# D3.14 – i4Q IIoT Security Handler v2

WP3 – BUILD: Manufacturing Data Quality

i4Q has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 958205.

## Document Information

| | | | |
|---|---|---|---|
| GRANT AGREEMENT NUMBER | 958205 | ACRONYM | i4Q |
| FULL TITLE | Industrial Data Services for Quality Control in Smart Manufacturing | | |
| START DATE | 01-01-2021 | DURATION | 36 months |
| PROJECT URL | https://www.i4q-project.eu/ | | |
| DELIVERABLE | D3.14– i4Q IIoT Security Handler v2 | | |
| WORK PACKAGE | WP3 – BUILD: Manufacturing Data Quality | | |
| DATE OF DELIVERY | CONTRACTUAL 31-Dec-2022 | ACTUAL | 30-Dec-2022 |
| NATURE | Report | DISSEMINATION LEVEL | Public |
| LEAD BENEFICIARY | IKERLAN | | |
| RESPONSIBLE AUTHOR | Aitor Uribarren and Maialen Eceiza (IKERLAN) | | |
| CONTRIBUTIONS FROM | 15-FBA, 24-FIDIA | | |
| TARGET AUDIENCE | 1) i4Q Project partners; 2) industrial community; 3) other H2020 funded projects; 4) scientific community | | |
| DELIVERABLE CONTEXT/ DEPENDENCIES | This document describes a service that distributes trust across the architecture using a hardware secure module as trust anchor point.<br>This document is the updated v2 of D3.6 i4Q IIoT Security Handler. | | |
| EXTERNAL ANNEXES/ SUPPORTING DOCUMENTS | None | | |
| READING NOTES | None | | |
| ABSTRACT | To fight against a growing range of cyber-related risks, industrial enterprises need rapid and demonstrable improvements in their Operational Technology (OT) and Industrial Control Systems (ICS) cyber security. Because of their potential impact on system performance, utilities and other users of these systems may be cautious to embrace popular security technologies.<br>This document presents general description and technical application i4Q IIoT Security Handler (i4Q$^{SH}$) which is a proposal of an implementation of a PKI to provide trust in the i4Q ICS ecosystem. | | |

## Document History

| VERSION | ISSUE DATE | STAGE | DESCRIPTION | CONTRIBUTOR |
|---|---|---|---|---|
| 0.1 | 07-Nov-2022 | ToC | ToC Draft and first input | IKERLAN |
| 0.2 | 25-Nov-2022 | 1st Draft | Available for Internal Review | IKERLAN |
| 0.3 | 30-Nov-2022 | Internal Review | Internal Review | FBA, FIDIA |
| 0.4 | 02-Dec-2022 | 2nd Draft | Addressing comments from internal reviewers | IKERLAN, FBA, FIDIA |
| 0.5 | 09-Dec-2022 | Final Draft | Submission for final draft for quality check | IKERLAN |
| 0.6 | 19-Dec-2022 | Final Draft | Images updated | IKERLAN |
| 1.0 | 30-Dec-2022 | Final Document | Final quality check and issue of final document | CERTH |

## Disclaimer

## Copyright message

# TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

## ABBREVIATIONS/ACRONYMS

| | |
|---|---|
| API | Application Programming Interfaces |
| CA | Certification Authority |
| CCID | Chip Card Interface Device |
| CIA | Confidentiality, Integrity and Availability |
| COTS | Commercial Off-The-Shelf |
| CRL | Certificate Revocation List |
| CSR | Certificate Signing Request |
| EHCI | Enhanced Host Controller Interface |
| EST | Enrollment over Secure Transport |
| HSM | Hardware Security Module |
| HTTP | HyperText Transfer Protocol |
| ICS | Industrial Control Systems |
| ID | Identification |
| IIoT | Industrial Internet of Things |
| IP | Internet Protocol |
| IT | Information Technology |
| JSON | JavaScript Object Notation |
| OCSP | Online Certificate Status Protocol |
| OHCI | Open Host Controller Interface |
| OPC | Open Platform Communications |
| OPC-UA | Open Platform Communications Unified Architecture |
| OpenSSL | Open Secure Sockets Layer |
| OT | Operational Technology |
| PHP | Hypertext Preprocessor |
| PKI | Public Key Infrastructure |
| RA | Registration Authority |
| REST | Representational State Transfer |
| RSA | Rivest–Shamir–Adleman |
| SCADA | Supervisory Control And Data Acquisition |
| SDK | Software Development Kit |

| | |
|---|---|
| SH | Security Handler |
| SOAP | Simple Object Access protocol |
| TCP | Fieldbus communication |
| TLS | Transport Layer Security |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| USB | Universal Serial Bus |
| VA | Validation Authority |
| WS | Web Service |
| WSDL | Web Services Description Language |

# Executive summary

This document i4Q **Security handler v2** (i4Q$^{SH}$) is an update of v1 of i4Q **Security handler** (i4Q$^{SH}$), for this reason it contains information of the 1st version together with the updates developed in this 2nd version.

This deliverable contains the specification and design of the i4Q **Security handler v2** (i4Q$^{SH}$), which is used to provide trust in an industrial control system (ICS) using a Public Key Infrastructure developed by IKERLAN in i4Q [1].

An ICS is a group of control systems that collaborate to achieve a specific industrial goal. The integration of operational and information technology (IT) in ICS has increased operational and financial efficiency, but it has also introduced new risks, such as system outages and espionage. Today's control systems are vulnerable to cyber-attacks in a variety of ways, and control system engineers must be well-versed in topics such as ICS security and SCADA security. This document proposes the use of X.509 certificates supply trust in the i4Q ICS ecosystem [1].

## Document structure

**Section 1:** Contains a brief introduction of the i4Q **IIoT Security Handler v2**, providing an overview and the list of main features provided by a PKI. It is addressed to final users of the i4Q Solution.

**Section 2:** Contains a general description and the technical specifications of the i4Q **IIoT Security Handler v2**, providing an overview and its architecture diagram. It is addressed to software developers.

**Section 3:** Details the implementation status of the i4Q **IIoT Security Handler v2**, explaining the current development status, next integration steps and summarizing the implementation history. It is addressed to software developers.

**Section 4:** Provides the conclusions.

**APPENDIX I:** Provides the PDF version of the i4Q **IIoT Security Handler v2** web documentation, which can be accessed online at: http://i4q.upv.es/6_i4Q_SH/index.html

# 1. General Description

## 1.1 Overview

According to several surveys, the amount of cyberattacks on operational technology (OT) is on the rise [2]. Almost 22,000 vulnerabilities were published in 2021 [3]. Organizations use OT to manage physical industrial equipment, assets, processes, and events using a combination of technology, software, and hardware. Industrial Control Systems (ICS) are widely used in these OT environments to monitor and control industrial processes such as those in the manufacturing, transportation, and pharmaceutical sectors, as well as critical infrastructures such as power plants, water treatment plants, and oil and gas refineries [4].

ICS has traditionally been designed to operate on customized hardware and/or software that is physically isolated from the outside world; however, this is no longer the case. ICSs have made use of a wide range of information technology (IT) solutions, including commercial off-the-shelf (COTS) components, remotely enabled connections, standardized operating systems, and cloud-based solutions. This advancement, in conjunction with the use of insecure industrial protocols such as DNP3, OPC, and MODBUS, increases the risk of security flaws and incidents [5][6]. As a result, ICSs are susceptible to the same vulnerabilities as any other system, such as buffer overflows, hardcoded credentials, authentication bypass, cross-site scripting, missing authentication, and hardware chip vulnerabilities [7].

Cryptography is one key strategy businesses use to protect the systems that house their most valuable asset or information, whether in transit or at rest. Client, worker, intellectual property, ICS business policies, and other classified information are examples of data. As a result, cryptography has become a critical infrastructure, as cryptographic solutions rely more and more on the security of sensitive data. It can help protect sensitive and secret information while increasing client-server communication security. In other words, even if an unauthorized individual or entity gains access to your data, they will be unable to read it.

Public Key Infrastructure (PKI) is a trust chain that enables a person, service, machine, or application to be authorized, a secure connection to be established, or the provenance of software or documents to be validated. This is done with certificates, which a PKI creates, manages, and distributes while also having the ability to rescind. A certificate's public key must be kept safe and secret, and the private key must be kept safe and hidden. It must be kept in a hardware security module (HSM) [1].

## 1.2 Features

A PKI certificate is a digitally signed document that functions similarly to a real identity card or passport in everyday life. Private and public keys are used in public-key cryptography, and the certificate is used to prove possession of the public key by storing it alongside information about the owner and some administrative data. The issuing CA signs the certificate, and the signature is included in the certificate. The most widely used digital certificate formats are defined in X.509 standard [8].

A digital certificate, as well as the infrastructure that issues the digital certificate, offer the necessary information and structure:

- Reducing the possibility to prove people's identities using Internet
- Protect messages in transit reducing the possibilities of being read by anyone different other that the receiver
- Protect electronic messages by lowering the chances of them being tampered with or altered in the way without the recipient's awareness.
- Allow for non-repudiation of transactions, so that no one can deny taking part in a legitimate electronic transaction.

In other words, and as specific use cases of PKI certificates can be:

- TLS certificates usage in Secure network communication
- To sign documents and code
- To sign and encrypt emails
- IoT certificates
- Personal authentication

The three most important concepts in information security are confidentiality, integrity, and availability. The CIA triad security model's relevance expresses itself, with each letter expressing a fundamental premise in cybersecurity.

### 1.2.1 Authentication

One of the most challenging aspects of doing business on the Internet today is determining whom you are dealing with.

The Public Key Infrastructure (PKI) enables trading partners to be identified online by trusted authorities in charge of issuing digital certificates and supplying procedures for identifying individuals who hold those certificates on behalf of a company [1].

### 1.2.2 Privacy

Do you know if anyone other than you have ever accessed communications or transactions you've sent over the Internet? Is it possible for only the intended recipient to read your message?

Digital certificates provide a method for protecting information. Messages can be encrypted to reduce the possibility of being captured or read by someone other than the intended recipient while in transit. This generates a digital copy of a registered letter sent via mail.

PKI can be used to make messages more private.

### 1.2.3 Integrity

When conducting business over the Internet, a company must be confident that no transaction or information provided will be altered or interfered with during transmission [1].

The PKI, as a fundamental component, ensures transaction security. The recipient of a communication can use PKI to verify that the message is still identical to what it was when it was transmitted [1].

### 1.2.4 Non-repudiation

PKI creates a mechanism for signing electronic transactions in the same way that a signature is placed on a document.

A PKI allows to produce one-of-a-kind signatures. When this is combined with suitable policies and processes, the sender is unable to refute or repudiate a message sent in compliance with these protocols.

A PKI can be created to ensure that no lawful transaction can be reversed.

# 2. Technical Specifications

## 2.1 Overview

To identify certificate issuance needs, next there will be defined and explained the certificate release use case. The idea is to show how external certificate requesters will interact with the PKI.

## 2.2 Certificate Life Cycle Management

Regarding the certificate issuance there is a main software component called 'Certificate Handler' which is responsible for getting the request via the defined API and redirect to the internal components depending on the request.

### 2.2.1 Certificate Handler

This is the main component which is in charge of receiving the requests and distribute them to the most suitable subcomponent.

It will provide different API interfaces to provide distinct ways to connect to the certificate requesters.

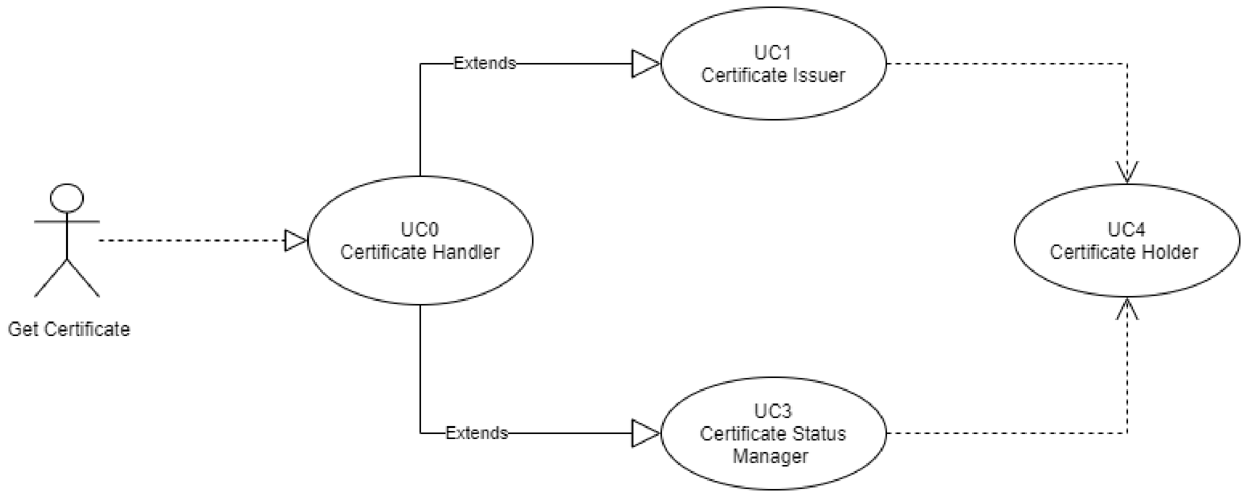


**Figure 1.** Generic certificate issuance process

### 2.2.2 Certificate Issuer

This is the component that oversees the creation of the certificates. It is closely related with the PKI to obtain the required certificates with the requested characteristic.

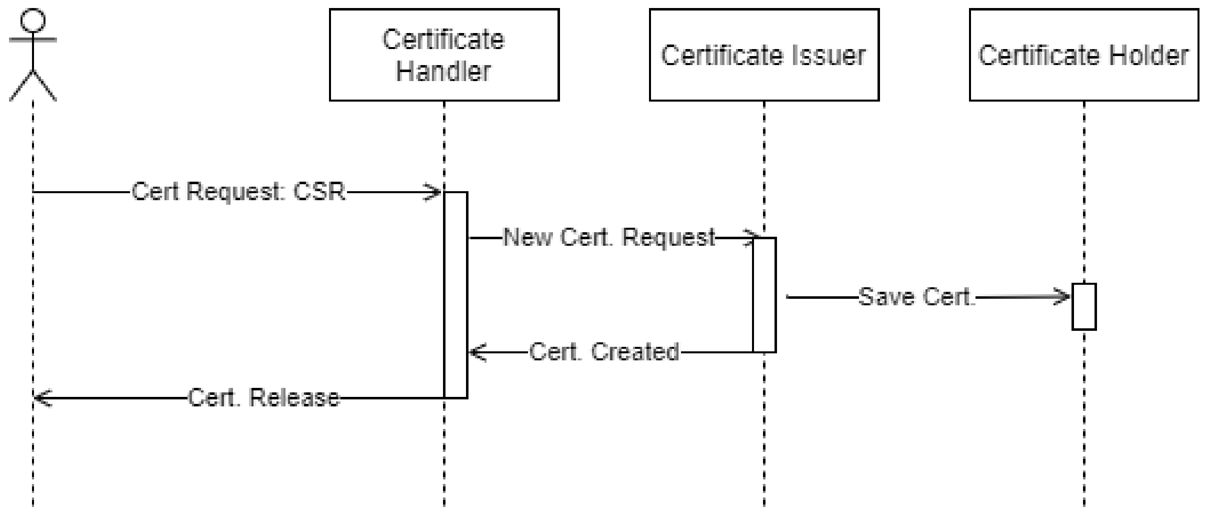


**Figure 2.** Process to generate a new certificate

As shown in the previous **Figure 2**, the certificate requester provides a certificate signing request (CSR) to the 'certificate handler'. If everything is ok, 'certificate issuer' is called in order to create the certificate. Once the certificate is saved in the 'certificate holder', the requested certificate is returned. And although it is not explicitly shown, if any inconsistence or error is founded in 'certificate handler' or in 'certificate issuer' or even in saving the certificate in 'certificate holder' the certificate request will be rejected, and error message will be returned.

### 2.2.3 Certificate Status Manager

This is the component which is in charge of the status of the certificates. It manages certificates status checks, renewals, and revocations.

#### *2.2.3.1 Status Checking*

The way the certificate status is checked by any of the clients is using an Online Certificate Status Protocol (OSCP) request.

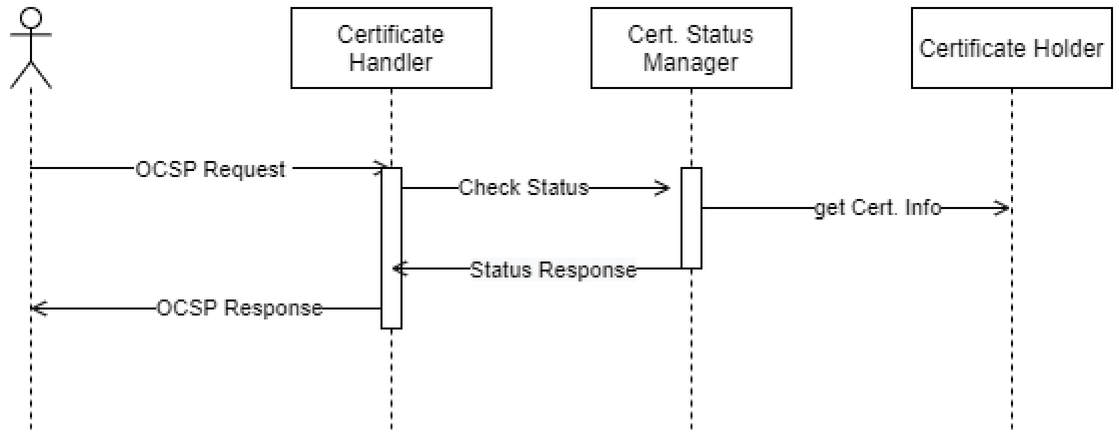


**Figure 3.** Process to check which is the status of the certificate

As shown in the previous **Figure 3**, any entity that wants to check the state of a certificate, sends a OCSP request. The 'certificate status manager' is the component to answer if the certificate is "good", "revoked", or "unknown".

### *2.2.3.2 Certificates Renewal*

To renew a previously released certificate, the clients will use the Enrollment over Secure Transport (EST) protocol. The request will be firstly received by the Certificate Handler.

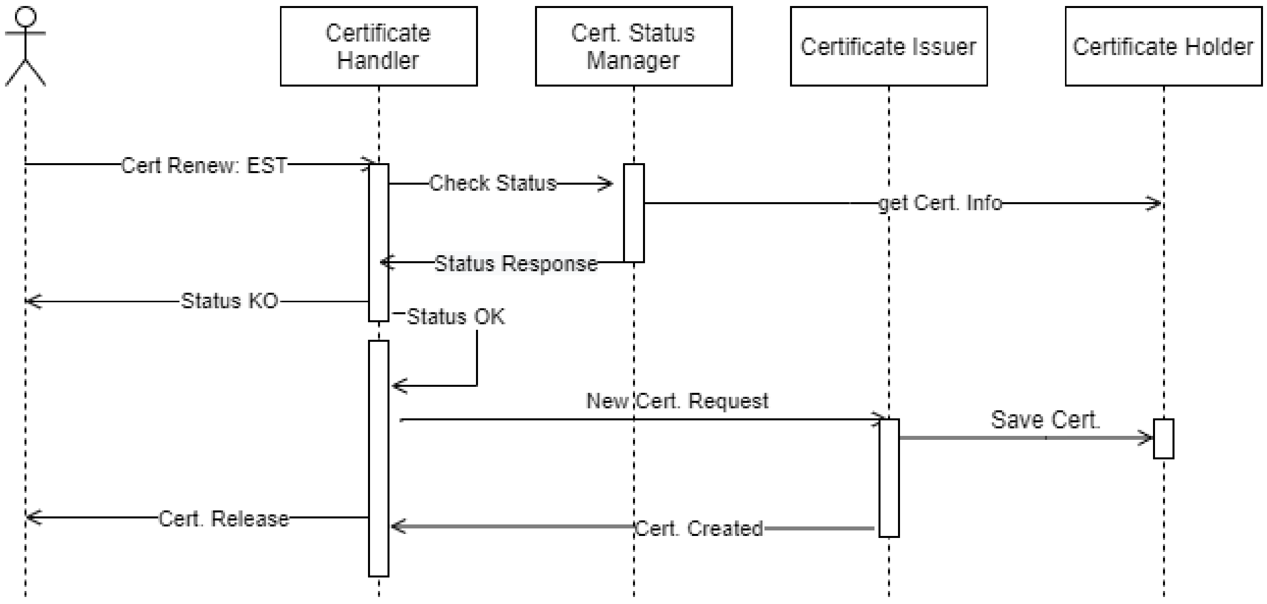


**Figure 4.** The certificate renewal process

As shown in the previous **Figure 4**, first the 'certificate status manager' checks that the request is ok, for example checking that it is not revoked, and in case that everything is ok, a new certificate is released by the 'certificate issuer'. Otherwise, the request is rejected by the not ok or 'status ko' message.

### *2.2.3.3 Certificate Revocation*

To revoke previously released certificate, the clients will use the corresponding API function providing the associated privileged parameters.

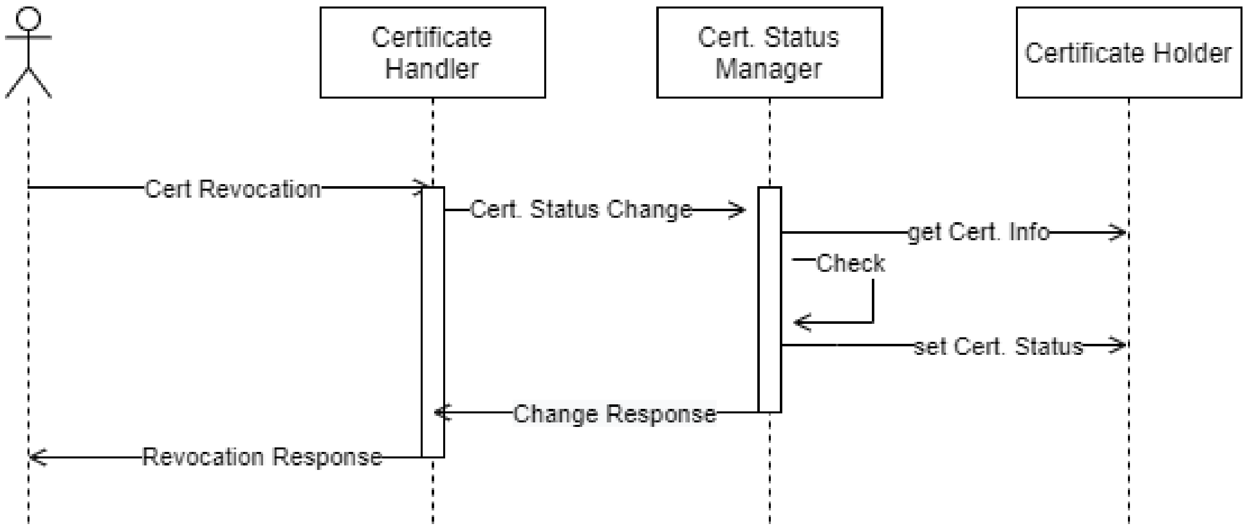


**Figure 5.** Process to revoke the certificate

As shown in the previous **Figure 5**, the revocation request will be analysed by the 'certification status manager'. If the provided credentials and the associated certificate status is correct, the certificate will be revoked setting the corresponding revocation reason.

### 2.2.4 Certificate Holder

This is the component which is in charge of the storage of the certificates. It will be closely related with the PKI storage system. Typical database functionality such as create, modify or save, associated to certificates will be held by this component.

## 2.3 OPC-UA

OPC Unified Architecture (OPC-UA) is a machine-to-machine communication protocol for industrial automation developed by the OPC Foundation [9]. It is a protocol for transferring data in the form of objects rather than discrete data points. This increases the accessibility of the plant floor data by allowing you to reuse information stored in a shared object. OPC-UA also has a service-oriented model, which improves security and platform interoperability.

The OPC UA transport protocol, among other things, offers a reliable and dependable communication infrastructure by handling lost messages, heartbeats, and failovers. OPC UA is a high-performance data exchange protocol that employs binary-encoded data. Commercial SDKs for C, C++, Java, and.NET are available. On the other hand, open-source stacks are available at least for C, C++, Java, Javascript(node), and Python.

Security is built into OPC UA and has been a design goal since its inception. OPC UA provides various technology mappings based on TCP/IP or SOAP-based web services. A secure channel is used on top of the transport layer by utilizing encryption and digital signatures to protect messages from unauthorized alterations and eavesdropping. Furthermore, this layer authenticates and authorizes specific instances of OPC UA applications using digital certificate-based authentication procedures. This enables administrators to implement fine-grained access control in critical infrastructures such as manufacturing facilities and power plants. A session is a connection between a client application and a server used to exchange payload representing plant information (e.g., valve status, temperature, level indicator status), settings, and commands. The previously mentioned secure channel thus protects session messages. Users who intend to establish client-side sessions must be authenticated and authorized by OPC UA servers. The specification supports three mechanisms: username/password combinations, digital certificates, and WS compliant user tokens.

The OPC UA security model is built on three layers: an application layer, a second communication layer, and a third transport layer. TCP is always used for data transfer. If the HTTPS protocol is used, TLS is used to encrypt transport layer traffic. The Communication Layer consists of a Secure Channel, which can perform message signing and encryption if desired. When a secure communication policy is chosen, this layer ensures the confidentiality and integrity of the messages exchanged. It also permits applications to communicate with one another to be authenticated. The Application Layer includes the session used to authenticate and authorize users. Unauthenticated users cannot view or edit data in the target system because all activities are performed within a session. The session is always connected via a secure, regularly renewed channel.

### 2.3.1 Connection Security

Establishing an OPC UA connection ensures application authentication and authorization using standard security techniques. An Application Instance Certificate is a standard X.509v3 certificate with additional fields for improved OPC UA validation for each application instance. The application developer must decide which certificate store the UA application will use. It is possible, for example, to use the PKI of an Active Directory. APIs for UA are available in several programming languages. When two programs establish a Secure Channel, the appropriate RSA public and private keys are used to complete a secure handshake. Both programs will perform the other side's authentication during the handshake, which is an Open Secure Channel service message [10].

### 2.3.2 Security Configuration

OPC UA allows for the free selection of the security mode to be used between each connection. The OPC UA specification defines three Message Security Modes: None, Sign, and SignAndEncrypt. These determine the level of security assigned to each message. Other Security Policies defined include Basic128Rsa15, Basic256, and Basic256Sha256. As security requirements evolve, new policies can be developed and old ones phased out. These define the enabled key sizes for Application Instance Certificates, as well as the signature algorithm and symmetric encryption algorithm used.

All OPC UA servers implementing the Standard Server Profile must support signing and encryption at the Basic128Rsa15 policy level or higher. Applications that only implement the Micro or Nano Embedded Server Profile are not required to provide any security capabilities. The server administrator can configure these policies. The client application always decides which security mode to use for each connection. This ensures that security is always available and can be easily activated [10].

### 2.3.3 User Authentication

User authentication happens at the session level. OPC UA defines alternative authentication mechanisms such as Anonymous, Username and Password, X.509 certificates, and external systems for user authentication, such as Kerberos via outer tokens. Again, based on the Server Profile that they implement, the server applications must support various alternatives. Moreover, once again, the client application chooses the authentication type for the connection from the options implemented and configured for the server [10].

### 2.3.4 Certificate handling

From the security perspective [11], is critical to provide with unique and appropriate only administrator read/write access to certificate stores used to hold private key. CRLs, trusted lists, and trusted CA lists are only accessible by an authorized administrator and, in the case of pull configurations, the application. Other eligible users may be granted read access, however the list of individuals who are allowed read access is a site decision.

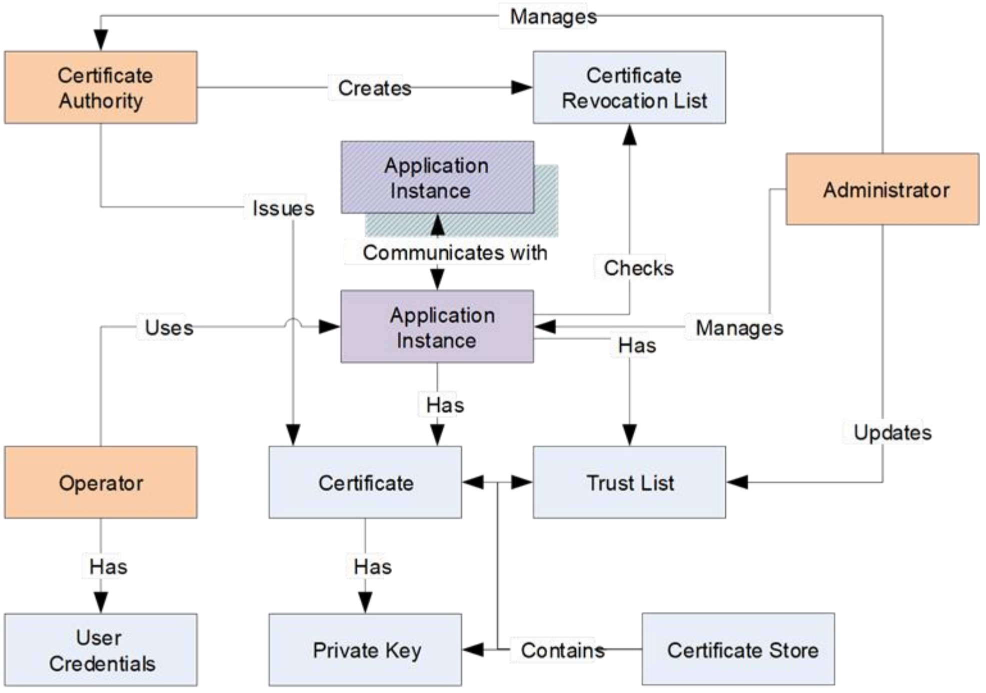


**Figure 6.** Process of the use of a certificate

To deploy a system with security requirements that uses CAs, the following are critical points:

i. An Application Instance is a single system component installed in the OPC UA environment. Each implementation has its own Application Identity Certificate, which it uses to interact to and identify itself with other OPC UA apps. Each Application Instance is identified by exclusive URIs. The OPC UA Application will communicate with other applications through a secure channel developed using asymmetric cryptography.

ii. Administrator is the one that control the security settings for Application Instances and handle the certificates related to an ICS system. This involves determining the contents of trust lists and overseeing any CA-related operations.

iii. Operator is the individual who utilizes the Application Instance is known as an Operator. For every particular OPC UA Application, there may be several Operators. Operator uses the assigned user credentials to verify access rights and continue with the activities of the Application Instance.

iv. An electronic ID that identifies an Operator/User is referred to as a User Credential. The Application Instance Certificate can be given to a Server after it has been used to construct a secure channel. It can be used to track activity and establish access rights (auditing).

v. Certificate Authority: A CA is a person or organization in charge of issuing and administering certificates. The CA verifies that the data in the Application Instance Certificate is genuine and signs it with a Digital Signature to ensure that it has not been tampered with. A certificate is issued to each CA and is used to generate Digital Signatures. CRLs are also the responsibility of a CA. Most of the time, it's a software suite that an administrator analyses or retrieves on a regular basis, usually when the application software raises a warning or notice that requires some sort of check action.

vi. OPC UA Application can have a certificate, which is an electronic ID. The ID contains information on the holder, the issuer, and a unique key for verifying Digital Signatures issued with the accompanying Private Key. These Certificates are commonly referred to as X.509 Certificates because their syntax conforms to the X.509 specification.

vii. A Certificate with no Certificate Authority is known as a self-signed Certificate. These kinds of certificates can be generated by any person and utilize in independently verifiable instances. There would be no CA or CRL in a system that only used self-signed certificates.

viii. A secret number that only the Certificate holder knows is known as private key. The holder of this secret can produce digital signatures and data decryption. The linked Certificate can no longer be trusted or used if this secret is leaked to unauthorized parties. It can be substituted or, if it has been issued by a Certificate CA, in can be revoked.

ix. The collection of certificates that an application program trusts is called trust list. If security is on, connections are not accepted from those that its certificate is not in the trust list.

x. Certificate Store is a file system location where Certificates and Private Keys can be stored. The Windows Certificate Store is a registry-based store that is available on all Windows systems. All UA systems can additionally support an OpenSSL Certificate Store, which is a file place where the certificates are stored. In all circumstances, the Certificate Store must be password-protected, with only administrators able to add new entries. Least privilege concept which means that only those with a genuine need for the data should be granted read or write access should be followed as concept.

xi. The revoked certificates by a CA that no longer are valid to run in applications, are identified in a revocation list.

### 2.3.5 Certificate Management

Since OPC UA security is based on X.509 certificates, the primary concern in practice will be certificate management. The OPC UA specification does not impose any specific strategy for certificate management.

Every OPC UA application stores certificates in its own Trust Store. All Application Instance Certificates encountered from connecting applications are categorized as trusted or rejected. When the number of connections is small, this selection can be made on a certificate basis. By default, the applications use self-signed certificates, which can only be trusted individually. A typical security environment builds trust in CAs, which sign certificates and manage CRLs. CA aids in the maintenance of the trust sequence between applications: all certificates signed by a trusted CA are automatically trusted until they are revoked [10].

Establishing Public Keys in Trust Lists can become time-consuming in systems with multiple Servers and Clients. Using a proper CA can significantly reduce installation and configuration difficulties in these cases. Certificate expiration management and CRL management are two additional services the CA provides. **Figure 7** gives an example of what this activity entails [12].

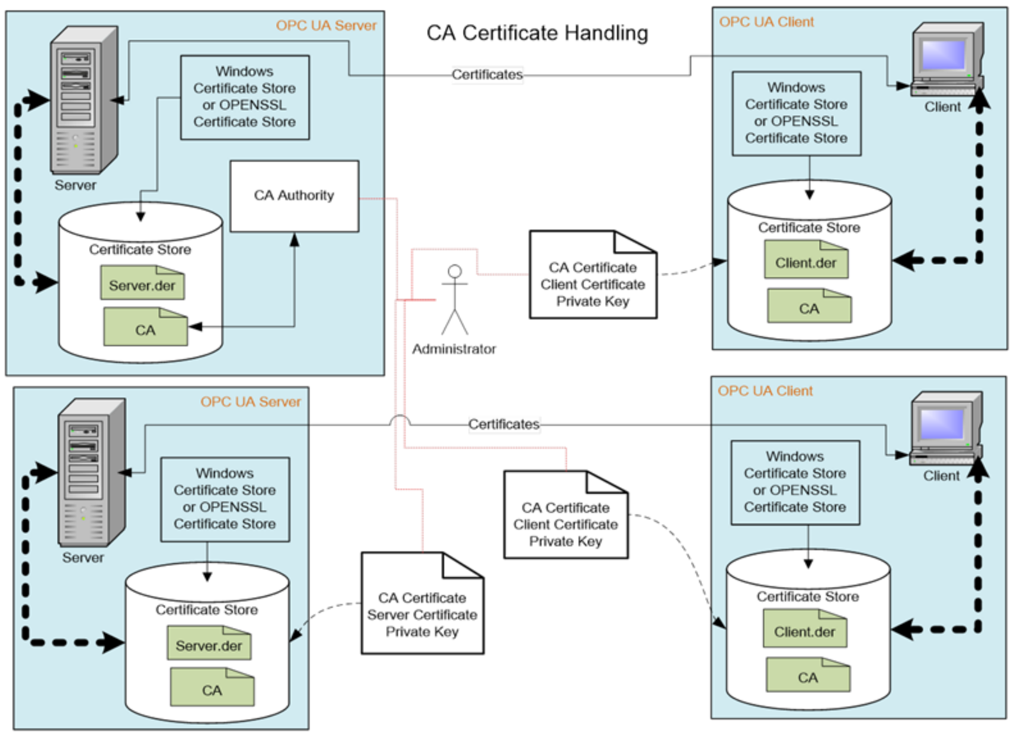

CA Certificate Handling
OPC UA Server
Windows Certificate Store or OPENSSL Certificate Store
Server
CA Authority
Certificate Store
Server.der
CA
Certificates
OPC UA Client
Client
Client.der
CA Certificate Client Certificate Private Key
Administrator
CA Certificate Server Certificate Private Key

**Figure 7.** CA Certificate handling

All clients and servers involved are provided by an CA signed certificate by the administrator, however just the CA Public Key must be deployed on all devices. When a Certificate expires and needs to be updated, the administrator simply needs to substitute the expired Certificate, and it is not necessary to copy the public key part to other sites.

A proprietary CA enables the issuing of certificates to be controlled by the firm. In most circumstances, using a commercial CA (such as VeriSign) is not advised. In most cases, an OPC UA application is configured to only trust other applications that the Company has identified as trustworthy.

All application developers must deal with certificate administration. Some apps may make advantage of system-wide certificate management, while in other cases, there will create self-signed certificates during the installation.

So, it is at this point where the Security Handler proposed in the beginning of this document to use it in the i4Q ecosystem is very well integrated with the OPC UA security part to integrate it as CA Authority to provide the required certificates to ensure crypto security to the communications.

# 3. Implementation Status

## 3.1 Current implementation

To start and in a nutshell, just mention that a PKI has been deployed to fulfil previously explained needs and the requirements in i4Q regarding to digital certificates and electronic identification. Next the used programs and installation steps will be described.

### 3.1.1 Virtualbox

To be flexible to install and integrate in already running ICS networks, Virtualbox virtualization product is proposed to setup the PKI infrastructure. It is flexible enough to integrate with most of operating systems.

CentOs operating system has been chosen to the first installation, but any other operating system can be used, such us, Ubuntu, to install in this case the EJBCA PKI manager.

So first, we are required to acquire and install CentOs operating system in a virtualbox running environment. Once the first installation is made, it would be suitable also to duplicate it and share between the i4Q partners, avoiding the initial installation phase.

### 3.1.2 HSM

HSM is a specialised device for keeping cryptographic keys that is extremely safe. It can be used for signing and authentication and can encrypt, decrypt, create, store, and handle digital keys. The goal is to keep sensitive information safe and secure, and no whole key can be extracted or exported from an HSM in a readable format. Businesses require centralized key creation, administration, and storage, as well as authentication and digital signature capabilities, which HSM delivers.

EJBCA can be setup to use different branding HSMs. Using PKCS#11 wrapper, HSM providers ease the integration of these devices in the required applications.

For i4Q the selected HSM has been the CardContact USB-Token [13] shown below in **Figure 8**.

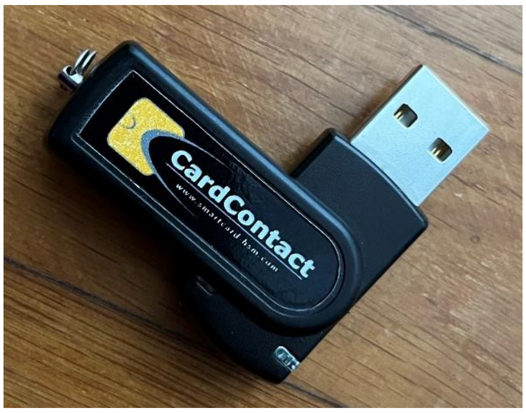


**Figure 8.** CardContact USB-Token HSM used in the project

It is a CCID compliant card reader combined with a smart card chip. Furthermore, it requires no additional hardware and works when connected to the computer's USB port.

As we have worked with VirtualBox, it does not automatically recognize the USB, so installing the Oracle extension for VirtualBox is necessary and reboot the machine. After that, it will be necessary to go to the virtual machine configuration and specify that the driver is USB 2.0 (OHCI+EHCI).

Once the USB-Token is visible from the virtual machine, it is possible to install it to be able to use it. First, we have to install a series of packages:

*sudo apt-get update*

*sudo apt-get install pcscd libccid libpcsclite-dev  libreadline-dev   docbook-xsl xsltproc git automake autoconf libtool pkg-config build-essential libssl-dev sqlite3 zlib1g-dev libsqlite3-dev cmake libseccomp-dev softhsm git-core*

With those steps the HSM is installed.

```
[ikerlan@localhost ~]$ pkcs11-tool -T
    Available slots:
    Slot 0 (0x0): Identiv uTrust 3512 SAM slot Token [CCID Interface] (55512030601
      token label: UserPIN (SmartCard-HSM)
      token manufacturer: www.CardContact.de
      token model: PKCS#15 emulated
      token flags: login required, rng, token initialized, PIN initialized
      hardware version: 24.13
      firmware version: 4.0
      serial num: DECC1200084
```

So, the system is ready to continue with the initialization procedure using the command “sc-hsm-tool” described in the installation procedure indicated before.

Once the HSM is ready, it must be associated with the PKI manager EJBCA. This is done modifying the web.properties file in ../ejbca/conf directory. The lines that must be modified are:

```
[ikerlan@localhost ~]$ vi ../Ejbca/conf/web.properties
    cryptotoken.p11.lib.60.name=OpenSC
    cryptotoken.p11.lib.60.file=/usr/lib64/opensc-pkcs11.so
```

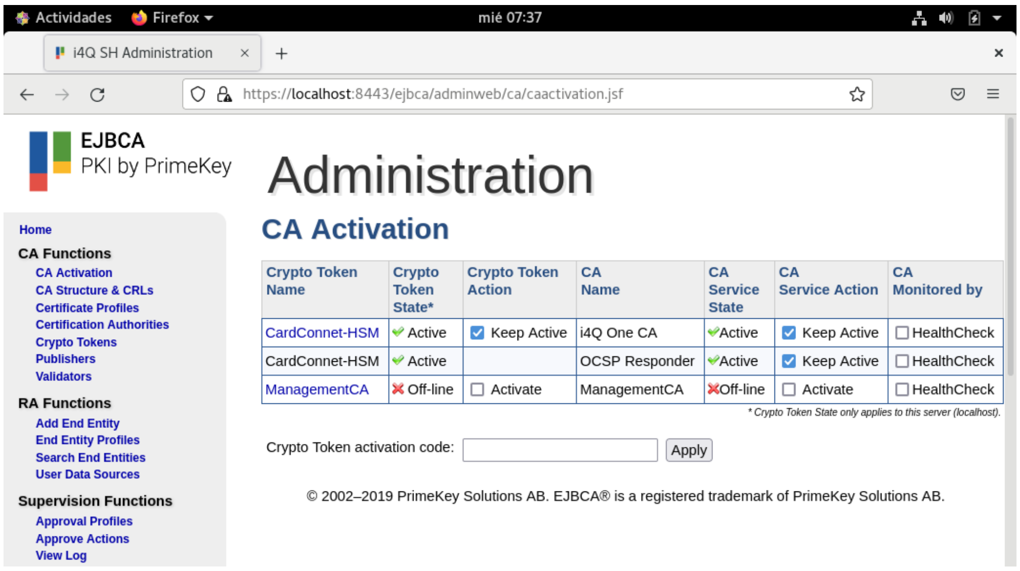
After saving the modifications Ejbca instance must be redeployed using ant command.

```
[ikerlan@localhost ~]$ cd ../Ejbca
[ikerlan@localhost ~]$ ant build deployear
```

### 3.1.3 EJBCA

As main PKI application program, EJBCA [14] community edition has been installed following the official installation guides [15]. The development and deployment are flexible enough if in the future a modification or extension is needed to support more complex PKI architecture, such as redundancy or the separation of the PKI roles like CA, VA and RA in different running instances.

Up to now, the PKI is ready to run. Next in 10 a view is shown.



Actividades   Firefox   mié 07:37

i4Q SH Administration

https://localhost:8443/ejbca/adminweb/ca/caactivation.jsf

EJBCA PKI by PrimeKey

# Administration

## CA Activation

Home

**CA Functions**
- CA Activation
- CA Structure & CRLs
- Certificate Profiles
- Certification Authorities
- Crypto Tokens
- Publishers
- Validators

**RA Functions**
- Add End Entity
- End Entity Profiles
- Search End Entities
- User Data Sources

**Supervision Functions**
- Approval Profiles
- Approve Actions
- View Log

| Crypto Token Name | Crypto Token State* | Crypto Token Action | CA Name | CA Service State | CA Service Action | CA Monitored by |
|---|---|---|---|---|---|---|
| CardConnet-HSM | ✓ Active | ☑ Keep Active | i4Q One CA | ✓Active | ☑ Keep Active | ☐ HealthCheck |
| CardConnet-HSM | ✓ Active | | OCSP Responder | ✓Active | ☑ Keep Active | ☐ HealthCheck |
| ManagementCA | ✗ Off-line | ☐ Activate | ManagementCA | ✗Off-line | ☐ Activate | ☐ HealthCheck |

* Crypto Token State only applies to this server (localhost).

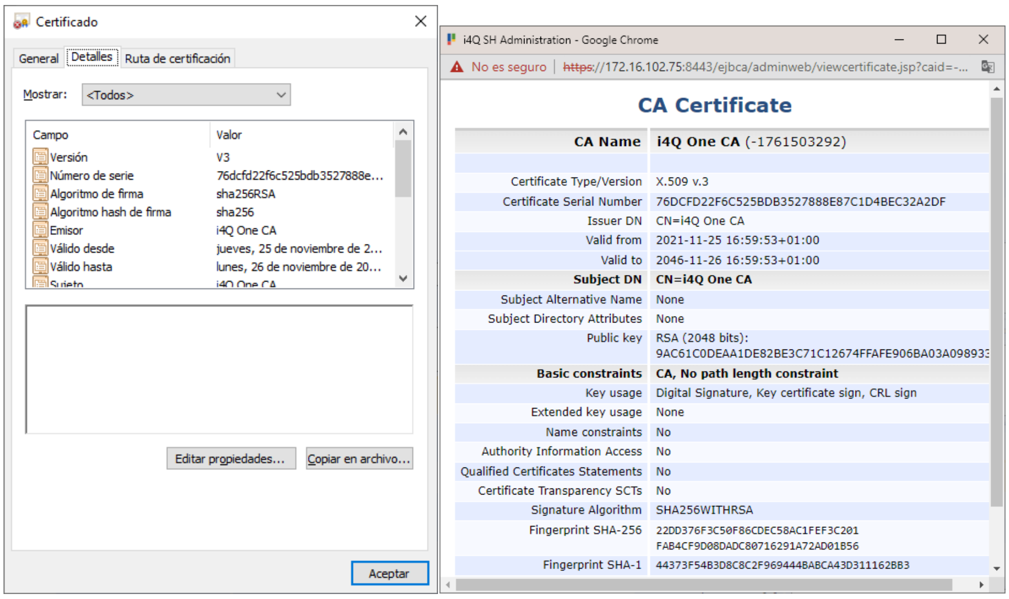
Crypto Token activation code: [ ] Apply

© 2002–2019 PrimeKey Solutions AB. EJBCA® is a registered trademark of PrimeKey Solutions AB.

**Figure 9.** Main view of Ejbca PKI manager

### *3.1.3.1 CA creation*

To check that PKI is able to create a CA, one root CA has been created with the common name "i4Q One CA" following the EJBCA procedures for this case.



Certificado

General | Detalles | Ruta de certificación

Mostrar: <Todos>

| Campo | Valor |
|---|---|
| Versión | V3 |
| Número de serie | 76dcfd22f6c525bdb3527888e... |
| Algoritmo de firma | sha256RSA |
| Algoritmo hash de firma | sha256 |
| Emisor | i4Q One CA |
| Válido desde | jueves, 25 de noviembre de 2... |
| Válido hasta | lunes, 26 de noviembre de 20... |
| Sujeto | i4Q One CA |

Editar propiedades... | Copiar en archivo... | Aceptar

i4Q SH Administration - Google Chrome

No es seguro | ~~https~~://172.16.102.75:8443/ejbca/adminweb/viewcertificate.jsp?caid=-...

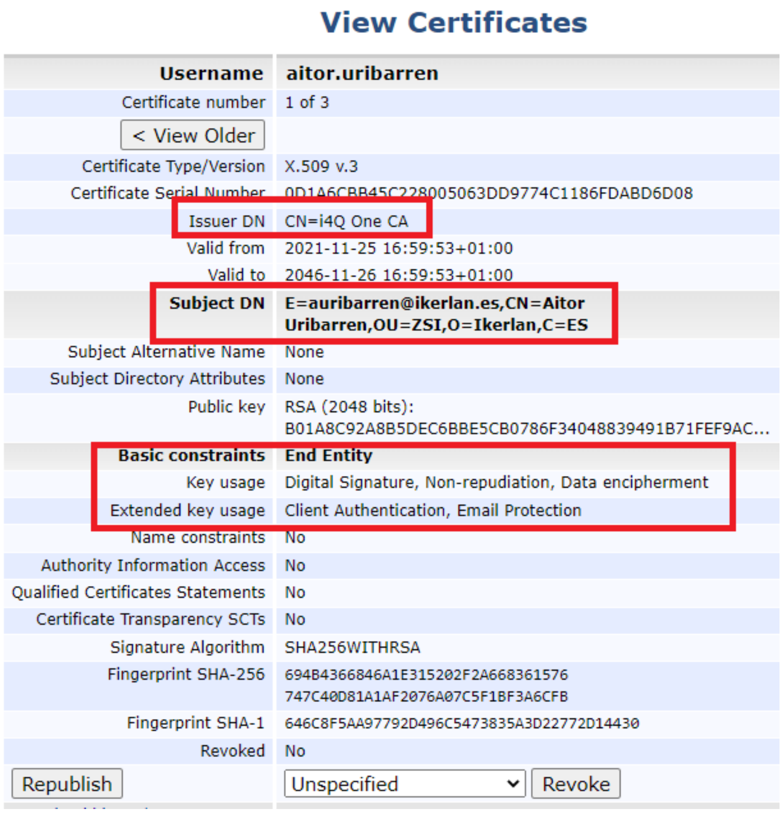
**CA Certificate**

| | |
|---|---|
| **CA Name** | **i4Q One CA** (-1761503292) |
| | |
| Certificate Type/Version | X.509 v.3 |
| Certificate Serial Number | 76DCFD22F6C525BDB3527888E87C1D4BEC32A2DF |
| Issuer DN | CN=i4Q One CA |
| Valid from | 2021-11-25 16:59:53+01:00 |
| Valid to | 2046-11-26 16:59:53+01:00 |
| **Subject DN** | **CN=i4Q One CA** |
| Subject Alternative Name | None |
| Subject Directory Attributes | None |
| Public key | RSA (2048 bits): 9AC61C0DEAA1DE82BE3C71C12674FFAFE906BA03A09893 |
| **Basic constraints** | **CA, No path length constraint** |
| Key usage | Digital Signature, Key certificate sign, CRL sign |
| Extended key usage | None |
| Name constraints | No |
| Authority Information Access | No |
| Qualified Certificates Statements | No |
| Certificate Transparency SCTs | No |
| Signature Algorithm | SHA256WITHRSA |
| Fingerprint SHA-256 | 22DD376F3C50F86CDEC58AC1FEF3C201 FAB4CF9D08DADC80716291A72AD01B56 |
| Fingerprint SHA-1 | 44373F54B3D8C8C2F969444BABCA43D311162BB3 |

**Figure 10.** i4Q One CA' certificate information

#### *3.1.3.2 User Certificate Creation*

Now with the previously created CA, there can be created 'end entity' certificates. It is named 'end entity' to the final destiny for a certificate, that is, a person, a device, a process… any relevant element that uses a certificate. Depending on the usage of the certificate, the key usage of the certificate will be different or shall be adapted. In the certificate shown in **Figure 11**, which is for a person, apart from the usual usages of digital signature, data encipherment and non repudiation, used in encrypted data transmission, email protection and client authentication flags were added, therefore the user can integrate it with the preferred email program to encrypt outgoing messages.



| | |
|---|---|
| **Username** | **aitor.uribarren** |
| Certificate number | 1 of 3 |
| Certificate Type/Version | X.509 v.3 |
| Certificate Serial Number | 0D1A6CBB45C228005063DD9774C1186FDABD6D08 |
| Issuer DN | CN=i4Q One CA |
| Valid from | 2021-11-25 16:59:53+01:00 |
| Valid to | 2046-11-26 16:59:53+01:00 |
| **Subject DN** | **E=auribarren@ikerlan.es,CN=Aitor Uribarren,OU=ZSI,O=Ikerlan,C=ES** |
| Subject Alternative Name | None |
| Subject Directory Attributes | None |
| Public key | RSA (2048 bits): B01A8C92A8B5DEC6BBE5CB0786F34048839491B71FEF9AC... |
| **Basic constraints** | **End Entity** |
| Key usage | Digital Signature, Non-repudiation, Data encipherment |
| Extended key usage | Client Authentication, Email Protection |
| Name constraints | No |
| Authority Information Access | No |
| Qualified Certificates Statements | No |
| Certificate Transparency SCTs | No |
| Signature Algorithm | SHA256WITHRSA |
| Fingerprint SHA-256 | 694B4366846A1E315202F2A668361576 747C40D81A1AF2076A07C5F1BF3A6CFB |
| Fingerprint SHA-1 | 646C8F5AA97792D496C5473835A3D22772D14430 |
| Revoked | No |

**Figure 11.** User certificate information

In the same way, the system is ready to configure and issue certificates for IIoT devices or services.

### 3.1.4 OPC-UA

#### *3.1.4.1 Overview*

We have analysed different options to implement OPC-UA in the project. First, it was decided to use OPC-Foundation, since it is the one who developed the OPC-UA protocol [9] and allows working with its own certificates. The problem is that to access part of the documentation it is necessary to be a member, so it has not been possible to implement the system to perform certified operations. Therefore, we decided to use the other option we analysed: Open62541. An open source OPC-UA that uses openSSL to generate the certificates.

### *3.1.4.2 Installation*

To install Open62541 it is necessary to install the following packages first:

```
[ikerlan@localhost ~]$ sudo apt-get install git build-essential gcc pkg-config cmake python3
                       sudo apt-get install cmake-curses-gui
                       sudo apt-get install libmbedtls-dev
                       sudo apt-get install check libsubunit-dev
                       sudo apt-get install python3-sphinx
                       pip install sphinx sphinx_rtd_theme
```

After instaling the packages, it is necessary to build Open62541.

```
        sudo cmake .. -DCMAKE_INSTALL_PREFIX=/home/maialen/install -
DUA_ENABLE_SUBSCRIPTIONS=ON -DUA_ENABLE_ENCRYPTION=ON -
DCMAKE_BUILD_TYPE=RelWithDebInfo -DUA_NAMESPACE_ZERO=Full ..
```

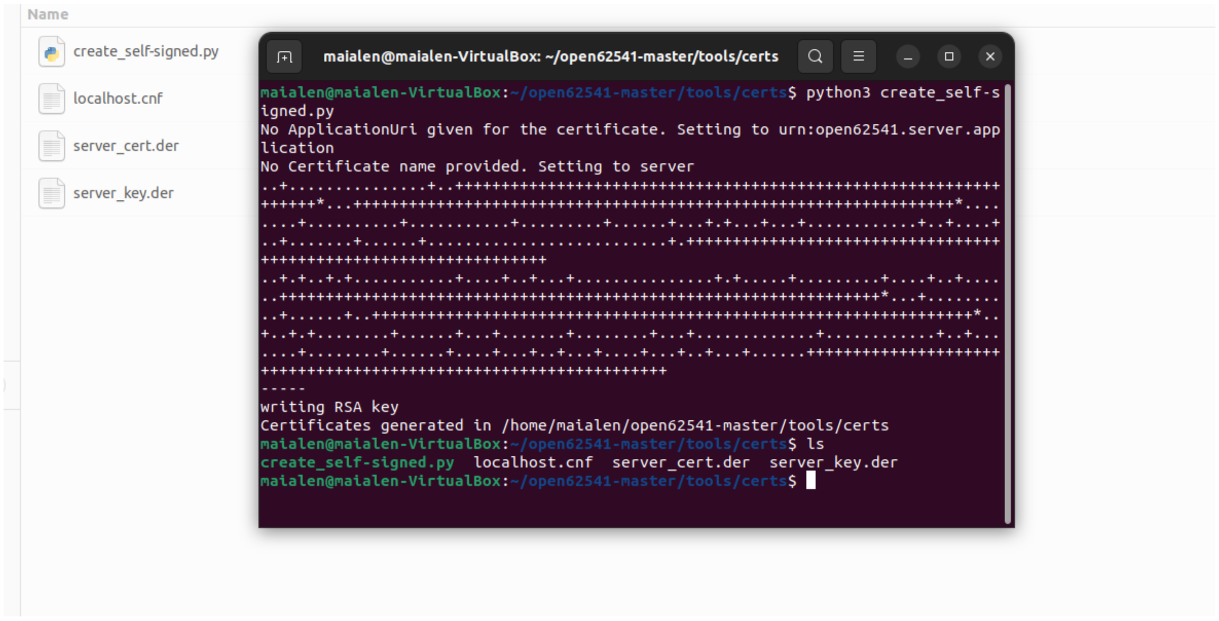
```
                       sudo make -j
                       sudo make install
```

Then, it is possible to use Open62541.

### *3.1.4.3 Create a certificate*

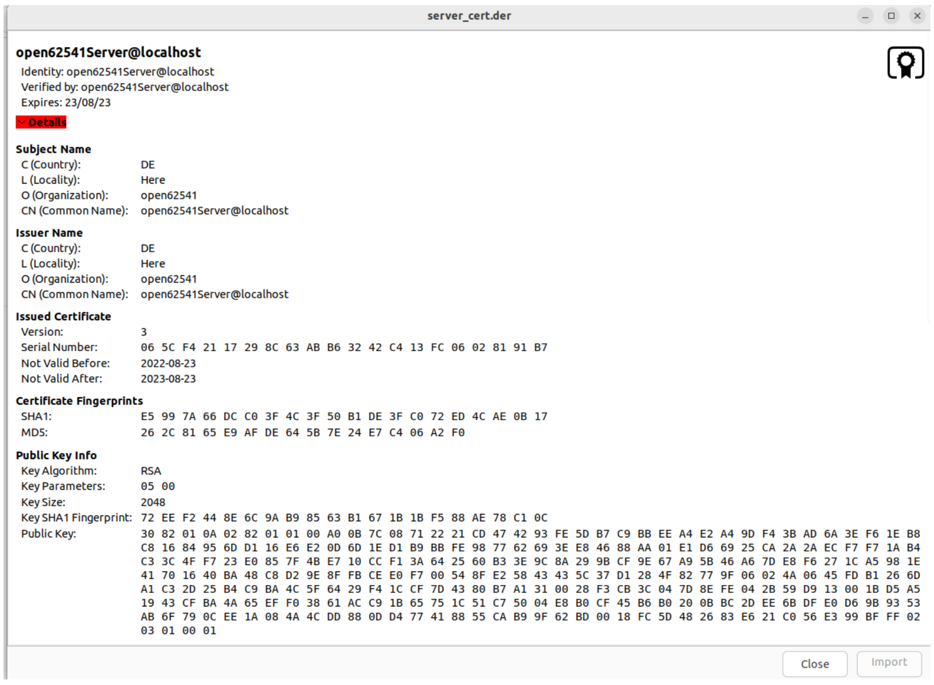
To perform different certified operations, it is necessary to create the certificates. These certificates can be generated in different ways, but to generate the certificate and the key for the initial tests code *create_self-signed.py* has been used. This code is on the same folder as Open62541, in /tools/certs path. It is prepared to generate certificates to use Open62541 to do certificated operations.

```
[ikerlan@localhost ~]$python3 create_self-signed.py
```



```
maialen@maialen-VirtualBox:~/open62541-master/tools/certs$ python3 create_self-s
igned.py
No ApplicationUri given for the certificate. Setting to urn:open62541.server.app
lication
No Certificate name provided. Setting to server
..+...............+..+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++*...+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*....
....+..........+..........+.........+......+...+.+...+...+............+..+....+
..+.......+......+...........................+.+++++++++++++++++++++++++++++++++
+++++++++++++++++++++++++++++++
..+.+..+.+...........+....+..+...+...............+.+.....+.........+....+..+....
..+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*...+........
..+......+..++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*..
+..+.+........+......+...+.......+........+...+.............+............+..+...
....+........+......+....+...+..+...+....+...+..+...+......+++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++
-----
writing RSA key
Certificates generated in /home/maialen/open62541-master/tools/certs
maialen@maialen-VirtualBox:~/open62541-master/tools/certs$ ls
create_self-signed.py  localhost.cnf  server_cert.der  server_key.der
maialen@maialen-VirtualBox:~/open62541-master/tools/certs$ 
```

**Figure 12.** Generation of certificate key



**open62541Server@localhost**
Identity: open62541Server@localhost
Verified by: open62541Server@localhost
Expires: 23/08/23

**Subject Name**

| | |
|---|---|
| C (Country): | DE |
| L (Locality): | Here |
| O (Organization): | open62541 |
| CN (Common Name): | open62541Server@localhost |

**Issuer Name**

| | |
|---|---|
| C (Country): | DE |
| L (Locality): | Here |
| O (Organization): | open62541 |
| CN (Common Name): | open62541Server@localhost |

**Issued Certificate**

| | |
|---|---|
| Version: | 3 |
| Serial Number: | 06 5C F4 21 17 29 8C 63 AB B6 32 42 C4 13 FC 06 02 81 91 B7 |
| Not Valid Before: | 2022-08-23 |
| Not Valid After: | 2023-08-23 |

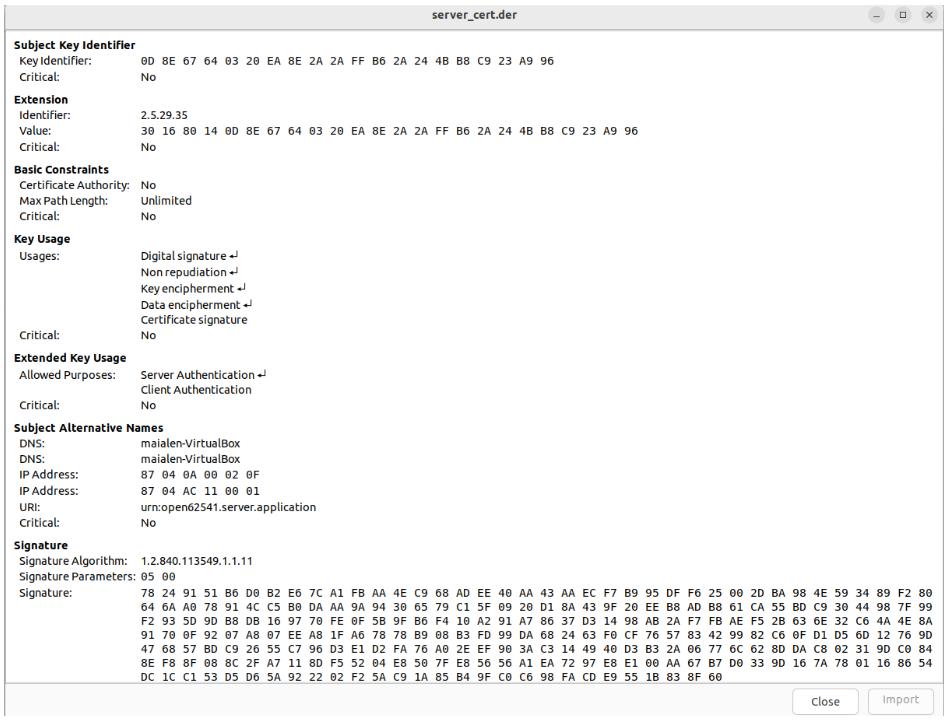
**Certificate Fingerprints**

| | |
|---|---|
| SHA1: | E5 99 7A 66 DC C0 3F 4C 3F 50 B1 DE 3F C0 72 ED 4C AE 0B 17 |
| MD5: | 26 2C 81 65 E9 AF DE 64 5B 7E 24 E7 C4 06 A2 F0 |

**Public Key Info**

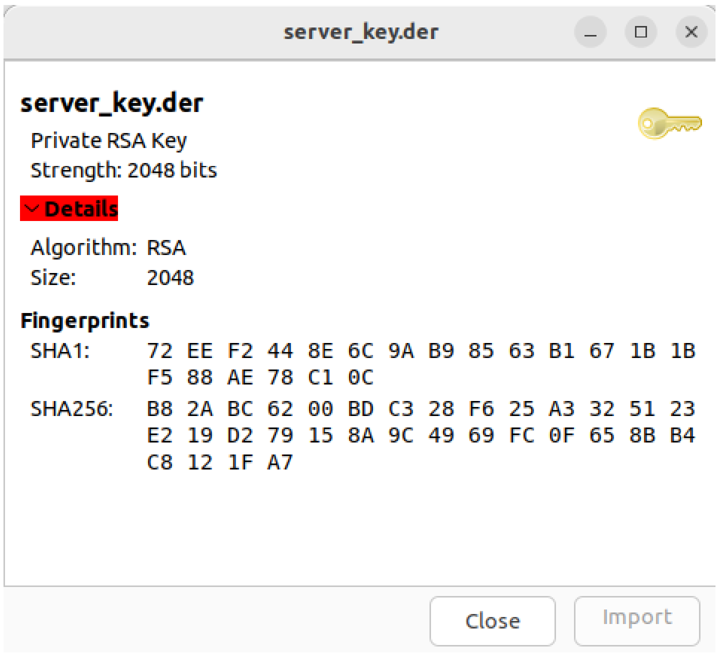
| | |
|---|---|
| Key Algorithm: | RSA |
| Key Parameters: | 05 00 |
| Key Size: | 2048 |
| Key SHA1 Fingerprint: | 72 EE F2 44 8E 6C 9A B9 85 63 B1 67 1B 1B F5 88 AE 78 C1 0C |
| Public Key: | 30 82 01 0A 02 82 01 01 00 A0 0B 7C 08 71 22 21 CD 47 42 93 FE 5D B7 C9 BB EE A4 E2 A4 9D F4 3B AD 6A 3E F6 1E B8 C8 16 84 95 6D D1 16 E6 E2 0D 6D 1E D1 B9 BB FE 98 77 62 69 3E E8 46 88 AA 01 E1 D6 69 25 CA 2A 2A EC F7 F7 1A B4 C3 3C 4F F7 23 E0 85 7F 4B E7 10 CC F1 3A 64 25 60 B3 3E 9C 8A 29 9B CF 9E 67 A9 5B 46 A6 7D E8 F6 27 1C A5 98 1E 41 70 16 40 BA 48 C8 D2 9E 8F FB CE E0 F7 00 54 8F E2 58 43 43 5C 37 D1 28 4F 82 77 9F 06 02 4A 06 45 FD B1 26 6D A1 C3 2D 25 B4 C9 BA 4C 5F 64 29 F4 1C CF 7D 43 80 B7 A1 31 00 28 F3 CB 3C 04 7D 8E FE 04 2B 59 D9 13 00 1B D5 A5 19 43 CF BA 4A 65 EF F0 38 61 AC C9 1B 65 75 1C 51 C7 50 04 E8 B0 CF 45 B6 B0 20 0B BC 2D EE 6B DF E0 D6 9B 93 53 AB 6F 79 0C EE 1A 08 4A 4C DD 88 0D D4 77 41 88 55 CA B9 9F 62 BD 00 18 FC 5D 48 26 83 E6 21 C0 56 E3 99 BF FF 02 03 01 00 01 |

**Figure 13.** Server certificate (part1)



```
Subject Key Identifier
  Key Identifier:        0D 8E 67 64 03 20 EA 8E 2A 2A FF B6 2A 24 4B B8 C9 23 A9 96
  Critical:              No
Extension
  Identifier:            2.5.29.35
  Value:                 30 16 80 14 0D 8E 67 64 03 20 EA 8E 2A 2A FF B6 2A 24 4B B8 C9 23 A9 96
  Critical:              No
Basic Constraints
  Certificate Authority: No
  Max Path Length:       Unlimited
  Critical:              No
Key Usage
  Usages:                Digital signature ↵
                         Non repudiation ↵
                         Key encipherment ↵
                         Data encipherment ↵
                         Certificate signature
  Critical:              No
Extended Key Usage
  Allowed Purposes:      Server Authentication ↵
                         Client Authentication
  Critical:              No
Subject Alternative Names
  DNS:                   maialen-VirtualBox
  DNS:                   maialen-VirtualBox
  IP Address:            87 04 0A 00 02 0F
  IP Address:            87 04 AC 11 00 01
  URI:                   urn:open62541.server.application
  Critical:              No
Signature
  Signature Algorithm:   1.2.840.113549.1.1.11
  Signature Parameters:  05 00
  Signature:             78 24 91 51 B6 D0 B2 E6 7C A1 FB AA 4E C9 68 AD EE 40 AA 43 AA EC F7 B9 95 DF F6 25 00 2D BA 98 4E 59 34 89 F2 80
                         64 6A A0 78 91 4C C5 B0 DA AA 9A 94 30 65 79 C1 5F 09 20 D1 8A 43 9F 20 EE B8 AD B8 61 CA 55 BD C9 30 44 98 7F 99
                         F2 93 5D 9D B8 DB 16 97 70 FE 0F 5B 9F B6 F4 10 A2 91 A7 86 37 D3 14 98 AB 2A F7 FB AE F5 2B 63 6E 32 C6 4A 4E 8A
                         91 70 0F 92 07 A8 07 EE A8 1F A6 78 78 B9 08 B3 FD 99 DA 68 24 63 F0 CF 76 57 83 42 99 82 C6 0F D1 D5 6D 12 76 9D
                         47 68 57 BD C9 26 55 C7 96 D3 E1 D2 FA 76 A0 2E EF 90 3A C3 14 49 40 D3 B3 2A 06 77 6C 62 8D DA C8 02 31 9D C0 84
                         8E F8 8F 08 8C 2F A7 11 8D F5 52 04 E8 50 7F E8 56 56 A1 EA 72 97 E8 E1 00 AA 67 B7 D0 33 9D 16 7A 78 01 16 86 54
                         DC 1C C1 53 D5 D6 5A 92 22 02 F2 5A C9 1A 85 B4 9F C0 C6 98 FA CD E9 55 1B 83 8F 60
```

**Figure 14.** Server certificate (part2)



```
server_key.der
  Private RSA Key
  Strength: 2048 bits
Details
  Algorithm:  RSA
  Size:       2048
Fingerprints
  SHA1:       72 EE F2 44 8E 6C 9A B9 85 63 B1 67 1B 1B
              F5 88 AE 78 C1 0C
  SHA256:     B8 2A BC 62 00 BD C3 28 F6 25 A3 32 51 23
              E2 19 D2 79 15 8A 9C 49 69 FC 0F 65 8B B4
              C8 12 1F A7
```

**Figure 15.** Server key

### *3.1.4.4 Server*

It is important to do every operation in open62541 folder.

```
mkdir myserver
```

Create the server code in this folder and compile it.

```
gcc -std=c99 -I$HOME/install/include -L$HOME/install/lib miservidor.c -lopen62541 -lmbedtls -lmbedx509 -lmbedcrypto -o myserver
```

Although the command to compile is the previous one a fatal error appears.

```
miservidor.c:23:10: fatal error: open62541/plugin/log_stdout.h: No such file or directory
```

To solve it we used the next command:

```
gcc -std=c99 -I /home/user/install/include -L /home/user/install/lib miservidor.c -lopen62541 -lmbedtls -lmbedx509 -lmbedcrypto -o myserver
```

```
./ myserver
```

Although the compilation is correct, there is an error when executing it, because it does not found some libraries. So firstly, it is necessary to find where they are the libraries and then export them.

```
sudo find / -name log_stdout.h
```

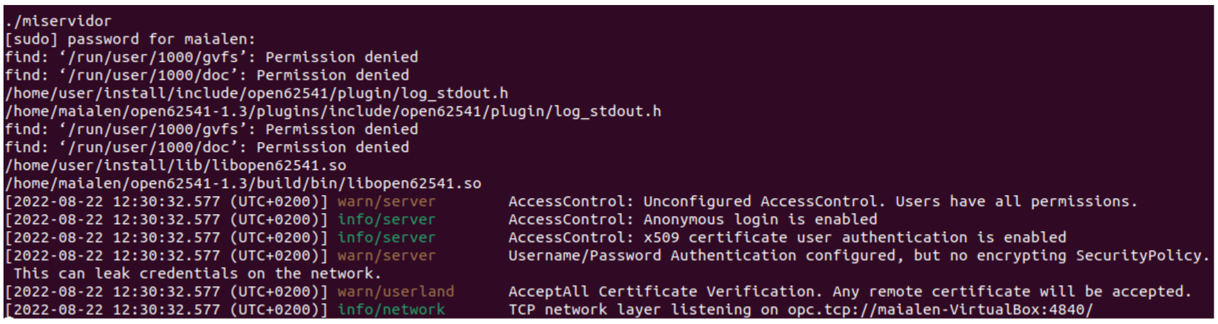
```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/user/install/include/open62541/plugin
```

```
sudo find / -name libopen62541.so
```

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/user/install/lib
```

```
./miservidor
```

Doing this steps the server will be running.



```
./miservidor
[sudo] password for maialen:
find: '/run/user/1000/gvfs': Permission denied
find: '/run/user/1000/doc': Permission denied
/home/user/install/include/open62541/plugin/log_stdout.h
/home/maialen/open62541-1.3/plugins/include/open62541/plugin/log_stdout.h
find: '/run/user/1000/gvfs': Permission denied
find: '/run/user/1000/doc': Permission denied
/home/user/install/lib/libopen62541.so
/home/maialen/open62541-1.3/build/bin/libopen62541.so
[2022-08-22 12:30:32.577 (UTC+0200)] warn/server        AccessControl: Unconfigured AccessControl. Users have all permissions.
[2022-08-22 12:30:32.577 (UTC+0200)] info/server        AccessControl: Anonymous login is enabled
[2022-08-22 12:30:32.577 (UTC+0200)] info/server        AccessControl: x509 certificate user authentication is enabled
[2022-08-22 12:30:32.577 (UTC+0200)] warn/server        Username/Password Authentication configured, but no encrypting SecurityPolicy.
 This can leak credentials on the network.
[2022-08-22 12:30:32.577 (UTC+0200)] warn/userland      AcceptAll Certificate Verification. Any remote certificate will be accepted.
[2022-08-22 12:30:32.577 (UTC+0200)] info/network       TCP network layer listening on opc.tcp://maialen-VirtualBox:4840/
```

**Figure 16.** Server running

#### *3.1.4.5 Client*

The steps to execute the client are very similar of the ones of the server. It is necessary to create a file that runs the client, compile it, and execute it.

In Opem62541 file:

```
mkdir myclient
```

```
cd examples
```

```
cp tutorial_client_firststeps.c /home/user/open62541-1.3/myclient
```

```
cd ..
```

```
cd myclient
```

```
gcc -std=c99 -I$HOME/install/include -L$HOME/install/lib miservidor.c -lopen62541 -lmbedtls -lmbedx509 -lmbedcrypto -o myclient
```

This is the command specified on the official documentation. However, an error appears because it has some problems to find log_stdout.h and libopen62541.so files. In this case, it is necessary to find them.

```
sudo find / -name log_stdout.h
```

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/user/install/include/open62541/plugin
```

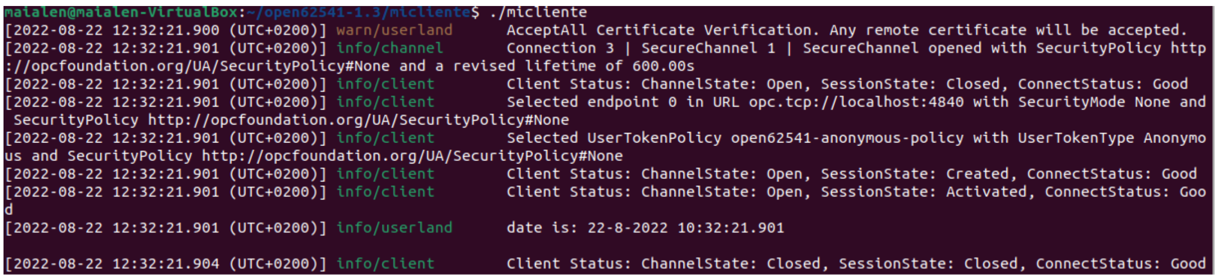
```
sudo find / -name libopen62541.so
```

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/user/install/lib
```

Once the files are located the code can be executed without problems.

```
./ myclient
```

With this process the client Will be running.



```
maialen@maialen-VirtualBox:~/open62541-1.3/micliente$ ./micliente
[2022-08-22 12:32:21.900 (UTC+0200)] warn/userland      AcceptAll Certificate Verification. Any remote certificate will be accepted.
[2022-08-22 12:32:21.901 (UTC+0200)] info/channel       Connection 3 | SecureChannel 1 | SecureChannel opened with SecurityPolicy http
://opcfoundation.org/UA/SecurityPolicy#None and a revised lifetime of 600.00s
[2022-08-22 12:32:21.901 (UTC+0200)] info/client        Client Status: ChannelState: Open, SessionState: Closed, ConnectStatus: Good
[2022-08-22 12:32:21.901 (UTC+0200)] info/client        Selected endpoint 0 in URL opc.tcp://localhost:4840 with SecurityMode None and
 SecurityPolicy http://opcfoundation.org/UA/SecurityPolicy#None
[2022-08-22 12:32:21.901 (UTC+0200)] info/client        Selected UserTokenPolicy open62541-anonymous-policy with UserTokenType Anonymo
us and SecurityPolicy http://opcfoundation.org/UA/SecurityPolicy#None
[2022-08-22 12:32:21.901 (UTC+0200)] info/client        Client Status: ChannelState: Open, SessionState: Created, ConnectStatus: Good
[2022-08-22 12:32:21.901 (UTC+0200)] info/client        Client Status: ChannelState: Open, SessionState: Activated, ConnectStatus: Goo
d
[2022-08-22 12:32:21.901 (UTC+0200)] info/userland      date is: 22-8-2022 10:32:21.901

[2022-08-22 12:32:21.904 (UTC+0200)] info/client        Client Status: ChannelState: Closed, SessionState: Closed, ConnectStatus: Good
```

**Figure 17.** Client running

### 3.1.5 Lamassu

Once we have set up the server and the client and see that it works correctly, we have tested the options for working with certificates.

To work in Open62541 with certificates it is necessary to take into account that it creates them as it has been explained in the section "Create a certificate". This certificate is stored in the folder certs and it is

from where the OPC-UA takes the certificates. Therefore, if you want to work with other certificates, it is enough to leave them in that folder. Therefore, we have used Lamassu to generate the certificates for the OPC-UA.

Lamasu is a PKI that has been designed to run hardware and cloud infrastructure for industrial scenarios . It is open source and allows flexible configuration and management of certificates. It is also scalable by adding new nodes and offers the ability to detect failures [16].

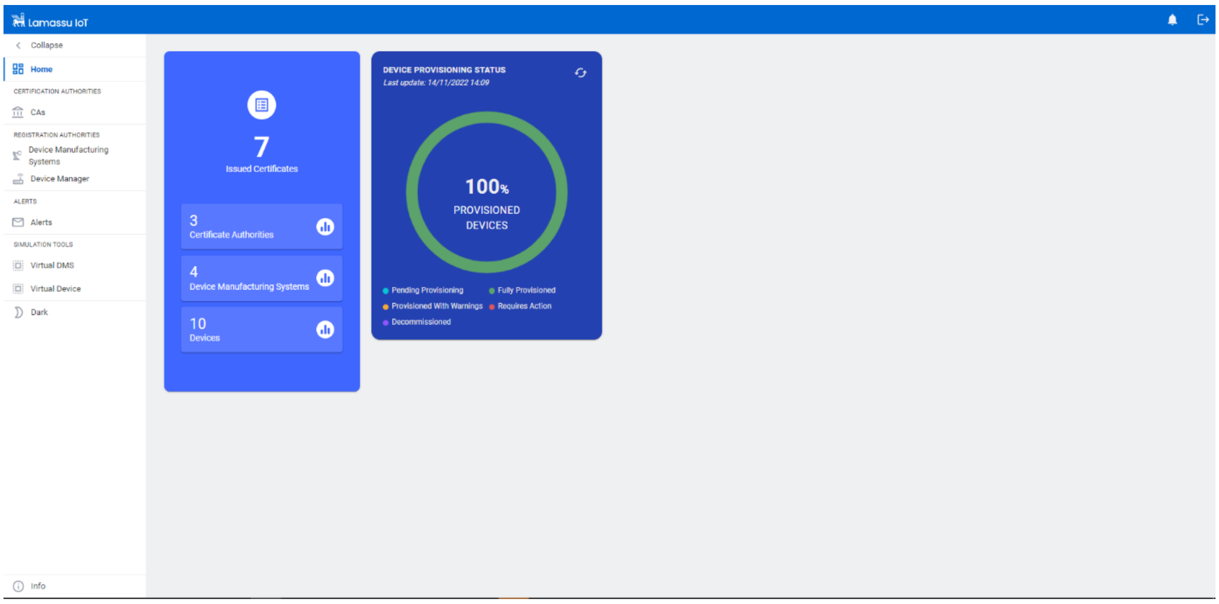


**Figure 18.** Lamassu interface

Lamassu will generate the certificates and store them in the /tools/certs folder so that OPC-UA can access them and perform certified operations.

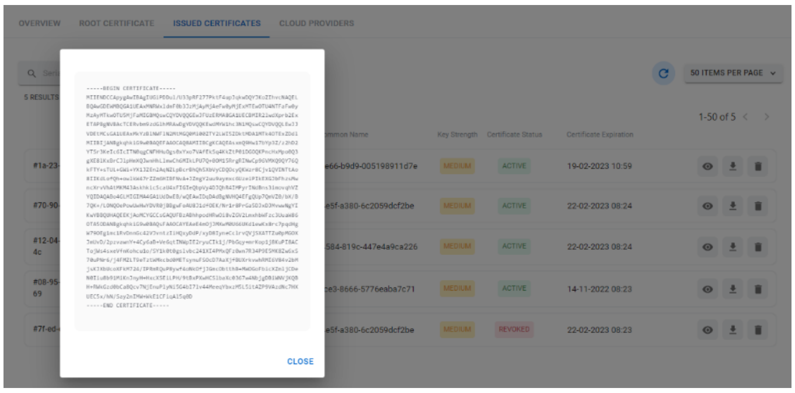


**Figure 19.** certificate created with Lamassu

### 3.1.6 Connection API's

Once the PKI infrastructure has been deployed, any device or electronic subject identified in the i4Q ecosystem will request an electronic certificate. Due to the variety of devices and software components that can request certificates, some standardized way needs to be defined. In computing this is achieved defining an API.

The fundamental functions are accessed remotely using a Web Service SOAP API Interface over client-authenticated HTTPS. All development languages that can process SOAP messages are compatible with the SOAP API. Java, C#, and PHP are examples of programming languages.

#### *3.1.6.1 API-WS Interface*

The JAX-WS 2.0 Web Service Interface is used to remotely access basic functions via client authenticated HTTPS.

Pointing to the EJBCA server URL in the https://SERVER-IP:8443/ejbca/ejbcaws/ejbcaws?wsdl could be obtained the corresponding WSDL with the description of the functions to use to connect using web service interface.

#### *3.1.6.2 API-Rest Interface*

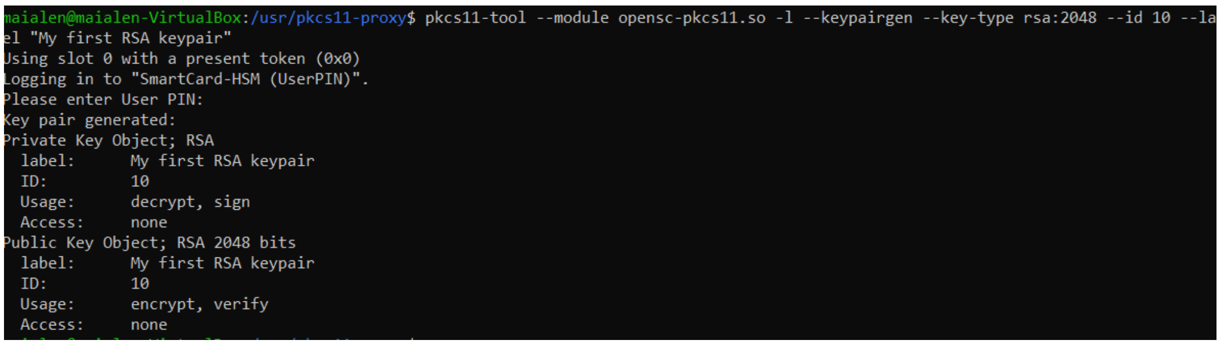
RESTful URLs are used to send API requests, which use the conventional HTTP methods of GET, POST, and PUT. A JSON request body is also accepted by some endpoints. An HTTP header with an RFC 2616 status code and an application/json response body are typically included in responses.

## 3.2 Operations with HSM

### 3.2.1 Generate key pair

Using keypairgen command it is possible to generate a key pair.

```
pkcs11-tool --module opensc-pkcs11.so -l --keypairgen --key-type rsa:2048 --id 10 --label "My first RSA keypair"
```



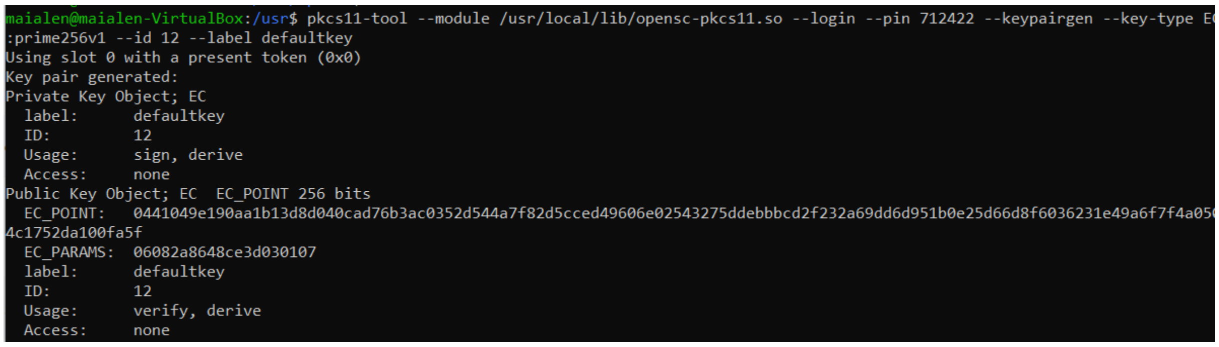
```
maialen@maialen-VirtualBox:/usr/pkcs11-proxy$ pkcs11-tool --module opensc-pkcs11.so -l --keypairgen --key-type rsa:2048 --id 10 --la
el "My first RSA keypair"
Using slot 0 with a present token (0x0)
Logging in to "SmartCard-HSM (UserPIN)".
Please enter User PIN:
Key pair generated:
Private Key Object; RSA
  label:      My first RSA keypair
  ID:         10
  Usage:      decrypt, sign
  Access:     none
Public Key Object; RSA 2048 bits
  label:      My first RSA keypair
  ID:         10
  Usage:      encrypt, verify
  Access:     none
```

**Figure 20.** Generate a RSA keypair

In the result of generating an RSA keypair appears, as it is shown in Figure 20. However, it is also possible to generate other type of keypairs such as EC:

```
pkcs11-tool --module /usr/local/lib/opensc-pkcs11.so --login --pin 712422 --keypairgen --key-type EC:prime256v1 --id 12 --label defaultkey
```

```
pkcs15-tool --read-public-key 12
```



```
maialen@maialen-VirtualBox:/usr$ pkcs11-tool --module /usr/local/lib/opensc-pkcs11.so --login --pin 712422 --keypairgen --key-type E
:prime256v1 --id 12 --label defaultkey
Using slot 0 with a present token (0x0)
Key pair generated:
Private Key Object; EC
  label:      defaultkey
  ID:         12
  Usage:      sign, derive
  Access:     none
Public Key Object; EC  EC_POINT 256 bits
  EC_POINT:   0441049e190aa1b13d8d040cad76b3ac0352d544a7f82d5cced49606e02543275ddebbbcd2f232a69dd6d951b0e25d66d8f6036231e49a6f7f4a05
4c1752da100fa5f
  EC_PARAMS:  06082a8648ce3d030107
  label:      defaultkey
  ID:         12
  Usage:      verify, derive
  Access:     none
```

**Figure 21.** Generate a EC key

**Figure 21** shows the result of generating a EC key.

### 3.2.2 Encrypting and Decrypting data

the data has been encrypted and decrypted using the HSM using the pin of the configuration.

```
dd if=/dev/urandom of=/tmp/mydata bs=1 count=245
```

```
sha256sum /tmp/mydata
```

```
openssl pkeyutl -inkey /tmp/publickey.pem -pubin -encrypt -in /tmp/mydata -out
/tmp/encrypteddata.pkcs1
```

```
pkcs15-crypt --decipher --key 10 --input /tmp/encrypteddata.pkcs1 --pkcs1 --raw >
/tmp/decrypteddata
```

```
sha256sum /tmp/decrypteddata
```

## 3.3 WISE 710

The WISE-710 industrial IoT gateway (see Figure 22) is an open platform with a Cortex A9 processor, equipped with 3 RS-232/485 serial ports Cortex A9 processor, equipped with 3 RS-232/485 serial ports, 2 10/100/1000 Ethernet ports and 4 DI/DO, making it a more than suitable solution for legacy equipment. DI/DO ports, making it a more than suitable solution for legacy equipment in new networks. of the new networks. All the information collected is secured by encryption technology using a key encryption technology by an embedded security chip. Due to its wide range of I/O I/O options and wide operating temperature range, this gateway plays a role as a datalogger for multiple applications. datalogger for multiple industrial applications [17].

It has the following features:

- Freescale i.MX 6 Dual Lite A9 1GHz
- Support for MQTT, LWM2M client for cloud communications
- Embedded Microsoft Azure certified security chip
- Micro SD and online firmware upgrade support
- Support for Node ID by software
- Micro SD data logger support
- Operating system Yocto 2.1/Ubuntu 16.04

- Operating temperature: -20 ~ 55 ºC

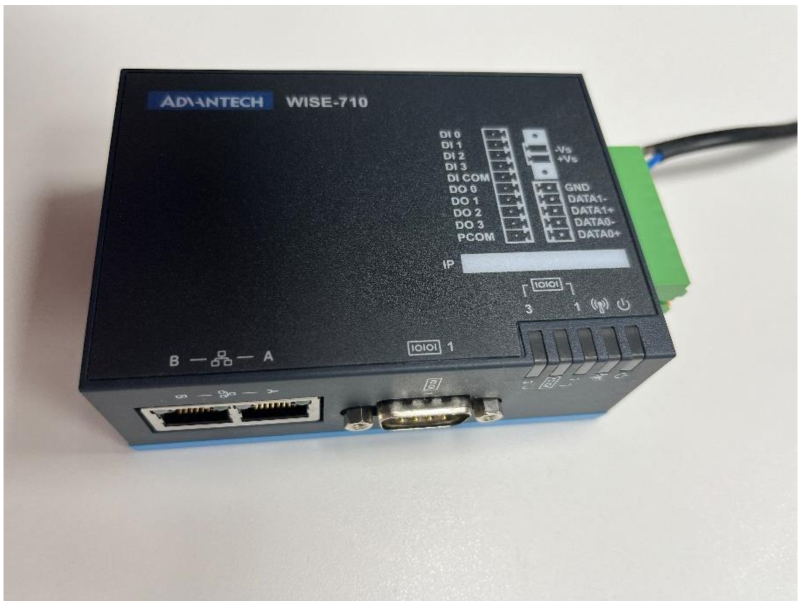


**Figure 22.** WISE-710

### 3.3.1 Configuration

The configuration and installation of WISE710 has been done in Ubuntu 18.04. The steps to configurate it, are the following ones:

```
sudo apt update
```

```
sudo apt upgrade
```

This process is long, and it is necessary to wait a considerable time. When it is upgraded, it is posible to Access and configure the WISE-710.

```
sudo reboot
```

```
sudo do-release-upgrade
```

Once finished, the device will reboot, and it will be possible to connect to it. With this update process, it is enabled to install pkcs11 and access the secure element. secure element. For this, two tools will be used: i2ctools and the cryptoauthlib library.

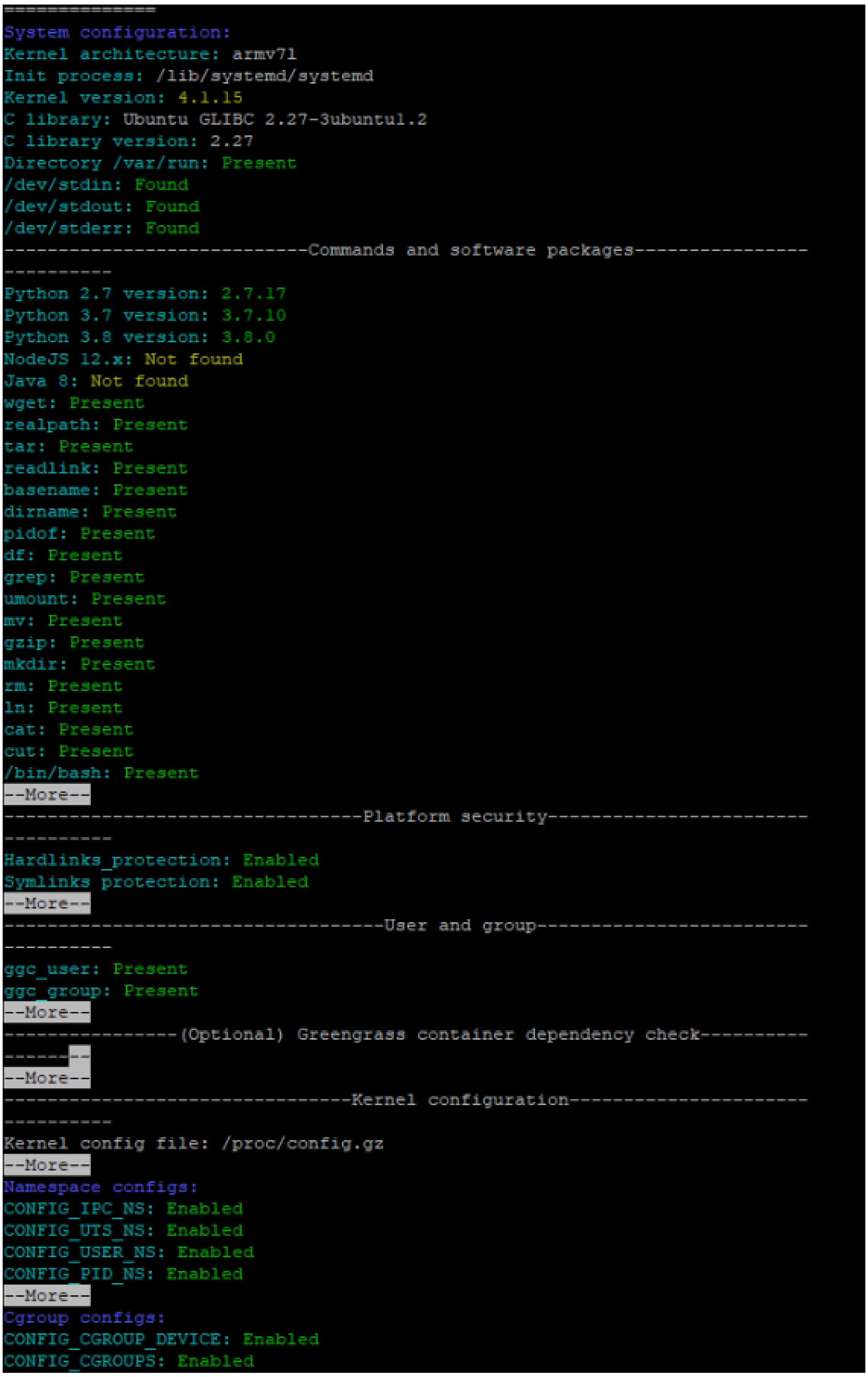
Next, we will proceed to install the AWS IoT Greengrass Core software on the device, which will serve to make the device act as the Core of the group.  device, which will serve to make the device act as the Core of the cluster, communicating with the AWS IoT cloud and the other devices in the cluster. To install it, it is necessary to have a package administrator (apt, yum or opkg), the permission to execute with sudo, and credentials of AWS.

```
$ mkdir greengrass-dependency-checker-GGCv1.11.x
```

```
$ cd greengrass-dependency-checker-GGCv1.11.x
```

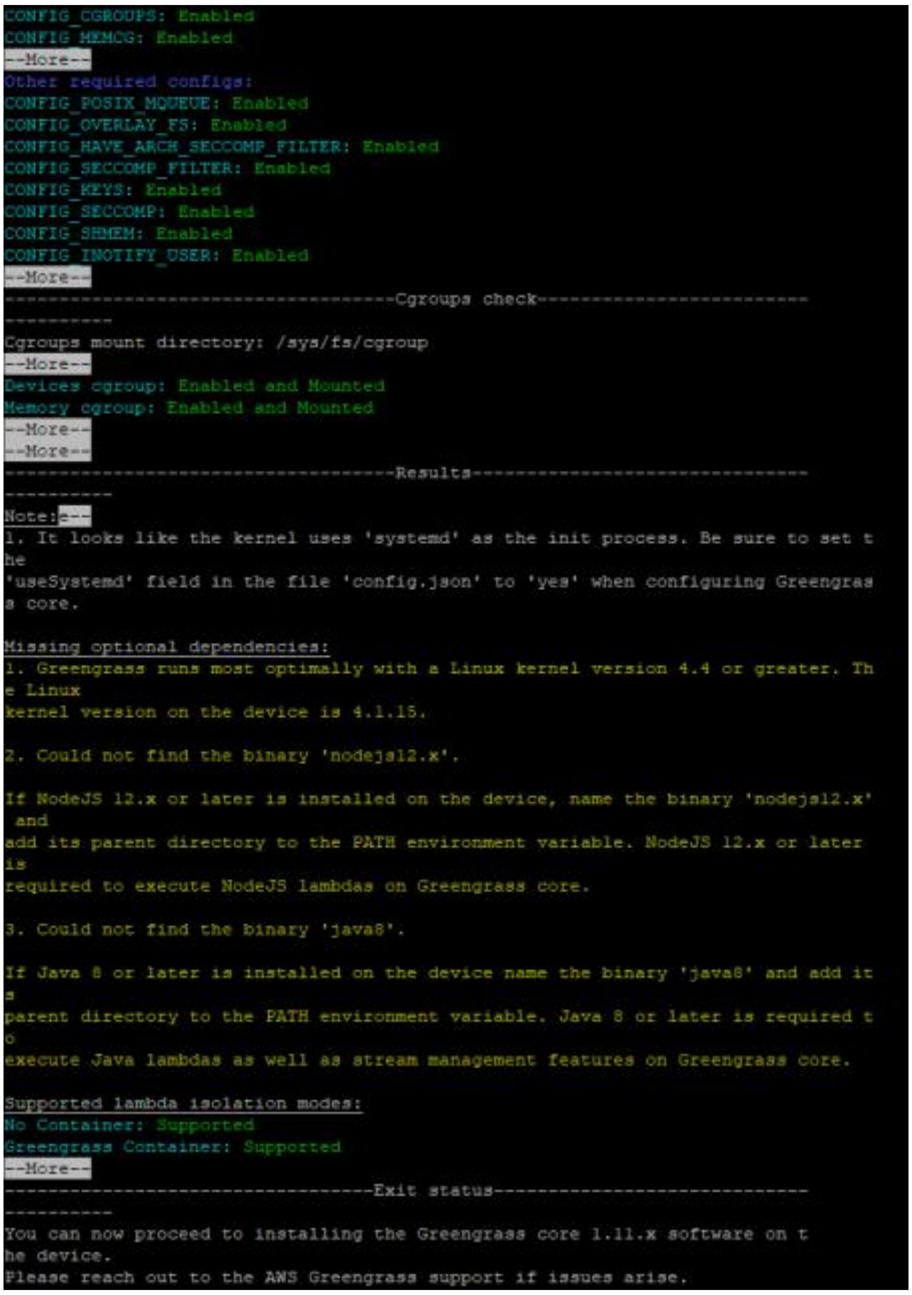
```
$ wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-
dependency-checker-GGCv1.11.x.zip $ unzip greengrass-dependency-checker-GGCv1.11.x.zip
```

```
$ cd greengrass-dependency-checker-GGCv1.11.x
$ sudo ./check_ggc_dependencies | more
```



```
===============
System configuration:
Kernel architecture: armv7l
Init process: /lib/systemd/systemd
Kernel version: 4.1.15
C library: Ubuntu GLIBC 2.27-3ubuntu1.2
C library version: 2.27
Directory /var/run: Present
/dev/stdin: Found
/dev/stdout: Found
/dev/stderr: Found
----------------------------Commands and software packages----------------
----------
Python 2.7 version: 2.7.17
Python 3.7 version: 3.7.10
Python 3.8 version: 3.8.0
NodeJS 12.x: Not found
Java 8: Not found
wget: Present
realpath: Present
tar: Present
readlink: Present
basename: Present
dirname: Present
pidof: Present
df: Present
grep: Present
umount: Present
mv: Present
gzip: Present
mkdir: Present
rm: Present
ln: Present
cat: Present
cut: Present
/bin/bash: Present
--More--
---------------------------------Platform security------------------------
----------
Hardlinks_protection: Enabled
Symlinks_protection: Enabled
--More--
-----------------------------------User and group-------------------------
----------
ggc_user: Present
ggc_group: Present
--More--
----------------(Optional) Greengrass container dependency check----------
--------
--More--
------------------------------Kernel configuration----------------------
----------
Kernel config file: /proc/config.gz
--More--
Namespace configs:
CONFIG_IPC_NS: Enabled
CONFIG_UTS_NS: Enabled
CONFIG_USER_NS: Enabled
CONFIG_PID_NS: Enabled
--More--
Cgroup configs:
CONFIG_CGROUP_DEVICE: Enabled
CONFIG_CGROUPS: Enabled
```

**Figure 23.** Installation of WISE-710 (part 1)



```
CONFIG_CGROUPS: Enabled
CONFIG_MEMCG: Enabled
--More--
Other required configs:
CONFIG_POSIX_MQUEUE: Enabled
CONFIG_OVERLAY_FS: Enabled
CONFIG_HAVE_ARCH_SECCOMP_FILTER: Enabled
CONFIG_SECCOMP_FILTER: Enabled
CONFIG_KEYS: Enabled
CONFIG_SECCOMP: Enabled
CONFIG_SHMEM: Enabled
CONFIG_INOTIFY_USER: Enabled
--More--
------------------------------------Cgroups check------------------------
----------
Cgroups mount directory: /sys/fs/cgroup
--More--
Devices cgroup: Enabled and Mounted
Memory cgroup: Enabled and Mounted
--More--
--More--
------------------------------------Results------------------------------
----------
Note:e--
1. It looks like the kernel uses 'systemd' as the init process. Be sure to set t
he
'useSystemd' field in the file 'config.json' to 'yes' when configuring Greengras
s core.

Missing optional dependencies:
1. Greengrass runs most optimally with a Linux kernel version 4.4 or greater. Th
e Linux
kernel version on the device is 4.1.15.

2. Could not find the binary 'nodejs12.x'.

If NodeJS 12.x or later is installed on the device, name the binary 'nodejs12.x'
 and
add its parent directory to the PATH environment variable. NodeJS 12.x or later
is
required to execute NodeJS lambdas on Greengrass core.

3. Could not find the binary 'java8'.

If Java 8 or later is installed on the device name the binary 'java8' and add it
s
parent directory to the PATH environment variable. Java 8 or later is required t
o
execute Java lambdas as well as stream management features on Greengrass core.

Supported lambda isolation modes:
No Container: Supported
Greengrass Container: Supported
--More--
----------------------------------Exit status-----------------------------
----------
You can now proceed to installing the Greengrass core 1.11.x software on t
he device.
Please reach out to the AWS Greengrass support if issues arise.
```

**Figure 24.** Installation of WISE-710 (part 2)

At the end of the check it informs us that there are optional dependencies that have not been found. found. It informs us that greengrass runs most optimally on a kernel version 4.4 or higher, which in our case is 4.1.15. kernel 4.4 or higher, which in our case is 4.1.15. In addition, he informs us that he has not found neither NodeJS 12.x nor Java 8, but these two are only for compatibility with the development of Lambda function development in those languages, which for this tutorial does not affect us. It is important to have Python version 3.7 installed, since it is used during the installation of the software and later on in of the software and later in Lambda function examples.

## 3.4 History

| Version | Release date | New features |
|---|---|---|
| V0.1 | 7/11/2022 | Instalation of Open62541 |
| V0.2 | 10/11/2022 | Manual of Open62541 |
| V0.3 | 21/11/2022 | WISE 710 configuration |
| V0.4 | 19/12/2022 | Images and their caption updates |
| V1.0 | 30/12/2022 | Final version |
| | | |
| | | |

**Table 1.** Versions' History

# 4. Conclusions

The i4Q **IIoT Security Handler** (i4Q$^{SH}$) aims to provide trust across the ICS architecture using a hardware secure module as trust anchor point. Once the trust is distributed, the software enables the mechanisms to expose cryptography operations that other i4Q Solutions can consume, adjusting security and safety policies at different levels to ensure trustability and privacy of data by means of the usage of a PKI infrastructure.

Installation and configuration guidelines for a PKI solution are described to facilitate the onsite inclusion and integration in the i4Q ICS ecosystem.

Additionally, own certificates have been used to perform secure operations in OPC-UA. For this purpose, the certificates have been generated by Lamassu and used to perform the operations.

## References

[1] i4Q IIoT Security Handler. [Online]. Available: https://i4q.upv.es/6_i4Q_SH/index.html#

[2] O. Andreeva et al., “Industrial control systems vulnerabilities statistics”, Kaspersky Lab, Report, 2016 [Online]. Available: https://www.researchgate.net/profile/Sergey_Gordeychik/publication/337732465_INDUSTRIAL_CONTROL_SYSTEMS_VULNERABILITIES_STATISTICS/links/5de7842e92851c8364600e7e/INDUSTRIAL-CONTROL-SYSTEMS-VULNERABILITIES-STATISTICS.pdf

[3] https://www.comparitech.com/blog/information-security/cybersecurity-vulnerability-statistics.

[4] M. Kravchik and A. Shabtai, “Detecting Cyber Attacks in Industrial Control Systems Using Convolutional Neural Networks”, Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy. 2018 [Online]. Available: http://dx.doi.org/10.1145/3264888.3264896

[5] Keith Stouffer (NIST), Suzanne Lightman (NIST), Victoria Pillitteri (NIST), Marshall Abrams (MITRE), Adam Hahn (WSU), “Guide to Industrial Control Systems (ICS) Security” 2015 [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-82/rev-2/final

[6] “Communication network dependencies for ICS/SCADA Systems”, 19-Dec-2016. [Online]. Available: https://www.enisa.europa.eu/publications/ics-scada-dependencies.

[7] S. Qayyum, S. Ashraf, M. Shafique, and S. Waheed, “Hardware devices security, their vulnerabilities and solutions”, in 2018 15th International Bhurban Conference on Applied Sciences and Technology (IBCAST), 2018, pp. 439–445, doi: 10.1109/IBCAST.2018.8312261 [Online]. Available: http://dx.doi.org/10.1109/IBCAST.2018.8312261

[8] KeyFactor. What is a PKI certificate? [Online]. Available:

https://www.primekey.com/wiki/what-is-a-pki-certificate/

[9] https://en.wikipedia.org/wiki/OPC_Unified_Architecture#UA_security

[10] j. Aro, H. Tahvanainen, P.P. Oy (2015). OPC UA Enables Secure Data Transfer and System Integrations in Private and Public Networks [J]. In *Automaatio XXI seminar 17.-18.3. 2015*.

[11] https://reference.opcfoundation.org/v104/Core/docs/Part2/8/

[12] https://reference.opcfoundation.org/v104/Core/docs/Part2/8.1.3/

[13] https://www.smartcard-hsm.com/features.html#usbstick

[14] EJBCA® Community Open Source PKI Software. https://www.ejbca.org/

[15] https://doc.primekey.com/ejbca743/ejbca-installation

[16] Lamassu. https://www.lamassu.io

[17] WISE-710. https://www.advantech.com/en/products/9a0cc561-8fc2-4e22-969c-9df90a3952b5/wise-710/mod_27f8e483-31d2-44c4-b8d7-7ad4e1fad21b

## Appendix I

i4Q **Solution name** (i4Q$^{SH}$) web documentation can be accessed online at: http://i4q.upv.es/6_i4Q_SH/index.html