D1.4

Final Component, Architecture, and Intercommunication Design

WP1 – Distributed System Architecture

SIFIS-Home

Secure Interoperable Full-Stack Internet of Things for Smart Home

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

Responsible partner: FSEC
Editor: Marko Komssi;
E-mail address: marko.komssi@f-secure.com

29/09/2021
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: Riccardo Coppola (POL), Andrea Saracino (CNR), Domenico De Guglielmo (DoMO), Håkan Lundström (SEN), Luca Barbato (LUN), Joni Jämsä (CEN), Olli Isohanni (CEN), Ossi Saukko (CEN), Otto Waltari (FSC), Marko Komssi (FSC), Marco Tiloca (RISE).

Approved by: Håkan Lundström (SEN), Elina Hirvonen (CEN)

Revision History

Version	Date	Name	Partner	Section Affected Comments
0.1	01/06/2022	ToC Defined	FSEC	All
0.2	7/08/2022	Inserted content from D1.3	CNR, FSEC, POL	All
0.3	20/08/2022	Definition of the DHT	DOMO	Section 3
0.4	25/08/2022	Update of the API	POL	Section 5
0.5	2/09/2022	Workflows definition	CNR, POL, DOMO, LUM, FSEC	Section 6
0.6	5/09/2022	Upgrade to D1.3 contents	All	All
0.7	15/09/2022	Ready for Review	All	All
1.0	29/09/2022	Ready to submit	All	All

Executive Summary

This deliverable reports the final design of the SIFIS-Home architecture and the SIFIS-Home framework. The SIFIS-Home architecture is the logical representation of the SIFIS-Home aware devices and their interaction in the smart home. The components of the SIFIS-Home architecture are Smart Devices and Not So Smart Devices. The former includes devices that either has or has not a network interface which enables connectivity outside of the smart home. Internet connected smart devices have a direct interface to send and receive network traffic out of the cyber perimeter, outside of Smart Home. SIFIS-Home architecture involves six main actors, such as SIFIS-Home Administrator and SIFIS-Home Tenant. SIFIS-Home framework recognizes each actor, for instance, by means of the usage, security, privacy and safety policies.

The SIFIS-Home framework is the software architecture installed in the devices of the SIFIS-Home architecture, used to provide services and manage safety, security and privacy aspects in the smart home. The architecture of the SIFIS-Home framework consists of five high-level parts that are SIFIS-Home Smart Device Framework, SIFIS-Home Application Framework, SIFIS-Home NSSD Framework, SIFIS-Home Cloud Framework and SIFIS-Home Development Tools. Each part represents a building block that follows microservices design pattern. Moreover, the higher-granularity architectural details are provided for more complex sub-components that can be decomposed furtherly.

The deliverable presents a specific set of APIs for the SIFIS-Home architecture and calls them SIFIS-Home APIs. One such example is Communication APIs that are used to retrieve logs, alerts and messages generated by the SIFIS-Home system. Another example is Application Manager APIs that provides operations to manage third-party applications.

Finally, this deliverable describes operative workflows related to the operations that will be supported by the SIFIS-Home framework. Each operative workflow is presented with a process flow diagram to illustrate the sequential flows of activities between the components of SIFIS-Home framework. Both the SIFIS-Home framework and the SIFIS-Home architecture are input for the activities of WP5, which will implement and deploy the testbed of the SIFIS-Home architecture.

Table of contents

Contents

1 Introduction

To manage security, privacy and safety in the smart home environment, the SIFIS-Home project deploys a software framework intended to run on the smart home devices, which can be customized by installing third party applications. According to the requirements elicited in the deliverables D1.1 and D1.2, the SIFIS-Home framework must be resilient, which implies the replication of functionalities and fault tolerant communications. Thus, the core of SIFIS-Home is based on a peer-to-peer (P2P) architecture, still a smart-home is a heterogeneous environment, where some devices that can be customized interact with a set of devices with limited possibilities of customization and functionalities. To represent such heterogeneity and the interaction between the devices and users, we also define the SIFIS-Home architecture, defining all the actors and their interactions.

In this deliverable, on one hand, we report the description of the SIFIS-Home architecture, also by presenting the concept of the Smart Home cyber-perimeter and some background information on the DHT model that will be used to implement the SIFIS-Home communication protocol. On the other hand, we describe in detail the final architecture of the SIFIS-Home framework, describing at first a high-level view of the architecture, followed by a detailed view of the components and their interactions. The SIFIS-Home framework has been designed following a top-down approach, as discussed in D1.1. The design pattern followed is the microservices design pattern, which favours both flexibility and modularity. Accordingly, the deliverable reports the architectural components in three levels of detail. Finally, the deliverable will report the SIFIS-Home APIs and the related operative workflows.

This deliverable D1.4 extends and complements the preliminary architecture of SIFIS-Home Framework as well as the preliminary set of APIs described in deliverable D1.3. WP1 and WP5 have co-operated in the architecture design and implementation. Both the SIFIS-Home framework and the SIFIS-Home architecture are input for the activities of WP5, which will implement and deploy the testbed of the SIFIS-Home architecture. D1.3 provided an input to the activities of WP5. Likewise, the initial implementation and deployment of the SIFIS-Home framework and architecture provided feedback to D1.4. The feedback has been used to complete the operative workflows of main SIFIS-Home operations together with the finalized list of the implemented APIs and components. The complete list of the changes between D1.3 and D1.4 are presented in Appendix A.

The deliverable is organized as follows. Section 2 of this deliverable introduces the SIFIS-Home architecture, while Section 3 introduces the SIFIS-Home framework. Section 4 describes mapping between the defined functional requirements and SIFIS-Home architecture. Section 5 presents the APIs with examples. Section 6 introduces a number of operative workflows of key SIFIS-Home operations and Section 7 concludes the deliverable.

2 SIFIS-Home Architecture

The *SIFIS Home architecture* is the representation of the devices and actors interacting with the *SIFIS-Home Framework.* More in details the architecture depicts at a logical level the devices that are present in a SIFIS-Home aware smart home, their interconnection and interactions.

2.1 *The Smart Home Cyber-Perimeter*

Protecting the Smart Home and its users from unintended disclosure of sensitive information requires defining a logical distinction between the outside and inside of the Smart Home. A concept that we are defining in the scope of SIFIS-Home, which will be relevant to define the SIFIS-Home architecture and for defining data privacy policy, is the Smart Home *Cyber-Perimeter*. The concept is illustrated in Figure 1.

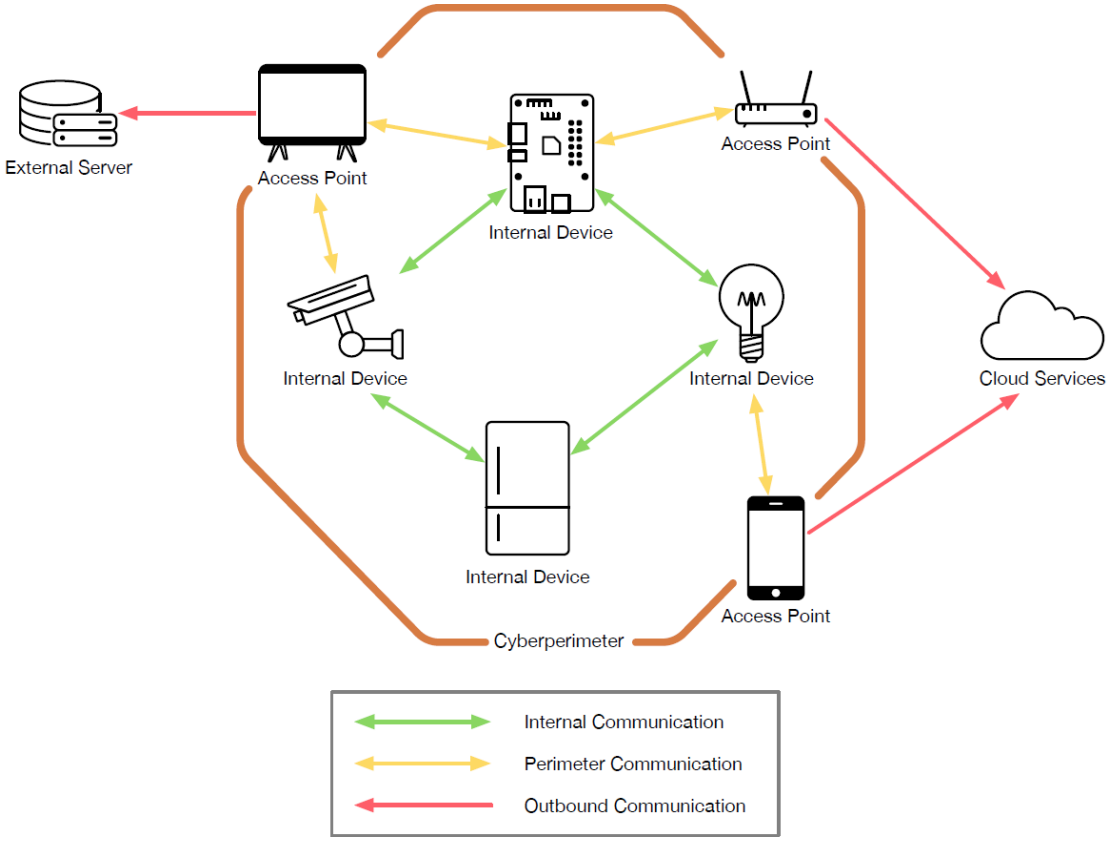

Figure 1: Concept of Smart Home Cyber-Perimeter

The Smart Home Cyber-Perimeter is the separation between the two domains: outside or inside of the Smart Home. The cyber-perimeter is a logical barrier, which identifies the elements (i.e., devices and application) of the Smart Home, which can be used to receive and send data toward entities that are not part of the Smart Home (i.e., external entities). For example, a smart speaker with an embedded voice assistant exploiting a cloud service to process voice commands is both an access (since it receives instructions from the cloud) and exit (since it sends data to the cloud) point of the Smart Home cyber-perimeter. Though even inside the Smart Home cyber-perimeter, there might be specific privacy constraints, a violation of the Smart Home privacy is performed when sensitive information leaves the Smart Home cyber-perimeter. Figure 2 illustrates the actors and elements of the SIFIS-Home architecture their relation to Smart Home Cyber-Perimeter.

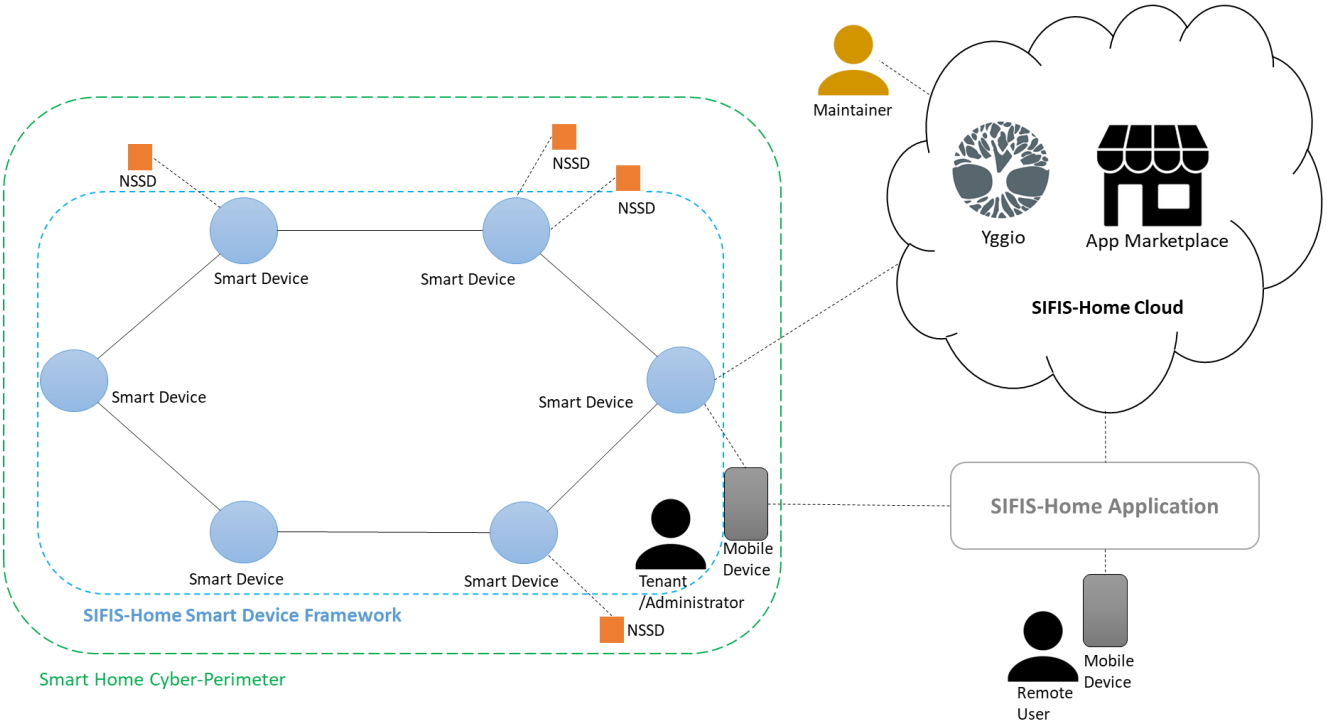

Figure 2: Actors and Elements of the SIFIS-Home Architecture

We base this definition of privacy violation on the worst possible case: once a piece of data leaves the Smart Home cyber-perimeter, the users potentially lose control of that data piece, which can thus be re-used and redistributed indefinitely. We derive that data can be safely exchanged among devices and services inside the Smart Home cyber-perimeter. The rationale behind this distinction is in the trade-off between ensured privacy and needed accuracy for data analysis algorithms, which are essential to provide smart services. Thus, inside the Smart Home cyber-perimeter, data can be exchanged and processed without applying privacy-enhancing techniques to maximize data analysis accuracy, providing the best service level. In fact, inside the cyber-perimeter, data cannot be shared with external entities and remains only available to the data owner, i.e., the Smart Home residents. The validity of this assumption depends on the devices and applications behaviour on the cyber-perimeter, which act as access points to the Smart Home. Their behaviours should be monitored and certified, when possible, to ensure that when they have access to sensitive information, they are not going to send them outside out of the cyber-perimeter. When this cannot be ensured, or it is known that a data piece is bound to leave the perimeter, it should be processed through specific privacy-enhancing techniques to avoid disclosing sensitive information.

2.2 *Components and Actors of the SIFIS-Home architecture*

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

- *Smart Devices:* These devices are characterized by a relatively good computational capability; they are based on general purpose computational hardware and their functionalities are managed through an Operative System (OS). Smart devices can be customized by installing third party software and have the capability of directly communicating among them, autonomously exchanging information. This intercommunication enables a Peer-to-Peer (P2P) logical model, which is easily represented by means of a distributed hash table (DHT). Example of Smart Devices are Smart TVs, Smart Refrigerators, Laptops/Desktops, Family Hubs.
    - *Internet Connected Smart Devices*: This is a subset of the Smart Devices which are

characterized by the presence of a network interface which enable connectivity outside of the smart home. Example of these devices are smart routers (connected to optical fiber or DSL), smartphones and tablets (with 4G/5G connectivity). Internet Connected Smart Devices are in general on the Smart Home cyber-perimeter, as they have a direct interface to send network traffic out of the cyber perimeter.

- *Not So Smart Devices (NSSD):* This set is made by those devices which present smart functionalities and present a network interface, yet they still have very limited computational power, and only present a firmware instead of a fully-fledged OS. For this reason, the NSSDs cannot be customized by installing third party software or applications. Examples of NSSDs are smart sensors, smart cameras, smart lights, smart speakers, smart locks. NSSDs can have Internet capabilities, but generally the Internet connection will be forced to happen through their responsible Smart Devices.

Figure 2 and Figure 3 illustrate the communication and interaction between components and actors of SIFIS-Home architecture. The Smart Devices are at the core of the SIFIS-Home architecture, since they will be the only devices installing the SIFIS-Home framework. Thus, the minimal *instance* of a SIFIS-Home architecture is the one made by a single smart device. When more than one smart device is added to a SIFIS-Home architecture instance, we consider them as logically interconnected. The interconnection and network communication are handled at application level by a DHT protocol. In SIFIS-Home we are using the Kademlia protocol as DHT, which is detailed in the next subsection. The DHT abstracts from the actual network implementation, i.e., the smart devices can be physically connected to the same Wi-Fi network, be connected to different Wi-Fi hot spots, or use ad hoc Wi-Fi routing protocols such as AODV [Perkins, 1999]. The SIFIS-Home architecture is, thus, oblivious of the actual network technology and topology, which makes it adaptable to any network configuration. The DHT allows to view the SIFIS-Home architecture from outside as a single entity, accepting requests from external services and applications as if it was a monolithic server. Following the DHT protocol, Smart Devices handle in a distributed way computational task, data storage, message forwarding, attribute retrieval and data analysis, implementing thus the services of the SIFIS-Home framework.

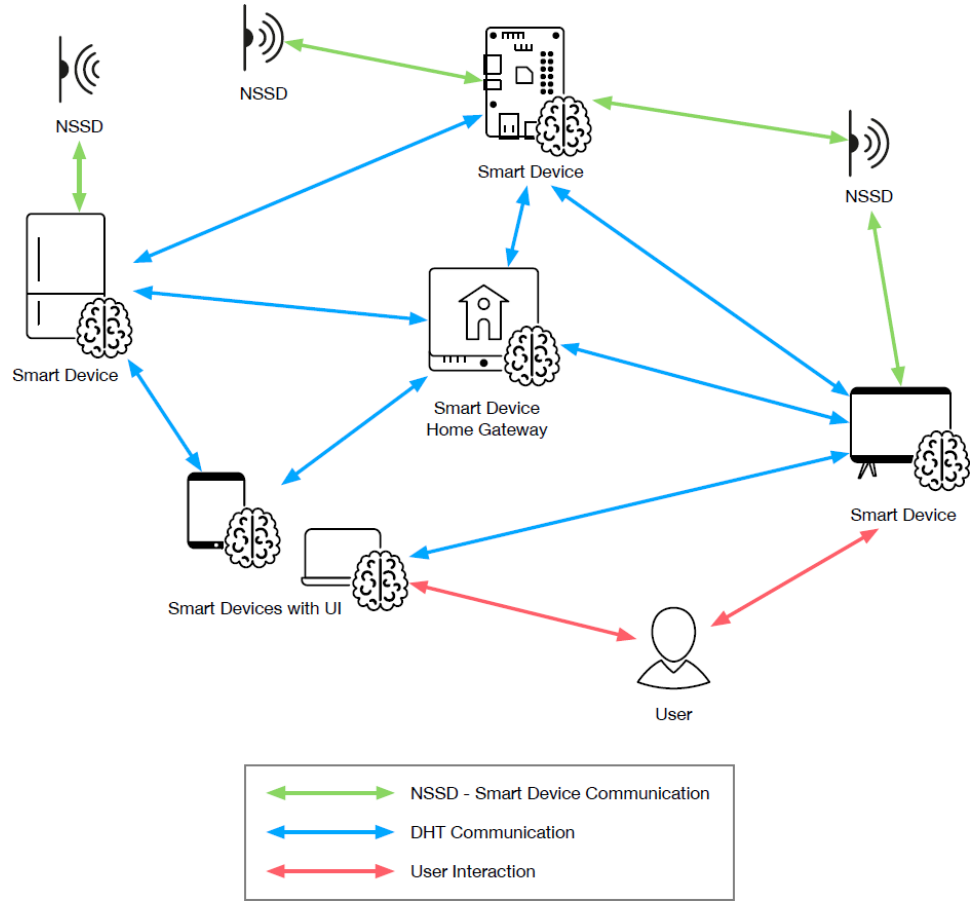

Figure 3: Communication and interaction between components and actors of SIFIS-Home architecture

Whilst smart devices manage framework operations and message exchange, the NSSDs provide the SIFIS-Home architecture with additional capabilities to read physical measures through sensing mechanisms, and to actively interact with the physical world through actuators. NSSDs are thus considered as peripheral devices under the direct control of one or more smart devices. When registered, a smart device connects via Wi-Fi, Bluetooth, or 802.15.4 protocols to one or more smart devices (number and topology will be selected by the SIFIS-Home administrator) and will only accept messages and commands from these devices. Each NSSD offers thus a set of APIs to receive queries on sensed data, or to perform operations, which are issued by the responsible smart devices. A very preliminary implementation of a P2P architecture of Smart Devices in a smart home environment has been presented in [La Marra et. al, 2017].

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

- *SIFIS-Home Administrator*: The administrator is a human user who is the owner of an instance of the SIFIS-Home architecture. Generally, this person coincides with the smart home owner. The administrator defines usage policies, assign roles to other tenants or guests of the home, sets up restrictions, application preferences, smart home routines and possible configuration. Since SIFIS-Home is human centred, the administrator is considered the highest authority, who can supersede at any time the SIFIS-Home framework decisions. The administrator can install third party applications on the framework and remove or change the authorizations to applications installed by other users.
- *SIFIS-Home Tenant*: The SIFIS-Home tenant is the standard user of the smart home. The tenant is a resident of the smart home and the main target of the SIFIS-Home system. This user

can set up preferences and configurations which should not be in contrast with those set up by the administrator. The tenant can give command to the smart home devices via voice command or via a user interface on devices like smartphone, tablet or PC, and he can install applications on the smart devices which are not in contrast with the administrator policies. The user can remove the applications he installed. Each tenant will have his or her own *profile*, describing usage preferences and configuration, which cannot be modified by other tenants. Only the administrator can modify the profile of another tenant. Generally, the administrator is also a tenant. Moreover, each tenant can define and configure one or more *usage modes* (e.g., "do not disturb"), which can be switched several times during the usage of the platform.

- *SIFIS-Home Maintainer:* The maintainer is an entity external to the smart home which is trusted by the administrator to correctly configure the smart home security, privacy and safety policies. In a commercial model, the maintainer can be the provider of the SIFIS-Home framework, holding the same right of the administrator. The maintainer can also install applications for handling specific management services, and they can remove applications installed from any user. The maintainer is not a mandatory actor, but when present has to be considered a trusted party.
- *SIFIS-Home Tenant with restrictions*: This user is a smart home tenant with restrictions on the functionalities they can use. The restrictions are needed to avoid possible damage or hazard to the tenant or to the home devices. Typical example of these tenants are children, who can still interact with the smart home and beneficiate from services, still they cannot use dangerous functionalities (e.g., turning on a stove). As the other tenants, they can install applications, however the set of available applications can be limited according to specific safety policies. As the other tenants, they can have a personal profile and their own usage policies. However, these policies are generally set by the administrator.
- *Guest*: A guest is a smart home user who is not resident in that smart home but is accessing and using the premises for a limited amount of time, upon authorization of the administrator or another tenant. Guests do not have profiles, nor they can set up policies, still they can use a subset of the house functionality. The services and functionality available are set by the administrator for all guests. Guests cannot install applications.
- *External Operator*: The external operator could be a technician, a plumber, gardener, or house maid, accessing the house for a limited amount of time, with the authorization of a tenant. Differently from guests, the operators will not use the smart home functionalities, still they might have specific policies needed to protect their privacy.

The SIFIS-Home actors are the entities defined and recognized by the SIFIS-Home framework. Thus, they can be used as subjects for the usage, security, privacy and safety policies.

As shown in Figure 2 apart from the smart home cyber-perimeter, including smart devices and NSSDs, on which the SIFIS-Home Smart Device and SIFIS-Home NSSD frameworks are integrated, there are two additional software components installed on external devices, namely the *SIFIS-Home Cloud* and the *SIFIS-Home Application*.

3 SIFIS-Home Framework

The design of the SIFIS-Home framework has been based on the *microservices* design pattern. In fact, to design the SIFIS-Home framework we have taken in consideration the requirements described in D1.1 and D1.2.

The final architecture of the SIFIS-Home Framework described in the present deliverable extends and complements the preliminary architecture described in deliverable D1.3. The SIFIS-Home framework architecture has been defined by following a top-down approach. By using microservices is possible to define a modular architecture where each component offers a specific set of functionalities, which can be invoked either by other architectural components, or externally. In the following we provide a view of the SIFIS-Home architecture as a whole and we will then analyse the functionalities of each subcomponent.

The high-level architecture of the SIFIS-Home framework is presented in Figure 4. These high-level architecture consists of five parts that are:

- **SIFIS-Home Smart Device Framework:** The SIFIS-Home Smart Device Framework is the set of software components that are executed on the Smart Devices (SD) present in the smart home. Refer to section 2.2 of the present document for the definition of Smart Device.
- **SIFIS-Home Application Framework:** The SIFIS-Home application Framework is the set of software components that are installed on a mobile device (smartphone) that is used to control the smart home. The SIFIS-Home Application provides a Graphical User Interface to the different typologies of users that will utilize the smart home.
- **SIFIS-Home NSSD Framework:** The SIFIS-Home Not-So-Smart Device Framework is the set of software components that are executed on the Not-So-Smart Devices (SD) present in the smart home. Refer to section 2.2 of the present document for the definition of Not So Smart Device.
- **SIFIS-Home Cloud Framework:** The SIFIS-Home Cloud Framework is the set of software components and applications that reside on the SIFIS-Home cloud that are mainly used to allow a user to control the smart home from a remote side.
- **SIFIS-Home Development Tools:** it is the set of developer tools that have been developed in the context of WP2. They are expected to be executed on the PC of an external developer.

In this high-level architecture, we separate the software components according to the physical device or system on which they are deployed. In particular, the software components can reside on the SIFIS-Home Cloud, on a smart device, on a Not So Smart Device (NSSD) or on the mobile device used by the user to control its house. In the next sections we detail these five parts.

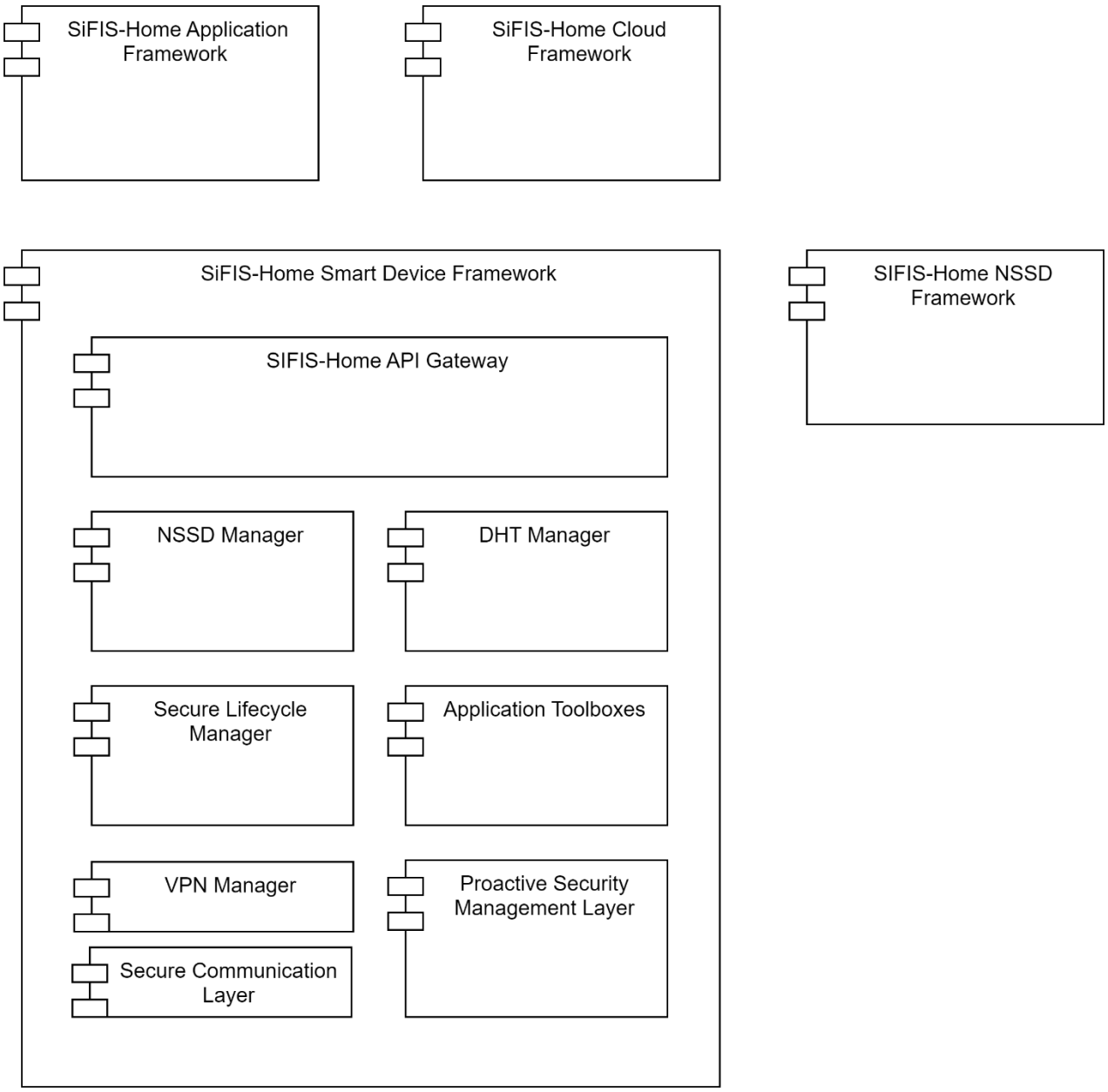

Figure 4: High-level architecture of the SIFIS-Home framework

Figure 5 provides a more detailed architecture and illustrates the main components of each building block. The following subsections expand the previous component diagrams by describing the internal components in which each of the high-level modules are divided, when a finer-grained decomposition is needed. Higher-granularity architectural details are provided for more complex sub-components that can be decomposed furtherly.

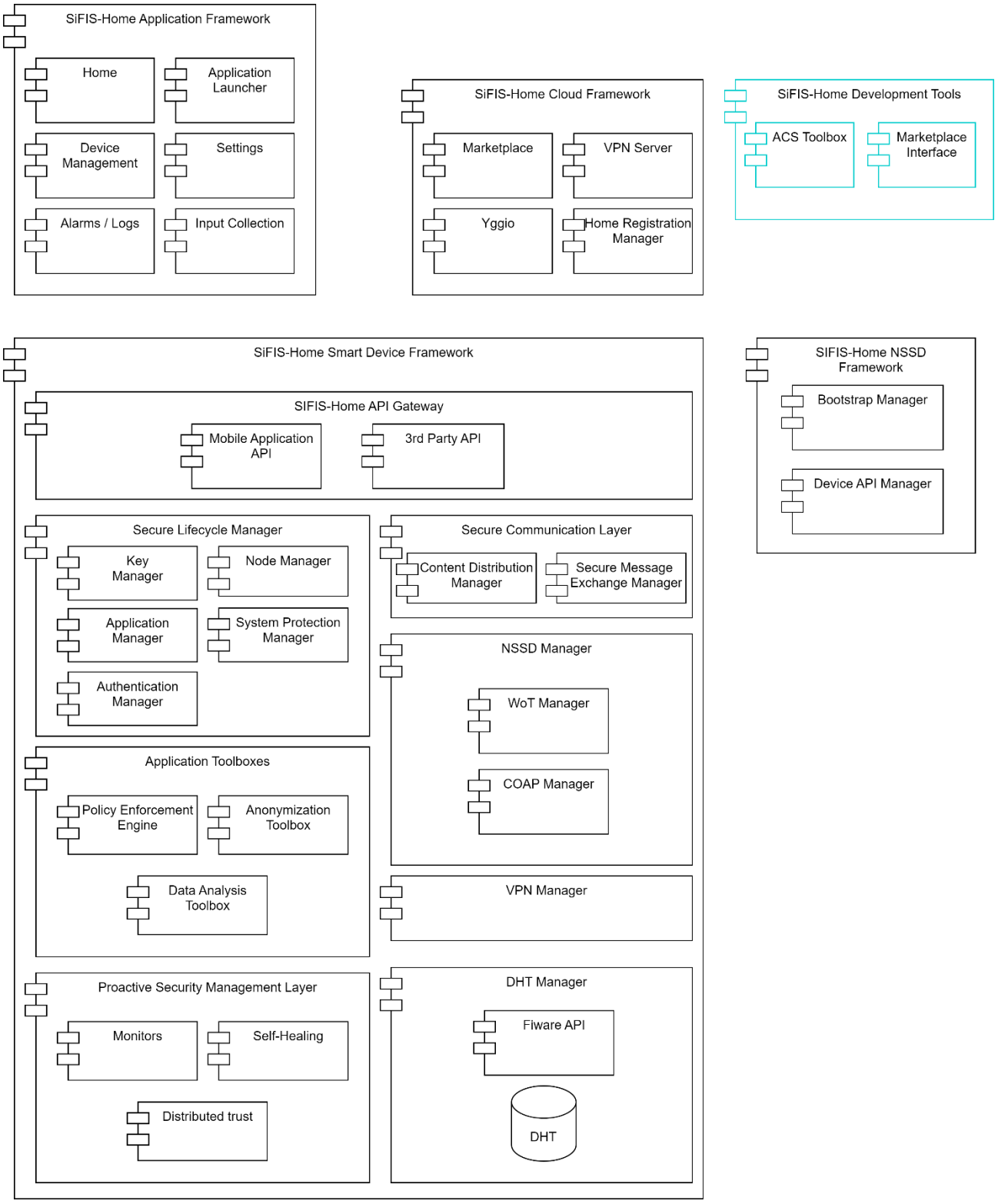

Figure 5: Detailed view of the SIFIS-Home framework, Cloud and Application

3.1 *SIFIS-Home Smart Device Framework*

The SIFIS-Home Smart Device Framework is the set of SIFIS-Home software components that are executed on all smart devices present in the smart home. The SIFIS-Home Smart Device Framework is composed of a set of macro-components providing the following functionalities:

- **SIFIS-Home API Gateway**: this component includes a set of high-level APIs and has the purpose of interfacing the SIFIS-Home Smart Device framework with external applications. In particular, it offers APIs that are used by the 3rd party applications and the SIFIS-Home control application.
- **Secure Lifecycle Manager**: this component is responsible of the bootstrap of a smart device. Also, it takes care to remove misbehaving nodes and applications. Also, it allows to install 3rd party applications on the smart devices of the smart home.
- **NSSD Manager**: it manages the communication with the NSSD devices.
- **Application toolboxes**: it is the set of software components that take care to perform data analytics and operations such as policy enforcement and data anonymization.
- **Proactive security management layer:** this component handles automatically all tasks related to management of safety and security, such as intrusion and anomaly detection, by triggering correction measures to stop or mitigate such misbehaviours.
- **DHT Manager**: it provides a mechanism through which all the different software components can interact following a publish/subscribe approach. Also, it takes care to communicate with the SIFIS-Home cloud to allow offering a FIWARE compliant API for a SIFIS enabled smart home.
- **VPN Manager**: all smart devices to a VPN server executing on the SIFIS-Home cloud. Its role is to allow access to the services offered by the smart devices from a remote side and, hence, allow users to control their smart home when outside.

In the following we describe every macro-component in detail.

3.1.1 SIFIS-Home API Gateway

The SIFIS-Home API gateway (design of which is reported in Figure 6) is the component that allows interaction of 3rd party applications and the SIFIS mobile application with the smart home. It is composed of two different components named Mobile Application API and 3rd Party API that we describe in the following.

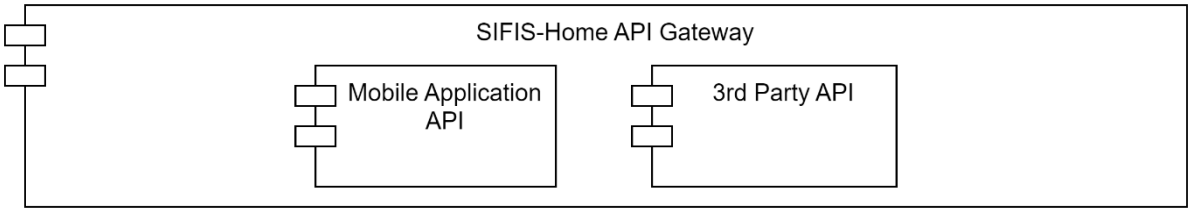

Figure 6: The SIFIS-Home API gateway

- **Mobile Application API:** this component is responsible for allowing the SIFIS mobile application to interact with the smart home. In particular, it offers APIs providing all the information needed by the control application. It provides APIs to install third-party applications on the smart devices, to retrieve the list of all the devices that are inside the network, to allow to change policies and settings of the smart home, to get alarms and logs from the system.

- **3rd Party API:** this component provides a set of APIs in the form of application libraries to

be used by part of 3rd party applications. The APIs will provide access to the information of the smart home (e.g. the set of lights of the smart home, the list of settings) and will also allow the 3rd party application to execute commands (e.g. turn on/off a certain light/appliance). The 3rd party API component communicates with the policy manager to check whether a certain action requested by an application is allowed or denied. Also, it communicates with the NSSD Manager component to control and interact with the physical devices of the smart home.

The set of exposed APIs is reported in Section 5 of the present deliverable.

3.1.2 Secure Lifecycle Manager

Figure 7 presents Secure Lifecycle Manager that is a core module of the SIFIS-Home framework. This module acts as orchestrator of the framework lifecycle, regulating presence and behaviour of both Smart Devices and applications. In particular, the Secure Lifecycle Manager enables and handles new device registration, as well as un-registration. Moreover, it manages installation and removal of third-party applications, according to the received instructions from the user or from the policy engine and the intrusion detection system.

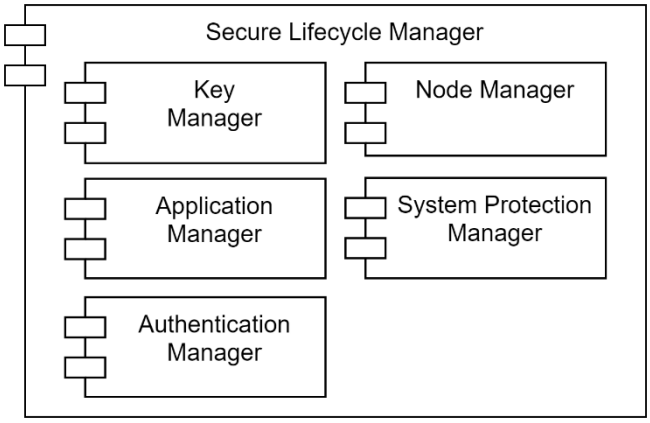

Figure 7: Architecture of the Secure Lifecycle Manager

The Secure Lifecycle Manager module includes the following components:

- **Node Manager**: the node manager registers and unregisters devices in the DHT allowing them to participate to the communication among the Smart Devices. In detail, it continuously checks if a certain smart device is misbehaving. If this is the case, it starts a DHT rekeying operation that forbids that smart device to access the DHT now on. The Node Manager interacts with the Device Registration Manager to handle the registrations of smart devices and with the System Protection Manager to get information on the DHT activities and trigger the removal of nodes which are misbehaving.
- **System Protection Manager**: this component acts as interface between the Proactive Security Management Layer and the other components of the secure lifecycle manager. More in details, the System Protection Manager regulates the collection of behavioural data from nodes and applications and forwards them for intrusion detection analysis. Thus, it receives commands which are dispatched to the application manager and node manager to block suspicious behaviours of both nodes and applications.

- **Application Manager**: the application manager is a component which is in charge of monitoring and controlling the behaviour of the third-party applications installed in the SIFIS-Home architecture. This component has the needed functionalities to install and remove applications, as well as interrupting their execution. Moreover, the Application Manager collects information on the application behaviour, related to their execution, the API they are invoking, the used resources and the invoked system calls. The application manager can launch an application execution, and interacts with the Marketplace (that resides on the SIFIS-Home cloud) to download and install the desired applications. Also, it communicates with the System Protection Manager to receive commands to halt and remove misbehaving applications.
- **Key Manager**: This component provides functionalities for requesting, establishing, updating and managing security material and credentials used to enforce secure communications and protection of message exchanges in the SIFIS-Home network.
- **Authentication Manager**: This component provides functionalities for requesting, updating, validating, consuming and managing security credentials used to enforce authentication, authorization and access/usage control of resources and services in the SIFIS-Home network.

3.1.3 NSSD Manager

The NSSD Manager (design is reported in Figure 8) is the component that allows the management of the Not-so-smart devices that are part of the smart home. It is composed of two different components, named WoT Manager and COAP Manager, that we describe in the following.

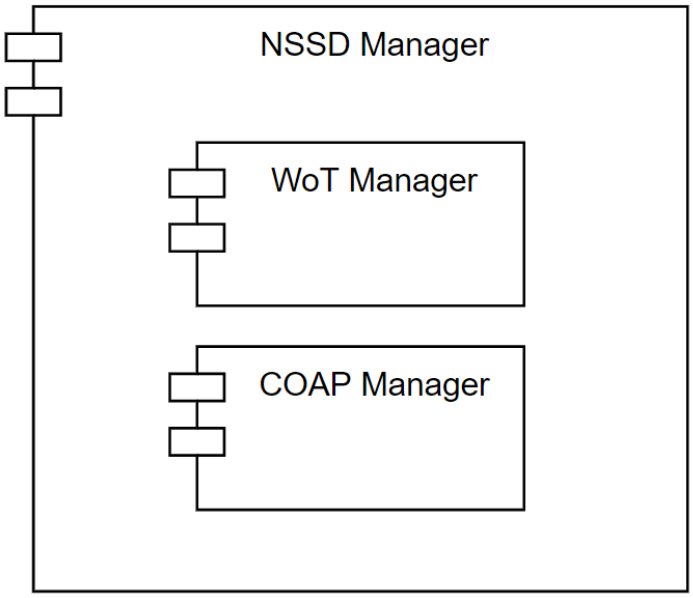

Figure 8: Components of the NSSD Manager

3.1.3.1 WoT Manager

This is the component that takes care to manage the interaction of the SIFIS-Home framework with NSSD devices that are compliant to the WebThings specification. In particular, this component takes care to perform the WebThings discovery procedure through which WebThings enabled devices are discovered. Also, it retrieves all the WebThings descriptions and makes them available in a WebThings directory and on the DHT, for further usage. Finally, it receives commands for the WebThings NSSD devices from the DHT and takes care to execute them using

the WebThings standard. The software structure and library dependencies of the WoT manager are depicted in Figure 9.

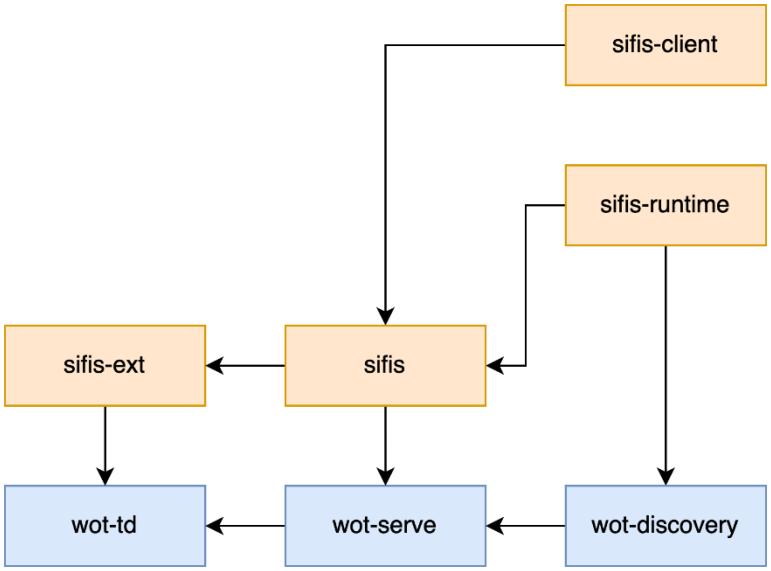

Figure 9: Software structure of the WoT Manager

A short description of the components is reported in the following.

- *sifis-client*: is the software used to develop third-party applications runnable on SIFIS-Home smart devices.
- *sifis*: is the software components used to translate WoT things representations in SIFIS-Home devices and resources to make them available to third-party applications.
- *sifis-runtime*: is the interface toward the DHT, which reads requests toward WoT devices or publish on topics related to resources controlled by the WoT NSSDs.

3.1.3.2 Interaction with WebThings

WebThings enables direct control of smart home devices over the web by giving them URLs, making them discoverable and linkable, also defining a standard data model and APIs to make the devices interoperable and to exchange data between devices and systems. WebThings implementation includes two main items:

- WebThings Gateway: allows the user to monitor and manage smart home devices and infrastructure over the web, it has also a rules engine to automate functions based on a predefined set of rules.
- WebThings Framework: consists of software components developed to directly interact with the WebThings API and use smart home services.

The SIFIS-Home developers APIs build upon WebThings model, which is used to abstract from the specific producer-based implementation of functionalities used to provide generic services, such as “Switch on Light”, “Open Lock”, “Increase Temperature”, etc. Following the Web of Things terminology, we can name these services “Capabilities”. The Capabilities help developers of third-party applications to provide applications able to invoke these generic services, without having to be worried about the actual implementation which in general is device specific. To clarify, let us suppose, for example, that two device manufacturers provide for their device “Refrigerator”, two different

implementations of the "lowerFridgeTemp()" API, to decrease the current temperature in the refrigerator by 1 °C. To offer this API to third party developers, not having to foresee two distinct invocations, one for Manufacturer 1 and one for Manufacturer 2, the manufacturers describe the API as a capability "lowerFridgeTemp()", exposed by SIFIS-Home. In such a way, the developer shall simply invoke the capability and the actual implementation, or the specific refrigerator present in a home, are fully transparent to him.

3.1.3.3 WebThings Architecture

Figure 10 shows the architecture of WebThings that is mainly composed of:

- **Things:** a physical or virtual entity in the smart home environment abstracted and integrated in the SIFIS-Home architecture. Things are thus representation of a resource in a IoT environment offering services and functionalities.
- **Thing Description:** a structured description of general metadata, domain-specific metadata, supported Protocol Bindings and links. These metadata are self-descriptive, so that consumers are able to identify Thing capabilities and how to use them based on the affordances concept.
- **Consumers:** entities that are responsible for interaction with Things and the processing of Thing Description. They can be either other devices, applications or web services.
- **Intermediaries:** entities that sits between Things and Consumers and are indistinguishable from Things for Consumers.

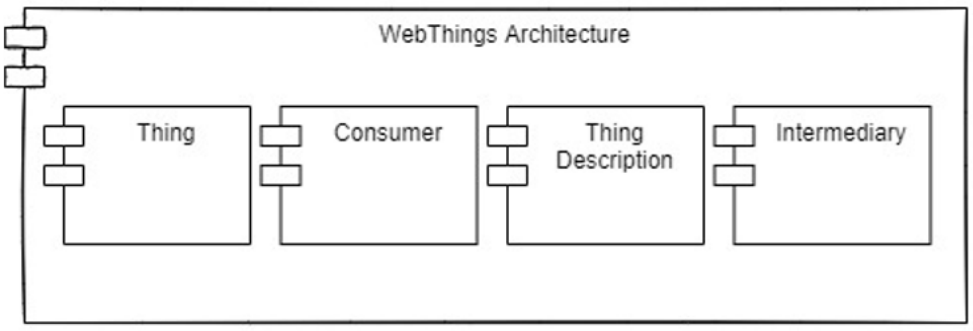

Figure 10: WoT Architecture

3.1.3.4 WebThings Components

When WebThings architecture components are implemented as a software stack to take the role of representing an Exposed Thing and making the WoT interface available to the Thing Consumers or representing a Consumed Thing and making this Thing available to the applications running on the servient and need to process Thing Descriptions to interact with Things. The resulted software stack is called a servient and its architecture is illustrated in Figure 11. WebThings servient model has two different implementations, either based on the WoT scripting language or based on native language implementation, as shown in and the components are:

- **Behaviour Implementation:** represents the overall application logic of the servient in terms of servient autonomous behaviour, interaction affordances, Consumer behaviour, Intermediary behaviour, and the definition of Things, Consumers, and Intermediaries that are hosted by the Thing.
- **WoT Scripting API:** is the interface between Behaviour Implementation and a scripting-based WoT Runtime. Where the WoT Runtime is a Scripting Runtime system that manages

WoT-specific aspects, interprets and executes application scripts.

- **Native WoT Runtime:** a Servient implementation without the WoT Scripting API. Any programming language may be used for the WoT Runtime application-facing API. But usually, it is the native language of the Servient software stack.
- **Private Security Data:** security data such as the secret key used for interaction with Thing are managed by WoT Runtime but not directly accessible by the application. This data is stored in a separate isolated memory and an abstract of a group of operations are made accessible.
- **Protocol Stack Implementation:** implements the WoT interface used by Consumers to use remote Things via Exposed Things and produces protocol messages for interactions over the network, where multiple protocols may be implemented supported by protocol bindings for different IoT platforms.

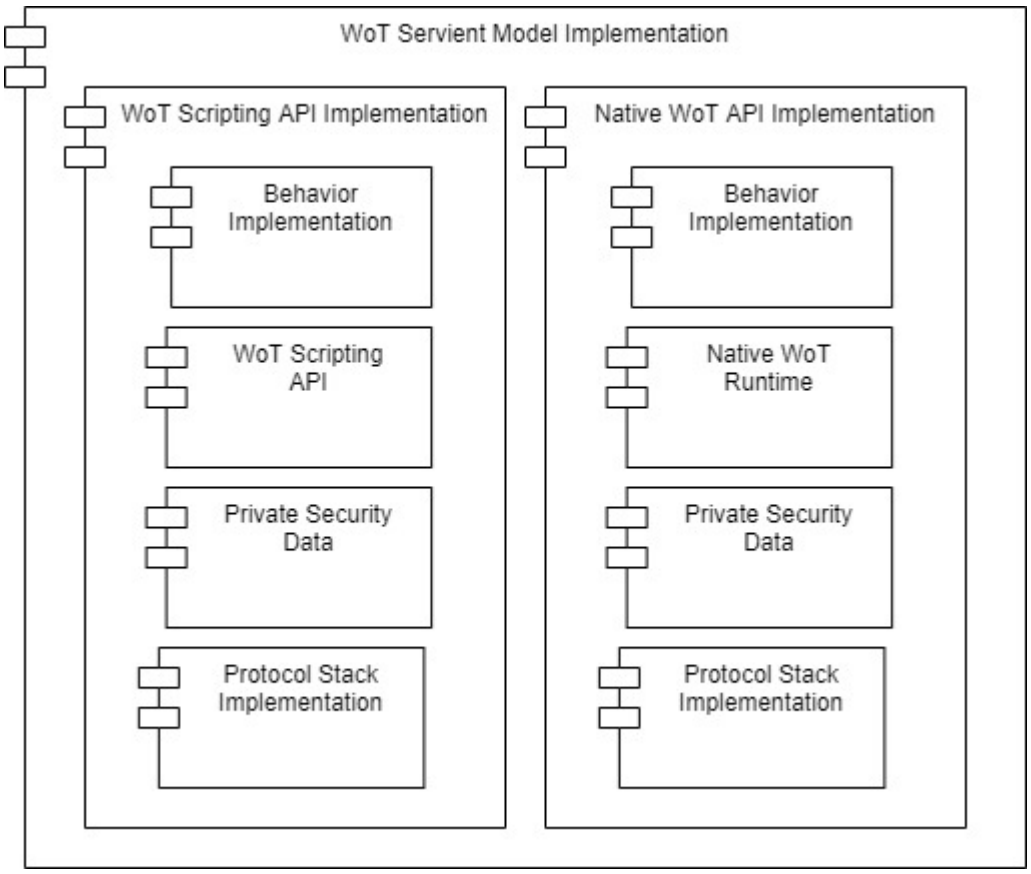

Figure 11: Architecture of the WoT servient

3.1.3.5 COAP Manager

This is the component that manages the communication of the SIFIS-Home framework with NSSD devices that are using COAP as their application protocol. The COAP Manager receives commands and retrieves information from the DHT manager, then it takes care to execute the requested operations toward the NSSD devices. The communication with the NSSD is protected and established by using the advanced protocols and solutions that have been studied in the context of WP3.

3.1.4 Application Toolboxes

The diagram in Figure 12 shows the structure of the Application Toolboxes module.

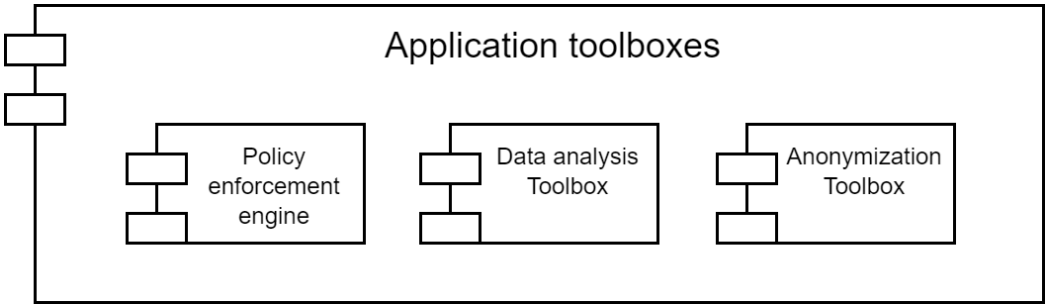

Figure 12: Components of the Application Toolboxes module

The Application Toolboxes module includes the following components that further elaborated in sub-sections:

- **Policy Enforcement Engine:** The policy enforcement engine is responsible of evaluating requests of actions and services in the SIFIS-Home framework, against the policies of usage, privacy and security defined by the administrator or maintainer.
- **Data Analysis Toolbox:** The data analysis toolbox includes all the mechanisms to perform statistical-based and machine learning data analysis. The analysis toolbox provides thus the main building blocks to analyse textual data, tabular data and multimedia data in order to perform predictions, analyse voice and gesture commands, detect intrusions and misbehaviours, as well as providing advanced smart services to the smart home users.
- **Anonymization Toolbox:** The anonymization toolbox contains software tools to preserve privacy of data during analysis. Depending on the data type and the desired level of privacy, the anonymization toolbox can generalize or suppress data information, as well as supporting differential privacy for privacy preserving data analysis.

Figure 13 illustrates in detail the sub-components of the **Policy Enforcement Engine component**. More in details whenever an actor (subject) or component of the SIFIS-Home Architecture is willing to perform an operation, or requesting a service which might have security, safety or privacy implications, a request is issued and sent to the policy enforcement engine. The request is matched with the relevant policies, being thus either authorized or denied. The decision is based on a number of conditions depending on the subject performing the request, the operation requested and the context in which the operation is performed. The policy definition and evaluation follow an access control model based on ABAC (Attribute Based Access Control), which also considers mutable attributes for decisions that might change over time. The model exploited by the SIFIS-Home framework is the Usage Control (UCON), which extends the classical ABAC by including the possibility to revoke previously granted authorizations.

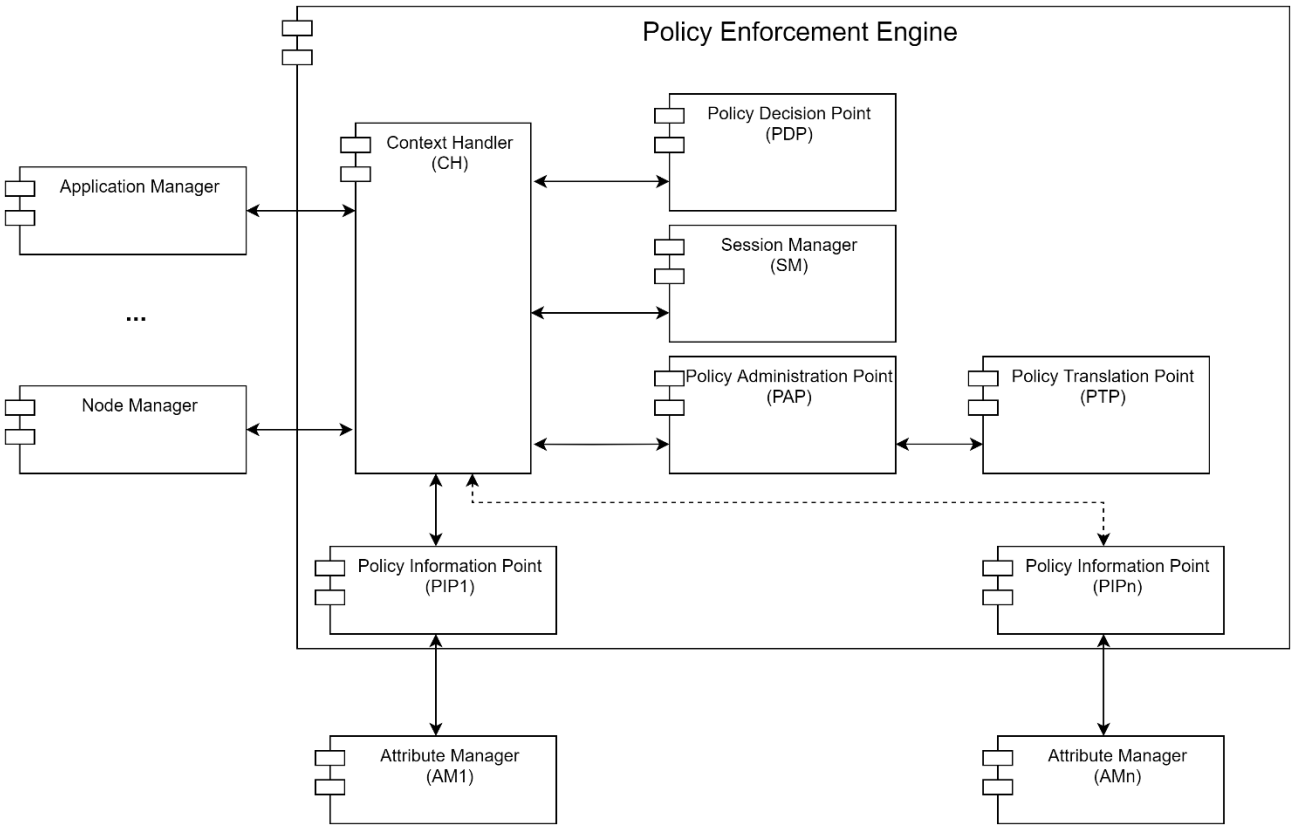

Figure 13:Sub-components of the Policy Enforcement Engine component

In the following we will give a description of the components of the policy enforcement engine:

- **PEP**: *Policy Enforcement Point*. This component acts as interface to the policy enforcement engine, sending and receiving messages from it. In particular, the PEPs intercept the issuing of a critical operation and prepares the request to be evaluated. Afterward, the PEP receives the decision on the policy evaluation and enforces the policy effectively granting, denying or revoking the access and usage to a resource, operation or service. In the SIFIS-Home framework, the PEPs are the various subcomponent of the framework issuing operations that will directly control both the running applications and devices, both Smart and NSSD.

- **PDP**: *Policy Decision Point*. This component is the actual policy evaluation engine. It takes as input an *access (usage) request* and an *access (usage) policy* returning one of three possible decisions: (i) PERMIT, (ii) DENY, (iii) UNDETERMINED. Policies and requests are expressed in XACML language [XACML, 2013].

- **PIP:** *Policy Information Point.* This component retrieves attributes related to the *subject* (i.e. the entity requesting a action or resource), *object* (i.e. the protected resource or action), or *environment* (i.e. the description of the context) of received access requests. Each PIP act as the interface of the Policy Enforcement Engine with a specific Attribute Manager. The implementation of each PIP is custom for the specific application, Attribute Manager and kind of attribute that should be retrieved.

- **CH:** *Context Handler*. This component is mainly a message dispatcher. It is at the core of the Policy Enforcement Engine and is responsible of forwarding the access requests first to the various PIPs for attribute retrieval, then forwards the complete access request to the PDP, finally returning the decision to the PEP. The CH also receives notification from PIPs for attribute value changes, forwarding the new value to the PDP for policy re-evaluation,

eventually notifying the PEP for a triggered access revocation. The CH is responsible of handling concurrency issues, also in applications considering several PEP and PIP.

- **SM:** *Session Manager.* This component is a database which stores all the active sessions, with the necessary information to perform policy re-evaluations.
- **PAP:** *Policy Administration Point.* This component stores the policies for the PDP and automatically retrieves the correct policy(es) to be matched with the requests sent by the PEPs.
- **PTP:** *Policy Translation Point.* The PTP modules enables translation between high level human readable policies (which will be described in deliverables of WP2 and WP4) and the enforceable XACML policies. PTP module firstly checks if there are conflicts between the available high-level policies. If this is the case, the Evaluator/Notifier and Alarms/Log modules are used to send an alarm to the user, e.g., in the form of a notifications. If there are no conflicts, instead, the policies are translated in the corresponding XACML policies.
- **AM:** *Attribute Manager:* This component is the interface between a PIP and the attribute environment. In the SIFIS-Home framework AMs are the various monitors provided by the Proactive Security Manager and the Distributed Storage itself.

Figure 14 presents in detail the sub-components of the Data Analysis Toolbox component. It includes four sub-components to carry out statistical-based and machine learning data analysis.

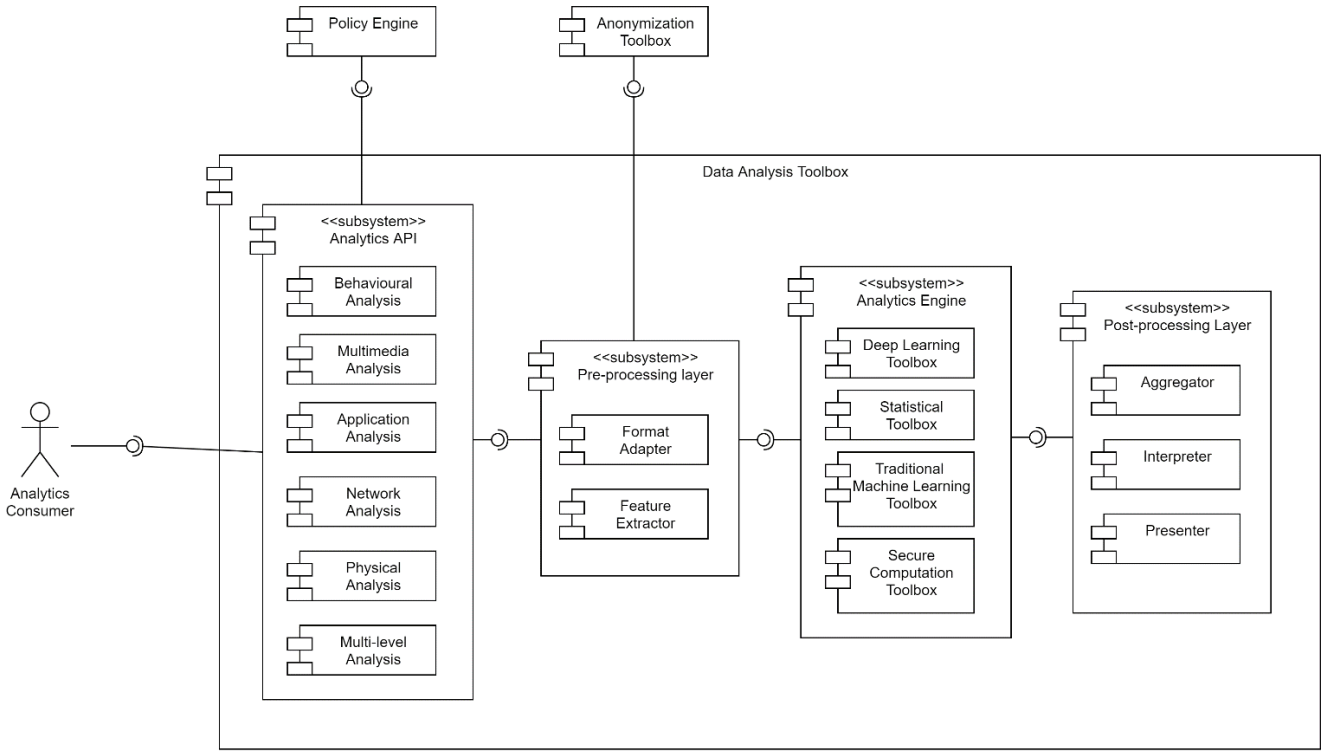

Figure 14: Sub-components of the Data Analysis Toolbox component

The building blocks of the data analysis toolbox are the following:

- The Analytics API: this component provides key data sources and enable data integration and visualization. It is responsible to grant access to data analysis mechanisms, provide the functionality to request these mechanisms, pass required parameters and datasets for them to be performed, and define the way of how the system should respond to the user and visualize the analysis results. This component has six analysis methods that enable real-time predictions

using datasets, models, clusters, and anomaly detectors. Data collected from SIFIS Home environment are analysed and then produce results to answer questions like whether a software intrusion has been detected, is there a person in a dangerous situation. It allows to extract important insights, and execute data manipulations, like converting speech to text and perform person recognition and identification.

- The pre-processing layer: this component is responsible for the data preparation functions in terms of cleaning, formatting, normalization, transformation into an understandable format, and feature extraction. It involves cleaning missing and noisy data, transforming data into an appropriate and unified format, normalizing data according to the range of values, and reducing data dimensionality by selecting or combining variables into features. The product of this component is the final dataset presented in a unified format ready to be analysed by the analytics engine.
- The Analytics Engine: this component receives the output dataset from the pre-processing layer after being anonymized as an input and provides methods and models to describe and analyse data based on machine learning, deep learning, and statistical approaches with secure computation. This component invokes the mechanisms to handle intrusion detection, identity recognition, object detection, parental control, multi-level intrusion detection, and software intrusion detection. After data analysis is performed, the results are sent to the last component to be aggregated and presented to the user.
- The Post Processing Layer: this component is responsible for the aggregation of the results obtained from the analytics engine and prepare them to be displayed either on-screen or on hardcopy to the user. It also interprets the results and communicate with the alerting and monitoring component in case an intrusion or an anomaly has been detected.

Figure 15 presents in detail the sub-components of the Anonymization Toolbox component. This toolbox is responsible for data anonymization to protect the privacy of sensitive data by performing multiple operations:

- The generalization component modifies original data to be in a more general version.
- The suppression component is used to delete or replace the values that disclose identifying information about the user with NULL value.
- The perturbation component replaces the original data by adding noise to dataset elements in a random manner.

After the anonymization operations are performed on the original dataset, the anonymized dataset is forwarded to the pre-processing layer in order to be shared with the Analytic Engine.

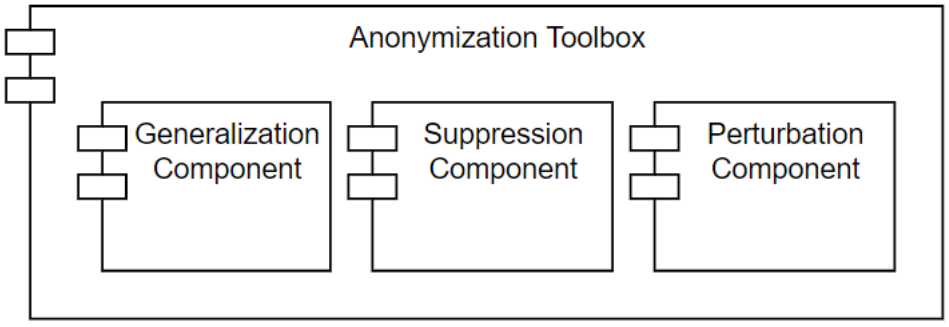

Figure 15: Sub-components of the Anonymization Toolbox component

3.1.5 Secure Communication Layer

The Secure Communication Layer is illustrated in Figure 16 and includes tools and functionalities for handling secure message exchange between the components of the SIFIS-Home architecture, in particular among smart devices.

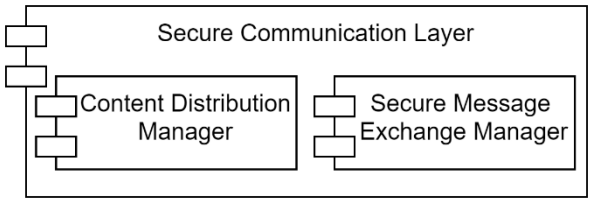

Figure 16: Secure Communication Layer

The components of the secure communication layer are described in the following.

- Secure Message Exchange Manager – This component provides functionalities to ensure (end-to-end) protection of communications and message exchanges, between the devices in the SIFIS-Home network as well as between such devices and auxiliary entities providing security/administrative supporting services in the network.
- Content Distribution Manager – This component provides functionalities to ensure that (protected) data and control messages are practically exchanged in the SIFIS-Home network infrastructure, between the devices in the SIFIS-Home network as well as between such devices and auxiliary entities providing security/administrative supporting services in the network. Support is provided for different message exchange models, such as one-to-one, one-to-many (group communication), and publish-subscribe.

3.1.6 Proactive Security Management Layer

The Proactive Security Management Layer (whose design is reported in Figure 17) is the component that handles automatically all tasks related to management of safety and security, such as intrusion and anomaly detection. It is composed of three different components, named Monitors, Self-Healing and Distributed trust, that we describe in the following.

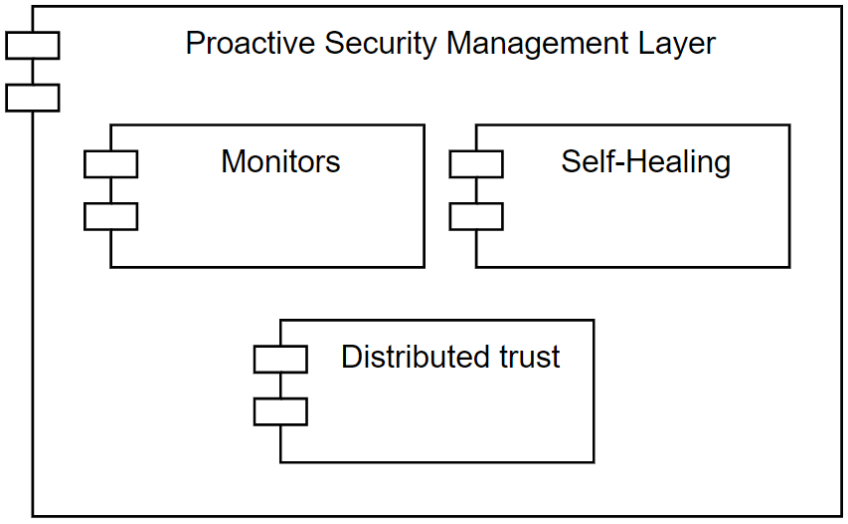

Figure 17: Components of the Proactive Security Management Layer

Monitors: this component is tasked with identifying misbehaviours, security and safety threats in a SIFIS-Home enabled smart home.

In SIFIS-Home we consider the following misbehaviour types:

- Physical Intrusion: An unauthorized person enters the house premises, or a Guest or Operator performs unauthorized operations (e.g. when not in the presence of the tenants, begins looking in the drawers, or they perform aggressive behaviours).
- Software Intrusion: An application performs intentionally behaviours aimed at either violating privacy, taking control of the SIFIS-Home architecture, hijacking communications between devices, escalate privileges, violating data integrity, causing physical damage to devices, the home or tenants, undermining the expected functionality of the smart home services and devices.
- Device or service misfunction: A device or a service unintentionally misbehaviour, due to a technical issue (e.g. broken sensor or actuator), or programming mistake.

The Monitors component, thus, extracts/receives features and data from daemons constantly monitoring all the layers of the SIFIS-Home architecture and framework. Features include information on operations on the DHT, readings from physical sensors, cameras and microphones, API monitors and System Calls. The received information are continuously analysed by the Monitors, exploiting the classifiers of the Data Analysis Toolbox. Afterward, in case a misbehaviour is identified, the Monitors component interacts with the Application Manager and the Node Manager in the Secure Lifecycle Manager to tackle the misbehaviour according to a set of pre-defined actions. The Monitors component also cooperates in both directions with the Distributed Trust Manager, to improve detection and prevention of misbehaviours coming from Smart Devices.

Self-Healing: this component automatically detects, analyses, and fixes network and devices failures with minimal human intervention to ensure the property of being self-tolerant and dynamically adapt to device failures and network requirements and changes. This is performed using the 'Secure and Dependable Communication Manager' for network communication between devices and distributed Hashtable architecture updates when a device is registered/deregistered. And the 'Secure Lifecycle Manager' for framework bootstrapping, registration/deregistration of the devices and forcing these operations when needed.

Distributed Trust: the distributed trust manager is a component in support of the anomaly detection. Being in a fully distributed peer-to-peer environment, a root of trust is lacking and in the event of having one or more devices (both smart devices and NSSDs) compromised, identification of the compromised device(s) will be based on a collective decision where every Smart Device participates. The collective decision is based on a weighted voting procedure on identifying the misbehaving device(s). Elements to this decision are features and readings on physical and software measures, coming from sensors and the various monitors offered by the Proactive Security Management Layer. The distributed trust manager exploits these readings from each device to constantly update a trustworthiness value, which is based on three parameters: belief, disbelief and uncertainty. When a voting procedure is performed, the vote from each device is weighted by its trustworthiness value. Moreover, nodes with a low trustworthiness level are immediately considered either as corrupted or damaged. Through the node manager, these devices are prevented to further participate to the architecture activities, unless the self-healing engine manages to fix the issue. Further details, together with a first implementation of this distributed trust mechanism can be found in [Faiella et al. 2016].

3.1.7 DHT Manager

This component (whose representation is provided in Figure 18) is in charge of managing the DHT (Distributed Hash-Table) and allowing access to it to other software components. Also, it manages the interaction with Yggio, that is the SIFIS-Home component responsible for providing a FIWARE compatible API to a SIFIS-Home enabled smart home. As it can be observed by looking at Figure 17, DHT Manager presents two macro-components, the DHT and the Fiware API ones.

The Publish/Subscribe DHT software component, hereafter referred to as SIFIS-Home DHT for the sake of brevity, is a software module that i) allows communication among SIFIS-Home software components and SIFIS-Home devices using a publish/subscribe pattern and ii) provides an easy-to-use mechanism to share information among devices and applications.

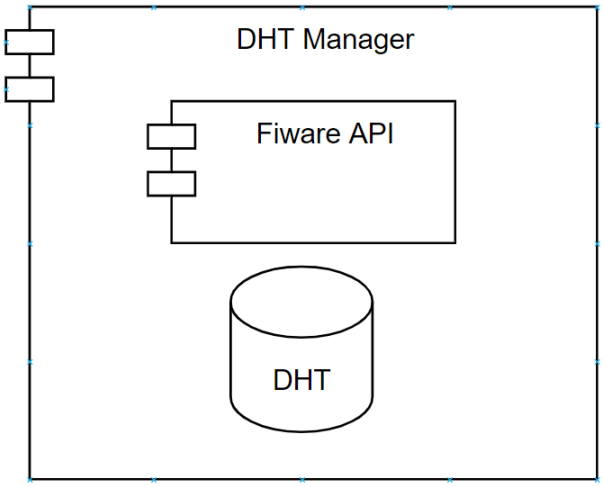

Figure 18: Components of the DHT Manager

SIFIS-Home DHT Reference scenario: the SIFIS-Home DHT component has been specifically designed for the smart home scenario, which presents a set of unique characteristics and challenges. As depicted in Figure 19, in a typical smart home, devices are interconnected by means of different network technologies. For example, we can have some devices (such as smart devices SD1 and SD3 in Figure 19), that are directly connected to the main home router (through a Ethernet or Wi-Fi connection), but there are also devices (SD2 and SD4 in Figure 19) that connect to the main home router and to the other smart home devices by means of a multi-hop mesh network. Also, devices can join and leave the network at any time and can experience failures. In addition, due to the use of wireless communication, the network may be unstable and network partitions can occur.

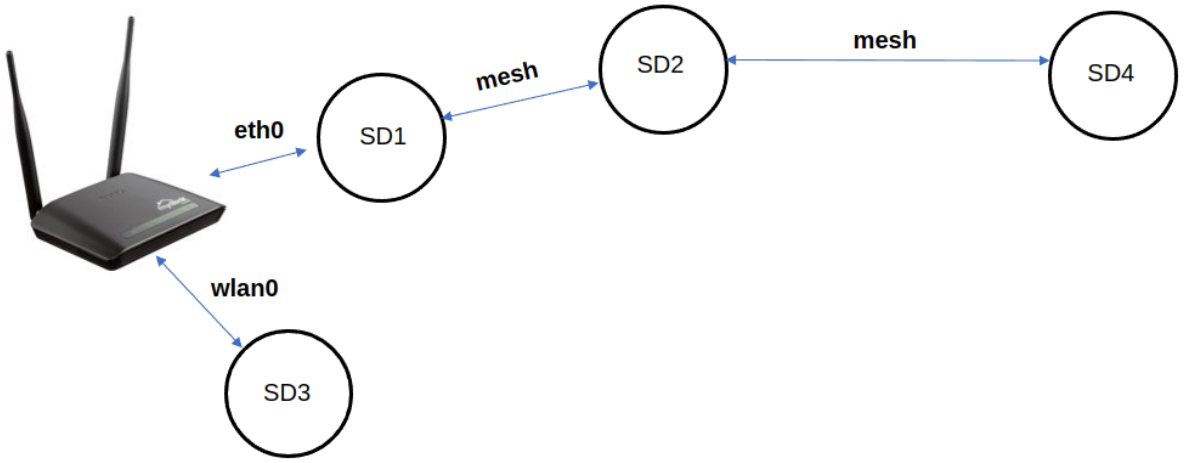

Figure *19*: DHT reference smart home scenario

In a SIFIS-Home enabled smart home, the smart devices are used to run a number of services and applications that take care to manage the smart home (see also Figure 20). Applications need to communicate with each other and, also, need to share information. Note that due to devices joining/leaving the smart home it is not possible (or very difficult in practice) to provide applications with the list (and addresses) of all the other applications that are in their same smart home. Hence, dynamic discovery mechanisms should be adopted.

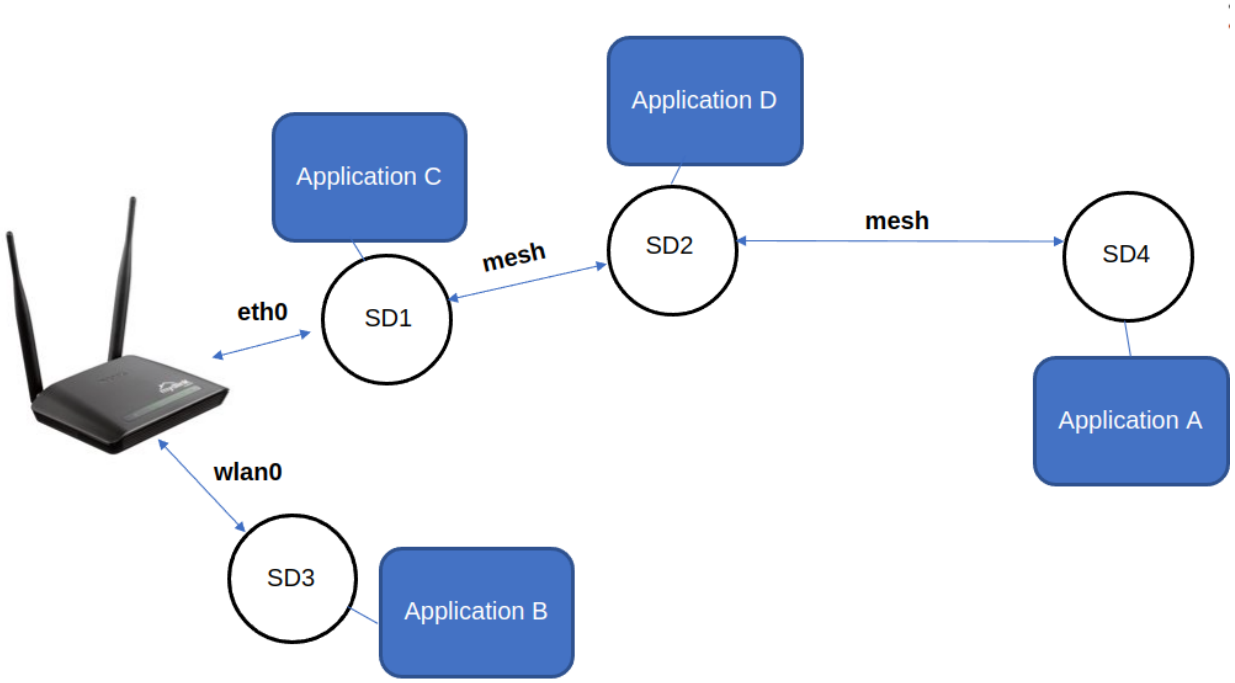

Figure 20: DHT reference scenario, applications running on different smart devices

SIFIS-Home DHT Requirements: the SIFIS-Home DHT is the mechanism through which the communication and data sharing challenges that applications experience in a typical smart home scenario are addressed in the SIFIS-Home project. In detail, we designed the SIFIS-Home DHT in such a way that it satisfies the following requirements:

- **Pub/Sub messaging without broker:** pub/sub messaging is a very effective way to allow different microservices to communicate. However, classic pub/sub protocols, such as MQTT,

are based on the presence of a central messaging broker in the system, that represents a Single Point of Failure (SPoF). The SIFIS-Home DHT allows applications to rely on a pub/sub messaging mechanism to exchange information but without the presence of a centralized broker.

- **Dynamic discovery:** due to the dynamic nature of a smart home, with nodes joining/leaving the network at any time, it is not possible to provide applications with the list of all the other nodes and services present in the system. For this reason, the SIFIS-Home DHT provides a built-in discovery mechanism through which nodes automatically discover each other. The discovery mechanism works even if nodes are connected through multiple hops.
- **Authentication, confidentiality:** despite the DHT presents a built-in discovery mechanism, only allowed nodes should be able to join the DHT and publish/receive messages. All communication in the SIFIS-Home DHT is encrypted and all nodes must be provided with a shared DHT group key in order to be part of the DHT.
- **Robust to network partitions:** due to the use of wireless communications, network partitions are likely to occur in a smart home scenario. The SIFIS-Home DHT is robust to network partition. In particular, applications are not affected at all by network partitions and can continue to operate normally. Also, specific mechanisms are used to solve possible problems (e.g. data conflicts) due to the network partitioning (see below).
- **Volatile and persistent messages:** the DHT allows applications to publish/receive two different types of messages: volatile and persistent messages. Volatile messages are used to propagate volatile information (e.g. commands for smart home devices, signalling of events), i.e. information that does not need to be stored in a persistent way. Conversely, persistent messages contain information that needs to be stored in a persistent way. For example, in a smart home scenario persistent messages are used to propagate and store temperature targets, user policies or the list of devices in the house.
- **Data conflicts management:** every time a new node joins/re-joins the DHT, a possible data conflict can occur, since the set of persistent messages possessed by the new joining node can be significantly different with respect to that present on the nodes that are already part of the DHT (referred to as DHT nodes in the following paragraph). In detail, data conflicts occur when: a) there are some messages stored by the joining node that are missing in the set of persistent messages available on the DHT nodes, b) the DHT nodes have some persistent messages that are not present in the joining node storage, c) some messages are present on both the new node and the DHT nodes but their values are different. The SIFIS-Home DHT provides a built-in mechanism to automatically solve data-conflicts.

In the following section we describe in detail the different mechanisms that are used by SIFIS-Home DHT to satisfy the above mentioned requirements.

SIFIS-Home DHT technical description: The SIFIS-Home DHT has been developed using the Rust language, due to its stability, performance and memory usage guarantees. Also, the SIFIS-Home DHT relies on the usage of the libp2p Rust implementation (https://github.com/libp2p/rust-libp2p). In detail, SIFIS-Home DHT makes use of the libp2p implementation of the **mDNS** and **gossipsub** protocols. mDNS is used to provide automatic node discovery. gossipsub is instead used to provide the SIFIS-Home software modules with a completely distributed pub/sub messaging system. In the following subsections, we report some details about libp2p and its mDNS and gossipsub protocols implementations.

libp2p basics: libp2p is a modular system of *protocols*, *specifications* and *libraries* that enable the

development of peer-to-peer network applications. In a libp2p network every node is identified through a PeerId. Every libp2p node has a private key, which it keeps secret from all other nodes. Every private key has a corresponding public key, which is shared with the other nodes. The public and private key (or "key pair") allow nodes to establish secure communication channels with each other. A PeerId is a cryptographic hash of the public key of a certain node. When nodes establish a secure channel, the hash can be used to verify that the public key used to secure the channel is the same one used to identify the node.

libp2p mDNS: multicast DNS (mDNS) protocol resolves hostnames to IP addresses within small networks that do not include a local name server. It is a zero-configuration service, using essentially the same programming interfaces, packet formats and operating semantics as unicast Domain Name Service (DNS). It takes a different approach than the well-known DNS. Instead of querying a name server, a client sends a multicast request asking which network node matches up with a particular host name. Multicast is a unique form of communication through which an individual message is directed at a group of recipients. The group can be made up of, for example, the entire network or a sub-network. In this way, the request also reaches the node who owns the host name that is being searched for. The latter responds to the entire network (also via multicast). All participants are informed of the connection between the name and IP address of the node, and can make a corresponding entry in their mDNS cache. As long as this notation is valid, no one in the network needs to request the host name again.

In the context of libp2p, the mDNS protocol is used to discover other nodes on the local network that support libp2p. In detail, when a libp2p node starts, it sends a specific mDNS message asking all the other libp2p nodes in the network to send an mDNS response with their information. As mDNS responses come in, the node adds the information of the other nodes into its local database of peers/nodes. The mDNS response sent by every libp2p node contains, in particular, the list of addresses that can be used to connect to the node as well as its PeerId (and, hence, its public key).

Through the usage of the mDNS protocol a libp2p node is able to discover all the other libp2p nodes that are connected to its same local network. SIFIS-Home DHT instructs libp2p to use all the available network interfaces to send mDNS discovery requests. In this way, whatever local network is used to connect two nodes, they are able to discover each other. Please note that the mDNS discovery mechanism alone does not provide multi-hop node discovery. SIFIS-Home DHT uses an additional mechanism to provide multi-hop node discovery (see below for details).

Additional details of the libp2p mDNS implementation can be found at https://github.com/libp2p/specs/blob/master/discovery/mdns.md.

libp2p2 gossipsub: In a publish/subscribe system nodes communicate by exchanging messages and every message pertains to a specific topic. A node must be subscribed to a certain topic in order to receive the messages pertaining to that topic.
In a distributed pub/sub system all nodes participate in delivering messages throughout the network. libp2p uses a gossipsub protocol to provide a completely distributed publish/subscribe messaging infrastructure. The name gossipsub is due to the fact that peers gossip to each other about which messages they have seen and use this information to maintain a message delivery network.
Before a node can subscribe to a topic, it should discover the other nodes in the network and establish a connection with them. Libp2p gossipsub does not provide mechanisms to discover other nodes, but expects the application using it to provide the addresses of the nodes to which a connection should be established. In SIFIS-Home DHT, other nodes are discovered through mDNS (see also sections below).

Subscribing operation: Nodes keep track of the topics their directly-connected nodes are subscribed to. Using this information each node is able to build up a picture of the topics around them and which nodes are subscribed to each topic (Figure 21):

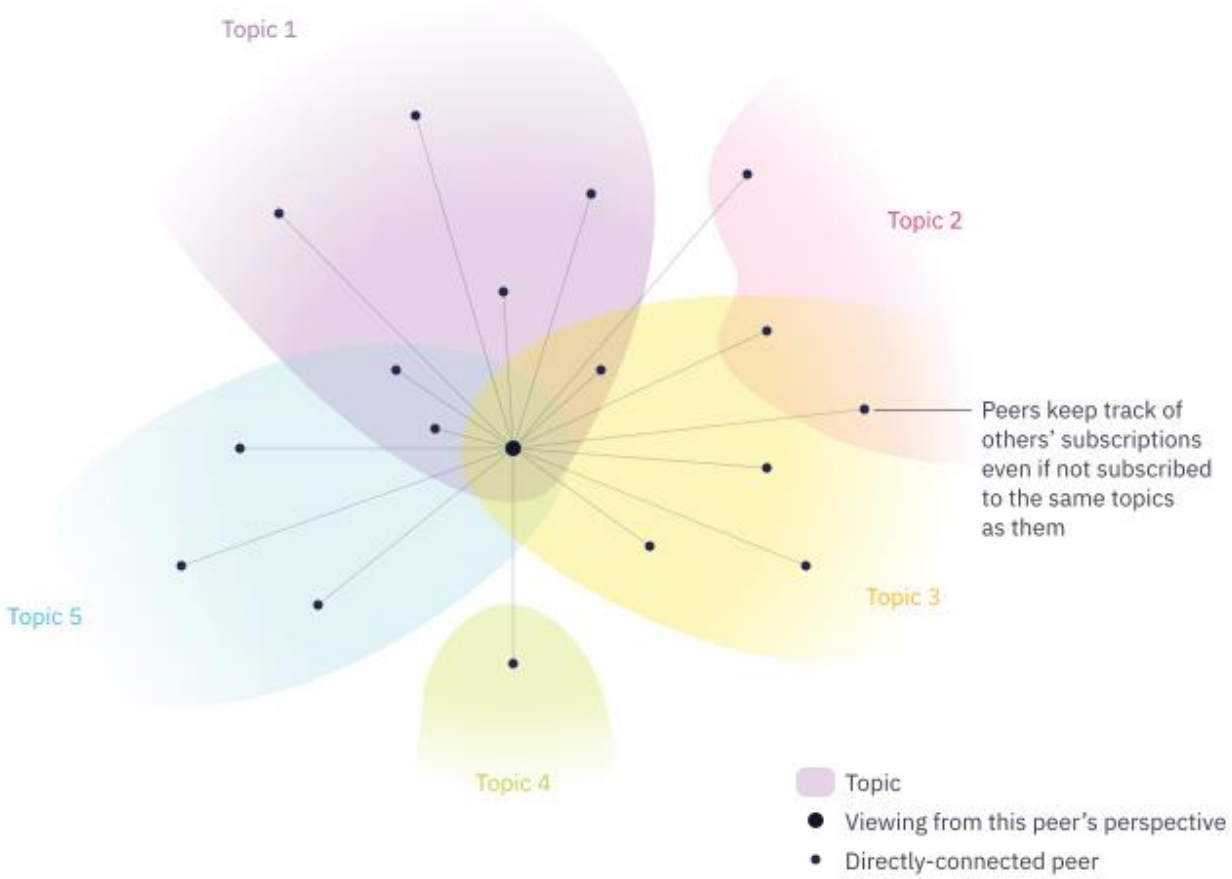

Figure 21: LibP2P subscription model

Keeping track of subscriptions happens by sending subscribe and unsubscribe messages (Figures 22-23). When a new connection is established between two nodes they start by sending each other the list of topics they are subscribed to:

Figure 22: Messages exchanged for subscribing to topics

Then over time, whenever a node subscribes or unsubscribes from a topic, it will send each of its peers a subscribe or unsubscribe message. These messages are sent to all the connected nodes regardless of whether the receiving node is subscribed to the topic in question:

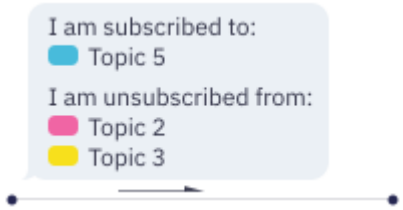

Figure 23: Messages exchanged for subscribing/ unsubscribing to topics

Publish operation: when a node wants to publish a message it sends a copy to all the nodes it is directly connected to (Figure 24).

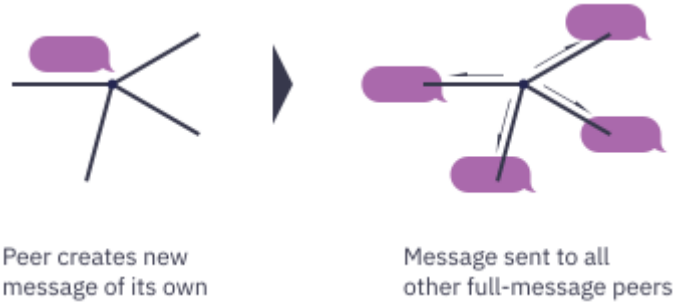

Figure *24*: Publish operation

Similarly, when a node receives a new message from another peer, it stores the message and forwards a copy to all other nodes it is directly connected to (Figure 25). In the gossipsub specification, nodes are also known as *routers* because of this function they have in routing messages through the network.

Figure 25: Forwarding of a published message

Additional details of the libp2p gossipsub implementation can be found at https://docs.libp2p.io/concepts/publish-subscribe.

SIFIS-Home DHT combined usage of mDNS and gossipsub: The libp2p gossipsub protocol does not have any way to discover nodes by itself. For this reason, every SIFIS-Home DHT node uses mDNS to discover other nodes in its local network. Every time a new node is discovered through mDNS, its addresses and PeerID are sent to the gossipsub protocol in order to start a connection with it and start setting up a pub/sub messaging infrastructure.

SIFIS-Home DHT access protection and DHT confidentiality: In order to prevent unauthorized nodes from joining the gossipsub system, every DHT node is equipped with a pre-shared group key named DHT key. It is a 32 bytes long key that can be generated using standard tools such as openssl (the command *"openssl rand -hex 32"* generates a 32 bytes long random key). Once a connection between two nodes is started, a proof of possession operation of the shared group key is performed. The connection is established only if the two nodes both possess the shared key. Please note that every

libp2p connection is both encrypted and authenticated. Hence, it is not possible for nodes external to the DHT to capture DHT information.

SIFIS-Home DHT details: in this section we describe the specific operations and algorithms that are executed by every SIFIS-Home DHT node. As mentioned before, SIFIS-Home DHT relies on the libp2p gossipsub publish/subscribe infrastructure. In detail, the SIFIS-Home DHT uses three different libp2p gossipsub **topics** to perform its operations:

- Topic **sifis-config**: it is the topic used to send DHT maintenance related messages. They are needed for the correct operation of the DHT. All DHT nodes subscribe to topic sifis-config. Hence, sifis-config messages are spread throughout the entire network.
- Topic **sifis-volatile-data**: it is the topic used to propagate volatile messages. Every DHT node subscribes to topic sifis-volatile-data and is, hence, able to receive all the volatile messages that are published into the system. Volatile messages are not stored using persistent storage. Volatile messages are JSON based messages that can contain arbitrary information.
- Topic **sifis-persistent-data**: it is the topic used to propagate persistent messages. Every DHT node subscribes to topic sifis-persistent-data. Hence persistent messages are received by all the DHT nodes. Every persistent message is a JSON based message. Differently from volatile messages, persistent messages pertain to a topic (whose name is **topic_name**) and are identified by means of a **topic_uuid**. Also, persistent messages are stored using persistent storage.

Every application using the SIFIS-Home DHT (the SIFIS-Home DHT can be embedded into an application) is provided with the following DHT operation primitives:

- **pub(value: json):** operation that allows the application to request the publication of a volatile message. The content of the published message is specified by the *value* parameter (json object).
- **put(topic_name: string, topic_uuid: string, value: json):** operation that allows to request the publication of a persistent message pertaining to topic *topic_name* and whose identifier is *topic_uuid*. Its content is specified by parameter *value* (json object).
- **get(topic_name: string, topic_uuid: string):** operation that allows to retrieve the persistent message identified by *topic_name* and *topic_uuid*.

In the following sections we highlight the operations that are performed when a pub, put, get operation is executed by a certain application. Also, we give details about the data structure that is used to store persistent messages on every DHT node. Finally, we describe the specific messages that are sent using the sifis-config topic that help in maintaining the DHT operational and solving possible data conflicts.

SIFIS-Home DHT pub() operation: an application using the DHT can request the publication of a volatile message using the **pub** operation. In detail, when the pub operation is executed a gossipsub message, whose content is equal to that specified by parameter **value** of the pub operation, is published on topic *sifis-volatile-data*. The message is received by all the nodes that are part of the system using the distributed messaging infrastructure built by libp2p.
SIFIS-Home DHT put() operation: persistent messages must be stored by the DHT in a persistent way. Every persistent message pertains to a specific topic, named topic_name, and has a specific identifier, named topic_uuid. SIFIS-Home DHT makes use of a BTreeMap to store persistent messages in the memory of every DHT node. In addition, persistent messages are stored also on disk using an Sqlite database. In detail, the BTreeMap stores items of type **SifisCacheElement**. Every

SifisCacheElement is composed of different fields and allows storing a single persistent message:

- **topic_name**: topic to which the stored persistent message pertains to
- **topic_uuid**: identifier of the persistent message
- **value:** json object used to store the content of the persistent message, it can contain arbitrary information
- **publisher_peer_id**: PeerId of the node that published the persistent message
- **publication _timestamp**: publication timestamp of the persistent message

Using a BtreeMap allows one to realize the abstraction through which persistent messages can be thought of as being stored in different tables (topic_names) using different rows (identified by the topic_uuid of a persistent message) as depicted in Figure 26.

topic_name: SIFIS::Lights			
topic_uuid	**value**	**publisher_peer_id**	**publication_timestamp**
first_light	{“state”: “on”}	node_1	15
second_light	{“state”: “off”}	node_2	30

topic_name: SIFIS::Sockets			
topic_uuid	**value**	**publisher_peer_id**	**publication_timestamp**
first_socket	{“state”: “off”}	node_3	17
second_sockt	{“state”: “off”}	node_3	89

Figure 26: SIFIS-DHT Abstraction

Every SifisCacheElement is also persisted on disk using an Sqlite database. This allows applications to retrieve persistent messages even in case the node where they execute is restarted.

An application can request the publication of a persistent message by using the **put** primitive. In detail, the application should specify the *topic_name*, *topic_uuid* and *value* of the persistent message to be published. A number of operations are performed after a put request. First, a SifisCacheElement is created and inserted in the BtreeMap of the DHT node where the publishing application is executing (it is also stored in the local Sqlite database). The *publisher_id* is set equal to the PeerId of the DHT node where the application is running while the *publication_timestamp* is the time at which the put operation has been requested (i.e. the current timestamp). The SifisCacheElement is then serialized and published on topic *sifis-persistent-data* to be received by all the other SIFIS-DHT nodes in the network. When a DHT node receives the persistent message, it executes the following operations. First, the topic_name and topic_uuid of the persistent message are extracted from the received serialized SifisCacheElement. Then, the BTreeMap of the receiving node is accessed to check if it already contains a SifisCacheElement identified by the topic_name and topic_uuid of the received persistent message (i.e. if a persistent message with those topic_name and topic_uuid has been already published in the past). If the message is not present in the BTreeMap, the received SifisCacheElement is inserted in the BTreeMap. Otherwise, the publication_timestamp of the received SifisCacheElement and the publication_timestamp of the stored SifisCacheElement are compared. The received

SifisCacheElement is inserted in the BTreeMap only if its publication_timestamp is more recent than the one of the SifisCacheElement already stored in the BTreeMap. In all the other cases, the received message is dropped. This realizes a ***Last Writer Wins*** policy that is extremely helpful while solving possible data conflicts.

BTreeMap initialization
The BTreeMap of a DHT node is initialized using the set of persistent messages that are stored inside the Sqlite file of the DHT node. This allows applications to retrieve persistent messages even in case of a node reboot.

SIFIS-Home DHT get() operation: an application can retrieve a persistent message by using the **get** DHT operation and providing the topic_name and topic_uuid of the persistent message to be retrieved. When it is executed, the BTreeMap of the local node is accessed and the content of the entry identified by topic_name, topic_uuid is returned. Please note that the **get** operation is very efficient since it does not generate any network message.

SIFIS-Home DHT maintenance and data-conflicts management: topic *sifis-config* is used to propagate information that is needed to maintain the DHT in an operational state. Also, the messages exchanged through topic *sifis-config* are necessary to solve possible data conflicts arising after network partitions or node joining events.

Usage of topic sifis-config
Every DHT node periodically sends a DHT State Message using topic *sifis-config*. A DHT State Message contains the following information:

- **peer_id**: peer id of the node publishing the State Message. Since State Messages are received by all the nodes in the system, they can be used by DHT nodes to discover the identity of nodes that are even more than one hop far from them (directly connected nodes are discovered through mDNS).
- **cache_hash**: it is the hash value of the current content of the BTreeMap of the node publishing the State Message. Please note that the cache_hash of a node changes every time the BTreeMap state changes (e.g. due to the insertion of a new persistent message or due to a change in the content of a persistent message stored in the BTreeMap). Also note that if the content of two different BTreeMaps is the same, also the computed hash value is the same.
- **publication_timestamp**: it is the timestamp at which the State Message has been published. State Messages with a quite old publication timestamp are dropped since they do not provide updated information.

Every DHT node stores the received State Messages in a specific table, named **Peers State Table.** The Peers State Table provides a compact view of the states of the BTreeMap of every DHT node in the system. It can be observed that when the DHT is stable, it is expected that the *cache_hash* of all the entries in the Peers State Table should be the same. This is because the set of persistent messages stored by every node should be exactly the same. Conversely, in case a new node joins the system or when some network nodes reconnect after a network partition event, the set of persistent messages on different nodes can be different, causing the cache_hash values of different entries in the Peers State Table to be different too. In such a case, a data conflict is present in the system and the DHT should take care to manage and solve it. In detail, every DHT node periodically checks the entries of its Peers State Table. If some entries report different cache_hash values, the following actions are performed. For every *hash_value* present in the Peers State Table (representing a certain set of persistent messages stored by a certain node or by a set of nodes), a **hash leader** is selected. The hash leader is the DHT node that will be responsible for *republishing* the content of its BTreeMap, i.e. the set of

persistent messages producing that particular hash_value. When all the hash leaders finish publishing the content of their BTreeMap the most recently published persistent message for every topic_name, topic_uuid will be stored in the BTreeMap of every DHT node (remember that a Last Writer Wins policy is used to store messages in the BTreeMap of the DHT nodes). Hence, the data conflict is automatically solved by the DHT thanks to the Last Writer Wins policy and to the periodic exchange of State Messages.

DHT usage: rust applications can use the SIFIS-Home DHT by *embedding* the DHT. The SIFIS-Home DHT is available as a library for Rust native applications. For non-Rust applications, the SIFIS-Home DHT provides a REST + WebSocket interface through which it is possible to publish volatile messages and store/retrieve persistent messages.

Fiware API: This component takes care of forwarding the information stored and published on the DHT to the Yggio component residing on the SIFIS-Home cloud. In addition, the Fiware API component receives configurations and commands from Yggio and stores/forward them on the DHT. The Fiware API component is expected to communicate with Yggio by using the well-known MQTT protocol.

3.1.8 VPN Manager

The VPN Manager component is a VPN client connecting to the SIFIS-Home VPN Server. Its functionality is to allow access to the smart devices from the outside, even in case the home router connection is under NAT.

3.2 *SIFIS-Home Application Framework*

The SIFIS-Home Application Framework is the set of SIFIS-Home software components that are installed and executed on the mobile devices dedicated to the control of the smart home.

The SIFIS-Home Application Framework is the main interaction element between the SIFIS-Home framework and the tenants and other users (e.g. administrator, maintainer). The Application Framework provides a user interface used to configure user preferences, interact with GUI-capable applications, install and remove applications, set-up usage, safety and security policies. SIFIS-Home Application Framework and its macro-components are illustrated in Figure 27.

The SIFIS-Home Application Framework is composed of a set of macro-components providing the following functionalities:

- **Home:** the principal component of the User Interface, containing commands that lead the resident user of the SIFIS-Home system to the lead features of the infrastructure.
- **Device management:** components for the configuration of the devices in the SIFIS-Home network.
- **Alarms/Log:** a component including features to show alarms to the user, and to gather logs of the functioning of the SIFIS-Home infrastructure.
- **Settings:** the component provides user interfaces for the configuration of the SIFIS-Home infrastructure. Different interfaces are provided to different actors of the SIFIS-Home system.
- **Application Launcher:** the component provides graphical user interfaces from which the user can visualize available applications to be installed on the SIFIS-Home system, download, install, update them if new versions are available, and launch them.

- **Input Collection:** modules in charge of providing the facility of collecting the inputs from the user, in all the forms that are allowed by the system.

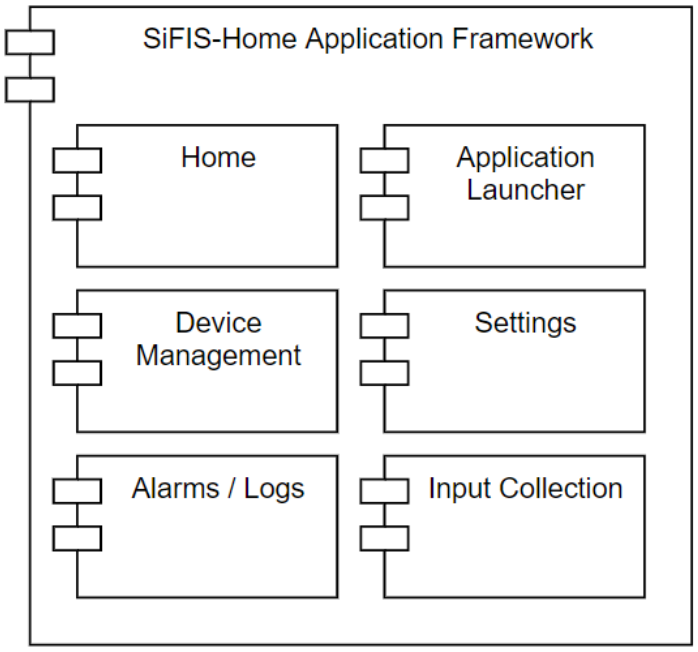

Figure 27: Macro-components of the SIFIS-Home Application Framework

3.2.1 Home

Figure 28 illustrates the Home module that consists of the most important features of the devices. An aim of the module is that users can see the overall status of their Smart Home at a glance, without having to navigate further.

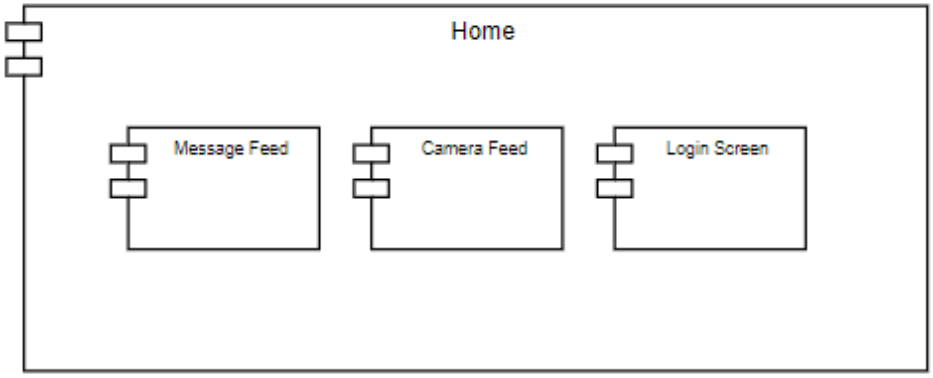

Figure 28: Components of the Home module

The module has five detailed components as follows

- **Message Feed:** This component stores and shows to the users a list of messages received from the smart devices of the SIFIS-Home architecture. Message Feed may contain crucial alerts, a link to the smart device they come from and call to action-buttons depending on the type of the message (e.g., show more-buttons, shut down an alarm and/or feature).
- **Camera Feed:** Shows a live camera feed and an easy way to browse through different cameras if there are more than one. In event of an alarm, Camera Feed should have a recording

of the source of the alarm if there are any cameras pointing at the location or (using facial recognition) a picture of an intruder.

- **Login Screen:** Users may choose to log-in using their biometric data or an access code. Login Screen provides the means for a quick and easy login.

3.2.2 Device Management

The Device Management module provides methods to view the list of SD and NSSD devices of the smart home. The user is allowed to add a certain SD or NSSD to the smart home as well as remove devices. Also, the Device Management module allows to create a list of favourite devices, rename devices, and indicate if a certain device is allowed to report alarms.

3.2.3 Alarms / Log

Figure 29 illustrates the Alarms / Log module that is used to show, visualize and set off alarms. Alarms are logged in an Activity Log and shown in both SIFIS-Home system and SIFIS-Home app in order to reach user wherever they are.

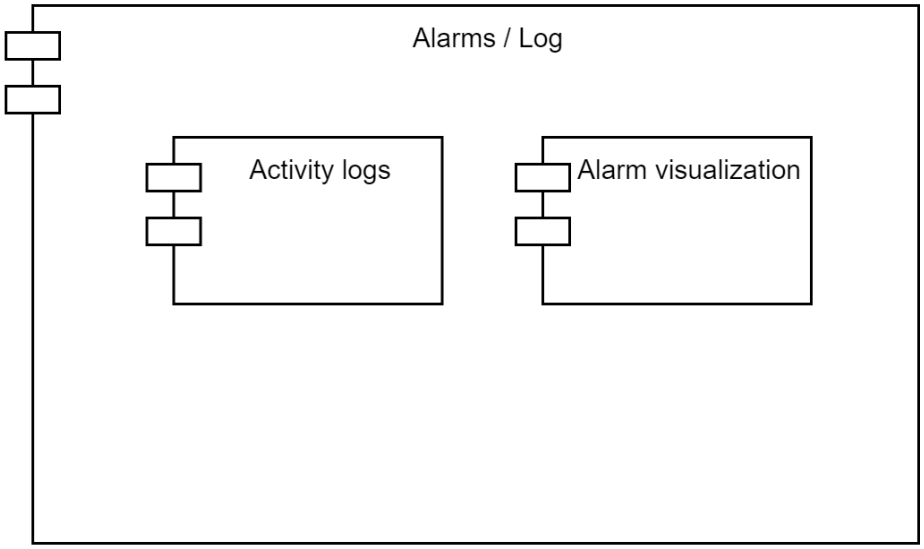

Figure 29: Components of the Alarms / Log module

The Alarms / Log module includes two components as follows

- **Activity Logs:** A sortable list of all the activity from the SIFIS-Home connected devices. Can be divided into *important events* and an *overview* of the events of the last day.
- **Alarm visualization:** A visual presentation of the alarm on SIFIS-Home and a mobile app. Can include both auditive and visual presentations.

3.2.4 Application Launcher

Presents all the available applications which can be installed on a SIFIS-Home enabled smart home. The user should be able to select the list of applications to be installed on the smart home, as well as remove applications.

3.2.5 Settings

Figure 30 presents the Settings module that is divided in two different modules: User Settings and System Setting that are explained in this sub-section.

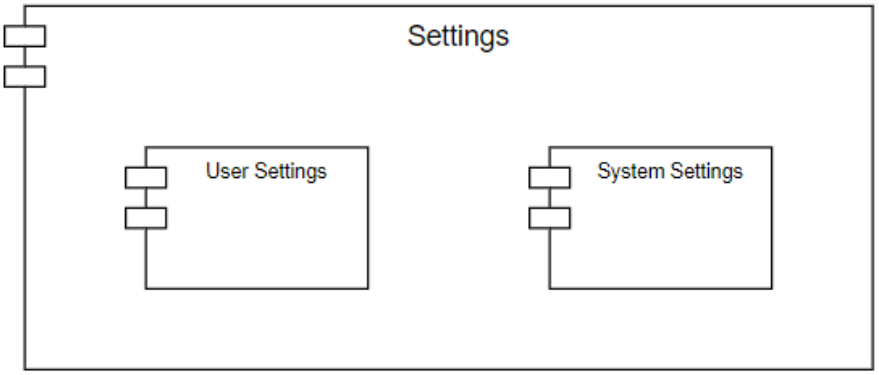

Figure 30: Structure and components of the Settings module

User Settings: this module has three parts

- **User Management:** allows to retrieve the list of users, create new ones and assign roles to users.
- **Biometric data:** Editing and viewing biometric data of the user. Users should be able to upload-record their Face model in order to be recognized by the system
- **Profiles:** allows to define different smart home user roles. For every role, it is possible to specify its rights and allowed/denied operations.

System Settings: this module offers general system-wide settings to control SIFIS-Home. The system settings are based on the definition of *policies*. The policy definition and evaluation follow an access control model based on ABAC (Attribute Based Access Control), which also considers mutable attributes for decisions that might change over time. The model exploited by the SIFIS-Home framework is the Usage Control (UCON), which extends the classical ABAC by including the possibility to revoke previously granted authorizations.

3.2.6 Input collection

Figure 31 presents the Input collection module that is used to collect various types of input data from the SIFIS-Home tenants and guests.

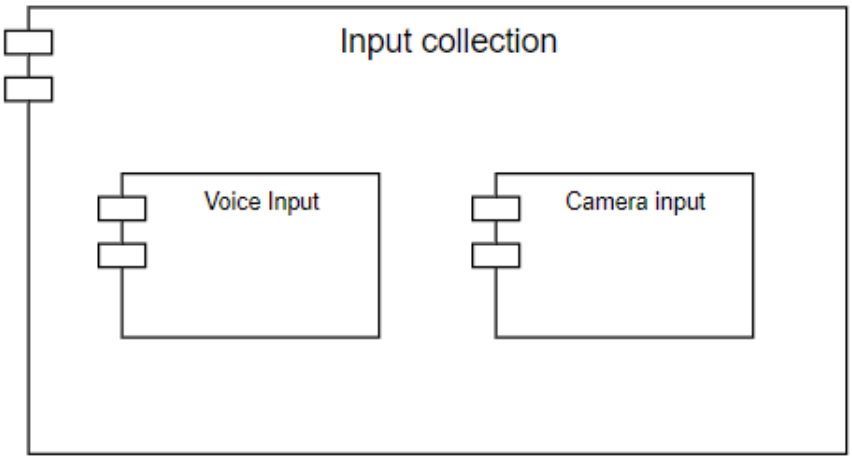

Figure 31: Components of the Input collection module

The Input collection module consists of the following components

- **Voice Input:** used to get voice commands from the user.
- **Camera Input:** used to get images from the camera to be used for various analytics tasks.

3.3 *SIFIS-Home NSSD Framework*

The SIFIS-Home NSSD Framework is the set of components that are expected to be present and every NSSD device that should be part of a SIFIS-Home enabled smart home. As illustrated in Figure 32, the SIFIS-Home NSSD Framework is composed of the Bootstrap Manager and Device API Manager components.

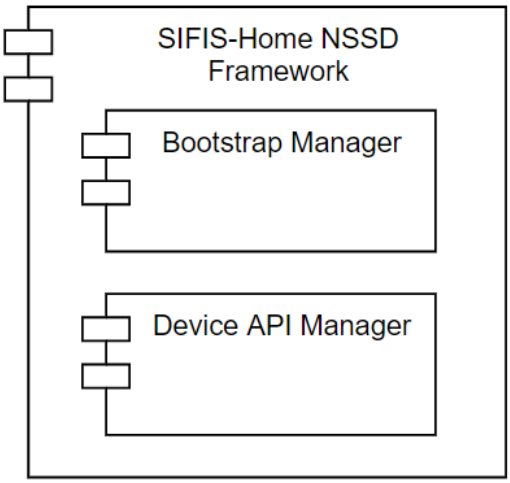

Figure 32: The SIFIS-Home NSSD Components

3.3.1 Bootstrap Manager

This component allows a NSSD device to receive all the information needed to allow it to join a SIFIS-Home network. In detail, the Bootstrap manager provides APIs that allow to i) set network connection information (e.g. SSID and password of the Wi-Fi network to which the NSSD device should connect to) and ii) set authentication credentials to access the NSSD so that only allowed applications are able to communicate and operate with the NSSD.

3.3.2 Device API Manager

This component offers the API through which a NSSD can be operated. In the case of a WebThings enabled NSSD, all the functionalities of the NSSD are offered by means of a set of properties, actions, events that can be set and retrieved by using standard Web-based protocols.

3.4 *SIFIS-Home Cloud Framework*

The SIFIS-Home Cloud framework is the set of SIFIS-Home components that reside and are executed on the SIFIS-Home cloud as illustrated in Figure 33. They are the following ones:

- **Marketplace:** it is a webservice component that provides a set of APIs through which it is possible to retrieve the list of SIFIS-Home 3rd party applications that can be installed on SDs. For every application it is reported the download URL as well its security assessment score. The marketplace webservice provides also developer APIs to upload and store 3rd party applications by part of external developers.
- **VPN Server:** a VPN server to which the SDs connect to. It is used to allow access to the SDs from the Internet even when the house is not provided with a public IP connection.

- **SIFIS-Yggio:** it is a specialized version of YGGIO, the SENSATIVE context broker. Its role is to provide a FIWARE compliant API to SIFIS-Home enabled smart homes. It communicates with the Fiware API component to receive messages published on the DHT and to send command to the NSSD devices of the smart home.
- **Home Registration Manager:** it provides an interface through which it is possible to create a new SIFIS-Home enabled house. When a new house is created, a VPN server and an Yggio instance dedicated to the house are created.

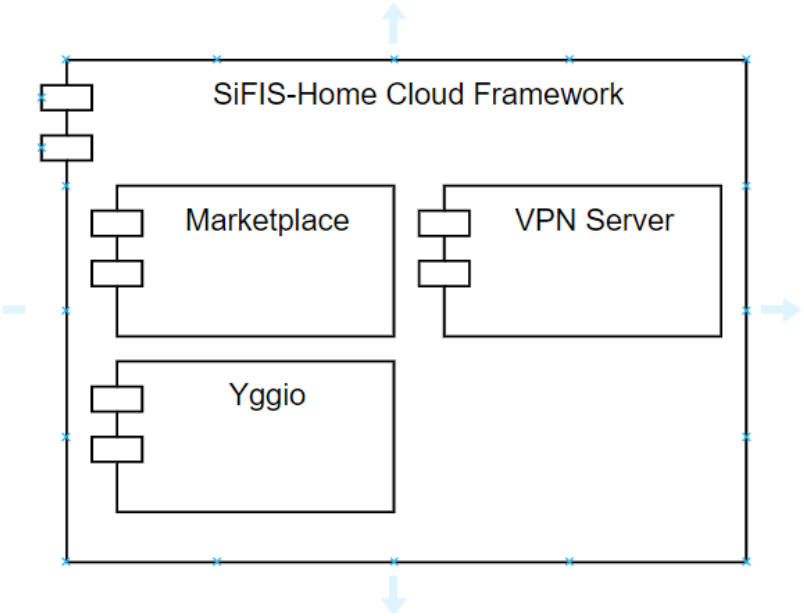

Figure 33: The SIFIS-Home Cloud Components

Picture above must get updated with “Home registration manager”

3.5 *SIFIS-Home Development Tools*

This component is the set of tools that are made available to developers of third party applications. As discussed, this component includes the toolboxes and the API set to interact with the marketplace described in the WP2 deliverables.

3.6 *Cryptography Management*

In SIFIS-Home, the assumption relevant for cryptography management is that both Smart Devices and NSSDs are honest and trusted. Thus, cryptography is thus used to protect the confidentiality and integrity of packets exchanged among smart devices and NSSDs by other entities. In fact, since the communications are based on wireless protocols, the system needs to be protected from eavesdropping and other attacks falling the sphere of Man-In-The-Middle attacks.

Key recovery attacks can rely on accessing the secret cryptographic material on a compromised device, or on exploiting weaknesses in the used cryptographic algorithms. As to the former approach, devices that can afford technologies such as Trusted Execution Environments should use those for securely storing and using secret cryptographic material.

Thus, having access to the cryptographic material from a key recovery attack implies that a device has been compromised and is misbehaving. Such issue does not fall under cryptography and is left for management through intrusion detection. Once a device is deemed as malicious, that device is isolated and all keys are regenerated.

3.6.1 OSCORE Security Protocols

SIFIS-Home relies on solutions for secure communication that in turn use standard, state-of-the-art cryptographic algorithms. Nevertheless, the following actions are taken to further mitigate the impact of potentially compromised keying material.

The following makes specific reference to the OSCORE Master Secret, that are used in the OSCORE and Group OSCORE security protocols to derive pairwise keying material and group keying material, respectively. Note that the OSCORE Master Salt is not intended to be secret.

When considering one-to-one communication protected with OSCORE, the two peers share the OSCORE Master Secret, from which they derive the keying material used to protect their communications with OSCORE.

If, irrespectively of the reason, the Master Secret gets compromised, the whole secure association between the two peers gets compromised. As soon as one of the two peers, say A, becomes aware of that, it terminates the secure association with the other peer B. After that, if A (B) still believes B (A) to be honest and non-compromised, then A (B) can establish a new secure association with B (A). Like when establishing the first secure association, this can happen, e.g., by using the authenticated key establishment protocol EDHOC, or the OSCORE profile of the ACE Framework for Authentication and Authorization in Constrained Environments.

As long as a secure association between A and B is legitimately ongoing, both A or B can securely update such association, e.g., by using the lightweight key update procedure KUDOS. In order to reduce the risk of key recovery attacks and of experiencing weaker security assurances, performing such an update is recommended with a regular cadence (depending on application policies) as well as upon approaching limits related to the usage of the keying material in the secure session. Such limits are peculiar of each different encryption algorithm, and typically concern the number of performed encryptions with the same symmetric key and the number of failed authenticated decryptions with the same symmetric key.

When considering one-to-many, group communication protected with Group OSCORE, all the peers in the security group share the same OSCORE Master Secret, from which they derive the keying material used to protect their communications with Group OSCORE. Together with other relevant parameters, the group members obtain the OSCORE Master Secret when joining the OSCORE group through the OSCORE Group Manager responsible for that group.

The Group Manager is also responsible for renewing the OSCORE Master Secret in the group (i.e., to perform a group rekeying). This can occur for a number of reasons:

- Periodically, according to application policies. Like for the case of one-to-one communication discussed above, this also limits the risk of key recovery attack.
- Upon a current group member's leaving, e.g., as a spontaneous group leaving or because compromised or suspected so. In such a case, the Group Manager does not provide the new OSCORE Master Secret to the leaving node, hence rekeying the group in such a way that logically evicts that node and ensures forward security.
- If the application requires backward security, the Group Manager rekeys the group upon a node joins as a new group member.

If, irrespectively of the reason, the OSCORE Master Secret gets compromised, the whole secure

association between the group members gets compromised. As soon a group member becomes aware of that, it stops using the secure association to communicate with the other group members. As soon as the Group Manager becomes aware of that, the Group Manager would rekey the group, hence all the group members can rely on a new OSCORE Master Secret to establish a new group secure association.
The Group Manager might specifically become aware of compromised group members, possibly through the assistance of an Intrusion Detection System. In this case, the group rekeying is performed in such a way to logically evict the compromised group members (see above).

The OSCORE Group Manager is not devoted to any particular group rekeying scheme. While it must support a basic, point-to-point rekeying scheme that leverages pairwise secure associations between the Group Manager and another single group member, the Group Manager can also take advantage of additional, more efficient rekeying schemes presented in the literature (e.g., based on key graphs or time-based sorting of administrative rekeying keys).

3.7 *Updates with respect to preliminary SIFIS-Home Architecture*

In this section we report a description of the updates with respect to the preliminary SIFIS-Home Architecture (discussed in deliverable D1.3).

In Table 1, we report all the components and subcomponents described in the preliminary architecture reported in deliverable D1.3. For each component (and subcomponent) we report the correspondent component (and subcomponent). We report and motivate the actions taken for each component in the architecture.

In summary, the following principal updates were applied on the SIFIS-Home Architecture:

- The SIFIS-Home API Gateway has been restructured, by defining only two sub-modules (Mobile Application API and 3Rd Party API) instead of 12. This modification has been performed in order to simplify the architecture and separate the two components according to their responsibility and not to the specific group of endpoints (APIs) offered.
- Six high-level components have been added to the architecture: SIFIS-Home Development Tools; SIFIS-Home Cloud Framework, SIFIS-Home NSSD Framework, DHT Manager, VPN Manager, NSSD Manager;
- The User Interface component of the original architecture has been moved outside the SIFIS-Home Application high-level macro-component and renamed SIFIS-Home Application Framework;
- The high-level component Secure Communication Layer has been removed from the level one architecture of the SIFIS-Home framework.

Table 1: Updates to component and subcomponents with respect to preliminary SIFIS-Home Architecture

Component and subcomponent in D1.3		Component and subcomponent in D1.4		Action
SIFIS-Home API Gateway	-	**SIFIS-Home API Gateway**	-	No action
SIFIS-Home API Gateway	Data Management API	-	-	Sub-modules merged into Mobile application API
SIFIS-Home API Gateway	Device Management API	-	-	Sub-modules merged into Mobile application API

SIFIS-Home API Gateway	DHT Management API	-	-	Sub-modules merged into Mobile application API
SIFIS-Home API Gateway	Application API	-	-	Sub-modules merged into Mobile application API
SIFIS-Home API Gateway	Policy Management API	-	-	Sub-modules merged into Mobile application API
SIFIS-Home API Gateway	Data Analysis API	-	-	Sub-modules merged into Mobile application API
SIFIS-Home API Gateway	Notification API	-	-	Sub-modules merged into Mobile application API
SIFIS-Home API Gateway	Communication API	-	-	Sub-modules merged into Mobile application API
SIFIS-Home API Gateway	Marketplace API	-	-	Sub-modules merged into Mobile application API
SIFIS-Home API Gateway	Home API	-	-	Sub-modules merged into Mobile application API
SIFIS-Home API Gateway	WoT Interfacing API	-	-	Sub-modules merged into Mobile application API and 3rd party API
SIFIS-Home API Gateway	Fiware Interfacing API	-	-	Sub-modules merged into Mobile application API and 3rd party API
User Interface	-	**SIFIS-Home Application Framework**	-	Component renamed and moved outside the SIFIS-Home Smart Device Framework
User Interface	Home	**SIFIS-Home Application Framework**	Home	No action
User Interface	Device Management	**SIFIS-Home Application Framework**	Device Management	No action
User Interface	Settings	**SIFIS-Home Application Framework**	Settings	Merged with Policy Manager
User Interface	Alarms / Logs	**SIFIS-Home Application Framework**	Alarms / Logs	No action
User Interface	Marketplace	**SIFIS-Home Application Framework**	Marketplace	No action
User Interface	Input Collection	**SIFIS-Home Application Framework**	Input Collection	No action
User Interface	Policy Manager	**SIFIS-Home Application Framework**	Settings	Merged with Settings
Secure Lifecycle Manager	-	**Secure Lifecycle Manager**	-	No action
Secure Lifecycle Manager	Application Manager	**Secure Lifecycle Manager**	Application Manager	No action
Secure Lifecycle Manager	Node Manager	**Secure Lifecycle Manager**	Node Manager	No action
Secure Lifecycle Manager	Authentication Manager	**Secure Lifecycle Manager**	Authentication Manager	No action
Secure Lifecycle Manager	Device Registration Manager	**Secure Lifecycle Manager**	Device Registration Manager	No action

Secure Lifecycle Manager	Key Manager	**Secure Lifecycle Manager**	Key Manager	No action
Secure Lifecycle Manager	System Protection Manager	**Secure Lifecycle Manager**	System Protection Manager	No action
Secure Communication Layer	-	**Secure Communication Layer**	-	No action
Secure Communication Layer	DHT Communication Manager	**DHT Manager**	Fiware API	Functions of the module moved into the DHT Manager module
Secure Communication Layer	External Communication Manager	**SIFIS-Home API Gateway**	3rd Party API	Functions of the module moved into the SIFIS-Home API Gateway Module
Secure Communication Layer	Secure Message Exchange Manager	**Secure Communication Layer**	Secure Message Exchange Manager	No action
Secure Communication Layer	Network Protection Manager	-	-	Functions of the module distributed into the individual components needing network protection features
Secure Communication Layer	Content Distribution Manager	**Secure Communication Layer**	Content Distribution Manager	No action
Application Toolboxes	Policy Enforcement Engine	**Application Toolboxes**	Policy Enforcement Engine	No action
Application Toolboxes	Data Analysis Toolbox	**Application Toolboxes**	Data Analysis Toolbox	No action
Application Toolboxes	Anonymization Toolbox	**Application Toolboxes**	Anonymization Toolbox	No action
Proactive Security Management Layer	-	**Proactive Security Management Layer**	-	No action
Proactive Security Management Layer	API Monitor	**Proactive Security Management Layer**	Monitor	Merged into the “Monitor” component
Proactive Security Management Layer	DHT Monitor	**Proactive Security Management Layer**	Monitor	Merged into the “Monitor” component
Proactive Security Management Layer	Network / System Monitor	**Proactive Security Management Layer**	Monitor	Merged into the “Monitor” component
Proactive Security Management Layer	Self Healing	**Proactive Security Management Layer**	Self-Healing	No action
Proactive Security Management Layer	Distributed Trust	**Proactive Security Management Layer**	Distributed Trust	No action
Proactive Security Management Layer	Evaluator / Notifier	-	-	Functions of the module distributed into the individual components
Distributed Data Storage	-	**DHT Manager**	DHT	Module moved in DHT manager

4 Mapping between Functional Requirements and SIFIS-Home Architecture

In Table 2, we report a mapping between the Functional Requirements of the SIFIS-Home Framework, described in deliverable 1.2, and the components of the architecture designed in the present deliverable.

Table 2: Mapping between functional requirements and architecture components

ID	Req. description	Architecture Component
F-01	The SIFIS-Home framework shall provide means of identifying the resident users and administrators inside the smart home through biometrics.	Input collection, System Protection Manager
F-02	The SIFIS-Home system shall provide means of authentication to resident users, administrators and guest users inside the smart home.	Authentication Manager
F-03	The SIFIS-Home system shall match read biometrics against a database of stored ones, in order to assess authentication.	Authentication Manager
F-04	The system shall make different features available and accessible to different users, based on their authenticated identity	Authentication Manager
F-05	The system shall activate a guest profile when the identity of the biometrics is not recognised.	Authentication Manager
F-06	The SIFIS-Home system shall provide Automatic Speech Recognition (ASR) to provide resident users and administrators the facility to control their home appliances through their speech.	Input Collection
F-07	The SIFIS-Home system shall have means to receive and interpret the voice commands provided by the user, and it shall be able to interpret those commands belonging to a predefined command set	Input Collection
F-08	The SIFIS-Home system shall be able to execute a predefined set of actions in response to a predefined set of recognizable voice commands	Input Collection
F-09	The SIFIS-Home system shall signal the presence of an intruder when the identity is not recognised and no residents are at home.	Input Collection, System Protection Manager
F-10	The SIFIS-Home system, following the detection of an intruder, shall track the intruder and attempt again to identify him/her	Input Collection, System Protection Manager
F-11	The SIFIS-Home system shall store the identity of the intruder if the face is recognized. If the face is not recognized, the video and audio recordings must be stored from the system as well	Input Collection, Alarms / Log, System Protection Manager
F-12	The SIFIS-Home system, following the detection of an intruder, shall track the intruder and attempt again to identify him/her.	Input Collection, Alarms / Log, System Protection Manager
F-13	The SIFIS-Home system may grant the access to recording to the maintainer.	Input Collection, Alarms / Log
F-14	The SIFIS-Home system may allow administrators and resident users to contact police to receive assistance in case of intrusions	Input Collection, Secure Message Exchange Manager
F-15	The SIFIS-Home system shall provide means of identifying anomalous situations and behaviours inside the smart home	Input Collection, Policy Enforcement Engine, Alarms / Log
F-16	The SIFIS-Home system shall provide means of recognition of allowed users in unusual locations or performing dangerous actions, and signal them to resident users and administrators.	Input Collection, Policy Enforcement Engine, Alarms / Log
F-17	The SIFIS-Home system shall provide means of recognition of prohibited objects inside the smart home and signal resident users and administrators.	Input Collection, Policy Enforcement Engine, Alarms / Log
F-18	The SIFIS-Home system shall provide means of recognition of allowed objects inside the smart home in unusual positions, and signal resident users and administrators.	Input Collection, Policy Enforcement Engine, Alarms / Log
F-19	The SIFIS-Home system shall detect, identify and disconnect infected devices.	Device Management
F-20	The SIFIS-Home system shall notify resident users and administrators when malware is detected.	Alarms / Log
F-21	The SIFIS-Home system shall be able to execute self-healing algorithms to transfer functionalities of devices that have been disconnected for security reasons to the others.	System Protection Manager
F-22	The SIFIS-Home system should allow means of verifying that the malware has not spread to other devices.	System Protection Manager
F-23	The SIFIS-Home system shall allow the resident user to register more components to the	Device Management

	system.	
F-24	The SIFIS-Home system shall allow the resident users and administrators to visualize a list of the registered devices, along with their characteristics.	Device Management
F-25	The SIFIS-Home system shall allow the administrators and device owners to unregister from the system a registered component.	Device Management
F-26	The SIFIS-Home systems shall expose a section where the device owners and administrators can configure the devices.	Device Management
F-27	The SIFIS-Home system shall prompt the administrator when unsolicited configuration changes are propagated to the devices.	Alarms / Log
F-28	The SIFIS-Home system must provide a marketplace function for the download of third-party applications on smart devices.	Marketplace
F-29	The SIFIS-Home system shall retrieve and provide information about the safety and security aspects of an application to the user.	Application Manager
F-30	The SIFIS-Home system must provide a feature to show the administrators a list of currently active policies.	Policy Manager
F-31	The SIFIS-Home system must provide a feature to show the administrators a list of currently active policies	Policy Manager
F-32	The SIFIS-Home system must allow the administrator to configure policies for (groups of) users.	Policy Manager
F-33	The SIFIS-Home system must allow the administrator to configure policies for (groups of) devices.	Policy Manager
F-34	The SIFIS-Home system must allow the administrator to view the policies related to features and/or resources either permitting or denying access or usage to (groups of) users.	Policy Manager
F-35	The SIFIS-Home system must allow the administrator to see the list of features/resources that are allowed or forbidden for all groups of devices.	Device Management, Policy Manager
F-36	The SIFIS-Home system must provide the user with a feature to list all the currently available profiles.	Policy Manager, Settings
F-37	The SIFIS-Home system must allow the user to configure his/her profiles.	Settings
F-38	The SIFIS-Home system may allow the user to switch his/her current profile.	Settings
F-39	The SIFIS-Home system should show the user a summary of the preferences associated to its current profile.	Settings
F-40	The SIFIS-Home system should show notifications to the user when the current profile is changed.	Settings, Alarms / Log
F-41	The SIFIS-Home system should offer aggregate analytics and statistics about the usage and behaviour of devices to the administrator.	Data Analysis Toolbox
F-42	The SIFIS-Home system should offer aggregate analytics and statistics about the usage of profiles to the administrator.	Data Analysis Toolbox
F-43	The SIFIS-Home system must offer remote authenticated and secure log-in features to configurer/maintainers of user profiles.	Authentication Manager
F-44	The SIFIS-Home system shall offer to the maintainers a panel with the remote homes he/she can manage.	Settings
F-45	The SIFIS-Home system must offer the maintainer an interface with the possibility to call the authorities or alert the administrator and residents, in case of intrusions.	Settings, Secure Message Exchange Manager
F-46	The SIFIS-Home system shall allow the residents to store personal content (video, audio, text).	Input Collection, Content Distribution Manager
F-47	The SIFIS-Home system must be able to map a policy defined by the administrator into one or more device-level policies.	Policy Manager, Device Management
F-48	The SIFIS-Home should be able to map the device-level policies with the capabilities of the involved devices.	Policy Manager, Device Management
F-49	The SIFIS-Home must be able to apply the active device-level policies to the actual devices.	Policy Manager, Device Management
F-50	The SIFIS-Home must be able to apply the active device-level policies when needed.	Policy Manager
F-51	The SIFIS-Home should notify when a device-level policy cannot be mapped onto any device.	Policy Manager, Alarms / Log
F-52	The SIFIS-Home should be able to identify redundant or conflicting policies.	Policy Manager
F-53	The description of the policies must be available to administrators, maintainers and tenants of the SIFIS-Home system.	Policy Manager

F-54	Administrators and configurers shall be able to create, configure and delete security groups.	Policy Manager, Authentication Manager
F-55	Administrators and configurers shall be able to register security groups and thus make them dynamically discoverable	Policy Manager, Node Manager
F-56	There must be a means for Administrators and devices to discover security groups, including their properties, how to join them, as well as their associations with application groups and their resources.	Application Manager, Device Management, Policy Manager
F-57	There must be a means for devices to join/leave a security group and retrieve/provide updated key material to communicate in the group	Policy Manager, Secure Message Exchange Manager

5 SIFIS-Home APIs

In the following sub-sections, we provide the APIs that have been defined for the set of blocks of the SIFIS-Home architecture.

We identify a separation between two different levels of APIs:

- Application APIs, that are going to be used by the house control application and the configuration portal, and are defined as REST APIs. We used the OpenAPI specification (https://www.openapis.org/) and Swagger Editor (https://editor.swagger.io/) to define the APIs. The swagger file is reported in Appendix A. OpenAPI is the de-facto industrial standard to define REST based APIs.
- Internal APIs, that are utilized by the SIFIS-Home architecture components as a means of internal communication and are defined as SDKs.

5.1 *Home APIs*

APIs used to authenticate users and get all the information that should be present in the house control dashboard.

5.1.1 Login API

- *Description:* API to get an authorization token using username and password. The API returns as a parameter the Authorization token to be used in further API invocations.
- *URI*: /home/login
- *HTTP method*: GET
- *Header Parameters:* username (string)
  password (string)
- *Responses:* - 200 OK
  Response schema: Authorization token (string)
  - 401 “Login Failure”

5.2 *Communication APIs*

APIs used to retrieve logs, alerts and messages generated by the SIFIS-Home system.

5.2.1 Messages

- *Description:* The API returns a list of messages. It returns the messages belonging to the topic to which the client is registered. The API allows an optional additional filter to be provided as a parameter.
- *URI*: /messages
- *HTTP method*: GET
- *Header Parameters:* authorization (string) - *authorization token of the client requiring the data feed*
- *Query Parameters*: searchstring (string) - *Optional filter string for the messages to be returned*
- *Responses:* - 200 “OK”
  Response schema: Message (object)
  - 401 “Login Failure”
  - 404 “No messages found”

5.2.2 Message Feed Register

- *Description:* The API allows to subscribe to data from alarms and activity logs pertaining to a certain topic. The API receives the topic to which the user wants to subscribe.
- *URI*: /messages/messageFeedRegister/{topic}
- *HTTP method*: POST
- *Header Parameters:* authorization (string) - *authorization token of the client willing to perform the registration*
- *Path Parameters*: topic (string) - *Topic of the required feed*
- *Responses:* - 200 “OK”
  Response schema: Message (object)
  - 401 “Login Failure”
  - 404 “Topic not registered”

5.2.3 Message Feed Unregister

- *Description:* The API allows to disable the subscription to data from alarms and activity logs pertaining to a certain topic. The API allows to disable the subscription to data from alarms and activity logs pertaining to a certain topic.
- *URI*: /messages/messageFeedUnregister/{topic}
- *HTTP method*: post
- *Header Parameters:* authorization (string) - *authorization token of the client willing to unregister*
- *Path Parameters*: topic (string) - *Topic of the feed*
- *Responses:* - 200 “OK”
  - 401 “Login Failure”

- 404 “Topic not registered”

5.2.4 Stream Camera Feeds

- *Description:* The API allows to access data from alarms and activity logs.
- *URI*: /messages/streamCameraFeeds/{topic}
- *HTTP method*: GET
- *Header Parameters:* authorization (string) - *authorization token of the client willing to stream the camera feed*
- *Path Parameters*: topic (string) - *Topic of the desired camera feeds*
- *Responses:* - 200 “OK”
  Response schema: CameraFeed (object)
  - 401 “Login Failure”
  - 404 “Topic not registered”

5.3 Device Management APIs

- APIs used to manage the list of devices present in the house. They allow to retrieve the list of devices that are present in the house as well as add/remove one device from the list of *)information about the devices*
- *Responses:* - 200 “OK”
  Response schema: Device (object)
  - 401 “User unauthorized”
  - 404 “Devices not found”

5.3.1 Add Favourite Device

- *Description:* The API receives the ID of a device that has to be put into the set of favourites
- *URI*: devices/addFavouriteDevice/{deviceID}
- *HTTP method*: POST
- *Header Parameters:* authorization (string) - *authorization token of the client willing to add the favourite device*
- *Path Parameters*: deviceID (string) - *ID of the device to be added to the favourites*
- *Responses:* - 200 “OK”
  - 401 “User unauthorized”
  - 404 “Device not found”

5.3.2 Remove Favourite Device

- *Description:* The API receives the ID of a device that has to be removed from the set of favourites

- *URI*: devices/removeFavouriteDevice/{deviceID}
- *HTTP method*: DELETE
- *Header Parameters:* authorization (string) - *authorization token of the client willing to remove the favourite device*
- *Path Parameters*: deviceID (string) - *ID of the device to be removed from the favourite devices set*
- *Responses:*
  - 200 “OK”
  - 401 “Login Failure”
  - 404 “Device not found”

5.3.3 Favourite Devices

- *Description:* The API returns information about all devices that are saved as favourite in the system
- *URI*: /devices/favouriteDevices
- *HTTP method*: GET
- *Header Parameters:* authorization (string) - *Authorization Token of the client requiring information about the favourite devices*
- *Responses:*
  - 200 “OK”
    Response schema: Device (object)
  - 401 “Login Failure”
  - 404 “Topic not registered”

5.4 *Application Manager APIs*

The Application Manager APIs provides operations to manage third-party applications. They allow to install/remove a third-party application as well as kill it and remove its data.

5.4.1 Install Application

- *Description:* The API is used to add a new third-party application to the SIFIS-Home system. It receives as parameters: i) Activate (indicates if the application should be started after its installation), ii) the application ID (identifier of the app to be added), iii) the application name and iv) application specific settings.
- *URI*: /applications/installApplication/{applicationId}
- *HTTP method*: POST
- *Header Parameters:* authorization (string) - *Authorization Token of the client requiring to install the application*
  Activate (boolean) - *Defines if the activation of the application has to be performed after the installation*
  Settings (string) - *Application-specific settings passed to the installation function*

- *Path Parameters:* applicationID (integer) - *ID of the application to install*
- *Responses:* - 200 “OK”
  - 401 “Client unauthorized”
  - 404 “Application not found”

5.4.2 Remove Application

- *Description:* The API is used to remove a third-party application. It receives as parameter the identifier of the app to be deleted. It returns an HTTP response with the operation result.
- *URI*: /applications/removeApplication/{applicationId}
- *HTTP method*: DELETE
- *Header Parameters:* authorization (string) - *Authorization Token of the client requiring to remove the application*
- *Path Parameters:* applicationId (integer) - *ID of the application to remove*
- *Responses:* - 200 “OK”
  - 401 “Client unauthorized”
  - 404 “Application not found”

5.4.3 Kill Application

- *Description:* The API is used to kill an active third-party application. It receives as parameter the id (identifier of the app to be killed). It returns an HTTP response with the operation result.
- *URI*: /applications/killApplication/{applicationId}
- *HTTP method*: GET
- *Header Parameters:* authorization (string) - *Authorization Token of the client willing to kill the application*
- *Path Parameters:* applicationId (integer) - *ID of the application to kill*
- *Responses:* - 200 “OK”
  - 401 “Client unauthorized”
  - 404 “Application not found”

5.4.4 Wipe Application

- *Description:* The API is used to wipe all the data of an application which is already installed in the solution. It receives as parameter the id (identifier for the app whose data has to be wiped). It returns an HTTP response with the operation result.
- *URI*: /applications/wipeApplication/{applicationId}
- *HTTP method*: GET
- *Header Parameters:* authorization (string) - *Authorization Token of the client willing to wipe the application data*

- *Path Parameters:* applicationId (integer) - *ID of the application whose data has to be wiped*
- *Responses:* - 200 “OK”
  - 401 “Client unauthorized”
  - 404 “Application not found”

5.5 *Device Management API*

Internal APIs required to manage the registered devices.

Request Authorization Credentials

- Method: RequestAuthCredential()
- Return type: result(b) [true: success, false: failure], credential(s) [authorization credential]
- Parameters: credentialTarget (s), credentialIssuer (s), scope (s) , [securitySpecificParameters (s), profileSpecificParameters (s)]
- Description: it is called by a subject to perform certain operations at certain resources of certain target network nodes. The API relates to the “Authentication manager component”. It receives the credential target, issuer, scope of the authorization, and security and profile-specific parameters, and returns the operational result (Boolean) and the authorization credential (if successful).

Upload Authorization Credentials

- Method: UploadAuthCredential()
- Return type: result(b) [true: success, false: failure], parameters(s)
- Parameters: credentialTarget (s), authorizationCredential (s), [securitySpecificParameters (s), profileSpecificParameters (s)]
- Description: The API relates to the “Authentication manager component”. It receives the credential target, the authorization credential, and security and profile-specific parameters, and returns the operational result (Boolean) and the security and profile specific related parameters. Note that an access request is automatically checked against the authorization credential at the credential target, when later receiving a request for resource access from the subject.

Check Status of Authorization Credentials

- Method: CheckAuthCredentialStatus()
- Return type: result(b) [true: success, false: failure], parameters(s)
- Parameters: authorizationCredential(s) , filter (s)
- Description: to perform certain operations at certain resources of certain target network nodes. The API relates to the “Authentication manager component”. It receives an authorization credential, the identifier of authorization credential and filter criteria. It returns as output the operation result, and security and profile specific parameters.

Perform the exchange of a security secret

- Method: ExchangeSecuritySecret()

- Return type: result(b) [true: success, false: failure], [securitySecret(s), securityInformation(s) [information derived from the security secret], extAuthData(s) [external authorization data]]
- Parameters: targetDevice(s), identityCredential (s), establishmentParameters (s), [externalAuthorizationData(s)], [wrappingContainer]
- Description: to establish an (end-to-end) security association with another network node. The API relates to the "Key Manager" component. It receives as input the target device, an identity credential, and establishment-specific parameters. It receives optionally external authorization data. As output, it returns the operation result, the security secret, and optionally information derived from the security secret and external authorization data.

Perform an update of pairwise key material shared with another network node

- Method: UpdatePairwiseKeyMaterial()
- Return type: result(b) [true: success, false: failure]
- Parameters: targetDevice(s)
- Description: The API relates to the "Key Manager" component. It receives as input the target device, identified with a string parameter, and returns as output the operational result.

Join a Security Group

- Method: JoinSecurityGroup()
- Return type: result(b) [true: success, false: failure], groupKey(s) [group key]
- Parameters: groupManager(s), groupName(s), roles (s), publicKey (s), [requiredInfo(s)
- Description: retrieving the group key material and related parameters; **further material/information retrieval** as group member; **leaving the security group.** The API relates to the "Key Manager" component. It receives as input the Group Manager of the group, name of the group, roles wished in the group, [own public key], [interest for other group information]. It produces as output the operation result (Boolean), group key material, [group members' public keys], [additional group information].

Discovery of network nodes and their resources

- Method: DiscoveryNetworkNodes();
- Return type: list<s>
- Parameters: targetDiscoveryRequest(s), [targetAttributes(s)]
- Description: The API relates to the "Key Manager" component. It receives the target of the discovery request and optional attributes for the request. It returns the list of the links to the discovered resources

Registration of network nodes and their resources (e.g. at Node Discovery)

- Method: RegisterNetworkNodes()
- Return type: result(b) [true: success, false: failure]
- Parameters: targetRegistrationRequest(s), links (list<s>)
- Description: The API relates to the "Key Manager" component. It receives as parameters the target of registration request, and list of links to register. It returns as output the operation result (a Boolean value).

Bootstrapping of a network node (e.g. at a LwM2M Bootstrap Server)

- Method: BootstrapNetworkNode()

- Return type: result(b) [true: success, false: failure], String [target server], String [security-mode specific information]
- Parameters: targetBootstrapServer(s), [securityModeSpecificInformation(s)]
- Description: The API relates to the "Key Manager" component. It receives as input the target bootstrap server, and [security-mode specific information]. It returns as output the operational result, the target device manager server, and security mode specific information.

Registration of a network node (e.g. at a LwM2M Device Manager Server)

- Method: RegisterNetworkNode()
- Return type: Boolean [true: success, false: failure], String [configuration parameters]
- Parameters: targetDeviceManagementServers(s), securityModeInformation(s)
- Description: The API relates to the "Key Manager" component. It receives as parameters the target Device Manager Server, and [security-mode specific information]. It gives as output the operation result, and configuration parameters

Registration of a new smart device in the SIFIS-Home architecture

- Method: RegisterNewSmartDevice()
- Return type: result(b) [true: success, false: failure]
- Parameters: String [digest], String [metadata]
- Description: This API is the high-level interface which triggers the joining of a new Smart Device to the SIFIS-Home architecture. The digest parameter is used to verify the integrity of the SIFIS-Home framework instance installed on the device, whilst the metadata provide information on the device type and functionalities.

5.6 *WoT Interfacing API*

The WoT interfacing API work as end point to query the devices exposing their functionalities through a WoT interface. A sample API is reported in the following:

Call WoT Turn Light On

- Method: CallWoTTurnLightOn() devices/lamp/{id}/{command} in this case turnOnLight
- Return type: result(b) [true: light turned on, false: error]
- Parameters: [lightID (f), roomID (f)], intensity (f), colour (f)]
- Description: It calls the WoT implementation of the API to turn on a specific light bulb, identified by the light-id parameter or all the lights in a specific room. If the first parameter is missing all available lights will be turned on. Intensity and colour can be used for configurable light bulbs and are ignored if the light bulb is not configurable. The invoked API returns *true* if the light has been turned on, or *false* if the light is already on, is not reachable or reported as not working.

5.7 *Secure Communication Manager*

This API allows to request the transmission of messages to other network nodes. Corresponding Responses/Acknowledgments/Reset messages are handled by the communication stack.

Send a Message

- Method: SendMessage()

- Return type: result(b) [true: success, false: failure]
- Parameters: data (s), transferParameters (s), securityProtocolParameters (s), targetRecipient(s)
- Description: This API relates to the "Content Distribution Manager" component and the "Secure Message Exchange Manager" component. The API is used to send a message to a recipient. It receives as parameters the data to send, transfer- and security-protocol specific parameters, the target recipient(s). It gives as output the operation results (a Boolean value).

Abort a Message

- Method: AbortMessage()
- Return type: result(b) [true: success, false: failure]
- Parameters: sentMessage (s)
- Description: This API relates to the "Content Distribution Manager" component and the "Secure Message Exchange Manager" component. The API is used to abort a message sending. It receives as parameters the pointer to a sent message or the corresponding transmission. It gives as output the operation results (a Boolean value).

6 Operative Workflows of main SIFIS-Home Operations

This section reports the final set of operative workflows related to the operations that will be supported by the SIFIS-Home framework. The final set of operative workflows is an extension of the preliminary set reported in D1.3.

6.1 *Register New Home*

This workflow, illustrated in Figure 34, is the first operation which is performed by an Administrator to register his own smart home as a SIFIS-Home instance. In order to register a new Home, the user should own a SIFIS-Home Smart Device, PC or smartphone. The user needs to download the SIFIS-Home mobile application on a smartphone or tablet, or access the SIFIS-Home application through the web portal and then register a new account on the SIFIS-Home Cloud. This triggers the generation of a dedicated VPN Server with a dedicated DNS record. Then an instance of the SIFIS-Home Smart Device framework is installed on the associated smart device and a DHT shared key is generated and stored in the smart device. Finally, Yggio is updated accordingly associating the user to the SIFIS-Home instance.

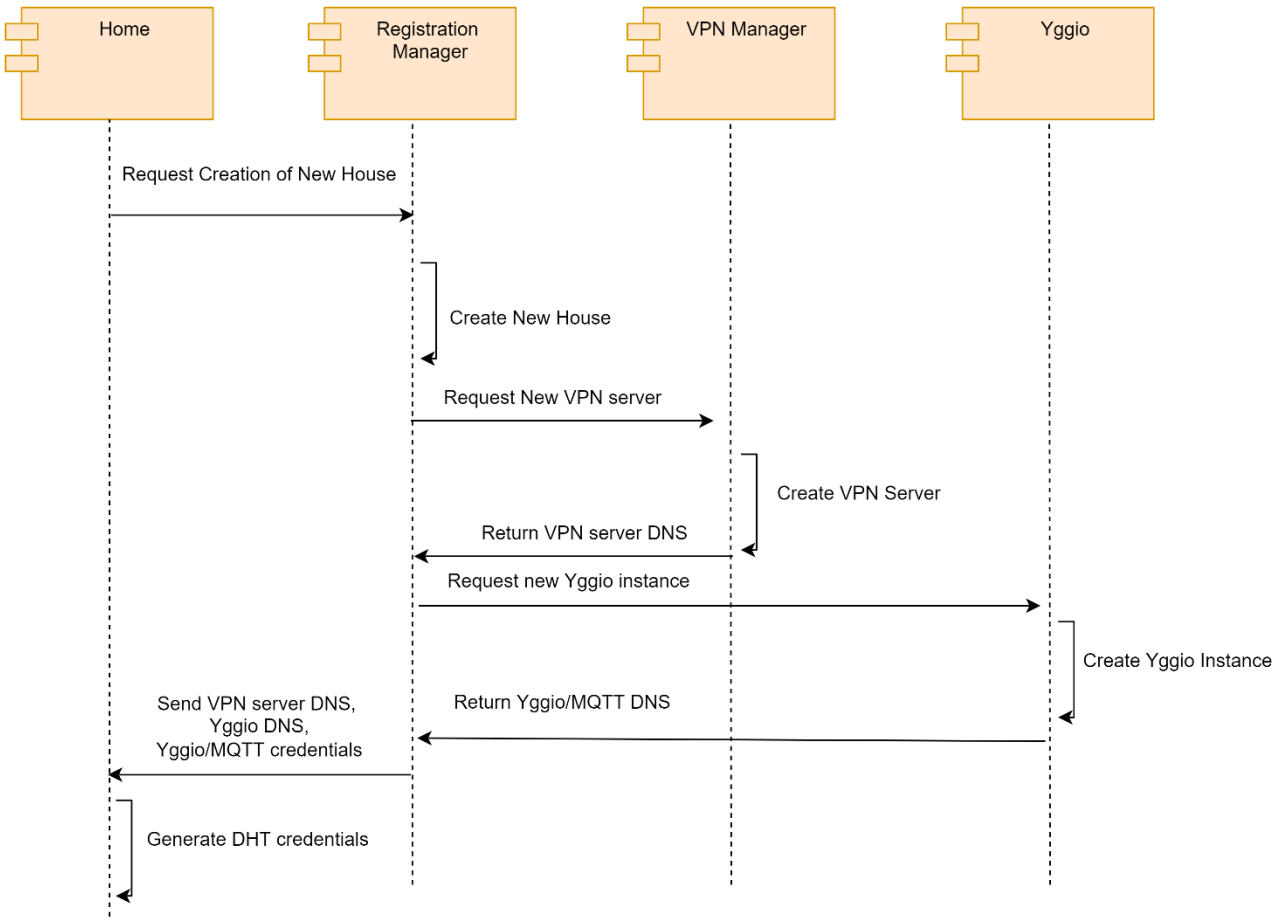

Figure 34: Register new home workflow.

6.2 *Register New Smart Device [UC05]*

This workflow, illustrated in Figure 35, is used to add a new smart device to a SIFIS-Home instance. This operation can also be used to register the first smart device after that a new house has been created. The needed devices for performing the registration of a new smart device are the smartphone/tablet used to create the new house and the smart device itself. The presence of the smartphone ensures that the registration is intentional and actually triggered by the authorized user. First of all, the user should install the SIFIS-Home SD Framework on the new device (if the device provided by SIFIS, the framework is already installed) and start the smart device in access point mode. Then it will use the mobile app to select the Add New Device option in the Device Manager panel. Then the mobile application asks the user to provide an identifier of the smart device (e.g. QR-Code) and the smart device receives from the mobile application a digest to verify the integrity of the software. Once the integrity is verified, the smart device uses the Key Manager to generate and store a pair of keys (public key). The public key is then sent to the Mobile Application for the establishment of a secure channel. Through this channel, the Mobile Application sends the DHT and Yggio MQTT credentials to the smart device. If this is the first smart device of the house, it generates the DHT, receives an Identifier and updates the list of devices also in the cloud by using the FIWARE API component. Otherwise, the smart device asks to the DHT to elect a leader node, which will perform the registration of the smart device and will assign an identifier to it. As before the device is also registered in the cloud to be accessible through Yggio remotely. This workflow satisfies the Use Case UC05.

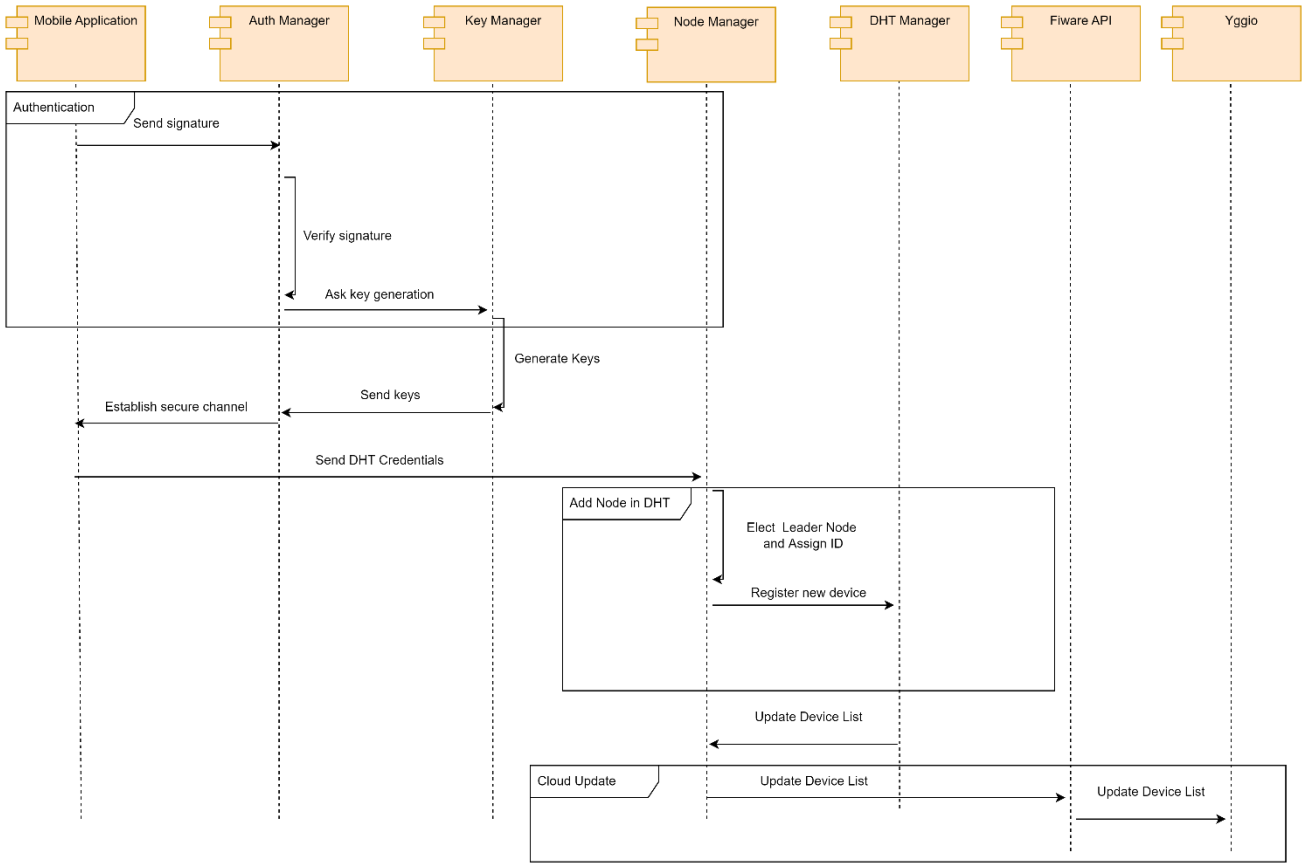

Figure 35: Register new smart device workflow.

6.3 *Register New NSSD*

This workflow, illustrated in Figure 36, details the installation of a new NSSD in a smart home instance. Also this workflows implements the use case UC05.

Assumptions: The NSSD comes with the SIFIS-Home NSSD framework. The NSSD has a Wi-Fi interface.

- NSSD connects to a default NSSD join network.
- The NSSD exposes a number of REST endpoints that allow a SD to give to the NSSD, the Wi-Fi credentials to use as well as a shared key that should be used to control the NSSD after the join procedure is completed.
- Device is registered on the DHT by the SD Node that performed the NSSD registration, and DHT is synchronized.
- The shared key to communicate with the NSSD should be stored into the DHT.

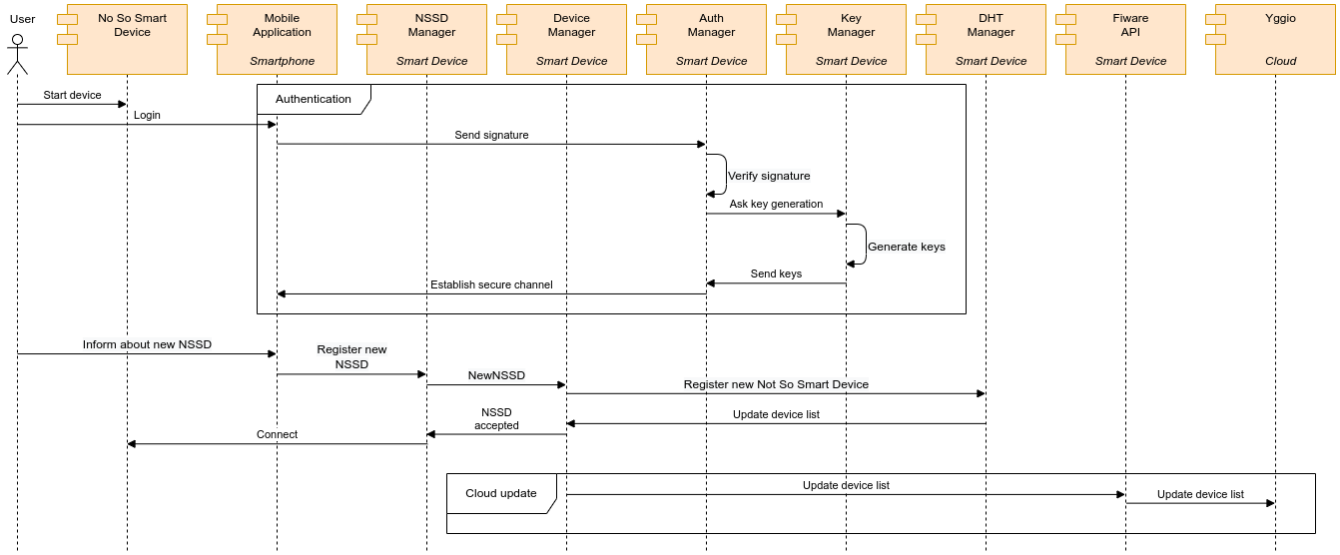

Figure 36: Register new NSSD workflow.

6.4 *Register New NSSD Using WoT [UC05]*

This workflow, illustrated in Figure 37, reports the registration of a NSSD which offers a Web of Things interface. The assumption for this workflow is that there is already at least a smart device in the SIFIS-Home instance. The NSSD when turned on will start an advertisement procedure to find a smart device to which it should be connected. The advertisement is interpreted by the WoT Manager in the NSSD manager. The WoT manager will then retrieve the Thing Description from the NSSD, extracting thus capabilities and resources, and associates to it an NSSD identifier. The smart device becomes responsible for the NSSD and the association between the NSSD identifier is stored in the DHT.

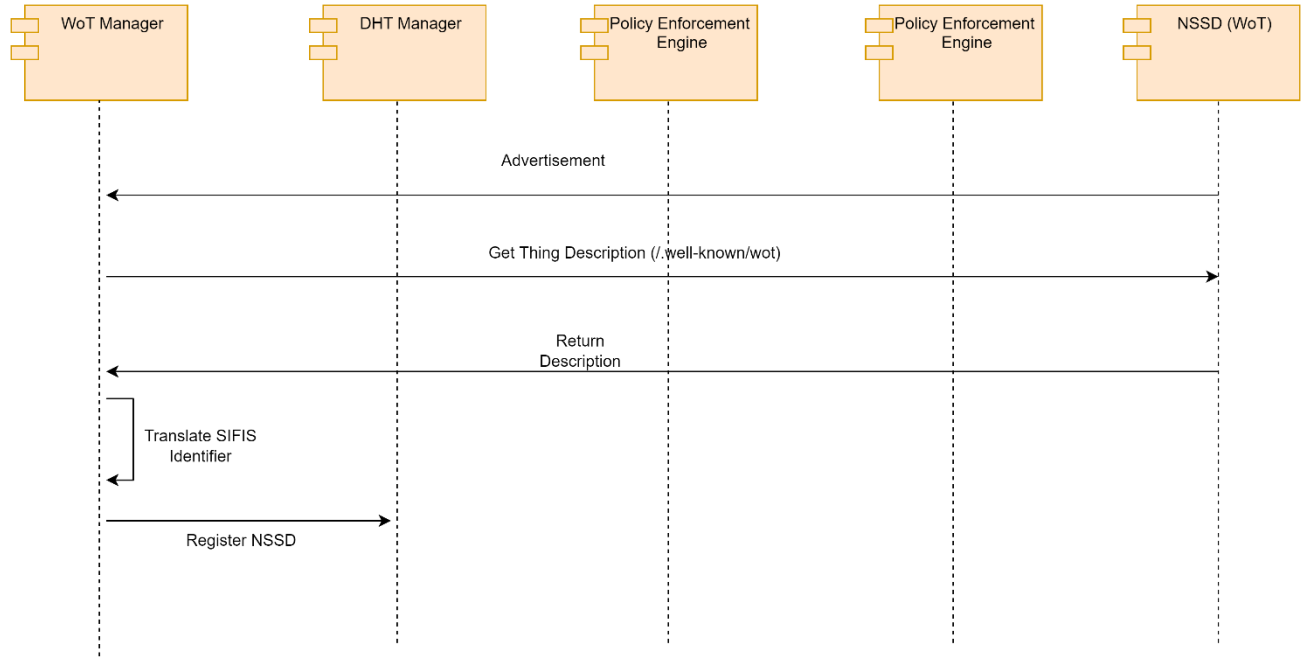

Figure 37: Register new NSSD Using WoT workflow.

This workflow enables the use case UC05.

6.5 *Control Resource on NSSD via Third Party App [UC03, UC12]*

This workflow, illustrated in Figure 38, describes the control of a NSSD based on WoT by using a

third party application. The workflow is initiated by the third party app, installed on a smart device, which looks for a specific resource type (e.g. a light) by querying through the WoT manager of the NSSD manager, the available resources. The resources are stored in the DHT and the query result is returned. Before the resource list is returned to the app, the SIFIS-Runtime checks the manifest to verify which resources are declared as relevant for this app. Thus, a filtered list is returned. Once the third party app has found the relevant resource(s), it can subscribe on the DHT to the topic related to this resource to receive status information and update the GUI (if any) accordingly. Now the request to operate the resource can be issued. The operation is performed by publishing on the topic of the resource a "turn ON" command. The DHT intercepts the publish and converts the topic to "Policy Engine" so that the request can be evaluated. Supposing a PERMIT response is returned, the status of the resource is changed on the DHT and the command to operate it is sent to the NSSD. If the operation is successful, the third party app is notified.

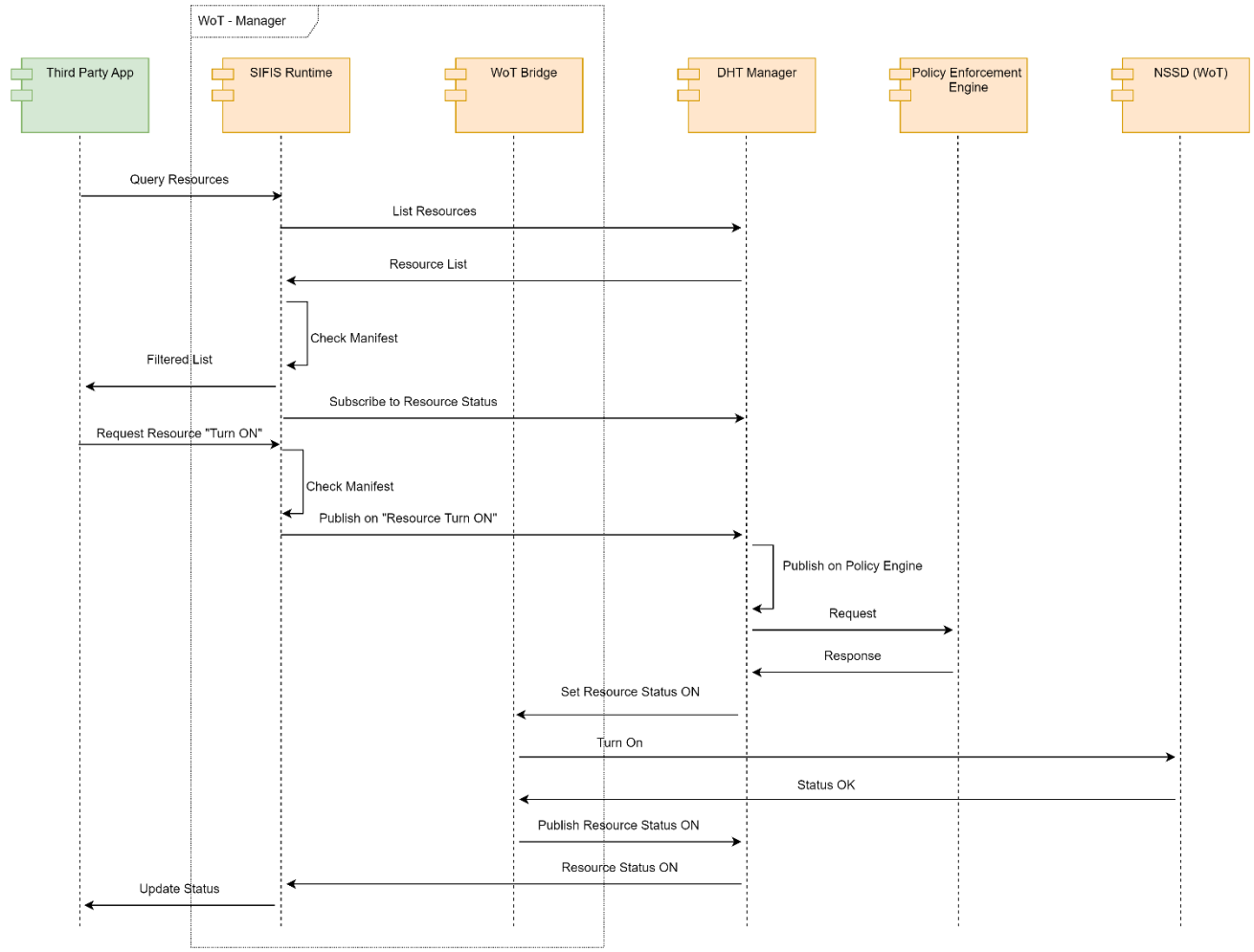

Figure 38: Third party app control of a NSSD resource

This workflow satisfies the use case UC05.

6.6 *Register New User, Set Role and assign its Settings [UC01, UC07, UC10]*

This workflow, illustrated in Figure 39, describes the procedure of the registration of a new user, following by the definition of its role and the assignation of its settings. The workflow is initiated by the administrator, with the login on the mobile application. After logging in, the administrator selects the Settings screen in the mobile app, and provides the information (name and role) for a new user to be added. The mobile applications at this point performs an authentication with the Authentication Manager module, with administrator privileges. The Policy Enforcement Engine checks the

authorization for the administrator and the mobile app to add a new user. Once the creation of the user is authorized, the Key Manager module generates the credentials for the new user. The user is then registered in the DHT, and his role and credentials are encrypted and stored. The system sends a notification to the administrator, and the new user is provided with a default password to be used for the first login in the system.
At this point, the new user can log in to the system, through an instance of the Mobile Application. The Authentication Manager receives the user's credentials, and provides access. The mobile application asks the user to insert a new password: once the user provides it, a new key pare is generated and the credentials in the DHT are updated. If the user does not log-in and perform an update of the default password in the first 24 hours after the registration, he is deleted from the system.

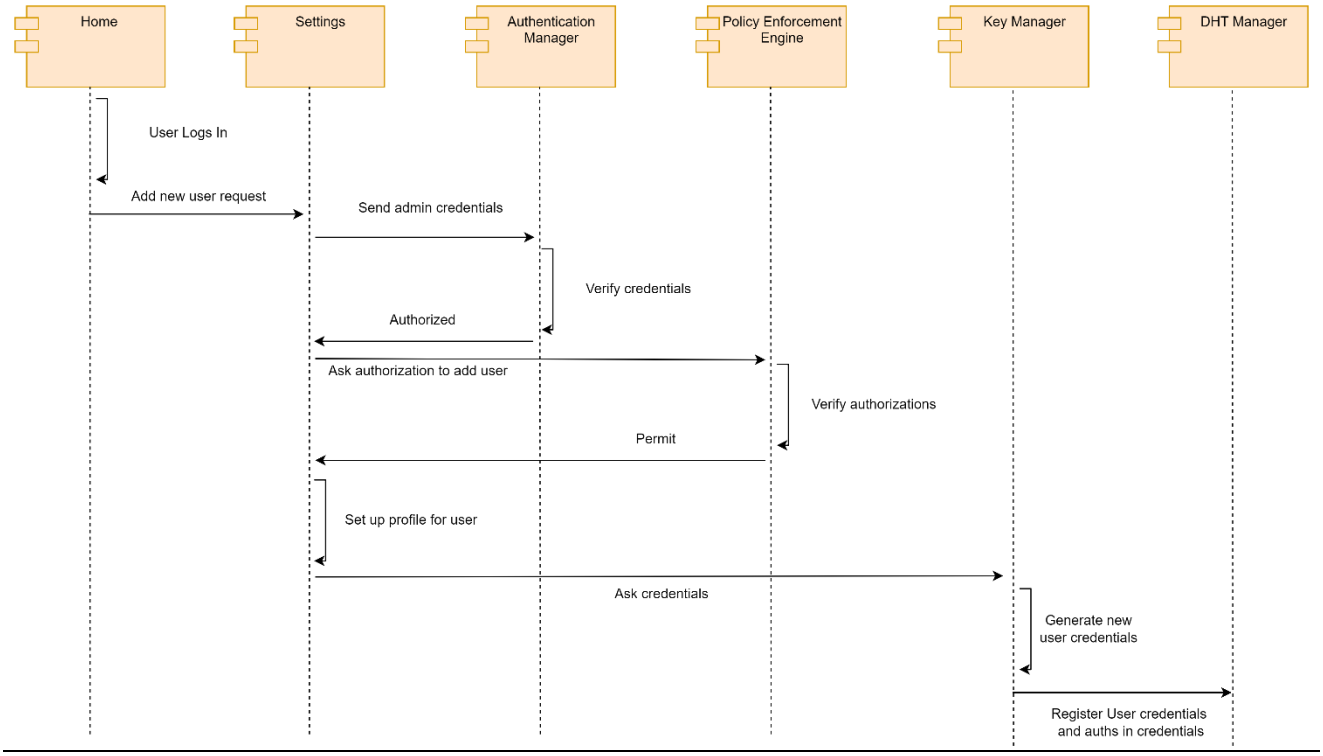

Figure 39: Workflow to register New User, Set Role and assign its Settings

6.7 *Anomaly detection analytic workflow [UC04, UC11, UC06]*

This workflow, which will be detailed in D4.2, shows the sequence of events in the execution of the anomaly detection analytic. As shown in Figure 40, the first part of the workflow consists of collecting input data from the various monitors and input interfaces in the GUI and Proactive Security Management Layer. The collection is done through either a polling mechanism, where the information is periodically stored in the Distributed Storage through the DHT communication manager, or through providing a handle representing a socket stored in the Distributed Storage. The latter data collection mechanism is suitable for e.g., network packet data, which must be subjected to an analytic as close as possible to real-time in order to minimize response time when reacting to e.g., network intrusions or threats. The Application Manager in the Secure Lifecycle Manager will handle the classification procedure through a dedicated service that will invoke the Policy Enforcement Engine, as shown in Figure 41. Data are anonymized before the analysis through the anonymization toolbox and then processed through a specific analytic. The output of the analytic is then processed by the Evaluator/Notifier which might either notify the user or trigger corrective actions through the Application Manager or the Node Manager (not shown in Picture). This workflow implements the use

cases UC04, UC11 and UC06.

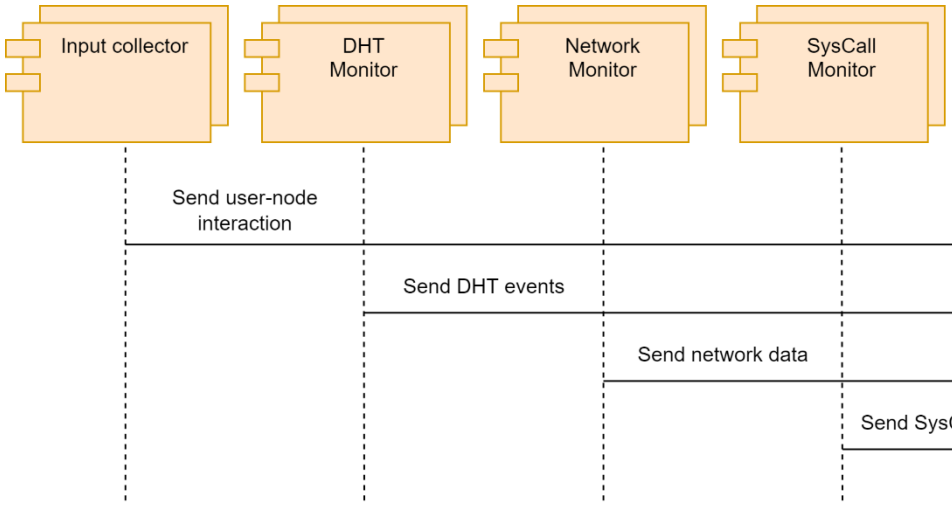

Figure 40: First part of the anomaly detection workflow: Illustration of sources providing input for various anomaly detection analytics.

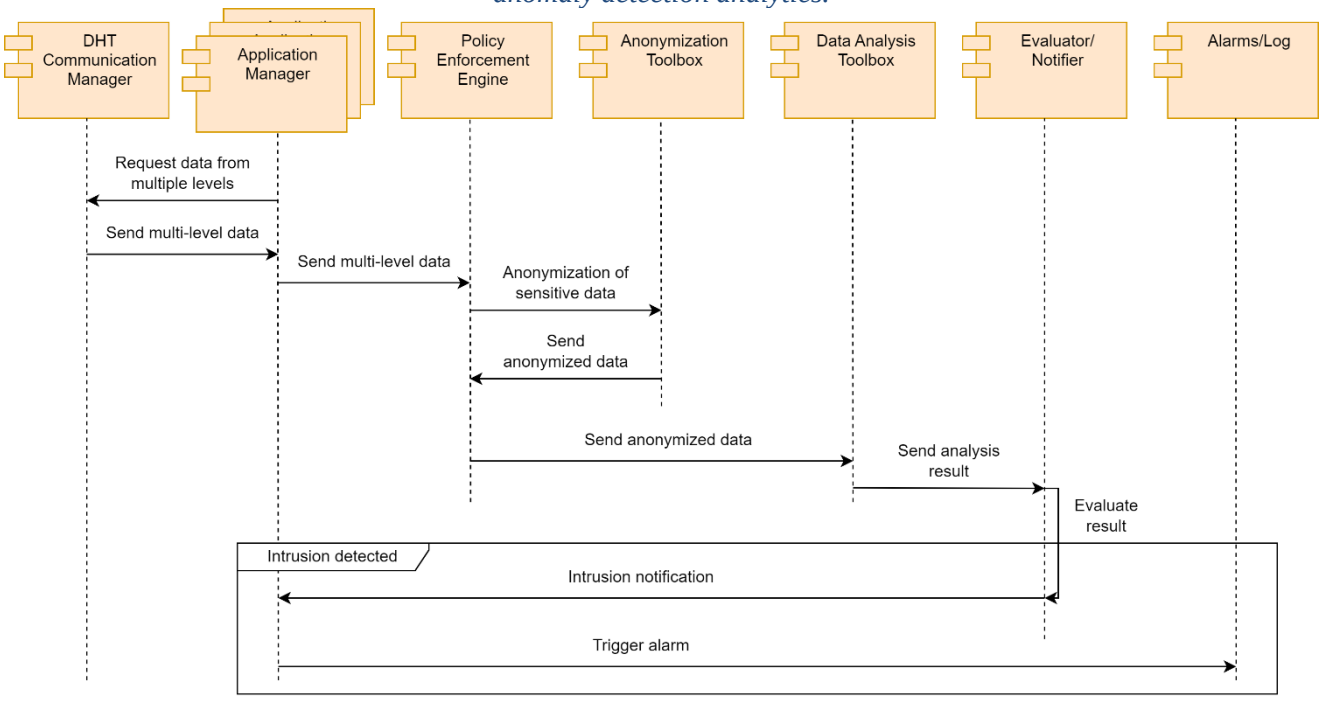

Figure 41: Second part of the anomaly detection workflow: Runtime communication between components

6.8 *Policy Translation Workflow [UC09]*

The workflow depicted in Figure 42 represents the operation performed by the Policy Translation Point (PTP) subcomponent of the Policy Enforcement Engine. The goal of this workflow is to translate high-level policies like "*WHEN evening AND in living room THEN deny audio registration*" into a set of low-level policies, preferably in a XACML-like formalism, that will be then used by the SIFIS-Home system to control the behaviour of the devices and applications installed in the smart home. In parallel, another goal of the task is to detect potential conflicts between high-level policies.

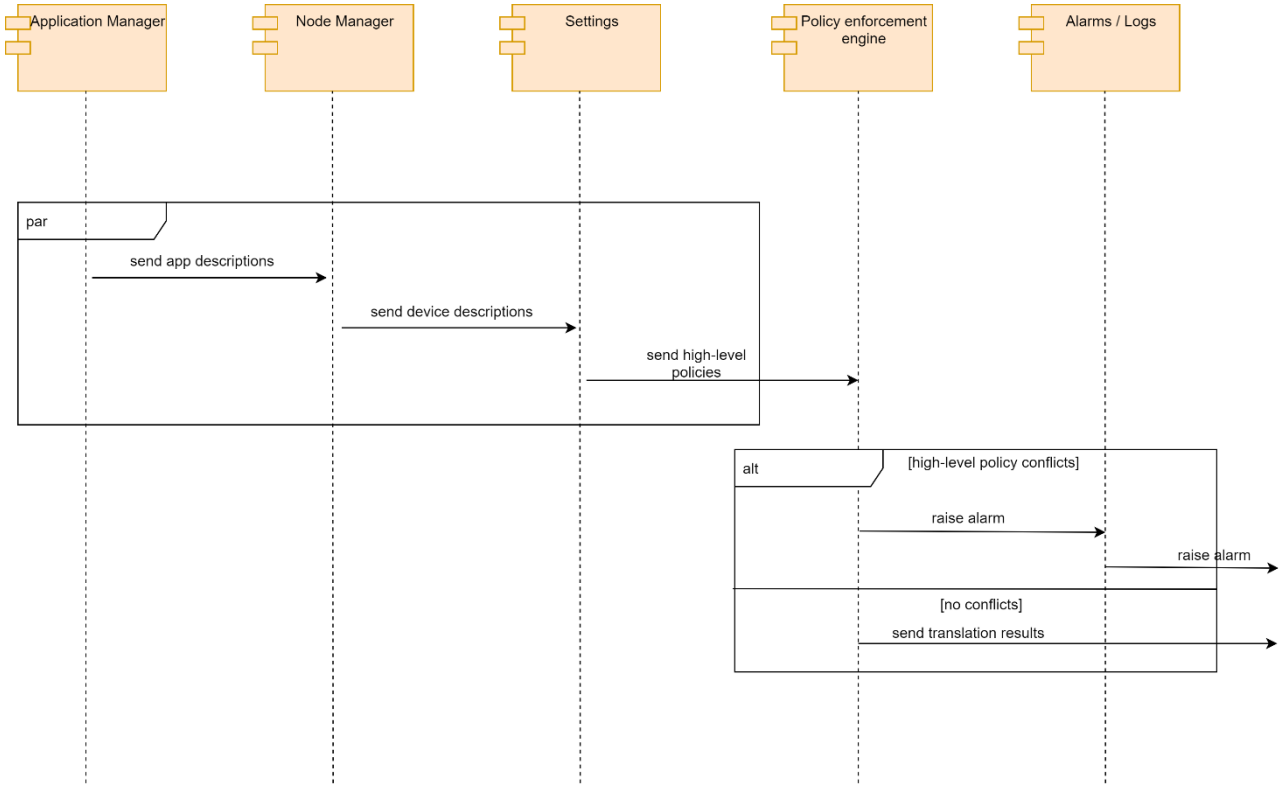

Figure 42: Policy Translation Workflow

These processes will happen through a reasoning process that will analyse contextual information as well as the capabilities of the devices and applications installed in the home. To this end, we will design a custom ontology, and we will exploit the tools and software provided by the Semantic Web framework.

In the SIFIS-Home architecture, the ontology and the developed software are included in the **Policy Enforcement Engine** module, within a specific sub-module named **Policy Translation Point (PTP)**. To implement the translation of high-level conflicts and the conflicts detection, we continuously get the following inputs from other modules:

- From the **Application Manager** module, we get information about the applications installed in the home, e.g., their capabilities, their current statuses, etc.
- From the **Node Manager** module, we get information about the devices installed in the home, e.g., their capabilities, their current statuses, etc. In parallel, we also get information about the home itself, e.g., the rooms, the position of the various devices, etc.
- From the **Settings** module, we retrieve the list of high-level policies to be checked and translated.

From the collected input, the PTP module firstly checks if there are conflicts between the available high-level policies. If this is the case, the **Alarms/Log** module used to send an alarm to the user, e.g., in the form of a notifications. If there are no conflicts, instead, the policies are translated in the corresponding XACML policies. Such policies are then sent to the same **Policy Enforcement Engine** module, which is in charge of enforcing them on the right devices and applications. This workflow satisfies the use case UC09.

6.9 *Provide and handle a voice command [UC01, UC02]*

This workflow, illustrated in Figure 43, shows the sequence of events required to issue a voice command to the DHT through the mobile app, leveraging the Data Analysis Toolbox to identify the

user and translate the voice to a Command and have the Policy Enforcement Engine evaluate it before allowing (or denying it). This workflow implement the use cases UC01 and UC02.

- A voice command is recorded through the Home component of the mobile app
- The command is recorded in the DHT manager and a post to the voice analysis topic is performed.
- The voice command is sent to the data analysis toolbox where the command is processed.
- The identity of the user is recognized.
- The analysis toolbox translates the command in a DHT command providing a topic and payload with parameters.
- The command is posted on the DHT with the topic for Policy Enforcement Engine.
- The policy enforcement engine verifies if the user can give such a command.
- If Permit the DHT manager publishes the payload on Topic, otherwise returns error to Home.
- The command is managed by the intended component.

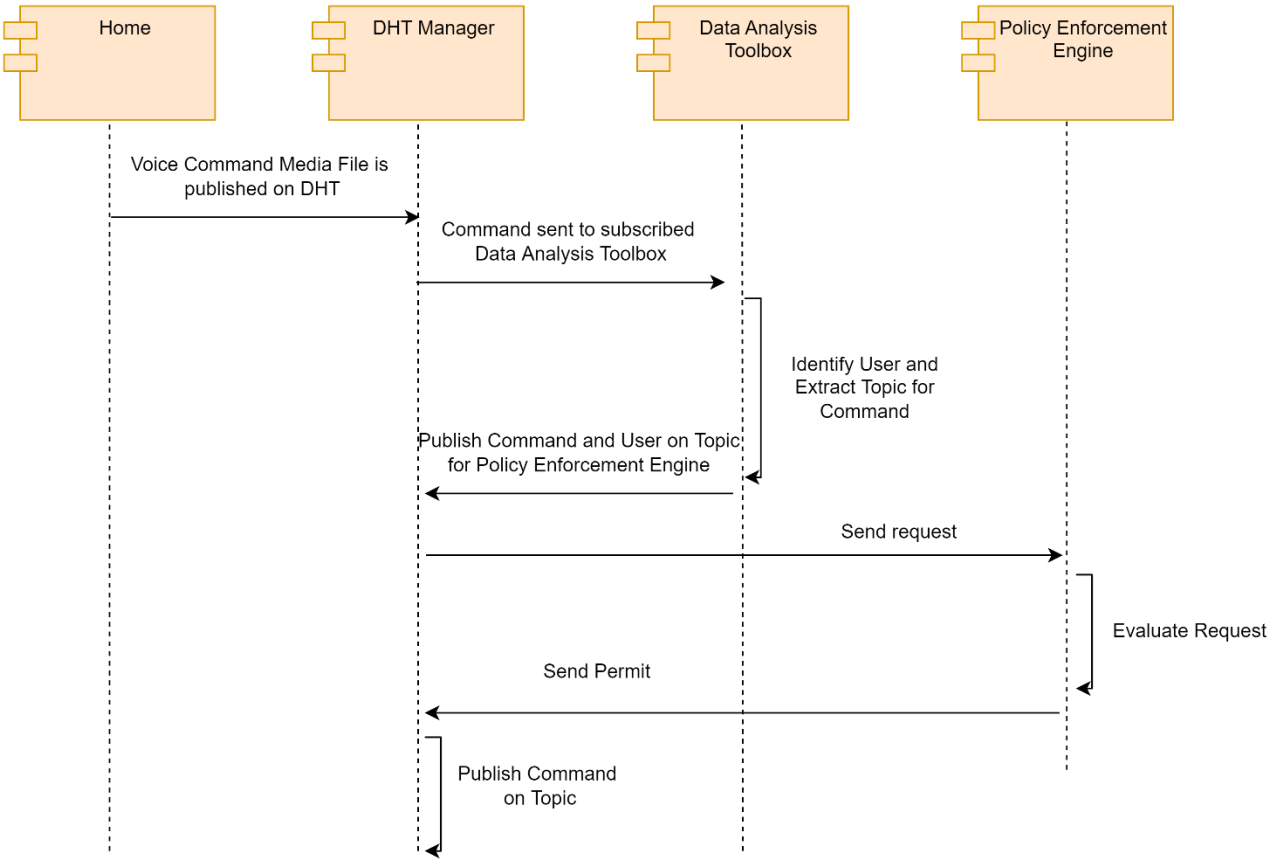

Figure 43: Workflow to provide and handle a voice command

6.10 *Access house functionality from remote device [UC12,UC13]*

As illustrated in Figure 44, this workflow is to access house functionality from a remote device. To access the house when being outside one must interface the cloud interface. The cloud interface will then let the user access the SIFIS Home network inside the house. Depending on role of the user different authorization rights to view and do things will be available:

- Continuously:
    - The DHT manager is subscribing to events from the FIWARE API related to the house
    - The DHT manager updates status in FIWARE as soon as any status in the house change, that is, the FIWARE API always holds the latest status from device, user, policy and general status point of view.
- Access the house and get the status
    - The user connects to Yggio via the mobile application. Depending on login credentials different access rights to the house will be granted.
    - Yggio request the FIWARE Context broker to provide latest status of the house for the logged in user. Yggio then provides this status the user.
- Perform an update action (device, policy or user):
    - The user request Yggio to update a resource (for example turn on/off a lamp) in the house, Yggio forward the request to the FIWARE Context broker and also confirm the request to the user.
    - The FIWARE Context broker execute the update request, if the user credentials are authorized the update will be done and if not, it will get rejected.
    - Since the DHT manager subscribed to FIWARE events if the update is approved it will get notified an authorized user wants to perform an action, the request is forwarded to the device manager that executes the request.
    - The user will then get confirmation that the request is done or if it was rejected.

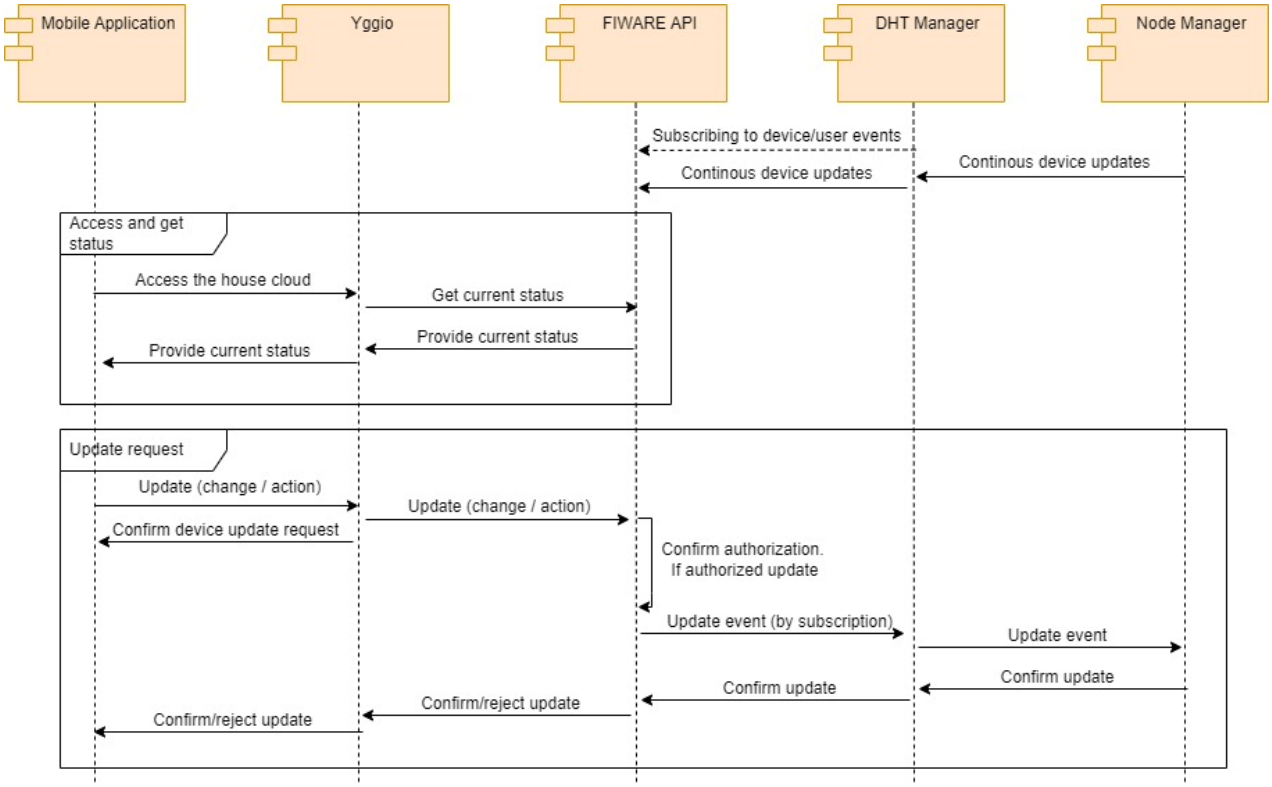

Figure 44: Workflow to access house functionality from a remote device

7 Security Analysis and Threat Models

The SIFIS-Home architecture has been designed to be dependable and secure, to be protected against both cyber attacks and devices misbehaviours. In the following we report a brief analysis of what are the security properties implemented in the SIFIS-Home architecture and the SIFIS-Home framework and we will discuss a set of threat models which are relevant for this environment and that will be tested against the SIFIS-Home security capabilities.

7.1 *Availability*

The capability of the system to be available, to perform its tasks and to satisfy user requests is extremely relevant in the smart home environment. Even partial unavailability can be catastrophic in a smart home environment. The functionalities of the SIFIS-Home framework are replicated among the smart devices. In particular, each smart device installs an instance of the SIFIS-Home SD framework. Thus, the availability of SIFIS-Home is ensured until at least one smart device is active. If there are no active smart devices, the SIFIS-Home architecture fails.
We assume that a SIFIS-Home instance includes a number of smart devices greater or equal than 3. If one device fails, the self-healing module will redistribute functionalities and resources assignment (especially NSSDs), and it will try to restore the fault device.

The failure of one device or of the connection to the SIFIS-Home Cloud, in SIFIS-Home is designed to generate a potential functionality degradation, but not an overall system failure. In fact, the unavailability of a smart device or of the SIFIS-Home cloud (which might be caused by the lack of Internet connection) will cause AT MOST the unavailability of the resources connected to that smart device, or of the functionalities accessed through the cloud. This is also ensured by the presence of the DHT, which replicates at a certain degree the information available on each smart device and can even be fully replicated.

7.1.1 Attacks to Availability

Physical removal of a smart device: we consider the attacker able to physically disconnect/remove from the house a number of smart devices between 1 and *m* where *m* is lesser than the total number of devices.
Denial of Service: the attacker is able to send a large number of packets toward a single device (smart device or NSSD) blocking all the useful network traffic toward that device and making it unreachable.
Physical removal of a NSSD: the attacker removes or turns off a NSSD making the resources controlled by it unavailable.
Wi-Fi shut down / Jamming: the attacker disconnects the Wi-Fi router or jams a specific Wireless channel to block the connection among smart devices.
Computation Overload: The attacker installs an application on the smart device which consumes all the CPU time, making it impossible to perform normal operations.

7.2 *Confidentiality*

Being able to access private information allows an attacker to violate the privacy of the smart home tenants, or can give access to credentials which can be used to escalate the attack and take control of the system. SIFIS-Home exploits security protocols and state of the art cryptography mechanisms to ensure that all transported data remain confidential. In particular, all communications happening among smart devices, and those among smart devices and NSSDs, they all rely on TLS or DTLS, ensuring that entities which are not part of the SIFIS-Home architecture, are not able to read the exchanged packets. Access to the SIFIS-Home architecture is protected through credentials and workflows which ensure that the access of a new device can only be performed when triggered by an administrator/maintainer. For those devices which are part of the SIFIS-Home architecture, still might be compromised, a first level of protection is ensured by differentiated encrypted storage. In particular, the DHT stores data always encrypted, where the encryption is performed on a topic base, meaning that a device which should not have access to a specific topic, cannot access, delete or modify those information. A rekeying workflow is triggered immediately once a device is found to have been compromised, to further reduce the malicious effects on confidentiality.

7.2.1 Attacks to confidentiality

Eavesdropping: The attacker connects the Wi-Fi network in promiscuous mode and intercepts packets attempting to read their content.
Spyware: The attacker installs a third party application which attempts at extracting private information from the DHT and accessing NSSDs resources. We assume that the application can query the DHT but does not have the authorization to read the target information and does not have the right to access the target resource.

7.3 *Integrity*

This is a key property of cybersecurity, which ensures that information are not maliciously modified either during transport or at rest. By exploiting TLS, the integrity at transport is ensured by design. Integrity at rest is ensured by data encryption and by means of access control mechanisms.

7.3.1 Attacks to Integrity

Man in the Middle (MITM): the attacker tries to intercept and modify packets while in transit between two legit entities. We assume that the attacker is able to listen on the wireless channel between two smart devices and between a smart device and an NSSD.
DHT spoofing: the attacker tries to modify the content of stored information in the DHT. We assume that the attacker is able to access the DHT and is allowed to modify the target information. We assume

that only one smart device is under the control of the attacker by means of a specific third-party application.

7.4 *Distributed System Security*

The lack of a root of trust is the weak point of distributed systems. While the distributiveness is an asset to counter denial of services, distributed systems have to rely on different techniques to understand when a device is misbehaving or it has been compromised. SIFIS-Home uses a trust-based model to identify and isolate misbehaving devices. In particular, many operations in SIFIS-Home require a general consensus among the smart devices. Such decisions could be access control decisions, reading of data from correlated sensors and anomaly detection decisions. These decisions are take through a collaborative vote. Each time such a decision has to be taken, the smart devices exploit the DHT to vote on the decision. Only the devices that are actually entitled to vote can participate to each voting procedure (e.g. the smart devices having access on correlated resources). The vote of each smart device is weighted according to a reputation score, which is increased every time the nodes vote in agreement with the common decision and is decreased (significantly) every time the node is in disagreement with the common consensus. A reputation threshold is established and nodes under the threshold cannot participate to a vote (but they can still express their decision to update the reputation score). This algorithm is used to help in the detection of compromised nodes, which intentionally (malicious) or unintentionally (malfunctioning) vote for wrong decisions. The mechanism is resistant to coalition attacks.

7.4.1 Attacks to distributed systems

Device Compromission: The attacker maliciously gains the control of a smart device or NSSD, by installing malicious third party app or software on it. We assume that for smart devices the attacker is able to install a malicious application and for the NSSDs is able to modify the firmware.
Sybil Attack: The attacker attempts to create fake devices with fake identities which become part of the DHT. This attack is escalated to generate network partitions and perform MITM attacks.

7.5 *Authorization and Access Control*

The management of access control is a main security aspect addressed by the SIFIS-Home framework. In SIFIS-Home each user has a role, to which specific authorizations are given for specific operations. At the same time, an authorization based system is also used to control the access to resources and operations, by both devices and applications. This granular control is enabled by the Policy Enforcement Engine, which is a tool based on dynamic attribute-based access control (ABAC). Through secure APIs and the presence of the DHT intercepting all operations and requests, it is possible to control in the SIFIS-Home framework virtually the authorizations for any operation, user, device and application. Still, the challenge resides in the correct definition of policies, which become more complicated as the complexity of a smart home environment increase, leaving the system potentially prone to specific attacks.

7.5.1 Attacks to Authorization and Access Control

Honest but curious: This is the typical threat model of any access control system. We consider any entity (user, application, device) requesting authorizations as an honest component, which means it totally abides to the protocols and given authorization. Still, if the entity has the possibility to access a resource which should not be accessed, the entity will exploit this possibility and access the resource. This attack becomes possible when there are mistakes in the access policies.
Privilege Escalation: Through this attack, an entity tries to progressively and maliciously take higher privileges, which should not be granted to it. This attack is made possible again through policy mistakes and lack of control on the minimum privilege paradigm.

8 Conclusion

In this deliverable we have presented the preliminary design of the SIFIS-Home Architecture and the SIFIS-Home framework. The SIFIS-Home architecture helps in providing a formal definition of SIFIS-Home actors, users and devices, which will be used to define security and safety policies, based on their rights to perform specific actions in specific contexts. The SIFIS-Home framework defines instead all the components which have been used to map the needed functionalities addressing the requirements specified in D1.1 and D1.2. We recall that the architecture of the SIFIS-Home framework presented in this document is preliminary, and though it has followed a monitored bottom-up approach, we are aware that some modifications might occur following the feedbacks from the implementation and deployment phase.

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/

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

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

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

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

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

[XACML, 2013] eXtensible Access Control Markup Language (XACML) Version 3.0, Oasis Standard, http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-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

Glossary

Acronym	Definition
DHT	Distributed Hash Table
FR	Functional Requirements
NFR	Non-functional requirement
OS	Operative System
P2P	Peer to Peer
SIFIS-Home	Secure Interoperable Full Stack Internet of Things for Smart Home
UC	Use case
US	User story
SD	Smart Device
NSSD	Not So Smart Device
XACML	eXtensible Access Control Markup Language
PEP	Policy Enforcement Point
PIP	Policy Information Point
PDP	Policy Decision Point
PAP	Policy Administration Point
PTP	Policy Translation Point
CH	Context Handler
AODV	Ad-hoc On-demand Distance Vector

Appendix A: JSON documentation of the APIs


```
{
 "swagger": "2.0",
 "info": {
  "description": "SIFIS-Home example APIs",
  "version": "1.0.0",
  "title": "SIFIS-Home example APIs"
 },
 "host": "web.sifis-home.eu",
 "basePath": "/v1",
 "tags": [
  {
   "name": "devices",
   "description": "APIs to get information and details about the managed devices"
  },
  {
   "name": "home",
   "description": "APIs to get credentials"
  },
  {
   "name": "message",
   "description": "APIs to get messages, logs and alarms from a SIFIS-Home system"
  },
  {
   "name": "application",
   "description": "APIs to manage the installation/removal of third-parties applications"
  }
 ],
 "schemes": [
  "https",
  "http"
 ],
 "paths": {
  "/devices": {
   "get": {
    "description": "The API returns information about all the devices registered in the SIFIS-Home system",
    "parameters": [
     {
      "in": "header",
      "name": "Authorization",
      "description": "Authorization Token of the client requiring information about the devices",
      "type": "string",
      "required": true
     }
    ],
    "tags": [
     "devices"
```

```
      ],
      "responses": {
       "200": {
        "description": "OK",
        "schema": {
         "type": "array",
         "items": {
          "$ref": "#/definitions/Device"
         }
        }
       },
       "401": {
        "description": "User unauthorized"
       },
       "404": {
        "description": "Devices not found"
       }
      }
     }
    },
    "/devices/addFavouriteDevice/{deviceID}": {
     "post": {
      "description": "The API receives the deviceID of a device that has to be put into the set of favourite devices",
      "tags": [
       "devices"
      ],
      "responses": {
       "200": {
        "description": "OK"
       },
       "401": {
        "description": "User not authorized"
       },
       "404": {
        "description": "Device with deviceID not found"
       }
      },
      "parameters": [
       {
        "in": "path",
        "name": "deviceID",
        "description": "ID of the device to be added to the favourite devices set",
        "type": "string",
        "required": true
       },
       {
        "in": "header",
        "name": "Authorization",
        "description": "Authorization Token of the client requiring information about the devices",
```

```
          "type": "string",
          "required": true
        }
       ]
     }
   },
   "/devices/removeFavouriteDevice/{deviceID}": {
    "delete": {
      "description": "The API receives the deviceID of a device that has to be deleted from the set of favourite devices",
      "tags": [
       "devices"
      ],
      "responses": {
       "200": {
        "description": "OK"
       },
       "401": {
        "description": "User not authorized"
       },
       "404": {
        "description": "Device not found"
       }
      },
      "parameters": [
       {
        "in": "path",
        "name": "deviceID",
        "description": "ID of the device to be removed from the favourite devices set",
        "type": "string",
        "required": true
       },
       {
        "in": "header",
        "name": "Authorization",
        "description": "Authorization Token of the client requiring information about the devices",
        "type": "string",
        "required": true
       }
      ]
     }
   },
   "/devices/favouriteDevices": {
    "get": {
      "description": "The API returns information about all the devices that are in the favourite devices set",
      "parameters": [
       {
        "in": "header",
        "name": "Authorization",
```

```
        "description": "Authorization Token of the client requiring information about the favourite devices",
        "type": "string",
        "required": true
       }
      ],
      "tags": [
       "devices"
      ],
      "responses": {
       "200": {
        "description": "OK",
        "schema": {
         "type": "array",
         "items": {
          "$ref": "#/definitions/Device"
         }
        }
       },
       "401": {
        "description": "User unauthorized"
       },
       "404": {
        "description": "Devices not found"
       }
      }
     }
    },
    "/home/login": {
     "post": {
      "description": "API to get an authorization token using username and password",
      "tags": [
       "home"
      ],
      "responses": {
       "200": {
        "description": "OK",
        "schema": {
         "type": "string",
         "description": "Authorization Token"
        }
       },
       "401": {
        "description": "Login failure"
       }
      },
      "parameters": [
       {
        "in": "header",
        "name": "username",
```

```
        "description": "username for the login procedure",
        "type": "string",
        "required": true
       },
       {
        "in": "header",
        "name": "password",
        "description": "password for the login procedure",
        "type": "string",
        "required": true
       }
      ]
     }
    },
    "/messages": {
     "get": {
      "tags": [
       "message"
      ],
      "summary": "Access data from alarms and activity log",
      "description": "The API returns a list of messages. It returns the messages belonging to the topics to which the client is currently registered. The API allows an optional additional filter to be provided as a parameter.",
      "produces": [
       "application/json"
      ],
      "parameters": [
       {
        "in": "header",
        "name": "Authorization",
        "description": "Authorization Token of the client requiring the data feed",
        "type": "string",
        "required": true
       },
       {
        "in": "query",
        "name": "search_string",
        "description": "Optional filter string for the messages to be returned",
        "type": "string",
        "required": false
       }
      ],
      "responses": {
       "200": {
        "description": "OK",
        "schema": {
         "type": "array",
         "items": {
          "$ref": "#/definitions/Message"
         }
```

```
        }
       },
       "401": {
        "description": "Client unauthorized"
       },
       "404": {
        "description": "No messages found"
       }
      }
     }
    },
    "/messages/messageFeedRegister/{topic}": {
     "post": {
      "tags": [
       "message"
      ],
      "summary": "The API allows to subscribe to data from alarms and activity logs pertaining to a certain topic ",
      "description": "The API receives the topic to which the user wants to subscribe.",
      "produces": [
       "application/json"
      ],
      "parameters": [
       {
        "in": "header",
        "name": "Authorization",
        "type": "string",
        "description": "Authorization Token of the client willing to perform the registration",
        "required": true
       },
       {
        "in": "path",
        "name": "topic",
        "description": "Topic of the required feed",
        "type": "string",
        "required": true
       }
      ],
      "responses": {
       "200": {
        "description": "OK"
       },
       "401": {
        "description": "Client unauthorized"
       },
       "404": {
        "description": "Topic not registered"
       }
      }
     }
```

```
  },
  "/messages/messageFeedUnregister/{topic}": {
   "post": {
    "tags": [
     "message"
    ],
    "summary": "The API allows to disable the subscription to data from alarms and activity logs pertaining to a certain topic",
    "description": "The API receives the topic to be unregistered from.",
    "produces": [
     "application/json"
    ],
    "parameters": [
     {
      "in": "header",
      "name": "Authorization",
      "type": "string",
      "description": "Authorization Token of the client willing to unregister",
      "required": true
     },
     {
      "in": "path",
      "name": "topic",
      "description": "Topic of the feed",
      "type": "string",
      "required": true
     }
    ],
    "responses": {
     "200": {
      "description": "OK"
     },
     "401": {
      "description": "Client unauthorized"
     },
     "404": {
      "description": "Topic not registered"
     }
    }
   }
  },
  "/messages/streamCameraFeeds/{topic}": {
   "get": {
    "tags": [
     "message"
    ],
    "summary": "Access data from alarms and activity log",
    "description": "The API receives as a parameter the topic of the camera feeds to read.",
    "produces": [
     "application/json"
```

```
      ],
      "parameters": [
       {
        "in": "header",
        "name": "Authorization",
        "type": "string",
        "description": "Authorization Token of the client willing to stream the camera feed"
       },
       {
        "in": "path",
        "name": "topic",
        "type": "string",
        "description": "Topic of the desired camera feeds",
        "required": true
       }
      ],
      "responses": {
       "200": {
        "description": "OK",
        "schema": {
         "type": "array",
         "items": {
          "$ref": "#/definitions/CameraFeed"
         }
        }
       },
       "401": {
        "description": "Client unauthorized"
       },
       "404": {
        "description": "No messages found"
       }
      }
     }
    },
    "/applications/installApplication/{applicationID}": {
     "post": {
      "tags": [
       "application"
      ],
      "summary": "The API is used to add a new third-party application to the SIFIS-Home system. It receives as parameters: i) Activate (indicates if the application should be started after its installation), ii) the application ID (identifier of the app to be added), iii) the application name and iv) application specific settings.",
      "parameters": [
       {
        "in": "path",
        "name": "applicationID",
        "type": "string",
        "description": "ID of the application to install",
```

```
          "required": true
        },
        {
          "in": "header",
          "name": "Authorization",
          "type": "string",
          "description": "Authorization Token of the client willing to install the application"
        },
        {
          "in": "header",
          "name": "Activate",
          "type": "boolean",
          "description": "Defines if the activation of the application has to be performed after its installation"
        },
        {
          "in": "header",
          "name": "Settings",
          "type": "string",
          "description": "Application-specific settings"
        }
      ],
      "responses": {
        "200": {
          "description": "OK",
          "schema": {
            "$ref": "#/definitions/Application"
          }
        },
        "401": {
          "description": "Client Unauthorized"
        },
        "404": {
          "description": "Application not found"
        }
      }
    }
  },
  "/applications/removeApplication/{applicationID}": {
    "delete": {
      "tags": [
        "application"
      ],
      "summary": "The API is used to remove a third-party application. It receives as parameter the identifier of the app to be deleted. It returns an HTTP response with the operation result.",
      "parameters": [
        {
          "in": "path",
          "name": "applicationID",
          "type": "string",
```

```
        "description": "ID of the application to remove",
        "required": true
       },
       {
        "in": "header",
        "name": "Authorization",
        "type": "string",
        "description": "Authorization Token of the client willing to remove the application"
       }
      ],
      "responses": {
       "200": {
        "description": "OK"
       },
       "401": {
        "description": "Client Unauthorized"
       },
       "404": {
        "description": "Application not found"
       }
      }
     }
    },
    "/applications/killApplication/{applicationID}": {
     "post": {
      "tags": [
       "application"
      ],
      "summary": "The API is used to kill an active third-party application. It receives as parameter the
id (identifier of the app to be killed). It returns an HTTP response with the operation result.",
      "parameters": [
       {
        "in": "path",
        "name": "applicationID",
        "type": "string",
        "description": "ID of the application to kill",
        "required": true
       },
       {
        "in": "header",
        "name": "Authorization",
        "type": "string",
        "description": "Authorization Token of the client willing to kill the application"
       }
      ],
      "responses": {
       "200": {
        "description": "OK"
       },
       "401": {
```

```
        "description": "Client Unauthorized"
       },
       "404": {
        "description": "Application not found"
       }
      }
     }
    },
    "/applications/wipeApplication/{applicationID}": {
     "post": {
      "tags": [
       "application"
      ],
      "summary": "The API is used to wipe all the data of an application which is already installed in the solution. It receives as parameter the id (identifier for the app whose data has to be wiped). It returns an HTTP response with the operation result.",
      "parameters": [
       {
        "in": "path",
        "name": "applicationID",
        "type": "string",
        "description": "ID of the application whose data has to be wiped",
        "required": true
       },
       {
        "in": "header",
        "name": "Authorization",
        "type": "string",
        "description": "Authorization Token of the client willing to wipe the application data"
       }
      ],
      "responses": {
       "200": {
        "description": "OK"
       },
       "401": {
        "description": "Client Unauthorized"
       },
       "404": {
        "description": "Application not found"
       }
      }
     }
    }
   },
   "definitions": {
    "Message": {
     "type": "object",
     "properties": {
      "id": {
```

```
      "type": "string",
      "description": "identifier of the message"
     },
     "type": {
      "type": "string",
      "description": "type of message",
      "enum": [
       "alarm",
       "log"
      ]
     },
     "topic": {
      "type": "string"
     },
     "description": {
      "type": "string",
      "description": "Message content"
     }
    }
   },
   "CameraFeed": {
    "type": "object",
    "properties": {
     "id": {
      "type": "integer",
      "description": "identifier of the camera feed",
      "format": "int64"
     },
     "topic": {
      "type": "string",
      "description": "topic of the feed"
     },
     "stream": {
      "type": "string",
      "format": "byte",
      "description": "image feed"
     }
    }
   },
   "Application": {
    "type": "object",
    "properties": {
     "applicationID": {
      "type": "string",
      "description": "identifier of the application"
     },
     "name": {
      "type": "string",
      "description": "Application-specific name"
     },
```

```
      "settings": {
        "type": "string",
        "description": "Application-specific settings"
      }
    }
  },
  "Device": {
    "type": "object",
    "properties": {
      "deviceID": {
        "type": "string",
        "description": "identifier of the device"
      },
      "name": {
        "type": "string",
        "description": "Name of the device"
      }
    }
  }
 },
 "externalDocs": {
  "description": "Find out more about Swagger",
  "url": "http://swagger.io"
 }
}
```

Appendix B – differences with D1.3

The present appendix discusses and motivates the modifications performed on the conceptual architecture of the SIFIS-Home framework between the preliminary version (presented in deliverable 1.3) and the final one (presented in this deliverable).

The main modification performed on the conceptual architecture has been the logical separation of macro-components according to the specific hardware device where their deployment has to be performed. This has led to the separation of the single top-level architectural block "SIFIS-Home Framework" into four different top-level architectural blocks: SIFIS-Home Application Framework, containing all the software components to be deployed on a mobile device to control the SIFIS-Home Framework; SIFIS-Home Cloud Framework, containing all the software components to be deployed on a Cloud infrastructure to support the other components of the architecture; SIFIS-Home NSSD Framework, containing all the software components to be installed on Not-so-smart Devices; SIFIS-Home Smart Device Framework, containing all the software components to be installed on the Smart Devices managing the SIFIS-Home infrastructure.

Minor modifications have been applied inside the macro-components, principally to simplify the architecture: an example of such practice is the more compact design of the SIFIS-Home API Gateway component, where a *Mobile Application API* sub-component has been defined to incorporate nine different components of the preliminary architecture. Other architectural changes have been performed to increase the cohesion of the software modules, e.g., the movement of the WoT manager in the NSSD Manager component and out of the SIFIS-Home API Gateway due to low cohesion with the latter software modules.

All the modifications performed in the architecture in the transition between deliverable D1.3 and deliverable D1.4 are described in table below. The table provides the corresponding component in the new architecture for each component of the architecture in deliverable D1.3 and describes the rationale for the changes (if any).

D1.3 macro-component	D1.3 component	D1.4 macro-component	D1.4 component	Notes
SIFIS-Home API Gateway	Device Management API	SIFIS-Home API Gateway	Mobile Application API	All API modules collected in a single module to simplify the architecture
SIFIS-Home API Gateway	Data Management API	SIFIS-Home API Gateway	Mobile Application API	All API modules collected in a single module to simplify the architecture
SIFIS-Home API Gateway	DHT Management API	SIFIS-Home API Gateway	Mobile Application API	All API modules collected in a single module to simplify the architecture
SIFIS-Home API Gateway	Application API	SIFIS-Home API Gateway	Mobile Application API	All API modules collected in a single module to simplify the

				architecture
SIFIS-Home API Gateway	Policy Management API	SIFIS-Home API Gateway	Mobile Application API	All API modules collected in a single module to simplify the architecture
SIFIS-Home API Gateway	Data Analysis API	SIFIS-Home API Gateway	Mobile Application API	All API modules collected in a single module to simplify the architecture
SIFIS-Home API Gateway	Notification API	SIFIS-Home API Gateway	Mobile Application API	All API modules collected in a single module to simplify the architecture
SIFIS-Home API Gateway	Communication API	SIFIS-Home API Gateway	Mobile Application API	All API modules collected in a single module to simplify the architecture
SIFIS-Home API Gateway	Marketplace API	SIFIS-Home API Gateway	Mobile Application API	All API modules collected in a single module to simplify the architecture
SIFIS-Home API Gateway	Home API	SIFIS-Home API Gateway	Mobile Application API	All API modules collected in a single module to simplify the architecture
SIFIS-Home API Gateway	WoT Interfacing API	NSSD Manager	WoT Manager	Component moved in the newly-defined NSSD Manager macro-component
SIFIS-Home API Gateway	Fiware Interfacing API	DHT Manager	Fiware API	Component moved into the new DHT Manager macro-component
User Interface	Home	SIFIS-Home Application Framework	Home	Macro-component renamed to SIFIS-Home Application

				Framework
User Interface	Device Management	SIFIS-Home Application Framework	Device Management	Macro-component renamed to SIFIS-Home Application Framework
User Interface	Settings	SIFIS-Home Application Framework	Settings	Macro-component renamed to SIFIS-Home Application Framework
User Interface	Alarms / Logs	SIFIS-Home Application Framework	Alarms / Logs	Macro-component renamed to SIFIS-Home Application Framework
User Interface	Marketplace	SIFIS-Home Application Framework	Marketplace	Macro-component renamed to SIFIS-Home Application Framework
User Interface	Input Collection	SIFIS-Home Application Framework	Input Collection	Macro-component renamed to SIFIS-Home Application Framework
User Interface	Policy Manager	SIFIS-Home Application Framework	Settings	Macro-component renamed to SIFIS-Home Application Framework; component incorporated into the Settings component
Secure Lifecycle Manager	Application Manager	Secure Lifecycle Manager	Application Manager	-
Secure Lifecycle Manager	Node Manager	Secure Lifecycle Manager	Node Manager	-
Secure Lifecycle Manager	Authentication Manager	Secure Lifecycle Manager	Authentication Manager	-

Secure Lifecycle Manager	Device Registration Manager	Secure Lifecycle Manager	Device Registration Manager	-
Secure Lifecycle Manager	Key Manager	Secure Lifecycle Manager	Key Manager	-
Secure Lifecycle Manager	System Protection Manager	Secure Lifecycle Manager	System Protection Manager	-
Secure Communication Layer	DHT Communication Manager	DHT Manager	DHT	Component moved in the DHT Manager component, features managed by the DHT sub-component
Secure Communication Layer	Secure Message Exchange Manager	Secure Communication Layer	Secure Message Exchange Manager	-
Secure Communication Layer	Content Distribution Manager	Secure Communication Layer	Content Distribution Manager	-
Secure Communication Layer	External Communication Manager	DHT Manager	Fiware API	Component moved in the DHT Manager component, features managed by the Fiware API sub-component
Secure Communication Layer	Network Protection Manager	Secure Lifecycle Manager	System Protection Manager	Component incorporated into the System Protection Manager component
Application Toolboxes	Policy Enforcement Engine	Application Toolboxes	Policy Enforcement Engine	-
Application Toolboxes	Anonymization Toolbox	Application Toolboxes	Anonymization Toolbox	-
Application Toolboxes	Data Analysis Toolbox	Application Toolboxes	Data Analysis Toolbox	-
Proactive Security Management	API Monitor	Proactive Security Management	Monitors	Component incorporated into the

Layer		Layer		Monitors component
Proactive Security Management Layer	DHT Monitor	Proactive Security Management Layer	Monitors	Component incorporated into the Monitors component
Proactive Security Management Layer	Self Healing	Proactive Security Management Layer	Self Healing	-
Proactive Security Management Layer	Network / System Monitor	Proactive Security Management Layer	Monitors	Component incorporated into the Monitors component
Proactive Security Management Layer	Distributed Trust	Proactive Security Management Layer	Distributed Trust	-
Proactive Security Management Layer	Evaluator / Notifier	SIFIS-Home Application Framework	Alarms / Logs	Component incorporated into the Alarms/Logs component