Symbiosis of smart objects across IoT environments

688156 - symbIoTe - H2020-ICT-2015

Final symbIoTe Middleware Implementation

The symbIoTe Consortium

Intracom SA Telecom Solutions, ICOM, Greece
Sveučiliste u Zagrebu Fakultet elektrotehnike i računarstva, UNIZG-FER, Croatia
AIT Austrian Institute of Technology GmbH, AIT, Austria
Nextworks Srl, NXW, Italy
Consorzio Nazionale Interuniversitario per le Telecomunicazioni, CNIT, Italy
ATOS Spain SA, ATOS, Spain
University of Vienna, Faculty of Computer Science, UNIVIE, Austria
Unidata S.p.A., UNIDATA, Italy
Sensing & Control System S.L., S&C, Spain
Fraunhofer IOSB, IOSB, Germany
Ubiwhere, Lda, UW, Portugal
VIPnet, d.o.o, VIP, Croatia
Instytut Chemii Bioorganicznej Polskiej Akademii Nauk, PSNC, Poland
NA.VI.GO. SCARL, NAVIGO, Italy
Universität Zürich, UZH, Switzerland

For more information on this document or the symbIoTe project, please contact:
Sergios Soursos, INTRACOM TELECOM, souse@intracom-telecom.com

Document Control

Title: Final symbIoTe middleware Implementation

Type: Public

Editor(s): Matteo Di Fraia, Alessandro Carminati (Unidata)

E-mail: m.difraia@unidata.it, a.carminati@unidata.it

Author(s): Matteo Di Fraia, Alessandro Carminati, Fabrizio Giuliano, Matteo Pardi, JakubToczek, Mikołaj Dobski, Vasileios Glykantzis, Pavle Skočir

Doc ID: D4.3 - v0.6

Amendment History

Version	Date	Author	Description/Comments
v0.1	June 4, 2018	Matteo Di Fraia, Alessandro Carminati	Initial structure of the document
v0.2	July 5, 2018	Matteo Di Fraia, Alessandro Carminati, Fabrizio Giuliano, Matteo Pardi, Jakub Toczek, Mikołaj Dobski	Merge Partners contribution
v0.3	July 18, 2018	Matteo Di Fraia, Vasileios Glykantzis, Pavle Skočir	Merge Partners contribution, revision section, add conclusion
v0.4	July 26, 2018	Maria Bianco	Tables and Figures references, minor typos, review
v0.5	July 31, 2018	Alessandro Carminati	Final version prepared
v0.6	September 13, 2018	Matteo Di Fraia, Zvonimir Zelenika	Final version after updates on sequence diagrams

Table of Contents

Table of Figures

Table of Tables

Executive Summary

The aim of deliverable 4.3 is to document the final set of components related to the symbIoTe Smart Space (SSP) Middleware (S3M) and provide tools to enable implementation of symbIoTe compliant local IoT environments.

As a documentation of the definitive version of the middleware, this document is intended to be as technical and precise as possible: the components and their functionality will be presented in a rigorous way.

After a general description of the architecture, presenting the final correlation between the SSP middleware components and the rest of the symbIoTe ecosystem, every component is described in detail. A chapter is dedicated to the security aspects of interaction between local devices inside the Smart Space, called Smart Devices (SDEVs), the symbIoTe compliant IoT Platform and the SSP middleware.

Inputs coming from Deliverable D4.1 [1] are used as base for this document and software implementation, primarily for the architecture. Inputs are taken from other WPs like WP3 for the security related topics and WP2 for Core communication and interaction between the SSP middleware and the Core, semantic description of the resources inside the SSP and software module from Core/Cloud components. Goal is to reuse the as much as possible from the already developed software modules and readapt these solutions in a more lightweight way.

This S3M is primarily written in Java, apart from the SDEV Agent that is specific to SDEV hardware platform and thus is implemented in C++ for Arduino ESP8266 platform. Java implementation means that the SSP gateway running the middleware and forming the SSP environment could be any machine that is capable of running Java Virtual Machine. A detailed chapter is dedicated to the installation of the middleware and the environment set up.

The main outcome of this Deliverable D4.3 is the source code and its documentation, published as an open source project on GitHub service [2]. This document is formally an accompanying report documenting the software.

(This page is left blank intentionally.)

1 Introduction

Deliverable D4.3 provides a final description of the set of components and associated features implementing the symbIoTe Smart Space Middleware (S3M), the system acting as a gateway in the physical Smart Spaces and implementing the two levels of compliance, the Smart Devices compliance (Level 3) and the roaming devices compliance (Level 4).

A Smart Space (SSP) is an environment where one or more IoT platforms coexist, each of them providing some kind of service. Such environments are typically identified with physical locations, which can range from wide spaces to small areas; a Smart Space defines abstract boundaries for the IoT services and platforms it embraces, and acts as a sort of gateway from local resources to the rest of the symbIoTe environment.

The system is composed of five components; four of which are implemented from scratch and one is derived from commercial products and/or third party solutions. The existing symbIoTe Libraries have been used and extended for the Smart Spaces necessities.

This document is meant to explain the decisions taken for the software and system design according with the directives decided in the WP1 (described in deliverable D1.4 [3]) and to report the state of its implementation.

The document is structured as follows:

- Chapter 2 contains a detailed description of the requirements the symbIoTe Smart Space needed/used as guidelines for the software implementation.
- Chapter 3 contains description of the implemented components and their interface specification. The main outcome of Task 4.3 is the symbIoTe software, published on the project's GitHub repository.
- In Appendix, additional resources are given, including the guide through Middleware deployment and required sequence diagrams.

The repository contains the source code of all implemented components, configurations and guidelines for software installation, setup and usage.

2 Requirements

The symbIoTe Smart Space Middleware is designed and implemented according to requirements defined in the earlier stages of the project (in deliverable D1.4 [3]). Table 1 lists the functional requirements relating to Smart Spaces, which support symbIoTe Level-3 and Level-4 compliance (L3 and L4), while Table 2 lists all the security requirements for Smart Spaces.

The majority of listed requirements relates only to Level-3 compliance, while some of the requirements also relate to Level-4 compliance, which offers support for device roaming. The compliance Level to which a certain requirement is related to is designated in column "CL".

All the listed requirements are functional, i.e., they describe the behaviour of the symbIoTe system, i.e., what the symbIoTe architecture should do. All of the requirements fall into one of the following categories: Interface, Management, or Security.

- Interface refers to the methods employed to enable the interaction between different entities in the symbIoTe architecture, as well as between the symbIoTe system and end users.
- Management refers to all types of functional and non-functional requirements related to the handling or control of resources in symbIoTe.
- Security-related requirements are listed in a separate table, and encompass all security aspects of the symbIoTe Smart Spaces architecture including authentication, authorization, privacy, etc. The security requirements for Smart Spaces are mostly shared with other compliance levels.

Each requirement is characterized by its importance level with respect to its fulfilment by the symbIoTe architecture and system. The level of each requirement is expressed within the corresponding description text using the appropriate terminology. Following the Best Current Practices, the following levels are considered:

- *MUST (SHALL):* this is an absolute requirement, it is mandatory for the symbIoTe architecture and system to conform to this requirement.
- *SHOULD (RECOMMENDED):* there may exist valid reasons within particular circumstances to ignore this requirement.
- *MAY (OPTIONAL):* a requirement for a feature or a property of the symbIoTe architecture that presents low priority within the project and may or may not be fulfilled, subject to time or other constraints. Usually such features are selected by different vendors subject to their market positioning or specific needs.

Each requirement is linked with the use case specified within the symbIoTe project. These are indicated via the following indexes:

1. Smart Residence,
2. EduCampus,
3. Smart Stadium,
4. Smart Mobility & Ecological Routing, and
5. Smart Yachting.

Some requirements may appear to apply to none of the described use cases; such requirements are considered generic and are applicable to additional use cases beyond those defined within the scope of the symbIoTe project.

2.1 Requirements for Smart Spaces

Index	CL	Type	Category	Importance	Note on importance	Description	Use Cases
54	3	Functional	Interface	MUST		The system MUST enable the discovery and registration of a new device that is willing to be registered with symbIoTe compatible platform middleware.	1, 2, 3, 5
55	3, 4	Functional	Interface	MUST		Any piece of equipment which needs to be integrated with symbIoTe is required to have a documented digital interface, providing either a standard or a properly described protocol.	1, 2, 3, 5
56	3	Functional	Management	SHOULD		The system SHOULD be able to prioritize the information sent to the platform (IMPORTANT information 1st)	1, 3
57	3	Non-Functional	Interface	SHOULD		The system SHOULD support the dynamic configuration of a subset of commercial sensors.	1, 3
58	3	Functional	Interface	MAY		Inside Smart Space multiple gateways MAY be used as an alternative fallback router for a given device.	1, 2, 3
59	3	Functional	Management	SHOULD		SymbIoTe smart spaces SHOULD be able to operate without a permanent Internet connection.	1, 2, 3, 5
60	3	Functional	Management / Interface	SHOULD	Useful in case of limited connectivity	Different local IoT Platforms SHOULD be able to interact locally (i.e. without mediation from cloud-based L2 symbIoTe components).	1, 2, 3, 5
61	3	Functional	Management / Interface	SHOULD		Different collocated IoT Platforms SHOULD (or even MUST) be able to interact locally with mediation from symbIoTe Cloud components.	1, 2, 3, 5
62	3	Functional	Management	SHOULD	Useful in case of limited connectivity	A device running a symbIoTe app or a Smart Device SHOULD be able to access a Smart Space even if Internet connectivity is not available	1, 2, 3
63	3	Functional	Management	MUST	Important in case of limited connectivity (similar to #62, but the device is already associated)	A device running a symbIoTe app, when already associated to a Smart Space, MUST be able to access a Smart Device in that same Space even if Internet connectivity is not available.	1, 2, 3
64	3	Functional	Management	MUST	Important for identification of roaming devices	An L4 Compliant Smart Device MUST have a globally unique identifier.	1, 3
65	3	Functional	Management / Interface	SHOULD	Useful for roaming devices	An app/enabler SHOULD be able to receive a notification whenever an L4 Compliant resource it is using changes Smart Space association.	1, 2, 3
66	3	Functional	Management / Interface	SHOULD	Useful in case of limited connectivity	There SHOULD be a way for a local symbIoTe app to directly interface with the hosting Smart Space, that is by accessing it through the LAN rather than the Internet.	1, 2, 3, 5
67	3, 4	Functional	Management /Interface	MUST		SymbIoTe MUST accept visiting devices to be merged in the visited Smart Space.	1, 2, 3, 5
68	3	Functional	Management	MAY		The system MAY support IoT service / platform operators to	1, 2, 3, 5

						alter the registration of their resources during runtime of applications.	
69	3	Functional	Interface	SHOULD		The symbIoTe on board gateway shall support the following digital interfaces: dry contacts, serial bus connections, Ethernet connections, other standard buses to be evaluated	5
70	3	Functional	Interface	MUST		The symbIoTe middleware components MUST be able to manage authentication and authorization functions.	1, 2, 3, 5
71	3	Functional	Interface	SHOULD		There SHOULD be a management interface to manage authN/authZ mapping between the local IoT Platform and symbIoTe core.	1, 2, 3
72	3	Functional	Management	SHOULD		The symbIoTe middleware SHOULD be able to interface with the local IoT Platform's functions to manage resource monitoring and accounting.	2, 3
73	3	Functional	Management	SHOULD		The symbIoTe middleware SHOULD be able to provide a mapping between potentially different metrics used across the Platform's border.	2, 3
74	3, 4	Functional	Interface	MUST		The symbIoTe middleware MUST be able to exchange information with the local IoT Platform regarding currently associated devices, as well as regarding devices leaving or requesting to join the local space.	1, 2, 3, 5

Table 1: Smart Space Requirements

2.2 Security requirements for Smart Spaces

Index	CL	Type	Category	Importance	Note on importance	Potential barrier for uptake	Description	Use Cases
1	1,2,3, 4	Functional	Security	MUST	Important for interoperability and to control the access to the resources exposed by an IoT platform. It is needed for the authorization functionality.		The system MUST offer mechanisms for the **authentication** of symbIoTe entities/actors i.e., users/application developers, IoT Platforms, developed applications and clients.	1, 2, 3, 4, 5
2	1,2,3, 4	Functional	Security	MUST	Important for interoperability and to control the access to the resources exposed by an IoT platform. Platforms want to control the access over the		The system MUST offer mechanisms for the **authorization** of symbIoTe entities/actors i.e., users/application developers, IoT Platforms, developed applications and clients.	1, 2, 3, 4, 5

					resources.			
4	SSP	Functional	Security	SHOULD	Useful to ensure the authentication and authorization requirements for use cases that won't be online all the time.		The authentication and authorization to a smart space SHOULD work even if the smart space is disconnected from the Internet.	1, 2, 3, 5
9	1, 2, 3	Functional	Security	MUST	Important to securely protect data and that anyone else to have access to it		The system MUST support encrypted data communication between all involved entities on level 1 and 2 (e.g. the SymbIoTe core, Platforms, etc.).	1, 2, 3, 4, 5
10	3	Non-Functional	Security	MUST	Important for privacy issues.		The system MUST ensure privacy protection on each layer, do not publicly expose e.g., devices information or services used by applications.	1, 2, 3
16	1,2,3, 4	Functional	Security	MUST	To simplify the way the access rules are defined.		Access rules MUST be defined as an access policy.	
17	1,2,3, 4	Functional	Security	MUST	Important for interoperability.		The system MUST allow entities to delegate access to specific resources to other entities (e.g. by the usage of bearer access tokens)	1, 2, 3, 4
20	1, 4	Functional	Security	MUST	To avoid to man-in-the-middle attacks and identity spoofing.		Mutual authentication must be supported by all security mechanisms. (I.e. NOT only the user/application/software/... must be authenticated against the platform but also vice versa in order to facilitate malicious platform detection) Mutual authentication must be provided also in the communication between smart devices	4
21	1, 3	Functional	Security	MUST	Important for interoperability. Using ABAC it is possible to cover more options. ABAC allows higher level of flexibility.		The access to resource MUST be handled through 'Attribute-Based Access Control (ABAC)' schemes. An 'attribute' refers to a generic property/role/permission that the application grants during the authentication phases.	1
22	4	Functional	Security	MUST	Interoperability and security between smart devices.	Constraints on the device	The link-level communication between two smart devices MUST be authenticated, encrypted, and integrity-protected. To this end, security mechanisms MUST be properly designed by considering specific security needs,	1, (5)

							the set of requirements expressed in terms of latencies, bandwidth and energy consumption, as well as the used communication technologies.	
23	1,2,3, 4	Functional	Security	MUST	To detect security attacks and discover not security related malfunctions.		The system MUST detect anomalies that appear in the usage of the system for instance abnormal consumption of resources like temperature sensors that indicates an attempt of a DoS/DDoS attack. Supposing that a temperature sensor in Smart Home is polled 8 times an hour on average. Suddenly we observe that in a given time interval this sensor has been polled 100 times in 10 minutes. Anomaly detection module should detect it and send a log to the Platform where the user that has polled the sensor was registered.	1, 2, 3, 4, 5
24	1,2,3, 4	Functional	Security	MAY	To confirm or not the trust in the platform federation.		The system MAY detect anomalies that appear in the metadata provided by Platforms and devices. (e.g. The system MAY provide secure mechanisms to provide trusted location/proximity information.)	3

Table 2: Smart Space Security Requirements

3 Final Design of L3/L4 components

Smart Spaces are local environments (e.g. residence, campus, vessel, stadium, city area, etc.) where multiple IoT Platforms, IoT Gateways and Smart Devices (SDEVs) co-exist. In order to homogenize the communication between these entities and enable their interaction with 3rd Party applications, proper software adapters are needed. This document refers to the software implemented under symbIoTe project for that purpose as symbIoTe Smart Space.

SymbIoTe Smart Spaces are entities exposing local registered resources in a homogeneous manner, regardless of whether these resources belong to IoT Platforms, exist behind IoT Gateways or are simply standalone SDEVs. Any entity inside the Smart Space, after it is successfully recognized and authorized by the system, has to be able to access any locally registered resources whose policies allow its access. Furthermore, symbIoTe Smart Spaces should also facilitate 3rd party applications wanting to join the Smart Space services, fully implementing the interoperable nature of symbIoTe. In any case, any incoming entity should be identified and authorized in order to access any given Smart Space resource and this has to be possible even in the case of temporary failure or degradation of Internet connectivity.

Since IoT Platforms manage their resources according to their internal protocols, it is the IoT Platforms which will be responsible for the discovery and management of their own resources. IoT Platforms must register and unregister their resources according to those availabilities within the symbIoTe Smart Space and symbIoTe Smart Space will then take the charge of publishing this information to upper layers and within the SSP itself, the same way it does for SDEV resources which are directly managed by the SSP middleware software running on the SSP gateways. Hence, SDEV resources will be indistinguishable from the resources of a native IoT platform by an application using symbIoTe SSP as mean to manage those.

Among other functions, symbIoTe Smart Space is also responsible for advertising Level-4 (L4) compliant SDEVs (the roaming devices) to symbIoTe Core. To make this functionality possible, a specific security scheme has been designed and implemented. For Level-3 (L3) compliant devices, advertising those local resources to symbIoTe Core is not mandatory and can be selected by configuring it during the resource registration process. Finally, the symbIoTe SSP software suite includes a Lightweight Security Protocol implemented to let resource-restricted devices communicate with an adequate level of confidentiality. All these are described in the paragraphs that follow.

The symbIoTe Smart Space software implementation is built upon software modules already available and used for higher symbIoTe layers (namely L1 and L2), aiming to maintain as much as possible the software architecture, the interfaces and maximize the code modularity and reuse. In conclusion, symbIoTe Smart Space tries to achieve the following high-level goals:

- Interact with the symbIoTe Core layer components, exposing local resources for queries and actuation.
- Interact with local Smart Devices and IoT Platforms and gateways, providing the means to connect and share new resources.
- Maintain a certain degree of autonomy allowing the middleware to function when no internet connection is available.

The symbIoTe Smart Space consists of three main components; the Innkeeper (INK), the Resource Access Proxy (RAP) and the Local Authentication and Authorization Manager (Local AAM). Furthermore, there are three secondary components which facilitate the integration of SDEVs and IoT Platforms to the symbIoTe Smart Space. These are the RAP Gateway, Platform Agent and symbIoTe Agent. The basic architecture is presented in the Figure 1:

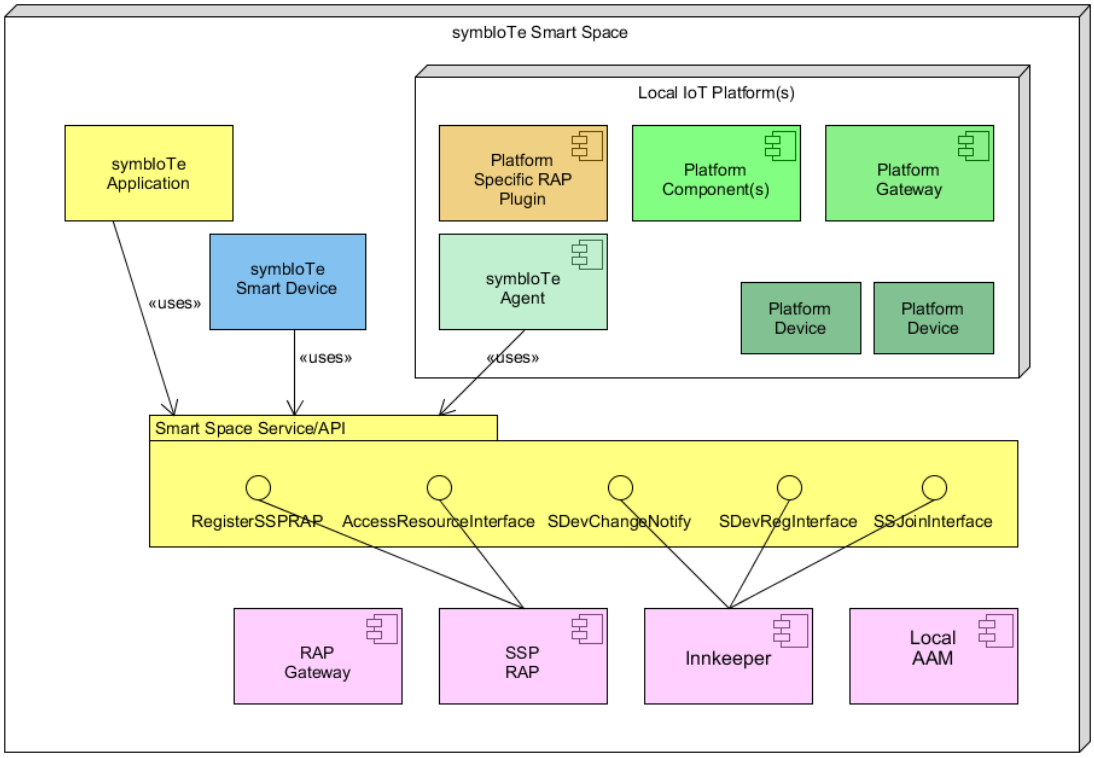


Figure 1: symbIoTe Smart Space Architecture

The role of each of these components are described in the sections below.

3.1 Components: Administration

3.1.1 Administration description

This component facilitates the control and administration of the symbIoTe Core Services by providing a web-based GUI. symbIoTe administrators have access to a control panel that allows them to perform management actions such as removing specific Platforms from the registry.

The administration will also provide features to non-administrator users. It will enable IoT Platforms and Applications to register with symbIoTe and to receive credentials that are required for the subsequent usage of symbIoTe services. Particularly for L3 and L4, Smart Space owners will be able to create or delete their Smart Spaces from the symbIoTe ecosystem.

The current Administration release provides the following L3/4-specific features:

- Operation management of Smart Spaces i.e. registration, update and delete from symbIoTe ecosystem.

- Creating Smart Space configuration for facilitating the deployment procedure.

3.1.2 Administration interfaces

Information on L3 and L4-specific Administration interfaces is presented in Table 3 below.

	Interface	Name(s)	Message Type	From	Message Consumers	Address/Queue	Payload	Response	Description
1	SSP Manage Request	SSP Manage Request	AMQP	Administration	Core AAM	Exchange: symbIoTe.AuthenticationAuthorizationManager Routing key: symbIoTe-AuthenticationAuthorizationManager.manage_smartspace_request	SmartSpaceManagementRequest	SmartSpaceManagementResponse	Administration informs Core AAM for a SSP Manage Request e.g. SSP registration, update, deletion
2	SSP Registration Request	SSP Registration Request	AMQP	Administration	Registry	Exchange: symbIoTe.ssp Routing key: symbIoTe.ssp.creationRequested	SmartSpace	SspRegistryResponse	Administration informs Registry for a SSP registration request
3	SSP Update Request	SSP Update Request	AMQP	Administration	Registry	Exchange: symbIoTe.ssp Routing key: symbIoTe.ssp.updatedRequested	SmartSpace	SspRegistryResponse	Administration informs Registry for a SSP update request
4	SSP Deletion Request	SSP Deletion Request	AMQP	Administration	Registry	Exchange: symbIoTe.ssp Routing key: symbIoTe.ssp.removalRequested	SmartSpace	SspRegistryResponse	Administration informs Registry for a SSP deletion request
5	SSP GetDetails Request	SSP GetDetails Request		Administration	Registry	Exchange: symbIoTe.ssp Routing key: symbIoTe.ssp.sspRequested	String sspId	SspRegistryResponse	Administration requests SSP Details from Registry

Table 3: L3 and L4-specific Administration interfaces

3.2 Component: Innkeeper (INK)

3.2.1 Innkeeper description

The Innkeeper (INK) is the component in charge to receive registration from devices/Platforms agents. It keeps the consistency of the Smart Space resources and communicates to the RAP for their reachability. The Innkeeper is also in charge to communicate with the core level components to let the Smart Space integrate with the rest of the symbIoTe ecosystem.

The Innkeeper is also in charge of communicating with L3/L4-compliant applications and devices in order to enable their registration and interaction with the SSP. It mainly fills the role of a local registry in the SSP, providing a list of applications and resources currently registered in the SSP. Furthermore, it could provide information about the SSP resources such as the current status and location.

The Innkeeper is also required for interaction with upper layers. Specifically, it updates information (e.g. location) of L4 roaming devices in upper layers by communicating with the Registration Handler of the IoT platform owing to the roaming device. Then, Registration Handler should forward the updated information to the symbIoTe core.

Innkeeper exposes REST interfaces to enable communication with L3/L4-compliant applications and devices. In the following section, Table 4 lists the Innkeeper Interfaces.

3.2.2 Innkeeper interfaces

	Interface	Name(s)	Message Type	From	Msg Consumers	Address/Queue	Payload	Description

1	Platform / L3/L4 SDEV Registration	/innkeeper/platform/register
		
		/innkeeper/sdev/register
			REST	Sym-Agent	Innkeeper	POST	{
							String symId,
							String sspId,
							String pluginId,
							String pluginURL,
							String dk1,(only for SDEV)
							String hashField (only for SDEV)
							}
								The L3/L4 SDEVs register in the SSP through the Innkeeper
								
								NOTE: if the registration request is provided by an L3 SDEV id is a “fresh Id”. If the registration is provided by a L4 SDEV Id contains an unique Id.

2	Platform / L3/L4 SDEV Unregistration	/innkeeper/platform/unregister
		
		/innkeeper/sdev/unregister
			REST	Sym-Agent	Innkeeper	POST	{
							String sspId (or symId)
							}
								Unregistration SDEV request.

3	SDEV Core Registration/Modify/Delete	{coreIntefaceUrl}/ssps/{sspName}/sdevs	REST	Innkeeper	Core	POST/PUT/ DELETE	{
							String symId,
							String sspId,
							String pluginId,
							String pluginURL,
							String dk1,
							String hashField
							}	

4	Resource Registration	/innkeeper/platform/join
		
		/innkeeper/sdev/join
			REST	Sym-Agent	Innkeeper	POST	{
							String internalIdResource,
							String sspIdResource,
							String sspIdParent
							IAccessPolicySpecifieraccessPolicy,
							IAccessPolicySpecifierfilteringPolicy,
							Resourceresource
							}	

5	Keep Alive	/innkeeper/keep_alive/	REST	Sym-agent	Innkeeper	POST	{
							String sspId (or symId)
							}
								Send a keep-alive message.

6	Resource Core Registration / Modify / Delete	{coreIntefaceUrl}/ssps/{sspName}/{sdevId}/ resources	REST	Innkeeper	Core	POST/PUT/ DELETE	{ Resourceresource }	
7	Public resources request	innkeeper/ public_resources	REST	*Innkeeper*	*App*	GET	See symbIoTe platform / SDEV L3/4 integration#3.3Searchforresources	Request the list of registered public resources in SSP, return: a JSON list of public registered resources

Table 4: Innkeeper Interfaces

3.2.2.1 Platform / L3/L4 SDEV Registration

Platforms or Smart Devices (SDEV) can request to be registered in a Smart Space (SSP). The Registration procedure for a Smart Device is provided by using the Lightweight Security Protocol . The message body of the registration request is encrypted and formatted as follow:

- Regarding the field ***symId*** (i.e. the symbIoTe id), if this is the first time the device connects to symbIoTe, then it should be an empty field; the Innkeeper then responds with the id that device should save in the Flash memory (if L4) and it should re-use in future interactions with the symbIoTe ecosystem.
- ***PluginId*** and ***pluginURL*** are metadata used by the RAP. The ***pluginURL*** is the ip:port/path where the RAP sends the request to the SDEV.
- ***Roaming*** indicates if the SDEV is a L3 or L4 device to the Innkeeper during the registration
- The field ***dk1*** represents the current session key.
- Regarding the ***hashField*** could be
    - all 0 when the SDEV joins for the first time or
    - hashField = H(symId || previous dk1)

If *SSP has internet access*, registration procedure returns a response message assigning an unique symbIoTe Id provided by core e.g.:

```
{
  "symId":"sym787",
  "sspId":"4",
  "result":"OK",
  "registrationExpiration":3600
}
```

where:

- **symId** is the symbIoTe Id of SDEV, provided by the Core
- **sspId** is the local SSP id of the registered SDEV, which is unique in the Smart Space domain
- **result** is a self-explained response message
- **registrationExpiration** is the lifetime of a Smart Device Registration.

Each Smart Device should implement a mechanism, which periodically sends a Keep-alive message to the Innkeeper in order to inform its the presence in the network.

If *SSP is OFFILNE*, the registration response is like:

```
{
  "symId":"",
  "sspId":"3",
  "result":"OFFLINE",
  "registrationExpiration":3600
}
```

If the Smart Device is already registered:

```
{
  "symId":"sym941",
  "sspId":"6",
  "result":"ALREADY_REGISTERED",
  "registrationExpiration":3600
}
```

If a Smart Device, with a previously assigned Symbiote Id, tries to register to another Smart Space, it should provide a registration request by filling the ***symId*** field, obtaining a registration response with a new ***sspId***. If a Smart Device tries to register using a non-existent symbIoTe Id, the Innkeeper returns a reject message:

```
{
  "symId":"0100fakeId4053",
  "result":"REJECTED",
  "registrationExpiration":0
}
```

3.2.2.2 Platform / L3/L4 SDEV un-registration

If a Smart Device owner needs to disconnect the from the SSP, it should send a unregistration message to the Innkeeper:

```
curl -H 'Content-Type: application/json'-d '
{
  "sspId":"3"
}
' -X POST -D - http://ssp.symbiote.org:8080/innkeeper/sdev/unregister
```

The only information needed is the ***sspId***. The Innkeeper will provide to delete the Smart Device instance with all its resources.

The response from the Innkeeper is just an HTTP OK 200 with no payload.

3.2.2.3 Resource Registration

In SSP a Resource is defined as a Json message obtained by SspResource class. The field contained are:

- **internalIdResource** is an internal id of the resource, it is not compliant with symbIoTe ecosystem and can be a MAC address or other type of identifier defined by Smart Device vendor;
- **sspIdResource** should be empty in first resource registration;
- **sspIdParent** is the SSP id of the Smart Device which contains the resource;
- **symIdParent** is the symbIoTe ID of the Smart Device which contains the resource;
- **accessPolicy** is a JSON of serialized AccessPolicySpecifier;
- **filteringPolicy** is a JSON of serialized AccessPolicySpecifier;
- **resource** is a JSON of serialized Resource instance.

A Smart Device can contain more Resources and each Smart Device should perform a resource registration for each resource. A resource registration response provided by the Innkeeper is a JSON message containing following information:

```
{
  "symIdResource":"sym970",
```

```
  "sspIdResource":"1",
  "symId":"sym609",
  "sspId":"0",
  "result":"OK"
}
```

where:

- ***symIdResource*** is the Symbiote ID of the registered resource and it is generated by the Core during the first registration of the Smart Device and stored in the Flash memory;
- ***sspIdResource*** is the local SSP id for a resource which is unique in the Smart Space domain;
- **symId** is the symbIoTe Id of the parent Smart Device which contains the registered resource. symId is unique and stored in Flash Memory of the Smart Device and obtained during the first registration;
- **sspId** is the local SSP id of the parent Smart Device which is unique in the Smart Space domain;
- **result** is a self-explained response message.

3.2.2.4 SDEV Core Registration/Modify/Delete

This is the payload of a SDEV registration to Core:

```
{
"body": {
"symId":null,
"sspId":null,
"pluginId":"SSP_TEST",
"pluginURL":"https://symbiote.tti.unipa.it",
"roaming":true,
"dk1":"newkey",
"hashField":"hashedsecret"
        }
}
```

3.2.2.5 Keep Alive

Keep alive messages provide two functionalities:

1. Announce periodically to the Smart Space that a Smart Device is working.
2. Update the symbIoTe Id for a Smart Device and its resources if an offline registration occurred. The message contains only the ssp ID.

An example of such a message is the following:

```
curl -H 'Content-Type: application/json'-d '
{
  "sspId":"6"
}
' -X POST -D - http://ssp.symbiote.org:8080/innkeeper/keep_alive
```

Show below is a JSON keep-alive response message to a Smart Device which contains four resources.

```
{
  "symId":"sym941",
  "sspId":"6",
  "result":"OK",
  "updatedSymId":[
    {
      "sspIdResource":"0",
      "symIdResource":"sym510"
    },
    {
      "sspIdResource":"1",
      "symIdResource":"sym728"
    },
    {
      "sspIdResource":"2",
      "symIdResource":"sym493"
    },
    {
      "sspIdResource":"3",
      "symIdResource":"sym843"
    }
  ]
}
```


3.2.2.6 Resource Core Registration / Modify / Delete

Some description would be useful, one-liner is more than OK.


```
body":
{
"5c:cf:7f:3a:6b:76":
{
"@c": ".StationarySensor",
"name": "Stationary 1",
"description": [
"This is stationary 1"
],
"interworkingServiceURL": "https://www.example.com/Test1Platform",
"locatedAt":
{
"@c": ".WGS84Location",
"longitude": 5.349014,
"latitude": 25.864716,
"altitude": 35,
"name": "SomeLocation",
"description": [
"Secret location"
]
},
"featureOfInterest":
{
"name": "Room1",
"description": [
"This is room 1"
```

```
],
"hasProperty": [
"temperature"
]
},
"observesProperty": [
"temperature",
"humidity"
]
}
},
"filteringPolicy":
{
"policyType": "PUBLIC",
"requiredClaims": {}
},
}
```

3.2.2.7 Public resources request

The SSP allows external applications to obtain the available resources. Currently, SSP implements an endpoint that returns the list of all *PUBLIC* resources:

```
curl -x GET http://ssp.symbiote.org:8080/innkeeper/public_resources
```

Here an example of the response message:

```
[
  {
    "internalIdResource":"5c:cf:7f:3a:6b:76",
    "sspIdResource":null,
    "sspIdParent":"0",
    "symIdParent":"sym48",
    "resource":{
      "@c":".Actuator",
      "id":"",
      "name":"ACT-aggeggio",
      "description":null,
      "interworkingServiceURL":",
      "locatedAt":null,
      "services":null,
      "capabilities":[
        {
          "parameters":[
            {
              "name":"r",
              "mandatory":true,
              "restrictions":[
                {
                  "@c":".RangeRestriction",
                  "min":0.0,
                  "max":255.0
                }
              ],
              "datatype":{
                "@c":".PrimitiveDatatype",
                "array":false,
                "isArray":false,
                "baseDatatype":"xsd:unsignedByte"
```

```
            }
          },
          {
            "name":"g",
            "mandatory":true,
            "restrictions":[
              {
                "@c":".RangeRestriction",
                "min":0.0,
                "max":255.0
              }
            ],
            "datatype":{
              "@c":".PrimitiveDatatype",
              "array":false,
              "isArray":false,
              "baseDatatype":"xsd:unsignedByte"
            }
          },
          {
            "name":"b",
            "mandatory":true,
            "restrictions":[
              {
                "@c":".RangeRestriction",
                "min":0.0,
                "max":255.0
              }
            ],
            "datatype":{
              "@c":".PrimitiveDatatype",
              "array":false,
              "isArray":false,
              "baseDatatype":"xsd:unsignedByte"
            }
          }
        ],
        "effects":null,
        "name":"RGBCapability"
      }
    ]
  }
 }
]
```


3.3 Component: SSP RAP

3.3.1 SSP RAP description

Smart Space (SSP) Resource Access Proxy (RAP) component enables symbIoTe-compliant access to resources within IoT Platforms located in a SSP or to SDEVs. It receives incoming access requests from applications/platform agents using a symbIoTe-compliant communication protocol and data format. A request must contain a unique identifier assigned to a resource. It checks if those security policies included in the request are valid and that access to a particular resource can be granted.

The data generated by IoT platform / SDEV must be returned in a format which complies with the symbIoTe information model.

3.3.2 SSP RAP interfaces

The SSP RAP exposes both REST and OData interfaces for direct resources' access. It also communicates with the Innkeeper component via function calls (both resides in the Middleware application). An additional interface is used for push mechanism, where notifications are linked via WebSocket with the client application.

Table 5 below contains a summary of SSP RAP's external interfaces.

#	Interface	Name	Message Type	From	Msg Consumers	Address/ Queue	Payload	Description
1a	Resource access read	/rap/Sensor/{resourceId}	REST	Application / Agent	RAP	GET	None - replies with the value of the resource	Event reading the value of a resource
1b	Resource access read	/rap/Sensor({resourceId})/Observations?$top=1	OData	Application / Agent	RAP	GET	None - replies with the value of the resource	Event reading the value of a resource
2a	Resource access read history	/rap/Sensor/{resourceId}/history	REST	Application / Agent	RAP	GET	None - replies with the history values of the resource	Event reading the value of a resource
2b	Resource access read history	/rap/Sensor({resourceId})/Observations	OData	Application / Agent	RAP	GET	None - replies with the history values of the resource	Event reading the value of a resource
3a	Resource access	/rap/Service/{resourceId}	REST	Application / Agent	RAP	POST	{ [“param_name”:“param_value”, ..] }	Event sending the value to a service
3b	Resource access	/rap/Service({resourceId})	OData	Application / Agent	RAP	PUT	{ [{“param_name”:“param_value”}, ..] }	Event sending the value to a service
4a	Resource access write	/rap/Actuator/{resourceId}	REST	Application / Agent	RAP	POST	{ [{ “capabiliy_name”: [{“param_name”:”param_value”}, ..] }, ..] }	Event writing the value of a resource
4b	Resource access write	/rap/Actuator{resourceId}	OData	Application / Agent	RAP	PUT	{ [{ “capabiliy_name”: [{“param_name”:”param_value”}, ..] }, ..] }	Event writing the value of a resource
5	Resource notifications	/notification	WebSocket	Application / Agent	RAP	client / server	the value of the resource	Event reading the value of a resource

Table 5: SSP RAP's external interfaces

3.4 Component: Local AAM

3.4.1 Local AAM description

Local Authentication and Authorization Manager (Local AAM) is a component that handles the authentication procedure for Smart Space (SSP) components, applications registered in a particular Smart Space federated with symbIoTe and Platform Agents (Table 6). It enables the core centric security function while internet connection is established, but it becomes Smart Space centric when no internet connection is available (allowing the SSP to work also when disconnected). After a successful authentication, the Local AAM releases a home token storing attributes, properties, roles and permission assigned to the component or application within the SSP where it is registered.

Local AAM performs the same function as Platform and Core AAMs on L2: it handles token validation, issuing the certificates, revocation of compromised credentials and user management. Additionally, all services are updated to work with SSP (getAvailableAAMs), and registration was extended for Platform Agents.

3.4.2 Local AAM interfaces

#	Interface	Name	Message Type	From	Msg Consumers	Address/ Queue	Payload	Description
1	Get available AAM	/get_available_aams	HTTP	REST client	AAM	GET	AvailableAAMsCollection, HTTP status: 200 on ok and 500 on error	Returns information about all available AAMs in the system
2	Get internally AAM	/get_internally_aams	HTTP	REST client	AAM	GET	AvailableAAMsCollection, HTTP status: 200 on ok and 500 on error	Returns information about all available AAMs in the system containing internal urls
3	Get component certificate	/get_component_certificate/platform/{platformIdentifier}/component/{componentIdentifier}	HTTP	REST client	AAM	GET	component certificate in PEM format, HTTP status 200 or 404 on missing, 500 on error	Returns component certificate in PEM format
4	GET user detail	/get_user_details	HTTP	REST client	AAM	POST	UserDetails, HTTP status code (200, 400 missing user, 401 bad user password)	Return registered user details
5	Manage user	/manage_users	HTTP	REST client	AAM	POST	ManagementStatus, HTTP status code	Used to manage users (create, update, delete…)
6	Issue new certificate	/sign_certificate_request	HTTP	REST client	AAM	POST	certificate in PEM format, HTTP status	Used to issue new certificate for client/component/platform
7	Get home token	/get_home_token	HTTP	REST client	AAM	POST	Headers with X-Auth-token containing token String for that client	Returns HOME token used to access restricted resources offered in SymbIoTe
8	Get foreign token	/get_foreign_token	HTTP	REST client	AAM	POST	Headers with X-Auth-token containing FOREIGN/ROAMED/FEDERATED token String for that client	Returns FOREIGN token used to access restricted resources offered in SymbIoTe federations
9	Get guest token	/get_guest_token	HTTP	REST client	AAM	POST	Headers with X-Auth-token containing GUEST token String	Returns GUEST token used to access public resources offered in SymbIoTe
10	Validate token	/validate_credentials	HTTP	REST client	AAM	POST	Headers with: X-Auth-token containing Authorization Token String for that client; (opt) X-Auth-Client-Cert containing PEM Certificate String matching SPK from token (opt) X-Auth-AAM-Cert containing PEM Certificate String used to sign the client	Verifies, if provided token is valid

							certificate (opt) X-Auth-ISS-Cert containing PEM Certificate String matching the ISS, IPK and signature from the FOREIGN token	
11	Validate revocation	/validate_foreign_token_origin_credentials	HTTP	AAM	AAM	POST	Foreign token String in body	Allows to confirm that the origin (HOME) credentials (SUB & SPK) used to issue the given FOREIGN token in another AAM have not been revoked
12	Revoke credential	/revoke_credentials	HTTP	REST client	AAM	POST	RevocationRequest	Allows to revoke compromised tokens and certificates

Table 6: Local AAM interfaces

3.5 Component: RAP GW

3.5.1 RAP GW description

Resource Access Proxy (RAP) Gateway (GW) component acts as a gateway for allowing access to local Smart Space resources for clients that are outside the SSP itself. This is necessary because the Smart Space Middleware could run inside a Local Area Network, which means that it does not necessarily have a public IP. RAP Gateway simply forwards messages from the external world (e.g. applications, enablers) to the SSP RAP. For this reason, a specific component is not needed, as many solutions and tools already exist. For instance, configuring a port forwarding on the local network router between the local IP address and a public IP is one of the solutions. A tool that would act as a RAP Gateway is *ngrok* [2], a commercial product that offers public URLs for exposing local web servers can be used for demo and testing purposes.

3.5.2 RAP GW interfaces

RAP Gateway exposes REST / OData interfaces in the place of symbIoTe SSP RAP component, so that they are reachable from the external world. Consequently, it does not expose any specific interfaces, but it just processes the request path in order to forward the message to the SSP RAP, that afterward will send to the appropriate recipient.

3.6 Component: SDEV Agent

3.6.1 SDEV Agent description

Smart Device (SDEV) Agent is a component that enables a device made from 3rd Parties to speak the symbIoTe language. Based on the interfaces defined in the following section, a device manufacturer that wants to transform its device in a symbIoTe enabled one (SDEV) should develop this agent on top of its system.

In the software release of the S3M, an agent for the Arduino platform ESP8266 is developed: the maker community is very active around the ESP8266 platform, so it is a good starting point to quickly create an SDEV.

The agent itself is composedof three main library parts:

- The lightweight security library, handling the security related registration process.
- The semantic library, building the semantic description of the SDEV resources, mandatory to present the resource to the symbIoTe ecosystem.
- The symbIoTe-agent library itself, linking the previous libraries together and it is the only class that should be embedded in the Arduino firmware of your SDEV.

The symbIoTe Agent library (named sym-agent) also links the custom function defined in the firmware for two types of possible behaviour of the SDEV: actuation and sensing.

A more detailed information is available in the last section of this Deliverable D4.3.

3.6.2 SDEV Agent interfaces

SymbIoTe Smart Device Agent interfaces are based on HTTP protocol. Two types of payload are defined: the encrypted payload, named SDEVP in the table below (??), and the non-encrypted payload as application/json in plain text.

Regarding the SDEVP, this is a custom name used to define the encrypted data payload carried by the HTTP POST json. Following example shows the body used in the SDEVP payload:

```
{
  "mti": "0x50",
  "sessionId": "RoOgqkr6",
  "data":                                                    "
/qRzoJJUdScoyt5amdL/qQW8CHkQjmMgCUycjlHOhAB/+99/+lyI9qIB/GKOog6"
}
```

Where:

- *mti* is the code defined in the Lightweight Security Protocol ;
- *sessionId* is the session identifier for the communication between Innkeeper and SDEV;
- *data* contains the encrypted JSON described as plain text in the table in the next page;

The two actors with whom the agent speaks are the Innkeeper and the SSP RAP. In Table 7 below, there is the list of interfaces the agent exposes:

#	Interface	Name	Message Type	From	Msg Consumers	Address/ Queue	Payload	Description
1	Registry	/innkeeper/sdev/registry/	SDEVP	sym-agent	Innkeeper	POST	{ String symId, String pluginId, String sspId, Bool roaming, String pluginURL, String dk1, StringhashField }	Registers SDEV. See note[1]
2	Join	/innkeeper/sdev/join	SDEVP	sym-agent	Innkeeper	POST	{ String internalIdResource, String sspIdResource, String sspIdParent, String symIdParent, "accessPolicy":{ "policyType": "PUBLIC", "requiredClaims": {} }, "filteringPolicy": { "policyType": "PUBLIC", "requiredClaims": {} }, "resource":{Resource} }	Register every single resource of the agent. See note[2]
3	Keep-alive	/innkeeper/keep_alive/	SDEVP	sym-agent	Innkeeper	POST	{ StringsspId }	Notify that SDEV is alive
4	GET Resource query	/rap/v1/request	REST	SSP RAP	sym-agent	POST	{ "resourceInfo": [{ String symbioteId, String internalIdResource, String type },{ String type="Observation" }], String type="GET" }	Get the value of a SDEV's resource
5	HISTORY	/rap/v1/request	REST	SSP RAP	sym-agent	POST	{	Get the history

[1] When the device connects to symbIoTe for the first time field *symIdSDEV* should be empty; the Innkeeper then responds with the id that device should save in the Flash memory (in case of L4) and device should re-use it in all future interaction with the symbIoTe ecosystem.
When the SDEV joins for the first time hashField could be (1) all "0" or (2) `hashField = H(symIdSDEV || (previous dk1) )`
The field dk1 represents the current session key.
[2] *resource* is a string containing the description of the SDEV using a semantic description compliant to symbIoTe ecosystem.
internalIdResource is the internal Id assigned from the agent to the resource, e.g. the mac address.

	Resource query						"resourceInfo": [{
							String symbioteId,
							String internalIdResource,
							String type
							
							},{
							String type="Observation"
							}],
							"filter": {
							String type,
							String param,
							String cmp,
							String val
							},
							Stringtype="HISTORY"
							}
								value of a SDEV's resource. *filter* can be *null*.

6	SET Resource	/rap/v1/request	REST	SSP RAP	sym-agent	POST	{
							"resourceInfo": [{
							String symbioteId,
							String internalIdResource,
							String type
							}],
							"body":{
							"{capability}": [{
							"{restriction}": "{value}"
							}]
							}
							String type="SET"
							}
								Actuate an action on the SDEV

7	Subscribe Resource	/rap/v1/request	REST	SSP RAP	sym-agent	POST	{
							"resourceInfo": [{
							String symbioteId,
							String internalIdResource,
							String type
							},{
							String type="Observation"
							}],
							String type="SUBSCRIBE"
							}
								Subscribe to a resource. SDEV periodically sends updates to RAP.

8	Un-subscribe Resource	/rap/v1/request	REST	SSP RAP	sym-agent	POST	{
							"resourceInfo": [{
							String symbioteId,
							String internalIdResource,
							String type
							},{
							String type="Observation"
							}],
							String type="UNSUBSCRIBE"
							}
								Unsubscribe to a resource.

9	PUSH data	/rap/v1/plugin/notification	REST	sym-agent	SSP RAP	POST	{
							String resourceId,
							"location": {location},
							String resultTime,
							StringsamplingTime,
							"obsValues": [{ObservationValue}]
							}
								Data packet sent from sym-agent to RAP.

10	Unregistry	/innkeeper/sdev/unregister	SDEVP	sym-agent	Innkeeper	POST	{
							StringsspId
							}
								Unregistration SDEV

Table 7: SDEV Agent interfaces

3.7 Component: Platform Agent

3.7.1 Platform Agent description

The Platform Agent component is the counterpart of the SDEV Agent for IoT Platforms. From one side, it is registering platform resources towards the SSP Innkeeper and, on the other side; it handles the access to these resources.

An IoT Platform provider/owner needs to provide the following information from the symbIoTe information model:

- IoT Device or Composite IoT Service description.
- Location with its properties.
- Observed Properties description and name.

Platform Agent will send the metadata describing resources to be registered to the Innkeeper that will store this information in a local registry and forwards (if needed) the registration to the symbIoTe Core. Moreover, when an external actor (i.e. application, enabler) is requesting the access to some platform resource, the agent needs to forward and adapt the request coming from the SSP RAP, to the proprietary platform APIs.

3.7.2 Platform Agent interfaces

The Platform Agent exposes a REST interface for receiving incoming SSP RAP messages that request access to resources. This endpoint is custom, and it is provided by the IoT Platform provider during the registration procedure, inside the resource description.

The interfaces of the platform agent are the same as the SDEV ones, the only two differences are:

- The path where the agent sends the request is `/innkeeper/platform/*` instead of `/innkeeper/sdev/*` .
- The keep-alive interfaces for Platform agents does not exist.

3.8 Security aspects: SDEV

In case of an SDEV, the challenge is to use a secure registration process without using complex computational power due to the lack of this resource in constrained devices as the SDEV typically are.

To resolve this issue, symbIoTe consortium defines a negotiation procedure that can be scalable in terms of computational power required by the end device. This procedure is called Lightweight Security Protocol .

Security services to be implemented between device and Gateway/Innkeeper, include:

1. algorithm negotiation,
2. peer authentication,
3. key agreement,
4. protection from attacks, including replay,
5. data confidentiality/authentication/integrity.

The protocol ensures a secure interaction: negotiated secrets are unavailable to eavesdroppers, even by an attacker who can place himself in the middle of the connection. Moreover, in the case the protocol ends successfully, communicating peers can protect their communication through symmetric cryptography (e.g., AES, ChaCha20, etc.) and reliable mechanisms (i.e., messages include an authentication tag which protects them against tampering). Indeed, the protocol provides in output all the details needed to support the 5-th goal of data confidentiality/authentication/integrity.

3.8.1 Negotiation protocol

Negotiation is initiated by the Smart Device (SDEV). Device and Gateway/Innkeeper agree on the cipher suite (i.e., cryptographic algorithms to use), negotiate and/or generate key material, and provide a proof of their authenticity. The protocol is designed to support simple approaches like pre-shared symmetric key (PSK), or more complex ones like Elliptic-Curve Diffie-Hellman (ECDH) with RSA or ECDSA (when a certificate is used) exchange modes. The key agreement mechanism is chosen based on device capabilities. As commonly accepted, key materials are generated through a Key Derivation Function (KDF).

Let Security Context be the set of security parameters useful to setup security services.

Figure 2 provides a high-level picture of the negotiation protocol. Let Message Type Indicator (MTI) be a field that identifies the type of message. It may assume the following values:

- **0x10**: *SDEV Hello*
- **0x20**: *GW_INK Hello*
- **0x30**: *SDEV AuthN*
- **0x40**: *GW_INK AuthN*

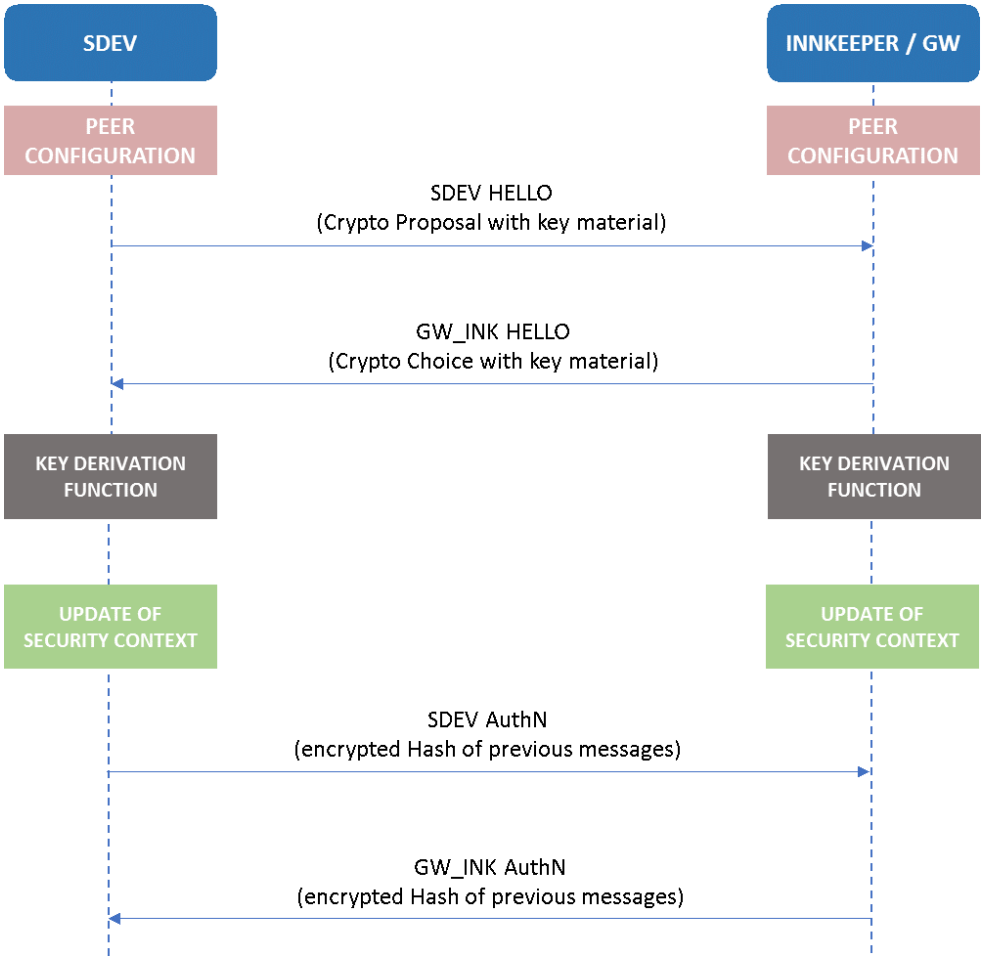


Figure 2: Negotiation Protocol

3.8.2 Pre-Shared Key Configuration

A unique PSK for each device must be defined to circumvent the problem of a compromised PSK in a smart space. Only one key natively must be stored at the Gateway/Innkeeper side. Three different levels of security can be identified:

3.8.2.1 Basic level

- Gateway/Inkeeper (GW/INK) stores a **Master Secret Key** (MSK), unique with the symbIoTe environment and shared among all SSPs.
- Every device directly stores the PSK = $H(MAC_{ADDR}||MSK)$, where H is a hashing function.
- GW/INK calculates PSK in real time.

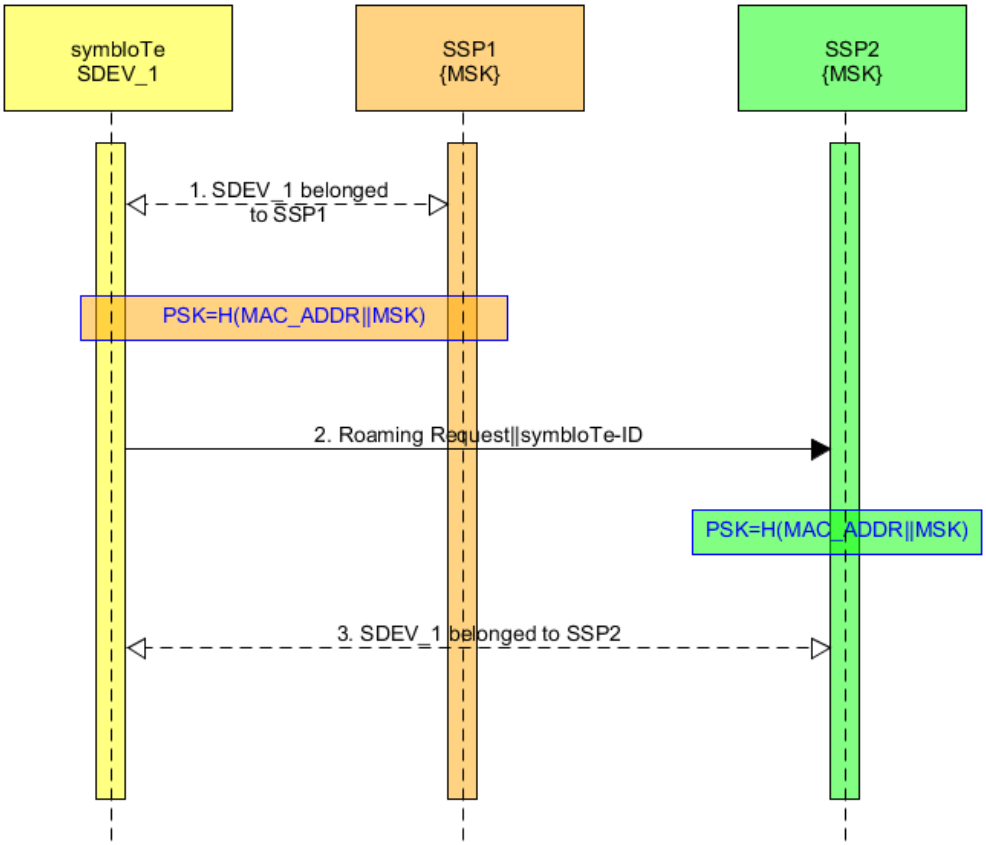


Figure 3: Basic Security Level

3.8.2.2 Intermediate level

- GW/INK stores a **Master Secret Key** (MSK), defined by the SSP owner for a given SSP. The core should know the MSK assigned to each SSP
- Each device belonging to the considered SSP is configured to store the PSK = $H(MAC_{ADDR}||MSK)$, where H is a hashing function
- GW/INK calculates PSK in real time
- In case of roaming, GW/INK of the new SSP could contact the core for obtaining the unique PSK assigned to a given device.

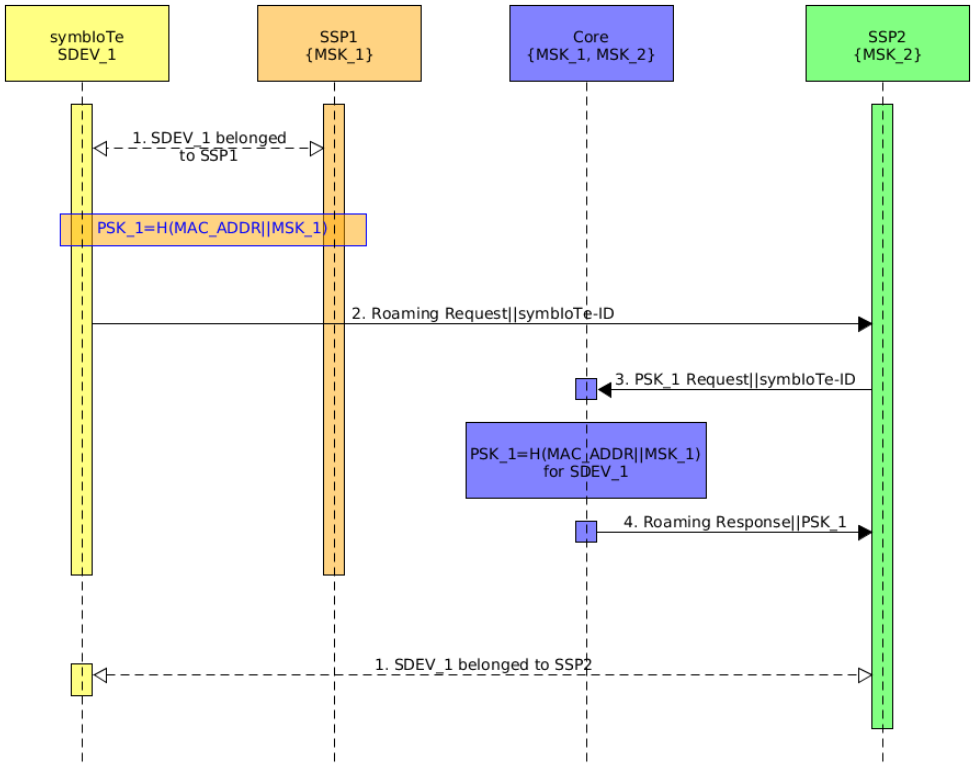


Figure 4: Intermediate Security Level

3.8.2.3 Advanced level

- Each device is configured with a unique PSK.
- The PSK is stored in GW/INK and core by the device owner
- In case of roaming, GW/INK could contact the core for obtaining the unique PSK assigned to a given device.

Despite the definition of multiple security levels, only the basic level is implemented at the time of the writing of the Deliverable. Other levels of security can be easily integrated in the future.

3.8.3 SDEV Hello message

SDEV Hello message is sent in plaintext by SDEV. It contains the related MTI code (that is MTI=0x10), the MAC address of the SDEV (namely $SDEV_{MAC}$), the supported Cipher Suites (namely $CRYPTO_{PROPOSAL}$) that contain a list with the related Cipher Suite IDs (It is recommended to use the ID defined by IANA for TLS Parameters [4]), the preferred Key Derivation Function (namely $KDF_{PROPOSAL}$) and a cryptographic nonce (namely $SDEV_{NONCE}$).

Optionally additional key material (namely $SDEV_{MATERIAL}$) is present when Public Key Cryptography is implemented. It stores a X.509 certificate.

$SDEV_{MAC}$ is used for identifying the Smart Device.

$CRYPTO_{PROPOSAL}$ describes the list of cryptographic algorithms supported by SDEV. They are expressed according to the following structure:

- Key Exchange Algorithm (to establish a mechanism by which both parties will negotiate a key to communicate authentically)
- Symmetric Encryption Algorithm (to encrypt the messages)
- Message Authentication Code Algorithm (to create a message digest of message)

More than one proposal can be reported within the list. The resulting protocol is, indeed, flexible. Possible examples include:

- **PSK_WITH_AES_128_GCM_SHA256**, where PSK is used to set a pre-shared key, AES as symmetric encryption algorithm with a 128 bit key, SHA256 as a pseudorandom function (PRF) based on HMAC (Hash-based message authentication code) with the SHA-256 hash function
- **PSK_WITH_CHACHA20_POLY1305_SHA256**, where PSK is used to set a pre-shared key, CHACHA20 as symmetric cipher with a 256 bit key, POLY1305 as a message authentication code that requires a 256 bit key and a message and produces a 128 bit tag
- **PSK_WITH_AES_128_CBC_SHA**, where PSK is used to set a pre-shared key, AES as symmetric encryption algorithm with a 128 bit key, SHA as a pseudorandom function (PRF) based on HMAC with the SHA hash function or as a message authentication code algorithm
- **ECDH_ECDSA_WITH_AES_128_CBC_SHA**, which requires that the GW/INK certificate's contain an ECDH-capable public key signed with ECDSA (both device and Gateway/Innkeeper perform an ECDH operation and use the resultant shared secret as the premaster secret), AES as symmetric encryption algorithm with a 128 bit key, SHA as a pseudorandom function (PRF) based on HMAC with the SHA hash function or as a message authentication code algorithm
- **ECDH_ECDSA_WITH_AES_256_CBC_SHA**, which requires that the GW/INK certificate's contain an ECDH-capable public key signed with ECDSA (both device and GW/Innkeeper perform an ECDH operation and use the resultant shared secret as the premaster secret), AES as symmetric encryption algorithm with a 256 bit key, SHA as a pseudorandom function (PRF) based on HMAC with the SHA hash function or as a message authentication code algorithm

The $KDF_{PROPOSAL}$ field proposes the Key Derivation Function (KDF) to be used for generating session keys. Possible proposal includes *PBKDF2* or *HKDF*. $SDEV_{NONCE}$ is used to protect the communication from replay attack together with GW_INK_{NONCE}. It builds the salt value in the KDF function. Optionally, $SDEV_{MATERIAL}$ is present when Public Key Cryptography is implemented. Therefore, it stores a X.509 certificate.

3.8.3.1 AEAD Mode

Some encryption algorithm like AES_128 with GCM or CHACHA20_POLY135, support **AEAD**. AEAD stands for "*Authenticated Encryption with Additional Data*" meaning there is a built-in message authentication code for integrity checking both the ciphertext and optionally additional authenticated (but unencrypted) data. In this protocol, $SDEV_{MAC}$||*sequence_number* is used as Additional Authenticated Data (AAD). The sequence number is the number of messages sent since the last handshake. It is incremented by 1 for each message. The sequence number could be start from a value obtained by the $SDEV_{NONCE}$ and GW_INK_{NONCE} sum, and it is used also to avoid replay

attacks. In fact, for "AEADless" encryption algorithms, a sequence number field (*sn*) must be defined.

3.8.4 *GW_INK Hello message*

As soon as the SDEV Hello is received, Gateway or Innkeeper verifies that $SDEV_{NONCE}$ is acceptable, $SDEV_{MAC}$ is stored within a database, and $CRYPTO_{PROPOSAL}$ contains an acceptable proposal. Then, it selects the most suitable Cipher Suite and sent back a new message containing the related MTI code (that is MTI=0x20), the selected Cipher Suite (namely $CRYPTO_{CHOICE}$), an optional Initialization Vector (IV), the nonce (namely GW_INK_{NONCE}), and optionally additional key material (namely $GW\_INK_{MATERIAL}$). $CRYPTO_{CHOICE}$ uses the same structure as the $CRYPTO_{PROPOSAL}$. IV can be used along with a secret key for data encryption. GW_INK_{NONCE} is used to protect the communication from replay attack and together with $SDEV_{NONCE}$. It builds the salt value in the KDF function. Optionally, $GW\_INK_{MATERIAL}$ is present when Public Key Cryptography is implemented. Therefore, it stores an X.509 certificate.

At this moment, SDEV and Gateway or Innkeeper calculate symmetric keys, according to the algorithm negotiated before. What however is important to remark is that communicating peers will calculate the following keys:

- DK_1: derived key used to provide data confidentiality
- DK_2: derived key used to provide data authenticity

3.8.5 *SDEV AuthN message*

This message is sent by SDEV that means the negotiation is completed and that the cipher suite is activated. It contains the related MTI code (that is MTI=0x30), the nonce (namely $SDEV_{NONCE\_2}$) to prevent replay attacks, and the encrypted hash of SDEVnonce||GWnonce. It should be encrypted since the negotiation is successfully done. After that the message is sent to GW/Innkeeper, the GW/Innkeeper can decrypt it and check if the received hashes match the calculated hashes.

3.8.6 *GW_INK AuthN message*

This message is sent by GW/Innkeeper that means the negotiation is completed, that the cipher suite is activated. It contains the related MTI code (that is MTI=0x40), the nonce (namely $GW\_INK_{NONCE\_2}$) to prevent replay attacks, and the encrypted hash SDEVnonce||GWnonce. It should be encrypted since the negotiation is successfully done. After that the message is sent to SDEV, the SDEV can decrypt it and check if the received hashes match the calculated hashes. At this point, SDEV and gateway/Innkeeper are authenticated.

3.8.7 Data Confidentiality

The encrypted messages can be exchanged by using HTTP Protocol, including an object (i.e. *JSON* [5] in the HTTP Message Body. If the AEAD algorithm is used, the payload is only encrypted by using DK_1.

$$\mathbf{ENC\_DATA = ENC_{DK1}\ (Data||}\boldsymbol{sequence\_number}\mathbf{)}$$

Otherwise, an HMAC signature must be calculated by using DK_2:

$$\mathbf{ENC\_DATA = ENC_{DK1}\ (Data||}\boldsymbol{sequence\_number}\mathbf{)}$$
$$\mathbf{SIGNATURE = HMAC_{DK2}(ENC\_DATA||}\boldsymbol{sequence\_number}\mathbf{)}$$

3.8.7.1 Key Material Derivation

The session key is derived through the KDF: the one used in the middleware is Password-Based Key Derivation Function 2. This process is typically known as key stretching.

3.8.7.2 PBKDF2

The PBKDF2 key derivation function has five input parameters:

$$\mathbf{DK_1 = PBKDF2(PRF, PSK, SDEV_{NONCE}||GW\_INK_{NONCE},i, dkLen)}$$

where:

- *PRF*: pseudorandom function of two parameters with output length hLen (e.g. a keyed HMAC-SHA-1)
- *PSK*: the master key (or premaster key) from which a derived key is generated
- $SDEV_{NONCE}||GW\_INK_{NONCE}$: cryptographic salt
- *i*: number of iterations desired
- *dkLen*: the desired length of the derived key (it depends by the chosen cipher suite)
- DK_1 is the derived key

If AEAD algorithm is not used, derive another key for sign the data is recommended.

$$\mathbf{DK_2 = PBKDF2(PRF,}\ \textit{firstpart}\mathbf{(PSK/2)||SDEV_{NONCE}||GW\_INK_{NONCE},}$$
$$\mathbf{SDEV_{NONCE}||GW\_INK_{NONCE}, i, dkLen)}$$

where DK_2 is the derived key used to sign the Message Authentication Code.

Please note that at this point of implementation LWSP supports only PBKDF2[3] and AES128 CBC with SHA1 as cypher suite. Other levels of security can be easily integrated in the future.

3.9 Security aspects: Platform

The SSP owners can create users and assigns users' properties (i.e. attributes) in the Local AAM. For a resource under its (SSP) control, the SSP owner defines access policies in the RAP to permit or deny access to resources. The SSP owner can also register Platform Agents, responsible for registering non-symbIoTe platform resources thanks to which they may be visible and accessible for the rest of the symbIoTe system. What's more, Platform Agent handles the access to those resources.

3.9.1 Privacy between SSP Middleware and Third Party IoT Platform

To avoid privacy issue between User/App, SSP Middleware Owner and 3rd Party IoT Platform and the escrow issue, the challenge and response mechanism already defined in L1/L2 is reused.

[3] Calculated with 4 iterations.

For L3 clients, symbIoTe offers a proprietary security payload holder in the SecurityRequest extension which can be implemented by customized clients needed to use by e.g. a hashing algorithm which could concatenate a secret (delivered to the client through a 3rd Party channel, therefore unknown in the symbIoTe ecosystem) and the username and/or clientID along with the operation timestamp available for the 3rd Party RAP plugin to be read from the symbIoTe Authorization token SUB claim.

This way the Platform Agent's authorization extension is able to recreate the hash on its side and verify if they match. The username/clientID and timestamp are delivered as already implemented in the L1/L2 CH-RESP mechanism.

4 Components basic information table

This chapter contains all basic information about symbIoTe system components implemented for the Release 2. For a better reading purpose, the information is presented in tabular style for each component in alphabetical order. The list misses RAP GW and Platform agent because the first is intended to be done with commercially available services and the latter should be in charge of the specific IoT platform owner.

4.1 Administration

Component/service name	Administration
URL of source codes	https://github.com/symbiote-h2020/Administration

4.2 Innkeeper

Component/service name	Innkeeper
URL of source codes	https://github.com/symbiote-h2020/SymbioteSmartSpace

4.3 Local AAM

Component/service name	Local AAM
URL of source codes	https://github.com/symbiote-h2020/SymbioteSmartSpace

4.4 SDEV agent

Component/service name	SDEV agent
URL of source codes	https://github.com/symbiote-h2020/SymbioteSmartSpace

Component/service name	Innkeeper
URL of source codes	https://github.com/symbiote-h2020/LWSPLibrary
Additional information	Generic C++ code to implement the Lightweight Security Protocol

4.5 SSP RAP

Component/service name	SSP RAP
URL of source codes	https://github.com/symbiote-h2020/SymbioteSmartSpace

5 Conclusions

The current document reports the final implementation of symbIoTe software related to L3/4 compliancy. The main outcome of this work is the source code and its documentation, published as an open source project in the GitHub service: https://github.com/symbiote-h2020 [2].

There are three major middleware components for the symbIoTe GW deployment plus an additional service based on the existing solution to address the RAP GW functionality, a component for the administration of the SSP and the external agent firmware for Arduino ESP8266 based devices. For our perspective, addressing a big makers community as Arduino one's is a great added value for the symbIoTe project.

There is also a wiki on the GitHub repository that illustrates the various step to build own gateway and setup the Smart Space; these steps are also reported in the appendix of this document.

6 References

[1] symbIoTe project Deliverable D4.1 - symbIoTe Smart Space Middleware Tools, Protocols and Core Mechanisms; February 2017.

[2] H-2020 symbIoTe Cloud Github Repository; https://github.com/symbioteh2020/SymbioteCloud; accessed on 11/09/2018

[3] symbIoTe project Deliverable D1.4 - Final Report on System Requirements and Architecture; July 2017.

[4] IANA Transport Layer Security (TLS) Extensions; accessed on 11/09/2018: https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xml

[5] RFC-7159 The JavaScript Object Notation (JSON) Data Interchange Format; IETF; March 2014.

[6] RFC-4868 Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec; IETF; May 2007

[7] RFC-5869 HMAC-based Extract-and-Expand Key Derivation Function (HKDF); IETF; May 2010

[8] RFC-8018 PKCS #5: Password-Based Cryptography Specification Version 2.1; IETF; January 2017

7 Definition, acronyms, abbreviations

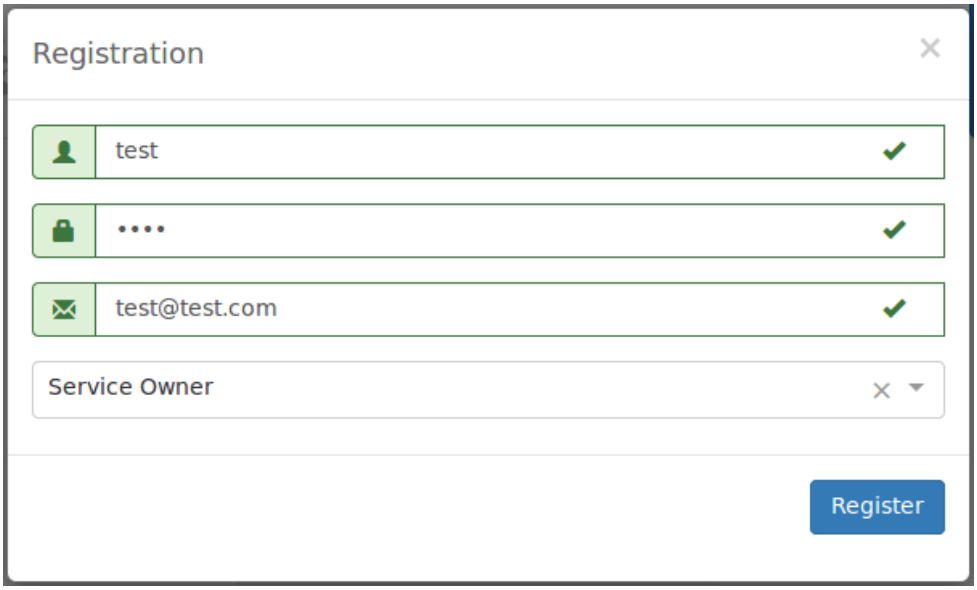
AEAD	Authenticated Encryption with Associated Data
AAM	Authorization and Authentication Manager
ECDH	Elliptic Curve Diffie–Hellman protocol
ECDSA	Elliptic Curve Digital Signature Algorithm
GUI	Graphical User Interface
GW	Gateway
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
ICT	Information and Communications Technology
INK	Innkeeper
IoT	Internet of Things
JSON	JavaScript Object Notation
KDF	Key Derivation Function
LAAM	Local Authentication and Authorization Manager
LWSP	Light Weight Security Protocol
MAC	Media Access Control address
MSK	Master Secret Key
MTI	Message Type Identifier
OData	Open Data protocol
PBKDF	Password Based Key Derivation Function
POPD	Protection of Personal Data
PSK	Pre-Shared Key
RAP	Resource Access Proxy
REST	Representational State Transfer
RSA	Rivest–Shamir–Adleman cryptosystem
S3M	symbIoTe Smart Space Middleware
SAAM	SSP Authentication and Authorization Manager
SDEV	Smart Device
SDEVP	Smart Device encrypted data Payload
SHA	Secure Hash Algorithm
SSP	Smart Space
TLS	Transport Layer Security

8 Appendix - Middleware deployment

8.1 Creating SSP owner

To create a SSP owner user, go to the symbIoTe Core Admin webpage (e.g. https://symbiote-open.man.poznan.pl/administration). During registration, you have to provide:

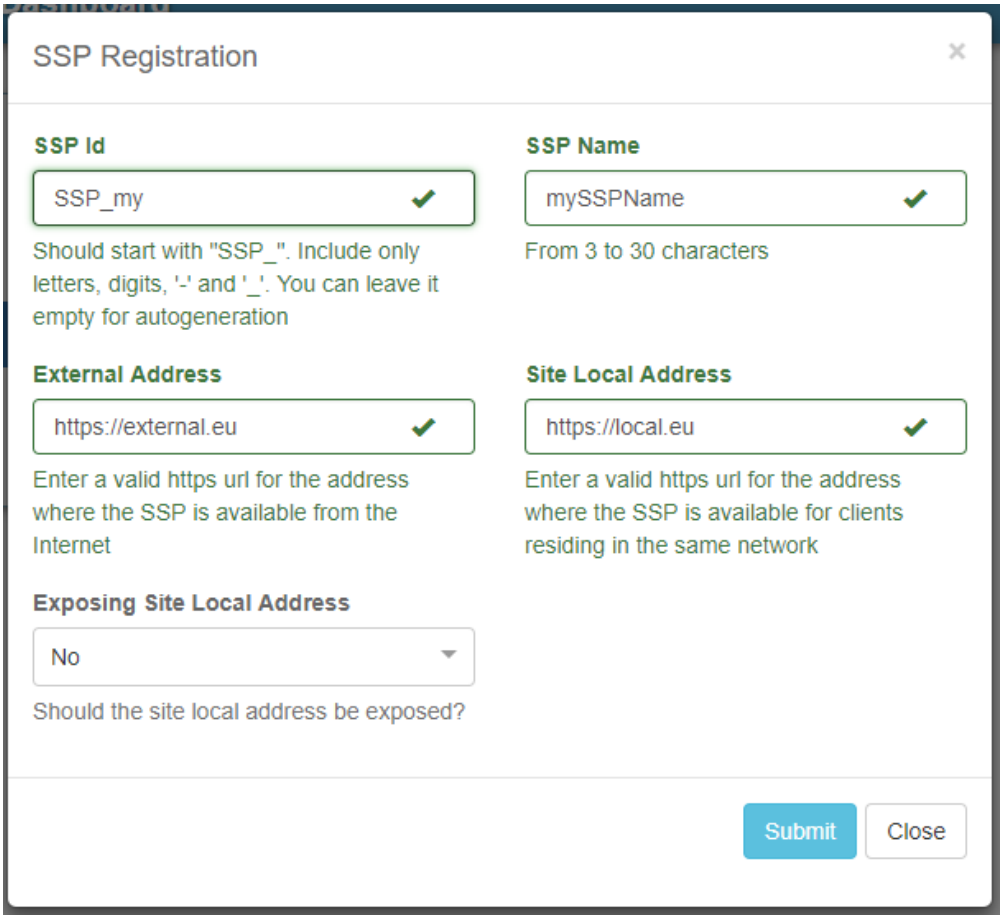
- username
- password
- email
- user role (i.e. Service Owner in this case)



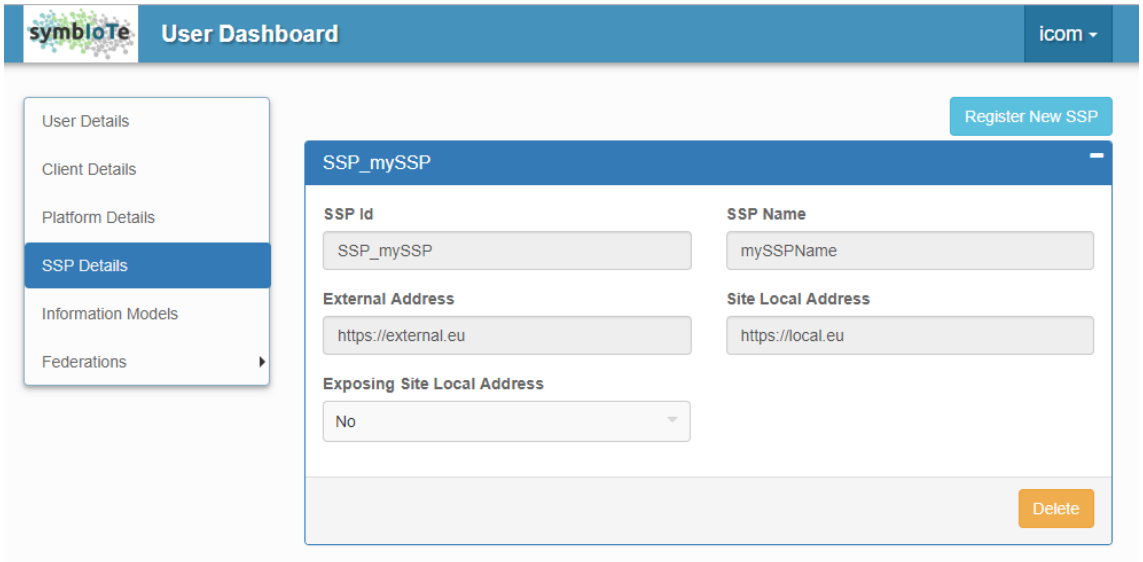
Afterwards, you can log in as the new user and register your smart space. To this end, you have to click on the ***SSP Details*** panel and then on ***Register New SSP*** button on the upper right corner.

Then, you have to provide the following details:

- Preferable SSP id (or leave empty for autogeneration)
- SSP Name
- External Address: a valid https url for the address where the SSP is available from the Internet
- Site Local Address: a valid https url for the address where the SSP is available for clients residing in the same network
- Choose if the site local address should be exposed



By this procedure your SSP is registered in the symbIoTe Core. You will see the panel of the newly registered SSP and check its details by clicking on its header.



Finally, you can ***delete*** the SSP by clicking the delete button on the bottom right corner of the SSP details.

8.2 Installing the requirements

SSP require the following software to be installed:

- Java Development Kit - You need Oracle Java 8 version 8u131+ or OpenJDK version 8u101+ (Letsencrypt certificate compatibility) because all services are implemented in Java.
- MongoDB - (latest stable, verifierd working 3.6.+) database used by cloud components.
- Gradle - (latest stable, verified working 4.6)

8.3 Downloading needed sources

The SSP components are available in github, in the repositories shown in Chapter 1. For a concrete example, let's say that we will install everything in directory ***/opt/symbiote*** on Linux machine.

You can download then using the following commands:

```
$ git clone https://github.com/symbiote-h2020/AuthenticationAuthorizationManager.git
$ git clone https://github.com/symbiote-h2020/SymbioteSmartSpace.git
```

Master branches contain the latest stable symbIoTe release version, develop branch is a general development branch containing newest features that are added during development and particular feature branches are where new features are developed. For symbIoTe smart spaces installation, the following components are currently being used and required to properly deploy a smart space in L3/4compliance:

- ***AuthenticationAuthorizationManager*** (abbr. LAAM or SAAM) - service responsible for providing a common authentication and authorization mechanism for symbIoTe
- ***SymbioteSmartSpace*** - service responsible for storing and searching for metadata as well as provide access to resources

8.4 Configuring and starting components

In this chapter, we describe the procedure to deploy the components required in L3/L4. In this example, we clone our components in the folder ***/opt/symbiote.***

- Start MongoDB server;
- Build and run S3M by using the following commands:

```
gradle assemble --refresh-dependencies
java -jar build/libs/{Component}
```

which need to be done in each directory. They is general remark, the concrete steps can be found in the following paragraphs.

8.4.1 SAAM – SSP Authentication and Authorization Manager

In order to configure SAAM (SSP Authentication and Authorization Manager) we need some symbIoTe certificates in a new keystore. Certificates needs to be created by using SymbIoTeSecuriy.

8.4.1.1 Creating AAM certificate keystore

- Open https://jitpack.io/#symbiote-h2020/SymbIoTeSecurity

At the time of writing this document latest release is e.g. 25.6.0

- Download JAR from link that is release dependent e.g.: https://jitpack.io/com/github/symbiote-h2020/SymbIoTeSecurity/25.6.0/SymbIoTeSecurity-25.6.0-helper.jar

- Download JAR from link: https://www.bouncycastle.org/download/bcprov-jdk15on-159.jar

```
# From CloudConfigPropertiesapplication.properties file:
symbIoTe.core.interface.url
coreAAMAddress=https://symbiote-open.man.poznan.pl/coreInterface

# The user registered through administration in the symbIoTe Core
serviceOwnerUsername=TODO_YOUR_USER_IN_CORE_ADMINISTRATION
serviceOwnerPassword=TODO_YOUR_PASSWORD

# The SSP ID registered to the given service Owner
serviceId=SSP_<TODO_WHAT_YOU_REGISTERED_IN_CORE>

# Generated keystore file name
keyStoreFileName=saam-keystore.p12

# used to access the keystore. MUST NOT be longer than 7 chars
# from spring bootstrap file: aam.security.KEY_STORE_PASSWORD
# Further more as the Java security package is working totally against the API -
ignores the privateKeyPassword.
# IT MUST BE THE SAME as spring bootstrap file: aam.security.PV_KEY_PASSWORD
keyStorePassword=pass123

# platform AAM key/certificate alias... case INSENSITIVE (all lowercase)
# from spring bootstrap file: aam.security.CERTIFICATE_ALIAS
aamCertificateAlias=saam

# root CA certificate alias... case INSENSITIVE (all lowercase)
# from spring bootstrap file:  aam.security.ROOT_CA_CERTIFICATE_ALIAS
rootCACertificateAlias=caam
```

- Start generation of certificate:
  - On Linux/Mac use the following command:

```
java -cp SymbIoTeSecurity-$symbIoTeSecurityVersion-helper.jar:bcprov-jdk15on-
159.jar
eu.h2020.symbiote.security.helpers.ServiceAAMCertificateKeyStoreFactory
cert.properties
```

- On Windows use:

```
java.exe -cp SymbIoTeSecurity-helper-$symbIoTeSecurityVersion-
helper.jar;bcprov-jdk15on-159.jar
eu.h2020.symbiote.security.helpers.ServiceAAMCertificateKeyStoreFactory
.\cert.properties
```

If everything is OK it will generate **paam-keystore.p12** file.

8.4.1.2 Configuring the SAAM component

Build the AAM module using command:

```
$ cd /opt/symbiote/SymbioteSmartSpace/AuthenticationAuthorizationManager
$ gradle assemble --refresh-dependencies
```

Once one has done previous actions, you need to create ***bootstrap.properties*** as in the following example:

```
spring.application.name=AuthenticationAuthorizationManager
spring.cloud.config.enabled=false
eureka.client.enabled=false
spring.zipkin.enabled=false


#port on which the AAM should listen for operations
server.port=8443
aam.database.name=symbiote-aam-database
logging.file=logs/AuthenticationAuthorizationManager.log
# AAM settings
# username and password of the AAM module (of your choice) -- master password
used to manage your AAM (e.g. register new users), not your credentials in the
Core, you need to put matching values in the SSP middleware configuration
aam.deployment.owner.username=sspAdmin
aam.deployment.owner.password=sspAdminP@ssw0rd
# absolute path to the saam-keystore.p12 file
aam.security.KEY_STORE_FILE_NAME=TODO
# name of the root ca certificate entry in the Keystore you produced using the
SymbIoTeSecurity Factory
aam.security.ROOT_CA_CERTIFICATE_ALIAS=caam
# name of the certificate entry in the Keystore you produced using the
SymbIoTeSecurity Factory
aam.security.CERTIFICATE_ALIAS=saam
# symbiotekeystore password
aam.security.KEY_STORE_PASSWORD=pass123
# symbiote certificate private key password
aam.security.PV_KEY_PASSWORD=pass123

# HTTPS only
# name of the keystore containing the letsencrypt (or other) certificate and key
pair for your AAM host's SSL, you need to put it also in your src/main/resources
directory
#server.ssl.key-store=classpath:TODO.p12
# SSL keystore password
#server.ssl.key-store-password=TODO
# SSL certificate private key password
#server.ssl.key-password=TODO
```

```
# http to https redirect
#security.require-ssl=TODO


# Cache settings. If validated token is in cache, component certificate or
available AAMs were aquired recently, value from cache is returned to avoid
communication with another AAM. In case of missing, default values are used.
# time (in milliseconds) for which valid token should be cached (DEFAULT: 60000)
aam.cache.validToken.expireMillis=60000
# size of validToken cache. If size set to -1, validToken cache has no limit.
(DEFAULT: 1000)
aam.cache.validToken.size=1000
# time (in seconds) for which componentCertificate should be cached (DEFAULT:
60)
aam.cache.componentCertificate.expireSeconds=60
# time (in seconds) for which availableAAMs should be cached (DEFAULT: 60)
aam.cache.availableAAMs.expireSeconds=60

#JWT validity time in milliseconds - how long the tokens issued to your users
(apps) are valid... think maybe of an hour, day, week?
aam.deployment.token.validityMillis=60000
# allowing offline validation of foreign tokens by signature trust-chain only.
Useful when foreign tokens are expected to be used along with no internet access
aam.deployment.validation.allow-offline=true

# needed to offer available aams service
symbIoTe.core.interface.url=https://symbiote-open.man.poznan.pl/coreInterface
# needed to expose oneself to other components
symbIoTe.localaam.url=http://localhost
# the external address for client to reach the AAM from the Internet
symbIoTe.interworking.interface.url=https://localhost
symbIoTe.siteLocal.url=http://localhost
# profile activating smart space AAM functionalities (do not change the value!)
spring.profiles.active=smart_space
```

After you have both, the:

- saam-keystore.p
- bootstrap.properties

files ready, then you need to put them in the directory next to the built jar file and run the aam as:

```
$ java -jar AuthenticationAuthorizationManager-3.1. 1 -run.jar
```

8.4.1.3 Verifying functionality of SAAM

Verify all is ok by going to:

http://localhost:8443/get_available_aams

If everything is OK there you should see the connection green and the content are the symbIoTe security endpoints fetched from the core.

8.4.2 SSP Middleware

In order to configure the SSP you need to create an ***application.properties*** file and put it inside the SSP directory. In this file you need to specify the ***SSP id*** and local ***username*** and ***password*** according to the following template:

```
ssp.id=<TODO the id of the SSP as registered in the Administration Panel of the
symbIoTe Core>

# The credentials of the SSP Owner account in the LAAM
symbIoTe.component.username=TODO
symbIoTe.component.password=TODO
```

This is a concrete file for our example:

```
ssp.id=SSP_UNIDATA
# The credentials of the SSP Owner account in the LAAM
symbIoTe.component.username=loc_sspunidata
symbIoTe.component.password=loc_sspunidata123
```

Note:
Username and **password** assigned in **application.properties** file should be ***different*** from the Service Owner credentials in symbIoTe core.

Before starting the middleware, you have to have a *wifi* access point service configured and running. The broadcasted SSID should respect the following syntax (Regex):

```
^sym-[0-9a-f]{20}$
```

While the *psw* associated should be the hex value where each 'f' should be replaced by '9' and each '5' should be replaced by 'a'.

For example, this is a valid symbiotic SSP-wifi:

```
SSID: "sym-00010203040506070809"
psw: "00010203040a06070809"
```

Also, ensure that you have installed a *dhcp server* and a *dns server* to resolve the name "ssp.symbiote.org".

The following are an example of configuration files for the SSP wifi infrastructure using the following software: *hostapd*, *isc-dhcpserver* and *bind9*.

- **hostapd.conf**

```
interface=wlp2s0
hw_mode=g
ssid=sym-00010203040506070809
hw_mode=g
channel=1
wpa=3
wpa_passphrase=00010203040a06070809
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
beacon_int=100
```

```
auth_algs=3
macaddr_acl=0
wmm_enabled=1
eap_reauth_period=360000000
ctrl_interface=/var/run/hostapd
```

- **dhcpd.conf**

```
ddns-update-style none;

log-facility local7;

subnet 10.20.30.0 netmask 255.255.255.0 {
  range 10.20.30.2 10.20.30.40;
  option routers 10.20.30.1;
  option domain-name "symbiote.org";
  option domain-name-servers 10.20.30.1;
  default-lease-time 600;
max-lease-time 7200;
}
```

- **bind9/symbiote.org**

```
$ORIGIN .
$TTL 907200     ; 1 week 3 days 12 hours
symbiote.org           IN SOA  ns.symbiote.org. ns.symbiote.org. (
1263535758 ; serial
10800      ; refresh (3 hours)
                               3600       ; retry (1 hour)
                               604800     ; expire (1 week)
38400      ; minimum (10 hours 40 minutes)
                               )
                        NS      ns.symbiote.org.
                        A       10.20.30.1
                        MX      10 ns.symbiote.org.
$ORIGIN symbiote.org.

Innkeeper                     A       10.20.30.1
ns                            A       10.20.30.1
```

You have to link the **bind9** config file usually named **named.conf.local** with the previously *symbiote.org* zone. E.g. adding this line in the **named.conf.local** file:

```
zone "symbiote.org" IN { type master; file "symbiote.org"; };
```

So this is the result of the file **named.conf.local**:

```
include "/etc/bind/rndc.key";
acl trusted    {
                10.20.30.0/24;
                localhost;
               };
view "trusted-view"
{
match-clients { trusted; };
```

```
zone "symbiote.org"           IN { type master; file "symbiote.org";      };
zone "255.in-addr.arpa"       IN { type master; file "/etc/bind/db.255";  };
 };
```

Then you can use the following command to set-up the network (example if using debian distribution):

```
hostapdhostapd.conf>/dev/null &
ifconfig wlp2s0 10.20.30.1/24
/etc/init.d/isc-dhcp-server start
/etc/init.d/bind9 restart
```

Then, issue the following commands to deploy the SSP:

```
$ cd /opt/symbiote/SymbioteSmartSpace
$ gradle assemble --refresh-dependencies
$ java -jar build/libs/SymbioteSmartSpace-1.0.0.jar
```

Now the Smart Space Middleware should be running.

8.4.2.1 SDEV side configuration

Once the SSP is up and running, you can fire up the SDEV burned with the example firmware[4] from the GitHub Repo. It will connect to the SSP WiFi, establish the secure communication tunnel session using the Lightweight Security Protocol and the begin to register its resources. After that, it handles the RAP request.

[4]The firmware is available for the Arduino ESP8266 platform.

9 Appendix – Component sequence diagrams

This section reports the schematic interaction of the S3M middleware components in 4 type of scenarios:

- SDEV that joins the SSP;
- Local platform that joins the SSP;
- Local access of resources;
- Remote access of resources;

9.1 SDEV joining the SSP

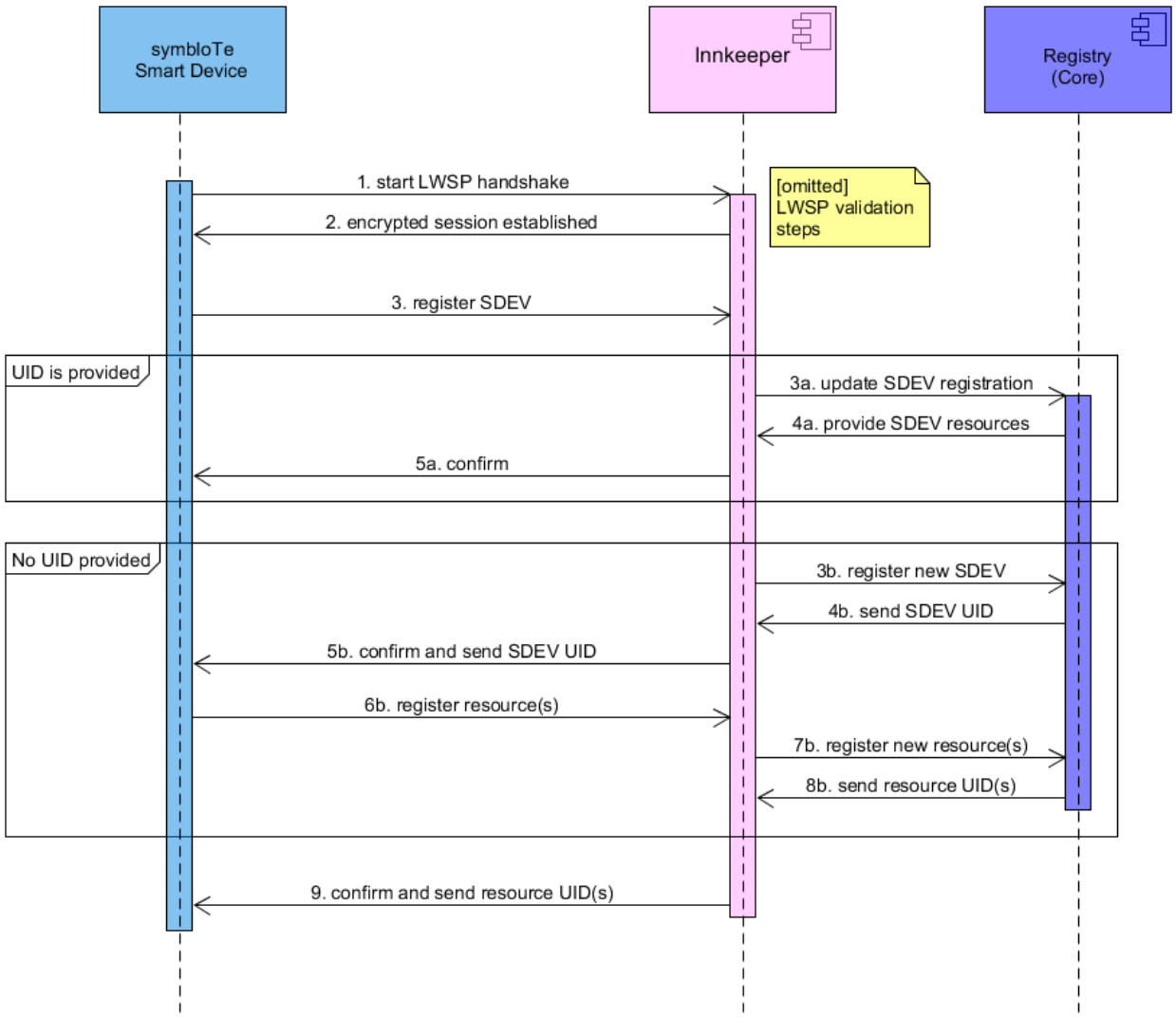


Figure 5: SDEV joins SSP

9.2 Local Platform joining the SSP

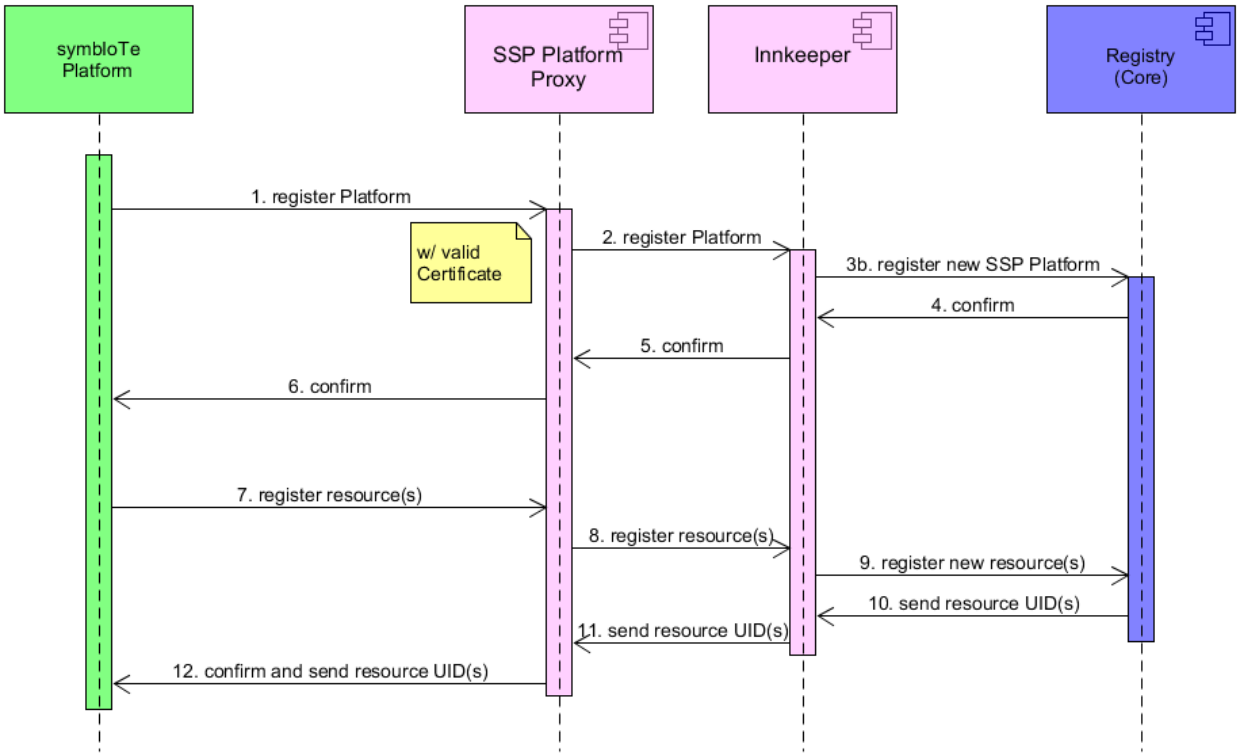


Figure 6: Local platform joins SSP

9.3 Local access of resources

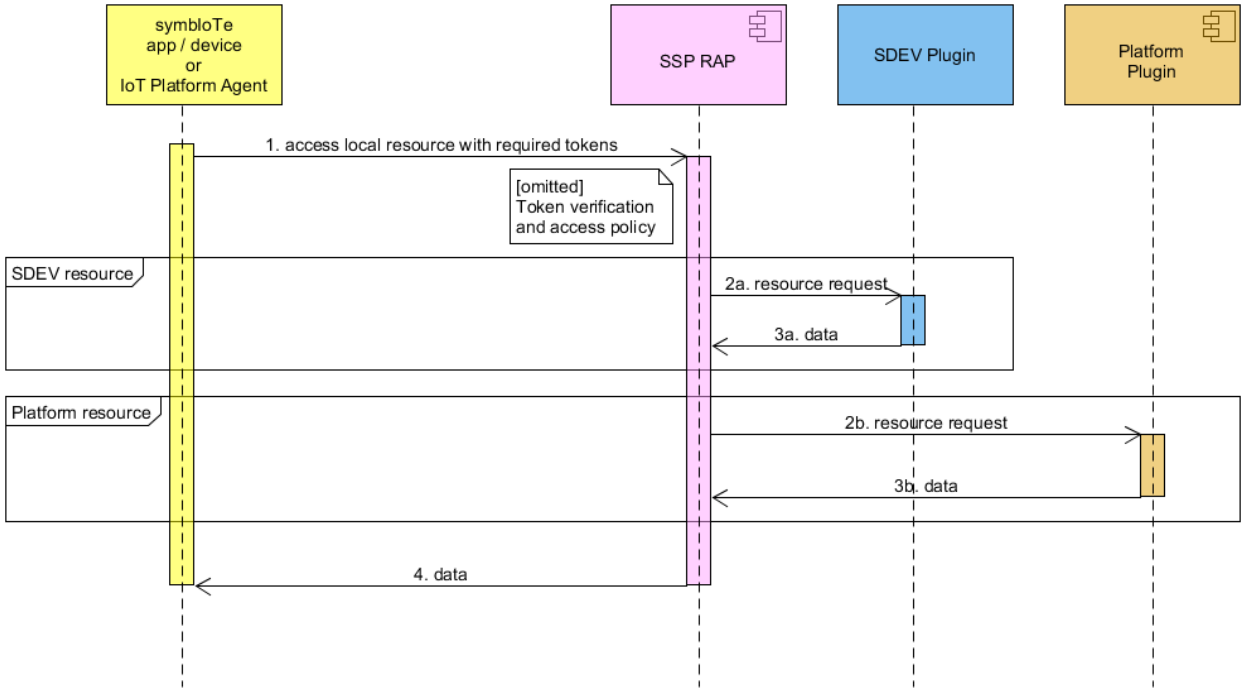


Figure 7: Local access of resources

9.4 Remote access of resources

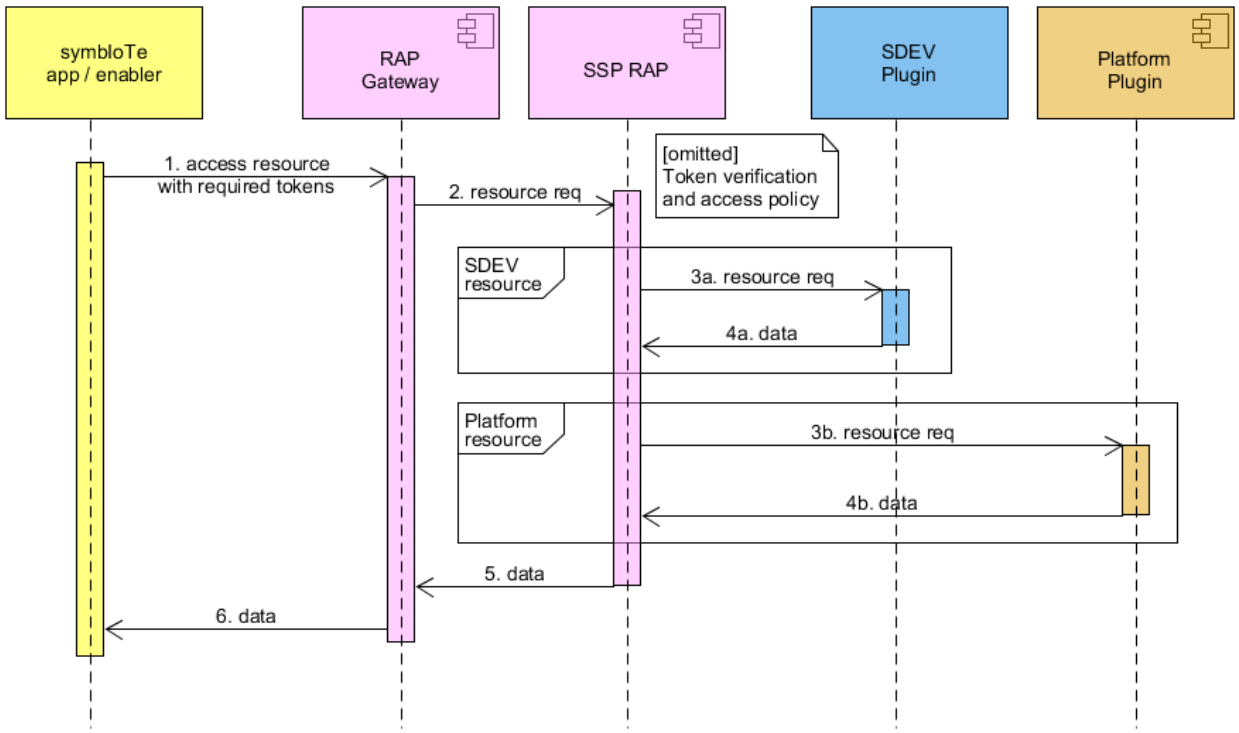


Figure 8: Remote access of resources