SEMIoTICS

Deliverable D4.12
SEMIoTICS Security and Privacy Mechanisms (final)

Deliverable release date	30/04/2020 (revised on 21.04.2021)
Authors	1. Felix Klement, Korbinian Spielvogel, Henrich C. Pöhls (UP) 2. Konstantinos Fysarakis, Manolis Chatzimpyrros, Thodoris Galousis, Michalis Smyrlis (STS), 3. Nikolaos Petroulakis, Emmanouil Michalodimitrakis (FORTH), 4. Łukasz Ciechomski, Micha Rubaj (BS)
Responsible person	Felix Klement, Korbinian Spielvogel, Henrich C. Pöhls (UP)
Reviewed by	Mirko Falchetto (ST), Emmanouil Michalodimitrakis, Nikolaos Petroulakis (FORTH), Konstantinos Fysarakis (STS), Pawel Gawron (BS)
Approved by	PTC Members (Vivek Kulkarni, Nikolaos Petroulakis, Ermin Sakic, Mirko Falchetto, Domenico Presenza, Christos Verikoukis) PCC Members (Vivek Kulkarni, Nikolaos Petroulakis, Christos Verikoukis, Georgios Spanoudakis, Domenico Presenza, Danilo Pau, Joachim Posegga, Darek Dober, Kostas Ramantas, Ulrich Hansen)
Status of the Document	Final
Version	1.0
Dissemination level	Public

Table of Contents

Acronyms Table	
Acronym	**Definition**
DoS	Denial of Service
DPI	Deep Packet Inspection
DNF	Disjunctive Normal Form
EoP	Elevation of Privilege
HTTP	Hyper Text Transfer Protocol
IDM	Identity Management
IIoT	Industrial Internet of Things
IoT	Internet of Things
JSON	JavaScript Object Notation
OAuth2	Open Authorization 2
PAP	Policy Administration Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
PoP	Proof of Possession
QoS	Quality of Service
RA	Robotic Assistant
REST	Representational State Transfer
RR	Robotic Rollator
SARA	Socially Assistive Robotic Solution for Mild Cognitive Impairment or mild Alzheimer's disease
SDN	Software Defined Networking
SFC	Service Function Chaining
SPDI	Security, Privacy, Dependability, and Interoperability
VIM	Virtual Infrastructure Manager
VNF	Virtual Network Function
SM	Security Manager
SSC	SEMIoTICS SDN Controller
WoT	Web of Things
ABE	Attribute Based Encryption

CP-ABE	Ciphertext Policy ABE
KP-ABE	Key Policy ABE
BAN	Body Area Network
GP	General Practitioner
CG	Caregiver

1. INTRODUCTION

This deliverable is the final output of **Task 4.5 – End-to-End Security and Privacy**, and as such, its purpose is to provide the design of the security mechanisms, enabling the satisfaction of relevant Security and Privacy properties, as defined in the context of the projects Security, Privacy, Dependability and Interoperability (SPDI) patterns, towards achieving end-to-end security and privacy properties in an IoT system.

Firstly, this document provides a description of the building blocks that SEMIoTICS uses to ensure security and privacy and how separate modules interact with each other to achieve a consistent and adaptable security and privacy enforcement throughout the different layers of the SEMIoTICS architecture. Figure 0 provides all components from the SEMIoTICS architecture in one aggregated view showing nicely that the components involved in security and privacy span across all layers of the architecture. Namely, we present in this document the identity management, authentication, Software-Defined-Network security, enforcement of policy-based authorisation, and SPDI patterns monitoring and enforcement.

Second, this deliverable evaluates how the provided mechanisms are implemented and how their generic interfaces can be tailored to help IoT and IIoT applications deployed in SEMIoTICS to tackle their very-particular security and privacy-related needs. Namely, we list in Section 5 how the previously gathered requirements for the three main application domains and the generic requirements are achieved with the help of SEMIoTICS' privacy and security components.

Third, this deliverable shows the applicability of the security-relevant mechanisms described in this document achieve the requirements of a particular scenario in the health-care domain, based on one of the project's use cases. By this it demonstrates the applicability of SEMIoTICS approaches to an IoT system and the advantages of our security and privacy-related solutions in a concrete scenario.

Finally, the implementation of these components is also nearly completed by now and integration-tests are underway, thus the components are expected to be used in the applications from our diver scenarios to meet not only the generic, but also help to address the very specific security and privacy requirements of each individual use case.

1.1. Security-and-Privacy-by-Design

As security and privacy cannot be discussed or evaluated without being grounded in a specific enough scenario, this deliverable presents a threat analysis of interactions needed in the applications discussed in a detailed-described scenario from use case 2. The application domain of use case 2 is assisted living for the elderly. In particular, this use case is arguably the most sensitive use case of the project in terms of privacy. Therefore, we have chosen the assisted living scenario to highlight our contributions towards end-to-end security and privacy. We provide the most-important details of security and privacy threat analysis for this scenario in Section 4. For the use case description, we have used information from SEMIoTICS usage scenarios and requirements (deliverable D2.2) and high-level architecture presented in D2.5. This analysis was already presented early in the development of SEMIoTICS. Namely, it was already concluded and presented in the earlier version of this document (D4.5). This helped to guide the design, stated in Section 2, and the development of the security and privacy components (also found in WP 5 (D5.7, D5.10)) as well as the design of SEMIoTICS overall framework and thus was facilitating the privacy-/security-by-design approach.

1.2. Outline

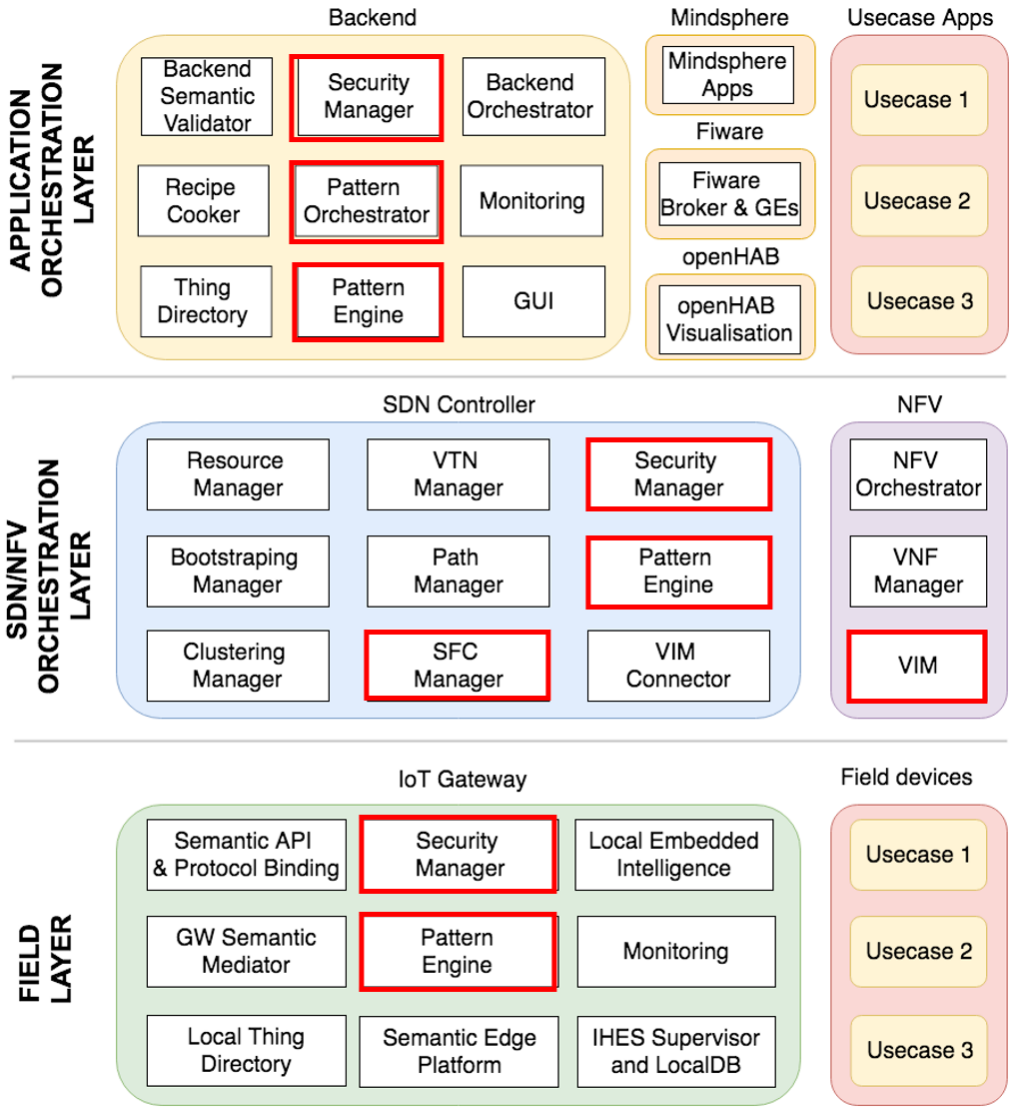
This document is structured as follows. We describe the design of mechanisms enabling end-to-end security and privacy in Section 2. There we also describe the background behind building blocks, like attribute-based encryption and our patterns-approach. Then we continue to use this design and we specifically explain the approaches which are behind the design, the implementation and the deployment of crucial components as part of Section 3:

- the backend Security Manager,
- the concept for a Security Manager's shadow copy for increased reliability and speed at the field layer,
- the role of the Software-defined network and especially the SDN controller,
- SEMIoTICS' approach to place multiple Policy Enforcement Points across SEMIoTICS' architecture and the use of Privacy and Security patterns.

This also highlights how the patterns are playing a central role when it comes to drive the design of the initial access control policies and also allow to monitor the compliance of the system's current policies. It further shows

that SEMIoTICS has adopted the layered IoT architecture by heart as we envisioned to uphold strong authorisation enforcement without scarifying speed and with an increased reliability in mind to cater for delays or short outages of the networked backend services.

Afterwards, we show the interaction across the aforementioned components in the architecture for security and privacy used in SEMIoTICS as part of Section 3. In Section 4, we present SEMIoTICS' incident detection and mitigation mechanisms as well as the procedure which comes into effect when an incident is detected. Last but



OVERVIEW OF THE SECURITY-RELATED COMPONENTS OF THE SEMIOTICS FRAMEWORKOVERVIEW OF THE SECURITY-RELATED SEMIOTICS

not least, we provide a general threat analysis of an example motivating security mechanisms inspired in use case 2 focused on assisted living for the elderly in Section 5. For the use case description, we have used information from SEMIoTICS usage scenarios and requirements (deliverable D2.2) and the first cycle of the high-level architecture presented in D2.5.

We conclude in Section 6 with evaluating our list of requirements gathered in D2.2, D2.3, D2.5. This document thus shows how SEMIoTICS security and privacy-related components described herein directly address the requirements or help addressing them. For specific use-case requirements the devil is in the details and thus we defer the reader to the use case specific deliverables, where possible we re-stated already available results

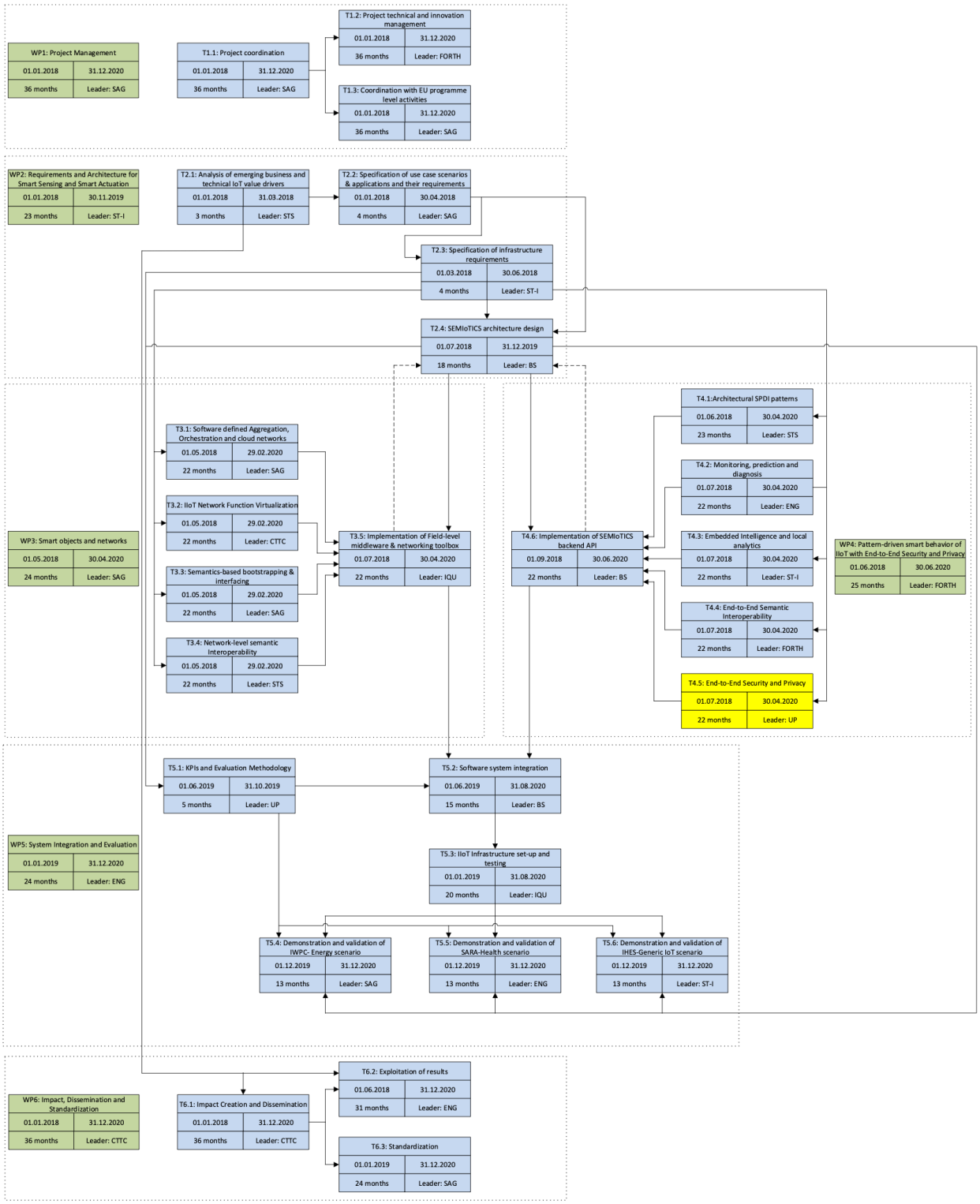
for an increased readability, but not all results are obtainable yet as especially the use-case related requirements can only be evaluated finally once they have been concluded at the end of the project.

1.3. Additions and changes compared to the interim version (deliverable D4.5)

Updated from the previous draft version (D4.5) this final report details herein the full set of SEMIoTICS security and privacy components and how they fit inside SEMIoTICS' architecture given in Section 3. Furthermore, the incident detection and response process has been explained in Section 4.
In addition to minor updates in other places, this document contains the evaluation of the list of requirements as previously gathered in Section 6.2.
Where possible, this final version also provides a direct evaluation, and if not possible, it guides the reader to where the evaluation is provided.

1.4. PERT chart of SEMIoTICS

This section quickly introduces an overview on a per-task / WP basis of the interactions between tasks in SEMIoTICS, with specifically T4.5 depicted in yellow. The PERT chart is kept on task level for better readability.



2. SECURITY DESIGN

The upcoming sections describe the goals of the SEMIoTICS security and privacy components and illustrate their functionality. To this end, we start by describing the goals of the security and privacy mechanisms. Also, we describe the functionality provided using some basic notions from a scenario with roles mapped to the assisted living use case (UC2) from the project. In particular, we highlight differences between a care giver, e.g., a relative taking care of a patient, and a general practitioner, i.e., a medical doctor. These notions will be clearly mapped to the particular security and privacy needs from UC2 in a threat analysis afterwards (D5.10).

2.1. Goals

SEMIoTICS tackles multi-tenant scenarios in a variety of levels, i.e., from the networking layer to the application layer. Therefore, SEMIoTICS requires the means to:

- Provide mechanisms to authenticate users and manage their identities.
- Provide mechanisms to manage identities of other entities, e.g., sensors.
- Support use case applications to enforce access to privacy-sensitive information within the application.
- Support use case applications to enforce access to privacy-sensitive information when the data is stored in a cloud server, e.g., by using attribute-based encryption and lightweight encryption algorithms.
- Provide end to end secure networking capabilities.

2.2. Security Building Blocks

To avoid re-implementing existing functionality, the SEMIoTICS Security Manager (see Figure 17) in the backend is based on an existing framework providing a subset or the functionality needed for the project. SEMIoTICS uses, and further extends, an attribute-based security framework called agile-security[1]. This framework is used to authenticate users, perform identity management and policy evaluations. However, it was initially designed for an IoT gateway; therefore, it was adapted during the lifespan of SEMIoTICS to perform efficiently in the backend, while keeping some modules needed in the IoT gateway, e.g., the Policy Enforcement Point (PEP), lightweight. Also, the framework has a user-centric model; more specifically, it supports only the evaluation of a security policy when a user accesses an entity, e.g., a device. However, SEMIoTICS requires the ability to define access policies that do not necessarily involve a user, for example when an application accesses a device. As a result, the framework must be adapted, and migrated to the latest versions of its runtime framework (Node.JS[2]). Furthermore, the existing security component has been extended to support the attribute-based encryption key management and other mechanisms needed for the decentralization of the access control in SEMIoTICS.
In the following sections, the building blocks needed to realise the security and privacy approach proposed by the project are presented. We start by describing the backend Security Manager and its modules. Specifically, we focus on authentication, identity management, device authentication metadata storage, and key management. In addition, we describe the security needed for the SDN layer, along with other components that can be applied at different layers of the architecture, such as the SPDI patterns, the attribute-based encryption, the PEP deployment, and lightweight encryption mechanisms relevant in IoT/IIoT scenarios.

2.2.1. AUTHENTICATION

To support authentication of users, agile-security is an OAuth2 provider. As such, use case applications or SEMIoTICS components can rely on authentication services offered by the security manager in the backend.
In essence, applications requiring authentication services needs to register as an OAuth2 client with agile-security to be able to redirect users to the authentication endpoint. The former approach helps when users have access to a browser (or a mobile device). Additionally, applications without explicit user interaction, e.g., batch or cron[3]-jobs, can authenticate towards the security manager by providing the client credentials they have, or by user a username and password tuple for a valid user.

[1] Agile-security is a previous result from a Horizon 2020 project called AGILE-IoT, which was funded by the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 688088.
[2] Node.js is an open-source, cross-platform JavaScript run-time environment, www.nodejc.org
[3] https://en.wikipedia.org/wiki/Cron

In addition, agile-security supports plug-ins to delegate authentication to external providers, e.g., Google. Therefore, in case a use case requires a custom integration with other identity provider or a particular protocol, the architecture of this component allows for a simple integration.

2.2.2. ATTRIBUTE-BASED IDENTITY MANAGEMENT

SEMIoTICS has a strong emphasis on bridging the gap with brownfield devices; therefore, the project requires a flexible way to identify entities involved in the platform, e.g., sensors, even though they may use legacy authentication or no authentication at all. In this regard, the Security Manager in the backend will provide a prototypical implementation which lets a security expert to define entities and their attributes. In some cases, the component in charge of handling attributes for entities is called Policy Information Point (PIP), instead of IDentity Management (IDM)

```
{
        "id": "/user",
        "type": "object",
        "properties": {
                "user_name": {
                        "type": "string"
                },
                "auth_type": {
                        "type": "string",
                        "enum": ["local"]
                },
                "password": {
                        "type": "string"
                },
                "role": {
                        "type": "string",
                        "enum": ["admin","doctor","technician"]
                },
                "health_record": {
                        "type": "object",
                        "additionalProperties": true
                }
        },
        "required": ["user_name", "auth_type","password"]
}
```

FIGURE 1 ENTITY SCHEMA FOR USER - EXAMPLE

The first step to define any entity is to list its attributes, and declare which ones are mandatory, as well as list possible values an attribute can take when it is restricted to a particular set of values. To this end, the attribute-based identity management uses JSON schema: a standard to specify which keys must be present in a JSON object, as well as their type.

Figure 1 shows an example describing the expected structure for an entity of type “user”. A user must be an object, with the properties: user_name, auth_type, password, health_record, and role. Most properties must be of type string, except for the health record, which is an object. Furthermore, the authentication type can only

have one value, i.e., local, while the role attribute can take one of the following values: admin, doctor, or technician. Attributes without specific enumeration of values can have any value of the type, e.g., string. In addition to strings, JSON schema also allows the definition of arrays, objects, among other types. However, the current support only covers strings and objects for now. Withal, these two attribute types allow for the definition of arbitrary objects (even with attributes as objects, which are nested). Particularly, a nested object in the previous example is the health record, which can contain any additional properties and values inside.

The identity management we plan to use can support the identification and definition of any entity. For now, we have described users, but with additional configurations, this component can also handle identifiers for gateways, devices, could servers, databases, etc. Considering that SEMIoTICS has a strong emphasis on brownfield devices, we discuss now our plan to collect metadata about authentication capabilities of things registered in the project's Thing Directory.

2.2.3. DEVICE AUTHENTICATION INFORMATION AND IDENTITY MANAGEMENT

From the point of view of the devices used in SEMIoTICS, the gateway will use standard security features when possible. Therefore, the Wind park scenario from UC1 may use security features available for specific industrial devices, e.g., SIMATIC S7 controller. Also, in the SARA use case (UC2), the security from protocols such as Bluetooth, Zigbee and the LwM2M[4] bootstrapping protocol will be used. UC3 will provide authentication during the node devices configuration phase, thus preventing un-authorized configurations to be instantiated in the UC3 field device level. Moreover the privacy is guaranteed by design in UC3 because sensed data are processed locally to each device, and only relevant, specific events are propagated to the higher infrastructure. In UC3 the semantic interoperability is ensured thanks to the adoption of standard JSON protocol over MQTT, thus opening-up the possibility to interact also with the Web of Things protocol used in other SEMIoTICS use cases. The WoT protocol is designed to support interoperability by providing an abstract semantic interface on top of widely adopted transport layers such as HTTP Rest APIs, CoAP, MQTT which also supports authentication and data encryption.

We leverage the opportunity provided by the semantic approach to collect security metadata associated to authentication capabilities of devices registered in the Thing Directory. This allows the use cases in SEMIoTICS, if desired, to evaluate security policies to ensure that only properly authenticated things interact with applications deployed in the architecture. Users contain authentication information reflecting the kind of authentication used for a particular user. In addition to the kind of authentication stored for users, SEMIoTICS can support the registration of security metadata related to the semantic description of devices. For this purpose, SEMIoTICS defines the Thing Directory component in our first cycle of the high-level architecture deliverable D2.4. In this regard, the Web of Things standard includes information about well-established security mechanisms to authenticate things [1].

Currently, the editor's draft for the Web of Things specification contains the following subclasses of the SecurityScheme class of metadata:

- PublicSecurityScheme: Public key asymmetric key security.
- PoPSecurityScheme: Proof-of-Possession (PoP) token authentication following particular token formats, such as JWT (JSON Web Tokens).
- APIKeySecurityScheme: Implies that the way in which the token is built is opaque to the protocol.
- DigestSecurityScheme: Similar to HTTP basic authentication, but has been extended to avoid man-in-the-middle attacks.
- BasicSecurityScheme: uses HTTP basic authentication
- PSKSecurityScheme: Pre-shared key authentication security
- OAuth2SecurityScheme: Supports all the authorization flows for OAuth2 (implicit, password, client, code)
- BearerSecurityScheme: following the bearer format for the token, even though the protocol does not use OAuth2.

[4] http://openmobilealliance.org/RELEASE/LightweightM2M/V1_1-20180612-C/OMA-TS-LightweightM2M_Core-V1_1-20180612-C.pdf

- CertSecurityScheme: uses certificate-based asymmetric keys with X 509 V3 certificates.
- NoSecurityScheme: indicates there is no authentication required

Considering the WoT support in the project, as already described in D3.3 “Bootstrapping and interfacing SEMIoTICS field level devices (first draft), we will leverage the semantic description of devices to register their identities as well as the security information related to their authentication mechanisms in the backend Security Manager. Technically speaking, this could be achieved in two ways: registering the subclass name, or storing the whole metadata required for authentication.

The specificity of the information stored in our Identity Management module within the backend Security Manager depends on the abstraction level required to define security policies on things in an understandable and useful manner. For instance, if we were to register the whole metadata, a thing requiring OAuth2 would contain the attributes: authorizationUrl, tokenUrl, refreshUrl, scopes, and flow. On the contrary, if we decide for a more concise representation, the metadata stored in the Security Manager could just specify that the kind of authentication used for this thing has the class “OAuth2SecurityScheme” explained earlier. Figure 2 shows the attributes of a device in SEMIoTICS based on both approaches. Currently, we are inclined towards a simple representation on the right for two reasons.

On the one hand, the simple representation is more intuitive, as it allows for the definition of a security policy based on a single monolithic attribute, i.e., authenticationInformation in Figure 2. On top, we avoid replicating information across the Thing Directory and the Security manager, which can lead to potential consistency issues.

Detailed Security Metadata	Basic Security Metadata

{
“id”: …",
"type": "device",
"authenticationInformation": {
“authorizationUrl”: “https://auth.server/auth”,
“tokenUrl”: “https://auth.server/token”,
“refreshUrl”: “https://auth.server/ref”,
“flow”: “code”
},
…
}
	{
	“id”: …",
	"type": "device",
	"authenticationInformation": “OAuth2Security”,
	…
	}

FIGURE 2 COMPARISON OF DETAILED AUTHENTICATION INFORMATION

2.2.4. ATTRIBUTE-BASED POLICY FRAMEWORK

Aside from defining which attributes belong to which type of entities, as well as possible restrictions on values they could take, i.e., with the enums presented for the entity's attribute values in Figure 2, the security framework must ensure that access to such attributes is controlled. The latter is of utmost importance to ensure that attributes are reliable information to make security decisions upon. This concept is commonly called *attribute assurance*.

In order to achieve attribute assurance, a system must define an attribute authority for each attribute. Essentially, the user allowed to set an attribute value is the attribute’s authority. For example, if any user can write to another user’s role attribute, and the role is used to make security-relevant decisions, users can easily bypass the security mechanism by upgrading their attribute themselves. We implement attribute authorities by loading a pre-defined configuration in the security manager, which contains read and write policies for attributes for each entity.

2.2.4.1. SECURITY POLICIES

To provide attribute assurance by ensuring that only particular users are able to certain attributes for different entities, the identity management must enforce write policies on attributes. Now, we describe the format for the policies. Later on, we describe how defining policies on entities could also help applications using the SEMIoTICS architecture to implement security validations with the help of the PEP.

2.2.4.1.1. ATTRIBUTE POLICIES

The security policies are defined on Usage Locks as part of UPFROnt: a policy framework[5]. Usage Locks are an extension of the Parametrized Locks [2]. Essentially, locks have a list of blocks, as shown in Figure 3. Each block is associated with a read, or write, action. When a policy is evaluated for an action, e.g., read, it is enough when one of the read blocks allows it. Thus, assuming that a grey block has evaluated to false, and a white block allows the action, the following policy evaluation would allow a read action, but not a write action. In other words, the blocks are evaluated with an OR logical operation.

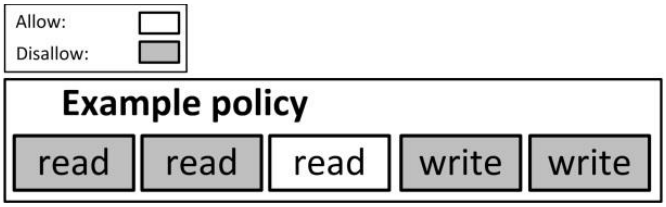


FIGURE 3 POLICY BLOCKS [3]

Each block contains multiple locks. Each lock has a reference to the attributes of entities interacting in the action. Figure 4 shows two read blocks within a policy, where each has three locks. Contrarily to the evaluation of blocks within a policy, locks within a block are evaluated with an AND logical operation. This means that blocks are only evaluated to true, when all the locks allow the operation. Following this reasoning, the block on the left-hand side of Figure 4 allows the operation, while the read block on the right would denies it. Nonetheless, as blocks of the same type are evaluated with an OR, the read action would be allowed in the case of Figure 4, as there is at least one block evaluated to true.

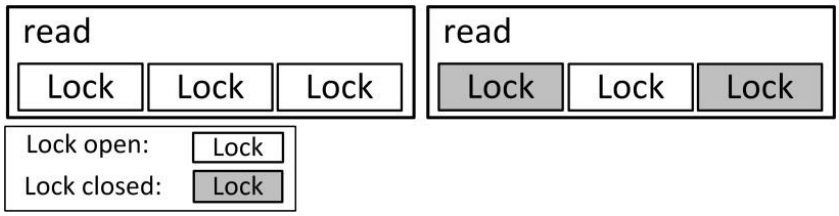


FIGURE 4 LOCK MECHANISM INSIDE A BLOCK [3]

The security policies are defined on Usage Locks, which are an extension of the Parametrized Locks [2], are still under development as part of UPFROnt: a policy framework.

At the moment, the Security Manager inherits the following locks provided by agile-security, although additional locks will be developed during SEMIoTICS, e.g., to support a more generic entity-to-entity security model:

- Attribute equals (attrEq): compare an attribute with a particular value
- Has type (hasType): compare whether an entity has a particular type, such as user or sensor.
- isOwner (isOwner): evaluates whether a user owns an entity

Figure 5 shows the implementation of a policy to protect the role attribute. To map policies to entity attributes, a separate object is created in the Policy Administration Point (PAP) for each entity. The PAP uses this structure to map read and write policies to each attribute; for instance, the role attribute has a counterpart in the object policy for the same user called role too.

[5] https://github.com/SEDARI/UPFROnt

Essentially, the policy from Figure 5 contains two blocks, one for the read and one for the write action. The read block without any locks allows any entity to read the role. On the contrary, the write lock only allows entities of type user, who have role equal to admin. In other words, this policy ensures that users cannot elevate their privileges by updating their role, unless they are already admins.

```
{
        op: "read"
}, {
        op: "write",
        locks: [{
                lock: "hasType",
                args: ["/user"]
        }, {
                lock: "attrEq",
                args: ["role", "admin"]
        }]
}
```

FIGURE 5 USER ROLE POLICY

Figure 6 shows a policy with two read and one write block. The first block ensures that entities of type users with role “doctor” can read the health record. The second read block allows the owner, who must also be a user, to read his health record, i.e., the patient can read his own data. As the read blocks are evaluated with an OR, it is enough that the user attempting to read the data is a doctor or that it is a patient himself. However, as locks are evaluated with an AND, it is not enough that an entity of type user reads the record, because he needs to fulfil all locks within a block. Formally speaking, the policies can be mapped to a logic formula in DNF (Disjunctive Normal Form).

```
{
        op: "read",
        locks: [{
                lock: "hasType",
                args: ["/user"]
        }, {
                lock:: "attrEq",
                args: [
                        "role",
                        "doctor"
                ]
        }]
}, {
        op: "read",
        locks: [{
                lock: "hasType",
                args: ["/user"]
        }, {
                lock: "isOwner",
        }]
}, {
        op: "write",
        locks: [{
                lock: "hasType",
                args: ["/user"]
        }, {
                lock: "isOwner",
        }]
}
```

FIGURE 6 HEALTH RECORD POLICY

2.2.4.1.2. ENTITY-RELATED POLICIES

In addition to the read and write policies enforced by the identity management used by SEMIoTICS, use cases may require custom enforcement of security policies within their applications. To this end, read and write policies can be defined over entity fields that are not necessarily attributes.

To tackle this, the same policy evaluation presented for attributes can be used to evaluate policies for arbitrary actions defined by the use case owner. Mainly, entities have a field in the policy structure that defines policies for actions that can be performed on them. Instead of creating a policy under an attribute name, e.g., role, every entity has a policy object field called “actions”. In this way, the object policies, in the PAP, holds additional policies associated with actions that can be performed on an entity.

For instance, if a use case application needs to ensure that the device owner, e.g., the patient, can see the status of a device, but only users with the role “technician” can change it. The application could configure the security framework to create the policy shown in Figure 7 under “actions.status” for every new device.

```
{
        op: "read",
        locks: [{
                lock: "hasType",
                args: ["/user"]
        }, {
                lock: "isOwner",
        }]
}, {
        op: "write",
        locks: [{
                lock: "hasType",
                args: ["/user"]
        }, {
                lock: "hasAttr",
                args: [“role”,“technician”]
        }]
}
```

FIGURE 7 POLICY FOR ACTION TO UPDATE STATUS ON A DEVICE –EXAMPLE

2.2.5. ATTRIBUTE-BASED ENCRYPTION

This section describes the possible approaches to attribute-based encryption available today. An overview is first provided, and then two ways to enforce security policies are explained, i.e., key policy and ciphertext policy.

2.2.5.1. OVERVIEW

Outsourcing data to cloud environments provides ease of access, provisioning, and cost benefits. On the other hand, the data could be more vulnerable to disclosure. This incomplete control over the data could be offset through encryption [4]. Specifically, traditional symmetric and asymmetric key encryption cryptographic techniques could be used in order to manipulate the encryption. However, these encryption methods offer the privacy, but not the access control. To avoid this problem, the Attribute Based Encryption (ABE) is proposed [5].

The ABE is a relatively recent approach that reconsiders the concept of public-key cryptography. For instance, in common public-key cryptography, a message is encrypted for a specific receiver using the receiver’s public-key. Identity-based cryptography and in particular identity-based encryption (IBE) modified the established structure of public-key cryptography by changing the public-key in an arbitrary string, the email address of the receiver. The ABE goes one-step further and determines the identity not as an individual but as a set of attributes (user roles). Due to that, the encryption of the messages is achieved by using the subsets of attributes –key policy ABE (KP- ABE) – or the policies, which are defined over a set of attributes –ciphertext policy ABE (CP-ABE). In comparison with the IBE, the ABE has meaningful benefit, because it offers flexible one-to-many encryption instead of one-to-one; it is considered as a promising method to manipulate the issue of secure and fine-grained data sharing and decentralized access control [6]. In the following subsections, the two primary forms of ABE will be dealt with in more detail.

2.2.5.2. KEY POLICY ABE (KP- ABE)

KP-ABE, a type of ABE schema, is a cryptosystem for fine-grained sharing of encrypted data [7]. In this method, the ciphertext is defined over the set of attributes and user key is embedded with policy i.e. access structure. A policy for each user is selected by the authority in order to determine which ciphertexts he/she can decrypt. A threshold policy system would be one in which the authority specifies an attribute set for the user, and the user is allowed to decrypt whenever the overlap between this set and the set associated with a particular ciphertext is above a threshold [8]. Figure 8 demonstrates this procedure by a detailed example, in which the data is encrypted using attributes (A, B, C, D) and the user key is embedded with policy: (A AND D) OR C. The user can decrypt the message when his/her credentials satisfy the specific access structure.

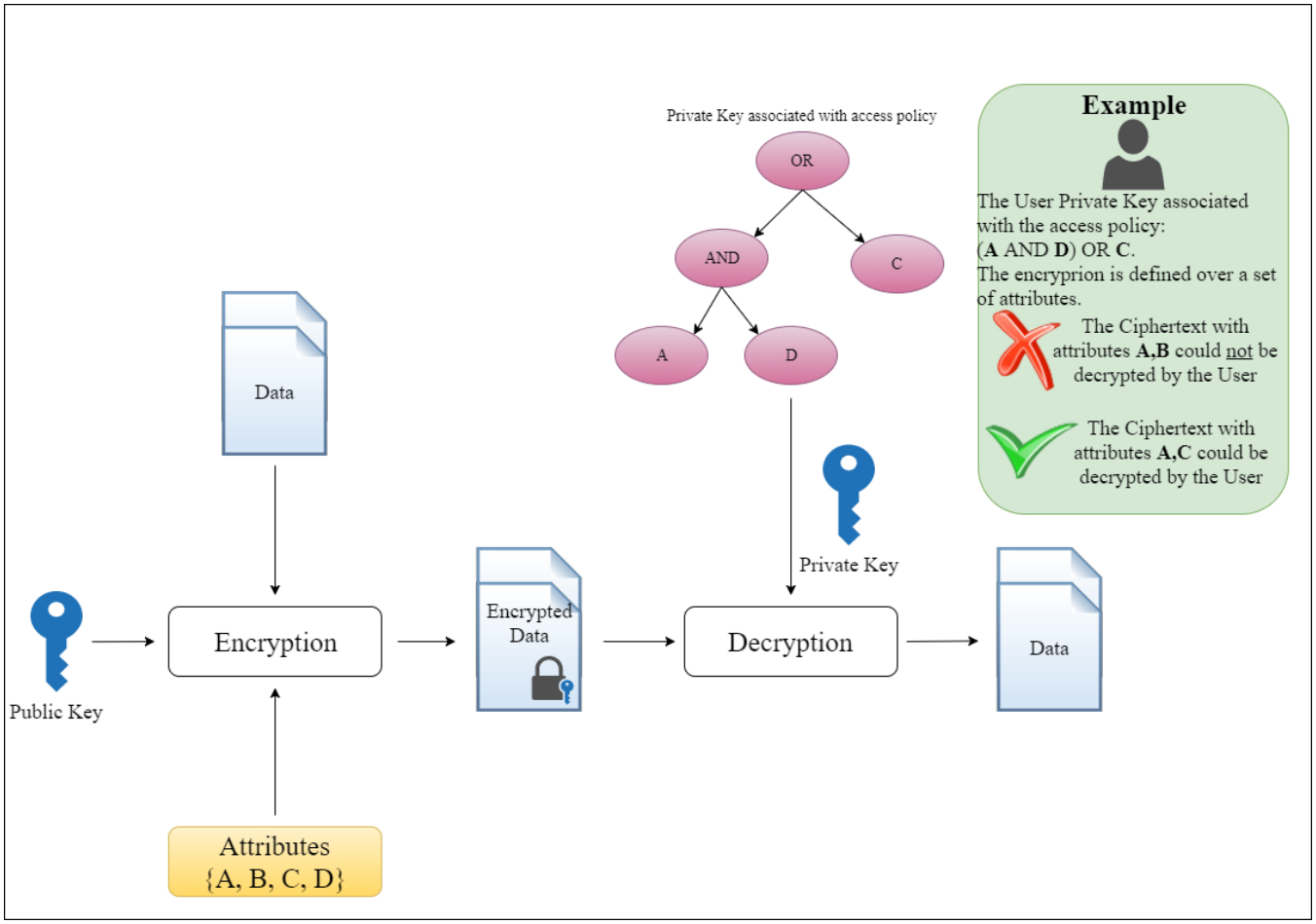


FIGURE 8 KEY POLICY ATTRIBUTE BASED ENCRYPTION –EXAMPLE

2.2.5.3. CIPHERTEXT POLICY ABE (CP-ABE)

In CP-ABE, any user is labelled with a set of attributes and can obtain a private key according to these attributes. The ciphertext is generated under a given access policy. One private key can be used to decipher a specific ciphertext only if the attributes related to this private key satisfy the policy embedded into the ciphertext [9]. A corresponding example is presented in Figure 9. It is evident that the access policy and the attributes are attached to secret keys and ciphertexts of the user in a reverse order in KP-ABE.

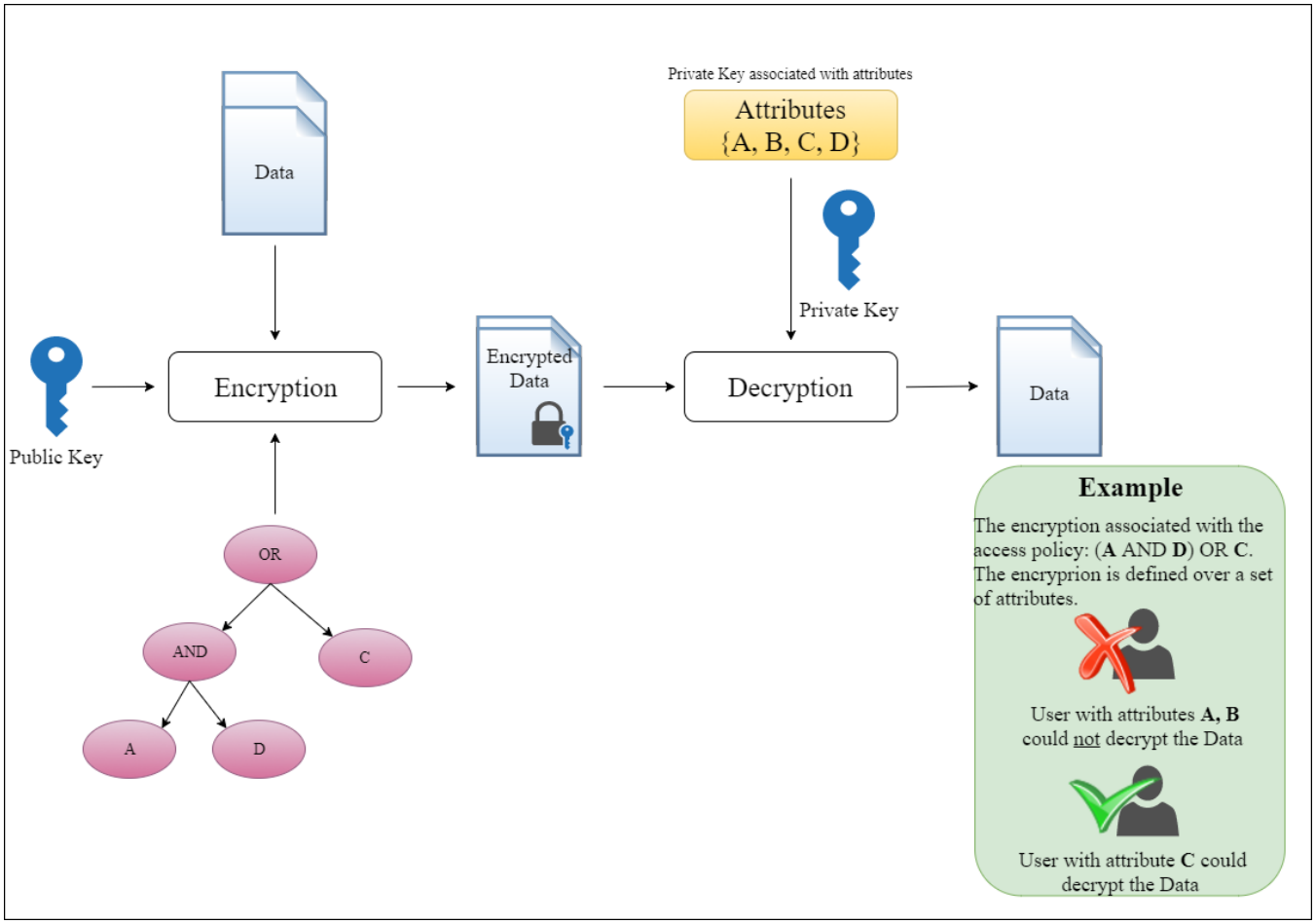


FIGURE 9: CIPHERTEXT POLICY ATTRIBUTE BASED ENCRYPTION – EXAMPLE

Apparently, the encryptor in the KP-ABE is unable to decide who should/ not should have access in the data. As a result, the CP-ABE is more suitable for approaches which aim to achieve flexible access control over sharing data (such as in the environment of cloud computing).

Therefore, in broad terms, KP-ABE gives control over who can decrypt data to the key generator, while CP-ABE ensures that the encryptor (data owner) retains control over who can decrypt her data. In the security analysis, we will discuss possibilities to apply ABE encryption within SEMIoTICS.

2.2.6. KEY MANAGEMENT

In this section, we describe how we can combine the attribute-based identity and policy management presented in Sections 2.2.2 and 2.2.4 with attribute-based encryption presented in Section 2.2.5 for enhanced privacy towards external parties. Particularly, we will now propose two possible mappings for KP-ABE and CP-ABE and entities and policies handled by the security framework.

2.2.6.1. KEY POLICY ABE (KP- ABE)

As every entity has a counterpart object in the Policy Administration Point (PAP), the object can be further extended to include a field called “policies.abe-key” to define a policy applicable for the entity’s private key with particular restrictions.
Essentially, as long as the policy only includes locks validating equality of attributes, i.e., attrEq lock, the security framework can transform the policy defined therein into a tree to generate a key for the entity.

Specifically, the DNF represented by the blocks (combined by an OR) shown in Figure 3, and the locks (combined by an AND) shown in Figure 4 for the policy defined for an entity into a tree of OR and AND attribute

validations, such as the one shown in Figure 8 for KP-ABE. However, in the case of the private key generation for KP-ABE, the security framework would only consider read blocks as relevant. This is a result of the fact that KP-ABE encodes a policy in the private key for the recipient on the data, therefore only the read policy is important. For the sake of clarity, Figure 10 shows an example of how the “abe-key” policy for an entity would be mapped to a tree with OR and AND nodes for the creation of the equivalent KP-ABE key.

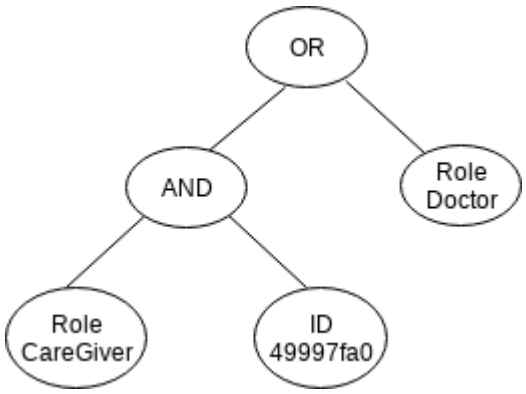
Policy for the ABE key of an entity	Tree Included in the Asymmetric key
{ op: "read", locks: { lock: "hasAttr", args: [“role”,“doctor”] }] }, { op: "read", locks:{ lock: "hasAttr", args: [“role”,“caregiver”] }, { lock: "hasAttr", args: [“identifier”,“49997fa0”] }] }	

FIGURE 10 KEY MANAGEMENT MAPPING BETWEEN ATTRIBUTES AND KP-ABE KEYS

2.2.6.2. CIPHERTEXT POLICY ABE (CP-ABE)

Considering that CP-ABE generates keys based on a user’s attributes, instead of a policy, the security manager can use the attributes of a user to generate his private key. This process is straightforward, as the only information needed is the set of attributes with their respective values. Therefore, this mapping will be naturally performed by using the list of attributes for a particular entity to generate its key.

2.2.6.3. ATTRIBUTE REVOCATION

We will evaluate the feasibility of extending the proposed approaches with an attribute revocation schema. A recent manual documenting a popular ABE library mentions two state-of-the-art possibilities [10]. One possible solution is to add an attribute value associated with a “version” of the key associated with the user’s attribute. Another option is to add a time attribute to the cypher text and validate that the user has an attribute within a particular window of time.

By regenerating keys after a certain period of time, the entity generating keys can ensure that a malicious entity can only use the key until the next cycle of key regeneration. With this approach, systems decrypting information can ensure that keys used are current. However, this generates significant overhead depending on the lifespan of each “version” of a key. For this reason, the project will analyse the specific requirements for the use cases and decide a reasonable trade-off between the security provided by key revocation and the overhead imposed on the system.

2.2.7. SDN SECURITY

Security in SDN is of paramount importance due to their increasing role in the implementation of IoT network involving integrated ICT and physical components and devices. Therefore, a careful investigation of the new security risks that are not relevant to legacy systems must be examined. In addition to the SDN security techniques and in consideration of the criticality of industrial networks, the following principles outline design considerations towards a secure and dependable SDN implementation [11]:

- Dynamic device association will ensure network function continuity and minimize downtime and data loss. Network elements should be able to dynamically associate to a backup controller should the primary controller get compromised or become inaccessible.
- Replication is an essential function for achieving dependability of a system or an entire infrastructure. Replicated instances of the controller as well as application replication will ensure failure tolerance and minimize downtime whether the threat is an attack or a physical disaster.
- Self-healing mechanisms whether proactive or reactive in combination with proper maintenance can provide diversity in the recovery process, thus ensuring enhanced protection towards attacks exploiting targeted vulnerabilities.
- Diversity of controller types (e.g. different OS, hardware) can improve dependability as it is unlikely that a variety of software and hardware combinations will have the same vulnerabilities.

The advancements of SDN regarding security are crucial due to the increasing role in SEMIoTICS supporting the IoT-enabled networks and in the critical industrial-grade infrastructures. Based on this, SEMIoTICS deploys different SDN-enabled mechanisms for enhancing security in the SDN-related scenarios. That includes the development of the three different modules inside the SDN controller able to handle different type of events and procedures.

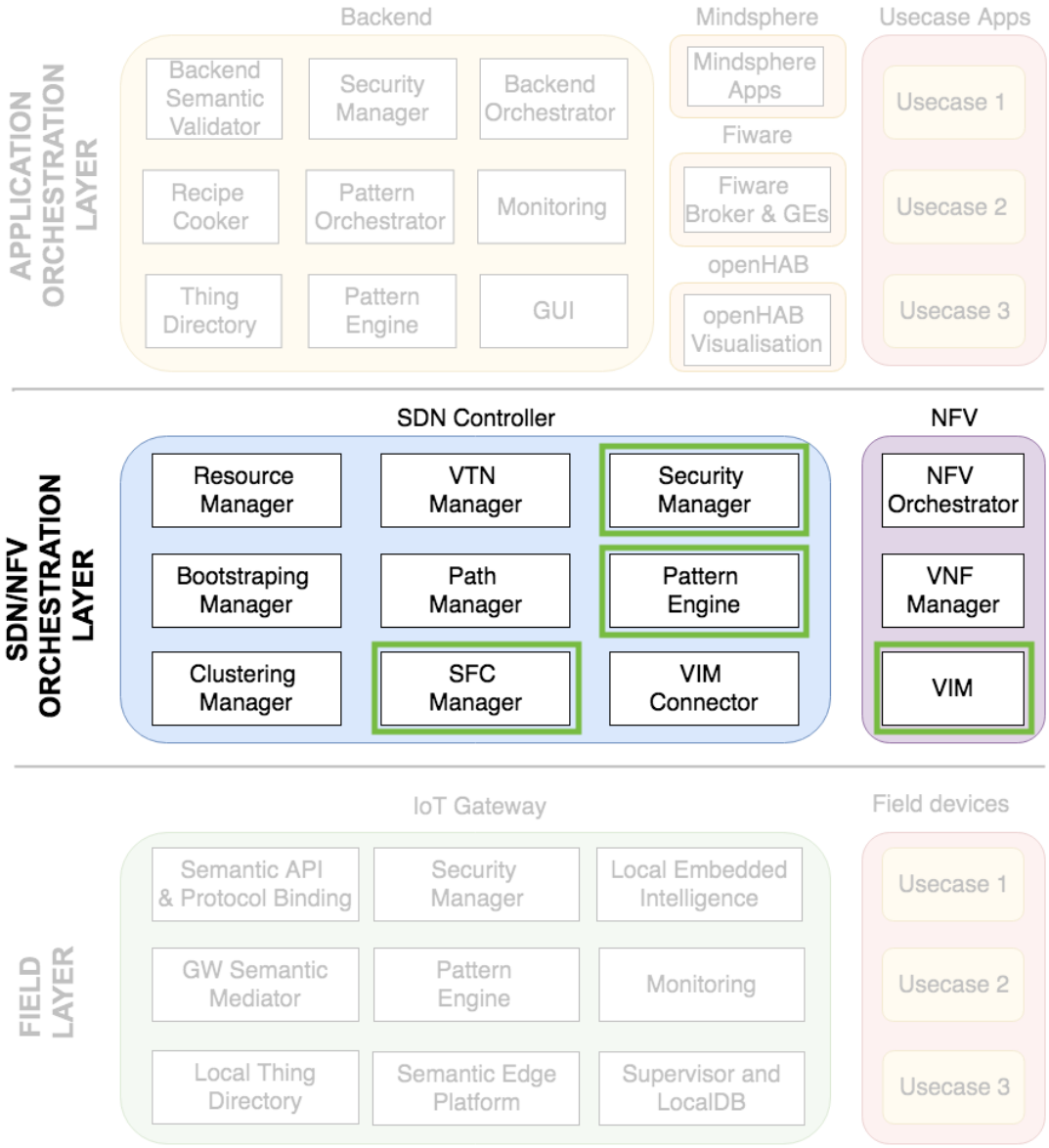


FIGURE 11 SECURITY-RELATED COMPONENTS IN THE SDN CONTROLLER

The respective described security modules in the SEMIoTICS architecture are presented in Figure 11 and detailed below:

- **Pattern Engine** in the SDN Controller is able to ensure SPDI operations of the SEMIoTICS network layer at design and runtime based on a rule engine, which is able to express SPDI patterns as production rules. Enables the capability to insert, modify, execute and retract patterns at design or at runtime in the SDN controller. Enabling reasoning, driven by production rules, appeared to be an efficient way to represent SEMIoTICS patterns. More specifically, since Drools rule engine is based on Maven, it can support the integration of all required dependencies with the ODL controller, as well as the integration of the entities that interact with the controller to run Drools at design and at runtime.
- **Security Manager** in the SDN Controller is responsible for providing authentication and accounting to the entities that interact with the controller. The main role of the Security Manager is the support for authentication and accounting services for administration of tenants and assignment of applications with respective tokens used for fast authentication during runtime. Security Manager should accomplish the authentication and accounting services to the rest of the SDN Controller as well as the users and applications that interact with the controller. Moreover, it exposes interfaces for the administration of

local SDN Controller accounts, in order to achieve authentication. Security Manager provides authentication capabilities based on credentials stored by exposing a method that has local credentials as input and which outputs an authentication token. It also exposes a token validation method that can be used by other controller components to verify that the provided token is valid, and that the bearer of the token is the one who he claims to be. The supported procedures by the Security Manager for the login phase and the authentication phase are presented in Figure 12 and Figure 13.

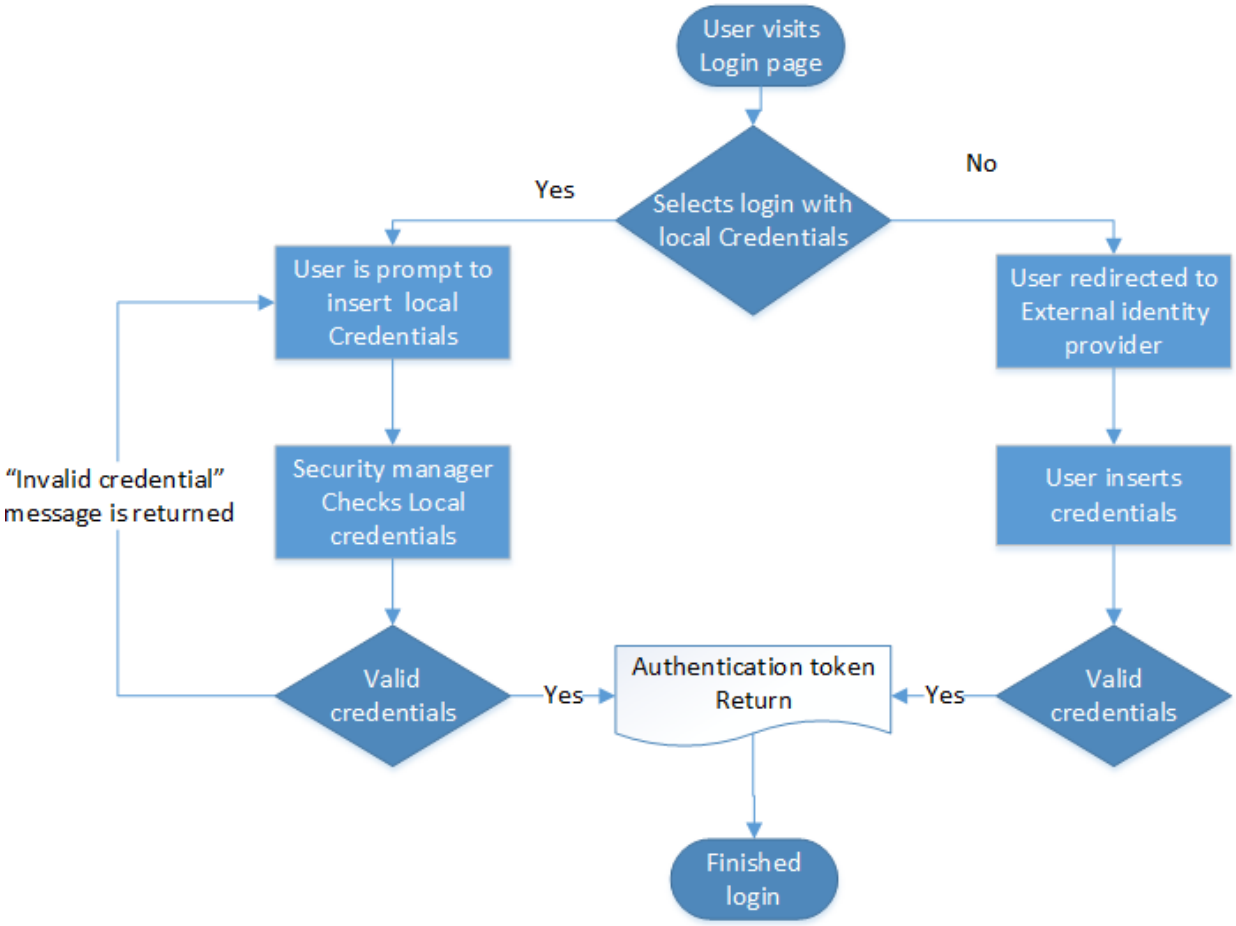


FIGURE 12 LOGIN PHASE WITH THE SECURITY MANAGER

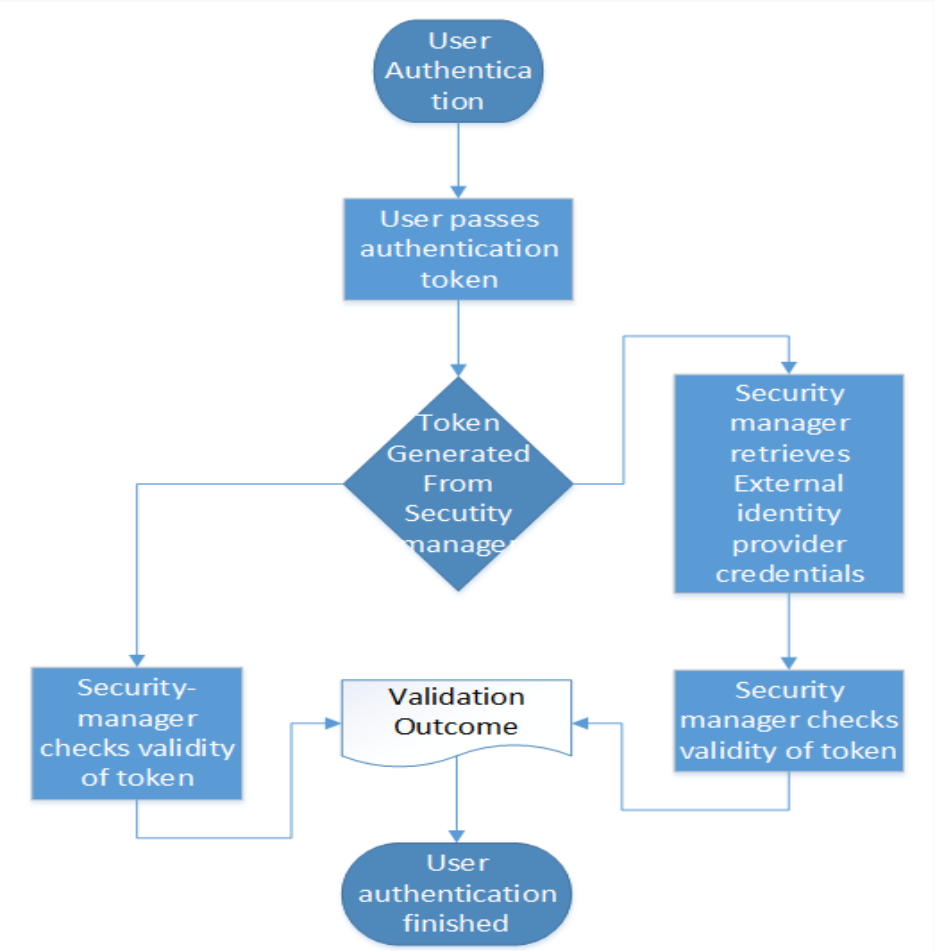


FIGURE 13 AUTHENTICATION PHASE WITH THE SECURITY MANAGER

- **SFC Manager** is responsible to add forwarding rules to network infrastructure. By those additions the traffic can be redirected through the defined components of those service chains. The respective security service network functions that compose those chains are handled by the NFV management and network orchestration (MANO). In the SEMIoTICS cases, service instances in service chains may include Firewall, IDS, DPI, and HoneyPot as described below:
  - A firewall is a network security system that monitors and controls incoming and outgoing network traffic based on predetermined security rules. A firewall typically establishes a barrier between a trusted internal network and untrusted external network, such as the Internet. Firewalls are often categorized as either network firewalls or host-based firewalls. Network firewalls filter traffic between two or more networks and run on network hardware. Host-based firewalls run on host computers and control network traffic in and out of those machines.
  - In Deep Packet Inspection (DPI) packet payloads are matched against a set of predefined patterns. DPI imposes a significant performance overhead, but nevertheless, in one form or another, is part of many network (hardware or software) appliances and middle boxes. As Bremler-Barr et al. [12] have demonstrated, extracting the DPI functionality and providing it as a common service function to various applications (combining and matching DPI patterns from different sources) can result in significant performance gains. In SEMIoTICS proof-of-concept DPI implementation, nDPI [13] can be employed to implement the DPI function, monitor incoming traffic, and assign it to the (sub-)set of security service functions intended for the corresponding traffic type.
  - Network-based honeypots have been widely used to detect attacks and malware. A honeypot is a decoy deployment that can fool attackers into thinking they are hitting a real network

whereas in the same time it is used to collect information about the attacker and attack method. A HoneyNet is a set of functions, emulating a production network deployment, able to attract and detect attacks, acting as a decoy or dummy target.

- Generic Intrusion Detection System (IDS) / Intrusion Prevention System (IPS) is a service able to monitor traffic or system activities for suspicious activities or attack violations, also able to prevent malicious attacks if needed (in the case of IPS).

Services may be the physical appliances or virtual machines running in network function virtualization infrastructures. They may be composed of one or multiple instances. The SFC Manager is responsible for administrating the services chain and mapping the operator's/tenant's/application's requirements into service chains. An expression of the interaction between the SDN controller and the NFV MANO is depicted in Figure 14.

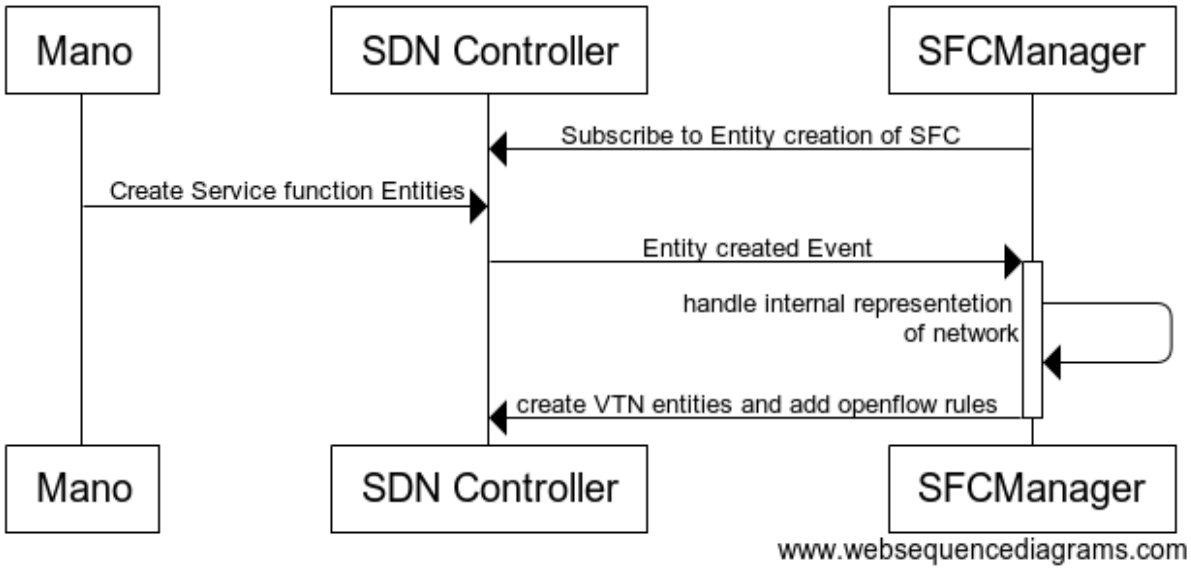


FIGURE 14: SERVICE FUNCTION CHAINING SEQUENCE

A more detailed description of the respective modules is given in D2.5, D3.7, D3.8, D3.10 and D4.8.

2.2.8. MACHINE LEARNING-BASED SECURITY

IoT aims at interconnecting thousands or millions of smart objects/devices in a seamless way by sensing, processing and analysing huge amount of data obtained from heterogeneous IoT devices. This rapid development of IoT-oriented infrastructures comes at the cost of increased security threats through various types of IoT network attacks. As a result, machine learning techniques can be applied in order to detect and even to prevent this kind of attacks. Next, we briefly overview possible machine learning methods that could be exploited within the context of IoT attack detection. The approaches described herein are covered in more depth in D4.9. Therefore, we recommend the reader to go to sections 4.2, 4.3, 4.4 in D4.9 for technical details.

Sparse representation can be used as a diagnosis mechanism for instant IoT botnet attack detection and the minimization of their impacts by immediate isolation of compromised IoT devices located at the edge of the IoT infrastructure. Due to limited computational capabilities which govern the edge IoT devices, it is of paramount importance to provide an algorithmic procedure which uses an amount as small as possible of training and testing data to implement an accurate IoT botnet attack detector. A sparse representation-based novel diagnosis technique is proposed under the SEMIoTICS framework [14], where the fundamental assumption is that there is no prior knowledge of malicious IoT network traffic data during the training procedure. The novelty is twofold. Firstly, a reconstruction error thresholding rule based on sparse representation is employed for IoT botnet attack detection assuming that only a very limited amount of both training and testing data is used to deal with low computational constraints as well as with fast reaction. Secondly, a greedy sparse recovery algorithm, dubbed

as orthogonal matching pursuit, is adopted since it involves tuning of only two hyper-parameters, i.e. the thresholding constant and the sparse representation level.

In many tasks, prediction is dependent on past samples such that, in addition to classifying individual samples, we also need to analyse the sequences of inputs. In such applications, a feed-forward neural network is not applicable since it assumes no dependency between input and output layers. Recurrent neural networks (RNNs) have been developed to address this issue in sequential (e.g., speech or text) or time-series problems (sensor's data) with various lengths. RNN is a deep learning architecture of an artificial neural network where connections between units form a directed circle. Thus, it can be seen as multiple copies of the same network each passing a message to a successor, giving RNN the ability to connect previous information to the current task. The input to an RNN consists of both the current sample and the previous observed sample. This type of neural network architecture has been already introduced within various intrusion detection paradigms.

Generative adversarial networks (GANs) consist of two neural networks, namely the generative and discriminative networks, which work together to produce synthetic and high-quality data. The former network (dubbed as the generator) is in charge of generating new data after it learns the data distribution from a training dataset. The latter network (termed as the discriminator) performs discrimination between real data (coming from training data) and fake input data (coming from the generator). The generative network is optimized to produce input data that is deceiving for the discriminator (i.e., data that the discriminator cannot easily distinguish whether it is fake or real). In other words, the generative network is competing with an adversary discriminative network. This type of algorithm can also be considered as a potential algorithmic procedure towards attack detection.

2.2.9. POLICY ENFORCEMENT

Enforcing policy in multicomponent architecture usually is not the easiest task in the modern IT world especially including the open source or legacy components. In such cases, adaptation of these components for new security requirements can be impossible or require a great amount of work multiplied by the number of components. Sidecar pattern can help in such situations [15].

The main idea behind the pattern is to deploy an additional component/application together with the primary application. The "sidecar" pattern shown in Figure 15 can provide additional supporting features for the primary app without making changes in its code. The primary application and the "sidecar" should share the lifecycle, it means every time we deploy or remove the primary application, the same should happen to the 'sidecar'. The lifecycle sharing requirement makes some deployment and packaging formats more suitable for sidecar pattern than others. One of the well-suited formats for 'sidecar' are containers, which will be used in our backend. Using a 'sidecar' proxy component introduces security-by-design to the application, by configuring primary application APIs to be only accessible from localhost so no other components will be able to access the primary application without communication through sidecar proxy.

In SEMIoTICS, we create a PEP as a separate application and prepare it in a way so it can be deployed as a standalone app next to the primary application but also as a second container in a pod for backend orchestrator. The development of the app started in Cycle 1 and should be finished in Cycle 2. In Cycle 1 the focus was to prepare simple polices for securing HTTP API access for end user or application. For each API call received by PEP, it will communicate with Security Manager to authorize and authenticate the call.

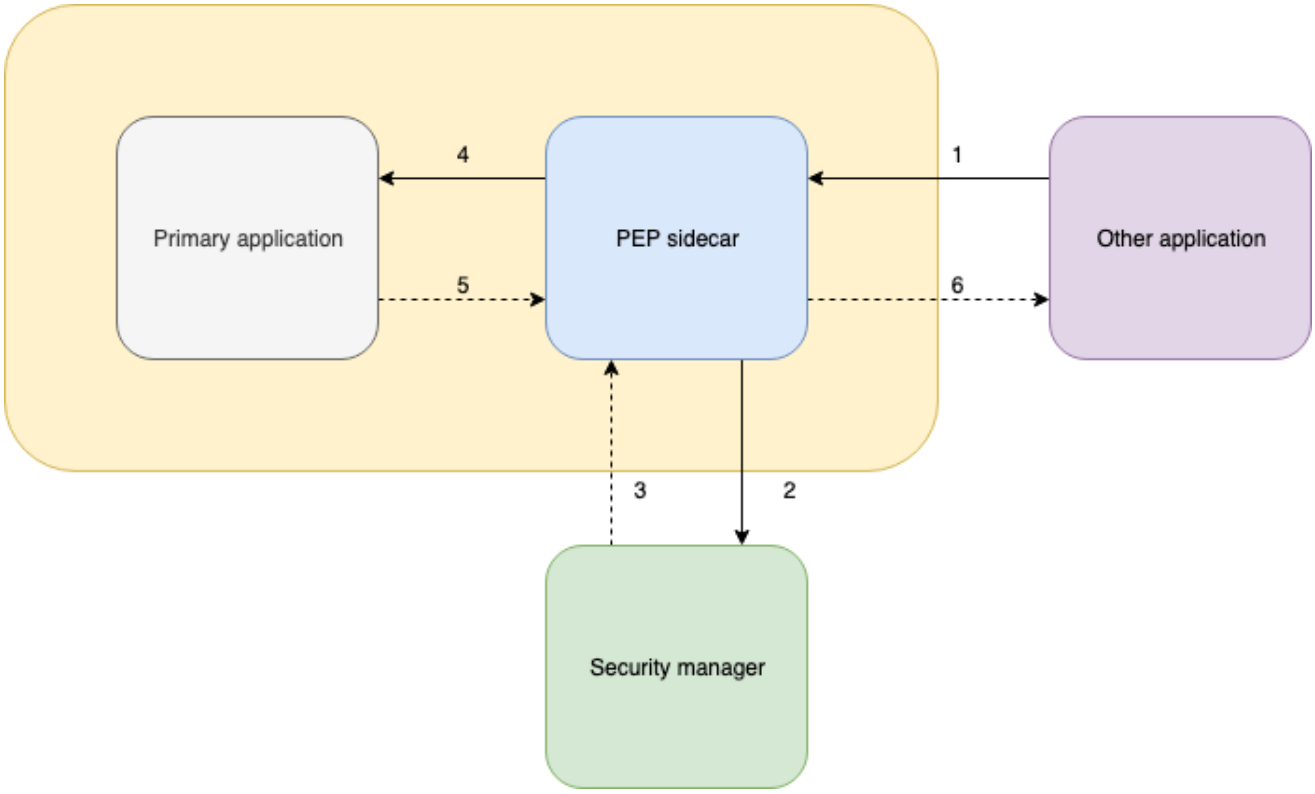


FIGURE 15 COMMUNICATION FLOW USING PEP

2.2.10. LEIGHTWEIGHT CRYPTOGRAPHIC MECHANISMS

Ensuring security in IoT applications that interconnect with a spectrum of smart devices and sensors with varying and low level computational and energy capabilities requires the deployment of lightweight security solutions for these objects.

The mechanisms deployed in the previous subsections, such as the use Attribute-based Encryption and the deployment of the pattern and authorization components at the field layer, in specific, cater for the intricacies of said resource-constrained objects. Nevertheless, some additional lightweight cryptographic primitives may be needed in some scenarios (e.g., to securely store local data on an IoT sensor). In this context, Lightweight cryptography (LWC) investigates the design and integration of cryptographic primitives and algorithms into resource-constrained devices, coming with very low resource requirements and mainly providing confidentiality and data integrity [16] [17].

Block ciphers, the main symmetric key cryptosystems, perform well in this field. Prominent solutions include the industry-standard AES, which can be extremely lightweight, or other standardized block ciphers such as PRESENT and CLEFIA, which are also suitable for lightweight cryptography [18]. Nevertheless, stream ciphers are also relevant in ubiquitous computing applications, as they can be used to secure the communication in applications where the plaintext length is either unknown or continuous, like network streams. In this case, AES-CTR, Enocoro, Salsa20, HC, Acorn, and WG-8 are the recommended safe solutions to be used [19]

2.2.11. SECURITY AND PRIVACY PATTERNS

SEMIoTICS adopts a pattern-driven approach, whereby machine interpretable patterns encode horizontal and vertical ways of composing parts of end-to-end IoT applications that can evidently guarantee SPDI properties. In more detail, architectural patterns in SEMIoTICS support: the composition structure of the IoT applications and platform components; the end-to-end SPDI properties guaranteed by the pattern; the smart object/component/activity level SPDI properties required for the end-to-end SPDI properties to hold; conditions

about pattern components that need to be monitored at runtime to ensure; end-to-end SPDI properties; and ways of adapting and/or replacing individual IoT application smart objects/components that instantiate the pattern if it becomes necessary at runtime (e.g., when some components stop satisfying the security properties).

Note, the patterns influence the applied mechanism. So if the security goal of a pattern requires the protection of confidentiality, the SEMIoTICS framework needs to offer a cryptographically sound and secure mechanisms which provides the required confidentiality, i.e. by Attribute-based Encryption confidentiality at rest and during transit can be protected. Thus, for very crucial patterns SEMIoTICS presents components which implemented several security mechanisms that provide protection towards the pattern-induced security goals of user authentication, data confidentiality and data integrity.
Hence, SEMIoTICS is able to fulfil SPDI patterns that have been set-forward as appropriate for the use-cases to address their individual requirements by the domain experts that have chosen existing patterns or written specific ones. By this approach the SEMIoTICS's framework stay inherently flexible: Patterns can be chosen from existing ones or newly designed to fit the SPDI requirements of the use-case at hand and their fulfilment towards security and privacy by help of using cryptographic as well as network security & privacy mechanisms is described within this deliverable.

While this pattern-driven approach is presented in detail within deliverable D4.8 ("SEMIoTICS SPDI Patterns (final)"), along with a first set of SPDI patterns, some key aspects are briefly re-iterated here for completeness.

2.2.11.1. SPDI PATTERNS DEFINITION

Out of the set of SPDI properties covered within SEMIoTICS, in the context of Task 4.5 the focus is mostly on Security (broken down to Confidentiality, Integrity and Availability) and Privacy. Dependability is, in some cases, linked with Availability (i.e. part of security), but in the context of SEMIoTICS it mainly refers to reliability, fault tolerance and safety aspects.

In all cases, the definition of the pattern language itself is driven by the IoT Orchestrations model presented in D4.1. Based on that, the SEMIoTICS pattern language is derived, which is used to define the IoT components and their orchestrations, along with their desired SPDI properties. Per the SEMIoTICS IoT Orchestration model mentioned above, Placeholders of different types (including IoT components and their orchestrations) may also be characterised by their SPDI properties. Additionally a property of a placeholder can be required or confirmed. A required property is a property that a placeholder must hold in order to be included (considered for) the orchestration. For example, if the required property of an orchestration defining a secure logging process is Confidentiality, then all placeholder activities involved in the orchestration and the links between them may be required to have the Confidentiality property. On the other hand, a confirmed property is a property that is verified at runtime, through a specific means. Said means of verification can include monitoring, testing, a certificate or a specific pattern rule. This means that the existence of a monitoring service or a testing tool allows the verification of the SPDI property of a placeholder activity. Such a monitoring service could, for example, justify that a service or a device is available at specific time windows if the desirable property is a specific target for availability. Another way of verifying SPDI properties could be a repository with certificates that are able to justify that a certain placeholder satisfies a certain property. In case of a pattern the Mean of verification is the pattern itself; in all the other cases we need an interface to a corresponding monitoring tool, testing service or certificate repository through which the verification can take place.

At deployment phase, instantiated versions of the abovementioned system model are transformed into Drools[6] rules and facts, to allow for machine-processable pattern verification and automated system adaptation. Drools business production rules, and the associated rule engine, apply and extend the Rete algorithm [20], which is an efficient pattern-matching algorithm known to scale well for large numbers of rules and data sets of facts, thus allowing for an efficient implementation of the pattern-based reasoning process.

Moreover, to assist in the definition of the orchestrations and the desired properties, this pattern approach is also integrated with Recipes [21] [22]. The user defines her desired IoT orchestrations and properties through Recipes, using the provided high level abstractions, and these are then transformed into SEMIoTICS patterns, which, in turn, get transformed into Drools rules and facts. Reasoning on the SPDI properties included in the

[6] https://docs.jboss.org/drools/release/7.15.0.Final/drools-docs/html_single/index.html

defined orchestrations happens through the deployment of pattern engines on all layers of the architecture, as is detailed in the next subsection.

2.2.11.2. PATTERN COMPONENTS

As mentioned, pattern-related components are present in all layers of the SEMIoTICS framework (see Figure 16). These allow the verification of SPDI properties and triggering of adaptations throughout the IoT deployment, also enabling the semi-autonomous operation (e.g., SPDI related reasoning and adaptation at the network or field layer, even if the backend is not available/offline).

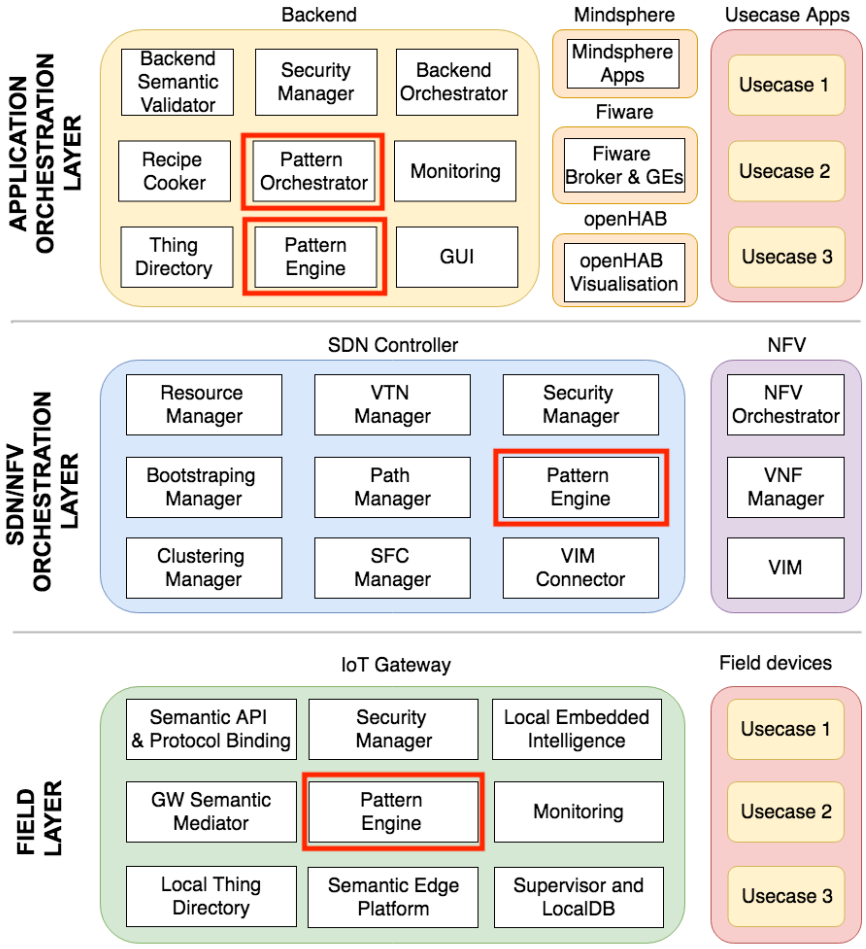


FIGURE 16. PATTERN MODULES WITHIN THE SEMIOTICS ARCHITECTURE

In more detail, the pattern-related components include:

- **(Backend) Pattern Orchestrator:** Module featuring an underlying semantic reasoner able to understand instantiated Recipes, as received from the Recipe Cooker module and transform them into composition structures (orchestrations) to be used by architectural patterns to guarantee the required properties. The Pattern Orchestrator is then responsible to pass said patterns to the corresponding Pattern Engines (as defined in the Backend, Network and Field layers), selecting for each of them the subset of these that refer to components under their control (e.g. passing Network-specific patterns to the Pattern Engines present in the SDN controller).
- **Backend Pattern Engine:** Features the pattern engine for the SEMIoTICS backend, along with associated subcomponents (knowledge base, reasoning engine). It enables the capability to insert, modify, execute and retract patterns at design or at runtime in the backend; these interactions will happen through the interfacing with the Pattern Orchestrator (see above), as well as other additional interfaces are introduced to allow for more flexible deployment and adjustments when needed. The Pattern Engines is also able to reason on the SPDI properties of aspects pertaining to the operation of

the SEMIoTICS backend. Moreover, at runtime the backend Pattern Engine may receive fact updates from the individual Pattern Engines present at the lower layers (Network & Field), allowing it to have an up-to-date view of the SPDI state of said layers and the corresponding components.

- **Network Pattern Engine:** Integrated in the SDN controller to enable the capability to insert, modify, execute and retract network-level patterns at design or at runtime. It is supported by the integration of all required dependencies within the network controller, as well as the interfaces allowing entities that interact with the controller to be managed based on SPDI patterns at design and at runtime. It features different subcomponents as required by the rule engine, such as the knowledge base, the core engine and the compiler.
- **Field Layer Pattern Engine:** Typically deployed on the IoT/IIoT gateway, able to host design patterns as provided by the Pattern Orchestrator. Since the compute capabilities of the gateway can be limited, the module is a lightweight version of the Backend Pattern Engine. Patterns are preinstalled in the module but can also be enriched during runtime by the Pattern Orchestrator. These patterns are able to guarantee SPDI properties locally based multiple factors. Some of them include the data retrieved and processed by the monitoring module, the thing directory in the IoT gateway, and the interaction with other components in the field layer.

For more details on these components, we defer the reader to the corresponding implementation deliverables (D3.11 and D4.6, D4.7 as well as D4.13).

2.2.11.2.1. PATTERN INTERACTIONS WITH SECURITY AND PRIVACY MECHANISMS

The concepts highlighted above, and most importantly specifically the need for automated verification of security and privacy properties as well as the triggering of required adaptations, necessitate the interaction of pattern components with the security and privacy mechanisms that may be deployed within the IoT environment.

In the case of property verification, important means of verification include, as mentioned, monitoring and testing; in some cases, this may entail monitoring the operation of specific security mechanisms (e.g., monitoring if requests go through access control mechanisms, if encryption is enforced) or their testing (e.g., capturing traffic to test that encryption mechanisms are operational).

Moreover, pattern-driven adaptations can also involve triggering changes in the operation and/or configuration of security mechanisms. Examples of these may include changes in the access control policies or triggering of encryption enforcements when certain property changes are needed (e.g., to increase security, reverting it to the desired state).

Some examples of the above are detailed in the subsections below, while the reader may refer to D4.8 where the final set of SEMIoTICS patterns, sketching the supported interactions with security mechanisms, are detailed.

2.2.11.2.2. INTERACTION SCENARIO – INTEGRITY

As an example, let us consider an integrity pattern which refers to the maintenance and assurance of the accuracy and consistency of data. Following the formal process detailed in D4.1 and D4.8, we can define a generic pattern for integrity at data at rest as the following:

$$\mathrm{Hash}(D^x(i)) = \mathrm{Hash}(D^x(i-1))$$

A high-level interpretation of the above is that whenever we check data at rest, those data must not be changed. More formally, this reflects that the cryptographic hash, i.e., Hash function, applied on the data at a given time I, i.e., $D^x(i)$, matches the value observed for the data previously, i.e., for time i-1.

This property is very important in various parts of a SEMIoTICS deployment, and one key area is the preservation of the integrity of the stored authorization policies, in the policy repository. Undoubtedly, a malicious user who tampers with the stored policies is able to invalidate all security provisions that the policy framework is supposed to provide. Therefore, it is important to be able to verify the integrity of said policies. This could be achieved, for example, through a mechanism that checks the integrity of the policies using hashing

mechanisms which, at predefined intervals, checks the hash of the active set of policies with the hash value of the initial set of policies defined at deployment. In line with the pattern defined above, and assuming that an agent is deployed or the policy repository code is decorated to perform this hash check, the result of this could be monitored by the corresponding pattern engine (backend pattern engine, if we assume the policy repository is deployed at the backend) in order to verify that the desired integrity property of the stored policies holds.

Therefore, in the above scenario, the pattern components monitor the operation and interact with either a specific security agent or the policy framework itself, depending on the exact deployment, giving the system owner an up-to-date view of the security status of the system (integrity of authorization policies, more specifically, in this case). Potential adaptation in case of integrity violation could involve trigger a reloading of the policies from a safe copy; e.g., retrieving a copy stored offline (requiring human intervention), or in an automated manner by replacing the policy repository with another instance of the component that is not compromised (e.g., spawning a new container with said component, in case of a virtualized backend infrastructure, as the one adopted in the SEMIoTICS backend).

2.2.11.3. INTERACTION SCENARIO – PRIVACY

There are some cases where the SPDI properties can transition to an initially undesired state due to special conditions that could emerge as the system operates and the context changes.

To give an example, let us consider a scenario inspired by the second use case of the project UC2, focusing around the ambient assisted living environment. It is expected that one of the key Privacy requirements will be to ensure that the patients' location remains protected, since location is private-sensitive information, but it could potentially be accessed from the user's mobile phone that is used to relay information at the backend.

Therefore, to alleviate such concerns, adequate protection mechanisms have to be put in place to guarantee such information cannot be leaked. One approach would be to prohibit such action (i.e. retrieval of patient's location) from caregivers and other involved actors, through the policy/authorization framework. In any case, these mechanisms will have to be interfaced/monitored by the Pattern reasoning engine, to ensure that we have a real-time verification that these privacy properties hold, leveraging the corresponding pattern rules.

Nevertheless, we can foresee extreme cases where the location must be retrieved: consider the case that a patient fall is detected; in such a case, it is imperative to retrieve the person's exact location, to allow for prompt intervention of caregivers or emergency services. Therefore, the policy framework should cater for the change in the permissions, e.g., by including such context parameter in the policies, thus allowing in such cases the caregivers to retrieve the patient's exact location. In such a case, even though this is a desired change, the case remains that the desired privacy property is no longer satisfied, since the patient's location is exposed to caregivers / emergency services. By monitoring the operation of the policy framework, such change in the privacy property will be visible at the backend, allowing SEMIoTICS operators to the change in real-time and also verify if this is indeed a needed/programmed violation of the Privacy property or a malicious or unexpected system behaviour.

3. ARCHITECTURE OF THE SEMIOTICS SECURITY COMPONENTS

This section details the high-level overview given in D4.5 and describes security components that all together comprise SEMIoTICS' security architecture, amongst them are components concerned with the central management of identities in the Security Manager. The Security Manager, as well as the other components, have been grouped – for ease of viewing – by their position within SEMIoTICS' 3-tier architecture (backend, network, field). An overview is given in the following section, followed by individual sections that take a closer look on how the different components are embedded and used to achieve the required security and privacy functionality of SEMIoTICS. The description also briefly explains their very specific placement in each layer of the architecture or how they are added to the applications for a use-case.

3.1. Architectural Overview

In this section, we cover from a high level the component interactions between the components required to fulfil the end–to-end security and privacy objectives of SEMIoTICS.

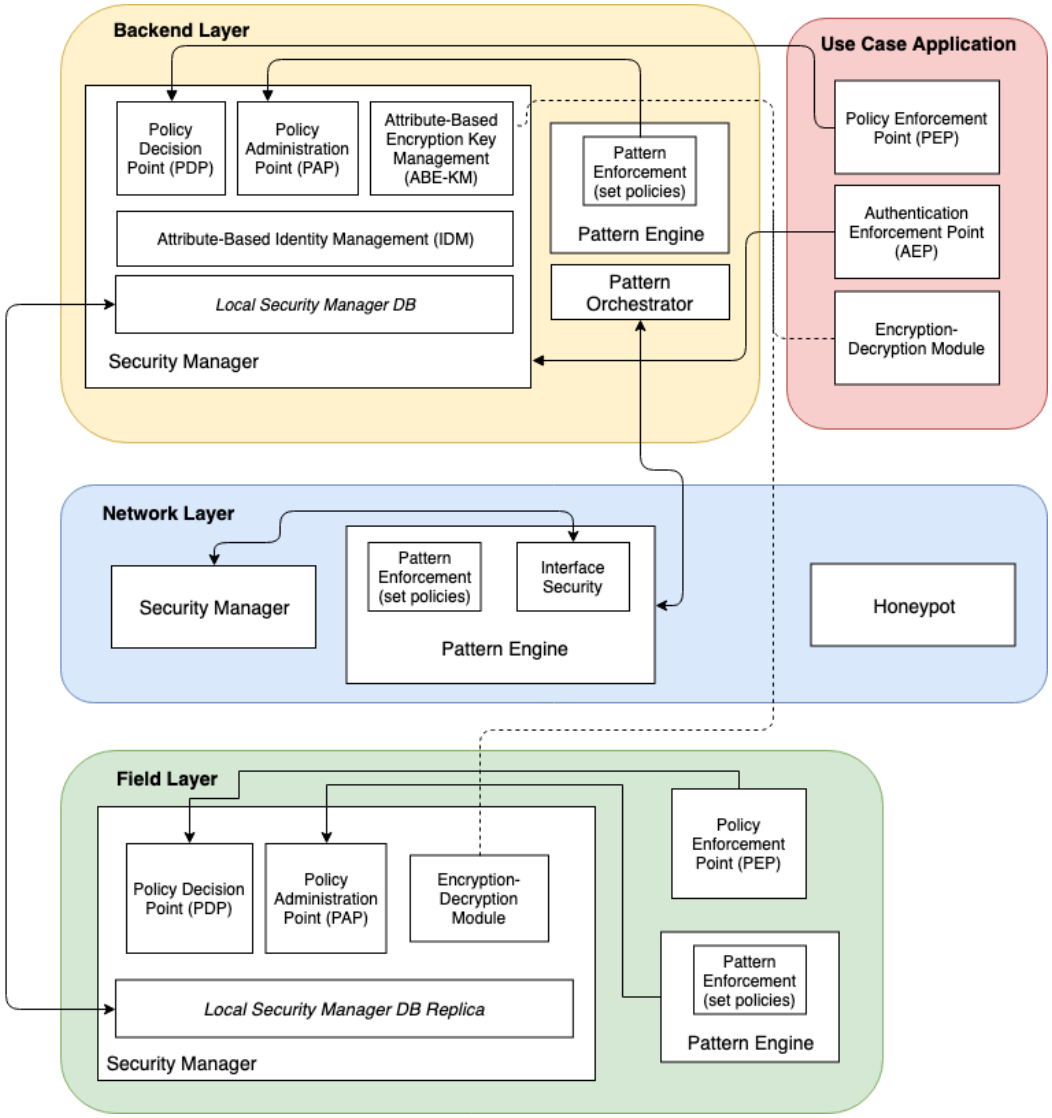


FIGURE 17 SECURITY COMPONENTS DESIGN AND ARCHITECTURE

Figure 17 shows how the pattern engines at different levels interact with the Policy Administration Point (PAP) to enforce specific SPDI patterns. For example, a system can have multiple states depending on privacy needs. In this scenario, the system would execute under normal conditions, but whenever a specific event is detected (an elderly fell down at his home), the pattern engine may interact with the Security Manager at the backend to relax privacy policies temporarily. This would give temporary access to information that is otherwise inaccessible to another user.

Furthermore, the SDN controller can also leverage the Policy Decision Point (PDP) in the backend to delegate needed security-relevant decisions, e.g., concerning role-based access control decisions. Also, we add functionality to the backend Security Manager to map attributes or policies to keys used for ABE encryption as the key management process. Furthermore, the ABE encryption can be performed in a distributed manner relying on the keys generated by the backend Security Manager, in order to protect privacy-sensitive user data across the architecture.

The SEMIoTICS security architecture, shown in Figure 17, remains flexible enough to support a wide range of use cases. Particularly, Use Cases which require a single, centralized, PDP can use the security manager in the backend and deploy multiple PEPs where needed, e.g., in the IoT gateway. However, thanks to the attribute-based encryption approach, the ABE encryption also enables security at rest and in transit for use cases with connectivity restrictions, or where data is stored in external clouds. Even for these cases the Security Manager and especially the ABE encryption ensures access control through encryption, with a central identity management in a more decentralized and loosely interconnected deployment.

3.2. Security Components in the Backend

The main security components of the Backend are the Security Manager which consists of the policy decision point, the policy administration point, the attribute-based identity management as well as the corresponding attribute-based encryption key management. The second component in the backend despite the SM is the Pattern Engine that provides safety for the SM.

3.2.1. SECURITY MANAGER IN THE BACKEND

3.2.1.1. POLICY DECISION POINT (PDP)

The policy decision point is responsible for authorization of system units that request corresponding decisions. It is part of a policy-based access control system RBAC, which decides whether a user's request should be approved or not based on the available information (attributes) and the applicable security policies. If an entity in the SEMIoTICS contexts wants to request access to any instance that is related with the Security Manager it first informs the PEP which forwards this information to the PDP. There the request will be evaluated on the base of the stored user attributes. The response is then sent back to the policy enforcement point, where it gets retransferred to the desired user origin.

In our PDP we have the following functions to check the policies:

- `CanReadPolicyField`
- `CanWriteToPolicyField`
- `CanWriteToAllAttributes`
- `CanReadArray`
- `CanRead`
- `CanDelete`
- `CanUpdate`
- `CanReadEntityPolicies`
- `CanSetEntityPolicy`
- `CanWriteToPolicy`
- `CanDeletePolicy`

- `CanReadEntityAttribute`
- `CanWriteToAttribute`
- `CanDeleteAttribute`

3.2.1.2. POLICY ADMINISTRATION POINT (PAP)

The policy administration point in the Security Manager is the interface and tool that enables the creation and the editing of the SEMIoTICS digital policies, or rules. The single policies can be defined by the provided REST-API or through the management UI that the Security Manager provides.

Figure 18 shows a basic interplay of all single components that are needed e.g. when Alice wants to view Bobs current location.

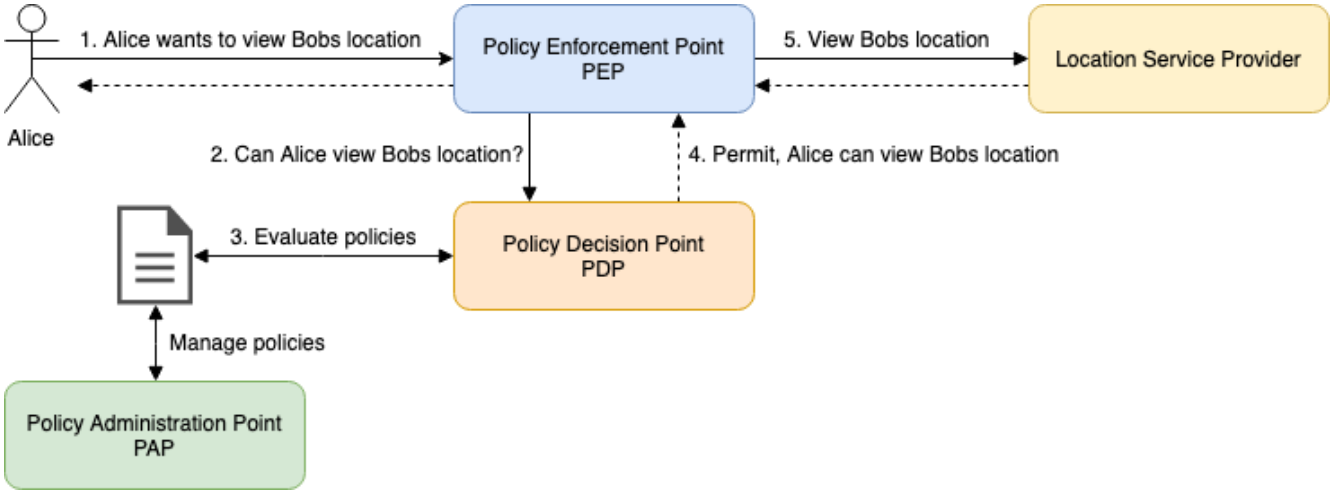


FIGURE 18 INTERPLAY OF PEP, PDP AND PAP

3.2.1.3. ATTRIBUTE-BASED IDENTITY MANAGEMENT

In large scale deployments SEMIoTICS foresees already one or more instances of the security manager in the backend, where its identity-related functions enable the security manager to act as an oauth[7] authentication provider for different realms within the same or even across organisations. Each “user”, who got an identity within the identity management system provided by SEMIoTICS Security Manager could be a device/component or a human user. Each can have an entity-ID and can be given different attributes to enable different policy decisions and different decryption-capabilities.

Traditionally, authentication is a proof of identity in the form of a password or identity document. In most cases, however, the precise identification of persons utilizing unique identification numbers and names is complete overkill. In most cases, a single attribute is often sufficient to know a person or to authorize a specific transaction. In the context of SEMIoTICs, once an entity gets registered in the system, the Security Manager assigns him/her/it several attributes, that describe this specific entity. Such attributes can e.g. be attributes that describe the role of the entity such as ‘doctor’, ‘admin’, ‘device’, but also attributes that describe certain characteristics of the entity such as ‘age=18’, ‘gender=male’, ‘microphone’, ‘camera’ etc. Afterwards, and before being able to use a service (e.g. access data, run a command), the entity has to be authenticated at the Security Manager (which also takes the role of an oauth authentication provider), by providing his/her/its credentials. The Security Manager then responds with a so called “token”, which functions as a cryptographic container, that hides the attributes. Thus, when the service that requests the authentication of an entity in order to check if he/she/it is authorised, it will only see a token. The service checking the token cannot derive information about other attribute values it contains. However, an entity can use this token to prove the validity and authenticity of his/her/its attributes towards the service. This way, the entity can authenticate him-/her-/itself without an identification (role-based access control or proof of membership). This increases the privacy and at the same time eases access control management.

[7] oauth is an industry standard for authorizations allowing for federated identity management

To sum up, the concept of attribute-based identity management in SEMIoTICS enables and enhances several security aspects:

1. **Unlinkability**; The attributes cannot be linked to an entity, as they are hidden within the token (=cryptographic container)
2. **Basis for advanced security mechanisms**; Attributes of an entity can be used for Attribute Based Encryption / Decryption (see next Section). Only entities with matching attributes can successfully decrypt the data. Encryption can happen loosely-coupled because at the time of encryption only the attributes of the potential entity that is able to decrypt them later are required, not the exact entity, e.g. one can encrypt data to be readable only by those who hold the attribute `role=doctor'.
3. **Simplification of Policy Management**; Instead of creating multiple policies for several users, one can create a policy for an attribute or a set of attributes. As multiple users may share the same attribute (e.g. 'role=doctor'), the policy can affect several users at the same time.

Further, the SEMIoTICS framework also allows a flexible deployment of one or maybe more security manager components, each spanning what we have termed a security domain in the figures below.
Additionally, please note that there are at least the following two reasons why one would have more than one security manager component:

- Reason 1: there are different security domains and the don't work together, which results in users/devices that are part of two domains having to have two IDs, one in each domain
- Reason 2: there are several security managers that are catering together for one domain, this is the case if we have deployed one or more SM at the edge.

The Figure below shows the simplest deployments scenario, with several users/devices and different realms and the different Security Managers. Then, if you would have to cater for more than one security domain, you would simply be "cloning" the Security Manager and if you would want to enable cross-realm interaction, then users/device are in need of having several tokens/credentials for each domain as device "B" that is in both realm 1 and 2 as shown in Figure 19 Security Manager deployment overview.

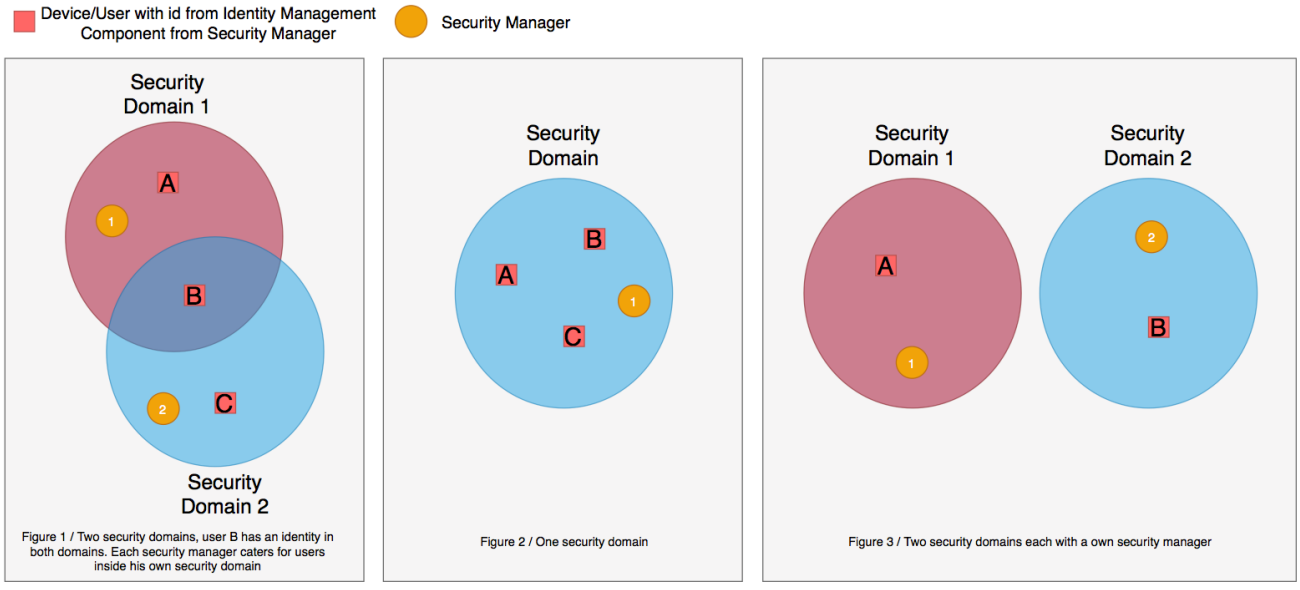


FIGURE 19 SECURITY MANAGER DEPLOYMENT OVERVIEW

3.2.1.4. ATTRIBUTE-BASED KEY MANAGEMENT

The attribute based key management component provides an API that can be used by the applications to encrypt data that they want to securely store in the Security Manager's database, using Attribute-Based encryption. It currently offers two state-of-the-art schemes, namely KP-ABE and CP-ABE. The advantage of ABE is that the entity that encrypts the data, does not have to know the public key of the entity that should

receive the data, at the time of encryption. Furthermore, data can be encrypted for multiple entities, which can be highly useful for the SEMIoTICS use-cases (especially use-case 2).

The complete documentation of the provided ABE API can be found in Deliverable D5.3.

3.2.1.4.1. REQUIRED FUNCTIONALITIES

This section describes the required four required functions that every ABE implementation must implement in order to provide full and secure functionality.

3.2.1.4.1.1. Setup

The setup function initializes the chosen Attribute-Based Encryption Scheme (CP-ABE or KP-ABE). It requires no input and outputs the master key (MK), which must be kept secret within the Security Manager, and the encryption key (EK), which is publicly known.

The resulting function looks as follows:

$$(\boldsymbol{MK}, \boldsymbol{EK}) = \boldsymbol{Setup}()$$

3.2.1.4.1.2. Key generation

The key generation function generates a decryption key DK_e (private key) for an entity e. This entity is either described by a set of attributes (CP-ABE Scheme) or limited by a policy (KP-ABE Scheme). As input, the function takes the master key MK as well as the information γ_e of the entity e (set of attributes (CP-ABE) or the assigned policy (KP-ABE) of the entity). The resulting decryption DK_e embeds the entity's information γ_e. This key generation can only be executed by the owner of the master key MK, which, in the context of the SEMIoTICS framework, is the Security Manager.

The resulting function looks as follows:

$$\boldsymbol{DK_e} = \boldsymbol{KeyGen}(\boldsymbol{MK}, \boldsymbol{\gamma_e})$$

3.2.1.4.1.3. Encryption

The encryption function is used to encrypt (privacy sensitive) data. This requires the access control data of the file (in the form of a policy in the case of CP-ABE or in the form of a set of attributes in the case of KP-ABE) $\boldsymbol{\tau}$ and the encryption key EK as input. The output is the encrypted data (ciphertext) C, in which the access control information τ is embedded.

The resulting function looks as follows:

$$\boldsymbol{C} = \boldsymbol{Encrypt}(\boldsymbol{M}, \boldsymbol{\tau}, \boldsymbol{EK})$$

3.2.1.4.1.4. Decryption

Decryption is successful only if the attribute set γ_e, which is embedded in the private key DK_e of an entity e, satisfies the policy τ, which is embedded in the ciphertext C (CP-ABE) or if the policy γ_e, which is embedded in DK_e allows to access the set of attributes τ, which is embedded in the ciphertext C (KP-ABE).

The resulting function looks as follows:

$$\boldsymbol{M} = \boldsymbol{Decrypt}(\boldsymbol{C}, \boldsymbol{DK_e})$$

3.2.1.4.2. KEY REVOCATION

As already described in Section 2.2.6.3, there are currently two state-of-the-art possibilities for attribute-based key revocation.

i. Attribute Version List:

The security manager maintains a list which contains all active and currently used attributes/policies for the keys. Each attribute/policy is thereby marked with a version number. Whenever an entity is compromised, the security manager revokes the key (and all associated attributed for this key) by increasing the version number of the affected attributes/policies. Afterwards, the Security Manager has to distribute new keys to all registered entities, that have at least one of those affected attributes/policies within their keys. Furthermore, all currently stored (privacy sensitive) data has to be re-encrypted with the new active attributes which are stored in the list.

E.g. using the CP-ABE scheme, there are two entities (A and B), which both have the attribute “Doctor_v1” embedded in their DK. Both entities can access the privacy sensitive data S, which contains the patient’s current location, as this data can only be decrypted by entities with the attribute “Doctor_v1”. After a while, it is detected that entity B has been compromised I.e. an intruder is now able to use B’s DK to access the patient’s location S. Therefore, the Security Manager revokes the key of B as well as all the associated attributes (which is “Doctor_v1”). The Security Manager then updates the version of the affected attributes (“Doctor_v1” gets updated to “Doctor_v2”), and distributes the new DKs to all entities that had the attribute “Doctor_v1” embedded in their DK i.e. entity A. Afterwards, all encrypted, stored data gets re-encrypted with the new version of the attribute (“Doctor_v2”). Now only A is able to decrypt it, as B has not received the new version of the attribute.

ii. Time attribute within the ciphertext:

The security manager embeds an additional time attribute within the ciphertext, so that the data can only be decrypted within a fixed time-window. Afterwards, when an entity requests a key, the used time attribute also gets embedded within the entity’s key, so that it can access the data within the defined time-window as long as it is allowed to decrypt the data. When this time-window expires, the security manager can decide to either re-encrypt the data with a new time limit and distribute new keys to all registered entities, so that they can decrypt the data again or simply “drop” the maintenance of the data, so that nobody can access it anymore. This is particularly useful if a certain data value should only be available for e.g. max 1 week.

However, both approaches generate significant overhead (distributing the new keys, updating the encryption of the data etc.). We will further analyse the feasibility of these concepts within the Use-Case deliverables (especially in Use-Case 2).

3.2.2. BACKEND PATTERN ENGINE

As already presented in D4.6, D5.2 and in this document in Section 2.2.11.2, the Pattern Engine is a component that is not strictly focused only on security rather it entangles security aspects. It is responsible for reasoning on the Security, Privacy, Dependability, and Interoperability (SPDI) properties. The said reasoning is accomplished with the help of patterns that are represented as Drools rules. These rules are inserted, modified, executed or retracted at design as well as at runtime. These interactions are conducted with the help of Pattern Orchestrator and are encrypted with the use of SSL certificates that are preinstalled in the components. Regarding the Security Patterns, are exhaustively presented in D4.8.

3.2.2.1. PATTERN ENFORCEMENT

The patterns that are used by the Pattern Engine can be distinguished to two types. The first one is the verification type which allows the Pattern Engine to verify whether an SPDI property is valid or not. The second one is the adaptation type which gives the capability to the Pattern Engine to be able to take appropriate actions, whenever possible, in order to restore the validity of an SPDI property.

When a security property is not satisfied we notice three scenarios. In the first scenario, there is a component that does not satisfy the security property however by applying configuration changes to the said component, it is possible for the security property to be satisfied. In this case the Pattern Engine will push the necessary configurations changes therefore enforcing the required security property.

Similarly, in the second scenario we assume that there is a component that does not satisfy the security property. This time there are no configuration changes that can be made to that component in order to have the security property satisfied. The Pattern Engine will then attempt to replace the said component by activating another component that can satisfy the security property and is available in the domain of operation. When this

is not possible, we arrive at the third scenario, where the Pattern Engine sends an alert to the Computer Emergency Response Team (CERT). The role of the CERT is explained in section 4.2.
An example of the first scenario is in UC1, where there is an MQTT Broker that may not use TLS/SSL to interact with its subscribers. The Broker is capable of using TLS/SSL when some configuration changes are applied, therefore the Pattern Engine can enforce the security by pushing these configurations to the Broker.

3.2.3. PATTERN ORCHESTRATOR

The Pattern Orchestrator component is responsible for the automated configuration, coordination, and management of different patterns and their deployment. More specifically, it:

- Receives instantiated recipes, also featuring the included SPDI & QoS requirements, from the Recipe Cooker via a well-defined API
- Extracts SPDI & QoS properties/requirements from instantiated recipes and maps them to specific SEMIoTICS patterns
- Converts patterns to Drools
- Classifies and distribute patterns (as Drools) to the different pattern engines in three layers (Backend, Network, Field)
- Collects the pattern status from said pattern engines at the various layers
- Maintains a link with the SEMIoTICS GUI to provide an up-to-date view of the status of the patterns in the context of a specific orchestration to the system (SEMIoTICS) operators

As such, the importance of the Pattern Orchestrator from a Security perspective is centred around is role as a key component in relaying the application security and privacy -related requirements to the underlying components, as well as its role as an aggregator of the current security and privacy (among other properties) status of the deployed IoT orchestrations implementing said applications.

3.2.4. USE CASE APPLICATION

3.2.4.1. POLICY ENFORCEMENT POINT (PEP)

In order to secure access to the components' resources, the APIs of components deployed on Backend Orchestrator are protected by Policy Enforcement Point. PEP provides security in two ways. Firstly, it uses the in-built Kubernetes mechanism known as pod networking - PEP as a sidecar along with the primary application are deployed as separate containers inside a single pod. They both share the same network allowing them to communicate through localhost. The main application is only accessible from within pod, because it does not expose its port outside the cluster, only PEP does that, meaning that in order to get the resource of the main application all HTTP traffic must go through PEP. Secondly, the HTTP request must bear a valid authorization token and the client which makes an HTTP call has to be authorized to do so. To evaluate client authorization, PEP uses the Security Manager's Policy Decision Point. After the evaluation of the client's policy, HTTP call is either allowed to reach the main application or it is rejected due to the insufficient permission. Policy Enforcement Point provides functionality to configure the way of mapping the request body which is sent to Policy Decision Point based on the properties of an intercepted request. That allow tightening or loosing security rules for the specified API endpoints.

```
{
   "actions": [
      {
      "entityId" : "application123!@!local",
      "entityType": "app",
      "method": "write",
      "field": "update"
      }
   ]
}
```

FIGURE 20 AN EXAMPLE OF THE EVALUATION POLICY REQUEST BODY

3.2.4.2. AUTHENTICATION ENFORCEMENT POINT (AEP)

Having communication between components secured in a microservices architecture is difficult and might be very time consuming to implement due to duplicating security-related code in each component. Since Policy Enforcement Point requires an authorization token in the header of HTTP request and moreover, to avoid duplicating security-related code, there was a need to provide a tool to ensure that the authorization token is present. Authentication Enforcement Point is created out of mitmproxy tool.
Authentication Enforcement Point it is a stand-alone application which proxies HTTP requests coming out of the container with the main application. The proxy alters the communication between two parties which believe that they are directly communicating with each other. The proxy intercepts all the traffic coming out of the container with the main application in order to add an authorization token to the HTTP requests header. By doing so we are able to authenticate HTTP requests without making any changes in the source code of the application.

To apply a valid token, AEP uses Security Manager's OAuth 2.0 Clients Credentials Grant flow (Figure 21). Every component is registered in Security Manager with its client-id and client secret. Using these values, Security Manager can generate a unique token for the component, which is added to the HTTP request header in order to not be rejected by Policy Enforcement Point.

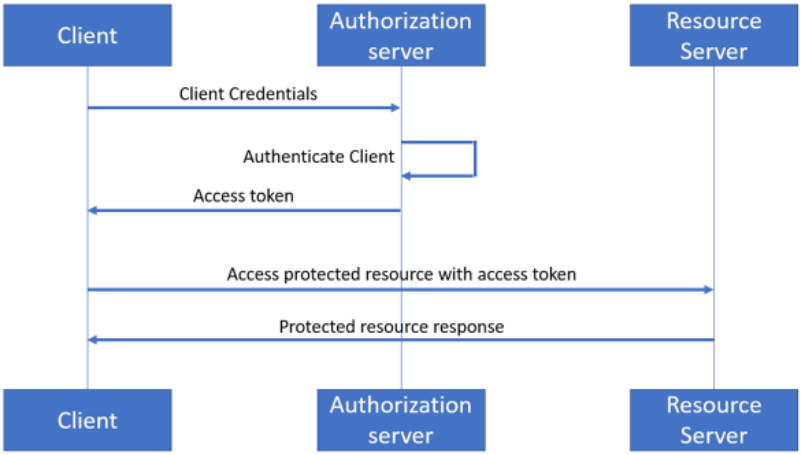


FIGURE 21 OAUTH 2.0 CLIENT CREDENTIALS FLOW

3.2.4.3. SECURITY MANAGER INTEGRATION WITH AEP AND PEP

To have a better understanding of how the security between components has been implemented, the figure below (Figure 22) shows an entire example of how AEP and PEP work along with the Security Manager. At first, the Primary Application requests data from the Resource Application. To do that, an HTTP request is sent to the Policy Enforcement Point of Resource Application, because only the PEP exposes its port the outer cluster (1). The request then is intercepted by Authentication Enforcement Point. AEP is configured based on which application is using a proxy, meaning that Primary Application's client-id and client secret are accessible. AEP authenticates the Primary Application with the Security Manager using its client-id and client secret (/oauth2/token endpoint) (2). Security Manager sends a response with an access token to AEP (3). At this point, AEP adds the token to the original HTTP request header and forwards the request to its original destination (4). As soon as the request reaches the Policy Enforcement Point, it is verified whether the application whence it came from has sufficient privileges to get access to the resource. The request is compared to the configuration kept in PEP to create a new, policy verification request to Security Manager. The policy verification request is sent to the Security Manager's Policy Decision Point (/api/v1/pdp/batch endpoint) (5). Security Manager evaluates the given request and responses with a boolean (6). At this moment, if the Security Manager's response is negative, the response of the original request will be 403 Forbidden (9), else if the Primary

Application has got sufficient privileges to get the access to the requested data, the original request is forwarded to the Resource Application (7). Finally, the API of Resource Application responds with requested data (8, 9).

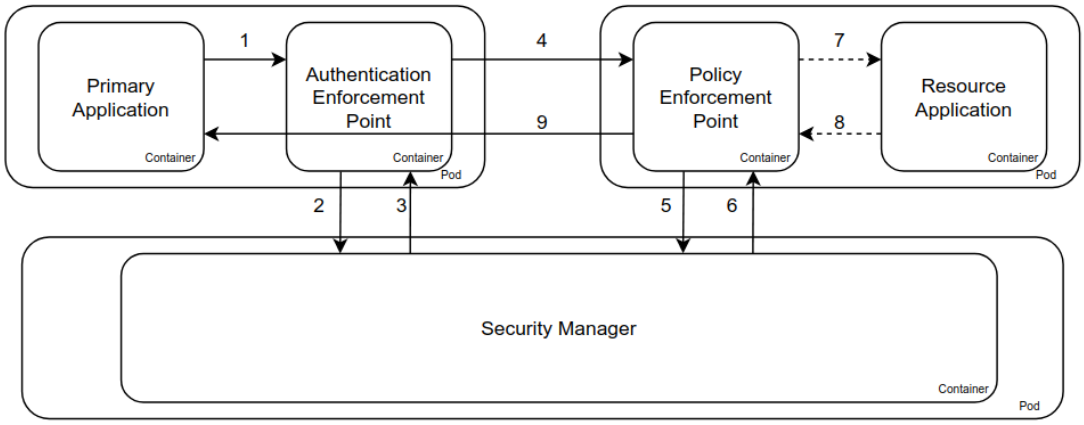


FIGURE 22 COMMUNICATION FLOW BETWEEN COMPONENTS

3.2.4.4. ATTRIBUTE-BASED ENCRYPTION/DECRYPTION MODULE

The Attribute-based Encryption and decryption module uses the provided ABE-API of the Security Manager. This API allows the Use-Case application to securely store (privacy sensitive) data encrypted under ABE (with a pre-defined set of attributes or a policy) within the Security Manager's database. Furthermore, the API can also be used to decrypt stored data, whenever an entity wants to access (privacy sensitive) data.
However, the entity can only decrypt the data successfully, if and only if the embedded entity information within the DK satisfies the requirements of the encryption information that is embedded within the ciphertext.

3.3. Security Components in the Network Layer

3.3.1. SECURITY MANAGER IN THE SDN

The Security Manager (SM), as presented also in D3.7, offers support for authentication and accounting services. The said services are realized to the rest of the SDN Controller as well as the users and applications that interact with the controller. Regarding authentication, the SM exposes interfaces for the administration of local SDN Controller accounts. Additionally the SM is able to generate authentication tokens based on local credentials thus providing authentication capabilities. Applications can present their credentials to SM and if proven valid they are issued authentication tokens. Any interaction with the SSC will include the presentation of said token to the SM for validation in order for the interaction to be allowed or not.

Authentication and authorization mechanisms protect the interfaces of the SSC. Interfaces relevant for SEMIoTICS, including the Security Manager, VTN Manager and Pattern Engine northbound interfaces, are protected by HTTPS digest authentication, thus supporting the R.S.7 requirement. To protect and isolate access to particular internal APIs of the controller, Security Manager enables role-based definition of authorities granted access to the service, thus fulfilling the requirement R.S.2.

3.3.2. NETWORK (SDN) PATTERN ENGINE

The Pattern Engine in the SEMIoTICS SDN Controller (SSC), as already presented in D3.7, is able to detect invalid rule configurations by means of component observations by subscribing to network topology events. The Pattern Engine additionally exposes a bidirectional interface towards backend. On each status change of an active pattern instance, the remote Pattern Engine is notified, so that additional reconfiguration steps can be partaken there. Due to the nature of the Pattern Rules, the interaction of PE with other components inside the SSC is not limited to specific components, but can also be extended to others. The ability to add new rules during runtime provides flexibility to interact with existing or future components of the SSC whether they are security related or not.

3.3.2.1. PATTERN ENFORCEMENT

Similar to Section 3.3.2.1 patterns, are distinguished to two types in the Pattern Engine at the SDN, verification and adaptation. Regarding the adaptation type, those rules tackle with service Function Chaining (SFC). SFC patterns are developed to achieve the requirement for end to end guarantees by the traffic forwarding through different security service functions. The patterns enforce the following requirements:

- Verify service function chains on chain requests
- Instantiate service function chains, if the required functions have been already instantiated.
- Verify functions to insert them in the request chain
- Instantiate not defined service functions, to satisfy service function chain requests.

The procedure of instantiation and the identification of the respective SFCs and the VNFs, this can be based on the actual interaction between the components of the SEMIoTICS architecture. Pattern Orchestrator forwards a specific chain request to the Pattern Engine for forwarding the traffic between entities through a specific chain of functions. Pattern Engine forwards this request to the SFC manager which is located in the SDN controller responding to the Pattern Engine whether the chain exists or not. If the chain exists, then a respond of the chain satisfaction is returned to the Pattern Orchestrator. If the chain does not exist, then a requested is forwarded from the MANO requesting whether the service functions exist or not. If functions exist in the VIM, then the chain can be instantiated in the SFC Manager and a respond of the chain satisfaction is returned to the Pattern Orchestrator. If functions do not exist in the VIM then, a function instantiation request is forwarded to the NFV Orchestrator, which is responsible to instantiate them in the VIM. Then, the chain can be instantiated in the SFC Manager and a respond of the chain satisfaction is returned to the Pattern Orchestrator.

3.3.2.2. INTERFACE SECURITY

Since the SDN pattern engine exposes a northbound interface (for specifying SPDI/QoS properties and to support other interactions with the backend, as mentioned above and presented in detail in D3.10) an important consideration were the security aspects of the exposed interface.
The SDN Pattern Engine adopts the security mechanisms available in the ODL controller, and by extension the SSC, which features basic authentication capabilities (via username and password). Through this feature, all modules used by the SDN controller are subject to this authentication. Case in point, the Pattern Orchestrator is forced to provide credentials in order to be able to communicate with the SDN Pattern Engine.
Additionally, security is hardened on a per-case basis, considering the intrinsic requirements (e.g., complexity of interactions) foreseen in each scenario. Said intrinsic requirements are addressed by adding encryption to the communication (E2E, where needed) by using SSL/TLS in the REST endpoints.
Moreover, advanced Authentication, Authorisation and Accounting (AAA) features are implemented with the help of the Security Manager who is responsible for providing tokens that enforce the said features. Leveraging these mechanisms, the communication between Pattern Orchestrator and pattern-driven NBI is hardened by the use of an authentication token. In all interactions, the Pattern Orchestrator will first request a token from the Security Manager that will later use to contact the pattern-driven NBI. If the token is verified to be able to grant access to the pattern-driven NBI then the communication proceeds successfully. This process prevents unauthorized use of the pattern-driven NBI.

3.4. Security Components in the IoT Gateway & IoT Devices (Field Layer)

3.4.1. SECURITY MANAGER ON THE GATEWAY

In addition to using the Security Manager on the Backend, as outlined in Section 3.2.1, the entire SM is also used on the gateway at the field level. It can be regarded as a fully working local replica of the SM in the Backend as well as the Policy Enforcement Point (PEP) to include redundant components. This is done because of reliability reasons and to ensure safety in case that the IoT Gateway is not able to establish a direct connection to the Backend.

3.4.1.1. POLICY ENFORCEMENT POINT (PEP)

In order to provide a redundant endpoint to accept incoming requests for an application provided by the field layer, the Security Manager's PEP can be deployed as a standalone replica on the Gateway. There the PEP is

responsible for bundling and forwarding all requests to the SM component. Without this PEP replica on the Gateway a “offline” operation of the Gateway couldn’t be ensured.

3.4.1.2. POLICY DECISION POINT & POLICY ADMINISTRATION POINT LOCAL-SHADOW REPLICA

From a deployment scenario the idea was to have enable local policy decisions to be taken for

- increased efficiency (lower latency, less communication outside of the gateway), and
- increased data-protection (less communication outside of the gateway), and
- increased availability (local shadow replica remains reachable if the network connection is too slow or not available).

This means that a subset of the functionality of the Security Manager will also be replicated and distributed onto local gateways. Of course, in large scale deployments there can be more than one gateway that is managed by a Security Manager and as we have discussed previously there can also be more than one Security Manager. In the following Figure you will find visualizations of the possible scenarios showing the deployment.

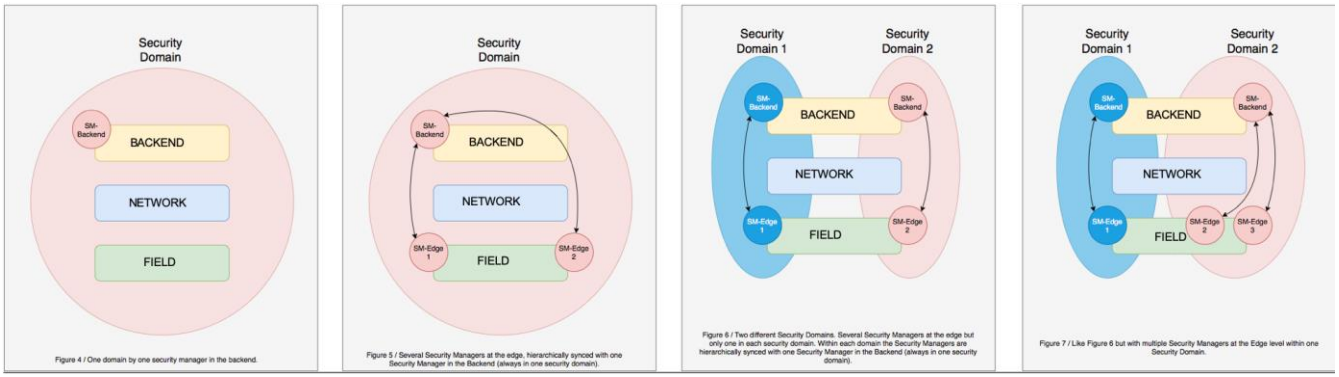


FIGURE 23 VISUALIZATIONS OF POSSIBLE DEPLOYMENT SCENARIOS

Note, the replicated Security Manager will provide the functionalities of only PEP and PAP locally. The local version is under normal conditions trying to obtains the policy decision directly from the Security Manager in the Backend (for PDP functionality) or update the policy directly on the database of the Security Manager in the Backend (for PAP functionality). However, when those direct requests are not possible the replicated shadow will provide the functionality locally, and synchronise back with the backend once possible. This local fallback allows the policy enforcement to still function with a local view on the policy. Additionally, the policy decision point could be configured to deny access if the last contact to its backend is too long ago.

The details of the implementation and the roll-out during the deployment of both Security Managers in the backend as well as their shadows in the local edge on the IoT gateway are discussed in the implementation specific deliverable D5.5.

3.4.2. FIELD PATTERN ENGINE (LOCAL INTELLIGENCE ON THE IOT GATEWAY)

As already explained in Section 2.2.11.2, the Pattern Engine is a component that is not strictly focused only on security rather it entangles security aspects. In accordance to that, the Pattern Engine on the gateway is a light version of the Pattern Engine that exists on the Backend due to the limitations on memory and processing capabilities. In same manner, it is responsible for reasoning on the Security, Privacy, Dependability, and Interoperability (SPDI) properties at the field layer. The said reasoning is again accomplished with the help of patterns that are represented as Drools rules. These rules are inserted, modified, executed or retracted at design as well as at runtime. Just as in the Backend so, in the field, the interactions are conducted with the help of Pattern Orchestrator and are encrypted with the use of SSL certificates that are preinstalled in the components. Finally, any fact that is added or deleted to the Pattern Engine on the gateway is also modified accordingly at the Pattern Engine at the Backend.

3.4.2.1. PATTERN ENFORCEMENT

Similar to Section 3.2.2.1 patterns are distinguished to two types in the Pattern Engine at the field layer, verification and adaptation. The same three scenarios described in Section 3.2.2.1 also apply here.
An example of the described scenarios is in UC1, where a sensor such as the inclinometer at the field layer may not use encryption to interact with the Broker. If the said sensor is capable to support encryption, the Pattern Engine at the field layer, will push the necessary configuration changes therefore enforcing the required security property. Alternatively, if the specific inclinometer does not support encryption, the Pattern engine will attempt to locate another inclinometer that exists in the installation and supports encryption. The information regarding the available components of the local installation can be retrieved from another SEMIoTICS component, the Local Thing Directory. Finally, if no replacement component can be found, an alert to CERT is sent in order to take appropriate actions.

4. INCIDENT DETECTION AND RESPONSE IN SEMIOTICS

4.1. Security Network Mechanisms for Monitoring and Incident Detection

To monitor the security incidents, a collection of tools and procedures are required to detect incidents when they happen or are near happening is required. As described previously, security in SEMIoTICS is strengthened by combining the various Security Functions, employing the flexibility of SDN/NFV–enabled to enhance service Function Chaining as described also in D3.8. More specifically, to offer continuous monitoring of incoming traffic, and detecting and adapting to different types of attacks. The variety of previously described security network functions such as Firewalls (FW), Intrusion Detection Systems (IDS), Deep Packet Inspection Systems (DPI), Honeypots (HP) and HoneyNets, can create a number of function chains, to forward traffic based on the type of traffic or running application.

4.1.1. DATA FEED BY HONEYPOTS

Honeypots appear to be efficient security mechanism to detect attacks and malware. In addition, a HoneyNet can be deployed, consisting of Honeypots emulating more than one network elements, as well as Honeypots emulating the operational systems. With this mechanism, malicious traffic can be isolated in the honeypot, allowing us to track the attacker, identify her purpose and keep her occupied. Simple Honeypots[8] and passive Honeypots (Early Warning Intrusion Detection Systems, EWIS, in specific[9]) are also capable to be part of the HoneyNet, acting as a Network Telescope on the production part of the industrial network, to monitor all activity

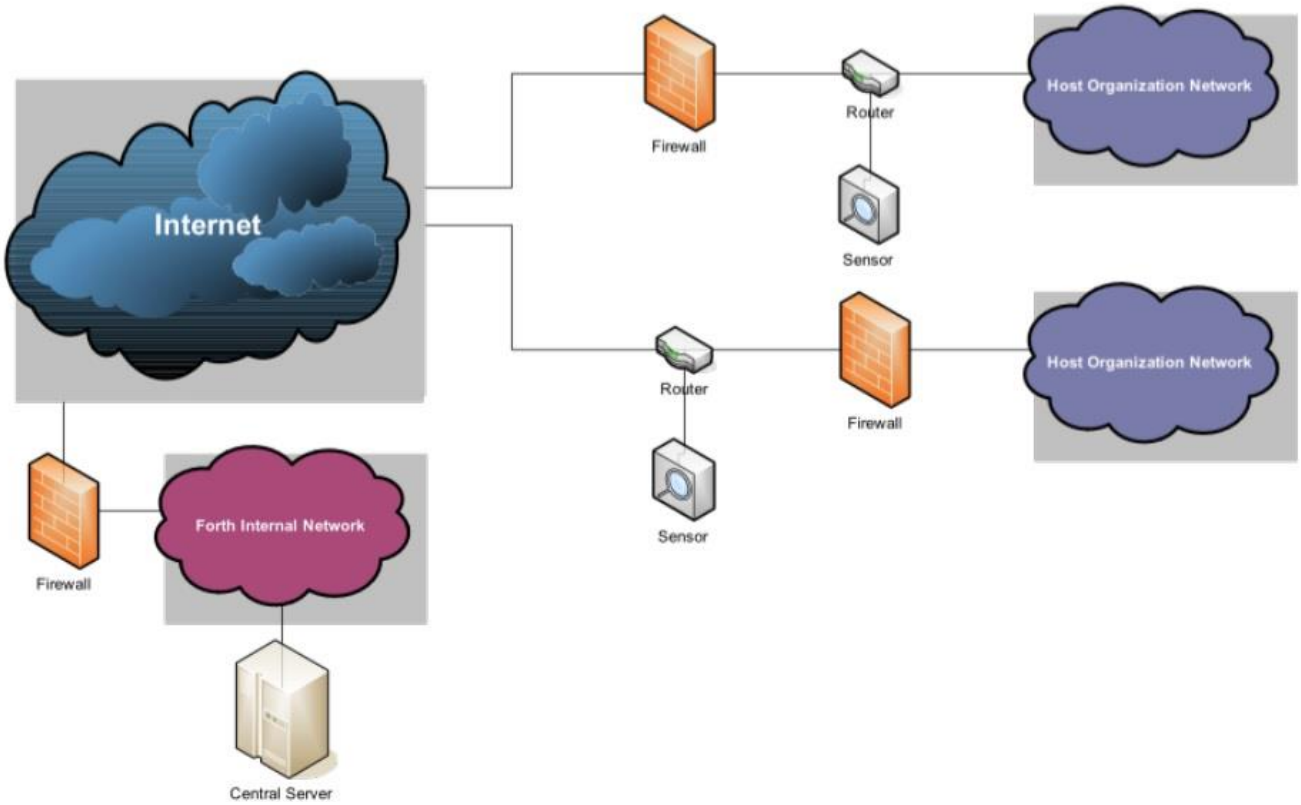


FIGURE 24 POSSIBLE SCENARIOS FOR EWIS SENSOR DEPLOYMENT

in normally unused parts of the network. The honeypot sensor network (EWIS) is a self-developed system

[8] http://www.honeyd.org

[9] Chatziadam, Panos, Ioannis G. Askoxylakis, and Alexandros Fragkiadakis. "A Network Telescope for Early Warning Intrusion Detection." International Conference on Human Aspects of Information Security, Privacy, and Trust. Springer International Publishing, 2014.

operated by FORTH. The honeypot sensor monitors darknet traffic and reporting back patterns that can potential indicate a network anomaly which in accordance with the data derived from the IDS devices can point to a "brewing" network incident.

The concept behind EWIS is to establish a system that would be cost effective to implement, easy to deploy and provide sufficient data to create an Early Warning System that could potentially detect large scale events on a global scale. As EWIS operates on un-used address spaces, all traffic reaching it can be classified as malicious thus avoiding unnecessary filtering of legitimate traffic. As security is a major concern, the sensors are built on a secured OS platform, run the bare minimum of services and are protected by a firewall. The fact that EWIS's sensors utilize a passive approach to data collection is also a favourable point by an organization that certainly doesn't need any devices within their infrastructure to be talking back to a potential attacker. From a network topology point of view, the sensors can be positioned outside the organization's network on the Demilitarized Zone (DMZ) or even the public section of the network just outside the organization's firewall(Figure 24). Isolating the sensor from the organization's Intranet and at the same time fully exposing it to the outside world is the best way for deployment. When the sensor is compromised, there would be no posed threat to the internal network of the host organization. In such an unlikely case, we can simply replace the sensor.

EWIS can be installed either as a virtual or physical server with specified hardware and bandwidth requirements and hardened OS on a Linux (Ubuntu 14.04 in our case). These requirements of virtual honeypot in a virtual infrastructure (i.e. Openstack) can be requested during its instantiation. EWIS honeypot requires a fast network connectivity to support the passive traffic monitoring and forwarding and mirroring Traffic. Therefore, the 100Mbit Internet access and 1Gbit access within the domain should be provisioned by the SDN Controller as well as the physical connectivity. These systems could be dummy controllers that will attract attackers and record their moves as well as identify other misbehaving control plane elements from the information received by other controllers. These systems will not participate as functional components of the network infrastructure. Prior to deployment, each sensor is assigned a dark network space that could span from a few IPs to entire subnets. When traffic arrives for a specific IP address the router broadcasts an ARP request in order to discover the host. When the sensor replies to the ARP request as the owner of the corresponding IP address, the router directs all traffic to the sensor. The sensor's monitoring daemon captures the received traffic and records it to the database. This in-house developed daemon uses the pcap library to capture information such as the source and destination IP addresses, the source and destination ports, the flow payload size, the protocol type (number), the TCP flag (in case the flow is of TCP type) and the associated timestamp.

4.1.2. DATA FEED BY INTRUSION DETECTION SYSTEM (IDS)

Data feed from the IPS/IDS monitoring device: IDS in SEMIoTICS include continuous network monitoring and intrusion detection for identification of attacks and run-time network adaptation for attack response and mitigation mechanisms. More specifically, IDS instances of Snort are deployed, with scripts to ensure that the most up-to-date rules are constantly active. A database for event monitoring is present, while provisions are made to allow for future extensions to transmit relevant information to a security backend (e.g. for more sophisticated pattern matching).

4.2. Incident detection and mitigation with a Computer Emergency Response Team (CERT)

A Computer Emergency Response Team (CERT) can provide services relating to information security incidents. In SEMIoTICS, the knowledge regarding the role of CERT is transferred by FORTHcert[10], FORTHcert is the Incident Response Team of the FORTH. FORTHcert operates under the Institute of Computer Science and provides services relating to information security incidents. Its mission is to perform security incident

[10] http://forthcert.gr

coordination services for its members, to disseminate information regarding information security issues, to provide a national point of contact to international security community and to promote education and training.
A CERT is responsible to enable information sharing capabilities and tool support between other security teams to manage incoming events and incidents. The trust between the CERT/CSIRT community [11] is guaranteed by the accreditation and certification of CERT Team by authorities such as TERENA[12]/FIRST[13]. A remote management service can enable the system administrator to change the system manually and update the reaction plans based on the latest security guidelines.

A CERT team can be authorized to take operational actions regarding vulnerabilities and mitigation of incidents. Such actions may include but are not limited to blocking access to the network. More specifically, the security team operates or uses these tools or has access to the results generated by them. Mitigation strategies include decision making (prevention, remedial), change roles of user privileges and correct of system problems. For prevention of incidents, a collection of tools aimed at preventing incidents from happening in the constituency. CERT security team operates or uses these tools or has access to the results generated by them. Especially, in SEMIoTICS, to provide system and network prevention, it is required the use of system preventive software backed up frequent network vulnerability security scans as a proactive method for preventing security breach incidents. More specifically, CERT teams can retrieve data and alarms from the previously described mechanisms.

4.2.1. INCIDENT MITIGATION: PREVENTION AND ADAPTATION

Mitigation strategies include decision making (prevention, remedial), change roles of user privileges and correct of system problems. For prevention of incidents, a collection of tools aimed at preventing incidents from happening in the constituency. SEMIoTICS security team operates or uses these tools or has access to the results generated by them. To provide system and network prevention in SEMIoTICS architecture, it is required the use of system preventive software backed up frequent network vulnerability security scans as a proactive method for preventing security breach incidents.

4.2.2. TRACE-BACK AND AUDIT MECHANISMS

The formulation of an anomaly detection hypothesis will have as a result the impact analysis of cause and effect. To provide trace-back and audit, a collection of tools aimed at resolving incidents after they have happened. The security team should operate or use these tools or has access to the results generated by them. Especially, for industrial environment such as a wind park, it is required a 24x7 call reporting of incidents problems. Incidents can be reported via telephone or email to the security experts and administrators. All reported incidents should be logged and tracked for resolution. Incidents should be assigned a priority level and receive technical and management attention according to the priority level assigned. Incidents are addressed as per the incident priority table below and the target resolution times. Although staff could carry cellular phones, during non-business hours, they may not be available to assist with problem resolution within a guaranteed timeframe.

4.2.3. ADOPTION OF MECHANISMS FOR CONDUCTING TRACE-BACKS AND AUDITS

In the adoption of a mechanism for conducting trace backs and audits, it is important that we keep a log of all the steps that we have taken to resolve an incident. When an incident alert arrives, all the information needed about the incident from the reporter should be recorded. Afterwards, a classification of incident priority according the emergency/severity. In case that there is lot of alerts coming, we need to set priorities according to the emergency of the incident so that it is handled in the right order. All incidents should be recorded into a ticketing system by filling out the fields needed (description on incident/whether resolution is reached) including the IP address, set incident priority (set a priority description), and the date incident took place. Any updates to the incident, or any progress to resolve it, the team members need to update the ticket with the relevant information.

All the processes should be recorded following the three-step approach: (i) Record all corresponders and all progress (ii) review what happened by writing a report and (iii) learning lessons for constituency. The availability

[11] https://www.enisa.europa.eu/activities/cert/support/guide2/introduction/what-is-csirt
[12] www.terena.org
[13] www.first.org

and application of an incident classification scheme is used to record incidents. Incident classifications usually contain at least “types” of incidents or incident categories. However, they may also include “severity” of incident. In order to ensure that we respond to incidents in a structured manner it is essential that incidents are classified and prioritized. How an incident is classified depends on various factors of which the most common/significant ones such as the nature of the incident and the criticality of the systems impacted. Furthermore, the number of systems impacted by the incident and the impact of the incident can have on the organization from a legal or public relations point of view and of legal and regulatory requirements

The existing ticketing system provides type and priority fields for incident classification. The type field is used to classify the incident into a major category of problems. The list of problem categories provides the following options: defacement, DoS, infringement, phishing, spam, system hacked, virus. The type field contains the most usual descriptions for the issues we face. If an additional classification or distinction of the incident is required, it should be inserted in the description body of the relevant ticket. The priority field will be used to assign a “weight” or priority to the incident. The assigned priority corresponds to the criticality of the incident as judged by the factors mentioned above. Currently the list of priorities provides the following options such as critical, major, minor, trivial.

4.3. Overview of real time monitoring of threats and attacks to increase awareness in SEMIoTICS

This deliverable describes mechanisms for monitoring and incident detection that can be facilitated in SEMIoTICS. In section 4.1 you find several mechanisms that monitor network-related attacks, and how to react to those is described in section 4.2. For example the deployment of an honey-pot is automatically done if requested by the virtual network provisioning.
Generally, SEMIoTICS considers an overarching approach, including, but not limited to network-centric mechanisms described in the above sections. So apart from that, SEMIoTICS allows to monitor the system to also detect attacks on the level of the whole system as violations of SPDI patterns, e.g. if the network-latency increases, this would be a violation of a dependability pattern and thus would be alerted; or if the current access policy would allow a doctor access to a patient's geo-location that could be seen as a violation of privacy codified in a privacy pattern.
This is achieved by monitoring of vital security functions with the help of the pattern engine. On the network level there are for example alerts of the honey-pot allowing the detection and alerting of real-time network centric threats. On the level of access control policies we have described the regular checks from the pattern engine to the policy decision point in the security manager which check that the actually currently enforced access control policy fulfils the intended checks as implied by the security and privacy pattern. For example that no entity has access to patients location. These functionalities have been implemented and they have been subject to testing and is described in this deliverable (e.g. section 2.2.11.3 describes the privacy compliance monitoring) as well as in the deliverables concerned with on implementation (e.g. D3.8 for the NFV).

The following threat analysis (see Section 5) will show which threats SEMIoTICS found to be most pressing after an analysis of a highly sensitive IoT application and it also shows that certain attacks (e.g. firmware attacks) are not within the scope of the most important problems identified SEMIoTICS. However, making clear and explicit statements about the assumptions made (e.g. attacker is not able to gain access to OS) we make it clear that certain attacks are not in SEMIoTICS scope and must be addressed via existing state-of-the-art or state-of-the-practice security technologies. In case of very specialised hardware attacks, like the above mentioned firmware attack, the use of secure update mechanisms and trusted platform modules (TPM), on the devices for a secure boot and update process should be considered.

Moreover to the technical detection, the handling of alerts or potential alerts to raise the user's awareness for security is achieved in SEMIoTICS through an interplay of various components: a centralised graphical user interface, pattern engine and security functions. The pattern engine will allow you to write patterns which allow to specify high level goals such as confidentiality for data, without the need to understand the full technical details. By this SEMIoTICS allows experts in the domain to set-up and deploy security as well as privacy policies for their specific use case without the need to understand all technical detail, e.g. encryption of network traffic

to guarantee the confidentiality during the transmission of data can be handled by an NFV function. Or the privacy pattern would specify that in a health care scenario the patient's location is not disclosed unless a medical emergency situation is detected, which would then result in regular checks that the access control list that shows who has access to the patient's location is empty and that the related access control policy which denies access is deployed all the way in the field device, e.g. the patient's rollator. All this real-time enforcement and monitoring is done without the need to understand all underlying technologies, thus allows the domain-expert to describe the rules that are necessary for a compliant and safe operation of the scenario in which SEMIoTICS is deployed.

Of course the users of the system will get an alert which they can see in a central graphical user interface, which we refer to as GUI hub. In such a centralised system the alerts will be given to the right person, e.g. from the pattern engine and those in charge will be provided with all the necessary information, as described in section 4.2. More details can be taken from deliverables that showcase the graphical user interface of the different use-cases (see for example the deliverables D5.4, D5.5 and D5.6) as well as more implementation related deliverables (e.g. D5.7).

5. THREAT ANALYSIS OF AN IOT APPLICATION

In this section, we provide a security analysis for a scenario based on the assisted living use case (UC2). In particular, this use case is arguably the most sensitive use case of the project in terms of privacy. Therefore, we have chosen the assisted living scenario to highlight our contributions towards end-to-end security and privacy.

We consider that providing a scenario that exemplifies some threats based on UC2 on a high level, and in a specific privacy-oriented narrative, has more value added than performing threat analysis for all the use cases for multiple reasons. For one, this approach helps the project to define a path to integrate the security features in the task coherently in a concise manner. At the same time, this deliverable helps the reader to understand the threats that our security and privacy frameworks aim to tackle with a "simple" scenario description. Last but not least, a complete risk analysis for all use cases is beyond the scope of the research activities performed in this project, as our goal is to do research activities and evaluate our approaches during the project, instead of focusing on specific threats on a per-scenario basis.

5.1. Methodology

We based our analysis on the Security Development Lifecycle. Thus, we follow the threat and attacker definitions provided by Howard and Lipner [23]. Particularly, a *threat* is defined as an attacker's objective, whereas an attacker or an adversary is also called a threat agent.

Our risk analysis follows the relevant methodology from the Security Development Lifecycle proposed by Microsoft; mainly, we produce data flow diagrams based on the application. The elements in the data flow diagrams can be: an external entity, a data flow, a data store, or a process. Different elements in the data flow diagrams (also sometimes called assets) can face particular attacks depending on their type; that is to say, a process can be attacked differently than an external entity, e.g., a user.

Since the architecture of SEMIoTICS and the use cases are continuously under development, i.e., we have only the first cycle of architecture definition; we perform a security threat analysis based on the external entities, the data flows and the data stores, and processes.

To navigate through potential threats systematically, we apply the STRIDE (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of Privilege) methodology to identify relevant threats to the system.

The STRIDE approach is an acronym was created to identify common six classes of threats. For clarity, we take excerpts of the definitions provided by Howard and Lipner [23]:

- **Spoofing Identity**: Spoofing threats allow an attacker to pose as something or somebody else (…)
- Tampering: Tampering threats involve malicious modification of data or code(…)
- **Repudiation**: An attacker makes a repudiation threat by denying to have performed an action that other parties can neither confirm nor contradict (…)
- **Information Disclosure**: Information disclosure threats involve the exposure of information to individuals who are not supposed to have access to it (…)
- **Denial of Service**: Denial of Service (DoS) attacks deny or degrade service to valid users – for example, by making a Web server temporarily unavailable or unusable (…)
- **Elevation of Privilege**: Elevation of Privilege (EoP) threats often occur when a user gains increased capability, often as an anonymous user who takes advantage of a coding but to gain admin or root capability (…)

Following these six classes, we identify existing threats against particular elements (data flows, data stores and external entities) of our data flow models; for example, data flows face different threats than external entities. Thus, with the help of this methodology we map relevant elements found in our data flow analysis to relevant threats. Last but not least, we relate possible countermeasures provided within SEMIoTICS to specific threats depending on each layer of the architecture shown in Figure 17.

We do not provide a rigorous risk analysis for various reasons. First of all, a detailed risk analysis needs to be performed in an environment when all the processes, software, and every specific technical aspect has been clarified, e.g., exact software versions for libraries. Second, even if we had this information (which is only available towards the end of the project), these results would be only applicable to a single usage scenario. In turn, this hinders the value of this document to transmit a clear message regarding the kind of threats we can address with SEMIoTICS. Instead, as previously mentioned, we consider that a basic scenario including sensitive information allows us to showcase the usefulness of our approaches.

5.2. Use Case Description

We use UC2, i.e., the "Socially Assistive Robotic solution for ambient assisted living" described in the SEMIoTICS usage scenarios and requirement deliverable D2.2.

5.2.1. OVERVIEW OF THE STORYLINE

UC2 provides support for an elderly patient with a mild cognitive impairment. One aspect of the use case is to provide mechanisms to engage the patient in multiple activities to keep him active. Additionally, there are other elements in the patient's home to help him move around his environment. The latter is of utmost importance for elderly patients.
As a natural consequence, the use case also requires features to detect situations when the patient is in distress; for example, if the patient faints, feels sick or falls to the ground. In such situations, the system generates an alarm to a care giver, e.g., a family member, and provides mechanisms to have joint 'telepresence' communication to ensure that the patient is fine. Furthermore, the system also provides the means for a care giver to request a 'telepresence' session with a general practitioner, i.e., medical staff, to ensure that the patient is healthy.

5.2.2. COMPONENTS AND ACTORS

The use case uses the following components:

- **Body Area Network** (**BAN**): comprising a wearable Inertial Measurement Unit (IMU) and mobile smartphone running a dedicated BAN app
- **Robotic Rollator** (**RR**): a semi-autonomous motorised wheeled walking frame for physical support in moving around;
- **Robotic Assistant** (**RA**): in this case Softbank's Pepper, a mid-sized humanoid robot

The actors are the following:

- **Care Recipient** (**Patient**): Person age > 60 or 65 with a slight but noticeable and measurable decline in cognitive abilities, including memory and thinking skills
- **General Practitioner** (**GP**): A medical professional
- **Caregiver** (**CG**): Any relative, partner, friend or neighbour who has a significant personal relationship with, and provides a broad range of assistance for, an older person or an adult with a chronic or disabling condition.

We start the use case analysis with our security assumptions, followed by the data flow diagrams. Then, we analyse the STRIDE threats. Our descriptions are based on the work by Howard and Lipner [23].

5.3. Security Assumptions

The security assumptions on top of which the following threat analysis rely upon are presented below:

- We assume that state-of-the-art cryptography mechanisms are well implemented and therefore cannot be broken by an attacker, i.e., to gain unauthorized access to the data.
- We assume that firewall rules are properly implemented and cannot be bypassed by an attacker
- We assume that an attacker cannot gain unauthorized access to the operating system, or any of the Security Managers or other components developed by SEMIoTICS.

5.4. Data Flow Diagrams

Each data flow diagram contains the following kind components (assets), according to the Security Development Lifecycle:

- A Complex process: depicted by a double circle, shows a logical abstraction to represent a process with multiple operations and interactions.
- A Process: depicted by a regular circle shows an element that performs a single task, Processes reflect software components in many cases.
- External entity: depicted by a square shows something or someone driving the application which is beyond the control of the application being developed, e.g., a user.
- Data Store: Shown by the rhomboid reflects persistent elements such as files or databases.
- Data Flow: Arrows showing that data moves between processes or entities.
- Privilege Boundary: dotted line showing where information flows between lower and higher privilege. In general, these boundaries help to identify locations where additional checks for the information flowing are needed.

Also, we initially depict some use cases through complex processes, in order to drill down into details and de-compose complex processes in atomic processes afterwards.

Regarding our security methodology, we follow the STRIDE methodology based on the data flow diagrams and their different elements. Essentially, this means that we consider different kinds of threats depending on the element type. This is a consequence of using the Security Development lifecycle. Now, we list the threats against each kind of element (also called asset sometimes), which is provided by [23].

- External entity: subject to Spoofing and Repudiation (SR)
- Data flow: subject to Tampering, Information Disclosure, and Denial of Service (TID)
- Data store: subject to Tampering, Information Disclosure, and Denial of Service. Also if the data store is a log, it can be subject to Repudiation (TID+R).
- Process: Spoofing, Tampering, Information disclosure, Denial of Service, Elevation or Privilege (STRIDE).

As previously described, we start with a set of complex processes to provide an overview of the use case and start the threat analysis.

The diagram sown in Figure 25 depicts the complex processes involved in the use case. Particularly, it shows how when a patient walks in his environment, e.g., his apartment, the monitors his behaviour, and simultaneously analyses data to detect possible distress situations. If there is a situation where the patient may be at risk, the care giver is notified. Once the care giver receives an alert, he could decide to start a remote ‘telepresence’ session with the patient. Also, the care giver could request a remote ‘telepresence’ session with a general practitioner.

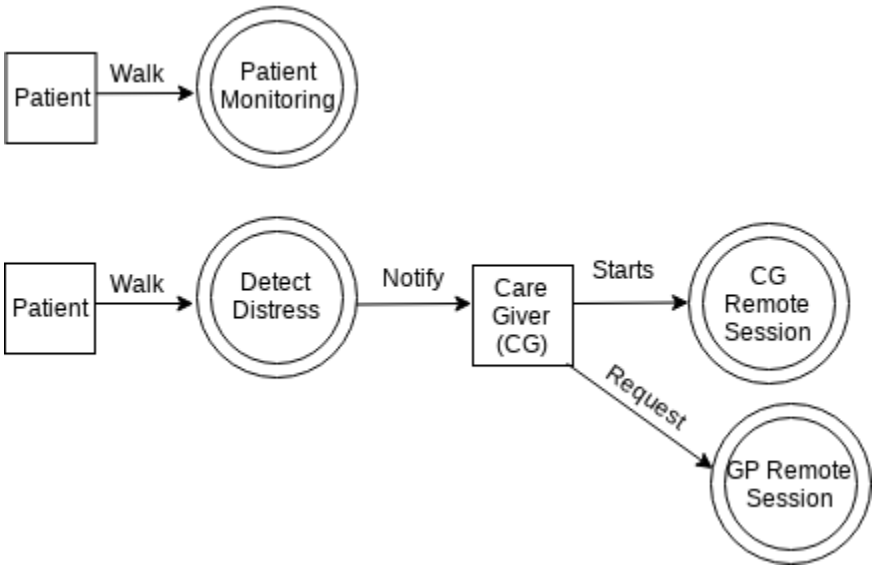


FIGURE 25 COMPLEX PROCESSES FOR UC2

Although the diagram of complex processes shows an overview of the security- and privacy-relevant actions in the use case, they are too course-grained for a threat analysis. Therefore, we decompose them in smaller processes to provide a more detailed view of the components involved and their data flows.

Figure 26 shows each complex process in a more fine-grained fashion to facilitate the analysis.

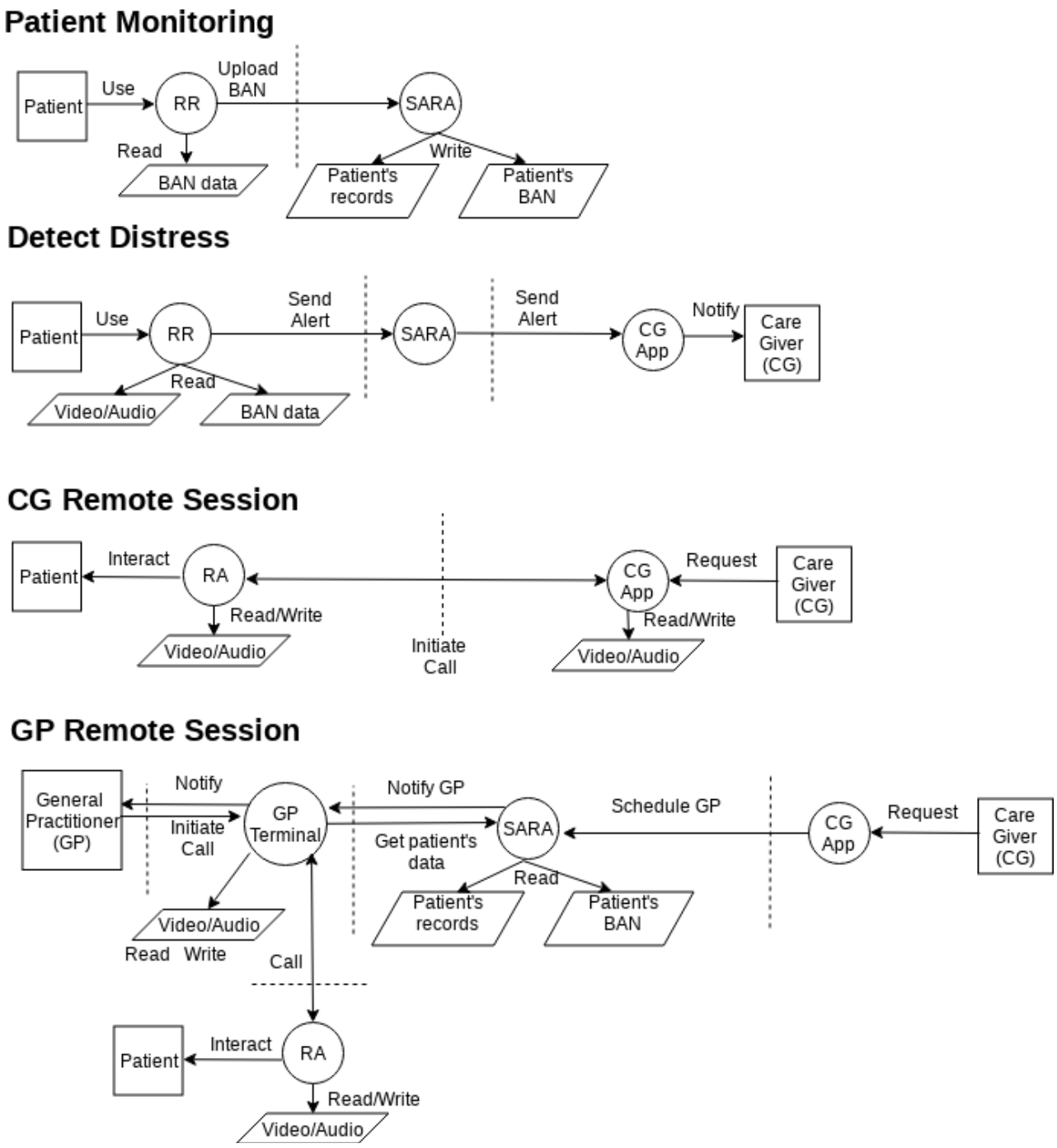


FIGURE 26 DATA FLOW DIAGRAMS FOR UC2

5.4.1. PATIENT MONITORING

This step happens constantly as the patient uses the RR. Particularly, the RR constantly uploads information obtained from the patient's BAN to ensure availability of the data to the proper parties in the cloud.

5.4.2. DETECT DISTRESS

Simultaneously with the patient monitoring, the rollator captures video and Body Area Network (BAN) data from the patient to detect situations when the patient may be in danger, such as irregular reading coming from the body sensors or detecting a fall (which is a big risk, especially for elderly patients). Based on this data, the RR detects that the patient is weak and generates an alert to the SARA cloud. Here, there is a first privilege boundary appears, between the home of the patient and the SARA cloud. Afterwards, there is a second boundary between the SARA cloud and the mobile application executed on the care giver's phone.

5.4.3. CARE GIVER (CG) REMOTE SESSION

In a situation when the care giver is concerned by the patient, he can initiate the 'telepresence' session with the patient. To this end, the CG can use his phone to start a video conference with the Robot Assistant (RA) to interact with the patient. In this scenario, both ends of the communication have 'read and write' access to multimedia information; essentially, this means that they can observe the environment, i.e., camera and microphone but also modify it with the screen and speakers. In this scenario, the main trust boundary appears between the mobile phone application of the care giver and the RA in the environment of the patient.

5.4.4. GENERAL PRACTITIONER (GP) REMOTE SESSION

This scenario involves three actors and starts from the right-hand side of the figure. The CG can request a session between the GP and the patient; here we find the first privilege boundary. Then, when the CG sends a request to the SARA cloud, the cloud notifies the terminal where the GP is connected to the SARA cloud. This is another privilege boundary. Once the GP is notified the SARA cloud provides access to the patient's records as well as existing BAN data reflecting his health status. Once the GP is ready to start the 'telepresence' session, he starts the connection between his terminal and the RA, where we find the last privilege boundary. In the session both, the GP's terminal and the RA require access to microphone, video, speaker and screen of both devices.

5.5. Assets and Threats

This section lists the assets based on their kind to assert based on their type. Also, within each element, we present threats against it. We have selected threats that are relevant to data stores, data flows and external entities with high impact, which can be mitigated by the approaches proposed in this document. For conciseness, we have grouped rows in the table when multiple data stores, flows or entities share the same threat.

5.5.1. DATA STORES

Data Store	Kind of threat	Description	Countermeasure
SARA (Patient's records) SARA (Patient's BAN)	Information Disclosure	A GP could read information from a patient he is not treating	We add a PEP in the SARA cloud to ensure that a GP only access information for a patient when he is treating him or her. For this, an attribute could be added to the patient indicating which doctor is treating him.
SARA (Patient's records) SARA (Patient's BAN)	Information Disclosure	An attacker compromising the SARA cloud could read information of any patient from the patient's record data base	Although this attack requires a highly skilled attacker, it can have high impact. A possible mitigation would be to employ attribute-based encryption to ensure that patient's records and BAN data is encrypted requiring a user with a private key with the attribute role equals to "GP" or "General Practitioner"

RR (BAN Data), RR (Video and Audio) RA (Video and Audio) GP Terminal (Video Audio)	Tampering	An attacker could try to tamper with the information stored the different components storing it.	To counter this threat the RR, RA, GP and SARA shall have proper access control mechanisms to access databases holding sensitive information. In case these mechanisms are not provided by default, a PEP enforcement point could be deployed, e.g., using the side car proxy, to ensure that components providing critical information validate security policies before sending information.
RR (BAN Data), RR (Video and Audio) RA (Video and Audio) GP Terminal (Video Audio)	Information Disclosure	A GP could open a remote "telepresence" session, or attempt to get privacy sensitive information such as the patient's exact location from the BAN data, which poses a privacy risk.	The policy framework can enforce strict policies, restricting access to video and audio streams monitoring the patient as well as other privacy-sensitive information, e.g., the patient's location, in normal system state. However, if a fall is detected the context changes, and the CGs/GPs will be able to retrieve the location for safety purposes. To enforce this change of state and dynamic update of security policies the security and privacy patterns can be used. Also, they can be used to monitor the state of the system and raise alerts for human intervention if needed.

5.5.2. DATA FLOWS

Data Flow Source/Dest	**Kind of threat**	**Description**	**Countermeasure**
SARA/ CG APP *Send Alert*	DoS	DoS for the message could harm the patient if he does not get proper medical attention.	Send messages through two channels (Cellphone and Wifi). SEMIoTICS is currently exploring the possibility to use (Virtual Network Functions) VNFs for this purpose.
RR/SARA *Upload BAN*	Information Disclosure	If an attacker manipulates the RR, it could store the BAN data under another user	The application could implement policies to ensure that the RR is authenticated as an OAuth2 client. Also, when the client writes to a patient, it can ensure that the owner of that RR is indeed the patient who owns the heath record too.

CG App/ SARA *Schedule GP*	Information Disclosure	An attacker could impersonate the CG App to generate a request to schedule a GP. This would let a GP violate the user's privacy if he starts a call to gain access to the patient's video stream.	To counter this threat, the application can register CG App as an OAuth2 client in the backend. Furthermore, if the GP application instance is associated with the patient, an attacker cannot authenticate on behalf of the CG app without compromising the phone of the caregiver.
SARA/ CG APP *Send Alert*	Tampering	An attacker could attack the underlying SDN network to redirect the traffic to malicious destinations by adding flow rules to SDN switches	SDN switches and controllers' communication could be encrypted to avoid third-party manipulation of network data flow. Also authentication should be used to avoid any kind of tempering using the NBI interfaces of the SDN controllers

5.5.3. EXTERNAL ENTITIES

Entity	**Kind of threat**	**Description**	**Countermeasure**
General Practicioner (GP)	Spoofing	An attacker could impersonate a GP during a session	Even though this threat is unlikely, as it needs physical access to the GP terminal, it can have high impact. To prevent the threat, the application could protect the SARA cloud service by relying on the authentication services provided by the Security Manager.

5.5.4. PROCESSES

Entity	**Kind of threat**	**Description**	**Countermeasure**

RR, RA, SARA, GP Terminal	Tampering	In the case of a vulnerability, an attacker could install malicious software in the rollator, the robotic assistant or any other data source to perform a more complex attack, e.g., install a botnet client in the devices to launch a DoS attack against third parties.	On the one hand, the machine-learning approaches presented before could be used to detect malicious behaviour, e.g., the botnet. Additionally, the SDN functionality could be used to redirect malicious traffic to a honeypot and analyse the attacker's behaviour. In this way, threat intelligence could be collected to prevent similar attacks in the future.

6. REQUIREMENTS AND KPIS

6.1. Overview

The project collected and categorised a number of requirements in Deliverable D2.2, D2.3 and D2.4. In section 5.2 we discuss those requirements that are security- and privacy-related and mention how or where they are addressed. The final evaluation of the KPI K.4.6 (i.e. the KPI that is related to Task 4.5 as provided in D5.1) has been achieved, as more than 3 new components have been developed. As this deliverable has shown they together achieve a real-time and reliable management of privacy and security across all layers. The KPI is discussed in detail in section 5.3

6.2. Security and Privacy-related requirements

This section lists the requirements that SEMIoTICS elicited following the extensive security and privacy analysis of IoT applications done in T.4.5 and taking into account the requirements gathered in the project. The full list can be found in D2.2, with the focus being strictly on security and privacy, in Task 4.5 we will only discuss those that directly relate to security and privacy in the following. Here it should be noted that the deferred requirements are mostly non-functional requirements, like legally motivated GDPR-compliance as e.g. minimal data being stored. Such non-functional requirements have been elicited with the help of domain knowledge experts in the respective use-cases, e.g. health-care, thus when checking the fulfilment the use-case must evaluate this. Therefore, the checking has been deferred to D5.5 or D5.10 (final version of D5.5) respectively for the health-care domain. The same holds true for the other use cases. The provision and check of functionality of the technical mechanisms that allow to achieve these requirements, e.g. ABE-implementation and the respective services in the field- and backend-layer, have been conducted and documented (e.g. unit tests) in the software development related deliverables, please see D5.8's Section on performance testing and KPI validation.

IoT Security and Privacy Requirements		**Evaluation**	**Reference**	**Status**
Req. ID	**Description**			
R.S.1	The confidentiality of all network communication MUST be protected using state-of-the-art mechanisms.	SDN connection with ovs switches by enabling SSL, + Communication between Pattern Orchestrator and Pattern Engines with SSL	Section 3.2.2, Section 3.4.2, D3.7	achieved
R.S.2	Authentication and authorization of the stakeholders MUST be enforced by the Network controller, e.g. through access and role-based lists for different levels of function granularities (overlay, customized access to service, QoS manipulation, etc.)	The SDN Security Manager can provide authorization to the users entering the controller	Section 3.3.1, D3.2	achieved

R.S.3	Sensors SHALL be identifiable (e.g. by a TPM module/smartcard) and authenticated by the gateway.	/	Will be discussed in D5.5	deferred
R.S.4	All components from gateway, via SDN Controller, to cloud platforms and their users MUST authenticate mutually.	Cross-layer deployment of Security Managers	Section 3.2, D3.7	achieved
R.S.5	Before sensitive data is being transmitted, the respective components SHALL be authenticated as defined by requirements R.S.3 and R.S.4	TLS-based client authentication on sensors or gateway + at the Backend (eg MQTT Brokers)	Section 3.2.2	achieved
R.S.6	Sensors SHALL be able to encrypt the data they generate, i.e. their CPU and memory SHALL be sufficient to perform these cryptographic operations.	Gateway encryption	Section 3.4.2	achieved
R.S.7	The negotiation interface of the SDN Controller SHALL be secure against network-based attacks	SDN Security Manager can provide such capability	Section 3.3.1	achieved
R.S.8	The honeypot SHALL be a dedicated or virtual server.	The honeypot is instantiated as virtual service network function in Proxmox and OpenStack as a part of SFC	Section 2.2.7, Section 4.1.1, D3.8, D5.5	achieved
R.S.9	The honeypot SHALL run a Linux based, hardened operating system.	The EWIS honeypot is installed on Ubuntu OS	Section 2.2.7, Section 4.1.1, D3.8, D5.5	achieved
R.S.10	The honeypot SHALL have hardware with sufficient computational capabilities (based on traffic).	The required honeypot computational capabilities are defined during its instantiation	Section 2.2.7, Section 4.1.1, D3.8, D5.5	achieved

R.S.11	The honeypot MUST have adequate bandwidth capacity to provide sufficient traffic for logging.	The required honeypot network capabilities are defined during its instantiation	Section 2.2.7, Section 4.1.1, D3.8, D5.5	achieved
R.S.12	The honeypot MUST have networking capabilities for redirecting traffic and mirroring ports	The required honeypot network capabilities are defined during its instantiation	Section 2.2.7, Section 4.1.1, D3.8, D5.5	achieved
R.S.13	The honeypot SHALL be able to execute software for capturing the suspicious traffic for further processing.	The required honeypot resource capabilities are defined during its instantiation	Section 2.2.7, Section 4.1.1, D3.8, D5.5	achieved
R.S.14	The honeypot SHALL provide a data repository and backend processing.	The required honeypot storage capabilities are defined during its instantiation	Section 2.2.7, Section 4.1.1, D3.8, D5.5	achieved
R.S.15	The honeypot MUST run on a flexible SDN infrastructure (probably open source).	The honeypot is chained with additional service functions and managed by the SDN controller	Section 2.2.7, Section 4.1.1, D3.8, D5.5	achieved
R.S.16	The honeypot system MUST act as a dummy SDN component (controller or switch).	The honeynet extend the capability of honeypot to include dummy components	Section 2.2.7, Section 4.1.1, D3.8, D5.5	achieved
R.S.17	There MUST be an interface between the network controller and the network administrators for the designation of the applications' permissions.	SDN Security Manager	Section 3.2, D3.7	achieved

R.S.18	All network functions SHALL be mapped to application permissions.	Network Functions should be stithce in chains where the required access control lists can give permissions to specific applications	Section 2.2.7, D3.7, D3.8	achieved
R.S.19	Policy conflicts among the controller applications SHALL be identified and resolved.	SDN Security Manager in the network layer can handle this	Section 3.2, D3.7	achieved
R.S.20	Cloud platforms MUST be protected by a firewall against network-based attacks.	/	Will be discussed in D5.5	deferred
R.P.1	The collection of raw data MUST be minimized.	/	Will be discussed in D5.5	deferred
R.P.2	The data volume that is collected or requested by an IoT application MUST be minimized (e.g. minimize sampling rate, amount of data, recording duration, different parameters).	/	Will be discussed in D5.5	deferred
R.P.3	Storage of data MUST be minimized.	/	Will be discussed in D5.5	deferred
R.P.4	A short data retention period MUST be enforced and maintaining data for longer than necessary avoided.	/	Will be discussed in D5.5	deferred
R.P.5	As much data as possible MUST be processed at the edge in order to hide data sources and not reveal user related information to adversaries (e.g. user's location).	With the help of local analytics this can be facilitated	Will be discussed in D5.5	deferred

R.P.6	Data MUST be anonymized wherever possible by removing the personally identifiable information in order to decrease the risk of unintended disclosure.	/	Will be discussed in D5.5	deferred
R.P.7	Data granularity MUST be reduced wherever possible, e.g. disseminate a location-related information (i.e. area) and not the exact address	/	Will be discussed in D5.5	deferred
R.P.8	Data MUST be stored in encrypted form.	ABE as described in Section 3.2.1.4 or other encryption can be of aid	Will be discussed in D5.5	deferred
R.P.9	Repeated querying for specific data by applications, services, or users that are not intent to act in this manner SHALL be blocked.	The Policy Enforcement Points (PEP) will be seeing all requests and can thus detect and mitigate such attempts	Will be discussed in D5.5	deferred
R.P.10	Wherever possible, information over groups of attributes or groups of individuals SHALL be aggregated	/	Will be discussed in D5.5	deferred
R.P.11	The data principal SHALL be sufficiently informed regarding which data are collected, processed, and disseminated, and for what purposes	/	Will be discussed in D5.5	deferred
R.P.12	During all communication and processing phases logging MUST be performed to enable the examination that the system is operating as promised	/	Will be discussed in D5.5	deferred

R.P.13	The user SHALL be able to control the privacy mechanisms (i.e. redemption period, data granularity and dissemination, and anonymization technique)	/	Will be discussed in D5.5	deferred
R.GSP.1	The Intrusion Detection System (IDS) MUST capture and process suspicious traffic.	The IDS can be installed to satisfy the requirement for Service Function Chaining	Section 2.2.7, Section 4.1.2, And D3.8	achieved
R.GSP.2	Accredited and certified Computer Emergency Report Team (CERT) MAY get informed about an occurring cyber incident (e.g. DoS).	The CERT team can support incident detection and response in SEMIoTICS industrial environment such as in UC2.	Section 4.2, D2.2	achieved
R.GSP.3	IoT gateway SHALL be able to estimate abnormal detection based on (un)-supervised model.	dedicated local analytics component at gateway level : IHES Supervisor component takes care of this together with dependability pattern designed in UC3 to detect faulty devices	Will be discussed in D5.6	deferred
R.UC1.8	Semantic and robust bootstrapping/registration of IIoT sensors and actuators with IIoT gateway MUST be supported.	/	Will be discussed in D5.4	deferred
R.UC1.12	Standardized semantic models for semantic-based engineering and IIoT applications MUST be utilized.	/	Will be discussed in D5.4	deferred

6.3. Evaluation of Security and Privacy-related KPI 4.6

SEMIoTICS has developed many components that enable the framework to increase the security and privacy of IIoT applications as described in this deliverable. Among those are the components related to pattern-based enforcement (Pattern Orchestrator and Pattern-Engine), the identity-based attribute based encryption (ABE-KM and Encryption-Decryption Modules), as well as those for the Intrusion-Detection-System (Honey-Pots in the Network layer) and finally the identity-based management of authorisations enabled by the interplay of the security manager's components (PEP, AEP, PDP, PAP and IDM). The following table provides an overview of the more than 10 components.

Number	Component	Area of Application	Position within the Architecture	Cross-Layer interaction
1	Pattern Orchestrator	Pattern-based enforcement and Monitoring	Backend	yes
2	Pattern-Engine		Backend	yes
3			Network	yes
4	Identity Management	Policy-based enforcement of access control and rights management (authorisation) based on Identity-based authentication	Backend	Yes, indirectly[13]
5	Policy Decision Point		Backend	yes
6			Field	
7	Policy Enforcement Point		Application	yes
8			Field	
9	Authentication Enforcement Point		Application	no
10	Attribute-based-Encryption (ABE) Key Management	Enforcement of confidentiality at rest and in transit beyond the SEMIoTICS framework	Backend	yes
11	Attribute-based-Encryption and Decryption		Application	
12			Field	
13	Honeypot	Security enforcement and monitoring of SDN	Network	no
14	Security Manager in Network Layer			

TABLE 1 OVERVIEW OF COMPONENTS

In total these are fourteen novel mechanisms that SEMIoTICS not only designed, implemented, tested and deployed across the different layers but also orchestrated them to make them interplay smoothly to achieve an overall security and privacy boost for IoT and IIoT deployments. All individual components can be tailored to adapt to different scenarios and specific application-driven demands. Together they achieve SEMIoTICS highly configurable and adaptable attribute- and pattern-based security and privacy enforcement. In the case of KPI-4.6, which covers non-functional requirements and is related to aspects which are not related to actual technical

implementations, the fulfilment is easily to be seen as more than 3 new components to increase the security as well as allowing to manage the security have been developed. This deliverable extensively describes how these mechanisms interplay and how they can be facilitated.

KPI ID	Description	Evaluation	Reference	Status
KPI-4.6	Development of a minimum of 3 new security mechanisms/controls enabling the secure management of smart devices and sensors over programmable industrial networks	Security Manager in Backend-, Network- and Field layer and their essential internal components	This Deliverable and Status of the Implementation as presented in Deliverables D4.6, D4.7, and D4.13	achieved

7. CONCLUSION

This deliverable has presented an overview of the various mechanisms improving security and privacy that were developed within the SEMIoTICS project and how they all combined and interwoven into the SEMIoTICS architectural framework achieve a constantly and how they help to achieve security and privacy requirements at key points throughout the SEMIoTICS architecture.

This deliverable shows that the KPI related to the objective of developing new security mechanisms has been achieved by the many components newly designed, adapted and fitted for the SEMIoTICS' overall security and privacy framework. Thus, the KPI-4.6 has been achieved. This contributes largely to SEMIoTICS' Objective 4: Development of core mechanisms for multi-layered embedded intelligence, IoT application adaptation, learning and evolution, and end-to-end security, privacy, accountability and user control.

With the mechanisms that we have developed (see Table 1) and described in this deliverable, SEMIoTICS has the tools ready to support horizontal (cross IoT platform) and vertical adaptation of security and privacy requirements of industrial IoT applications. All of which is driven by SPDI patterns, which control via attribute-based policies the access to services, devices and data. The new mechanisms we developed provide end-to-end authentication and are context-aware through updatable policies, e.g. if a pre-defined situation, like a medical emergency is detected by local embedded intelligence, tight and GDPR-conformant restrictions will get dynamically lifted for the time of the emergency only. With logging and the monitoring infrastructure SEMIoTICS checks compliance at runtime to ensure that the user controls the allowed actions (data production, data access, data processing including transfer and data storage) and checks that they are always subject to appropriate user authorization rights.

This deliverable also highlights the real-time management of cross-layer security and privacy of SEMIoTICS, namely that all components interplay across different layers or focus specifically on security and privacy problems specific to the layer in IoT deployments. This achieves security overall. Particularly, we have described how identities are managed by our Identity Management component, the authentication capabilities offered to end users, the support for policies based on authentication metadata provided by semantic components in the project, and our generic attribute-based security framework for policy decisions. In addition to this, the deliverable presented how SEMIoTICS plans to support attribute-based encryption for enhanced privacy, how keys will be managed and consistently mapped to our generic attribute-based solution for policy evaluations. Moreover, we presented the support for security applied at the SDN layer and the SPDI patterns deployed in SEMIoTICS.

In addition to the above, we used a scenario inspired on one of SEMIoTICS' use cases to highlight our design choices and components developed within this task. An attribute-based approach in the security manager was chosen because it has been shown that attribute-based access control can support previously used mechanisms, such as role-based access control [24]. This allows SEMIoTICS to support role-based access control decisions, e.g., at the SDN controller level, while keeping increased flexibility for more complex scenarios requiring validation of particular attributes. An example of the latter is present in the description of the motivating IoT application when calls should be generated by a caregiver of a particular patient. Particularly, this cannot be achieved by a simple role-based access control decision because there is no system-wide role to allow a user to start telepresence calls with a particular patient. Instead, the application must validate that a caregiver is indeed associated with a given patient. Thus, this property can be best asserted, avoiding the so-called role explosion, by mapping attributes of a caregiver to a patient, e.g., by referencing the patient in the caregiver representation stored in the identity management.

Updated from the previous draft version (D4.5) this final report evaluated positively 20 of 38 requirements. The remaining 18 are use case specific and their evaluation has been deferred to later deliverables. The list of requirements was previously gathered and consolidated in D2.2, D2.3 and D2.4. This document thus shows how SEMIoTICS' security and privacy-related components can be flexibly tailored to achieve generic as well as more specific requirements directly or aid to address them.

Finally, the implementation of these components is also nearly completed by now and integration-tests are underway, thus the components are expected to be used in the applications from our divers scenarios to meet not

only the generic, but also help to address the very specific security and privacy requirements of each individual use case. For 18 use-case-specific requirements an evaluation will be done in WP5 deliverables before the final prototype is ready at the end of the project.

8. REFERENCES

[1] S. Kaebisch , T. Kamiya, M. McCool and V. Charpenay, “Security Vocabulary Definitions - Web of Things (WoT) Thing Description,” 09 May 2019. [Online]. Available: https://w3c.github.io/wot-thing-description/#sec-security-vocabulary-definition. [Accessed 14 May 2019].

[2] N. Broberg and D. Sands, “Paralocks – Role-Based Information Flow Control and,” in *Proceedings of the 37th annual ACM SIGPLAN-SIGACT symposium on*, Madrid, 2010.

[3] J. D. Parra Rodriguez, “A Generic Lightweight and Scalable Access Control Framework for IoT Gateways,” in *In Information Security Theory and Practice (WISTP)*, Brussels, 2018.

[4] M. G. Solomon, V. Sunderam and L. Xiong, “Towards secure cloud database with fine-grained access control,” in *Data and Applications Security and Privacy XXVIII*, Berlin, 2014.

[5] A. Sahai and B. Waters, “Fuzzy Identity-based Encryption,” in *Proceedings of the 24th Annual International Conference on Theory and Applications of Cryptographic Techniques*, Berlin, 2005.

[6] C. Wang and J. Luo , “An Efficient Key-Policy Attribute-Based Encryption Scheme with Constant Ciphertext Length,” in *Mathematical Problems in Engineering*, 2013.

[7] V. Goyal, O. Pandey, A. Sahai and B. Waters, “Attribute-based Encryption for Fine-grained Access Control of Encrypted Data,” in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, New Yotk, 2006.

[8] M. Chase and S. S. Chow, “Improving Privacy and Security in Multi-Authority Attribute-Based Encryption,” in *Proceedings of the 16th ACM conference on Computer and communications security* , New York, 2009.

[9] Q. Wang, L. Peng, H. Xiong, J. Sun and Z. Qin, “Ciphertext-Policy Attribute-Based Encryption With Delegated Equality Test in Cloud Computing,” *IEEE Access,* vol. 6, pp. 760-771, 2018.

[10] Zeutro LLC, “The OpenABE Design Document Version 1.0,” Zeutro LLC, 2018.

[11] D. Kreutz, F. M. Ramos and P. Verissimo, “Towards Secure and Dependable Software-defined Networks,” in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, New York, 2013.

[12] A. Bremler-Barr, Y. Harchol, D. Hay, and Y. Koral, “Deep Packet Inspection as a Service,” in *10th ACM International on Conference on Emerging Networking Experiments and Technologies*, 2014.

[13] “http://www.ntop.org/products/deep-packet-inspection/ndpi/,” [Online].

[14] C. Tzagkarakis, N. Petroulakis and S. Ioannidis, “Botnet Attack Detection at the IoT Edge Based on Sparse Representation,” in *Global Internet of Things Summit (GIoTS)*, Aarhus, Denmark, 2019.

[15] Microsoft, “Sidecar pattern,” Microsoft, 23 June 2017. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/patterns/sidecar. [Accessed 14 May 2019].

[16] X. Zhang, H. M. Heys and C. Li, “Energy Efficiency of Encryption Schemes Applied to Wireless Sensor Networks,” *Security and Communication Networks,* vol. 5, pp. 789-808, 2012.

[17] C. Manifavas, G. Hatzivasilis, K. Fysarakis and K. Rantos, “Lightweight Cryptography for Embedded Systems; A Comparative Analysis,” in *Data Privacy Management and Autonomous Spontaneous Security DPM*, Berlin, 2013.

[18] G. Hatzivasilis, F. Konstantinos, I. Papaefstathiou and C. Manifavas, “A Review of Lightweight Block Ciphers,” *Cryptographic Engineering ,* vol. 8, no. 2, pp. 141-184, 2018.

[19] C. Manifavas, G. Hatzivasilis, K. Fysarakis and Y. Papaefstathiou, “Security and Communication Networks,” *Security and Communication Networks,* vol. 9, no. 10, pp. 1226-1246, 2016.

[20] C. L. Forgy, “Rete: a fast algorithm for the many pattern/many object pattern match problem,” *Artificial Intelligence,* vol. 19, pp. 17-37, 1982.

[21] A. S. Thuluva, A. Broering, G. P. Medagoda, H. Don, D. Anicic and J. Seeger, “Recipes for IoT Applications,” in *Proceedings of the Seventh International Conference on the Internet of Things*, Linz, Austria, 2017.

[22] J. Seeger, R. A. Deshmukh and A. Broring, “Running Distributed and Dynamic IoT Choreographies,” in *2018 Global Internet of Things Summit (GIoTS)*, Bilbao, Spain, 2018.

[23] H. Howard and S. Lipner, The Security Development Lifecycle, Washington: Microsoft Press, 2006.

[24] V. Hu, K. Scarfone, R. Kuhn and K. Sandlin, “Guide to Attribute Based Access Control (ABAC) Definition and Considerations DRAFT NIST Special Publication 800-162 Guide to Attribute Based Access Control (ABAC) Definition and Considerations,” National Institute of Standards and Technology (NIST), 2013.