(FP7 614100)

D4.4 Security of resource-constrained subsystems

26.1.2016 – Version 1.0

Published by the IMPReSS Consortium

Dissemination Level: Public

Project co-funded by the European Commission within the 7th Framework Programme and the Conselho Nacional de Desenvolvimento Científico e Tecnológico
Objective ICT-2013.10.2 EU-Brazil research and development Cooperation

Document control page

Document file:	IMPRESS D4.4 Security of resource-constrained subsystems_1.0.docx
Document version:	1.0
Document owner:	Janne Takalo-Mattila
Work package:	WP4 Mixed criticality resource management
Task:	4.4 Security of resource-constrained subsystem
Deliverable type:	P
Document status:	☒ approved by the document owner for internal review ☒ approved for submission to the EC

Document history:

Version	Author(s)	Date	Summary of changes made
0.1	Janne Takalo-Mattila	14.10.2015	Initial version and draft of the index
0.2	Enrico Ferrera	4.1.2016	Added section 4
0.3	Janne Takalo-Mattila	13.1.2016	Added Executive summary and conclusion
0.4	Janne Takalo-Mattila	18.1.2016	Updated architecture picture and updated section order
0.5	Janne Takalo-Mattila	21.1.2016	Updated based on Eduardo Souto's comments
1.0	Janne Takalo-Mattila	26.1.2016	Vinicius Fraga internal review fixes. Ready for external review.

Internal review history:

Reviewed by	Date	Summary of comments
Eduardo Souto	20.1.2016	Accepted with minor modifications
Vinícius Fraga (UFPE)	25.1.2016	Accepted with modifications.

Index:

1. Executive summary

This deliverable describes the work done in task 4.4 Security of resource-constrained subsystems. Deliverable specifies how security of IMPReSS resource-constrained subsystems is handled. IMPReSS security architecture consist of two parts. First part of the security solution focuses on resource authorization in resource constrained devices and applications. Second part of the security solution focuses on security between wireless sensors and the Resource Abstraction Interface (RAI).

Communication between resource abstraction interface, other resources and applications in IMPReSS platform is implemented with standard web technologies such as Hypertext Transfer Protocol(HTTP) and MQ Telemetry Transport(MQTT). Therefore standardized and well-known technologies for securing communication in application layer such as HTTPS and MQTT over TLS can be used. In order to get access to a protected resource, users, applications and clients willing to use the resource needs to be authorized. This task defines the requirements and implements OAuth2.0 based authorization system targeted for Internet of Things (IoT) based systems, where applications can be extremely limited.

IPv6 over Low Power Wireless Personal Area Networks (6LowPan) secured with IPSec is used as reference communication protocol for resource-constrained IMPReSS subsystem. The main component in the security between wireless sensors and the RAI is an adaptive security manager (ASM), which can select the required level of security and inform the resource about it. This makes it possible to support both high level of security and also long lifetime of battery powered wireless devices when needed. Based on commands from the ASM, wireless sensors can select the suitable key from pre-shared keys.

2. Introduction

The main aim of the task 4.4 is to investigate and define dynamically adjustable security method suitable especially for resource constrained devices. Task has been divided into two parts.

The first part of this task focuses on secure communication between applications and the IMPReSS platform, especially from the authentication point of view. Application can be seen as user level client utilizing information from IMPReSS platform. In this context, clients that communicate with IMPReSS platform can be also resource limited. Therefore typical authentication technologies such as OAuth2.0 can't be utilized without modifications. Communication between applications and IMPReSS platform has typically been developed using standard web technologies such as HTTP and MQTT. Therefore we have also utilized standardized and well-known technologies for securing communication in application layer such as HTTPS and MQTT over Transport Layer Security (TLS).

The second part describes how communication can be secured in wireless sensor networks that are typical resource constrained devices in IoT-based systems. In order to support both the high level of security and the need for long lifetime of battery powered wireless devices, flexible means to adjust the level of security at runtime is needed. To this end, we developed a security technology that can be used to adjust the level of security based on the context.

2.1 Purpose, context and scope of this deliverable

This deliverable describes the work done in task 4.4 Security of resource-constrained subsystems. Chapter 3 describes how communication is secured between applications and IMPReSS platform starting from RAI. Chapter 3 also discusses about the authentication in resource limited environments and shows how the authentication is implemented in IMPReSS platform using modified OAuth2.0 technology.

Chapter 4 describes how communication can be secured in wireless sensor networks. Section covers security starting from individual sensor node to the RAI. As a communication technology 6LowPan has been used. It is very suitable communication method for low-power devices making those devices part of Internet. IPsec has been used to secure communication networks by authenticating and encrypting IP packets in communication session.

2.2 Background

Often wireless sensor networks and other Internet of Things –type systems collect information that can be sensitive. Therefore, it is important that the security is on adequate level concerning both the confidentiality of the data and the amount of risks in the given environment. Often wireless sensor networks can operate unattended in remote places and are therefore good targets for malicious attacks. Additionally, in wireless communication the risk of eavesdropping is higher than in wired communication. Many of the risks related to wireless communication can also be difficult to avoid (e.g. denial-of-service types of attacks in resource constrained networks).

There are many kinds of technologies available for distributed systems to prevent different types of malicious attacks, but these technologies are typically too demanding considering the requirements for computational capacity, storage space and energy efficiency in resource limited devices and networks such as wireless sensor networks. In our approach we have tackled these challenges e.g. by using lightweight solutions such as IPSec for wireless sensor networks and developing methods for selecting suitable key-lengths.

Security is typically modelled with the triangular Confidentiality, Integrity and Availability (CIA) security model (Dargie et al 2010) presented in Figure 1.

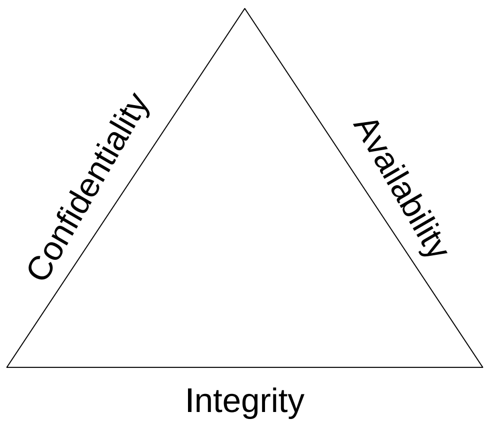

Figure 1. CIA triad of confidentiality, integrity and availability

The CIA triangular consists of following edges:

- Confidentiality: Confidentiality limits the access to data. Security technologies need to guarantee that only the authorized parties are allowed to access the data. This prevents confidential and sensitive data to leak for unauthorized attackers or individuals. Confidentiality is approximately equivalent to privacy.
- Integrity: Messages sent between parties needs to stay unmodified. Possible attackers can't modify or destroy sensitive information. Integrity guarantees that information is trustworthy and accurate.
- Availability: Availability guarantees that information can be reliable accessed by the authorized parties. Typical way to measure availability is to compare downtime to uptime as a percentage value e.g. availability 99.9% ("three nines") meaning 8.76 hour downtime per year.

Wireless communication especially between resource restricted devices has special types of challenges compared to traditional communication between high power devices such as PCs. Many traditional security mechanisms are not suitable for wireless sensor networks because of limitations in resources. Quite often resource limited devices in IoT have less than 16 kB of RAM and 128 kB of storage space. This limits e.g. used key lengths. In addition to resource limitations, communication can also be prone for errors. Remote and distributed operation of IoT devices makes also possible to physically attack to the devices. That is how attackers can also be able to get security keys from devices by physically attacking them. Resource limitations often restrict making the devices fully tamper-proof. Typical features in IoT devices affecting to security are summarized in list below:

- Memory restrictions affecting e.g. to key lengths
- Communication prone for errors
- Remote location, which can make sensor vulnerable for physical attacks
- Limited user control limiting the access control types such as passwords

The required level of security depends on the nature of the data to be transmitted, but also the location of the devices can have effect for the needed level of the security. If there is a higher risk for attacks, the needed level of the security is also higher. Therefore security mechanisms need to be flexible for the changes.

3. Application level information security

3.1 Application level information encryption

In application level typical security protocols are TLS and its predecessor Secure Sockets Layer (SSL). These are typically used between browsers and web servers and all the major websites such as Google, Facebook and Twitter are using these technologies. In this task we have selected to use TLS as an application level security technology encrypting the communication between IMPReSS platform and client applications.

Fehler! Verweisquelle konnte nicht gefunden werden. presents communication between different entities in IMPReSS platform. Security from RAI to devices such as wireless sensor is covered in section 4. Applications and other clients need secured communication to following entities:

- MQTT broker, MQTT-connection
- Global Resource Manager, Websocket-connection
- Authorization server, HTTP-connection
- Resource abstraction interface, HTTP-connection
- Local Resource Manager(LRM), HTTP-connection

All used communication protocols i.e. Websocket-, MQTT- and HTTP support TLS-based data encryption, which can be used to guarantee the confidentiality of transmitted information. This section covers only the server authentication using X.509 certificates. Client authentication is covered in chapter 3.4.

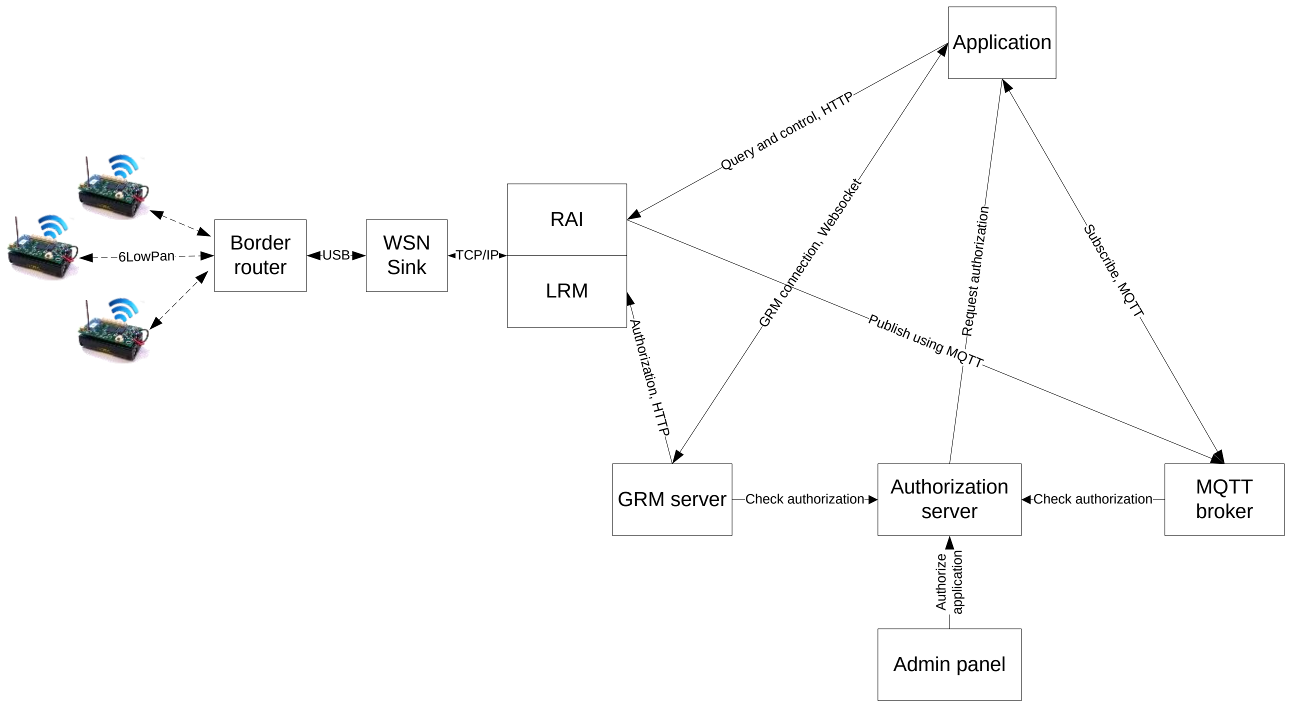

Figure 2 Communication between entities

In IMPReSS project, we have selected Mosquitto (Mosquitto 2016) to implement MQTT broker functionality in the system. By default MQTT uses TCP as a connection protocol, which is not encrypted. In order to support MQTT connection over TLS, server hosting MQTT broker needs X.509 certificate, which is signed by certification authority. In the beginning of TLS connections, client and server have to make a TLS handshake and the client validates the X.509 certificate in the server including the public key of the server. After that, a secure connection between client and server can be made. In our demonstration scenario, self-signed certificate can be used instead of certificate from certification authority. We have used OpenSSL library to generate certificates. In practise, this happens as follows.

First, certificate signing request is created using following command:

```
openssl req -out server.csr -key server.key -new
```

Then, Certificate signing request file (CSR) can be sent to certification authority or it can be self-signed with following openssl command:

```
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -days
365
```

After generating the certificate, a Mosquitto configuration file mosquitto.conf needs to be modified with the following information:

```
Port 8883 #standard port for MQTT connection over TLS
cafile file #PEM encoded CA certificate
certfile file #PEM encoded server certificate
keyfile file #PEM encoded keyfile
```

Global Resource Manager in IMPReSS platform uses a Websocket-based protocol to communicate with clients. In addition to MQTT, Websocket-protocol also uses TLS for data encryption. Autobahn library uses Twisted-network engine as its back-end. Twisted in server side is configured to utilize TLS over Websocket instead of plain Websocket over TCP by making it listen to SSL with command:

```
reactor.listenSSL(8000, factory,
                      ssl.DefaultOpenSSLContextFactory(
            'keys/server.key', 'keys/server.crt'))
```

Key file and certification files can be generated in similar way than in MQTT case presented earlier in the section. In client side using TLS/SSL connectivity is straightforward. Client is set to use TLS/SSL connection instead of plain Websocket over TCP with command:

```
reactor.connectSSL('localhost', 8000, factory, ssl.ClientContextFactory())
```

Authorization server, Resource Abstraction Interface and Local Resource Manager use HTTP as communication protocol. HTTPS is communication over HTTP within connection encrypted by TLS. Configuring Authorization server, Resource Abstraction Interface and Local Resource Manager to use HTTPS instead of plain HTTP follows the same principle as MQTT and Websocket servers i.e. servers hosting the services needs to have certificates and corresponding key files and services need to be configured to use TLS.

3.2 Resource authorization using OAuth2.0

In order to get access to protected resource, users, applications and clients using the resource needs to be authorized. Typically users get access to protected resource by authenticating themselves with their username and password. Instead of using usernames and passwords directly in authentication, access to restricted resources can be delegated. This is how third party applications can get access to user resources such as Facebook account without getting access to actual username and password. OAuth2.0 (OAuth 2.0 2012) is popular and open standard for authorization. In OAuth2.0 there are four different roles:

- **Resource owner** is an entity that is able to grant access to protected resource. For example in typical web usage, Facebook user authorizes a 3rd party application to access user account. Therefore user is the resource in that case.
- **Resource server** is a server that hosts protected resources. It accepts tokens sent by the client.
- **Client** is an application requesting protected resources from resource server. Client is authorized by the resource owner to use protected resources.
- **Authorization server** is the actual server giving temporary access tokens for clients. These access tokens are used as a replacement for username and password combination. Access tokens needs to be accepted by the resource server

Process of delegating is quite simple. Resource owners grant access to protected resources by giving their username and password combination to an authorization server or using some other method to verify the ownership of the protected resources. If ownership of resources can be guaranteed, authorization server issues an access token for the client which can be used as a temporary password. Figure 3 shows an example of how OAuth2.0 works in typical web usage, where a user authorizes a third party application to utilize its private data from a resource server.

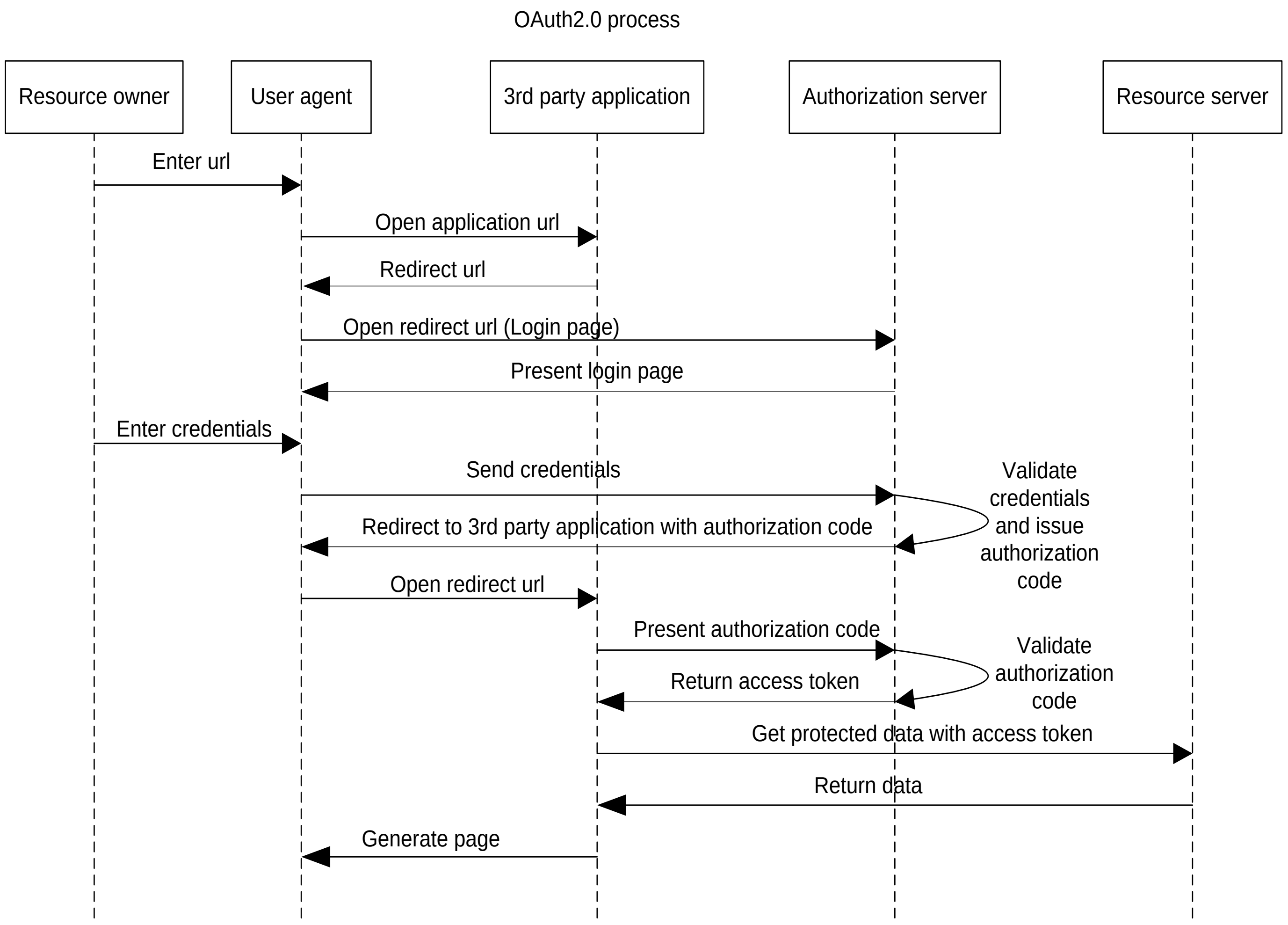

Figure 3 OAuth2.0 authorization process

As can be seen in Figure 3, in a typical OAuth2.0 process a resource owner authorizes a third party application to use protected resources through user agents, which is often a web browser. Therefore, this type of information flow requires a web browser and a data entry method for entering the user credentials. Current OAuth2.0 specification is not perfectly suitable for devices with limited input capabilities and other limitations, but various methods to tackle these challenges have been presented (Google 2016)(OAuth2.0 2010).

3.3 Special requirements for authorization system in IMPReSS platform

OAuth2.0 is a mature technology for authorization in web domain, so it provides a stable foundation for building authorization scheme also for Internet of Things - type of system such as the IMPReSS platform. As mentioned in previous section, currently OAuth2.0 authorization technology does not perfectly fit for Internet of Things–style use cases. There are many special requirements for an authorization system targeting the IoT.

In IoT domain it is often required that devices and applications running in resource limited devices act on behalf of user owning the resources. However, many of these devices such as smart TV and other home appliances can have only limited input capabilities. In extreme cases applications running in very resource restricted devices can also have limited output capabilities or no output capabilities at all. For example, in IoT domain a user can download an application or script and install it into an embedded device such as the Raspberry Pi. In many cases the user is unable to see any feedback from application, which makes it impossible to use the common OAuth2.0 process. Even OAuth2.0 process for devices proposed by Google is not feasible without modification, because it requires output capabilities (Google 2016)

IoT system is typically distributed including many different devices. In typical OAuth2.0 use cases the authorization server and the resource server are located in the same physical host, so both of these instances have access to the same database. Therefore, resource server can check the authorization just by doing the lookup to the database. But in IoT systems there are typically many resource servers (such as MQTT broker, GRM server, Resource Abstraction Interface etc.) and only one authorization server. Therefore, the resource servers need a service in the authorization server that can be used to check the validation of the access token at runtime. There is already an OAuth2.0 draft available that proposes this type of behaviour, referred to as *token introspection* in the draft (Oauth2.0 2015). Using token introspection service, a resource server can send an access token to an authorization server in order to confirm the validity of the token. Figure 4 presents how authorization and resource servers can be located either in a) the same physical host or b) separate hosts.

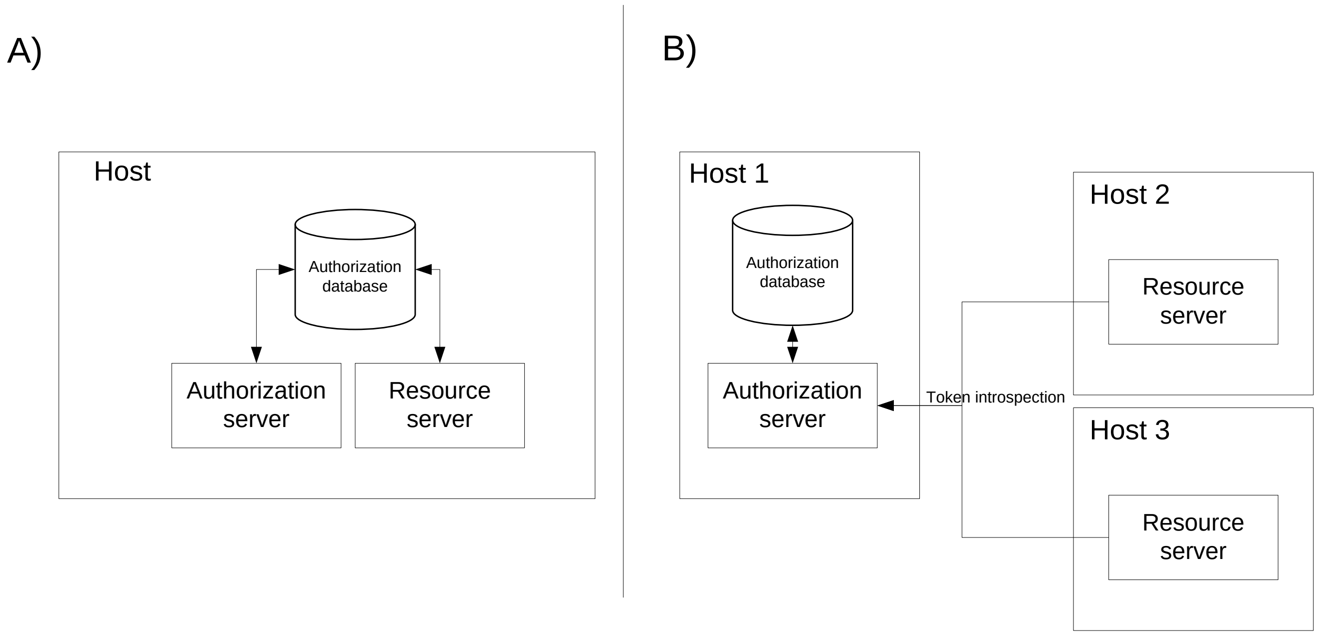

Figure 4 Authorization server running A) in the same host with resource server and B) in different host than resource server

The OAuth2.0 concepts can also be easily mapped to the IoT domain:

- Resource owner in IoT domain is a system administrator managing distributed resources in his system.
- Resource server can be for example MQTT broker storing protected information produced by sensors.
- Client is an application requesting protected resources. In IoT domain clients can be extremely limited by their resources. Applications can be simple scripts running in embedded devices, which purpose can be controlling lightning based on human presence information.

- Authorization server is a server managing the access for protected resources. In IoT domain the authorization server typically is hosted in a different server from resources.

3.4 Client authorization in IMPReSS using authorization server for IoT

In this task, we developed a customized OAuth2.0 based authorization system, targeted especially for IoT based systems, where applications can be extremely limited in terms of input and output capabilities. Developed authorization fulfils the requirements presented in section 3.3. Implemented authorization system is developed using Python-language and it uses Django web development framework. MySQL has been used as a database. Gunicorn has been used as a WSGI HTTP server and serving static context and proxying HTTP requests to Gunicorn nginx has been used. The internal architecture of authorization server is presented in Figure 5.

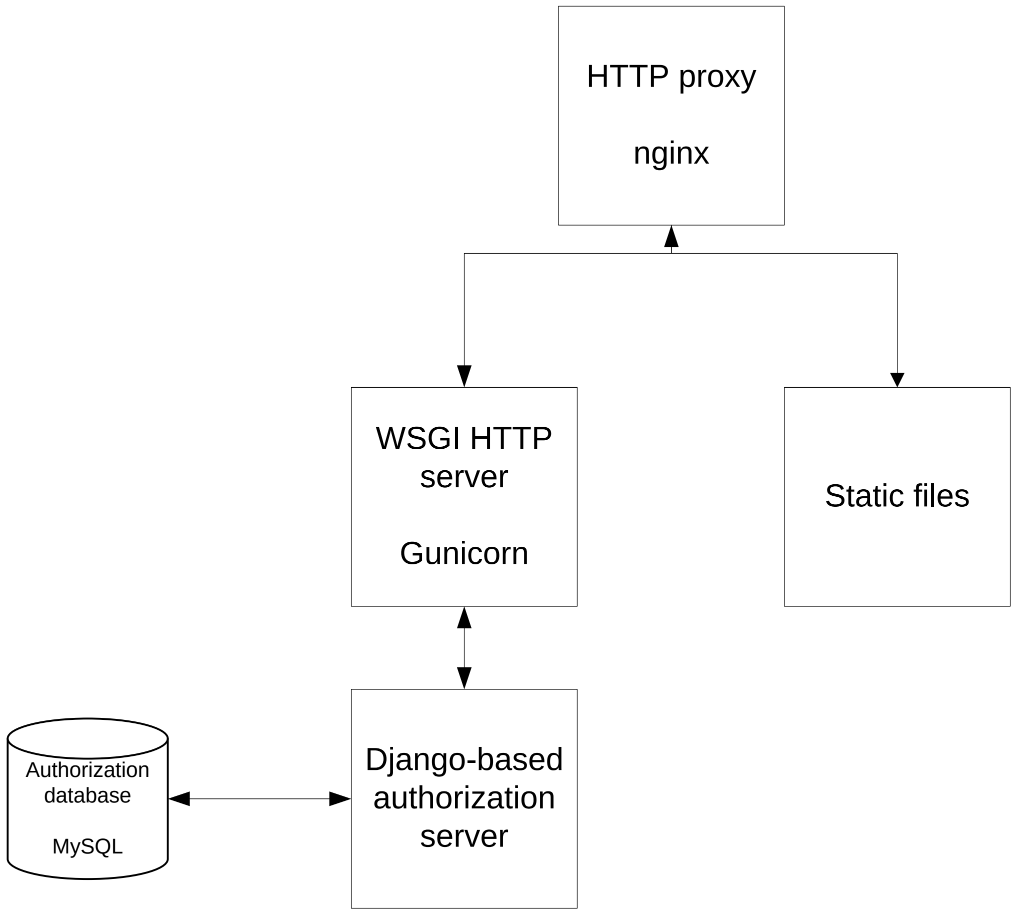

Figure 5 Architecture of an IMPReSS authorization server.

In IMPReSS we assume that all the applications are downloaded and installed through a network manager component. In order to access the resources provided by the IMPReSS SDP the applications need an access token, which can be used as a temporary password for resource servers in IMPReSS platform. Authorization server supports the operation presented in Table 1.

Table 1 Operations supported by authorization server

Operation	Description
Register	Applications register themselves to authorization server.
Request_access	Applications can request access to resources.
Authorize_client	Resource owner authorizes client to use resources.

Token_introspection	Resource servers use it to check if access token is valid.

The process of getting an access token begins with registration. After installation applications need to register themselves to the authorization server. All the operations to the authorization server are done using HTTPS interface and JSON is used as a payload serialisation format. In the registration phase, data presented in Table 2 is needed.

Table 2.Register request parameters

Parameter	Obligatory	Description
client_identifier	Yes	Unique identifier for the client to identify itself in authorization server
name	No	Human readable client name
scope	no	Application tells an authorization server what permissions they will need on resources they are accessing. Default is application_all, which means all resources available for applications. It is formatted as a space separated list e.g. "mqtt grm".

The supported options for scope parameter are presented in Table 3. The client can be an application or a resource such as the MQTT broker.

Table 3 Scope options

Client type	Scope	optional parameters	Description
application	application_all		Access to all resources in the system
	mqtt	Scope mqtt can be further limited in topic-level e.g. mqtt:/topic/subtopic mqtt:/topic/subtopic+r is read-only, +rw read&write	Access to MQTT broker
	grm		Access to GRM server
resource	resource		Access to token_introspection endpoint

Following is an example request for registering the client:

```
POST /provider/register
Content type: application/json
{
          "client_identifier": "abc",
          "name": "client name"
          "scope": "application_all"
}
```

If the request is successful a response with parameters presented in Table 4 is received:

Table 4 Register response parameters

Parameter	Description
verification_code	Authorization server generates a verification code. It is associated with client identifier. Verification code is 32-character hexadecimal string generated by uuid.
request_link	Link for the client to request access token.
activate_link	Link for the resource owner (system administrator) to accept the client registration. Address for accepting the clients can also be known beforehand by the resource owner.
interval	Recommended interval for polling in seconds, if resource owner has accepted client registration. Polling is done to authorize_link address.
scope	Given permissions

Following there is example response from successful registering:

```
{
        "verification_code": "16fd2706-8baf-433b-82eb-8c7fada847da",
        "request_link": "/provider/request_access",
        "activate_link": "/provider/authorize_client",
        "interval": 3600,
        "scope": "application_all"
}
```

If client has already been registered, the following response is received:

```
{"error": "already registered"}
```

Resource owner (system admin in IMPReSS case) then needs to accept the registration. This is done by an activate_link address or an admin panel page. Purpose of the admin panel page is to list all the registered clients, accept pending registrations, remove client permissions etc. Currently client registration can be accepted only by using activate_link. Following parameters are needed in this process:

Table 5. Authorizing the client

Parameter	Obligatory	Description
client_identifier	Yes	Client to be authorized.

An example request for authorizing the client is as follows:

```
POST /provider/authorize_client
Content type: application/json
{
        "client_identifier": "abc"
        }
```

The Resource owner needs to be known by the authorization server. Two options to authenticate the resource owner can be used:

- TLS-based client certificate is deployed to resource owner and is used for authentication.
- HTTP Basic authentication with username and password is used together with HTTP over TLS.

Following there is an example of response from a successful authorizing:

```
{"authorized client": "abc"}
```

If client cannot be authorized, following message is received:

```
{"error": "not able to authorize client"}
```

After successful registration application can start polling the access token from /request_access address. Polling is done according to the value of the parameter "interval" received as a response for registration. In request access phase, the data presented in **Fehler! Ungültiger Eigenverweis auf Textmarke.** is needed in the request.

Table 6. Request access parameters in request

Parameter	Obligatory	Description
client_identifier	Yes	Unique identifier for the client to identify itself in authorization server
verification_code	Yes	Code received in registration phase

Following, there is an example of a request for access:

```
POST /provider/request_access
Content type: application/json
{
        “client_identifier”: “abc”,
        “verification_code”: “16fd2706-8baf-433b-82eb-8c7fada847da”
}
```

If resource owner has not accepted the application yet, following response is received:

```
{“error”: “authorization pending”}
```

Case the verification_code is not valid, following response is received:

```
{“error”: “code not valid”}
```

If application is not registered, following response is received:

```
{“error”: “client not registered”}
```

Finally, if the request is successful, response with parameters presented in Table 7 is received.

Table 7. Request access parameters in response

Parameter	Description
access_token	Authorization server generates a access token, which can be used to access the resource servers. In IMPReSS access token is 32-character hexadecimal string generated by uuid.
expires_in	The lifetime of access token in seconds.
refresh_token	Refresh token is used to obtain new access code after access code has been expired. In IMPReSS refresh token is 32-character hexadecimal string generated by uuid.

Following there is an example request of a successful access request:

```
{
        “access_token”: “12345678-8796-2314-gtyu-joujouhytg5”,
        “expires_in”: 3600,
        “refresh_token”: “87654321-5454-14a1-24da-jhyhygr457ga”
}
```

Access token replaces different authorization methods such as username and password combinations with a single token. Therefore it makes authorization simpler for resource server, because only one type of authorization method needs to be understood. Refresh token can be used to issue new access token, if the old one has expired. The use of refresh tokens is optional in the authorization server, but it is highly recommended. Refresh token adds following extra security benefits:

- Authorization is only temporary. Resource owner can limit the time, how long application is allowed to use resource

- Access tokens found from resource servers memory or log files are less useful for attackers. If attacker gets his hands into access token found from server memory, it is often expired or only valid for limited amount of time

When the access token expires, an application can issue a new one by sending the request to /request_access address. When refreshing the token, the verification_code parameter is replaced with refresh_token parameter in The Resource owner needs to be known by the authorization server. Two options to authenticate the resource owner can be used:

- TLS-based client certificate is deployed to resource owner and is used for authentication.
- HTTP Basic authentication with username and password is used together with HTTP over TLS.

Following there is an example of response from a successful authorizing:

```
{“authorized client”: “abc”}
```

If client cannot be authorized, following message is received:

```
{“error”: “not able to authorize client”}
```

After successful registration application can start polling the access token from /request_access address. Polling is done according to the value of the parameter "interval" received as a response for registration. In request access phase, the data presented in **Fehler! Ungültiger Eigenverweis auf Textmarke.** is needed in the request.

Table 6. Otherwise the request is the same. Following there is an example request to obtain a new access token.

```
POST /provider/request_access
Content type: application/json
{
         “client_identifier”: “abc”,
         “verification_code”: “87654321-5454-14a1-24da-jhyhygr457ga”
}
```

If the refreshing is successful, following response is received:

```
{
         “access_token”: “12345678-8796-2314-gtyu-joujouhytg5”,
         “expires_in”: 3600
}
```

Wrong refresh token generates following response:

```
{“error”: “unable to refresh token”}
```

The whole process starting from registration and ending to successfully receiving the access token is presented in Figure 6 .

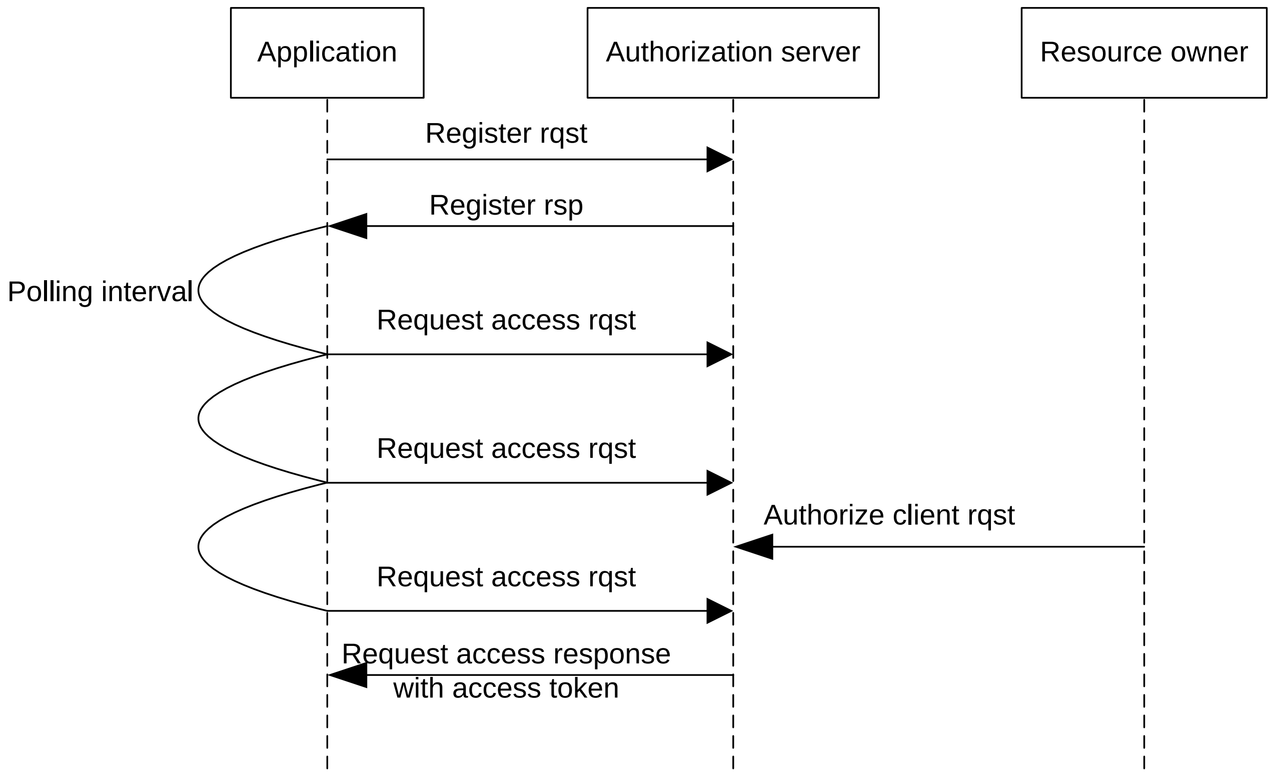

Figure 6 Process for issuing access token

After successfully issuing an access token, the application can use it to receive data from resource server. Access token is sent to resource server inside a normal HTTP request's header over TLS. Access token is transmitted to resource server in the authorization header. Following there is an example of a HTTP GET request with an access token:

```
GET /resource HTTP/1.1
   Host: server.example.com
   Authorization: Bearer 12345678-8796-2314-gtyu-joujouhytg5
```

For example with Python Requests-library aforementioned HTTP request can be done with following code:

```
headers = {"Authorization": "bearer " + access_token}
response = requests.get("https://server.example.com", headers=headers)
```

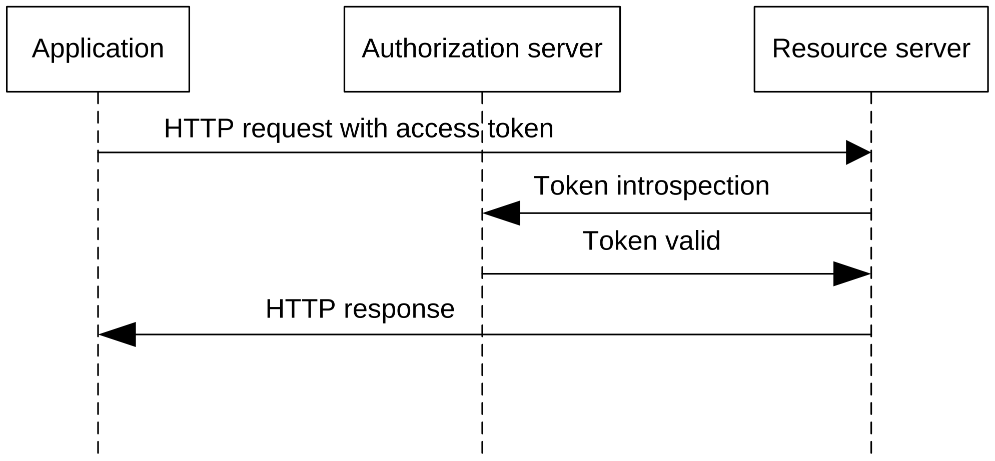

Figure 7. Requesting data from resource server

Resource server can check the validity of an access token by using token introspection operation. Operation is performed by sending HTTP Post-request to /token_introspection endpoint. Access_token parameter is the only required parameter to be sent in the request. Following there is an example:

```
POST /provider/token_introspection
Content type: application/json
Authorization: Bearer 874ltytf-gtgt-jbu5-cf4f-joujoudwfh
{
        “access_token”: “12345678-8796-2314-gtyu-joujouhytg5”
}
```

Token endpoint needs authentication to be able to access endpoint. Normal OAuth2.0 access token can be used or separate client authentication. Response from successful token introspection request has parameters as presented in Table 8.

Table 8. Token introspection response

Parameter	Description
active	Boolean indicator if token is active.
exp	Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating when this token will expire.
client_id	Client identifier for client the has requested access token
scope	A space-separated list of strings representing the scopes associated with this token.

Following is an example response from token introspection endpoint:

```
{
        “active”: true,
        “client_id”: “12345678-8796-2314-gtyu-joujouhytg5”,
        “exp”: 1449826353,
        “scope”: “application_all”
}
```

4. Securing communication in resource-constrained network

4.1 Secure communication in 6LowPAN networks

Resource constrained devices adopted within IMPReSS project operate on IEEE 802.15.4 radio for its communication. 6LoWPAN standard provides the adaptation layer for IPv6 connectivity to those resource constrained devices. In general, security to these devices shall be provided in network layer (IPSec), transport layer (DTLS) (Raza et al. 2012b) or in Application layer.

Internet Protocol Security (IPSec) is de-facto standard to provide security to IPv6 traditional networks. It is supported by the key management protocol "Internet Key Exchange version 2" (IKEv2). It is not trivial to adapt these security protocols for constrained environments due to resource constraints. The first lightweight IPSec solution was realized by Raza et al. (Raza et al. 2012a). It implemented only Encapsulation Security Payload (ESP) method in the security options of IPSec protocol. The ESP provides packet-level encryption using symmetric cryptography algorithm, whereas the Authentication Header (AH) of IPSec provides protection for the IP packet header.

Raza et al. (2014) introduced a compressed version of IKev2 protocol, which is a key management protocol for IPSec. A partial implementation of IKEv2 was realized in Contiki operating system by Jutvik (Jutvik 2014), but this implementation requires more flash memory in the resource constrained nodes and does not operate on Telos-B motes (SKY platform in Contiki) due to resource constraints.

4.2 Secure communication between WSN Sink and RAI

Architecture of the 6LoWPAN network with WSN Sink and the resource constrained nodes are shown in Figure 8. In general, the WSN sink is a more powerful Linux host device such as a Raspberry Pi and it allows sending sensor data to remote processing, visualization and storage systems. The keys within the 6LoWPAN resource constrained nodes are pre-shared while flashing the firmware and each node is provided with a unique key. These unique keys are mapped with their IP addresses of the nodes and are provided as security policies in the Linux host as shown below in the Figure 9 **Fehler! Verweisquelle konnte nicht gefunden werden.**.

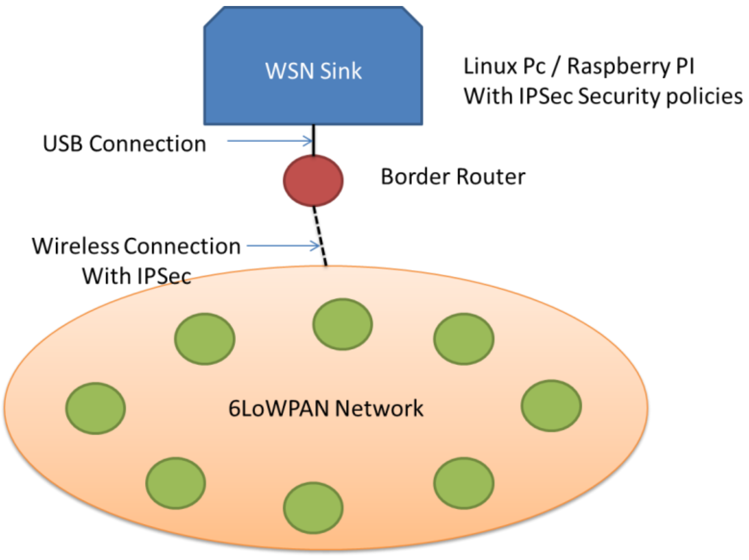

Figure 8: Secure 6LoWPAN Network

Figure 9 shows the IPSec configuration for a node with IP addresses aaaa::1. In particular, it defines the key, the encryption algorithm to be used by the WSN sink when a packet has to be encrypted and sent to a node, and the decryption algorithm to be used when a packet arrives from a node.

```
Edit the IPSEC-tools configuration file in a linux host.

                    ▪  /etc/ipsec-tools.conf

# ESP AES-CTR using 160-bit key (128 for aes key, 32 for ctr nonce)

add     aaaa::1     aaaa::2     esp     0x0002     -E     aes-ctr
0xcf5faaca70ee5ec4c8f43158a45c036369bbc0c9     -A     aes-xcbc-mac
0xcf5faaca70ee5ec4c8f43158a45c0363;

add     aaaa::2     aaaa::1     esp     0x0002     -E     aes-ctr
0xcf5faaca70ee5ec4c8f43158a45c036369bbc0c9     -A     aes-xcbc-mac
0xcf5faaca70ee5ec4c8f43158a45c0363;

# Security policies

spdadd aaaa::1 aaaa::0/64 any -P out

ipsec esp/transport//require;

spdadd aaaa::0/64 aaaa::1 any -P out

ipsec esp/transport//require;
```

Figure 9. IPSec-Tools Configuration

Adaptive security can be achieved by assigning two or more keys for each node. When more sensitive information is transferred a stronger key is used. If energy consumption is a constraint, then a lighter key can be used.

4.3 Adaptive Security Framework

4.3.1 Architecture

The concept of adaptability can be shaped starting from making some consideration of traditional secure links. Traditional links are autonomous and self-managed entities supporting a protected communication between two endpoints. The security protocol is in charge of managing them during the communication, without the need of external intervention. In order to make these links adaptive security ready, it is required to maintain a further active link with a control unit. The control unit is in charge of broadcasting security parameters when the network level of threats exposure changes appreciably. The new parameters should be applied to the already established secure links possibly without noticeably affecting the application operations. It starts defining the need of a software component dedicated to the purpose of receiving instructions by an Adaptive Security Manager (ASM) and applying them to the secure link it is handling. This component, dedicated to control operations, should be tightly coupled to the service socket used by the application to exchange data. It is hence defined a wrapper class, embedding both the control and service components in a unique solution. The components packaged can be better managed by the application which is only in charge of instantiating the wrapper class, from this point on adaptive socket (AS).

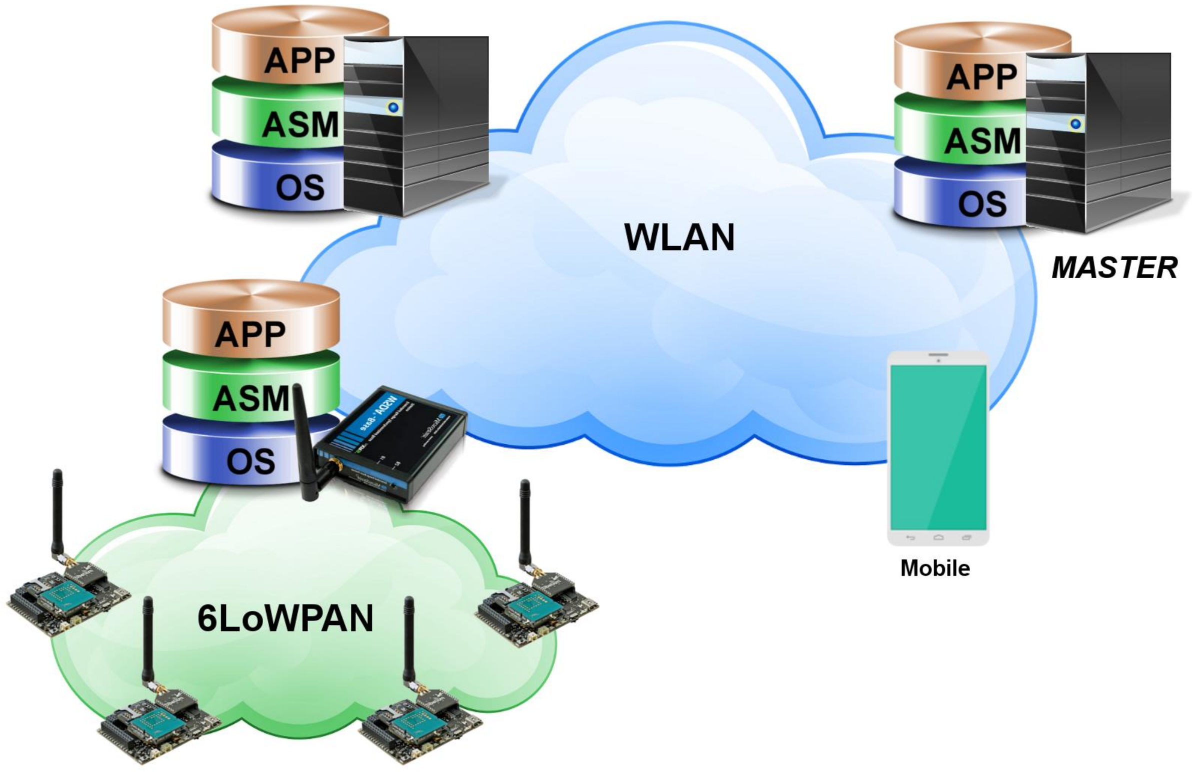

Figure 10 – Architecture

Figure 10 shows a possible deployment of the architecture with all the components discussed so far. The ASM is a daemon process running on a capable machine and instantiated with administrative privileges. This privileged level is required in order to be able to create a communication channel with the kernel and thus controlling those security protocols operating at kernel level. The ASM is in charge of waiting for the network security evaluator to issue a security level update request. The ASM processes the request and assesses its validity. The request is then broadcasted to all other system components contributing to the adaptive security service, namely remote ASMs and adaptive sockets. Despite of centralized solutions, where the control unit intervention is limited to authenticate the endpoints and to provide a session configuration for a secure channel, here the role of the ASM is much more present. As a matter of fact, network security level variations may occur once in a very long time or every ten seconds. The latter case would mean that every ten seconds the intervention of the ASM service might be requested and the system should start updating all the security links administrated by the adaptive security service.

The architecture is defined so that the system is able to serve resource-constrained devices, such as wireless sensors or mobile phones, avoiding the instantiation of an ASM process on each platform, resulting in considerable spare of resources. The adaptive sockets instantiated on these devices can simply register to one ASM as remote consumers. Multiple ASMs instances can be instantiated and configured to form a cooperating group. In each group of ASMs instances there must be a master instance which acts as adaptability service coordinator, while the remaining instances are limited to carry out the directives the master issues. Each ASM deployed on the network provides access to the adaptive security service; it is thus required to be registered to one ASM in order to access the service. Further, it is required that two adaptive sockets are registered to the same ASMs group in order to be mutually reachable through the respective ASMs. Adaptive sockets can be instantiated on the same machine that runs the ASM. In this case, the authentication and registration are performed by exploiting local host communications.

The ASM instances deployment makes no particular assumptions about the shape of the existing network architecture. Indeed, it works at an abstraction level, which might spans multiple networks logically separated, but that still use the same network layer protocol. The ASM can also be deployed on a boundary node placed at separation of two networks operating with different IP versions. However, in order to make a user capable of reaching a resource deployed on a different network operating with a different IP version, an intermediary application, with relay capabilities is required. This application must be registered on the boundary ASM and must instantiate two adaptive sockets. The tasks of the application consist of forwarding packets coming from a socket into the other without altering their content. This solution is quite common in wireless sensor networks where a remote entity intends to access the resources deployed in the WSN.

4.3.2 Adaptive Security Manager

A system meant to provide per-socket adaptive security and subject to strict reaction times is lead to generate a considerable amount of network traffic to carry out its task. In order to mitigate the effects that a simultaneous links update operation has on the network load, it is advisable to deploy a highly distributed solution, possibly with a multi-tier organization. The number of tiers is a value, which is directly bound to the response time of the system. More levels increase system scalability, but lead to a poorer response time. The solution proposed presents a two tier architecture with a master node and several slave nodes.

Master

The master node is in charge of interfacing the network security evaluator (NSE) and waiting for it to issue requests. When a new security level has been defined, the NSE sends an update request to the master ASM. The master evaluates parameters correctness and forwards the request to the slave ASMs. Master ASM starts its own update procedure with the entities registered to it, namely slave ASMs, users or resources.

Slave

A slave ASM is an offshoot of the master one and its task represents a prosecution of the master work. Despite its apparent reduced operating space, it plays a crucial role in keeping the service operative by taking charge of a workload share. Its preliminary task consists in trying to reach the network master node and establishing a secure session link with it. Absence of a secure link toward the master, leads to node isolation. This might be taken as a hint that there is an ongoing attack over the network. The ASM should tackle the situation by 1) increasing the security level with all the adaptive sockets it is able to reach and 2) attempting to restore the communication with the master.

4.3.3 Adaptive Socket

The adaptive socket, as adverted in the previous sections, wraps functionalities of a basic transport layer socket. It is thus possible to perform read and write operations through the socket. As previously stated an adaptive socket does not simply carry out traditional operations, but it is also in charge of maintaining an open channel toward its authoritative ASM. The AS is thus defined so that it can operate two links at the same time; a control link with the ASM and a service link with a remote user/application, each requiring its dedicated socket. As clarification, the adaptive socket makes no distinction between master and slave ASMs. It includes a module for dealing with the ASM and remote AS. A further module is charge of managing the service link.

The purpose of the adaptive socket is to overcome some design limitations of the state-of-the-art security protocols, which are based on agreements carried out exclusively between two endpoints. The concept behind the adaptive security architecture assumes the participation of a third entity to the agreement and, unlike Kerberos (Kerberos 2016), its presence is determinant as long as secure communications last. The AS is in charge of multiplexing communications directed to or coming from the ASM, remote control/service sockets and the application.

In Figure 11 it is shown a high level view of constitutive blocks of the adaptive socket. The application interface module acts as a proxy. It thus provides access to the embedded service socket and mediates the interactions between the application and the service socket. Socket controller represents the logic that governs the AS. Its main task is to enable or disable the application interface.

- Enabled: the interface becomes transparent to the application, thus it just forwards the function calls to the underling service socket. However, the cost given by forwarding the function calls is limited at an additional stack allocation. The data structures used to contain application data are not copied, thus resulting in a negligible overhead.

- Disabled: the interface decouples the application from the service socket by suspending any read or write operation. The operation on the AS blocks whatever is the calling thread.

The socket controller processes commands sent by the ASM or by a remote adaptive socket. Any time there is a request to update the secure channel parameters it leverages on the application interface and suspends all operations toward the service socket.

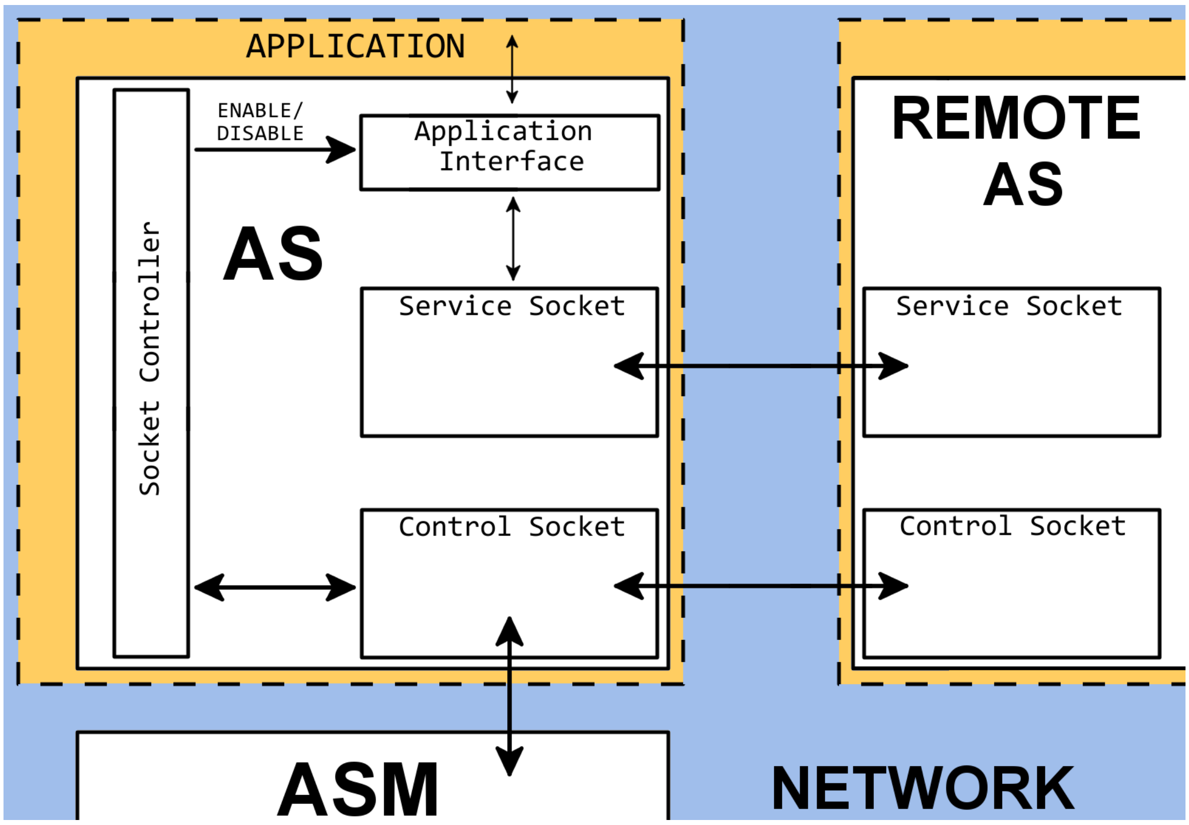

Figure 11 - Adaptive Sockets

4.4 System's components communication

The system architecture is designed to operate following a distributed model, which means that various components must synchronize across the network when it is needed to carry out an operation that involves multiple system components. The adaptive socket is intended to be instantiated inside the application. While the AS object is being constructed, it directly attempts to establish a secure link with the ASM.

4.4.1 Transport protocol reliability

The lack of support for connection oriented protocols in platforms with serious hardware resource limitation forces to design the ASM protocol to rely on the UDP capabilities. In order to carry out a typical ASM operation, which requires several messages exchange among components, it is fundamental to maintain a state of the various operations advancements. UDP requires some extra effort to improve its reliability level by implementing timers for retransmission, queues of messages waiting to be acknowledged and data integrity checks.

4.4.2 Authentication overhead

In traditional scenarios, clients and servers are able to authenticate each other and to establish a secure channel by means of asymmetric cryptography techniques and relative certificates. Due to inherent limitations of underling protocols, this framework cannot make use of data structures like certificates and asymmetric keys. Implementing authentication by means of digital signature would require a remarkable amount of memory space. Besides, exchange of large chunks of data, leads to high degrees of fragmentation with resulting network congestion. Some solutions provide support for authentication using asymmetric keys on resource-constrained devices. However, using asymmetric encryption for authentication makes no sense if done without the attached certificates.

4.4.3 Kerberos legacy

The software solution developed is inspired by Kerberos (Kerberos 2016) and especially how it utilizes symmetric cryptography for client authentication and session keys distribution. This approach allows accessing the adaptive security service by means of just two keys used for authentication and encryption. Further keys required for secure session creation are generated and distributed by the ASMs and partially by the adaptive sockets. The key distribution follows an imperative approach that is the key is established by one of the two endpoints. This approach lacks of supporting forward secrecy, nevertheless it was adopted in order to not complicate the protocol.

4.4.4 Authentication and Integrity

The protocol provides authentication of all messages exchanged among the entities appertaining to the same ASM network. This solution exploits techniques derived from both connection oriented and connectionless protocols.

The preliminary authentication to the system requires a full handshake with the exchange of nonces from both the parties, what occurs both for user/resource and ASM. This phase proves the identity of the service applicant and ensures that the message has not been replayed. This operation is performed only once, at AS instantiation time. Next operations do not require a nonce exchange. Instead these operations rely on a counter incremented at each operation and reset on session renewal. Here the overhead to maintain a progressive counter is justified by the fact that the communication channel toward the ASM remains open as long as the user/resource benefits of the service.

The protocol can count on several level of protection against attempts to undermine messages integrity ensured by HMAC and other techniques based on cryptographic algorithms. Creation of keyed digest is assisted by the following hash functions: SHA1-160, SHA2-256 e SHA2-384.

The choice of these algorithms follows the recommendations mentioned in the NSA suite B[1] document.

Authentication by means of AES-XCBC-MAC was provided because IPSec does not support AES-CCM (McGrev, 2012), the last one natively supported by all devices operating the IEEE 802.15.4 standard. It is defined in (Frankel 2003) and implemented for IPSec, which is useful for those wireless platforms with low computational capabilities but equipped with AES cryptography hardware accelerators. It produces a digest with a size equal to an AES block. However, this digest is truncated to 96 bits to comply with the IETF RFC.

The packet layout is similar to what used in the IPSec with a MAC section in the trail, see Figure 12. MAC computation follows the - encrypt then authenticate - technique, which avoids the overhead of decrypting the packet before being able to check its authentication and integrity. In case of invalid MAC the system discards the packet.

The ASM message is completely authenticated, including its outer header, which is left in clear for performance related reasons. The secure envelope contains either authentication nonces or packet sequence value; it depends on the operation it is associated to. This header can be checked at early times. This saves from performing operations such as decryption, integrity test and allocating space on thread's stack.

[1] https://www.nsa.gov/ia/programs/suiteb_cryptography/

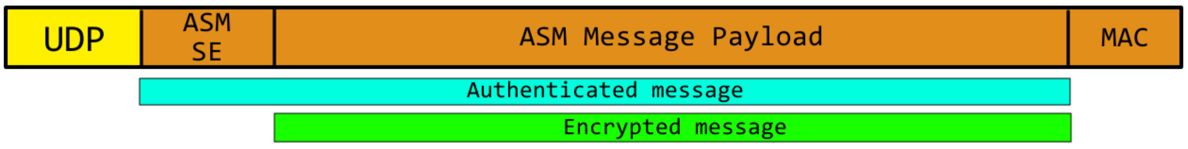

Figure 12 - Packet layout

4.4.5 Confidentiality

The protocol offers message confidentiality for the whole packet except the previously mentioned ASM secure envelope. The following algorithms are supported for encryption and decryption phases: ASM-CBC-128, ASM-CBC-192 and ASM-CBC-256.

The choice of these algorithms follows the recommendations mentioned in the NSA suite B document too. However, AES is widely supported by resource-constrained platforms. The algorithms require an IV 128 bits long and must be distributed alongside the encryption key. All the packets that are used to create, update or delete secure channel operations between two endpoints are secured by means of the PSK. Information exchanged for resources monitoring or secure link management data related to adaptive sockets is secured by means of session key.

4.5 Key management in resource-constrained networks

Figure 13 shows the key provisioning mechanism for distributed environments implemented in this solution. Each phase strongly depends from the previous one, the whole handshake is valid only if each phase has been completed with success and in the correct order. Each time a branch of the procedure breaks it gets tried again several numbers of times. Each attempt is delayed by a back off timer, which starts after 500 milliseconds from the first attempt and continues by doubling the previous value.

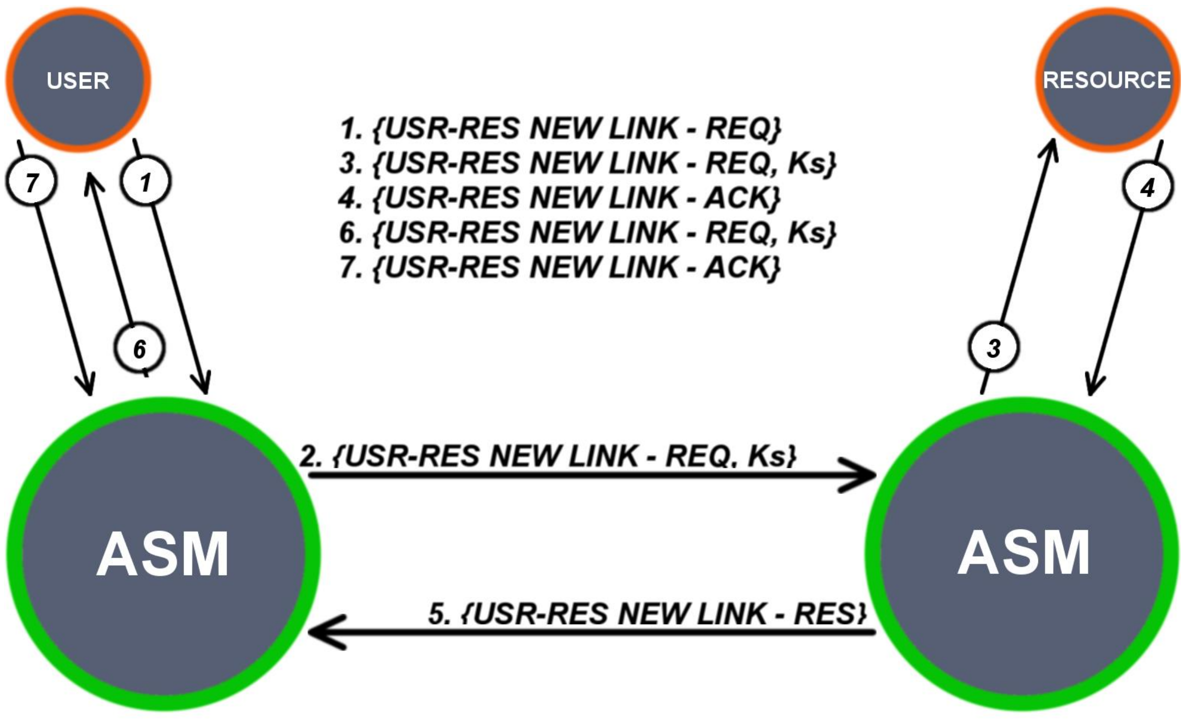
USER
RESOURCE
1. {USR-RES NEW LINK - REQ}
3. {USR-RES NEW LINK - REQ, Ks}
4. {USR-RES NEW LINK - ACK}
6. {USR-RES NEW LINK - REQ, Ks}
7. {USR-RES NEW LINK - ACK}
7
1
6
3
4
2. {USR-RES NEW LINK - REQ. Ks}
5. {USR-RES NEW LINK - RES}
ASM
ASM

Figure 13 - Session key distribution between user and resource

4.5.1 Secure service link establishment

The interactions between the RAI and a resource follow the client/server paradigm, in which the client issues a request and the server produces a response. The session keys distribution and the secure channel establishment comply with this interaction model. The ASM creates a master key that will be used by the RAI and a resource to establish their secure service link. The ASM first sends the master key to the resource and waits for its acknowledgment. The resource retain the key and acknowledges the receipt, it is now ready to establish a secure channel with the client. The ASM is aware that the resource successfully received the key and can now proceed to send the master key to the user. The RAI retains the key and acknowledges its receipt. Both RAI and resource have the shared secret. The RAI can then start the establishment of the session link with the remote resource because the protocol guarantees that the master key has been already delivered to the resource. The master keys delivered to the service consumers are static keys. They must be stronger enough to prevent the need to update them. However, these keys are used only for creating session links, thus the overhead is limited. Master keys delivery can occur directly when both user and resource are registered to the same ASM. Conversely, the operation requires the intervention of the ASM the resource is registered to.

4.5.2 Session link update

The link update procedure is triggered by the master ASM by broadcasting the request to all service consumers registered to it and to the other ASMs that take part to the adaptive security service provision. The request contains information about the new security parameters, and the session keys used to update the secure link with the ASM. The keys renegotiation proceeds as follows:

- Session links toward the ASM: the ASM sends the parameters and session keys to the service consumers it is authoritative for and its slave ASMs if any. The system components that received the update requests and the relative details reply with an acknowledgment. If any of the recipients missed the packet containing the keys, the ASM resends the update request but this time without the security parameters and session keys. The components that missed the former update and received a secondary update, are in charge to explicitly request to their authoritative ASM the new parameters and session keys.
- Session link between user and resource: they are agreed between user and resource by means of the master control link, without requiring the intervention of the ASM.

The first point allows speeding up the update procedure by sending directly the new session keys. In case someone missed the first update, the ASM adopts a conservative approach since some of them, for a number of reasons, might be definitively unreachable. In these cases keeping at sending keys is a considerable waste of bandwidth and hardware resource. The renegotiation of session links between user and resource avoids direct participation of the ASM, resulting in a considerable save in terms of requests traffic handled by the ASM.

4.5.3 Session link dismissal

The link dismissal procedure is agreed between the two endpoints that established it. Afterwards both send a notification to the respective authoritative ASM, thus informing it that the adaptive security service is no more required for this link. From this moment on the authoritative ASMs stop providing adaptability for it.

4.5.4 Development choices

The system development process was set up accordingly to the elicited requirements and several cornerstones have been formulated. The project is developed by using C++ with intensive use of the STL (C++ standard libraries). Modern STL versions provide a wide set of objects and functionalities which, unlike C, offer a reasonable level of abstraction from the operating system with a high degree of portability. Since STL lacks in network abstraction objects, sockets portability has been provided by Boost C++ libraries. This library is platform independent and provides a common interface to access network sockets and exploits their functionalities. Security of the whole system relies on OpenSSL library, with extensive use of its standard compliant implementations for the many secure algorithms available. The system status has been rendered persistent and able to recover after catastrophic failures, by exploiting MySQL DBMS3 functionalities. Its application allowed storing information related to users, resources, ASMs, parameters and keys of the secure links inside specifically defined tables.

4.6 Validation

This chapter aims at providing details about how much the implemented solution is effective in carrying out adaptive security related operations. The first part is dedicated at presenting results obtained by testing ASM operative response in the worst case. It will be than possible to infer system limits and the steps that would probably enhance the performances. The second part focuses on proving that the adaptive model is a sustainable approach for security provision in resource-constrained environments, by providing at support of it, estimations relatively at the amount of energy saved in a realistic use case.

4.6.1 Stress test

The worst case, mentioned in the chapter introduction, occurs when one ASM instance is in charge of carrying out the whole network update procedure and each service consumer is registered on the same network interface of the others. The maximum level of parallelism is obtained when the workload required to carry out system operations, is equally shared among the network interfaces. Conversely when the operations concern only one interface the parallelism is reduced to two levels only, that is only the Dispatcher and the UDPReceivers can operate concurrently. Indeed, since each thread that contribute to the Dispatcher operativeness, must to lock the interface as long as it takes to complete an operation, the other threads are thus forced to wait their turn and to work sequentially.

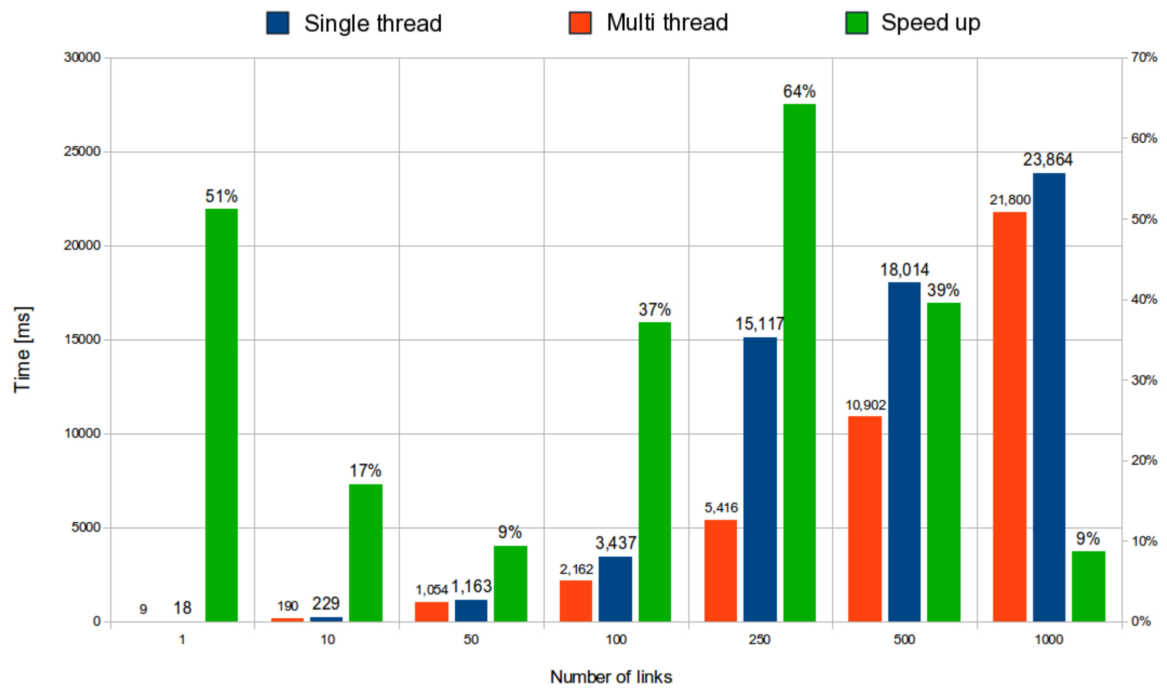

Figure 14 - System performances to carry out security level update in the worst case

The chart shown in Figure 14, reports the average time required by the ASM to perform a link update procedure. Results consider a multi-threaded execution, blue bars, and a single threaded execution, bars in orange. The green bars represent the gain that a multi-thread execution brings. The horizontal axis represents the number of nodes considered for the test. The behaviour of the system performs reasonably well up to one hundred links handled at a time. When the number of links exceeds this threshold, the centralized approach to network management shows its limits. Indeed the time spent in updating the nodes becomes considerable high, especially in scenarios the where requested system response time is very strict. It is now understandable why the system was meant to operate as a distributed architecture from the very beginning. A distributed system is able to carry out the same number of operations in a time t, whose values follow the equation: $t = \frac{T}{n} f(n,b)$. T is one of the possible time values defined in the chart, n is a discrete value relative to the number of ASM instances deployed in the network and b is the network bandwidth. The function f(n, b) models the network congestion and depends on the number of ASM instances. The values assumed by t reduces almost linearly for low values of n, however when b is constant, for high values of n, f grows at same rate of n and thus t stops decreasing. Thus exceeding a certain threshold of redundant instances deployed, the network reaches the bandwidth saturation point and throttles the system throughput. Wireless sensors adopt the IEEE 802.15.4 standard as communication protocol for physical and medium access control layers. Devices designed to be compliant with this protocol can reach a maximum transfer rate of 240 kbit/s. This threshold limits the ASM throughput by increasing the time required to complete a frame transfer between two nodes. In situations where the bandwidth is considerably limited, the time values exposed in the previous chart are likely to increase.

The number of keys issued per second, represents another interesting perspective to evaluate system performances. The chart in Figure 14, serves this purpose. The number of session keys issued per second, depicted in green, shows a relatively high keys throughput only for very limited number of managed links. However, these values have little importance considering the size of the links managed. The system throughput level quickly stabilizes at a value of 46 keys per second, as soon as the number of links concurrently updated exceeds the 10 units. The blue bars of the chart show the amount of network traffic generated to perform its task, considering a packet with a size of 186 bytes to carry the needed security parameters. It is evident that the results relative to the amount of data delivered to the network is a direct consequence of the number of session keys issued.

4.6.2 Energy spare and the adaptive security service

Some estimation has been realized to endorse the choice of deploying an adaptive security service at protection of a WSN communications. The results are based on actual measurements relatively to the amount of energy required to transfer packets containing the new security parameters and performing encryption and authentication operations to protect this information.

The scenario proposed here considers a wireless sensor intent at carrying out operations that requires transmission of data at regular intervals and, for sake of simplicity, only two levels of security.

- Suite A: AES-CBC-128 for encryption and HMAC-MD5 for authentication/integrity.
- Suite B: AES-CBC-256 for encryption and HMAC-SHA256 for authentication/integrity.

In order to make a reliable estimation of energy required to operate the adaptive security service, it is important to take two aspects in consideration: the energy required by the radio unit to transfer the required security parameters and the energy consumed by the hardware to perform encryption and authentication operations. The platform used to compute the following estimation has been the Telos, with this specification: MCU: MSP430; Clock: 8 MHz; Flash: 60 KiB; RAM: 2 KiB; Storage: 512 KiB; Radio: CC2420.

Sensor's hardware/software support for the AES encryption algorithm is explicitly recommended by the standard IEEE 802.15.4. It is widely adopted on a myriad of platforms; therefore, it is possible to easily setup operations. Table 9 reports the values extracted from (Othman 2012) and relative to operations carried out on one single block of data. The key setup column refers to energy required to initialize AES algorithm with the required key, however, this operation is performed just once for the whole set of data, regardless of its size.

Table 9 - AES energy requirements

Key size bits	Key Setup μJ	Encryption μJ	Decryption μJ
128	62.32	39.08	89.90
192	68.45	46.48	108.55
256	76.88	53.89	116.19

The values relative to energy consumed by HMAC-MD5 and HMAC-SHA256 on wireless sensors are more difficult to find. However, throughput performances in relation to AES are available. The energy required to transmit data is several orders of magnitude greater than what required performing one CPU instruction. The values in Table 9 are to be intended for the whole operations, which consists of thousand The Telos platform requires 15 µJ both for sending and receiving a byte of data (Culler 1998). The steps considered to carry out a link update procedure are as follows:

- The sensor receives the security parameters from the ASM and reset the control channel with the new session keys just received.
- The sensor starts a link updating procedure with its remote endpoints by sending the required security parameters.

5. Conclusion

In this deliverable, a dynamically adjustable security solution suitable for resource constrained devices was described. IMPReSS' security architecture can roughly be divided into two parts. First part of the security solution covers security from RAI to applications using the information produced by resource limited devices. Second part of the security solution starts from individual sensor node or other resource limited device and ends to Resource Abstraction Interface. Appendix I describes how the OAuth2.0 server can be installed and started in Linux environment.

This project has selected 6LowPan as a reference communication protocol for wireless sensor networks that can be part of IMPReSS platform. IPSec has been used to secure the communication in that the network. In order to support both high level of security and on the other hand need for long lifetime of battery powered wireless devices, flexible level of security is needed. In this task adaptive security manager has been designed and implemented. It can select the required level of security and inform the resource about the selected security level. Pre-shared keys have been used to secure communication. Based on the commands from adaptive security manager, wireless sensor can select the suitable key. In order to validate the approach, stress test for security level update was made and we have also estimated how much energy can be saved using lower level security when possible.

Deliverable also describes how communication between applications and IMPReSS platform is secured. Communication between applications and IMPReSS platform has typically been developed using standard web technologies such as HTTP and MQTT. Therefore we have also utilized standardized and well-known technologies for securing communication in application layer such as HTTPS and MQTT over TLS. In order to get access to protected resource, users, applications and clients willing to use that resource need to be authorized. We have analysed the requirements for authorization in resource limited devices and applications. Based on this analysis we have developed customized OAuth2.0 based authorization system especially targeted for IoT based systems, where applications can be extremely limited.

6. References

(Dargie et al 2010)	Dargie.W, Poellabauer.C,:Fundamentals of Wireless Sensor Networks, 2010
(Mosquitto 2016)	Mosquitto open source message broker, http://mosquitto.org/
(OAuth 2.0 2012)	The OAuth 2.0 Authorization Framework, 2012, https://tools.ietf.org/html/rfc6749
(Google 2016)	Google Identity platform , https://developers.google.com/identity/protocols/OAuth2ForDevices
(OAuth2.0 2010)	OAuth2.0 protocol draft, 2010, https://tools.ietf.org/html/draft-ietf-oauth-v2-05
(OAuth2.0 2015)	OAuth2.0 Token introspection draft, https://tools.ietf.org/html/draft-ietf-oauth-introspection
(Milagro et al 2008)	Milagro, F., Antolin, P., Kool, P., Rosengren, P., Ahlsén M. (2008). SOAP tunnel through a P2P network of physical devices, Internet of Things Workshop, Sophia Antopolis.
(Chen et al 2007)	Chen, Y.C., Liu, C.H., Wang, C.C., Hsieh, M.F. (2007). "RFID and IPv6-enabled Ubiquitous Medication Error and Compliance Monitoring System", 9th International Conference on e-Health Networking, Application and Services, 2007, 19-22 June 2007 Page(s):105 - 108.
(Jutvik 2014)	Jutvik V IKEv2 Implementation for 6LoWPAN based on Contiki OS. https://github.com/vjutvik/Contiki-IPsec.
(Raza et al. 2012a)	Raza S, Duquennoy S, Höglund J, et al (2012) Secure communication for the Internet of Things-a comparison of link-layer security and IPsec for 6LoWPAN. Secur Commun Networks n/a–n/a. doi: 10.1002/sec.406
(Raza et al. 2012b)	Raza S, Trabalza D, Voigt T (2012) 6LoWPAN Compressed DTLS for CoAP. In: 2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems. IEEE, pp 287–289
(Raza et al. 2014)	Raza S, Voigt T (2014) Lightweight IKEv2: A Key Management Solution for both the Compressed IPsec and the IEEE 802.15.4 Securitty. http://www.tschofenig.priv.at/sos-papers/ShahidRaza.pdf. Accessed 2 Dec 2014
(Kerberos 2016)	Kerberos network authentication protocol, http://web.mit.edu/kerberos/
(McGrev, 2012)	D. McGrew. AES-CCM Cipher Suites for Transport Layer Security. RFC 6655. July 2012.
(Frankel 2003)	S. Frankel and H. Herbert. The aes-xcbc-mac-96 algorithm and its use with ipsec. RFC 3566, RFC Editor, September 2003.
(Othman 2012)	S. Ben Othman, A. Trad, and H. Youssef. Performance evaluation of encryption algorithm for wireless sensor networks. In Information Technology and e-Services (ICITeS), 2012 International Conference on, pages 1{8, March 2012).
(Culler 1998)	David Culler, Joseph Polastre, Robert Szewczyk. Telos: Enabling ultra-low power wireless research, 1998.

Appendix I OAuth2.0 for IoT, installation

== Installation ==

1. Create virtualenv: virtualenv env
2. Activate virtualenv: source env/bin/activate
3. Install dependencies: pip install -r requirements.txt

== Startup ===

In development phase:
Startup: python manage.py runserver
Run tests: python manage.py test provider

In production:
1. Install nginx: pip install nginx
2. Install gunicorn: pip install gunicorn
3. Copy oauth_nginx file: cp oauth_nginx /etc/nginx/sites-enabled/ (edit the file if needed)
4. Run python manage.py collectstatic to collect all the static resources at location specified by STATIC_ROOT in settings.py
4. Restart nginx to serve static files and proxy: sudo service nginx restart
5. Start gunicorn to serve dynamic content: gunicorn oauth2_provider.wsgi --bind=127.0.0.1:8001

== Example oauth_nginx ==

```
server {
   listen localhost:80;

   location / {
      proxy_pass http://127.0.0.1:8001;
   }

   location /static/ {
      autoindex on;
      alias /home/janne/projects/impress/oauth2_provider/staticfiles/;
   }
}
```