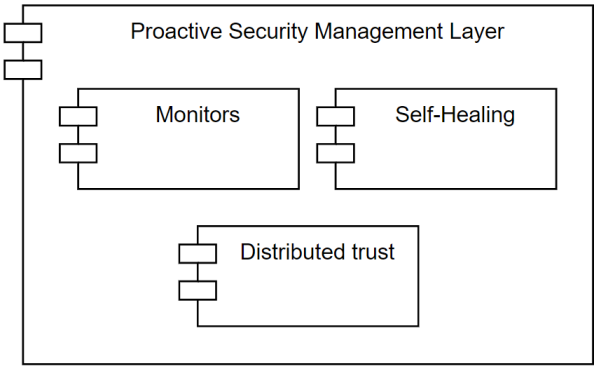


Figure 25 Proactive Security Management Layer

This module, shown in **Errore. L'origine riferimento non è stata trovata.**, is responsible for proactively maintaining the security requirements of the SIFIS-Home infrastructure fulfilled. Proactivity implies taking preemptive measures before an incident occurs. The following proactive security measures have been developed as part of the SIFIS-Home project, and are part of the security architecture:

- **Monitors**
  This component is a collection of services that log specific events at the following, different levels:
  - **DHT Monitor:** Implemented through the libP2P library, it logs the number and type of operations performed on the DHT.
  - **Application Monitor:** This service in-lines security critical APIs to log and control the event in which they are invoked by 3rd party applications. For example, when a SIFIS-Home Developer invokes an API that might perform actions related to security, privacy or safety a log event is triggered and saved. This function is paired with the Policy Enforcement. Application monitor can also be in-lined in Linux-based 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 by capturing packets via iptables. It is intended to run on central networking devices, such as routers through which other devices communicate both inside the SIFIS-Home network and to the outer Internet.
  - **SysCall Monitor:** It collects system call events through a REST API and conveys them for further assessment to the responsible analytic in the Data Analytic Toolbox component of the Application Toolboxes module.
- **Distributed Trust**

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

The distributed trust component continuously assigns to each smart device a trust score and manages distributed decisions under biased voting. This ensures that the devices with a trust score below a certain threshold cannot participate in further voting procedures, until their trust level is assessed as restored. The component is implemented in Java 8 and it is based on standard libraries. A porting to Rust has been made and at the implementation level, the distributed trust management has been integrated as a functional module of the Node Manager.

The Distributed Trust component supports the node manager to enable the execution a set of workflows, such as the joining or leaving of a smart device, which trigger a voting protocol among all the other smart devices. Each smart device has a *reputation score*, which ranges between 0 and 1. The starting value of the reputation score is of 0.5. If the value is over the threshold of 0.5, then the vote of that node is considered in the voting process, discarded otherwise. More in details, the final decision on each voting procedure is given by the sum of the votes for a specific decision (e.g. "node 3 is misbehaving, *yes* or *not*"), weighted by the reputation score for the smart node casting the vote. Hence, a higher reputation implies a major impact on the vote. However, if the reputation is lower than 0.5, the vote is not considered in the decision. The reputation is updated at each voting procedure for each node, depending on the agreement or not with the taken decision.

Algorithm Updating ρ_l

$\rho_l = b_l - d_l - u_l$
$b_l + d_l + u_l = 1$
for all $u \in U^j$ **do**
  if u vote is aligned **then**
    $b_l = b_l + \Delta_b$
    $u_l = u_l - \frac{\Delta_b}{2}$
    $d_l = d_l - \frac{\Delta_b}{2}$
  else
    if u vote is not verified **then**
      $u_l = u_l + \Delta_u$
      $b_l = b_l - \Delta_u$
    end if
  else
    if u vote is not aligned **then**
      $d_l = d_l + \Delta_d$
      $b_l = b_l - \frac{\Delta_d}{2}$
      $u_l = u_l - \frac{\Delta_d}{2}$
    end if
  end if
end for

Figure 26 Reputation update algorithm

The reputation update algorithm is reported in Figure 26 Reputation update algorithm, where u are the smart nodes, ρ_l is the reputation value and b_l, d_l, u_l are respectively the level of belief, disbelief, and uncertainty that define the reputation of each node at each vote. The belief component is increased when the vote of a node is in agreement with the collective decision. The disbelief is instead decreased when the vote is in contrast with the final decision. The uncertainty component is increased when a node does not vote. Even if the vote of nodes under

threshold is not considered to take the collective decision, the reputation value of nodes below threshold is still updated according to the reported algorithm. Thus, a node below threshold can increase its reputation value through good behaviour. Still, nodes with a critically low reputation level can be removed from the system by the Node Manager component. The increment or decrement of each component is a customizable parameter, depending on the desired effect of each correct, wrong or missing vote.

- **Self Healing**
  The self-healing component has two functionalities, which are executed once a device has been identified as compromised and made unable to interact with the SIFIS-Home architecture, thanks to the Node Manager, which performs a rekeying. The first functionality triggers a reconfiguration of the SIFIS-Home architecture to avoid network partitions and reconnects NSSDs which remained isolated after the removal of the responsible Smart Device they were associated with.
  The second functionality automatically runs a set of routines on Smart Devices which have been identified as misbehaving and therefore removed from the DHT. Depending on the identified misbehaviour, a number of actions can be taken, e.g., an application deemed as malicious can be removed forcefully, and then, after a satisfactory system analysis, the device can be re-integrated in the network. As an alternative, the self-healing can trigger a full reset of a Smart Device, upon request from the System Protection Manager component of the Secure Lifecycle Manager module.

4.3.4 Application Toolboxes

This component, shown in **Errore. L'origine riferimento non è stata trovata.**, collects related and interconnected sub-components that are all services inside the SIFIS-Home infrastructure.

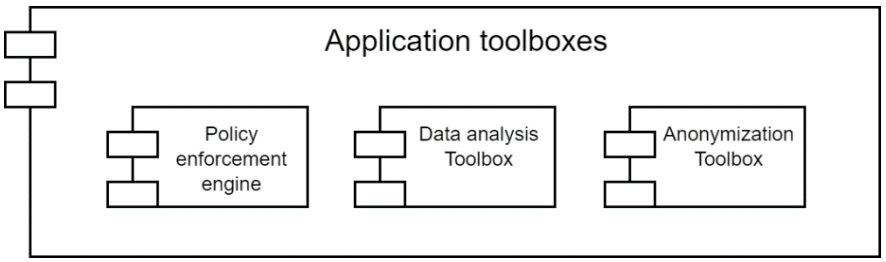


Figure 27 The application toolboxes

- **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, in order to analyze voice and gesture commands, provide advanced smart services to the Smart Home users, and detect misbehavior, intrusions, and failures. This component is activated by other components of the framework that 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 all such topics. When a component of the SIFIS-Home Framework needs the execution of an analytic, it simply publishes a message on the topic related

to that analytic, 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 multiple times, etc.). As shown in Figure TBD, the interaction between the Data Analysis Toolbox and the DHT starts with a message published to a specific topic associated with the analytic to be performed on the data shared in the 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. Data inputs include an audio file and privacy parameters for privacy-aware speech recognition, or a list of temperature values for device anomaly detection analytics. This JSON message is published via HTTP REST or a WebSocket-based API.

```
ws_req = {
        "RequestPostTopicUUID": {
            "topic_name": "SIFIS:Privacy_Aware_Device_Anomaly_Detection",
            "topic_uuid": "Anomaly_Detection",
            "value": {
                "description": "Device Anomaly Detection",
                "requestor_id": str(requestor_id),
                "requestor_type": str(requestor_type),
                "request_id": str(request_id),
                "connected": True,
                "Data Type": "List",
                "Temperatures": temp
            }
        }
    }
ws.send(json.dumps(ws_req))
```

Figure 28 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. Once a published message has been received by the Data Analysis Toolbox, the data analysis service related to the specified topic is called and executed on the provided input as shown in the code in **Errore. L'origine riferimento non è stata trovata.**.

```
if json_message["topic_name"] in SUBSCRIBED_TOPICS:
    if json_message["topic_name"] == "SIFIS:Publish_Alarms_Request":

        Address = json_message["value"]["Address"]
        Port = json_message["value"]["Port"]
        within_time = json_message["value"]["Within Time"]
        device = json_message["value"]["Device name"]

        if Address is not None and Port is not None and within_time is not None:
            print('Time is not None')
            (success, message) = netspot_alarm_check(Address,Port,within_time)
        elif Address is not None and Port is not None and within_time is None:
            print('Time is None')
            (success, message) = netspot_alarm_check(Address,Port)
        else:
            print('Error, no variables were passed')

        if success:
            if message is None:
                return 0
            # We have an alarm message. Let us create DHT message from it.
            ws_req = {
                    "RequestPostTopicUUID": {
                    "topic_name": "SIFIS:Netspot_Control_Results",
                    "topic_uuid": "AlarmResult",
                    "value": {
                        "description": "Netspot alarms check results",
                        "Device": device,
                        "Statistic": message['stat'],
                        "Status": message['status'],
                        "Probability": message['probability'],
                        "Time": message['time']
                    }
                }
            }
```

Figure 29 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 offers a set of data preparation functions that also clean missing and noisy data, convert data into an appropriate and unified format, normalize data according to a given range of values, and reduce data dimensions by selecting or combining variables into features.

Pre-processing functions are provided by the Data Analysis Toolbox based on the analytics being invoked. The analytics integrated within the Data Analysis Toolbox have the required pre-processing steps integrated within the analytics services.

The Data Analysis Toolbox embeds a set of analytic engines that actually implement the provided analytics. Since each analytic engine requires its own execution environment (e.g., based on a specific library or a specific version of a library, on specific or customized utilities, etc.) and the environment of one engine could be not compatible with the environment of other ones, we have isolated such environments by using the container virtualization technology. Hence, each analytic engine is deployed in distinct containers, using the Docker technology, and the Data Analysis Toolbox invokes each such engine through the HTTP REST or the WebSocket-based interface that each engine exposes.

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 service 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 by including the analytics Requestor ID and Request ID parameters, again in a JSON format as shown in **Errore. L'origine riferimento non è stata trovata.**.

```
ws_req_final = {
                "RequestPostTopicUUID": {
                "topic_name": "SIFIS:Privacy_Aware_Speech_Recognition_Results",
                "topic_uuid": "Speech_Recognition_Results",
                "value": {
                    "description": "Speech Recognition Results",
                    "requestor_id": str(requestor_id),
                    "requestor_type": str(requestor_type),
                    "request_id": str(request_id),
                    "analyzer_id": str(analyzer_id),
                    "analysis_id": str(analysis_id),
                    "connected": True,
                    "filename": filename,
                    "Private_Text": Private_Text,
                    "Private_Audio": "Private_Audio.mp3",
                    "private_audio_path": str(os.getcwd())
                }
            }
        }
```

Figure 30 The data analysis result is returned to the requester component.

- **Anonymization Toolbox**
  The anonymization toolbox contains software tools that preserve privacy of data before, during, and after the analysis of such data. 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. The Anonymization Toolbox uses several privacy mechanisms listed below, according to the data type and the analysis function to be performed on such data:

  1. **Time Series Data Analysis:**

     Input data: autoencoders are a type of neural networks used for anomaly detection and data reconstruction, so they are used to anonymize original data and share only reconstructed data [ZP17]. 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 analysed by keeping only the most important features, thus improving performance and privacy. Instead, Differential Privacy is used for privacy preservation and for 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, including spoken information and speaker's voice which could reveal his identity. **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 value over time. For instance, the number of people in a room is an example of mutable attribute. A request to be authorized to perform an operation (access request) is evaluated against a policy, and, if the policy is satisfied, access is granted. If mutable attributes are included in the policy, the policy is re-evaluated as soon as any of the mutable attributes' value changes. If the re-evaluation produces a negative decision (Deny), the right to access, which was previously authorized, is revoked.

  A Java implementation of the Usage Control System (UCS) has been exploited to implement the Policy Enforcement Engine. A codebase collecting this implementation is accessible at [USAGE-CONTROL], as available for use, integration and testing within the SIFIS-Home project and especially used in the WP5 testbed. This implementation is an extension of the XACML reference architecture [XACML, 2017] to enforce 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 Sections 3.9 and 5.3 of D3.4.

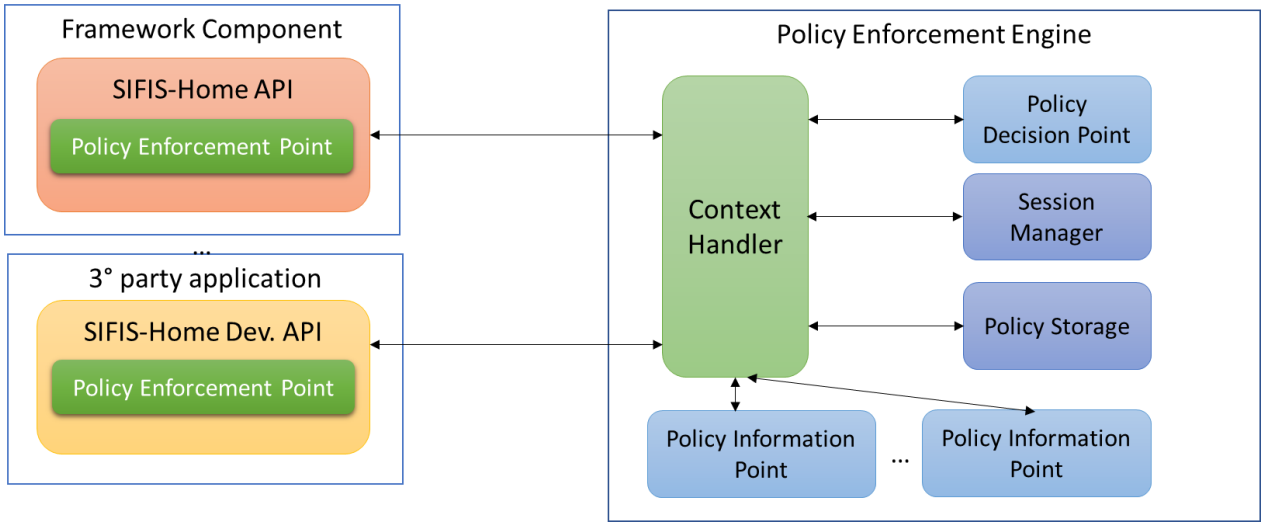


Figure 31: Policy Enforcement Engine

[USAGE-CONTROL] https://github.com/sifis-home/usage-control

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 implemented in Java and in Python languages, and it is used 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 communication between PEP and UCS leverages the DHT. In particular, the PEP publishes an XACML request on the topic the UCS is subscribed to, while the UCS publishes the response on the topic the PEP is subscribed to. The connection to the DHT is performed through web sockets.
Internally, 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 32 shows an XACML request coming from the PEP after the manipulation performed by the UCS.

```
<?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 32 An "enriched" XACML request. The original XACML coming from the PEP has a white background color, 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. Those are composed of three different sections, i.e., pre-, on- and post-sections, which are evaluated separately and at different times. **Errore. L'origine riferimento non è stata trovata.** 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 33 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 it to 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 positive (Permit).

A PIP monitors a mutable attribute, which is stored at an Attribute Manager (AM). In our implementation, AMs can be a database or a file, and PIPs 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 right to access must be revoked. In such a case, the CH sends a revocation message to the PEP, which interrupts the access to the resource and notifies the CH.

4.3.5 API Gateway

This module, as shown in **Errore. L'origine riferimento non è stata trovata.**, provides the API's used by applications executed on the SIFIS-Home Smart Devices.

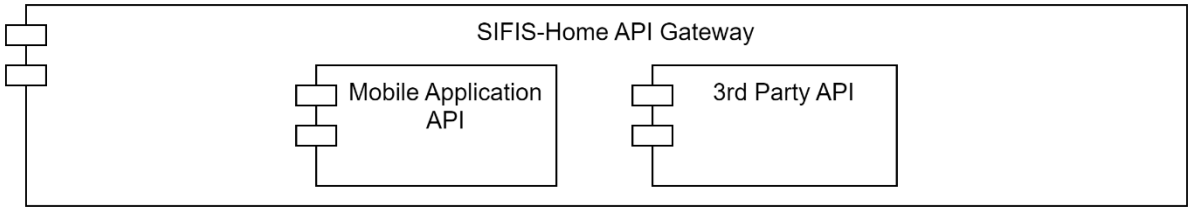


Figure 34 The API Gateway with the mobile and 3rd party API

- **Mobile Application API**

  This component allows the SIFIS-Home mobile application to interact with the Smart Home and to initialize new SIFIS-Home-compliant Smart Devices. The smart device contains a Mobile Application API which communicates with the smart phone mobile application.

  When the Smart Device is started for the first time, it starts in initialization mode. In initialization mode, the device presents itself as a Wi-Fi access point which allows the smart phone mobile application to communicate with the Mobile Application API on the smart device.

  The mobile application requires an access token to use Mobile Application API securely. The access token consists of pre-generated 256 random bits. This token is delivered as a QR code with the Smart Device. All API functions require this access token except the device info which

is available without the token. Device info has product name and UUID.

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

The Smart Device Mobile Application API provides the following functionalities:

- Read and set device configuration:
  - DHT shared key
  - User defined device name
- Retrieving device status:
  - CPU usage
  - Memory usage
  - Disk space usage
  - Uptime
  - Load average
- Sending commands:
  - Factory reset
  - Restart device
  - Shut down device

Endpoints

- Device information and configuration:
  - [GET] device/status
  - [GET, PUT] device/configuration
- Commands:
  - [POST] command/factory_reset
  - [POST] command/restart
  - [POST] command/shutdown

Smart Device initialization

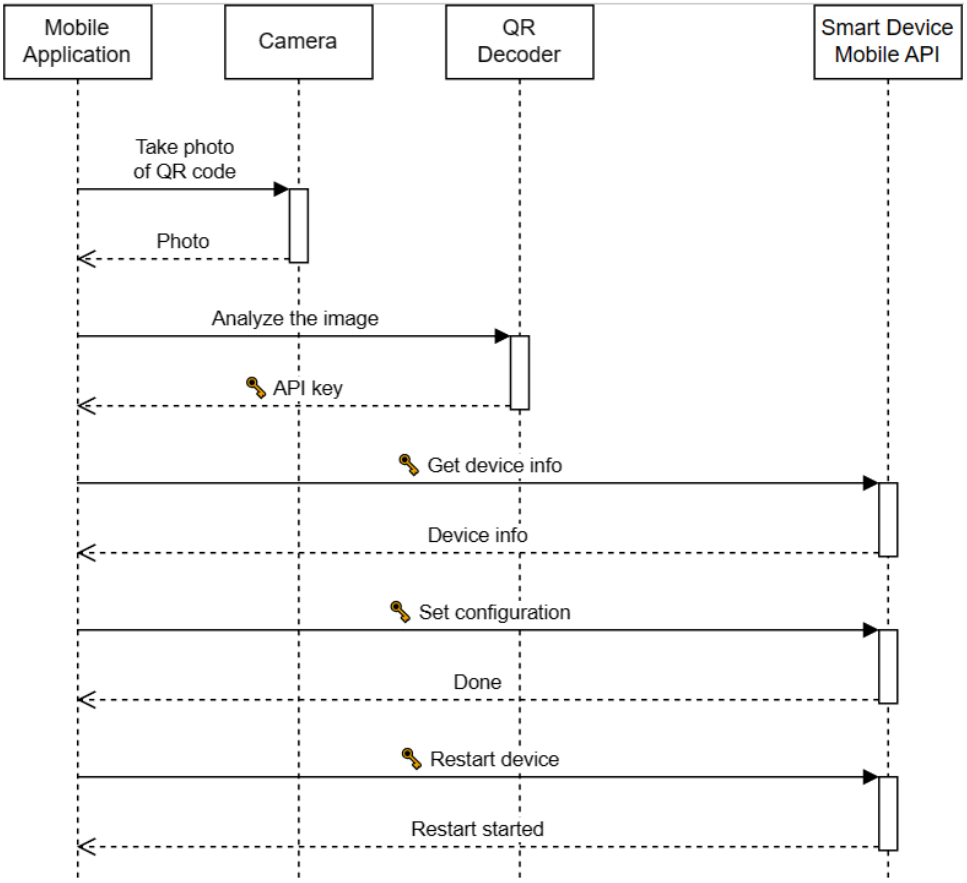


Figure 35 Smart Device initialization

The mobile application defines the Smart Device's configuration.The configuration file is stored on the device in the `/opt/sifis-home/config.json` file. The boot process is shown in figure 24 Smart Device Boot.

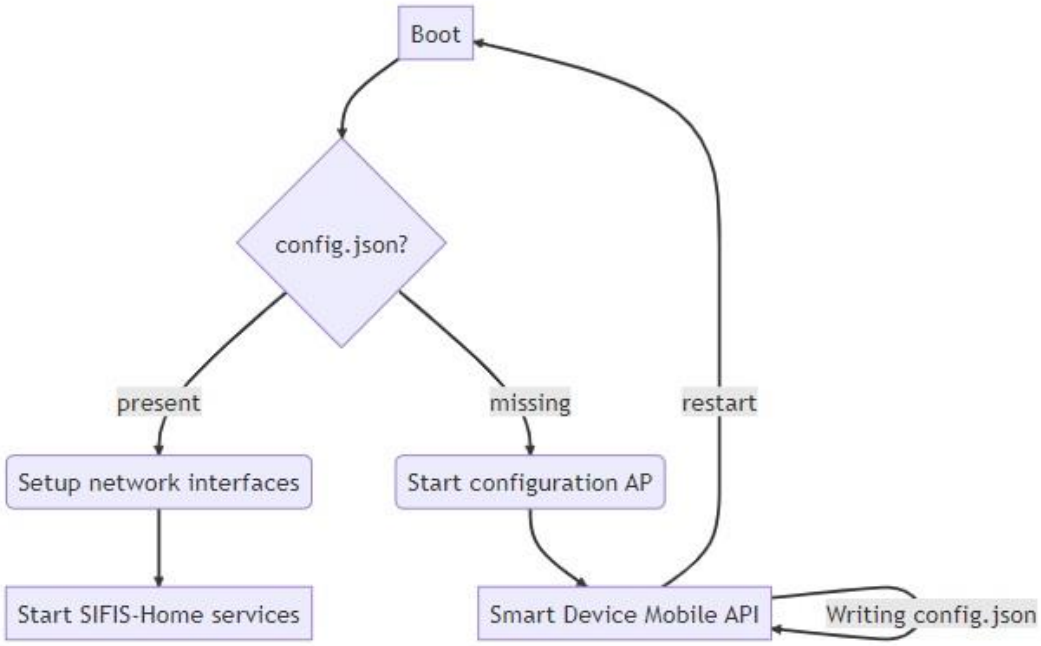


Figure 36 Smart Device Boot

All SIFIS-Home services running on the Smart Device can read this file, but only operations requested through the Mobile Application API can write it. Other SIFIS-Home services are not started on the device if the file is missing.

The system service manager is probably the most common solution for managing services on Linux systems. into. System target refers to a different states where Linux system can boot into. Each target defines a set of services and resources that are available in that state.

Figure 37 below shows a simplified graph of default boot targets and services to the multi-user target. The multi-user target has everything running except the graphical user interface. It was chosen as a state for the system because SIFIS-Home devices do not have embedded display.

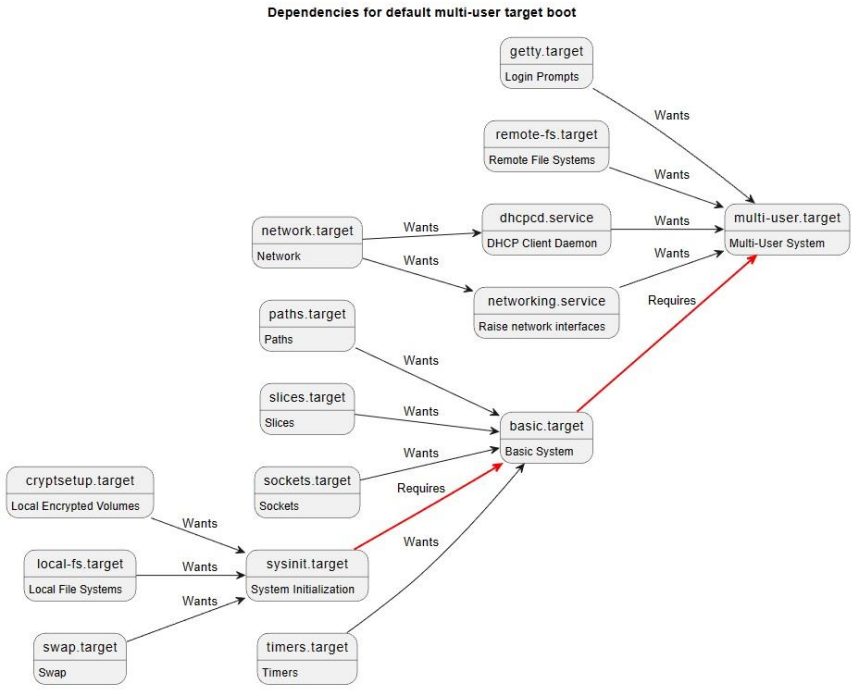


Figure 37 Default Boot Targets

There are two targets for the SIFIS-Home Smart Device. The selection of which target is active is based on whether the config.json file exists. Services that are only needed for configuration are installed under sifis-config.target, and services for the fully configured system are installed under sifis-home.target. Figure 26 below shows added targets with their conditions.

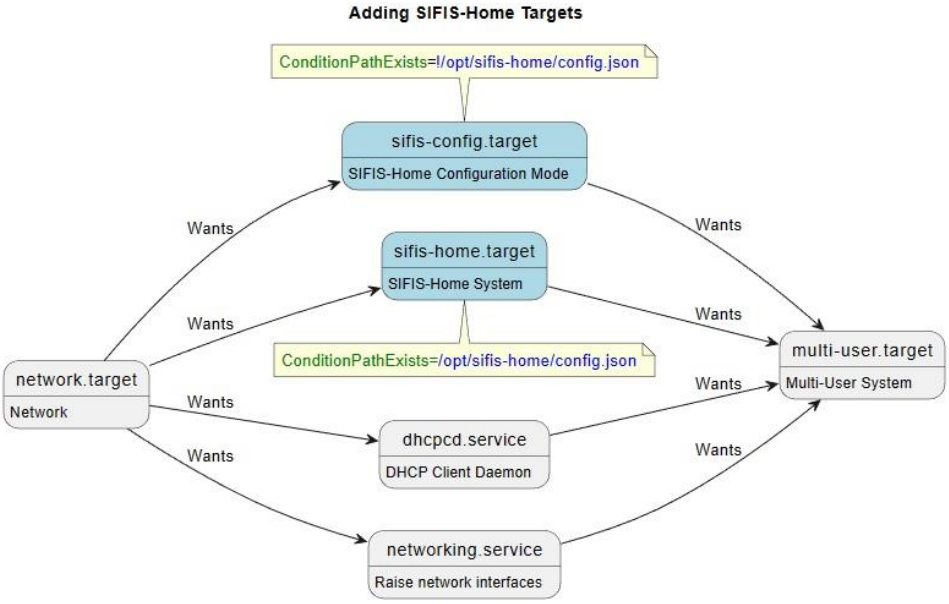


Figure 38 Added targets and conditions.

These targets are added to the /etc/systemd/system directory. Other targets relevant to the boot process were left out of Figure 26 for simplicity of illustration.

- **3rd party API**
  This component provides the API that allows downloaded 3rd party applications to interact with the Smart Home system.
  It is implemented as a rust crate based on "tarpc" to implement both applications (that act as untrusted clients) and runtimes (that act as trusted endpoint and interface to the DHT).
  The API models the following devices:
  - Lamp
  - Sink
  - Door
  - Fridge

  The application binaries are expected to run in a segregated environment such as "ujail" or docker and interact only with the runtime via a Unix socket.

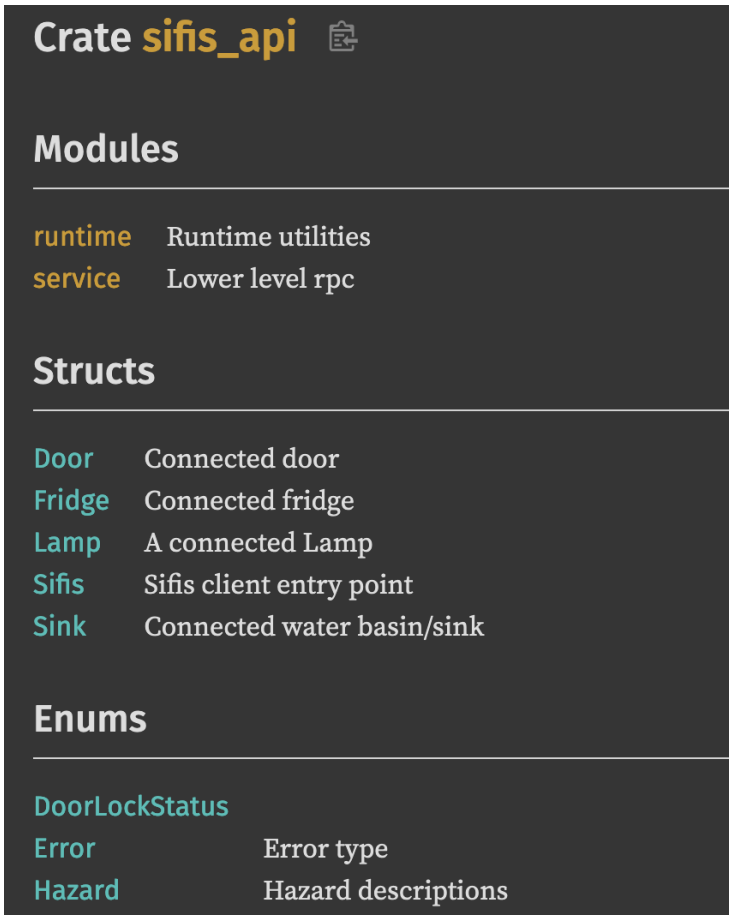


Figure 39 The manifest generator/validator

The manifest generator/validator detects the API usage by inspecting the binary and the runtimes can consume the manifest to further restrict the application capabilities.

4.3.6 NSSD Manager

The NSSD Manager is the module 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 performed and protected by using the advanced security protocols and solutions that have been designed and developed in the context of WP3.

  This approach enables a more convenient, remote provisioning of commands to devices acting as CoAP client, issued by applications within or outside the Smart Home. This 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 acts as pub-sub client in the DHT-based pub-sub system enforced in the Smart Home. That is, the CoAP client interacts with a DHT-based pub-sub broker in order to receive published commands. Then, it takes an appropriate course of action based on the received command, and accordingly communicates with the target CoAP server(s). Finally, it publishes corresponding results to the DHT-based pub-sub broker, so that that they are also available to applications subscribed to the DTH pub-sub broker for obtaining such results.
  Further details about the implementation of 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, thus making them discoverable and linkable, and also by defining a standard data model and APIs to make the devices interoperable and to enable the exchange of data between devices and systems.

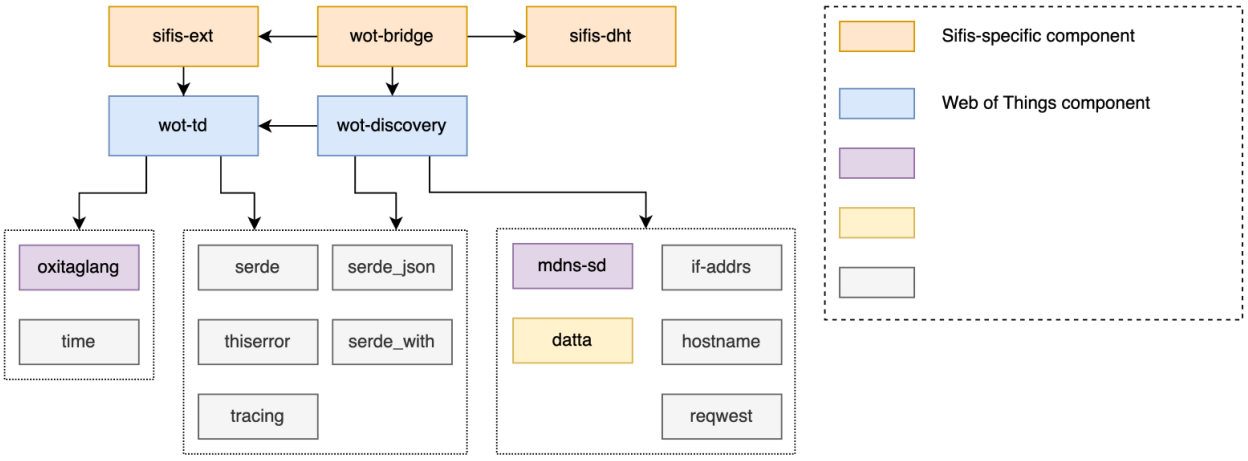


Figure 40 The WoT manager interaction flow.

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

The interface between the DHT and the **demo-things** is based on the wot-bridge used also in WP6, it uses the **sifis-dht** crate, wot-discovery, and wot-consume.

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	Release 0.1
libsifis	Initial Proof of Concept	Being replaced by **sifis-ext**
sifis-ext	Thing Description extension	Work in Progress

Table 1 WoT sub-components

4.3.7 DHT Manager

The DHT Manager (see **Errore. L'origine riferimento non è stata trovata.**) is composed of two different software components, namely the DHT and the Fiware API. The DHT component manages the DHT and allows other applications and services to access it. Instead, the Fiware API component interacts with Yggio in order to provide a FIWARE-compatible API [FIWARE, 2021] to a SIFIS-Home-enabled Smart Home.

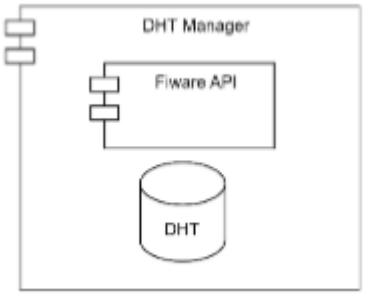


Figure 41 The DHT manager components

- **FIWARE API**
  This component forwards the persistent messages published through the DHT to the Ratatosk FIWARE Context broker that is part of the Yggio instance residing on the SIFIS-Home cloud. Also, it forwards commands entered by the user in the Yggio user interface to the DHT. The Fiware API component uses both Rest API and the MQTT protocol to set up and receive/publish messages from/to Yggio. More in detail, the FIWARE API component is provided with a set of dedicated credentials that allow it to access the Yggio Rest API and MQTT broker. It then uses the Rest API to associate and reserve a dedicated MQTT topic for each DHT topic. The FIWARE API component has been developed using the Rust language. Its source code can be found at https://github.com/sifis-home/dht-to-mqtt.

- **DHT**
  The SIFIS-Home DHT is a component that offers a completely distributed publish/subscribe mechanism (i.e., a central broker is not present), through which SIFIS-Home applications can exchange messages. The SIFIS-Home DHT allows to publish persistent as well as volatile messages. Persistent messages are stored in a persistent way on Sqlite and PostgreSQL databases, so that they are available even after a node reboot Volatile messages are instead

delivered to all the running applications but are not stored. The SIFIS-Home DHT has a built-in mechanism to solve possible data conflicts that can arise when a network partition occurs. In particular, every time a message is published on the DHT, the DHT also em a publication timestamp to it. Then, the publication timestamp is used to assure that only the most recently published messages will be stored and made available to the applications.
The SIFIS-HOME DHT has been developed using the Rust language. Rust applications can include the DHT by embedding it as a library. Non-Rust applications can access the DHT by means of a REST + WebSocket API provided by the DHT Manager. The current version of the SIFIS-Home DHT code can be found at .

4.4 *VPN Manager*

Since the VPN Server was replaced by the MQTT publish subscribe mechanism to manage interactions between the SIFIS Home network inside a Cyper Security perimeter the VPN Manager is no longer required and will not get implemented.

4.5 *NSSD Framework*

The SIFIS-Home NSSD Framework (see **Errore. L'origine riferimento non è stata trovata.**) 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 particular, the NSSD Framework is composed of the Bootstrap Manager component and the Device API Manager component. Please refer to D6.2 for all the details of the NSSD Framework implementation.

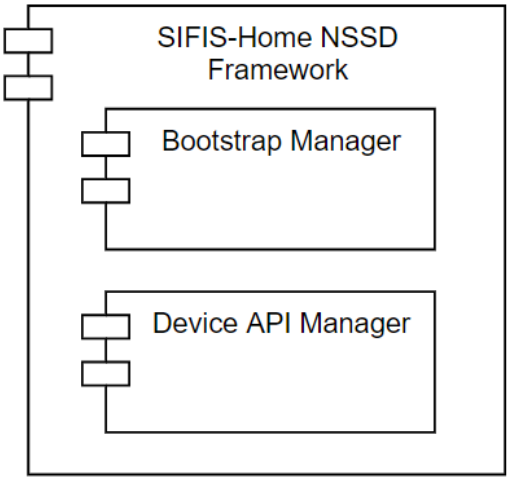


Figure 42 The NSSD Framework components

5 Integration of analytics and security solutions

5.1 *Overall integration strategy*

The design of the SIFIS-Home framework in WP1 is based on Docker containers using the *microservices* design pattern. Each microservice defines an API, usually a Rest API or specific DHT messages, for interfacing with it. In order 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 the APIs that they provide or via the DHT. This makes it fast and easy to add new functionalities.

5.2 *Analytics integration*

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 their version). For this reason, we decided to use virtualization and to deploy each analytic in its customized container, leveraging on the Docker technology. Hence, each analytic 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 deploying the analytics. A repository of Docker container images for analytics is created to easily store and retrieve available images.
3. The analytics running within the Docker containers resulting from the previous step are invoked by the Analytics APIs. Each analytic provided by the Data Analysis Toolbox is paired with a DHT topic that is dedicated to publishing input data for each analytic invocation. Moreover, each analytic has a separate topic for the results obtained by running it on some data, so that the results are published to another topic that is dedicated only for the analytic results.
4. The interactions between the Data Analysis Toolbox and the other components of the SIFIS-Home Framework that need to ask for analytics executions occur through the DHT, which provides a publish/subscribe mechanism.
5. Interactions start from the analytics' consumer side requesting the execution of an analytic. This request is sent as a message in JSON format published on the topic related to the analytic in question. The JSON message contains invocation details including the topic name, topic ID, description, in addition to a set of input data to be processed by the analytics and optional privacy-related parameters.
6. Upon receiving a message published on a topic, the Analytics APIs subsystem invokes the analytic associated with that topic. As previously mentioned, each analytic engine is deployed in distinct containers, using the Docker technology. The invocation of an analytic by the Analytics APIs subsystem occurs by means of the HTTP or the WebSocket protocol, depending on the analytic deployment on the related container. Each analytic relies on one of the two protocols and needs to be provided with different parameters.
7. Once the analytic 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.

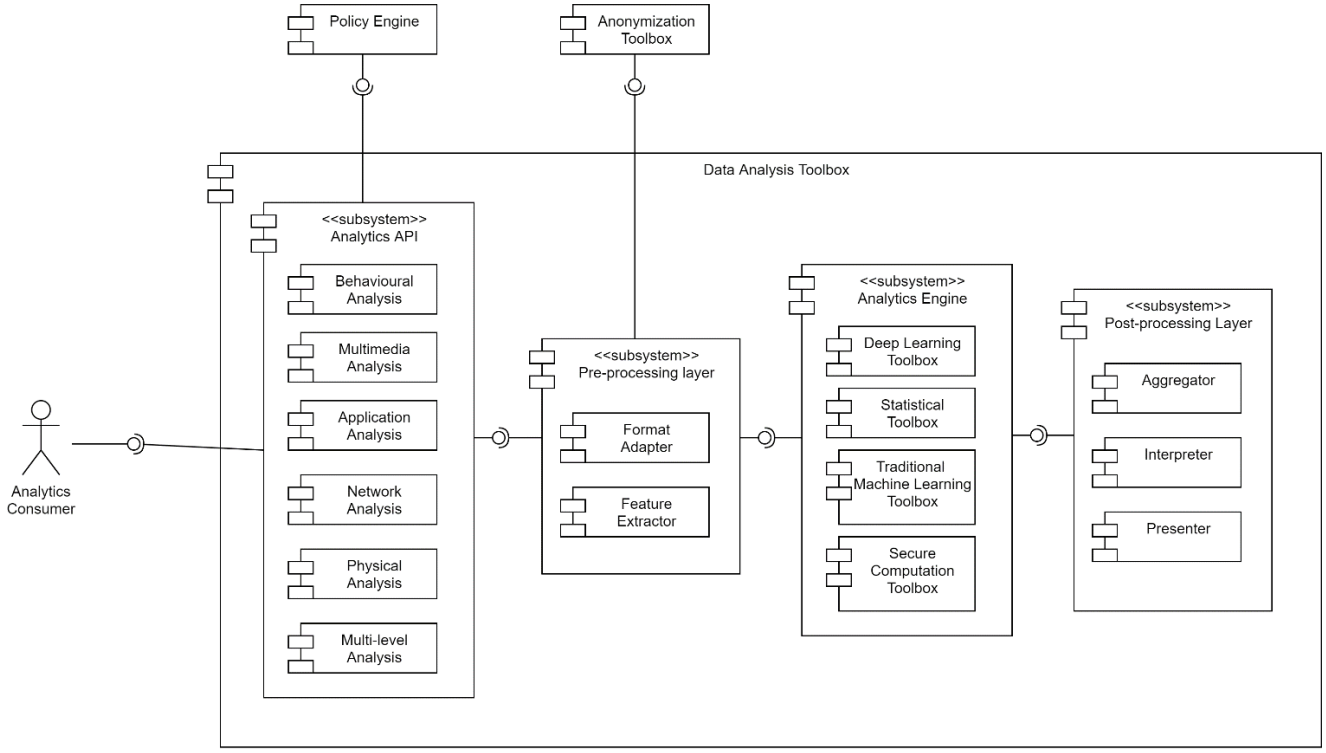


Figure 43 Analytics components

- **Pre-Processing Layer/Post-Processing Layer**
  This sub-system pre-processes data (by adjusting its format and removing unnecessary information) coming from different sensors and devices, before sending it to the different SIFIS-Home analytics. Also, it collects the information produced by the different analytics and prepares 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 performed before the giving control to the analytics sub-system on the device performing data analysis. On the other hand, data post-processing is performed directly after the execution 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 sub-system interacts with the DHT for subscribing to the topics representing all the analytics provided by the Data Analysis Toolbox and for collecting the requests sent by the other components of the SIFIS-Home framework through their publication on these topics. Depending on the topic published by the component requesting the execution of an analytic, the right sub-component of the Analytics API sub-system is invoked. Such a sub-component knows how to interact with the requested analytic running in its Docker container. If a pre-processing operation is required, the right function of the pre-processing layer sub-system is also invoked. The following sub-component are included in the Analytics API sub-system:

- **Behavioural Analysis**
  This API uses WebSocket to invoke analytics toolboxes responsible for the analysis of behavioral and environmental data collected from IoT devices like sensors, smart watches, smart thermostats, and smart cameras in order to derive insights, track user behaviour, predict user behaviour, and identify anomalous actions, or for activity classification. Behavioral and environmental data are captured by SIFIS-Home sensors and devices and published with the device ID to the required analytic topic. The analytic topic is used to invoke the WebSocket-based API that the analytic expose, in order to execute the responsible Docker container with the provided shared data and input 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 behaviors 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 behavior 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 provide REST API for retrieving service status (start/stop/restart/is_running), available interfaces, configurations, 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 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 the 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. The AUD manager provides a REST API for starting and stopping the analytic. The REST API also provides methods to retrieve the 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**
  The WebSocket-based API of this sub-component 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, The produced results are published by the Analytics Engine to the related topics.

- **Application Analysis**
  Application data are captured by SIFIS-Home components and published with the component ID to the related analytic topic. This results in the invocation of the WebSocket-based API that the analytic exposes, with the consequent execution of the related Docker container that takes as input the provided 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 related analytic topic. This results in the invocation of the WebSocket-based API that the analytic exposes, with the consequent execution of the related Docker container that takes as input the provided shared data and parameters. The results are then published by the Analytics Engine to the related topic.

- **Analytics Engine**
  This sub-system performs the execution of the SIFIS-Home analytics, providing statistical-based, machine learning, and deep learning tools for the analysis of data collected by sensors. The analytics engine processes collected data and makes logical predictions based on the provided data input and instructions that it has been configured on. Finally, it releases the data as output and shares it back with the analytics consumer.
  The Analytics Engine is implemented as a collection of Docker containers, invoked based on the received message topic from components publishing data to be processed. Depending on the topic where the message was published, the right Docker container is selected and executed, taking as input the data and parameters included in the message. After the Docker execution is finished, the results are also published by the Data Analytics Engine to the related analytic result topic. The details concerning the implementation of each single analytic are provided in D4.2 and D4.3.

5.3 *Network and security solution integrations*

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 deliverable D3.3.

Irrespective of the specifically considered, possibly combined, security solutions, the interacting parties are devices acting as CoAP client and/or CoAP server. In stand-alone, proof-of-concept implementations used for focused demonstrations (see Annexes C and D of deliverable D3.3), the user could interact with a CoAP client, provide it with a command available from a basic set, and make it 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 a keyboard terminal to the CoAP client, which in turn provided any relevant output (e.g., responses obtained from the CoAP servers) on a display monitor.

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. Also, 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, it would be more convenient for the user to remotely provide CoAP clients with commands to process.

To this hand, the following strategy was considered and implemented during the activities in WP5 that integrated the security solutions developed in WP3 into the SIFIS-Home solution at large.

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 keyboard terminal; and ii) relay back any relevant output following the course of action triggered by a command, as content published on a related, device-specific topic rather than as output on a display monitor.

Practically, this has been enforced by leveraging the DHT-based pub-sub system used in the SIFIS-Home solution and deployed in the Smart Home network. That is, the DHT enforces a distributed pub-sub broker, exposing an interface based on WebSocket for publishing on and subscribing to its topics.

Then, a device acting as a CoAP client also acts as a pub-sub client, by bi-directionally interacting with the DHT-based pub-sub broker through WebSocket. In particular, the CoAP client subscribes to a topic on which it receives user commands, and publishes on a separate related topic in order to provide the output from the actions performed following those commands. Instead, the user application that issues such commands takes the reversed pub-sub roles, in order to publish the issued commands and subscribe to the outputs from the CoAP client. The two companion topics are specific for the exact CoAP client in question, i.e., one for providing commands to that client and one to disseminating the outputs from that client.

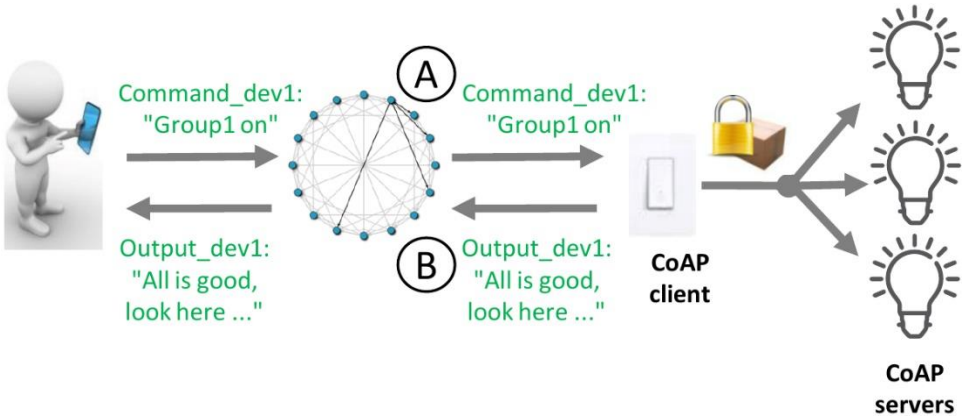


Figure 44 Control of CoAP devices through the DHT-based pub-sub broker

Figure 44 shows an example of complete interaction workflow discussed above, considering some of the security solutions developed in WP3. The example assumes that the CoAP client acts as a subscriber for the pub-sub topic "Command_dev1", and as a publisher for the pub-sub topic "Output_dev1". Conversely, an application run by the user within the Smart Home (e.g., a mobile application) acts as a publisher for the pub-sub topic "Command_dev1", and as a subscriber for the pub-sub topic "Output_dev1". The CoAP client is a member of a device group, within which communication is based on CoAP and protected with the security protocol Group OSCORE (see Section 4.3.2). The group also includes three CoAP servers, and all the group members have previously joined the group and obtained the required keying material from an OSCORE Group Manager (not shown in the figure), following an access control workflow based on the ACE framework and its OSCORE profile (see Sections 4.3.1 and 4.3.2).

At step A, the user application publishes the command "Group1 on" to the pub-sub topic "Command_dev1". As it is subscribed to the same topic, the CoAP client receives such a command from the DHT-based pub-sub broker. In particular, the command is intercepted by the "CoAP Manager" component of the "NSSD Module" running on the CoAP client, and it triggers the CoAP client to take the same course of actions as it would have when receiving the same command from a keyboard terminal. That is, the CoAP client sends a CoAP group request protected with Group OSCORE, in order to securely switch on the lights controlled by the three CoAP servers within the group "Group 1". After

that, the three CoAP servers reply to the CoAP client, confirming that the operation succeeded.

Then, at step B, the CoAP client collects the responses from the different CoAP servers, and provide those to the "CoAP Manager" component, by means of which the CoAP client publishes the output from those responses to the pub-sub topic "Output_dev1". As it is subscribed to the same topic, the user application receives such an output, and can display it to the user.

The actual interaction between the DHT-based pub-sub broker and the "CoAP Manager" component at CoAP clients has been implemented by relying on the Eclipse Tyrus library available at https://projects.eclipse.org/projects/ee4j.tyrus. The codebase of the WP3 security solutions available at https://github.com/sifis-home/wp3-solutions also comprise such a component to enable the interaction between the CoAP clients and the DHT-based pub-sub system.

As discussed above, a pair of topics has been considered for each CoAP client. That is, one topic where commands are published and to which the CoAP client subscribes to; and one where the same CoAP client publishes outcomes from the execution of those commands, and the original command issuers subscribe. To this end, a consistent and easy-to-synchronize semantics was enforced for the topic names, as based on the following schema: pub-sub topics where commands are published have a name "command_devX" while pub-sub topics where outcomes from the execution of those commands are published have name "output_devX", where X is an identifier of the CoAP client in question within the Smart Home. Of course, it is possible to easily customize the topic names.

Building on the same principle, 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, aided by an MQTT-to-WebSocket bridge, will interact with the DHT-based pub-sub system deployed in the Smart Home, just like the user would do for the case discussed above where presently at home. It is worth noting that this has no impact and makes no difference for the CoAP clients, as they would still interact via WebSocket only with the DHT-based pub-sub broker deployed in the Smart 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 pre-established prefix to use for uniquely identifying the specific Smart Home and thus enforce per-smart-home topic namespaces.

Finally, the same principle used by the "CoAP Manager" component on CoAP clients has been also used on two devices acting as ACE Authorization Server (AS) and OSCORE Group Manager (GM). By doing so, the two devices are able to interact via WebSocket with the DHT-based pub-sub broker, and to publish log messages documenting relevant, pertaining events. Such log messages are published on a dedicated pub-sub topic and would thus persist on the DHT in the first place. In addition to that, such log messages will be available to any interested subscriber to such topic.

One such particular subscriber that has considered and tested is in fact the Yggio platform. That is, by taking advantage of the MQTT-to-Websocket bridge mentioned above, a cloud instance of Yggio subscribes to such log-related pub-sub topic at the DHT-based pub-sub broker in the Smart Home.

Consequently, the logs published by the AS and GM on the logging pub-sub topic effectively reach the Yggio platform, that can effectively store such logs, show them to authorized users logged into Yggio and potentially perform automatic analysis and monitoring based on the logged events.

Specifically, on a dedicated pub-sub topic “log”, the GM publishes log events related to successful or failed attempts from CoAP endpoints to join an OSCORE group. Similarly, on the same pub-sub topic “log”, the AS publishes log events related to: i) successful or failed attempts from endpoints acting as ACE Clients in requesting Access Tokens as evidence of their authorization towards accessing protected resources at an ACE Resource Server; as well as ii) Access Token revocation operations.

6 Continuous integration and deployment

6.1 *Process*

The CI (continuous integration) and CD (continuous deployment) are implemented based on GitHub actions, Docker MultiArch files, the GitHub container registry (ghcr.io) and a Docker feature called “Watchtower”. Once a Docker MultiArch file, which contains executables for both x86 and ARM architectures, is built with the help of GitHub actions, it is uploaded to the GitHub container registry. The exact steps for GitHub actions to build each file are defined by the partner developing the component.

The first time a new SIFIS-Home Smart Device is deployed, some manual interaction is required to install on the target device all the microservice Docker containers that the SIFIS-Home framework consist of. However, once a device is deployed, the Watchtower feature checks every five minutes if any newer version of any of the included Docker containers is available. If that is the case, then the Watchtower automatically updates the Docker container with the latest version. Since all SIFIS-Home devices run Watchtower within a 5 minute interval, all of them converge to having the same collection of Docker containers. Using Docker Watchtower is an elegant way to use state-of-the-art tools to complete the most difficult step in the CI/CD tool chain, namely the actual continuous automatic deployment to all devices after the initial manual deployment via the Ansible deployment script. Figure 22 below illustrates all the steps in the SIFIS-Home CI/CD process.

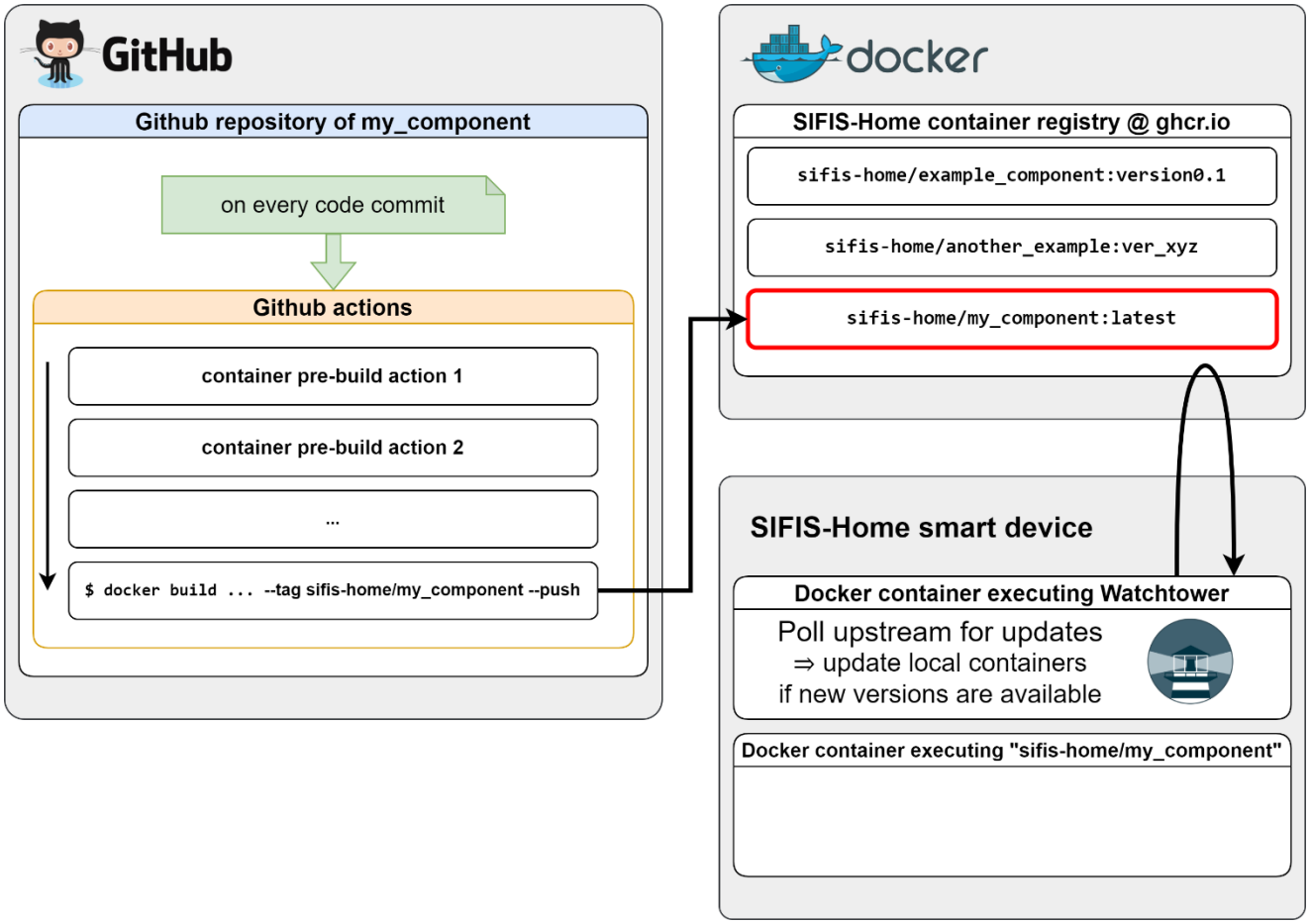


Figure 45 The SIFIS-Home CI/CD process

Example CI/CD scenario:

1. The developer commits new code to GitHub.
2. The GitHub actions are triggered by the new commit. These include automatic testing of the new code and if they fail the build will automatically fail as well.
3. The final GitHub actions step is a build command that produces a Docker image. The new Docker image is pushed automatically to the GitHub's container registry at ghcr.io.
4. For every SIFIS-Home Smart Device, once an updated version of a Docker image is detected by Watchtower, Watchtower will upgrade any local Docker image running older versions.

Since SIFIS-Home is a research project, our decision was to just use one step in the deployment phase of new software.

6.2 *GitHub*

The code developed in SIFIS-Home is available in GitHub at the link sifis-home (github.com), the code is reviewed by the partners as part of the WP2 activities. The SIFIS-generate tool is available to streamline the process of adding GitHub Actions to projects to have a working Continuous Integration.

The **sifis-generate** crate is also using its own continuous integration template as shown in **Errore. L'origine riferimento non è stata trovata.**.

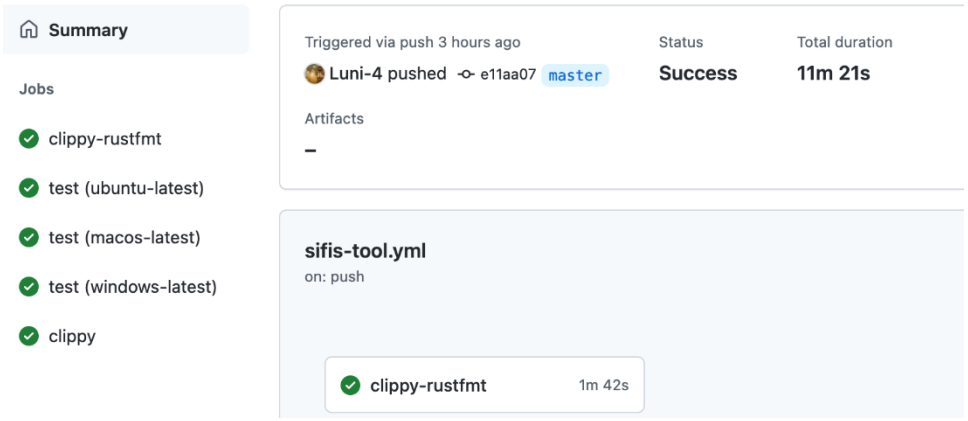


Figure 46 Continuous integration template for sifis-generate

Being an executable, it also peruses the deployed components to provide prebuilt binaries (see Figure 47) for Linux, Windows and macOS (for every release).

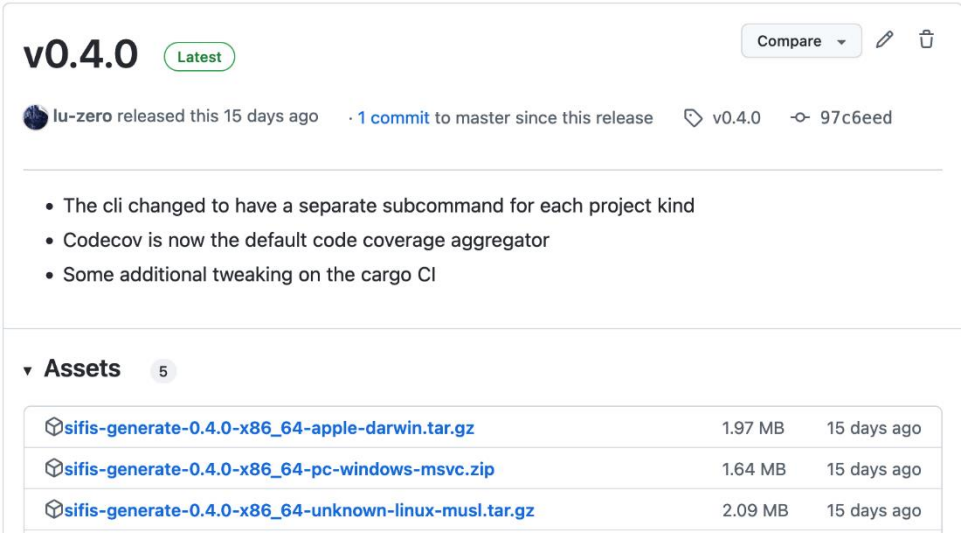


Figure 47 Prebuilt binaries

Figure 48 The SIFIS generate tool.

A checklist had been provided to the partners to make them aware of what to check about the status of their code and self-assess what is missing and what should be better addressed.

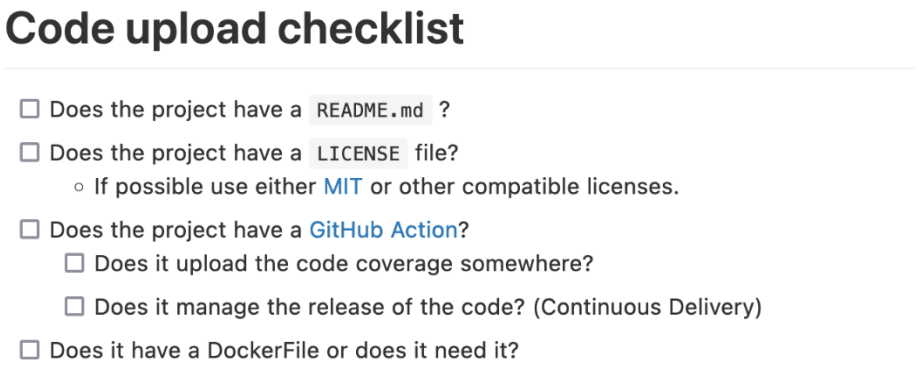


Figure 49 Code integration check list

Currently there are 87 code repositories under the SIFIS-Home GitHub organisation. There are many activities listed and regular code contributions are made.

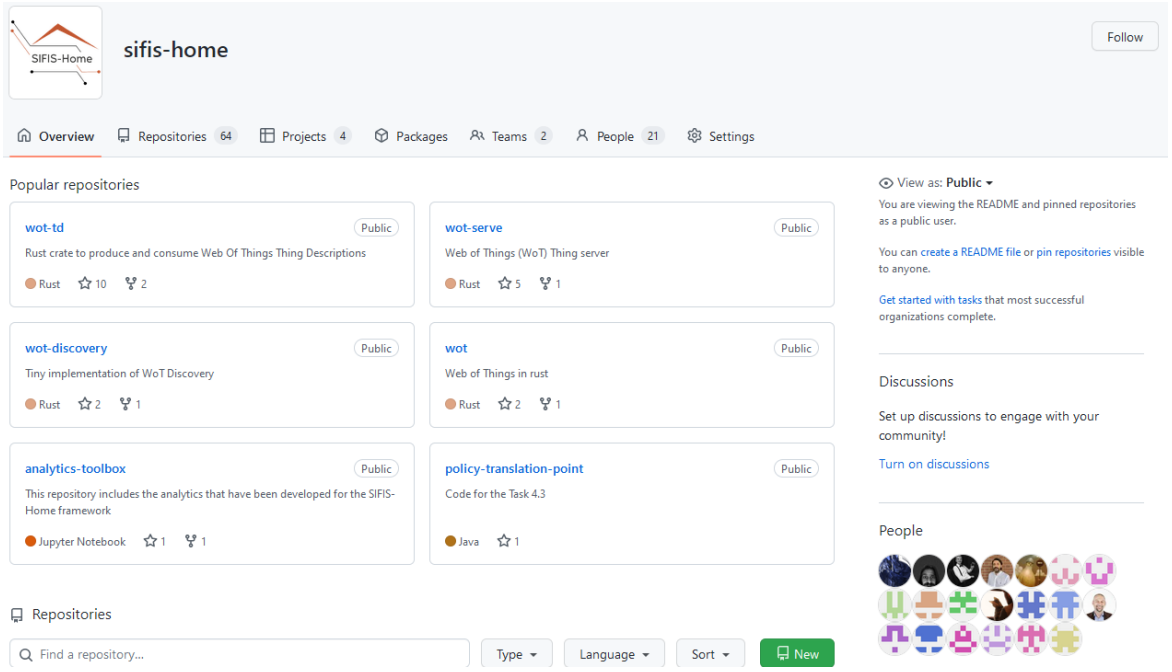


Figure 50 The SIFIS-Home repositories

GitHub links:

The source code for the developed SIFIS-Home components is available on GitHub via the following link: https://github.com/sifis-home

Some direct links to components are provided here for reference:

- Secure Message Exchange Manager, Content Distribution Manager, Authentication Manager, Key Manager, CoAP Manager (Group OSCORE communication, Group OSCORE key provisioning using the ACE framework, ACE OSCORE profile, EDHOC key establishment, DHT-based command/output interface for CoAP clients) https://github.com/sifis-home/wp3-solutions
    - Californium (CoAP and OSCORE) https://github.com/eclipse/californium
- Network Protection manager: https://github.com/sifis-home/wp5_network_protection_manager.
- Network analysis: https://github.com/sifis-home/wp4-edge_ids/releases
- MUD adaptation and AUD Manager: https://github.com/sifis-home/wp4-aud_manager.
- DHT:
    - https://github.com/sifis-home/libp2p-rust-dht.
    - https://github.com/sifis-home/domo-wot-bridge
- Smart Device Mobile API: https://github.com/sifis-home/wp6_mobile_application_api
- WebThings Arduino Library: https://github.com/WebThingsIO/webthing-arduino
- WoT Wi-Fi actuators Firmware: https://github.com/sifis-home/domo-wot-actuator
- Privacy Dashboard: https://github.com/sifis-home/privacydashboard

- WoT rust:
  - https://github.com/sifis-home/wot-td
  - https://github.com/sifis-home/wot-serve
  - https://github.com/sifis-home/wot-consume
  - https://github.com/sifis-home/wot-discovery
  - https://github.com/sifis-home/demo-things
- Behaviour testing PoC for demo things: https://github.com/sifis-home/wot-test
- SIFIS-Home developer API: https://github.com/sifis-home/sifis-api
- Policy Enforcement Engine: https://github.com/sifis-home/usage-control
- Analytics Toolbox: sifis-home/analytics-toolbox: This repository includes the analytics that have been developed for the SIFIS-Home framework (github.com)
- Mobile app: https://github.com/sifis-home/sifis-home-mobile
- FIWARE Context Broker (Ratatosk): https://github.com/sifis-home/yggio-ratatosk
- SIFIS-Home Cloud UI (Yggio): https://github.com/sifis-home/yggio-components
- The Market Place: https://github.com/sifis-home/yggio-components/tree/master/control-panel-v2/src/pages/apps

The DHT topics used in the SIFIS Home security architecture is defined in Github as JSON schemas.

- DHT Topics: sifis-home/json-schemas: A collection of json schemas defined within SIFIS-Home (github.com)

7 Validation, verification status and results

Verification and validation have been carried out through the following separate tests:

- Unit test: This test is usually designed and executed by the developers of the code under testing, to confirm that the implementation works as expected.
- Integration test: These are usually automatic tests that are executed once code has been committed. If the integration test fails, then the code will get rejected by the CI system.
- System test: These tests are executed on the fully integrated and deployed system, in the available test beds and according to appropriate test cases. In SIFIS-Home, both emulated and the simulated test beds are considered.
- Validation: This provides a high-level confirmation that the execution of a software implementation results in the expected behaviour. For example, it confirms that that the data is flowing between the various parts of the system in the expected way.

In the SIFIS-Home project, the partners have different internal environments to develop and verify their code before committing it to the pertaining repositories. Also, except for what is mandated by WP2 and for the continuous integration and continuous deployment processes, the partners could rely on the Github actions for configuring and adapting the unit test and integration test strategy to suit their specific requirements. Some of the partners, e.g. during the development of the cloud interface, have also run very extensive tests that go far beyond unit test before committing the related code to the SIFIS-Home

repositories. Several partners have also developed automatic test scripts that are automatically executed via Github actions on code that is being integrated and then automatically deployed via Docker Watchtower to all SIFIS Home systems that have a deployment of the component .
The system tests were defined in deliverable D1.2 to be executed on the test beds, and the produced results are provided in Section 7.6 System verification and validation. It is worth noting that the verification of a system like SIFIS-Home can be carried out to a considerable extent by simply validating the flow of data in the system, and confirming that all the expected services are running and that data is flowing between the different system entities as expected.

7.1 *wot-rust crates*

All the crates use the continuous integration setup for GitHub-Actions provided by the **sifis-generate** (see Figure 50).

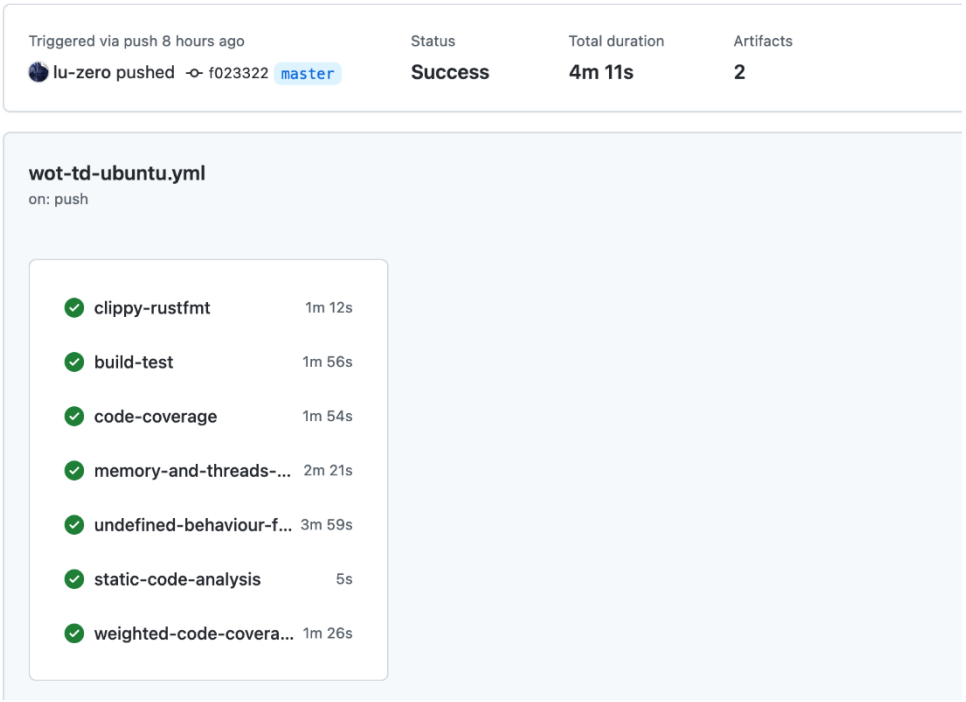
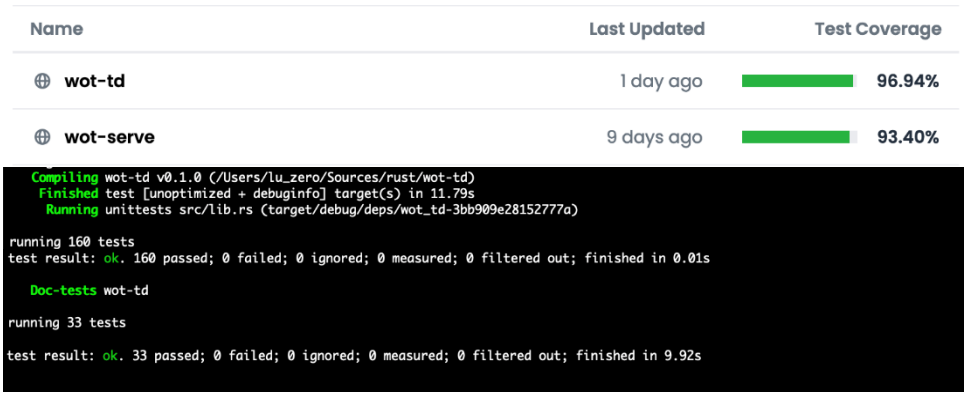


Figure 51 Continuous integration setup

The crates aim at staying above 90% coverage for every commit (Figure 51).



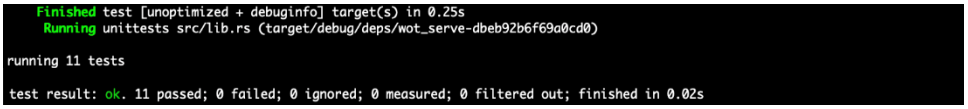


Figure 52 Automated tests of every commit

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

7.2 *UX*

7.2.1 Cloud interface

The SIFIS Home cloud interface (Yggio) uses a combination of automatic integrations tests for every PR –Pull Request (PR) to the code base and of system tests before releasing the code. Every PR is also code reviewed by an independent developer according to a check list, to ensure that the code design follows architectural and security guidelines as well as that the code quality as such is high without obvious flaws. Before integrating an updated version of the Cloud interface to the SIFIS-Home repository and deploying to the test beds in the Panarea server, the software must also pass the release test, which is equivalent to reach commercial quality level of the cloud software.

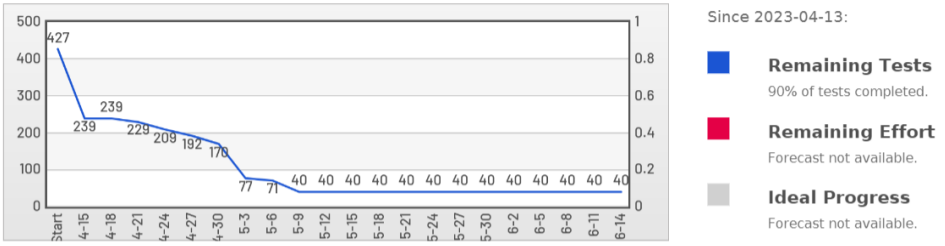


Figure 53 Test execution ongoing. In the end 40 test cases were not executed.

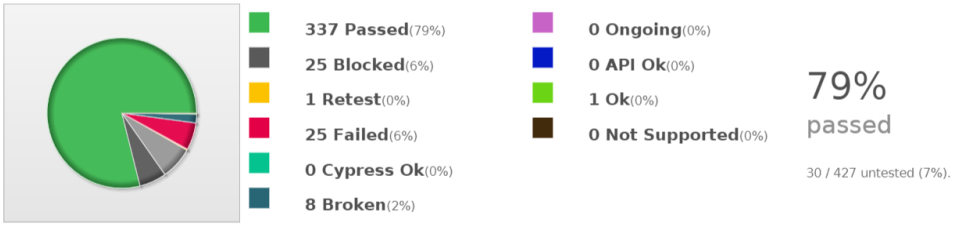


Figure 54 Release test result of Yggio v3.19 that become the commercial release v3.20.

Blocked test cases are items that for some reasons cannot be verified altogether. Often, this is caused by missing hardware equipment or other type of environments that are required to execute the tests. Failures are tests that do not successfully pass. All of these will get analysed one by one, towards making a judgement on whether any of them is a blocker for a release or not.

7.2.2 Mobile Application

The SIFIS-Home Mobile Application in the UI component is developed using Javascript technology. Most of its functionality is provided by services used through an exported API. The Mobile Application itself is tested through functional testing methods to validate that it meets all the pertaining functional requirements.

The API is tested using the Python unittest framework. The unit tests call operations on the same

interfaces as the actual Mobile Application, and test that the returned values are correct and as expected.

```
MBA:sifis-home-mobile aegir$ python3 test/test.py -v
test_dht_get (__main__.TestSifisMobile) ... ok
test_github_containers (__main__.TestSifisMobile) ... ok
test_mobile_api_device_configuration (__main__.TestSifisMobile) ... ok
test_mobile_api_device_info (__main__.TestSifisMobile) ... ok
test_mobile_api_device_status (__main__.TestSifisMobile) ... ok
test_yggio_login (__main__.TestSifisMobile) ... ok
test_yggio_organization (__main__.TestSifisMobile) ... ok
test_yggio_users_me (__main__.TestSifisMobile) ... ok

----------------------------------------------------------------------
Ran 8 tests in 3.953s

OK
```

Figure 55 Mobile Application unit test verification results

7.3 *DHT*

The current implementation of the SIFIS-Home DHT has been tested using standard unit tests. The following **Errore. L'origine riferimento non è stata trovata.** reports the current test results obtained using the *cargo test* utility.

```
running 18 tests
test webapimanager::tests::test_webapimanager_websocket ... ok
test webapimanager::tests::test_webapimanager_rest ... ok
test domobroker::tests::domo_broker_test_websocket_get_topicname ... ok
test domobroker::tests::domo_broker_rest_post_test ... ok
test domobroker::tests::domo_broker_rest_get_topicuuid ... ok
test domobroker::tests::domo_broker_test_websocket_get_topicuuid ... ok
test domobroker::tests::domo_broker_empty_cache ... ok
test domobroker::tests::domo_broker_rest_get_all ... ok
test domobroker::tests::domo_broker_rest_pub_test ... ok
test domobroker::tests::domo_broker_rest_get_topicname_not_present ... ok
test domobroker::tests::domo_broker_test_websocket_getall ... ok
test domobroker::tests::domo_broker_test_websocket_pub ... ok
test domobroker::tests::domo_broker_test_websocket_empty ... ok
test domobroker::tests::domo_broker_test_websocket_post ... ok
test domobroker::tests::domo_broker_test_websocket_delete ... ok
test domobroker::tests::domo_broker_rest_get_topicuuid_not_present ... ok
test domobroker::tests::domo_broker_rest_delete_test ... ok
test domobroker::tests::domo_broker_rest_get_topicname ... ok

test result: ok. 18 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 11.37s

     Running unittests src/main.rs (target/debug/deps/sifis_dht_broker-f50841e6f359d4ae)

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

   Doc-tests sifis-dht-broker

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

Figure 56 The DHT verification results

7.4 *Security solutions (WP3)*

The security solutions developed in WP3 and indicated in Sections 4.3.1 and 4.3.2 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.

Further details on the two focused demonstrators are in Annexes C and D of deliverable D3.3.

Building on the two focused demonstrators above as a starting point, such security solutions have been integrated in the SIFIS-Home solution within the WP5 activities. The integration process encompassed especially two aspects.

On one hand, it enabled the actual interoperation of such security solutions for CoAP within the SIFIS-Home solution at large, mainly by means of the "CoAP Manager" component of the "NSSD Manager" architecture module (see Section 4.3.6) and relying on the DHT-based pub-sub broker used in the context of the SIFIS-Home solution, as an additional interface for circulating commands as well as resulting output. On the other hand, the integration process covered the practical automation of code verification and deployment, leveraging state-of-the-art approaches relying on multi-architecture containerization of Software images, in concert with partners providing the required infrastructure.

Further details on the integration of the security solutions from WP3 are provided in Section 5.3.

Finally, the security solutions from WP3 can also undergo automatic testing within the WP5 testbed, through the Python script at https://github.com/sifis-home/wp3-solutions/blob/master/wp3-tester.py

From a high-level point of view, the Python script in question: i) sends commands to be delivered to the CoAP clients through the DHT; and then ii) checks that the correct, expected messages conveying the result of such commands execution are received back from the DHT, as relaying responses received by the CoAP client from the targeted CoAP server(s).

The script effectively tests CoAP communications between one CoAP client and one CoAP server protected with OSCORE, as well as between one CoAP client and multiple CoAP servers in a group protected with Group OSCORE (see Sections 4.3.1 and 4.3.2). The script produces an output like the one shown in the example below.

```
rikard@sifis-device3:~/rise-wp3$ python3 wp3-tester.py

Will perform tests for the WP3 solutions applications.
Sends commands to the DHT and confirms reception of correct message back.

Test Group OSCORE Client: PASS
Test CoAP Client: PASS
Test EDHOC Client: PASS
rikard@sifis-device3:~/rise-wp3$
```

Figure 57 Example of output from the automatic testing script for the WP3 security solutions

As to the one-to-one communication case protected with OSCORE, the script indirectly tests also the prior, successful execution of the EDHOC key establishment protocol yielding an OSCORE Security Context between the CoAP client and the CoAP server (see Sections 4.3.1 and 4.3.2). This is because

the establishment of such an OSCORE Security Context is a pre-requirement for OSCORE-protected communications to occur in the first place. That is, if such initial establishment was not successful, the script would not pass successfully either.

Similarly, as to the group communication case protected with Group OSCORE, the script indirectly tests also the successful execution of the authorized group joining through the ACE framework, with the consequent acquisition of Group OSCORE keying material and parameters from the Group Manager for the CoAP client and CoAP servers joining as group members (see Sections 4.3.1 and 4.3.2). This is because the prior, successful group joining and related key provisioning is a pre-requirement for group communications protected with Group OSCORE. That is, if such initial group joining and key provisioning were not successful, the script would not pass successfully either.

7.5 *Network Anomaly Detection / AUD Manager*

Basic functionality and operation of the AUD manager (see Section 5.4) have been tested with virtual mock devices and synthetic test scenarios. The anomaly detection analytic implemented in the AUD manager relies on learning network activity patterns of connected devices in a live system. Therefore, validation of operation with synthetic tests does not directly translate to operation in a feature-complete testbed and/or real system deployment.

7.6 *System verification and validation*

This section reports the verification and validation results of the non-functional and security requirements of the SIFIS-Home framework from deliverable D1.2 that we used to design and verify the security architecture.

Independent of the programming language used to develop the different components, the SIFIS Home architecture were designed via Github and Docker Watchtower to support both continuous integration and continuous deployment. The integration and deployment processes enabled the SIFIS-Home test beds to be continuously up and running with a live communication flow from the Smart Devices, the NSSD – Not so smart devices, the Analytics, as well as some standard IoT validation devices to support a fast and efficient total verification and validation of the architecture including bug fixing.

The verification has been performed by testing the requirements independently on the emulated and simulated testbed. When performance are reported, the worse performance between emulated and simulated testbed is reported. Validation results are reported in the following table:

Req. ID	Req. description	FR	Priority	Validation strategy	Validation status
PE-01	The user authentication shall happen in less than 2s.	F-02 F-03	Critical	UI performance test	Through automated UI performance test the user authentication is verified to be performed in less than 900 ms
PE-02	The user recognition (identification/ biometric-based) shall happen in less than 5s.	F-06	Critical	UI performance test	Through automated UI performance test the user recognition is verified to be performed in less than 5 seconds.
PE-03	Biometric-based authentication should be performed in less than 5 seconds.	F-03	Standard	Under consideration	Through automated UI performance test the biometric user

					authentication is verified to be performed in less than 5 seconds.
PE-04	Activation of features based on user identity (biometric recognition) should be performed in less than 5 seconds.	F-04 F-05	Standard	UI performance test	Through automated UI performance test the biometric user authentication is verified to be performed in less than 5 seconds.
PE-05	Recognition of the start of an interaction through voice command should be performed in less than 2 seconds.	F-06	Standard	UI performance test	Through automated UI performance test the start of interaction through voice commands is verified to be performed in less than 2 seconds.
PE-06	The interpretation of the voice commands provided by the user should be performed in less than 2 seconds.	F-07	Standard	UI performance test	Through automated UI performance test the interpretation of voice commands is verified to be performed in less than 2 seconds.
PE-07	A command should be invoked within 5 seconds from the event that triggered its execution	F-08	Standard	UI performance test	Through automated UI performance test the activation of commands is verified to be performed in less than 200ms
PE-08	The maintainer must be able to access and watch a recording in less than one minute.	F-13	Standard	UI performance test	Not verified
PE-09	If requested to, the SIFIS-Home system shall contact law enforcement or private surveillance services to receive assistance in less than 30 seconds.	F-14	Optional	Event performance test. Use test mobile as receiver.	Partially verified. Notifications and communications are verified to be sent in less than 5s.
PE-10	An abnormal (suspicious) behavior caused by malware shall be identified and notified within 60 seconds	F-19	Optional	Event performance test.	Partially verified. Notifications and communications are verified to be sent in less than 5s.
PE-11	The user should be informed of the presence of malware no later than 5 seconds after the malware is recognized.	F-20	Standard	Event performance test.	Partially verified. Notifications and communications are verified to be sent in less than 5s.
PE-12	Self-healing algorithms should be started in less than 60 seconds if available when malware is recognized.	F-21	Critical	Event performance test.	Notifications and communications are verified to be sent in less than 100ms after malware identification.
PE-13	The registration of a new device should be completed in less than 30 seconds.	F-23	Standard	Event performance test.	Through automated UI performance test the registration of a new device is verified to be performed in less than 1 second.
PE-14	The list of registered devices shall be shown by the SIFIS-Home system in less than 30 seconds.	F-24	Standard	UI performance test	Through automated UI performance test the listing of registered devices is verified to be performed in less than 1 second.
PE-15	The de-registration of a device should be completed in less than 30 seconds.	F-25	Standard	UI performance test	Through automated UI performance test the

					de-registration of devices is verified to be performed in less than 1 second.
PE-16	The correct configuration changes should be propagated successfully in less than 30 seconds.	F-26	Critical	UI performance test	Through automated UI performance test the configuration changes are verified to be propagated in less than 1 second.
PE-17	The current configuration of a device should be retrieved in less than 10 seconds.	F-26	Standard	UI performance test	Through automated UI performance test the visualization of the current configuration of a device is verified to be performed in less than 200ms
PE-18	The marketplace should be accessible in less than 60 seconds.	F-28	Standard	UI performance test	Through automated UI performance test the access to the marketplace is verified to be performed in less than 2s
PE-19	The configuration of policies for groups of users should be applied and enforced in less than 60 seconds.	F-32	Critical	UI performance test	Through automated UI performance test the configuration of policies is verified to be performed in less than 200ms
PE-20	The configuration of policies for groups of devices should be applied and enforced in less than 60 seconds.	F-33	Critical	Event performance test	Through automated event performance test the configuration of policies for devices is verified to be performed in less than 200ms
PE-21	The list of policies should be retrieved in less than 30 seconds.	F-30	Standard	Event performance test	Through automated event performance test the list of policies is verified to be retrieved in less than 100ms
PE-22	The configuration of profiles should be applied and enforced in less than 60 seconds.	F-37	Critical	Event performance test	1s
PE-23	The change of current profile should be performed in less than 60 seconds.	F-38	Critical	Event performance test	1s
PE-24	The statistics about usage and behavior of devices should be presented to the administrator in less than 30 seconds.	F-41	Standard	UI performance test	Not verified
PE-25	The statistics about usage of profiles should be presented to the administrator in less than 30 seconds.	F-42	Standard	UI performance test	Not verified
PE-26	Remote log-in should be performed in less than 60 seconds.	F-43	Critical	Event performance test	1s
PE-27	In case of an incomplete or unsuccessful command execution, an error response should be sent within 5 seconds	F-08	Standard	Event performance test	Partially verified. Notifications and communications are verified to be sent in less than 5s.
PE-28	The used solutions for communication and system security shall be as much as possible lightweight to enforce in terms of performance and especially feasible also	All	Critical	By design	Verified by design of the SIFIS-Home framework architecture

	for resource-constrained devices.				
PE-29	The performance impact due to communication and system security shall not result in unacceptable impact on the user experience.	All	Critical	UI performance test	Actuation with overhead of less than 200ms.
PE-30	The network infrastructure shall provide means also for one-to-many message delivery, e.g., over IP multicast.	F-47 F-48 F-49 F-50	Critical	By design	Verified by design of the SIFIS-Home framework architecture
PE-31	It must be possible to have multiple security groups simultaneously active in the system.	F-47 F-48 F-49 F-50	Critical	By design	Verified by design of the SIFIS-Home framework architecture
PE-32	When relevant, support shall be ensured for communication intermediaries performing, e.g., message forwarding and/or (transport-) protocol translation. This applies also in secure scenarios and in (secure) group communication scenarios.	All	Critical	By design	Verified by design of the SIFIS-Home framework architecture
PE-33	When relevant, it shall be possible to enable one-to-many response messages, sent at once to multiple requesters. This applies also to secure communication scenarios and in presence of communication intermediaries.	All	Critical	By design	Verified by design of the SIFIS-Home framework architecture
PE-34	When relevant and limited to read-only operations, it shall be possible to enable cache ability of response messages at communication intermediaries, also when protected end-to-end.	All	Critical	By design	Verified by design of the SIFIS-Home framework architecture
PE-35	Devices should, if available, utilize low-power modes of operation to further mitigate the performance impact of ongoing (D)DoS attacks.	All	Standard	By design	Verified by design of the SIFIS-Home framework architecture
PE-36	There should be a means to enable an optimized, combined establishment of a cryptographic secret with a first message protected with key material derived from that secret.	All	Standard	By design	Verified by design of the SIFIS-Home framework architecture
PE-37	In case of an incomplete or unsuccessful configuration change, an error message should be returned within 5 seconds	F-26	Standard	UI performance test	Partially verified. Notifications and communications are verified to be sent in less than 5s.
RE-01	The system shall not fail more than once a week (on average).	All	Critical	Event performance test	Not testable in the scope of the project
RE-02	The system shall not take more than one day to be repaired (on average).	All	Critical	Event performance test	Repair of the system after failure is verified to be completed in less than 1h
AV-01	The SIFIS-Home system services and devices shall be available 99% of the time	All	Critical	Event performance test	Not testable in the scope of the project

AV-02	The SIFIS-Home system shall ensure basic services availability in case of system failures.	All	Critical	Event performance test	By design
AV-03	Support should be ensured for devices to dynamically react to (D)DoS attacks, by gradually adapting their availability. This includes relying on communication intermediaries for traffic offloading during intense (D)DoS attacks.	All	Standard	By design	By design
AV-04	Devices under (D)DoS attacks should be able to continue providing a (best-effort) service to legitimate requests, i.e., by displaying a graceful degradation of quality of service.	All	Standard	Event performance test	By design
US-01	The system shall be easy to use for users with no technical background	All	Critical	UI performance test	The requirement is planned to be verified once the pilot is deployed
US-02	The SIFIS-Home system shall be autonomous and learn based on the users' habits, still according to defined privacy policies.	All	Critical	By design	By design
US-03	The SIFIS-Home system shall consider special cases in its design, such as color blindness.	All	Optional	By design	By design
US-04	The SIFIS-Home system shall preserve consistency among all devices, related databases, and constraints.	All	Critical	By design	By design
US-05	The SIFIS-Home hardware components should be easy to use for the elderly and users with no engineering background.	All	Optional	Not Verifiable	Not Verifiable
US-06	The SIFIS-Home system shall have an explorable interface.	All	Standard	By design	By design
US-07	Proper and easy hardware installation should be considered.	All	Standard	By design	By design
US-08	The image-based identification through biometrics in a room (interior) or in an open space (exterior), without obstacles or face covering elements, it should be performed by the system in a radius of at least 10 meters from the device.	F-01	Standard	Event performance test	Limited by image resolution.
US-09	An untrained user should be able to understand that an attack is ongoing in less than a minute from reading the SIFIS-Home alert or notification.	F-09 F-13	Critical	UI performance test	The requirement is planned to be verified once the pilot is deployed
US-10	An untrained user should be able to recognize a software intrusion in less than one minute.	F-19 F-20	Critical	UI performance test	The requirement is planned to be verified once the pilot is deployed
US-11	An untrained user should be able to perform the device registration procedure in less than 5 minutes.	F-23	Standard	UI performance test	The requirement is planned to be verified once the pilot is deployed
US-12	An untrained user should be able to perform the device de-registration procedure in less than 5 minutes.	F-26	Standard	UI performance test	The requirement is planned to be verified once the pilot is

					deployed
US-13	An untrained user should be able to perform the configuration of devices in less than 5 minutes.	F-26	Standard	UI performance test	The requirement is planned to be verified once the pilot is deployed
US-14	An untrained user should be able to perform the installation of an application in less than 5 minutes.	F-28	Standard	UI performance test	The requirement is planned to be verified once the pilot is deployed
US-15	An untrained user should be able to complete the configuration of policies for groups of users in less than 5 minutes.	F-32	Standard	UI performance test	The requirement is planned to be verified once the pilot is deployed
US-16	An untrained user should be able to complete the configuration of policies for groups of devices in less than 5 minutes.	F-33	Standard	UI performance test	The requirement is planned to be verified once the pilot is deployed
US-17	An untrained user should be able to complete the configuration of profiles in less than 5 minutes.	F-37	Standard	UI performance test	The requirement is planned to be verified once the pilot is deployed
US-18	An untrained user should be able to perform a profile change in less than 30 seconds.	F-38	Standard	UI performance test	The requirement is planned to be verified once the pilot is deployed
US-19	An untrained user should be able to access the statistics for visualizing and interpreting them in less than 5 minutes.	F-41	Standard	UI performance test	The requirement is planned to be verified once the pilot is deployed
US-20	The Multi-Level Anomaly Detection system (MLADS) must monitor network traffic provided by several input sources and several locations.	F-15 F-16 F-17 F-18	C	By design	By design
US-21	The workload of the devices should be available to the MLADS.	F-15 F-16 F-17 F-18	C	By design	By design
US-22	The list of applications running on each device should be available to MLADS.	F-15 F-16 F-17 F-18	C	By design	By design
US-23	Raw sensor data must be available to be analyzed by MLADS.	F-15 F-16 F-17 F-18	C	By design	By design
US-24	Features from different devices should be aggregable directly or by means of pre-processing through specific analysis tools.	F-15 F-16 F-17 F-18	C	By design	By design
US-25	When possible, a dataset should not be present as a whole on a single device for analysis.	All	S	By design	By design
US-26	The presence of a GPU is needed to	F-15	S	By design	By design

	perform DL-based analysis.	F-16 F-17 F-18			
DE-01	Identification through biometrics should be performed correctly in more than 95% of cases.	F-01	Critical	Test on public datasets	Verified as reported in deliverable 4.3.
DE-02	The start of an interaction command should be recognized properly and correctly in more than 99% of cases.	F-06	Critical	Event performance test	Verified as reported in deliverable 4.3.
DE-03	The commands to execute should be recognized properly and correctly in more than 95% of cases.	F-06 F-07	Critical	Event performance test	Verified as reported in deliverable 4.3.
DE-04	Record of intrusions must be available for a configurable time (default six months) after the recording.	F-10	Standard	By design	By design
DE-05	Identity of the successfully recognized intruders must be available for a configurable time (default six months) after the recording.	F-12	Standard	By design	By design
DE-06	Core functionalities should be replicated on multiple devices to avoid single points of failure.	F-21	Critical	By design	By design
DE-07	The registration of a new device should be successful in at least 99% of the cases.	F-23	Critical	Event performance test	The operation is verified to be successful in 100% of cases if the device is properly functioning
DE-08	The de-registration of a new device should be successful in at least 99% of the cases	F-25	Critical	Event performance test	The operation is verified to be successful in 100% of cases if the device is properly functioning
DE-09	The configuration changes should be propagated successfully to the devices more than 99% of the time.	F-26	Critical	Event performance test	The operation is verified to be successful in 100% of cases if the device is properly functioning
DE-10	The SIFIS-Home system should be able to restore the previous configurations if there are errors in applying configuration changes.	F-26	Standard	Event performance test	The operation is verified to be successful in 100%
DE-11	The installation of the selected app should be completed successfully in at least 95% of cases.	F-28	Critical	Event performance test	The operation is verified to be successful in 100% of cases if the app is properly functioning
DE-12	The application of policies should always be completed successfully.	F-31 F-34 F-33	Critical	Event performance test	The operation is verified to be successful in 100%
DE-13	The configuration of profiles should be completed successfully in at least 99% of cases.	F-37	Critical	Event performance test	The operation is verified to be successful in 100%
DE-14	The change of current profile should be completed successfully in at least 99% of cases.	F-38	Critical	Event performance test	The operation is verified to be successful in 100%
DE-15	The statistics must be shown correctly in at least 99% of cases.	F-41	Critical	Event performance test	The operation is verified to be successful in 100%
DE-16	Remote log-in for the configure should be successful in at least 99% cases.	F-43	Critical	Event performance test	The operation is verified to be successful in 100% if

					network is available
DE-17	The SIFIS-Home system should be able to distribute the processing among multiple machines in separate places if required.	All	Critical	Event performance test	The operation is verified to be successful in 100% if network is available
DE-18	The SIFIS-Home system is required to be fault tolerant, it should continue to operate, even if one or more of the nodes fail.	All	Critical	Event performance test	The operation is verified to be successful in 100% if network is available
DE-19	The SIFIS-Home system is required to be scalable dynamically by adding or removing nodes according to demand.	All	Critical	Event performance test	The operation is verified to be successful in 100% if network is available
TE-01	The SIFIS-Home system needs Java version 8 or higher to interact with the ontology.		Critical	By Design	By Design
TE-02	The process for getting and inserting information into the ontology will be through APIs provided via HTTP(S).		Critical	By Design	By Design
TE-03	The software for handling the ontology should be hosted on a high-availability server.		Critical	By Design	By Design
TE-04	Internet connectivity should be present.		Standard	By Design	By Design

Table 2 Finalized list of Non-Functional requirements for the SIFIS-Home framework.

Table 3: Validation for the security requirements

Req ID	Requirement Description.	FR	Testable	Priority	Validation
SE-01	APIs for the communication with internal devices must be secured.	C-02	NT	Critical	-
SE-02	APIs for the communication with external devices must be secured.	C-04	NT	Critical	-
SE-03	Personal data stored must be encrypted.	F-49 F-56 S-04	T	Critical	By design
SE-04	The system shall protect and avoid disclosure of sensitive information.	F-56 F-62 S-04	NT	Critical	-
SE-05	The SIFIS-Home system shall prevent data alteration or deletion.	F-56 F-61 S-04	NT	Critical	-
SE-06	Wifi access should be protected against known WiFi security attacks.	C-01 C-03	T	Critical	By design
SE-07	Biometrics must be stored safely in the SIFIS-Home database.	F-03	NT	Critical	-
SE-08	Log-in information should be stored in a protected database.	F-62	NT	Critical	-
SE-09	The information about the registered devices, their characteristics and their configurations should be stored in a protected database.	F-25 F-38 F-44	NT	Critical	-
SE-10	The information about policies should be stored in a protected database.	F-33	NT	Critical	-
SE-11	The information about user profiles and configuration aspects should	F-39	NT	Critical	-

	be stored in a protected database.	F-42 F-44			
SE-12	Data paths should be identified to allow data tracking and detect data leaving the smart-home perimeter, according to policies.	General	T	Critical	By design
SE-13	Data confidentiality shall be ensured all the time.	General	NT	Critical	-
SE-14	The system should not be affected by MITM attacks.	General	T	Critical	By design
SE-15	Software and apps shall only be installed with authorisation of the smart home administrator or resident users.	General	T	Critical	-
SE-16	Users must be able to configure and allow the usage of data from the SIFIS-Home framework and third- party software.	General	T	Critical	By design
SE-17	Anomalous device behaviours should be identified and signalled in less than 60 seconds.	General	T	Critical	Verified via dedicated analytics.
SE-18	Minimum needed privilege principle must always be enforced.	General	NT	Critical	-
SE-19	Access to devices functionalities should be protected and controlled	General	NT	Critical	-
SE-20	Access to critical functionalities and services of the SIFIS-Home framework shall be protected and controlled.	General	NT	Critical	-
SE-21	Privacy preferences shall be configurable for data, analytics and functionalities.	General	NT	Critical	-
SE-22	Analytics shall be able to work with anonymized data when possible.	General	NT	Critical	-
SE-23	The SIFIS-Home architecture shall be resilient to network-based attacks.	General	T	Critical	Verified through dedicated analytics.
SE-24	The SIFIS-Home architecture shall be resilient to DoS attacks.	General	T	Critical	Verified through dedicated analytics.
SE-25	The SIFIS-Home architecture shall be resilient to sybil attacks.	General	T	Critical	By Design
SE-26	The SIFIS-Home architecture shall be resilient to device compromising attacks.	General	T	Critical	Verified through Node Manager
SE-27	The SIFIS-Home architecture shall be resilient to Internet connection failure.	General	T	Critical	By Design
SE-28	The SIFIS-Home architecture shall be resilient to physical device damage or failure.	General	T	Critical	By Design
SE-29	Devices must have unique identifiers.	General	NT	Critical	-
WP3 - SE-30	Unless thoroughly assessed and acceptable for the specific application, communications in the networked environment shall be secured, by ensuring confidentiality/integrity/authenticity of messages, as well as protecting from replay protection.	General	T	C	By Design
WP3 - SE-31	It shall be possible and feasible to provide devices with the necessary key material to establish their security associations and to communicate securely, with preference for automatic procedures.	General	T	C	By Design
WP3 - SE-32	It shall be possible to achieve end-to-end protection of CoAP messages at the application layer, by ensuring confidentiality/integrity/authenticity of messages, as well as protecting from replay protection. This applies also in case communication intermediaries are used, as well as for both one-to-one and one-to-many (group) communication.	General	T	C	By Design
WP3 - SE-33	Cryptographic binding between a protected request message and one or many corresponding protected response(s) shall be ensured.	General	T	C	By Design

WP3 -SE-34	Source authentication of protected messages shall be ensured, also in a group communication setup where one-to-many messages are exchanged.	General	T	C	By Design
WP3 -SE-35	Cryptoagility shall be ensured, as a way to allow a seamless possible switch to different existing algorithms as well as a seamless possible migration to future algorithms.	General	T	C	By Design
WP3 -SE-36	Operations related to the creation, configuration, deletion, registration and discovery of security groups shall be secured and shall be allowed only to authorized entities.	F-23 F-25 F-26 F-30 F-31 F-32 F-33 F-34 F-35 F-47 F-48 F-49	T	C	By Design
WP3 -SE-37	When relevant, it shall be ensured that a possible communication intermediary can securely identify its adjacent communication hops.	General	T	C	By Design
WP3 -SE-38	It shall be ensured that possible secure cacheable response messages do not break security properties that are critical for the application and specific communication exchanges.	General	T	C	By Design
WP3 -SE-39	Devices should be able to detect ongoing (D)DoS attacks based on intensity and distribution of invalid traffic.	General	NT	S	-
WP3 -SE-40	The system shall provide a means to enforce flexible, fine-grained and reactive authorized access control for devices to access remote resources at other devices.	General	T	C	By Design
WP3 -SE-41	It shall be possible to establish security material to use for end-to-end secure (group) communication in an authorized way, achieving confirmation of the established material.	General	T	C	By Design
WP3 -SE-42	The system shall provide a means for enabling devices to get agile and possibly automatic notification, in order to signal pertaining access credentials that have been revoked while still unexpired.	General	T	C	By Design
WP3 -SE-43	There shall be means for two devices to securely establish a new cryptographic secret with perfect forward secrecy, while also achieving mutual authentication and confirmation of the established material.	General	T	C	By Design
WP3 -SE-44	There shall be an authorization-based means to securely join/leave a security group and retrieve/provide updated key material to communicate in the group.	F-23 F-25 F-26 F-30 F-31 F-32 F-33 F-34 F-35 F-50	T	C	By Design
WP3 -SE-45	There shall be a means to securely renew the key material in a security group, both periodically and in case the application requires backward/forward security.	F-19 F-23	T	C	By Design

		F-25 F-26 F-50			
WP3 -SE-46	When limits on usage of cryptographic material for encryption and decryption are exceeded, devices owning that key material shall stop using it and specific actions shall be taken to acquire new material before possibly resuming communication. The just invalidated key material may be temporarily retained and used only for processing incoming messages for a limited, pre-configured amount of time.	General	T	C	By Design
WP3 -SE-47	There shall be a means for two devices to securely update their pairwise key material.	General	T	C	By Design
WP4-SE-01	Device administrable domain should be known.	all	NT	S	-
WP4-SE-02	Definition of a template for each type of device which describes the features of the specific type of device.	all	NT	S	-
WP4-SE-03	The identity of the speaker should not be identifiable if the analysis is outsourced to external services.	UC-02	NT	S	-
WP4-SE-04	The background noises in the audio streams must be anonymized if the analysis is outsourced to external services.	UC-02	NT	S	-
WP4-SE-05	Personal information recognizable from audio (e.g., name, telephone number, email address, age, physical condition) must be anonymized if the analysis is outsourced to external services.	UC-02	NT	S	-

A number of requirements are verified by design, thanks to the SIFIS-Home framework and/or SIFIS-Home architecture design that ensure by construction all requirements concerning confidentiality and integrity of data. Validation of some usability requirements has been moved to WP6, since involvement of non-trained users in the tests will be performed directly on the use case.

8 Conclusion

In this deliverable, we have presented the final design, implementation, and integration of the SIFIS-Home security architecture, along with the latest verification and validation results. The implementation and integration consisted in a comprehensive and extensive effort, leveraged state-of-the-art technologies and tools.

The allocation of the SIFIS-Home framework components, completed in 2021 and revised in 2022, has proven to be a vital decision. The collaboration with the right partners for each component has been instrumental in reaching this point. All partners have worked diligently together to complete the necessary implementation and testing tasks, address challenges, and resolve issues that arose during the process. This includes technical aspects, communication interfaces, security, access rights, data flow, as well as the processes for the software continuous integration and continuous deployment.

The verification and validation activities, including unit testing and system testing in the test beds, required detailed coordination and clarification of expectations among all the partners. The success of the third iteration of the SIFIS-Home security architecture defined in deliverable D1.4 served as the backbone for both the implementation and deployment of the test beds and of the actual security architecture.

The utilization of the Distributed Hash Table (DHT) technology, which forms the core of the SIFIS-Home network, has been proven suitable for creating distributed and self-healing networks.

In summary, based on our self-assessment, verification and validation results, we have succeeded in implement, integrate, deploy, verify and validate the SIFIS Home security architecture with help of the test beds. The next step will be to finalize the pilot in WP6.

9 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/

[ANSIBLE] Continuous integration and delivery with Ansible, https://www.redhat.com/en/engage/delivery-with-ansible-20170906

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

[FIWARE NGSI v2 API] FIWARE NGSI v2 API definition: https://swagger.lab.fiware.org/

[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