D3.4.2

Client and server side security solutions for the smart grid

Protocols for authentication, authorization and credential distribution

Final report

NOTICE

The research leading to the results presented in the document has received funding from the European Community's 7th Framework Programme under the Grant Agreement number 619437.

Document Information

Call identifier	FP7-ICT-2013-11
Project acronym	**SUNSEED**
Project full title	**Sustainable and robust networking for smart electricity distribution**
Grant agreement number	619437
Deliverable number	D3.4.2
WP / Task	WP3 / T3.4
Type (distribution level)[1]	PU
Due date of deliverable	30.11.2016 (Month 34)
Date of delivery	23.12.2016
Status, Version	V1.0
Number of pages	21 pages
Responsible person, Affiliation	Herve Ganem GTOSA
Authors	Herve Ganem (GTOSA), Christian Richter (GTOSA)
Reviewers	Casper van den Broek (TNO), Jimmy Jessen Nielsen (AAU)

1

PU	Public
RP	Restricted to other programme participants (including the Commission Services)
RE	Restricted to a group specified by the consortium (including the Commission Services)
CO	Confidential, only for members of the consortium (including the Commission Services)

Table of contents

List of Figures

Abbreviations and acronyms

2.5G	2.5th generation mobile technology, GPRS
4G	4th generation mobile technology
AAA	Authentication, Authorization And Accounting
AAC system	Authentication And Access Control System
AAT	Authorization API Token
ACL	Access Control List
API	Application Programming Interface
CA	Certificate Authority
CAR management	Credentials and Access Rights management
CAR system	Credentials And Access Right System
CSR	Certificate Signing Request
HSM	Hardware security module
IoT	Internet of Things
IPoE	IP over Ethernet
LAN	Local Area Network
LAR	Local Aggregation Router
LTE	Long Term Evolution, a 4G technology
LTE	long term evolution, 4 G
LV	Low Voltage
M2M	Machine-to-Machine
PAT	Protection API Token
PKI	Public key Infrastructure
PPPoE	PPP over Ethernet
Radius	Remote Authentication And Dial-In User Service
RBAC	Role Based access control
REST API	Representational State Transfer API
RPT	Requesting Party Token
SDP	Service Delivery Point
SM	Smart Meter
SNMP	Simple Network Management Protocol
SOTA	State-of-the-art
UMA	User Managed Access
WAMS	Wide Area Measurement System
WAN	Wide Area Network

SUNSEED project

SUNSEED proposes an evolutionary approach to utilization of already present communication networks from both energy and telecom operators. These can be suitably connected to form a converged communication infrastructure for future smart energy grids offering open services. Life cycle of such communication network solutions consists of six steps: overlap, interconnect, interoperate, manage, plan and open. Joint communication networking operations steps start with an analysis of regional overlap of energy and telecommunications operator infrastructures. Geographical overlap of energy and communications infrastructures identifies vital DSO energy and support grid locations (e.g. distributed energy generators, transformer substations, cabling, ducts) that are covered by both energy and telecom communication networks. Coverage can be realized with known wireline (e.g. copper, fiber) or wireless and mobile (e.g. WiFi, 4G) technologies. Interconnection assures end-to-end secure communication on the physical layer between energy and telecom, whereas interoperation provides network visibility and reach of smart grid nodes from both operator (utility) sides. Monitoring, control and management gathers measurement data from wide area of sensors and smart meters and assures stable distributed energy grid operation by using novel intelligent real time analytical knowledge discovery methods. For full utilization of future network planning, we will integrate various public databases (e.g. municipality GIS, weather). Applications build on open standards (W3C) with exposed application programming interfaces (API) to 3rd parties enable creation of new businesses related to energy and communication sectors (e.g. virtual power plant operators, energy services providers for optimizing home energy use) or enable public wireless access points (e.g. WiFi nodes at distributed energy generator locations). SUNSEED life cycle steps promise much lower investments and total cost of ownership for future smart energy grids with dense distributed energy generation and prosumer involvement.

Project Partners

1. TELEKOM SLOVENIJE D.D.; TS; Slovenia
2. AALBORG UNIVERSITET; AAU; Denmark
3. ELEKTRO PRIMORSKA, PODJETJE ZA DISTRIBUCIJO ELEKTRICNE ENERGIJE D.D.; EP; Slovenia
4. ELEKTROSERVISI, ENERGETIKA, MERILNI LABORATORIJ IN NEPREMICNINE D.D.; ES; Slovenia
5. INSTITUT JOZEF STEFAN; JSI; Slovenia
6. GEMALTO SA; GTOSA; France
7. GEMALTO M2M GMBH; GTOM2M; Germany
8. NEDERLANDSE ORGANISATIE VOOR TOEGEPAST NATUURWETENSCHAPPELIJK ONDERZOEK - TNO; TNO; The Netherlands
9. TOSHIBA RESEARCH EUROPE LIMITED; TREL; United Kingdom

Project webpage

http://www.sunseed-fp7.eu/

Executive Summary

The SUNSEED Project involves the deployment of a large number of WAMS (Wide Area Measurement System) communication devices, continuously monitoring the functioning of the grid. Data measurements from WAMS devices are being sent periodically via secure publish/subscribe mechanisms to be received by a number of authorized applications.

For operational reasons, the deployment of the WAMS devices in the field needs to be as easy and automated as possible, and this involves in particular a simplification of the security bootstrap of the device applications, needed to authorize them to send their measurements and provide them with the credentials required to secure their communication.

This need to keep things simple and automated, led to explore the concept of “plug and play” security. In essence, master credentials are initially provisioned in a secure element embedded in the WAMS device. Upon the first network connection of the device, they are used to authenticate the device, generate shorter term application credentials, and obtain all required authorizations.

This deliverable focuses on this notion of “plug and play” security. Although this concept is not new in itself, it has been explored in SUNSEED in the context of IOT applications executing as part of one or more workflows. The workflow role(s) assigned to the different software entities executing in the device will define the sequence of security operations occurring at first network connection of the device. The methodology to achieve this “plug and play” workflow specific security constitutes the main innovation presented in this document.

Such “plug and play” workflow security can be implemented in two successive steps:

- **At manufacturing time:** the manufacturer of the communicating devices (WAMS) uses a secure element already personalized and containing master secrets (PKI [1] secret keys) from the secure element manufacturer. Related secrets (PKI [1] public keys) will later on be independently loaded in an authorization server. The manufacturer of the device (WAMS) can use the secure element as purchased from the secure element manufacturer and embed it in the device hardware without the need for any further operations.
- **At device deployment time**: all WAMS devices are flashed with the same software image, and differ only from the information contained in the embedded secure element. Upon first connection of each device to a communication network, an initial bootstrap script is executed which is tightly linked to the workflow scenario(s) to which the device applications should participate. This script results in the dynamic declaration in the authorization server of all the access rights needed for the device client applications in the role assigned to them in the workflow. It also prepares things so that the device’s applications may obtain the credentials needed to execute successfully.

A step-by-step description of the steps occurring in this initial bootstrap script and when the client application executes is provided for the SUNSEED “state estimation” use case.

1 Introduction

The previous SUNSEED deliverable D3.4.1 [2] has presented the security architecture used in SUNSEED.

During the preparation of the field trial involving the wide scale deployment of a large number of devices (WAMS-PMC, WAMS-SPM) the need to make the devices ready for “plug and play” deployment, particularly from the security standpoint, has progressively emerged.

After a brief recap of the security architecture in chapter 2, the “plug and play concepts” are detailed in chapter 3 and we show that from the security perspective two distinct steps for “plug and play” should be distinguished:

- Plug and play use of a pre-configured secure element to provision initial secrets in the WAMS device at device manufacturing time
- Plug and play for automated security bootstrap at device deployment time. This will detail the information stored inside the secure element embedded in the device and the way this information is used.

1.1 Relation to the rest of the project

The results presented in this report are used to implement the security mechanisms protecting the data communication from and to the SUNSEED devices, but also from and to the SUNSEED applications possibly executing in the cloud such as the **state estimation** application. The goal is to achieve end to end security on the data path connecting devices and applications.

2 SUNSEED Communication architecture summary

The overall communication and security architecture used in SUNSEED is displayed in Figure 1. In this figure WAMS devices (PMC or SPM type) either directly connected to the SUNSEED platform (devices using LTE modem) or located behind a router, are sending (publishing) data to the SUNSEED platform using either MQTT [3] or XMPP [4] publish/subscribe protocols.

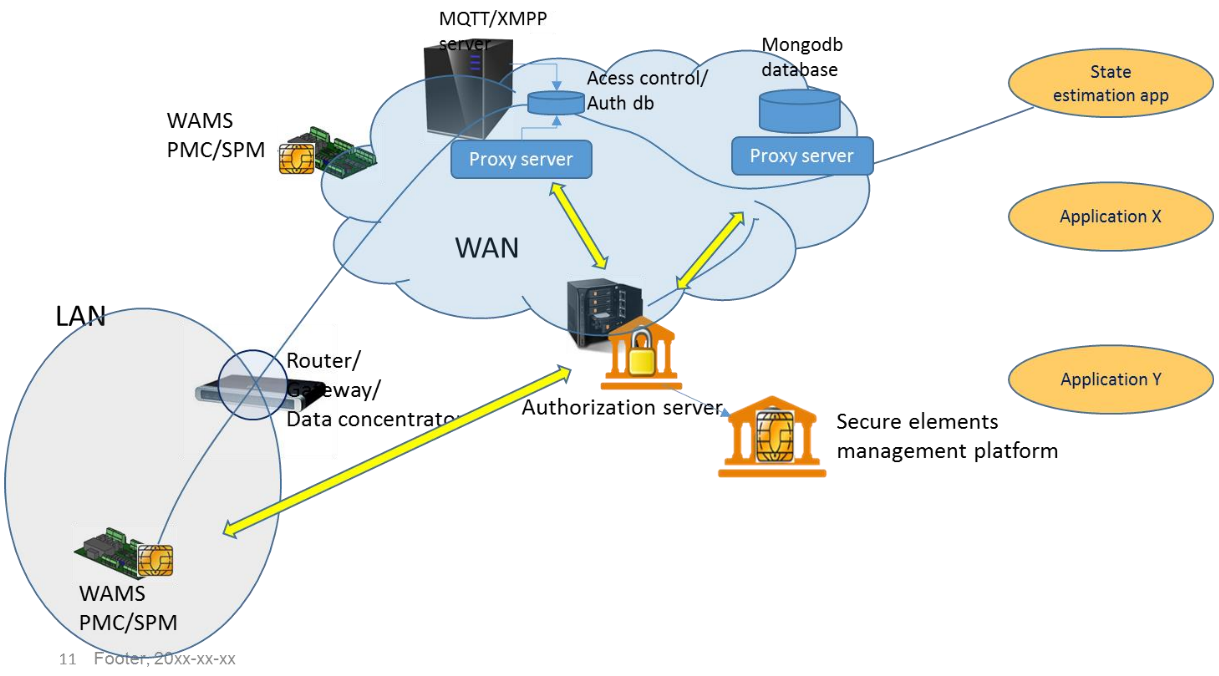


Figure 1: overall security architecture

On the right hand side of this picture, applications such as the state estimation application are subscribing to the data from the WAMS devices. The post computing results from the state estimation application are stored in a mongoDB data base part of the SUNSEED platform.

The protection at the transport level of the data link from the WAMS to the publish/subscribe servers and from the publish/subscribe servers to the SUNSEED application is achieved using the TLS protocol protected via PKI[2] certificates managed by the application management team. The protection at the network level is achieved using VPN managed by Telekom Slovenia IT team and is not discussed here.

The client certificates are issued and signed by the Authorization server acting as a certificate authority (CA). The whole SUNSEED ecosystem has been planned to accept and trust certificates delivered by the authorization server by distributing the root CA both in the devices but also to the applications.

[2] PKI involved the use of key pairs, one secret key which should be kept private and a public key which may be communicated to others. See https://en.wikipedia.org/wiki/Public_key_infrastructure

In this scenario both the publish/subscribe server and the mongoDB database are considered as resource servers controlling access to protected resources. For the publish/subscribe server the protected resources are the “publish” or “subscribe” operations on a particular topic. For the database, the protected resources are the databases and associated data, which should be read or written.

In a first step, the resources to protect are created in the resource servers. The administration of access rights to those resources is then delegated to the authorization server using the UMA Oauth rest API described in [2] (A1.2.3). At the end of this phase all the resources for which access is to be protected are described in the authorization server and it is possible to grant access to those resource for a particular software entity using the authorization server web interface.

Software entities may be either SUNSEED applications or the data publish clients running in the SUNSEED WAMS. The applications are declared in the authorization server via its Web user interface; whereas the data publishing clients will be automatically registered in the authorization server upon first connection of the WAMS to a network, typical in the deployment operation.

Software “clients”, in our case running either in the WAMS devices or in the cloud first have to negotiate access rights for the resources they need to access. This happens once during a security bootstrap phase.

The access right decisions made by the authorization server are materialized by the delivery of access tokens delivered to the software clients. When resource servers are OAUTH compliant, they are, after verification of the access tokens presented by the clients, able to abide to the decision made by the authorization server and grant or deny access to their protected resources according to this decision.

Resource proxies are needed for resource servers which are not OAUTH compliant and rely upon their own authentication and access control (AAC) system. In this case the proxy server communicates with the authorization server using the protected API described in [2] (A1.1.3). They perform appropriate modification upon the resource server AAC system. In this case the access token delivered to a software client actually contains the credentials required for this client to authenticate with the resource server and be granted access to needed resources.

3 Plug and play security

During the planning of the SUNSEED field trial the need emerged to simplify WAMS device deployment and move as close as possible to a “plug and play” model, where a random device can be deployed with a simplified deployment process at a given location. This excluded lengthy procedures to configure the device and manual provision of credentials at deployment time.

Before WAMS are deployed, the Edison board[3] embedded inside the device needs to be flashed with a software image. However, for the sake of simplicity, the same image is typically flashed on all devices and this excluded the possibility to simply provision credentials on the device before deployment.

A process was needed that would result in an automatic customization of the Linux image upon first connection of the WAMS to a network connection.

3.1 WAMS manufacturing and deployment process

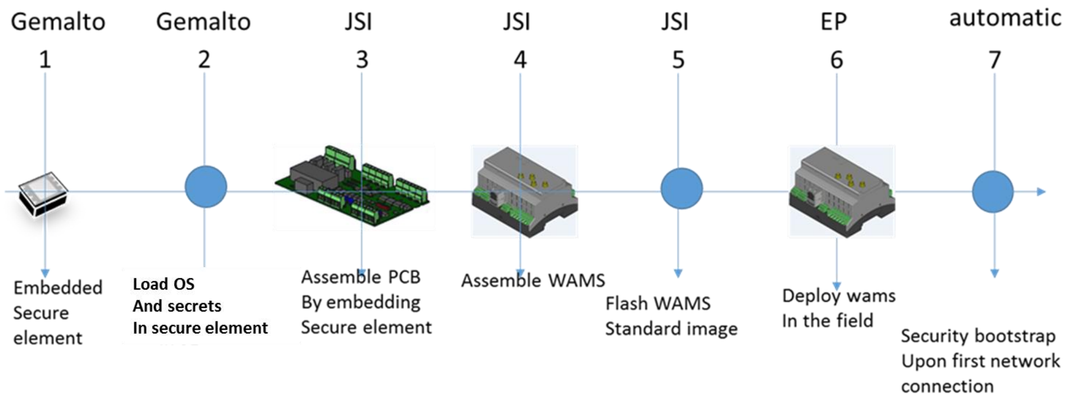
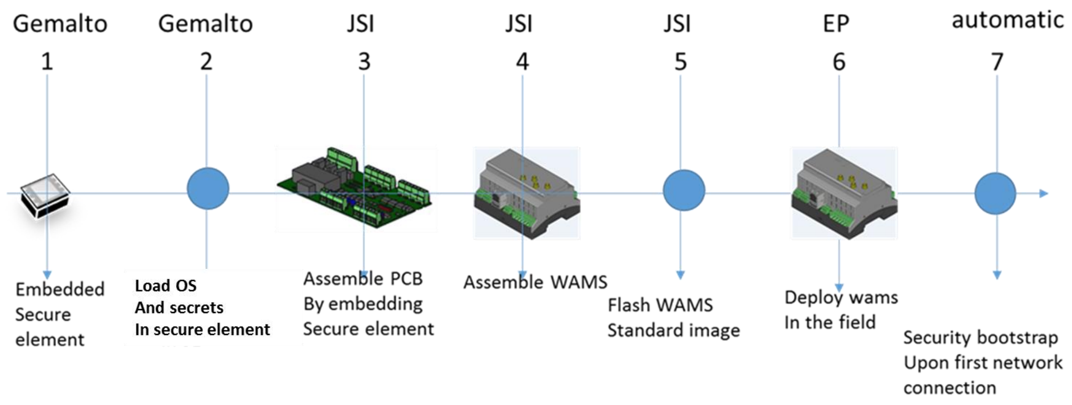

Gemalto
1
Gemalto
2
JSI
3
JSI
4
JSI
5
EP
6
automatic
7
Embedded Secure element
Load OS And secrets In secure element
Assemble PCB By embedding Secure element
Assemble WAMS
Flash WAMS Standard image
Deploy wams In the field
Security bootstrap Upon first network connection

Figure 2 outlines the WAMS manufacturing and deployment process used in SUNSEED. The different steps of this process may be described as follows:

[3] Linux Board from Intel constituting the computing core of the wams. See https://software.intel.com/en-us/iot/hardware/edison



Gemalto
1
Gemalto
2
JSI
3
JSI
4
JSI
5
EP
6
automatic
7
Embedded Secure element
Load OS And secrets In secure element
Assemble PCB By embedding Secure element
Assemble WAMS
Flash WAMS Standard image
Deploy wams In the field
Security bootstrap Upon first network connection

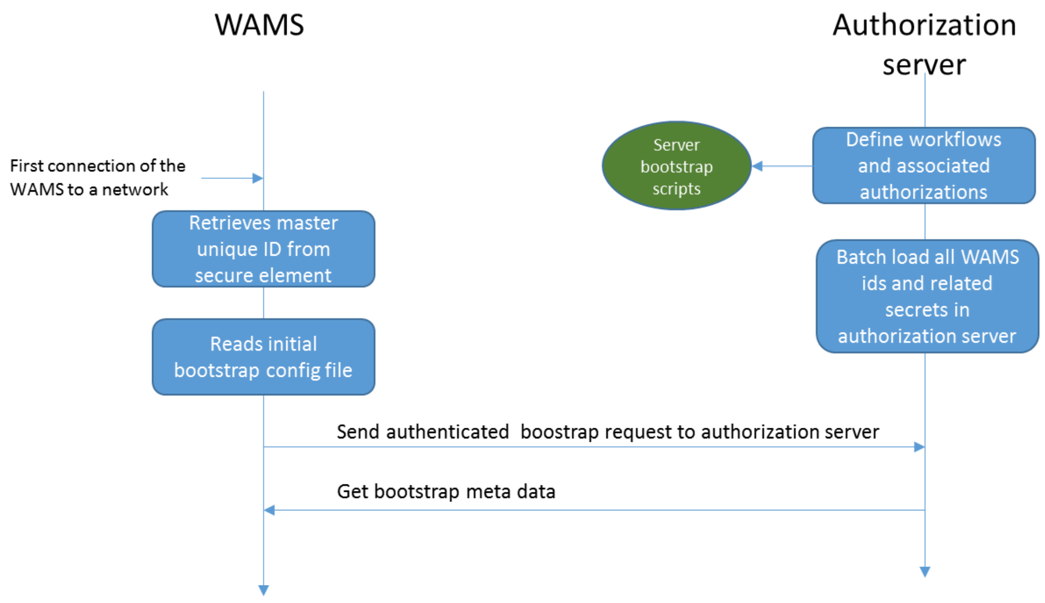
Figure 2: SUNSEED WAMS manufacturing and deployment process

1. Provision embedded secure element chips (Gemalto)
2. Personalize secure elements by loading Secure element operating system as well as initial master IDs and secrets. Each secure element will be identified with a master ID associated to a root secret (Gemalto)
3. The personalized secure elements are used as such for the manufacturing of WAMS (JSI responsibility). They are embedded on the WAMS PCB
4. The various PCB parts of the WAMS are assembled together and tested in order to produce a functional WAMS device
5. The Edison board in the WAMS is flashed with the Linux image, **which is common to all WAMS devices**. Such an image includes SUNSEED data acquisition and transmission software, as well as a security initial bootstrap application which will be executed at deployment time when the WAMS device is first connected to a network. This step completes the WAMS manufacturing process and results in WAMS devices that are ready to be deployed
6. The WAMS devices are deployed in the field. Upon first connection of each device to a network connection, the initial security bootstrap script is executed, which results in both the personalization of:
    - The Linux image to include identifiers used to authenticate the software clients executing on the WAMS,
    - The secure element data with the creation of derived secrets used to authenticate the clients executing on the WAMS
7. Each application executing on the WAMS performs an application security bootstrap to obtain needed access rights and credentials. If the credentials delivered are ephemeral, the application bootstrap process will have to run again to renew the credentials once they expire. There is one application security bootstrap per application. Such bootstrap operation It is not the same as the initial security bootstrap which is executing when the WAMS is first connected to a network, and prepares the grounds so that all application bootstrap scripts may later execute

3.2 Initial security bootstrap

The initial security bootstrap process occurring upon first connection of the WAMS to a network will involve a dialog between a WAMS device and the authorization server.

The steps involved in this initial bootstrap process are summarized in 3 and may be described as follows:


WAMS
Authorization server
First connection of the WAMS to a network
Retrieves master unique ID from secure element
Reads initial bootstrap config file
Server bootstrap scripts
Define workflows and associated authorizations
Batch load all WAMS ids and related secrets in authorization server
Send authenticated boostrap request to authorization server
Get bootstrap meta data

3: Initial security bootstrap overview

1. In a first step, before the WAMS device is deployed, a workflow is defined. This workflow will involve a number of software entities that communicate together, executing either in IOT devices (like the WAMS) or on gateways or cloud servers. For example, a SUNSEED workflow involves WAMS software clients publishing WAMS power measurement data via the MQTT protocol on a specific topic, and the state estimation application, executing on a remote node subscribing to the information published by the WAMS. A workflow will be associated to the definition of a number of roles and attached access rights. With our example workflow, we may define 2 roles: “power measurement publisher” and “power measurement subscriber”. The first one will require the right to publish on a specific MQTT topic, and the second the right to subscribe to data published on this topic.
2. The definition of the roles involved in the workflow and attached access rights leads to the definition of “an authorization server bootstrap script” to be loaded in the authorization server. Such a script will perform in an automated mode the definition of the proper roles, possibly create clients associated to the WAMS in the authorization server and in resources servers, grant the associated access rights for each connecting WAMS node. This script will be triggered and executed independently for each WAMS node at the time of execution of the initial bootstrap script.
3. All IDs and secrets associated to a WAMS are batch loaded in the authorization server and each such preloaded WAMS data is associated to one or more bootstrap scripts (A WAMS may participate to more than one workflow).
4. When a WAMS device is deployed and first connected to a network, the initial bootstrap script executes. The master unique ID is retrieved from the secure element. The retrieval of such ID does not require any credentials (such as pin code). An authenticated REST request is made to a bootstrap API endpoint in the authorization server. This REST request is authenticated using the **master secret key** stored in the secure element, and the associated **master public key** which has been preloaded in the authorization server. The use of the master secret key of the secure

element to perform this authentication does not require a pin or other secret to be presented to the secure element. The network address of the authorization server as well as a few other parameters are read by the bootstrap script from a configuration file (the same for all WAMS), part of the WAMS common Linux image.

5. Upon reception of the REST request and after authentication of the device, the authorization server executes the bootstrap scripts mentioned above, which possibly creates roles and associated access rights, possibly create clients associated to the requesting WAMS in the authorization server and in resources servers. A number of “derived parameters” are computed by the script and sent back with the response to the WAMS. In some cases a pin code is returned that will enable the initial bootstrap script on the WAMS to request some operations on the secure element such as the generation of further keypairs, later used to generate signed client certificates.

3.3 Application security bootstrap

Once the initial security bootstrap has been executed, the WAMS device Linux image has been configured and the access rights needed for the workflow have been defined automatically in the authorization server. Applications planned to be executed on the WAMS may start executing and the initial bootstrap script is not needed any more.

The pseudo code of a sample application requesting credential for publishing data using the MQTT protocol has been described in [2] (section 4.3.4)) and is reproduced below:

```
Var mqttBroker=my.mqttbroker.com
Var myTopic=« helloTopic »;
Var SecurityConfigFile=”/etc/myconfig”

//security bootystrap will obtain an AAT token from the authorization server if needed; token will be
stored in the security config file
if (!securityBootstrap(SecurityConfigFile))
  exit ()

//security credentials are not statically defined; they are obtained dynamically, and possibly stored in
the config file; several steps have been aggregated inside the getCredentials function.
Var myCredentials=getCredentials(mqttBroker);

// dynamically request access rights to publish on “myTopic” topic
If (!getPermission(myTopic, »publish »))
    exit()  // could not get permission or permission expired; may attempt to renew

// now everything has been set; connect as usual
Var handle=  Mqtt_Connect(myCredentials, mqttBroker)
If (handle)
  handle.publish(myTopic,   »hello World »)
```

This code involves a succession of 3 calls:

- **securityBootstrap** will authenticate the client with the authorization server and obtain a token

(AAT token) enabling the client to request authorizations and credentials from the authorization server

- **getCredentials** will return the credentials needed to connect to the MQTT server
- **getPermission** will request the permissions to be defined to be able to publish on a particular topic

This show that the software client will perform another security bootstrap in order to obtain the application credentials needed.

In the case of automated deployment, the last call: getPermission requesting access rights to publish on one topic is redundant because all permissions have already been set up by the initial bootstrap script.

3.4 Recapitulation of the process

When a WAMS device is deployed on the field, an initial security bootstrap takes place. The bootstrap client authenticates with the authorization server using master secrets stored in the secure element and triggers the definition of the roles and access rights for all software clients, planned to execute on the WAMS.

Finally, the initial bootstrap script creates bootstrap credentials (in the secure element) for each of the software clients planned to execute on the WAMS. Those credentials will enable the client applications to authenticate with the authorization server and obtain credentials enabling access to the resources servers planned in the workflow (AAT token first, then credentials) during the application security bootstrap

4 Practical application: security flows for SUNSEED state estimation use case

The implementation of the security has been done in SUNSEED using the APIs described in [2] . We describe here a practical implementation done with the state estimation use case explained below, one of the SUNSEED use cases. This implementation is based upon the general principles explained before, but contains its own specificities. It also provides the opportunity to point out how security may be improved on some specific points.

A software client in the WAMS is gathering power measurements from the WAMS hardware and publishing such measurements on a specific MQTT topic. The WAMS is using a secure element, to provide secure storage for credentials. The secure element embeds a PKI [1] application. The functions exposed by this application are:

- Create or delete key pair
- Cipher data (return ciphered input data)
- Sign data (returns signed input data)
- Generate certificate signing request. (a message sent from an applicant to a Certificate Authority in order to apply for a digital identity certificate)

The secure element of the WAMS are personalized at manufacturing time with the following elements:

- A unique master ID
- An already generated master private key
- A pin code necessary to request most operations on the secure element
- A secret used as a seed by a pseudo random generator embedded in the secure element. We will refer to this secret as the "**OTP secret"**

The following information for each WAMS is preloaded in the authorization server (before any communication between the WAMS and authorization server has taken place):

- The Secure element master ID
- The public key corresponding to the master private key in the WAMS
- The pin code
- The OTP secret
- The authorization bootstrap script(s) as described in section 3.1

In a first phase and before any communication takes places an MQTT topic is created in the MQTT broker and administration of the access rights to this topic is delegated to the authorization server using the UMA API described in [2] . From that point in time, access rights to the topic may be managed by the authorization server.

During the initial bootstrap occurring when the WAMS device is deployed and first connected to a network, the initial bootstrap script will retrieve the ID of the secure element and issue an authenticated REST request to the authorization server. The authentication of this request will rely on the master public and private key associated to the particular secure element. The private master key was stored in the secure element when it was personalized; the public master key has been preloaded inside the authorization server.

Upon reception of the authenticated bootstrap REST request, the server will execute the bootstrap script(s), create the “power measurement publisher” role if it does not exist already, and setup publish permission on the MQTT topic for this role. Then it creates a specific client (identified by a **client ID** and which will be authenticated with a **Client secret**) in the authorization server and associates it to the role of “power measurement publisher”.

The bootstrap REST request returns a pin code to the bootstrap script to perform operations on the secure element, the client ID characterizing the authorization server client and the associated Client Secret. Client ID and Client Secret will be stored in a file in the file system. The pin code is kept temporarily in memory and not stored in the file system.

Using the pin code the Initial bootstrap script requests the creation of a key pair in the secure element. This key pair (the MQTT key pair) will be used to authenticate the MQTT client with the MQTT server. This completes the initial bootstrap script.

The actual MQTT client application is then started and performs two steps:

1. The **securityBootstrap** call involves calling the token endpoint API to obtain an AAT token as described in [2] (A1.2.1.2). This REST call includes the client ID and Client Secret obtained in the previous call. However, the storage of such secret in an unsecure storage creates a security weakness as stealing the secret can enable to impersonate the MQTT client application. So, a better way to do things is to use the secret and an pseudo random generator application in the secure element to derive a One-time password that will be used instead of the Client secret in the request to the token API endpoint. This request should return an AAT token in the response as described in [2].
2. The **getCredentials** will use the AAT token to request the delivery of the credentials to connect to the MQTT server (in our case a client certificate). This happens in two steps:
    - The client makes a request to the resource proxy requesting access to the MQTT server. The response to this request contains a document (the ticket) listing of security profiles supported by the MQTT server (typically PKI certificates or username/password). If the PKI certificate profile is supported, the client sends a request to the authorization server CSR endpoint including the AAT token in the request header. The response of this request includes a pin code and a username. Using the pin code the client requests a CSR generation from the secure element where the common name field is set to the username.
    - The client makes a REST request to the authorization server RPT end point API as described in [2] (A1.2.2.1) passing as parameter the CSR and the ticket obtained in the previous step and gets back in the response a signed client certificate to be stored in the file system. This client certificate will be used to authenticate the client with the resource server. However, such an authentication will need to use the private key securely stored in the secure element. This explains why the client certificate may be stored in the file system. It is only useful along with the associated private key, and therefore may hardly be used outside the WAMS device.

In order to improve the security, the AAT token included in the header of the two last requests should be replaced by a one-time password obtained from the AAT token itself as described above.

5 Conclusion

One specificity of the SUNSEED project is the use of an embedded secure element in the WAMS, the communicating device used in the project. While a previous deliverable [1] has presented the security architecture and REST APIs, the need for “plug and play” security has emerged in the preparation of the field trial deployment and this document explained how this need has been addressed.

Plug and play security relying upon a secure element on the device is in fact implemented in three distinct steps:

The first one involves the personalization of the secure element at manufacturing time, which can then be embedded in the IOT device without further operation.

In a second step associated secrets and parameters are loaded in an authorization server before device deployment.

The third and final step occurs when WAMS devices are deployed in the field in an initial bootstrap configuration script specific to the workflow(s) in which the WAMS client applications are involved. This initial bootstrap script prepares the ground for application specific bootstrap scripts executed with the applications, resulting in the delivery of application specific credentials. The initial bootstrap script is executed once upon first connection of the device to a network, while the application bootstrap scripts are executed whenever there is a need to obtain or to renew credentials.

A step by step description of the operations involved has been provided for The SUNSEED state estimation use case.

6 References

[1] Johannes A. Buchmann and Evangelos Karatsiolis, *Introduction to Public Key Infrastructures*, springer, Ed., 2013.

[2] SUNSEED Project, "Deliverable D3.4.1: Security architecture for smart grids," 2015.

[3] OASIS, "MQTT Version 3.1.1," 2014. [Online]. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

[4] XMPP standards fondation. XMPP specifications. [Online]. http://xmpp.org