Multi-layered Security Technologies

for hyper-connected smart cities

D4.9: M-Sec overall end-to-end security

December2019

Grant Agreement No. 814917

Multi-layered Security technologies to ensure hyper-connected smart cities with Blockchain, BigData, Cloud and IoT

Project acronym	M-Sec
Deliverable	D4.9 M-Sec overall end-to-end security
Work Package	WP4
Submission date	December 2019
Deliverable lead	CEA / KEIO
Authors	CEA, KEIO, WLI, YNU, WU, NII
Internal reviewer	ICCS, WU
Dissemination Level	Public
Type of deliverable	DEM
Version history	- v00, 08/11/2019, CEA, Full ToC, assignments and initial content - v01, 26/11/2019, CEA, introduction and initial contribution - v02, 04/12/2019, KEIO, initial contribution - v03, 10/12/2019, CEA, second contribution - v04, 11/12/2019, KEIO, second contribution - v05, 20/12/2019, CEA, final integration, ready for review - v06, 27/12/2019, CEA, reviews integrated - v07, 30/12/2019, CEA, ready for submission

The M-Sec project is jointly funded by the European Union's Horizon 2020 research and innovation programme (contract No 814917) and by the Commissioned Research of National Institute of Information and Communications Technology (NICT), JAPAN (contract No. 19501).

Table of Contents

List of Figures

Glossary

AAA	Authorization, Authentication and Accounting
API	Application Programming Interface
CA	Certification Authority
CoAP	Constrained Application Protocol
CSR	Certificate Signing Request
DNS	Domain Name System
IETF	Internet Engineering Task Force
JSON	Javascript Object Notation
LDAP	Lightweight Directory Access Protocol
MQTT	Message Queuing Telemetry Transport
PKI	Public Key Infrastructure
REST	Representational state transfer
RPC	Remote procedure call
SASL	Simple Authentication and Security Layer
SSH	Secure Shell
SSL	Secure Session Layer
TLS	Transport Layer Security,

1. Introduction

1.1 Scope of the document

We present in this document the reference design demonstrating M-Sec end-to-end security as worked out in task 4.5. End-to-end Security has a particular approach in M-Sec Project and WP4 as it is a combination of the output from the 4 other tasks of the project, completed by a Security manager ensuring a secured and smooth interoperation of each element of the architecture. The functions provided by this security manager can be used directly by the assets of each task or can be interfaced using the two main middlewares of the project, inherited from the BigClouT project: SOXFire and sensiNact.

1.2 Relationship to other workpackages and tasks

The deliverable D4.9 regarding end-to-end security is strongly related to tasks 4.1 (IoT Security), 4.2 (Cloud and data-level security), 4.3 (P2P level security and blockchains), and 4.4 (Application level security) as shown in Figure 1.

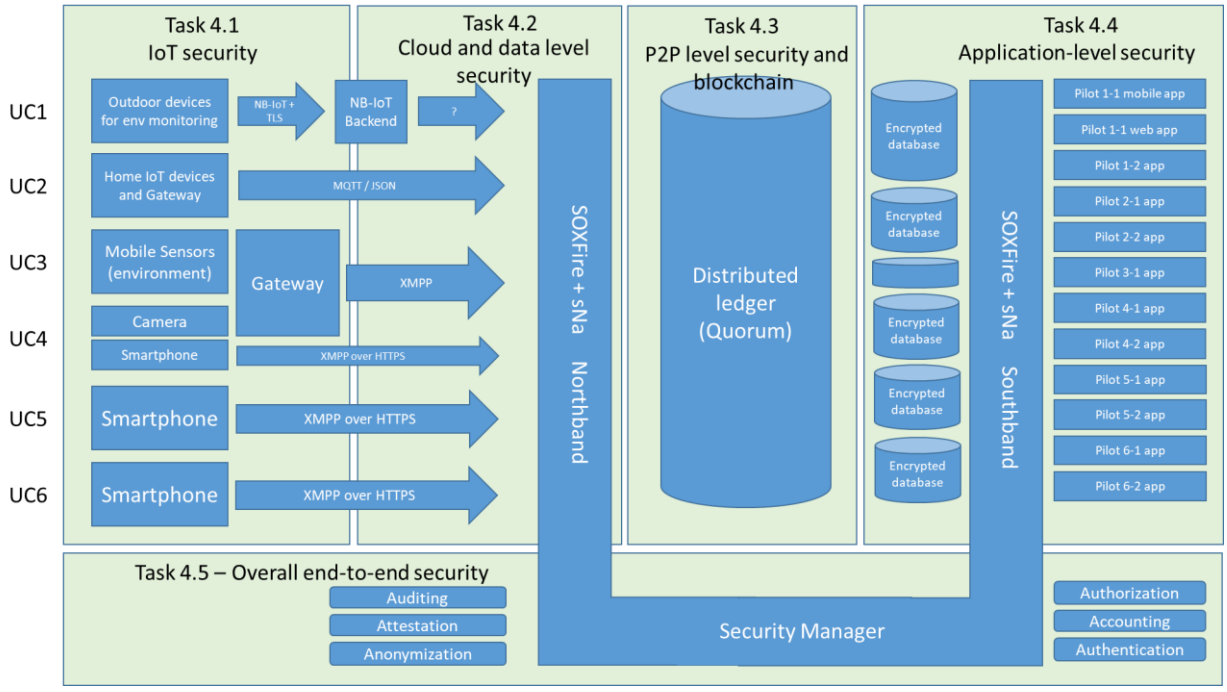

Figure 1. Overall M-Sec topology

1.3 Methodology followed

The M-Sec WP4 is built upon an IoT reference model, having in our case four distinct domains: IoT, Data and Cloud, P2P with Blockchains and Applications. Each one of these domains is securing itself by integrating state of the art technologies. Such technologies demonstration is proposed with

deliverables 4.1, 4.3, 4.5 and 4.7. Thus, having end-to-end security requires to have common functions to ensure a continuum within many aspects such as authorization, authentication, anonymization, attestations, etc.

2. Security Manager

2.1 General Description of the Prototype

The security Manager is a set of centralized security functions that are necessary to ensure end-to-end security, privacy and therefore digital trust. It is designed to support several security functionalities aggregated in a single backend using the LDAP standard, as described in Figure 2.

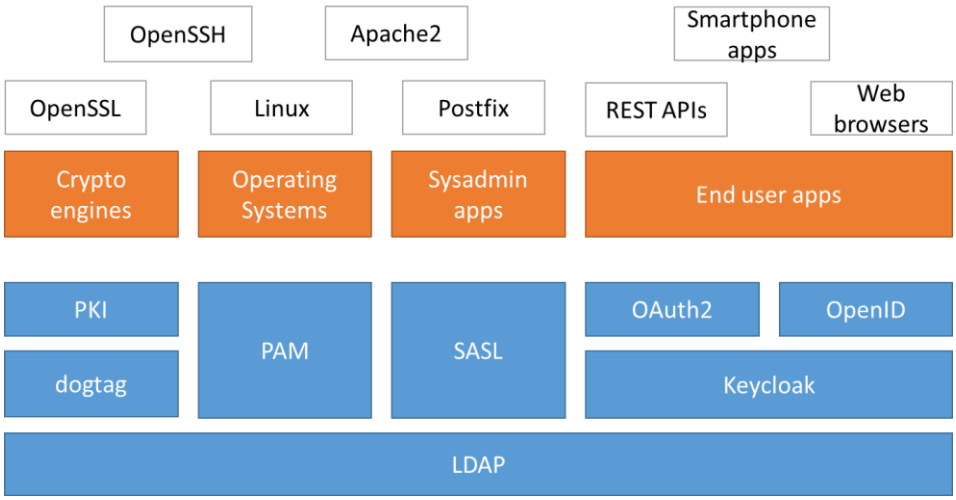

Figure 2. Stack for the M-Sec "Security Manager" with components described in blue

In terms of M-Sec implementation, this security manager is planned to be integrated into the project as follows:

- Within IoT devices in T4.1 for provisioning devices with initial manufacturer credential for firmware authentication
- In T4.2 between IoT devices and the cloud for communication authentication and encryption. This is most likely an encrypted channel between IoT devices and M-Sec middleware such as SOXFire and sensiNact. Firsts steps of this integration are described in the next demonstrators.
- In T4.3 to provide marketplace authentication and authentication coordination between SOXFire/sensiNact and the blockchain implementation
- In applications within T4.4 in order to authenticate end-users in the system.

2.2 Components

Directory service

The central element for the security manager is a directory service containing all information to manage security services for clients, such services known as AAA for Authentication, Accounting and Authorization. In the implementation form, we use an LDAP directory as it is commonly accepted as a free and open standard from the IETF.

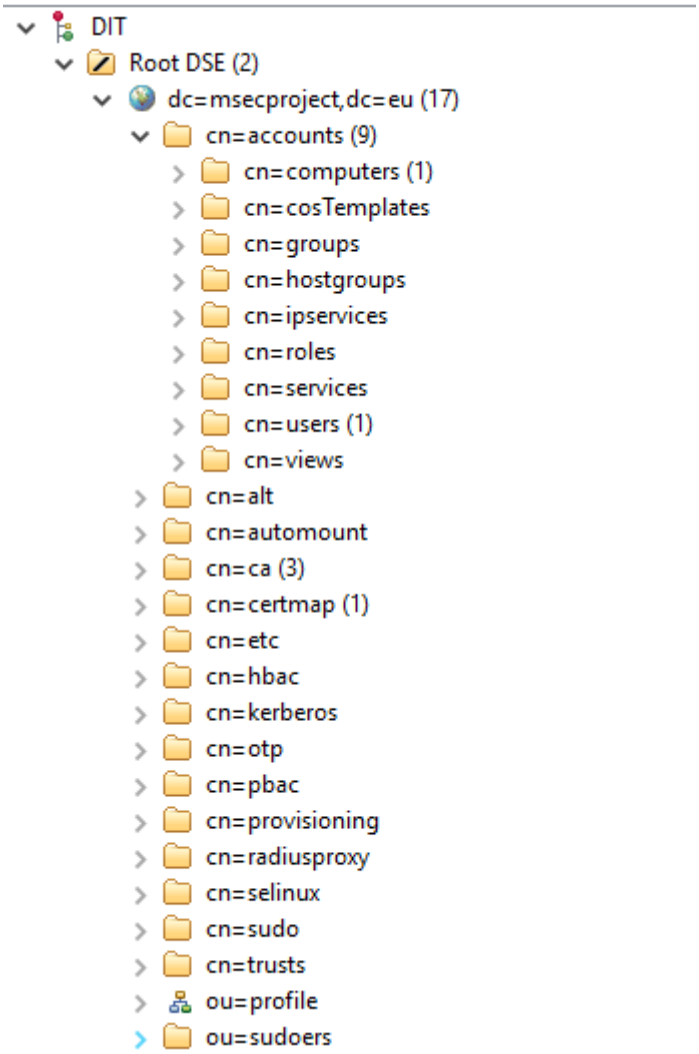

Figure 3. The first level of the directory service for the security manager

Public Key Infrastructure

Public key infrastructure is an essential component in modern communication patterns enabling asymmetric cryptography between untrusted parties. A public key infrastructure enables to bound public keys to identities (such as people or devices). It is usually managed by an authority associated with many roles such as validation authority, registering authority, etc.

We use the dogtag PKI solution, which has native capabilities to use LDAP as described earlier.

2.3 Package Information & Installation Instructions

The demonstration in this first version aims to show how a client can talk to an endpoint with a secure channel managed by the Security Manager. It handles the configuration of the security manager, the enrolment of the user and the configuration of the endpoint.

Required Tools and dependencies

We assume that all three entities, the security manager, the client and the endpoint are running a Linux Operating System. In the demonstration, the client is a Raspberry Pi, the endpoint and the security manager are both virtual machines.

DNS

One of the first steps is to have a proper DNS configuration, even if services are running locally. To reflect this configuration, we force the name resolution in the /etc/hosts file as follows:

```
10.255.0.242      manager.msecproject.eu
10.255.2.168      client.msecproject.eu
10.255.2.150      endpoint.msecproject.eu
```

It is important for the security manager that these entries are placed at the very beginning of the file as only the first resolution is taken into account while resolving certificates domains.

FreeIPA

We use the FreeIPA toolkit which integrates many components, yet to be configured, such as an LDAP server, the dogtag PKI and Kerberos system. FreeIPA is available as a package on most Linux distributions. An installation script provides an interactive configuration tool.

```
apt-get install freeipa-server
/usr/sbin/ipa-server-install
```

Note that FreeIPA is not yet fully supported on Debian-based targets and require unstable packages (sid). Installation is most likely to fail and require few tweaks.

More information about installing FreeIPA is available here:

https://www.freeipa.org/page/Quick_Start_Guide

Download and Run Demonstrator

The demonstrator shall be composed of three distinct machines: a device (Linux based), an IoT backend and the Security Manager. We describe here steps to deploy the security information. In this setup, we handle two kinds of flows:

- TLS encryption for HTTP based exchanges (REST)
- SSH tunnels for legacy/third party unencrypted data streams

Configuring FreeIPA

Once FreeIPA has been installed and initialized with the previous steps, one can log into the management page following the URL https://[domain name filled during installation]. Once logged in, we can manage the user on the webpage described in Figure 4.

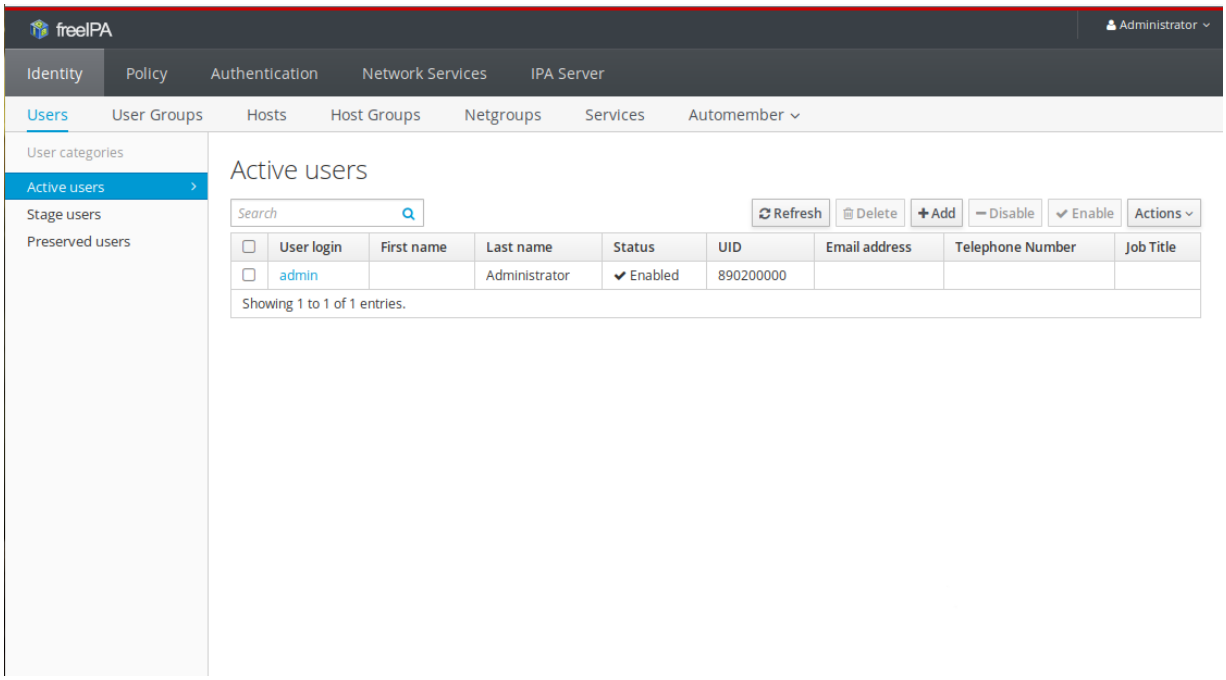

Figure 4. Users management page

From this webpage it is possible to add a user or to edit an existing user as described in Figure 5

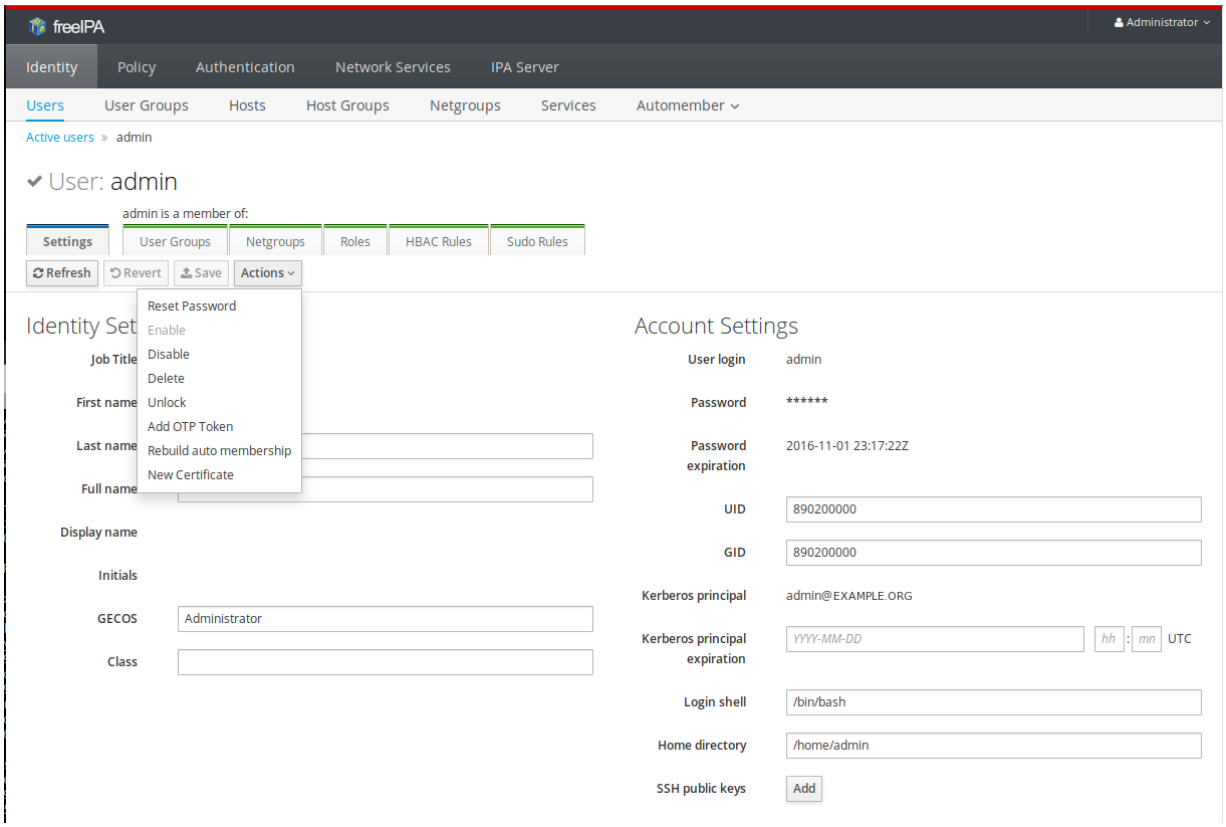

Figure 5. User edition

We assume that we have a user “demo” for the next steps. The user is just initialized and doesn’t have any SSH public key or any certificate attached to it.

In addition, we initiate a signing certificate authority for SSH. SSH uses its own key system and is not compatible with a PKI. At the end of this demo, each user will have two public keys: one for SSH and one bound to the PKI.

To generate an SSH CA, we use the following command which will generate two files: ca.pub as the public key and ca as the private key. This last key shall be stored in the safest place possible.

```
ca$ ssh-keygen -t rsa -N '' -C 'admin@msecproject.eu' -f ca
```

Configuring the Client

The client is bound to a user in FreeIPA. This user in our case is ‘demo’. It will use this account bound to each outgoing connection, whether it’s an SSH tunnel or a natively encrypted tunnel.

SSH Tunnel for the client

For the SSH tunnel, we need to generate a key with the FreeIPA account using the following command.

```
client$ ssh-keygen -t rsa -N '' -C 'demo@msecproject.eu' -f ~/.ssh/id_rsa
```

Then we can copy the generate file “id_rsa.pub” and copy it to the CA in order for the CA to sign it, making it trustful for other peers managed by this CA.

```
ca$ ssh-keygen -s ca -I 'demo@msecproject.eu' -n demo -O "clear" -O "permit-port-forwarding"
id_rsa.pub
```

This command will output a file named “id_rsa-cert.pub” that can be inspected with the following command, especially to validate some extra options regarding the SSH capabilities (shell, X11 forwarding, etc).

```
ca$ ssh-keygen -L -f id_rsa-cert.pub
```

We can then copy back the signed certificate to the client and configure it properly to use this certificate and to trust the certification authority with the known hosts:

```
ca$ scp id_rsa-cert.pub client:.ssh/
ca$ echo "@cert-authority * $(cat ca.pub)" > known_hosts
ca$ scp known_hosts gateway:.ssh/
```

OpenSSL/Certificate management for the client

For other means than SSH, such as TLS tunnels, we can use OpenSSL to interact with the PKI. OpenSSL will generate a private key with an associated CSR (Certificate Signing Request) that will be sent to the PKI module of FreeIPA. FreeIPA will associate the public key to the user and make it available for remote parties.

We first start with the private key and CSR generation on the client using OpenSSL

```
client$ openssl req -out demo.csr -new -newkey rsa:2048 -nodes -keyout private.key
```

This command will output the private key in the file private.key (which shall be stored in the safest place possible) and the CSR in file demo.csr, containing the request as well as the public key. The content of the CSR can then be copied and associated with the user using the web interface as shown in Figure 6 and Figure 7.

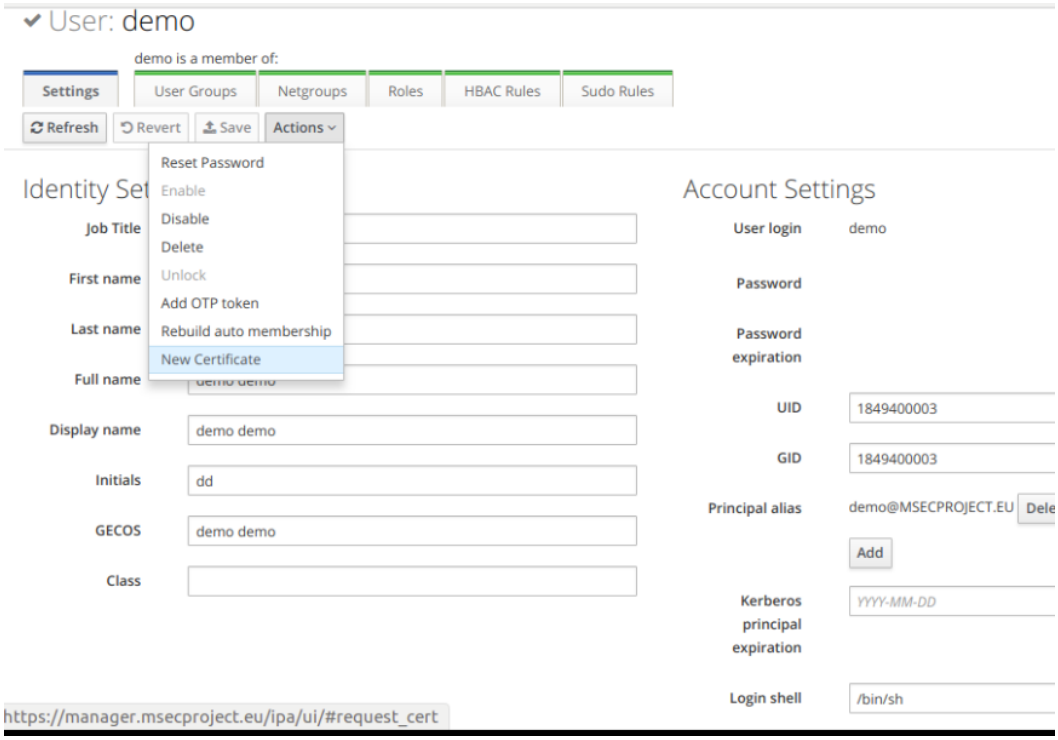

Figure 6. web user page for requesting a new certificate

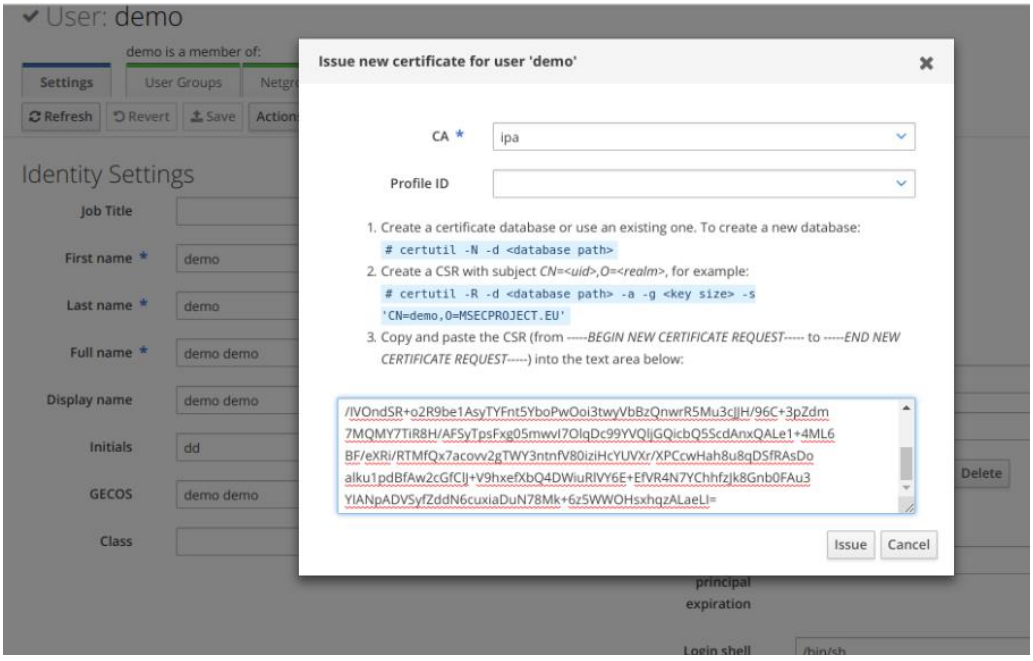

Figure 7. Prompt to insert the CSR while requesting a new certificate

At this stage, the PKI has knowledge about the client but the client is not enrolled within the PKI. To do this enrolment, we need to provide the following command

```
client$ kinit demo
```

Configuring the Endpoint

SSH configuration

As for the client, the endpoint's SSH key must be signed by the CA. Assuming that the endpoint is accessible from the public network with 'endpoint' as a hostname, we can use the following command from the CA to extract the public key and to sign it.

```
ca$ ssh-keyscan -t rsa endpoint | sed "s/^[^ ]* //" > endpoint.pub
ca$ ssh-keygen -s ca -h -I endpoint endpoint.pub
```

The « *endpoint-cert.pub* « file will be generated and is to move back to the endpoint along with the CA's public key

```
ca$ scp endpoint-cert.pub endpoint:/etc/ssh/
ca$ scp ca.pub endpoint:/etc/ssh/
```

Then, we need to configure SSH on the endpoint so it recognizes and trusts the incoming SSH connections signed by the CA.

```
endpoint$ echo "HostCertificate /etc/ssh/endpoint-cert.pub" | sudo tee -a
/etc/ssh/sshd_config
endpoint$ echo "TrustedUserCAKeys /etc/ssh/ca.pub" >> /etc/ssh/sshd_config
endpoint$ service ssh restart
```

OpenSSL configuration

From the endpoint, the configuration depends on the software used to collect data streams from the client. Documentation to use FreeIPA for a web-based application is proposed here: https://www.freeipa.org/page/Web_App_Authentication

Licensing

Most of the Security Manager relies on FreeIPA which is itself a combination of many open source softwares, which may be interchanged. Licenses are listed here: https://www.freeipa.org/page/License

We also have used some security clients on both the endpoint and the client, namely OpenSSH (BSD license) and OpenSSL. (https://www.openssl.org/source/license.html). Thus, other clients may be used in the future with potential commercial licenses.

Summary

This demonstration shows the concept of a Security Manager handling accounting, authorization and authentication for the M-Sec architecture. We illustrate some functionalities of this security manager by authenticating and encrypting data from a device to an endpoint using two mechanisms, either for legacy/unconfigurable end-to-end communications using SSH tunnels or to newer configurable end-to-end configuration with SSL.

Next Steps

The Security Manager presented in this demonstration offers many possibilities that need to be integrated and iterated in both WP4 tasks but also given the use case needs in terms of security level but also in terms of provisioning and life cycle.

3. End-to-end Encryption Middleware for SOXFire

3.1 General Description of the Prototype

As shown in Figure 1 and introduced in the Security Manager description, this prototype is a middleware system that accommodates secure delivery of hyper-connected city data from their producers (i.e., publishers) and consumers (i.e., subscribers), without leakage to the smart city platform. The middleware will be integrated into the KEIO mobile sensing platform.

3.2 Components

In order to implement the end-to-end encryption between publisher and subscriber, we need to first implement a key management module that enables the publisher to securely deliver the encryption key to subscriber without leakage to other possible attackers, and then implement the encryption-decryption module that allows publisher to encrypt its data and the subscriber to decrypt the data. Figure 8 shows such an architecture where the smart city platform possesses no knowledge of the encryption key and it cannot access the encrypted data, avoiding data leakage in case on intrusion by an attacker on this platform. In our prototype, we integrate the modules into the SOXFire, the basis of the KEIO mobile sensing platform. It can be extended to other pub/sub platforms such as MQTT and One M2M.

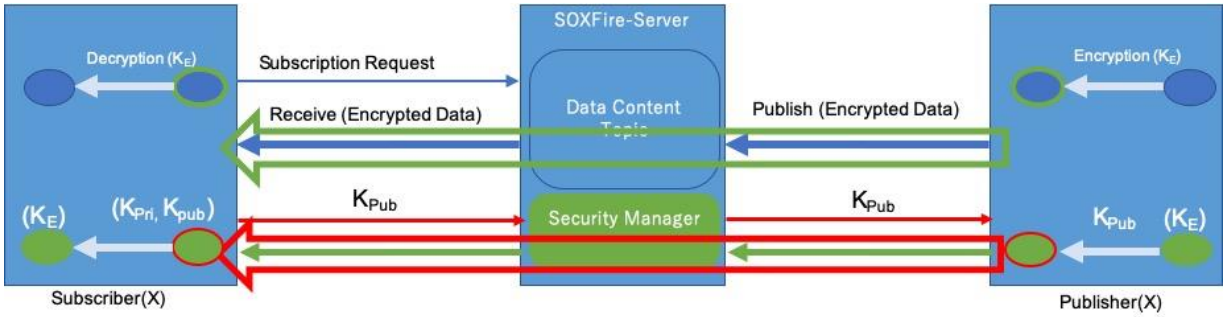

Figure 8. Publisher/Subscriber pattern with potential MITM in yellow and counter-measure in red

Key Management Module

In the key management module, the subscriber first sends its public key to the publisher via an open channel in the SOXFire. When the publisher receives the public key, it will encrypt the encryption key with and then sends it back to the subscriber. Since the SOXFire server does not possess the private key of the subscriber, it is ensured that the encryption key is securely delivered to the subscriber.

End-to-end encryption Module

In the end-to-end encryption module, the publisher encrypts its data with the encryption key and then publishes it to the SOXFire server. The subscriber receives data from the SOXFire server and then decrypts the data with the encryption key obtained using the above key management module.

3.3 Package Information & Installation Instructions

SOXFire is developed based on Openfire. Therefore, SOXFire execution environment depends on Openfire's specifications. In addition, the algorithm has many variations to encrypt data, and you can choose your favorite algorithm. In this section, we describe only SOXFire.

Required Tools and dependencies

- Java
- MySQL(5.7.XX) 'XX' means any version.
- Ant
- SOXFire source code

Download and Run Demonstrator

1. Java
   Download and install Java environment. Also set $JAVA_HOME in your shell.
2. MySQL
   Download and install MySQL. Then, create user and sxofire database like this:

```
mysql> CREATE USER 'soxfire'@'localhost' IDENTIFIED BY 'YOUR_PASSWORD';
```

```
mysql> GRANT ALL RPVILENGES ON *.* To ‘soxfire’@’localhost’ WITH GRANT OPTION;
msyql> CREATE DATABASE soxfiredb DEFAULT CHARACTER SET utf8mb4;
```

3. Ant
   Install Ant.
4. SOXFire
   Download SOXFire source code.
   http://sox.ht.sfc.keio.ac.jp/soxfire/soxfire_src-1.0.3.zip
5. Compile and start
   Uncompress a downloaded zip file. Then move to build directory and build.

```
% cd soxfire_scr/build
% ant
```

You can find compiled files in soxfire_src/work and soxfire_src/target directory. To run the SOXFire server, got to target/openfire/bin directory and run openfire.sh.

```
% cd soxfire_src/target/openfire/bin
% ./openfire.sh
```

6. Configuration
   Open http://localhost:9090 in your browser. You can see the following setup interface in Figure 9

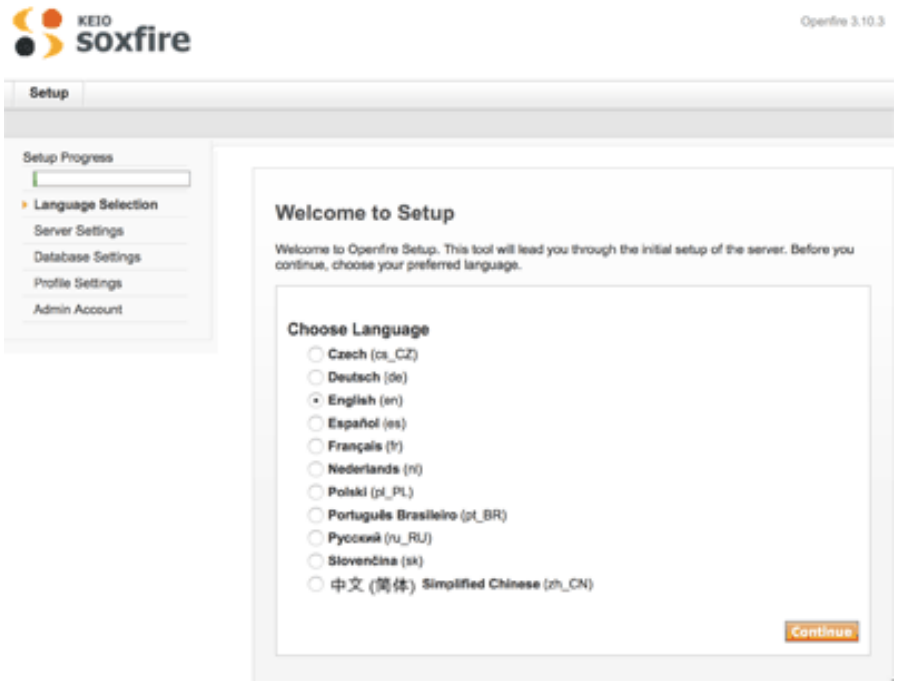

Figure 9. SOXFire setup screen

You can basically follow Openfire setup example as shown in the above link. Especially for SOXFire database, select MySQL.

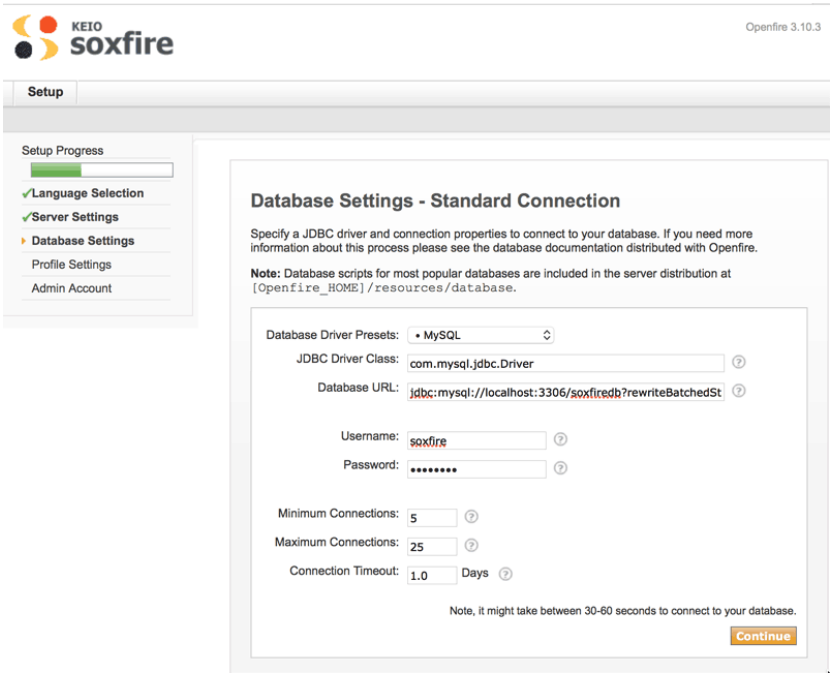

Figure 10. SOXFire database configuration

User Manual

The user manual is available on:

http://sox.ht.sfc.keio.ac.jp/soxfire/index.html

This website is shown the same content of 'Download and Run Demonstrator'.

Licensing

SOXFire license depends on Openfire license.

- Apache License 2.0

Reference use case

The reference use case for this solution is use case 3.

Summary

In this chapter we have described the integration of security within the SOXFire middleware that is used within M-Sec. The demonstration aims to have a fully secured channel between a device and the end user's application. The security threat we address is the fact that in some implementation of the publisher/subscriber pattern intermediate servers such as brokers may decode the data and therefore exposing it to potential malicious attackers.

Next Steps

In the next step the interface between the SOXFire middleware and the security manager will be improved beyond simple key/session management for bootstrapping and other functionalities that can be brought by the security manager and requested for use cases.

4. sensiNact - Secured IoT Middleware

4.1 General Description

The sensiNact Gateway implements the basic blocks for connectivity, service abstraction, device management, virtualization and remote access. The sensiNact Gateway allows the interconnection of different networks to achieve secured access and communication with embedded devices. The sensiNact platform defines a generic service model, described in Figure 11, and a set of generic access methods:

- the **Service provider**, **Service** and **Resource** triptychs the spine of the model:
  - the **Service provider** is attached to one location
  - the **Service** is attached to one business concept
  - and the **Resource** is attached to one business data
- A **Resource**, on which apply **Access Methods,** is a collection of **Attributes**, characterized by **Metadata**
- **Access Methods** allow reading (client/server or publish/subscribe model) to **write**, and to **actuate** when applicable

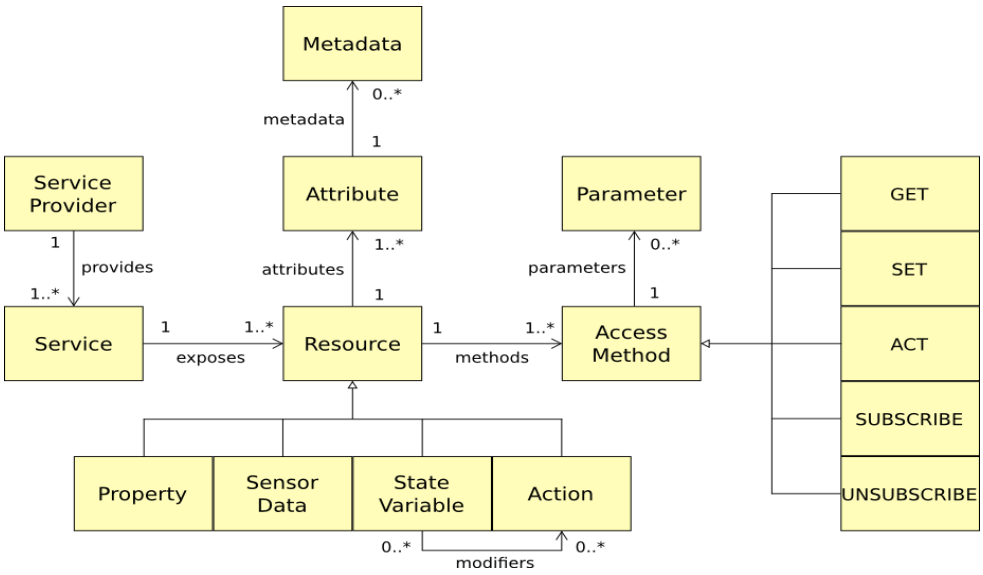

Figure 11. sensiNact service model

An example of the service model implementation is given in Figure 12.

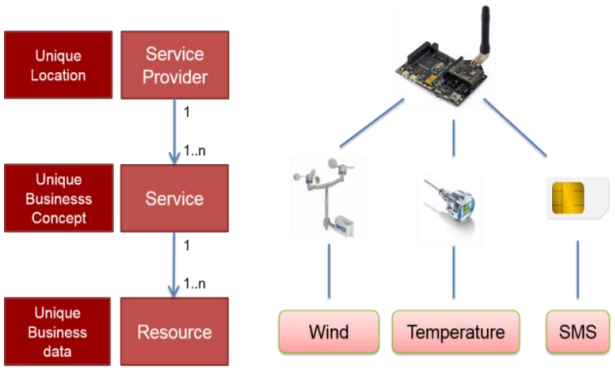

Figure 12. sensiNact service model mapping example

4.2 Components

Connectivity

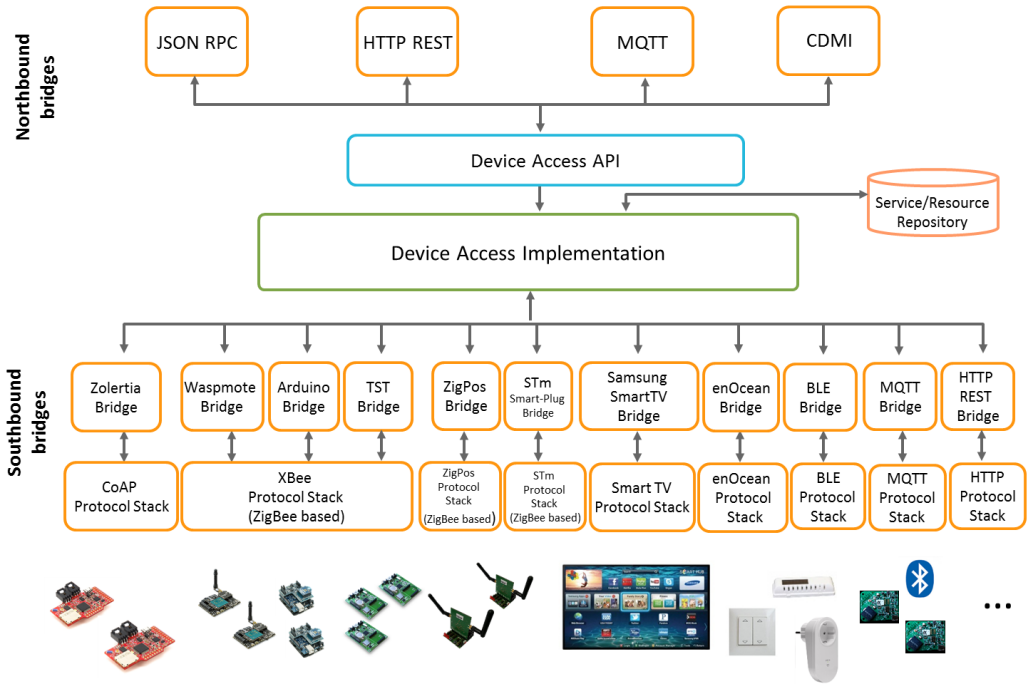

Figure 13. sensiNact gateway northbound and southbound connectivity

Figure 13 shows the southbound side the sensiNact gateway which allows to cope both with “physical device” protocols and “virtual device” ones, allowing a uniform and transparent access to an BLE network, or an HTTP Restful web service for example (pell-mell a non-exhaustive list of supported protocols: EnOcean, Bluetooth Low Energy, MQTT, CoAP, NGSI, Openhab, SoxFire, etc). On the

northbound side the sensiNact gateway provides both client/server and publish/subscribe access protocols (MQTT, JSON-RPC, HTTP Restful, NGSI, CDMI, SoxFire, etc…)

Security Components

In sensiNact three sites allow providing a secured encapsulation of the data relayed by the platform as illustrated in Figure 14:

- The encryption of data coming from connected counterpart is easily implementable over each southbound bridge;
- The modules signature allows to define which module will be allowed to provide a feature, or a remote system/device connection;
- Finally an OAuth2 / OpenID intermediation service provides northbound access security

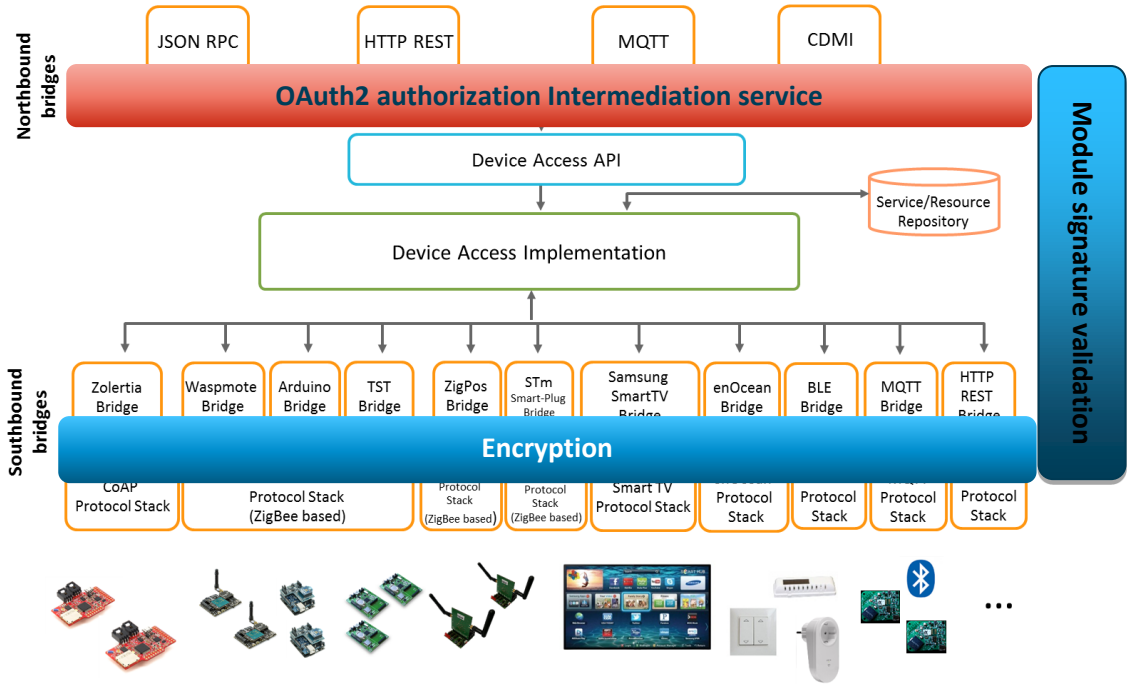

Figure 14. Secured connectivity

Encryption

The modular and generic southbound bridge template allows including a securitization process between the sensiNact and the connected counterpart, like keys sharing allowing encrypting exchange data without any extra implementation or update of the platform.

Module signature validation

The signature of the modules that compose the platform at build time permits validating that an installed module is allowed to connect to the others and to provide a new feature or a secure access to a connected counterpart. Figure 15 shows the deployment of a signed bridge within the container.

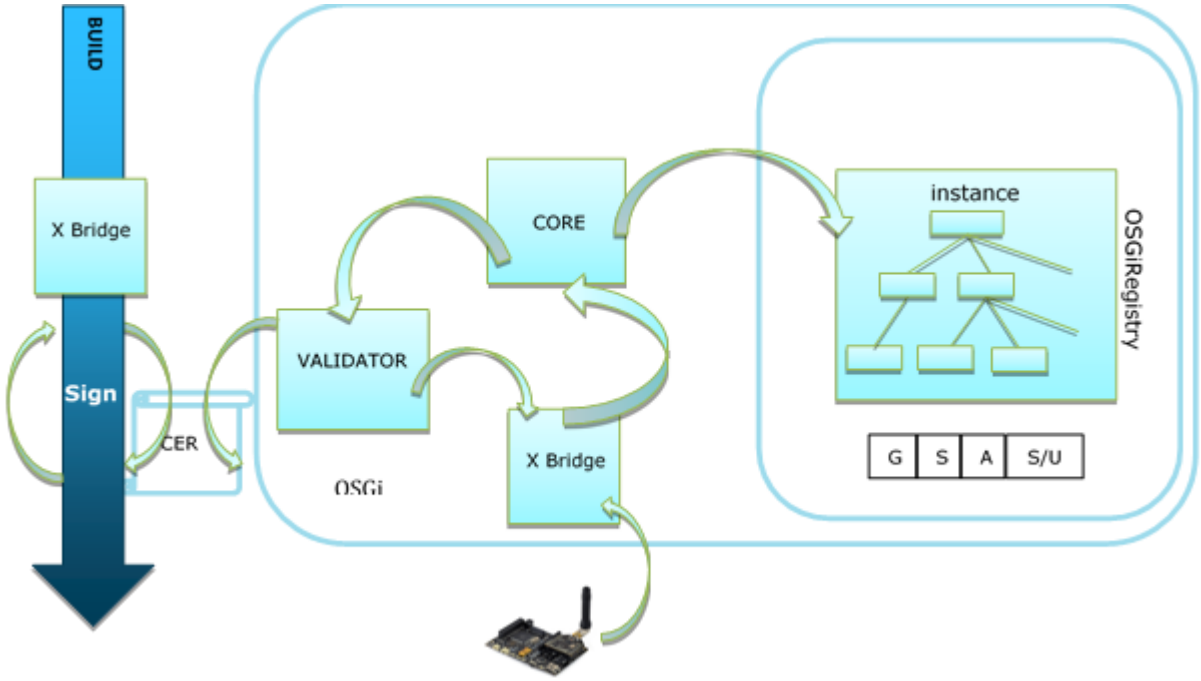

Figure 15. Sign the X bridge

OAuth2 authorization intermediation service

An oAuth2/OpenID filter, implementing for now Authorization code and Resource owner password credentials flow, offers a standard northbound secure access solution. Figure 16 and Figure 17 shows the authorization/verification flow.

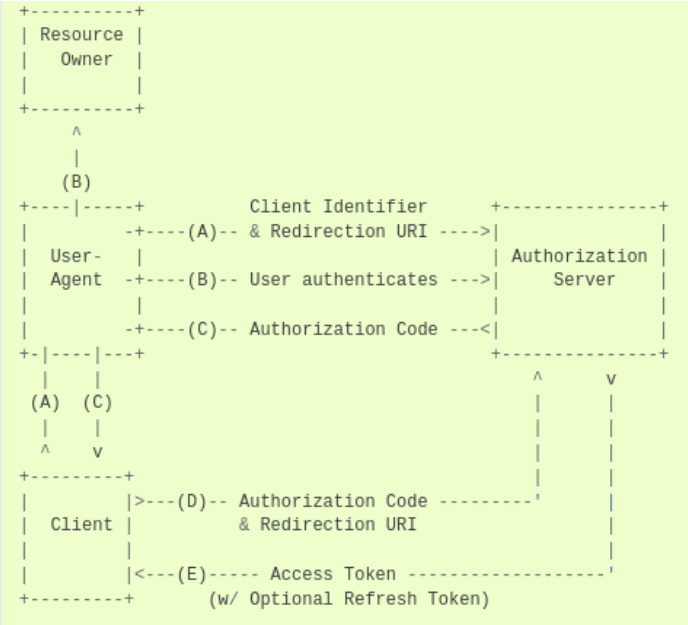

Figure 16. Authorization code flow

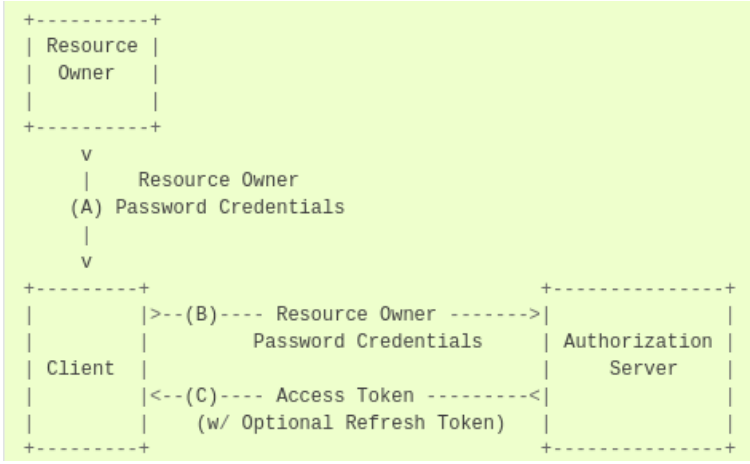

Figure 17. Resource owner password credential flow

This oAuth2/OpenID filter feature is the one presented in the demonstrator.

4.3 Required Tools and dependencies

In order to compile and deploy sensiNact, the following dependencies are required:

- Git
- Maven 3.5.3
- Java 1.8
- sensiNact source code
- Keycloak 7.0.1

Download and Run Demonstrator

JavaJDK 1.8

cf. https://openjdk.java.net/projects/jdk8/

When your environment has been installed define the JAVA_HOME targeting the JDK installation directory.

Git

cf. https://git-scm.com/

Maven

cf. http://maven.apache.org/

Keycloak

cf. https://www.keycloak.org/index.html

When the Keycloak server is installed you can ask for a standalone execution to configure it (initial credentials are admin:admin). You need now to configure client, roles, groups and users; The JSON data structure below presents the properties that need to be configured for this sample demonstration:

```
{
	"realm": "test",
	"enabled": true,
	"sslRequired": "none",
	"registrationAllowed": true,
	"requiredCredentials": [ "password" ],
	"users":
	[
		{
			"username": "testRealmAdmin",
			"enabled": true,
			"credentials":
			[
				{
					"type": "password",
					"value": "testRealmAdminPassword"
				}
			],
			"clientRoles":
			{
				"realm-management": [ "realm-admin" ],
				"account": [ "manage-account"]
			}
		},
		{
			"username": "anonymousTester",
			"enabled": true,
			"credentials":
			[
				{
					"type": "password",
					"value": "anonymousTester"
				}
			],
	"groups": [
	    "/anonymousGroup"
	]
		},
		{
			"username": "adminTester",
			"enabled": true,
			"credentials":
			[
				{
					"type": "password",
					"value": "adminTester"
				}
			],
	"groups": [
	    "/adminGroup"
	]
		}
	 ],
	 "roles" :
```

```
            {
                  "realm" :
                  [
                           {
                                    "name": "anonymous",
                                    "description": "Have anonymous privileges"
                           },
                           {
                                    "name": "admin",
                                    "description": "Have Administrator privileges"
                           }
                  ]
           },
"groups" : [
    {
        "name": "anonymousGroup",
        "attributes": {
            "topAttribute": ["true"]
        },
        "clientRoles": {
            "testClient": [ "anonymous" ]
        }
    },
    {
        "name": "adminGroup",
        "attributes": {
            "topAttribute": ["true"]
        },
        "clientRoles": {
            "testClient": [ "admin", "anonymous" ]
        }
    }
],
       "clients":
       [
              {
                  "clientId": "testClient",
                  "secret": "testClient",
                  "enabled": true,
                  "fullScopeAllowed": true,
                  "baseUrl": "http://localhost:8899",
                  "bearerOnly": false,
                  "consentRequired": false,
                  "standardFlowEnabled": true,
                  "directAccessGrantsEnabled": true,
                  "publicClient": false,
                  "frontchannelLogout": false,
                  "protocol": "openid-connect",
                  "nodeReRegistrationTimeout": -1,
                  "redirectUris":
                        [
                                 "http://localhost:8899/auth",
                                 "http://localhost:8899/sensinact",
                                 "http://localhost:8899/sensinact/*"
                        ],
                        "protocolMappers": [
                     {
                       "name": "UserClientRoleMapper",
                       "protocol": "openid-connect",
                       "protocolMapper": "oidc-usermodel-client-role-mapper",
                       "consentRequired": false,
                       "config": {
                            "multivalued": "true",
```

```
                                "userinfo.token.claim": "true",
                                "id.token.claim": "true",
                                "access.token.claim": "true",
                                "claim.name": "roles",
                                "jsonType.label": "String",
                                "usermodel.clientRoleMapping.clientId": "testClient"
                            }
                        }
                    ]
                }
            ]
}
```

You have created two clients, anonymousTester and adminTester, respectively belonging to anonymousGroup and adminGroup for which (respectively again) the anonymous and admin roles apply.

sensiNact

Clone the sensiNact's git source code repository:

```
git://git.eclipse.org/gitroot/sensinact/org.eclipse.sensinact.gateway.git
```

In the root directory of the newly created local repository build with maven, skipping tests to speed up the process:

```
mvn clean install –DskipTests
```

The `distribution/generator/target/sensinact` now contains a distribution that we will use for the demonstration**. Paths mentioned below refer to this location**

Configuration

Clear the actual default configuration files directory: `rm cfgs/*`

Make sure that the script is executable: `chmod +x ./sensinact`

Create the configuration file defining the oauth2 mechanism `./cfgs/sensinact-security-oauth2.config` and define the content as below:

```
discoveryURL=http://localhost:8080/auth/realms/test/.well-known/openid-configuration
certsURL=http://localhost:8080/auth/realms/test/protocol/openid-connect/certs
client_secret=testClient
client_id=testClient
slider=admin:GET:/sensinact/slider/*
```

Configure the HTTP service provided by the felix framework in `conf/config.properties`**:**

```
org.osgi.service.http.port=8899
org.apache.felix.http.debug=true
org.apache.felix.http.jettyEnabled=true
org.apache.felix.http.whiteboardEnabled=true
```

Configure the OAuth2 configuration file in `conf/config.properties`:

```
org.eclipse.sensinact.security.oauth2.config=cfgs/sensinact-security-oauth2.config
```

Copy the necessary modules:

```
cp load/simulation/slider-2.0-SNAPSHOT.jar ./bundle/
cp load/rest/*.jar ./bundle/
cp ../../../../platform/sensinact-security/sensinact-security-oauth2/target/*.jar ./bundle/
```

You can now run sensiNact:

```
./sensinact
```

Now If you try to access to the definition of the simulated slider device using your browser (http://locahost:8899/sensinact/slider), you will be asked to authenticate, and if you do as `adminTester` (credentials `adminTester:adminTester`) that is the only one allowed to access to the slider, you will see the description of the targeted device. Figure 18 and Figure 19 shows the interfaces corresponding to those steps.

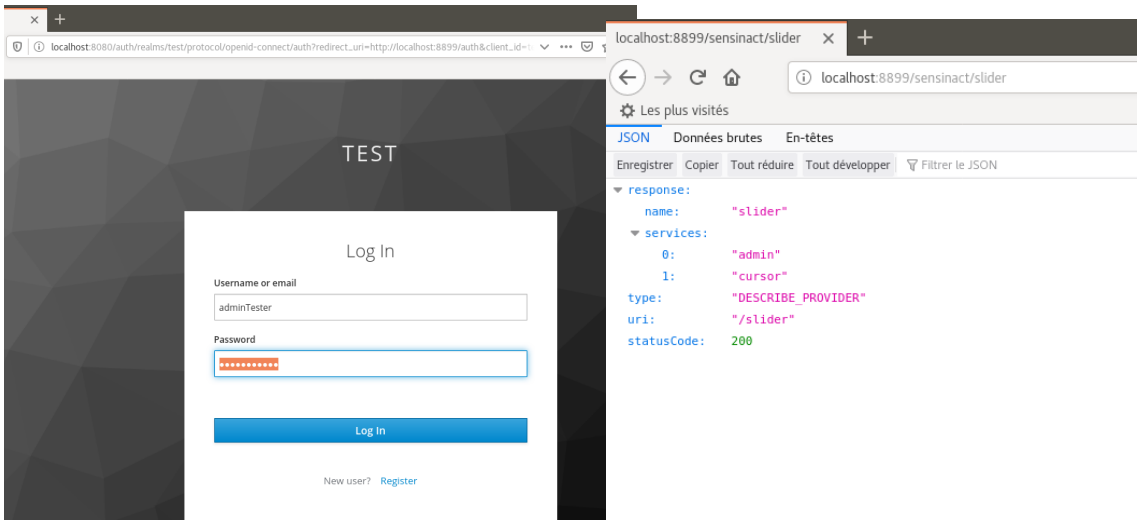

Figure 18. Login as adminTester

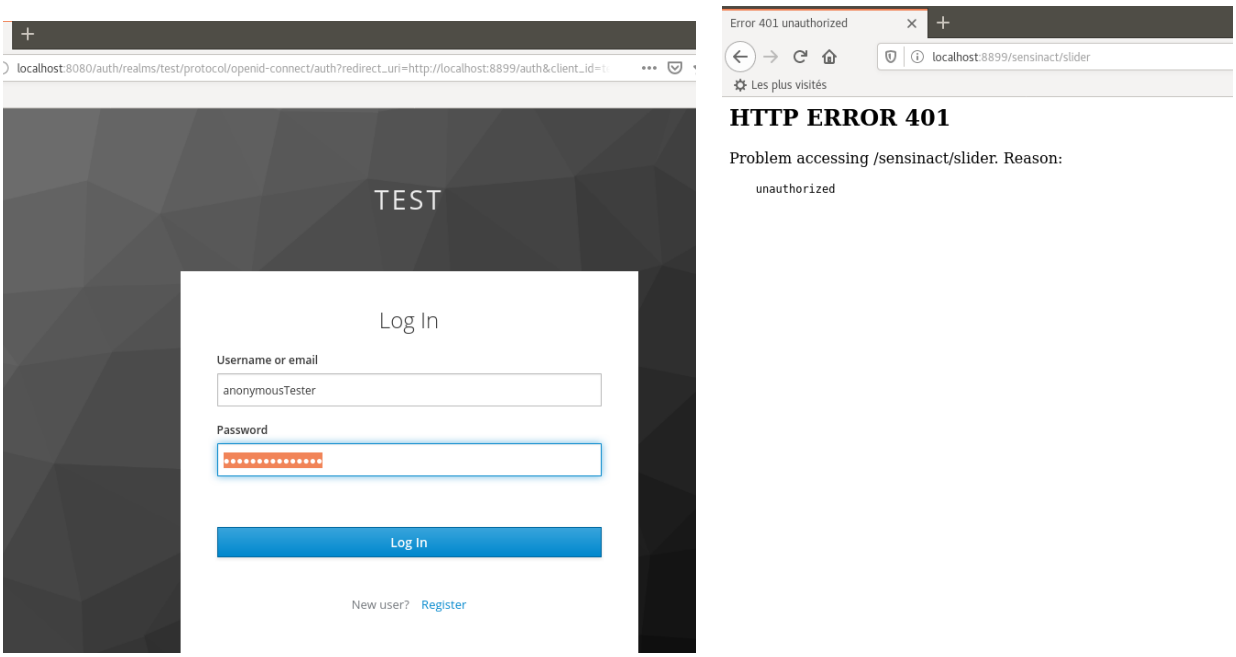

Figure 19. Login as anonymousTester

Licensing

sensiNact is an Eclipse project and is so under Eclipse licence.

- Eclipse Public License - v 1.0 cf.https://www.eclipse.org/legal/epl-v10.html

Summary

This demonstration shows the security functionalities that have been integrated within sensiNact. Two main functions are described, the first one is the inner security with a sign/verify mechanism in order to validate the code which is loaded into the middleware. It allows better stack management with trusted third parties. The second function is the authentication of the northbound APIs for the end-users with a component of the security manager, providing an easier enrolment of the citizen within the M-Sec architecture using their existing identities.

Next Steps

During the second period this demonstrator will integrate more functions provided by the security manager, with prioritization given the use case needs and the risk analysis associated with both the technologies and the use case requirements.

5. Conclusion

In this document we have presented three demonstrations bound together to provide end-to-end security. At first we have presented a Security Manager which provides a core directory to store any “M-Sec” entity security information such as certificates, passwords and so on. This Security Manager is designed to provide security functions to the overall M-Sec component developed in other WP4 tasks such as IoT Devices, communication channels, blockchain system and applications.

First integration of this Security Manager has started in the M-Sec middlewares such as SOXFire and sensiNact, enabling safer collection and distribution of data at a smart city level. Having a common security manager on both this middleware also provides interoperability for cross-border use-cases with the central management of credentials. The integration will continue in the next period and will be iterated with the Use-Case needs.