SEMIoTICS

Deliverable D4.5
SEMIoTICS Security and Privacy Mechanisms

Deliverable release date	Date
Authors	1. Juan David Parra, Tobias Marktscheffel (UP) 2. Konstantinos Fysarakis, Iason Somarakis, Konstantina Koloutsou (STS), 3. Othonas Soultatos, Eftychia Lakka, Nikolaos Petroulakis (FORTH), 4. Łukasz Ciechomski, Michał Rubaj (BS)
Responsible person	Juan David Parra (UP)
Reviewed by	ST, FORTH, STS
Approved by	PTC Members PCC Members
Status of the Document	Final
Version	1.0
Dissemination level	Public

Contents

Acronym	Definition
DoS	Denial of Service
DPI	Deep Packet Inspection
DNF	Disjunctive Normal Form
EoP	Elevation of Privilege
HTTP	HyperText 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
WoT	Web of Things

1 INTRODUCTION

This deliverable is the first output of **Task 4.5 – End-to-End Security and Privacy**, and as such, its purpose is to provide the design of mechanisms related to entities' identification, key management, and Security, Privacy, Dependability, and Interoperability (SPDI) patterns to achieve end-to-end security and privacy properties in an IoT system.

First of all, this document covers the building blocks used to ensure security and privacy and how separate modules interact with each other. Particularly, we present the identity management, authentication, SND security, and SPDI patterns and how they can generically help IoT and IIoT applications deployed in SEMIoTICS to tackle particular security and privacy-related needs.

Also, this deliverable presents a fine-grained architecture for the security-relevant mechanisms previously described. For greater clarity, we extend our design description with a particular scenario, based on one of the project's use cases. In turn, this approach allows us to demonstrate the applicability of our approaches to an IoT system and the advantages of our security and privacy-related solutions in a concrete scenario.

This document is structured as follows. We describe the design of mechanisms enabling end-to-end security and privacy in section 2. Specifically, we explain the approaches followed by the backend Security Manager, the SDN controller, an approach to place multiple Policy Enforcement Points across SEMIoTICS, and the use of Privacy and Security patterns.

Afterwards, we show the interaction across the aforementioned components in the architecture for security and privacy used in SEMIoTICS as part of Section 3. Last but 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 4. 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 D2.4. Finally, we conclude and present future work in Section 0.

This report is the first draft documentation for the end-to-end security efforts. However, the final version of the work described here will be presented in D4.12; furthermore, D4.12 will report on any relevant updates or additions to our approach. Moreover, the first implementation of these components is part of milestone 2 of the implementation of the SEMIoTICS backend APIs (Task 4.6) which will be completed in the 23rd month of the project, the latest.

2 SECURITY DESIGN

We describe now the goals of our 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.

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 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]. The framework will be used to authenticate users, perform identity management and policy evaluations. However, it was initially designed for an IoT gateway; therefore, it must be 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; specifically, it supports only the evaluation of a security policy when a user accesses an entity, e.g., a device. However, SEMIoTICS requires the possibility 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 will be extended to support the attribute-based encryption key management and other mechanisms needed for the decentralization of the access control in SEMIoTICS. Now, we will discuss the building blocks needed to realise the security and privacy approach proposed by the project. We start by describing the backend Security Manager and its modules. Specially, 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.

[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

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-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.

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., SIMANTIC S7 controller. Also, in the SARA use case (UC2), the security from protocols such as Bluetooth, Zigbee and the LwM2M[3] bootstrapping protocol will be used. Withal, from a more abstract point of view, the semantic interactions in SEMIoTICS will rely on the Web of Things protocol, which also supports authentication.

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.
- CertSecurityScheme: uses certificate-based asymmetric keys with X 509 V3 certificates.

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

- 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 COMPARISSON 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 1, 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[4]. 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.

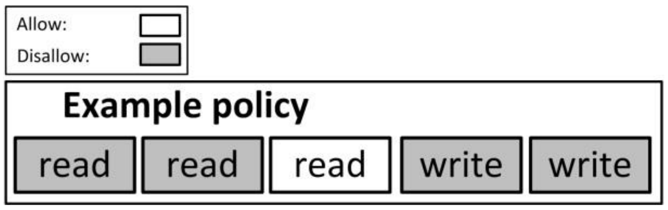


FIGURE 3 POLICY BLOCKS [3]

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.

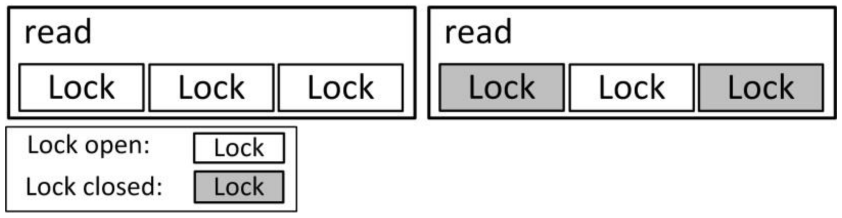


FIGURE 4 LOCK MECHANISM INSIDE A BLOCK [3]

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

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

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.
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**Error! Reference source not found.** 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. We cover first an overview, and then we explain both ways to enforce security policies, 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, i.e 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 (i.e 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.

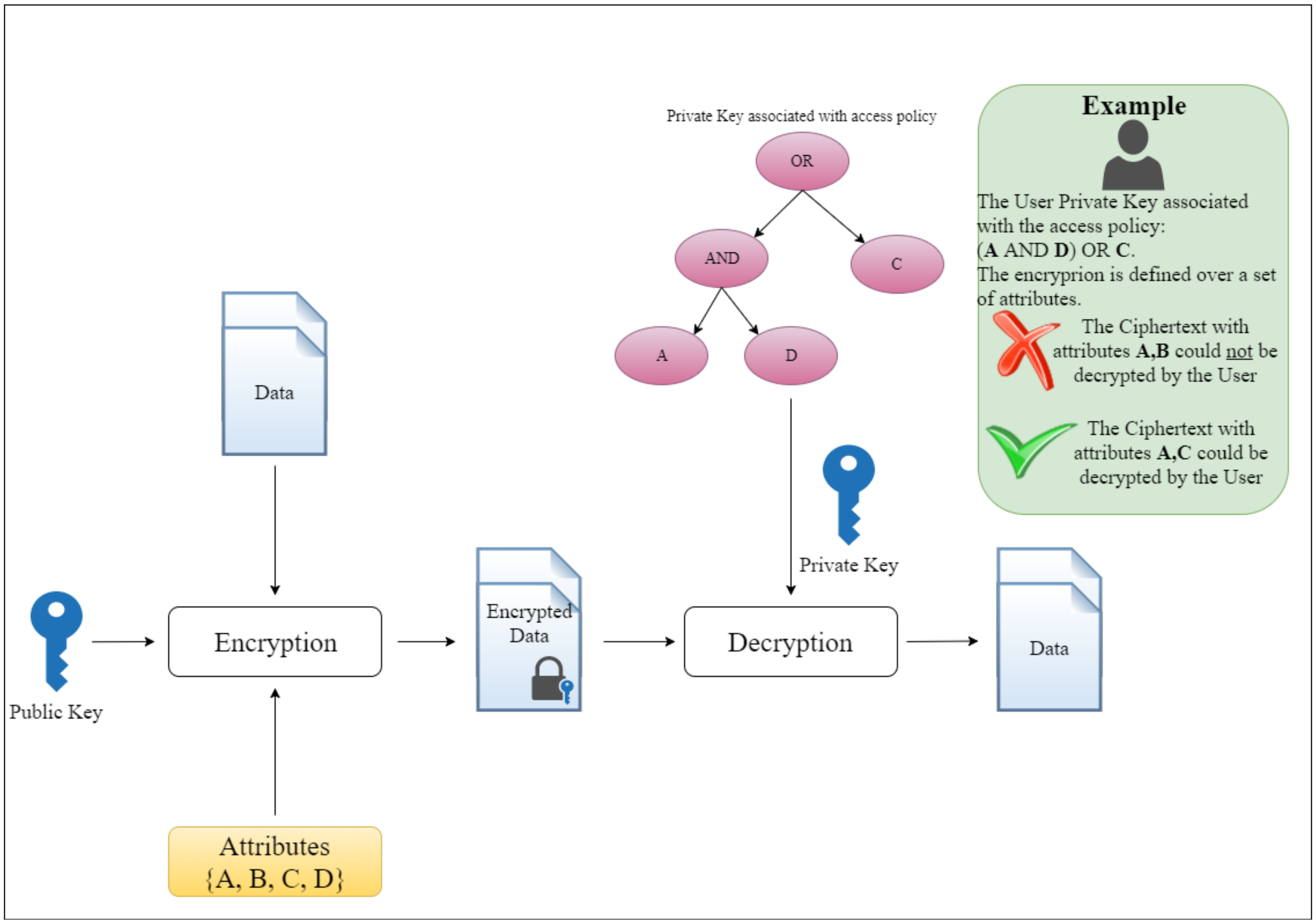

Private Key associated with access policy
OR
AND
C
A
D
Example
The User Private Key associated with the access policy:
(A AND D) OR C.
The encryprion is defined over a set of attributes.
The Ciphertext with attributes A,B could not be decrypted by the User
The Ciphertext with attributes A,C could be decrypted by the User
Data
Private Key
Public Key
Encryption
Encrypted Data
Decryption
Data
Attributes {A, B, C, D}

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.

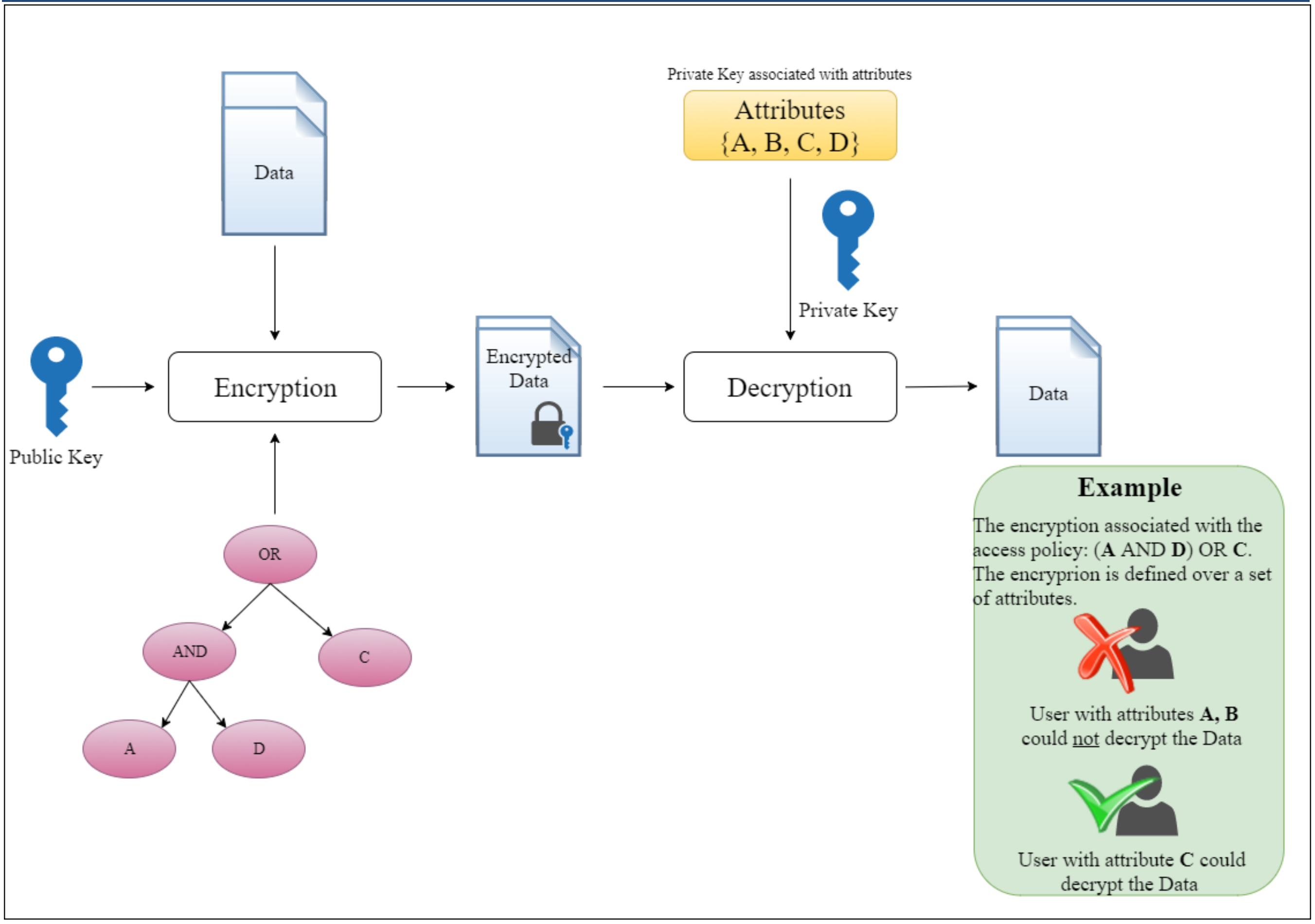

Data
Private Key associated with attributes
Attributes {A, B, C, D}
Private Key
Public Key
Encryption
Encrypted Data
Decryption
Data
OR
AND
C
A
D
Example
The encryption associated with the access policy: (A AND D) OR C.
The encryprion is defined over a set of attributes.
User with attributes A, B could not decrypt the Data
User with attribute C could decrypt the Data

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.

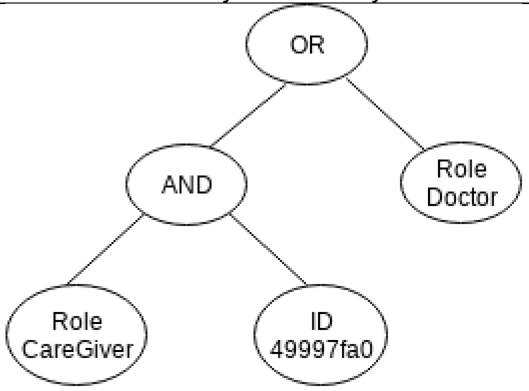
2.2.6 KEY MANAGEMENT

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"] }] }`	

```
{
        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.

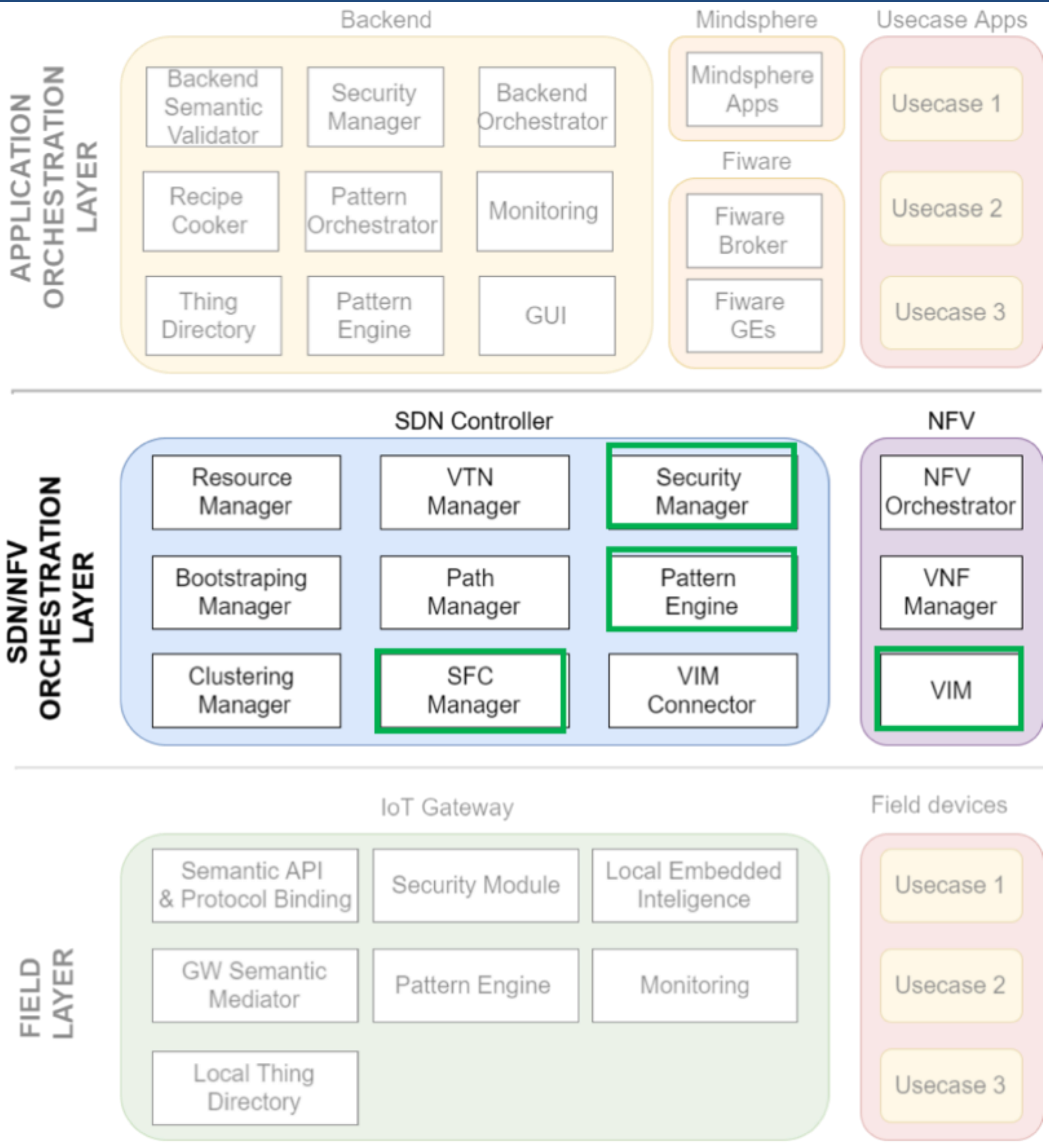


FIGURE 11 SECURITY-RELATED COMPONENTS IN THE SDN CONTROLLER

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.

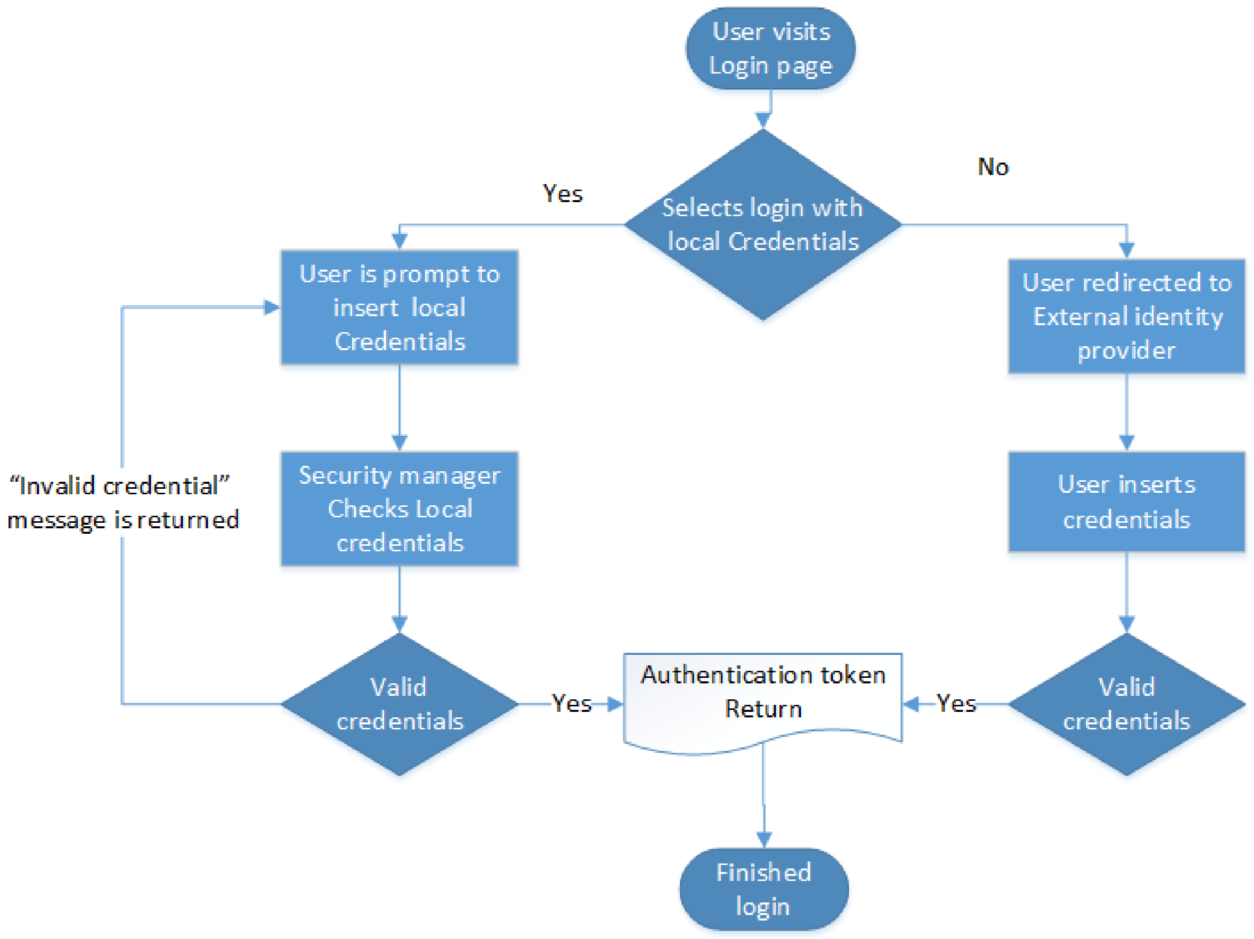

User visits Login page
Selects login with local Credentials
Yes
No
User is prompt to insert local Credentials
User redirected to External identity provider
"Invalid credential" message is returned
Security manager Checks Local credentials
User inserts credentials
Valid credentials
Yes
Authentication token Return
Yes
Valid credentials
Finished login

FIGURE 12 LOGIN PHASE WITH THE SECURITY MANAGER

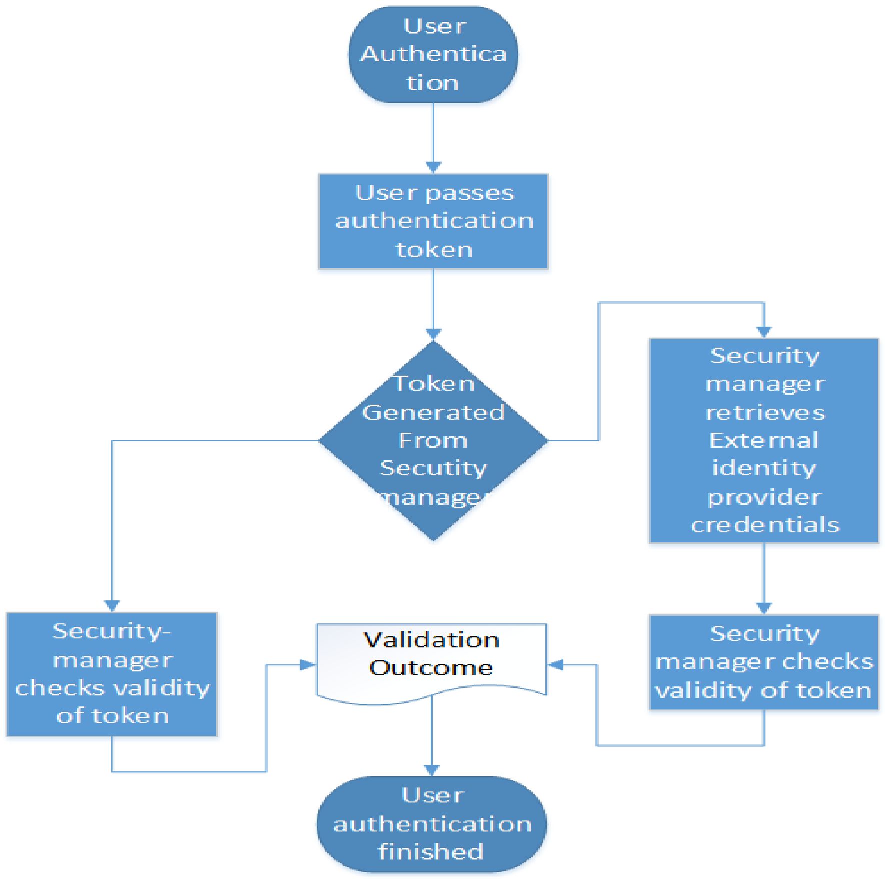

User Authentication
User passes authentication token
Token Generated From Secutity manager
Security manager retrieves External identity provider credentials
Security-manager checks validity of token
Validation Outcome
Security manager checks validity of token
User authentication finished

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.

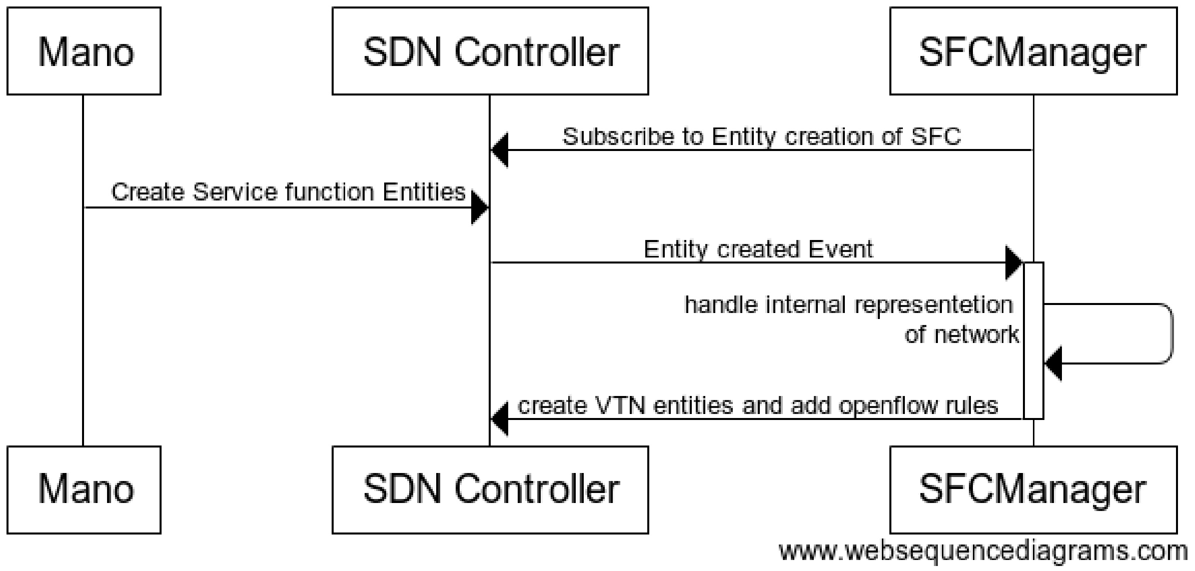

Mano
SDN Controller
SFCManager
Subscribe to Entity creation of SFC
Create Service function Entities
Entity created Event
handle internal representetion of network
create VTN entities and add openflow rules
Mano
SDN Controller
SFCManager
www.websequencediagrams.com

FIGURE 14: SERVICE FUNCTION CHAINING SEQUENCE

A more detailed description of the respective modules is given in D2.4, D3.1, D3.2, D3.4 and D4.1.

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 D4.2. Therefore, we recommend the reader to go to sections 4.2, 4.3, 4.4 in D4.2 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 will 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 is 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.

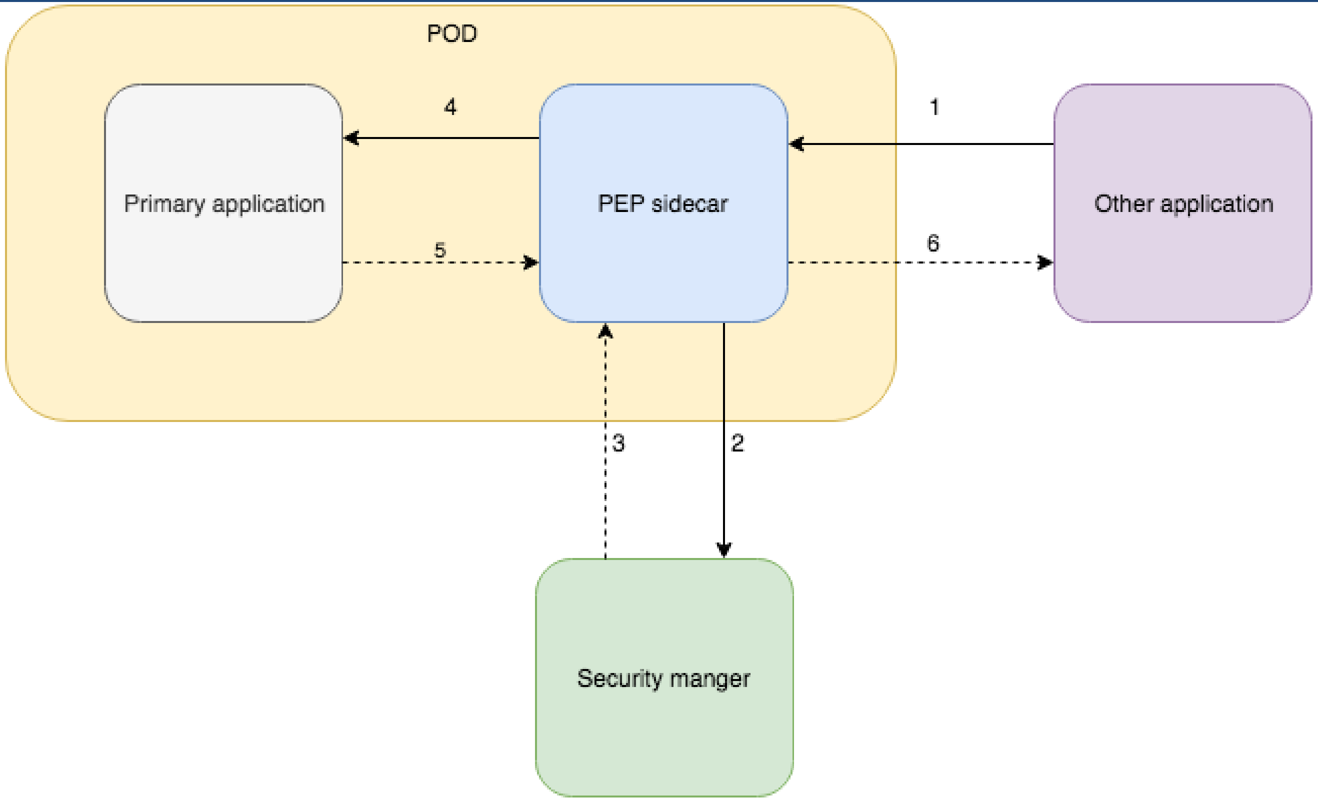


FIGURE 15 COMMUNICATION FLOW USING PEP

2.2.10 SECURITY AND PRIVACY PATTERNS

SEMIoTICS adopts a pattern-driven approach, whereby machine interpretable patterns encode horizontal and vertical ways of composing parts of or 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).

While this pattern-driven approach is presented in detail within deliverable D4.1 ("SEMIoTICS SPDI Patterns (first draft)"), along with a first set of SPDI patterns, some key aspects are briefly re-iterated here for completeness.

2.2.10.1 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. 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, instantiated versions of the abovementioned system model are transformed into Drools[5] 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 [16], 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 [17] [18]. 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 defined orchestrations happens through the deployment of pattern engines on all layers of the architecture, as is detailed in the next subsection.

2.2.10.2 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).

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

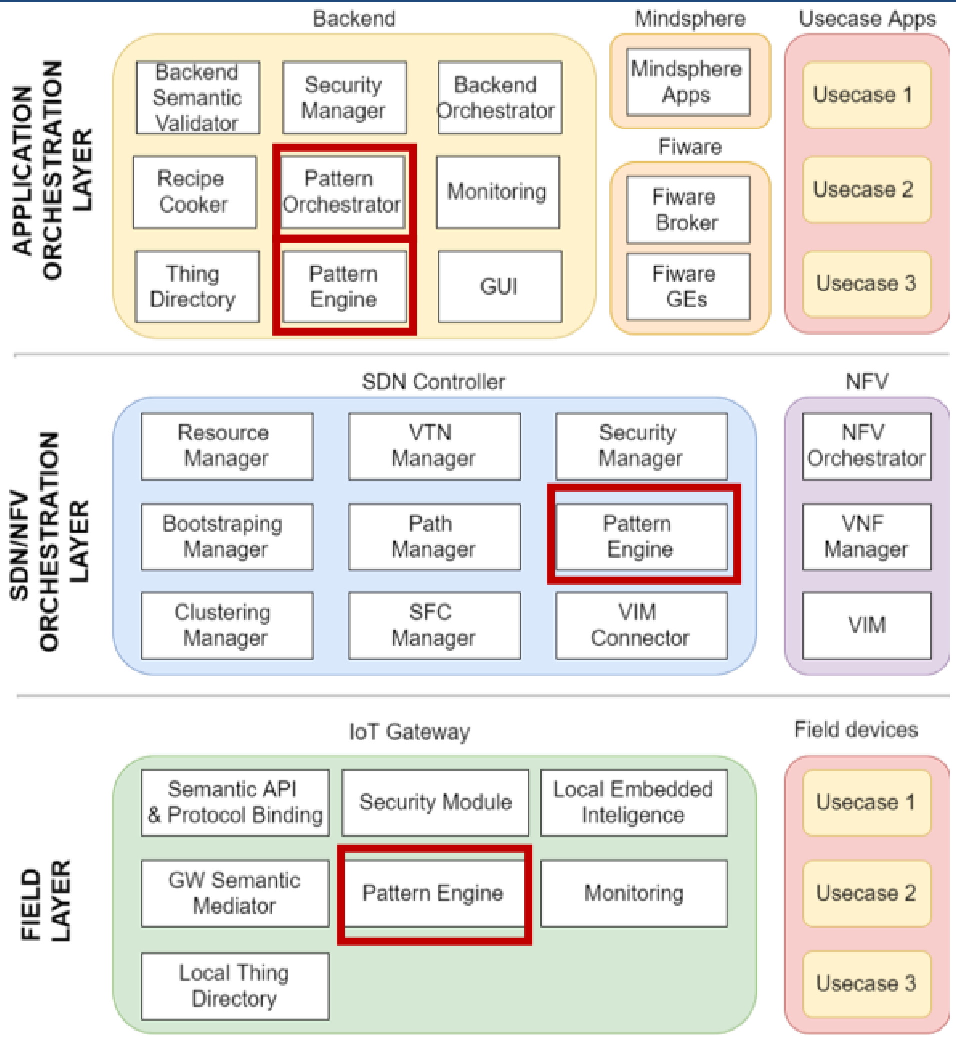


FIGURE 16. PATTERN MODULES WITHIN THE SEMIOTICS ARCHITECTURE

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 Modules (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 Module present in the SDN controller).
- **Backend Pattern Module:** Features the pattern engine for the SEMIoTICS backend, along with associated subcomponents (knowledge base, reasoning engine). It will enable 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), though additional interfaces may be introduced to allow for more flexible deployment and adjustments if needed. Will be able reason on the SPDI properties of aspects pertaining to the operation of the SEMIoTICS backend. Moreover, at runtime the backend Pattern module may receive fact updates from the individual Pattern Modules 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 Module:** Integrated in the SDN controller to enable the capability to insert, modify, execute and retract network-level patterns at design or at runtime. It will be 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. Will feature different subcomponents as required by the rule engine, such as the knowledge base, the core engine and the compiler.
- **Field Layer Pattern Module:** 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 will be able to host patterns in an executable form compared to the pattern rules as provided in the other layers. The executable patterns will be 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. The field layer pattern module will store pattern executables in a local knowledge base that will be updated by the pattern orchestrator as needed and requested

The implementation details of the above are documented in the corresponding deliverables of WP4 (e.g., D4.6 for the backend components), including their API specification. Nevertheless, it is worth pointing out that all interactions between these components happen through a secure connection, protected with the industry-standard TLS protocol.

2.2.10.2.1 PATTERN INTERACTIONS WITH SECURITY AND PRIVACY MECHANISMS

Deliverable D4.1 details a first set of patterns, covering all of the SEMIoTICS targeted core properties, i.e. Confidentiality, Integrity, Availability, Privacy, Dependability, and Interoperability. Of those, the first four properties, and the corresponding patterns, are of particular interest in the context of this deliverable. Moreover, additional patterns will be defined as the project progresses, aiming to reach a total of at least 36 patterns by the end of the project.

Nevertheless, 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.

2.2.10.2.2 INTERACTION SCENARIO – INTEGRITY

A first set of SEMIoTICS patterns is defined in D4.1, where the reader can refer to for more examples. 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, 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).

2.2.10.3 INTERACTION SCENARIO – PRIVACY

There are some cases where the SPDI properties can transition to an initially undesired state due to special conditions that 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.

2.2.11 LIGHTWEIGHT 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 [19] [20].

Block ciphers, the main symmetric key cryptosystems, perform well in this field. Prominent solutions include the industry-standard AES, which can be extremely lightweight, as well as PRESENT and CLEFIA, which are also standardized block ciphers for lightweight cryptography [21]. 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 [22]

3 ARCHITECTURE FOR THE SECURITY COMPONENTS

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.

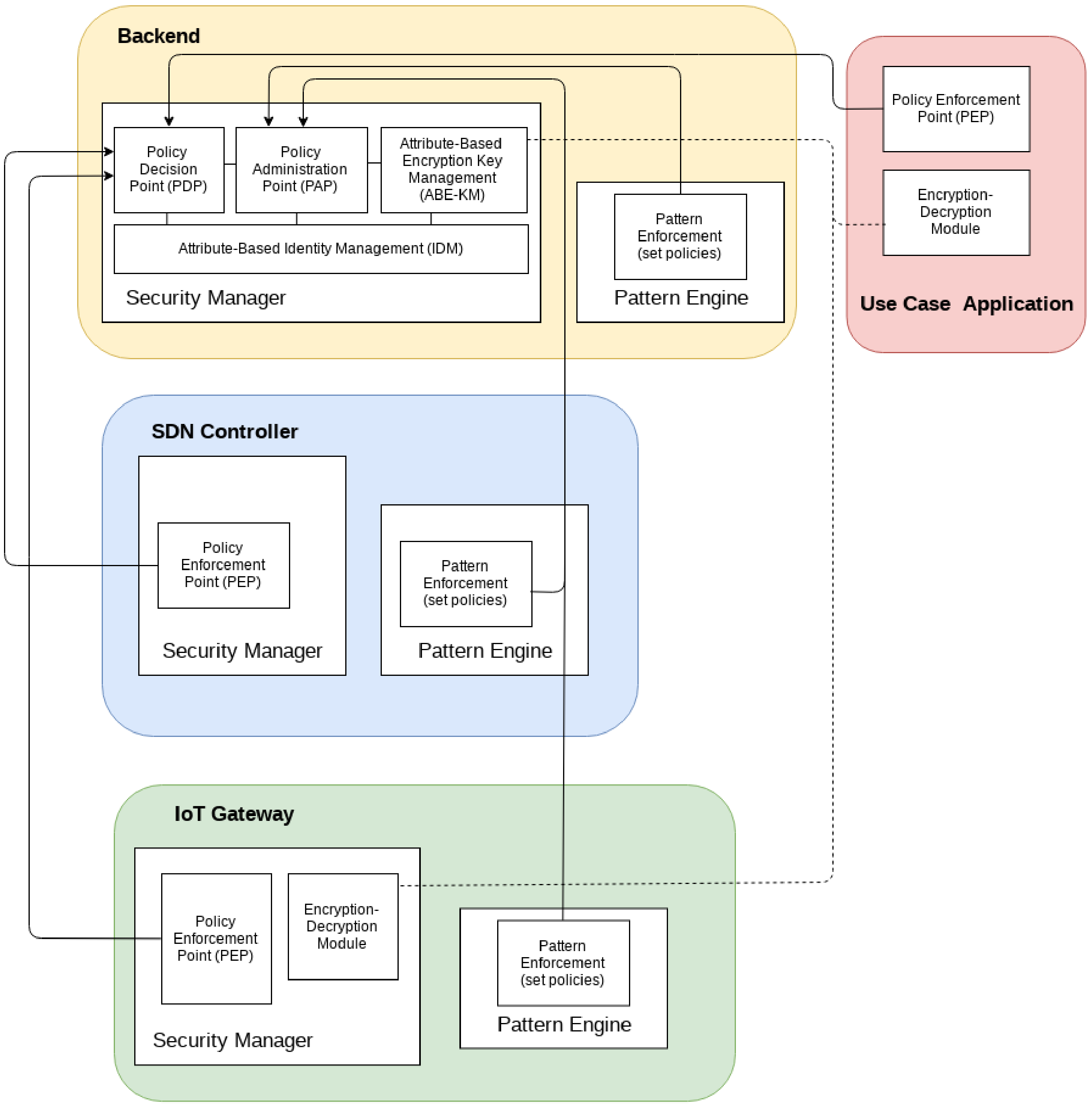

Backend
Policy Decision Point (PDP)
Policy Administration Point (PAP)
Attribute-Based Encryption Key Management (ABE-KM)
Attribute-Based Identity Management (IDM)
Security Manager
Pattern Enforcement (set policies)
Pattern Engine
Policy Enforcement Point (PEP)
Encryption-Decryption Module
Use Case Application
SDN Controller
Policy Enforcement Point (PEP)
Security Manager
Pattern Enforcement (set policies)
Pattern Engine
IoT Gateway
Policy Enforcement Point (PEP)
Encryption-Decryption Module
Security Manager
Pattern Enforcement (set policies)
Pattern Engine

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 some 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, we also let use cases with connectivity restrictions, or deployed in external clouds to ensure access control through encryption, in a more decentralized manner.

4 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 readers 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.

4.1 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.

4.2 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.

4.2.1 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.

4.2.2 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].

4.3 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.

4.4 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 18 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.

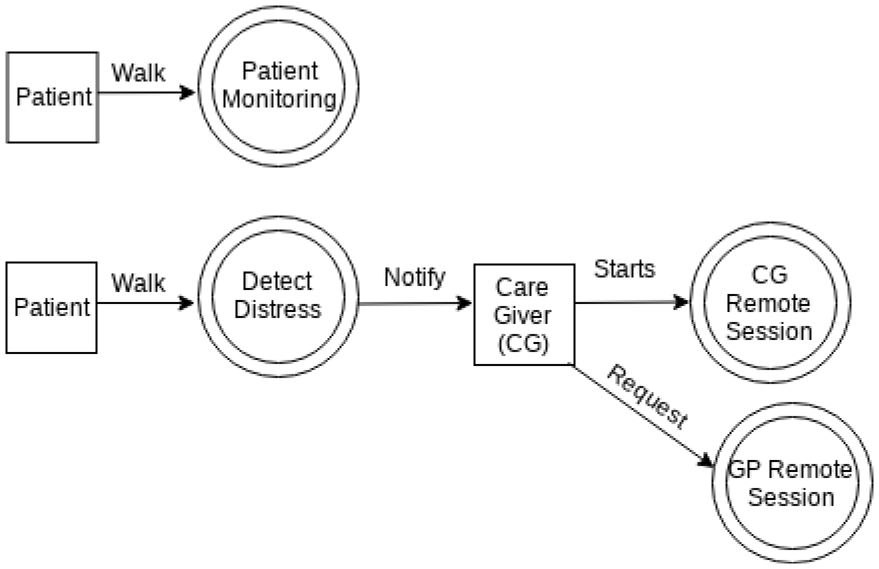


FIGURE 18 COMPLEX PROCESSES FOR UC2

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

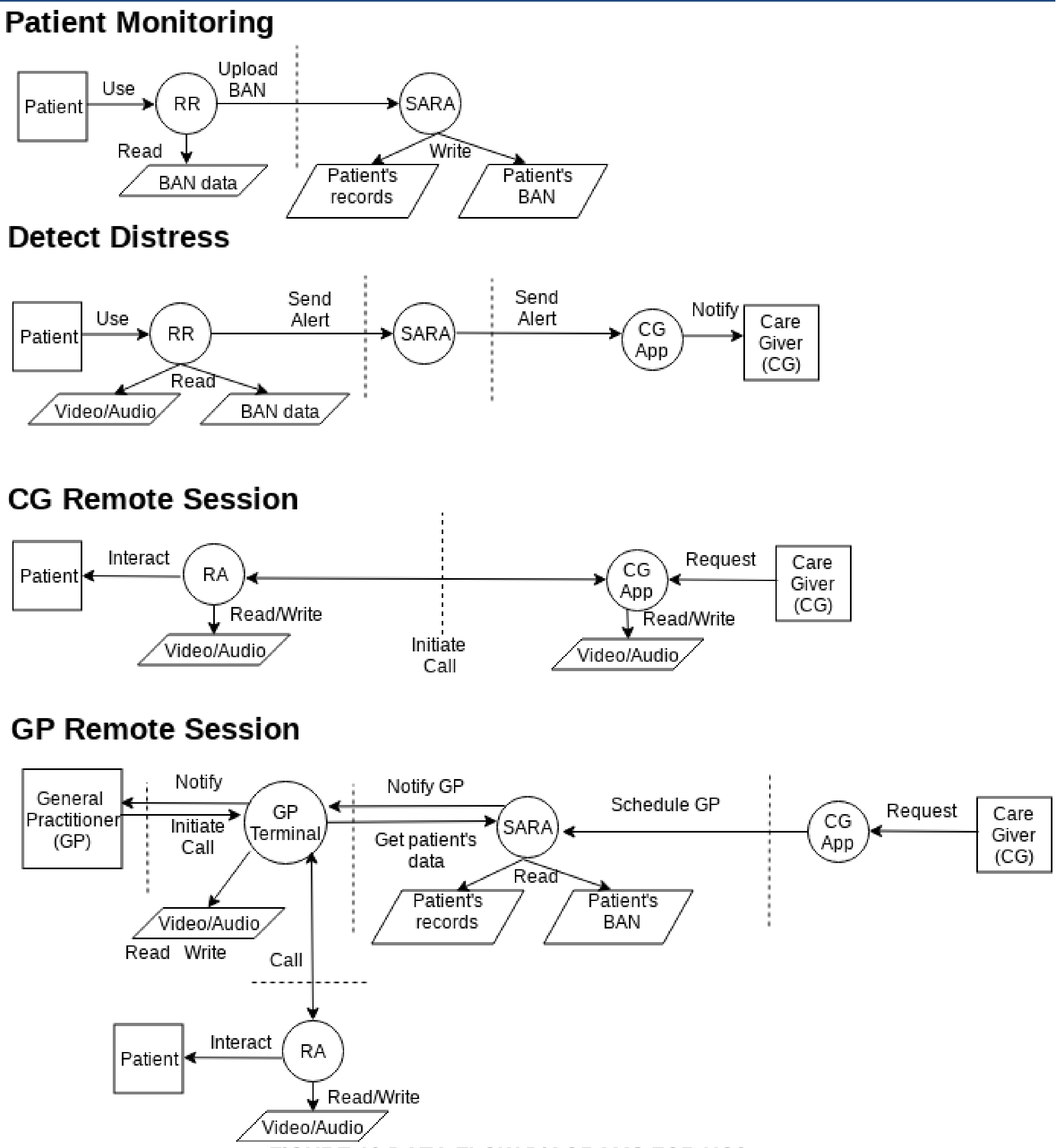


FIGURE 19 DATA FLOW DIAGRAMS FOR UC2

4.4.1 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.

4.4.2 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.

4.4.3 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.

4.4.4 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.

4.5 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.

4.5.1 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),	Tampering	An attacker could try to tamper with the	To counter this threat the RR, RA, GP and SARA shall have proper access control mechanisms to

RR (Video and Audio) RA (Video and Audio) GP Terminal (Video Audio)		information stored the different components storing it.	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.

4.5.2 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*	Tampering	An attacker could attack the	SDN switches and controllers’ communication could be encrypted to avoid third-party

Alert		underlying SDN network to redirect the traffic to malicious destinations by adding flow rules to SDN switches	manipulation of network data flow. Also authentication should be used to avoid any kind of tempering using the NBI interfaces of the SDN controllers

4.5.3 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.

4.5.4 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.

5 CONCLUSION

This deliverable has presented an overview of the various mechanisms improving security and privacy throughout the SEMIoTICS architecture.

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.

This document serves as the blueprint for the development of our security mechanisms in Task 4.5. In the next steps, the project will implement and/or extend the components presented herein in the next implementation cycles in the context of Task 4.6. Furthermore, SEMIoTICS will report the final description of the mechanisms used, along with the final specification of interactions across components, in the final delivery of the security and privacy mechanisms (D4.12). Last but not least, D4.12 will include the description of specific scenarios involving the end user to highlight the user awareness and accountability possibilities of our framework.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[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.