



D3.6 – i4Q IIoT Security Handler

WP3 – BUILD: Manufacturing
Data Quality



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 | D3.6 – i4Q IIoT Security Handler | | | |
| WORK PACKAGE | WP3 – BUILD: Manufacturing Data Quality | | | |
| DATE OF DELIVERY | CONTRACTUAL | June 2022 | ACTUAL | June 2022 |
| NATURE | Report | DISSEMINATION LEVEL | Public | |
| LEAD BENEFICIARY | IKERLAN | | | |
| RESPONSIBLE AUTHOR | Aitor Uribarren (IKER) | | | |
| 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. A second version will be provided namely “D3.14 i4Q IIoT Security Handler v2”. | | | |
| EXTERNAL ANNEXES/SUPPORTING DOCUMENTS | None | | | |
| READING NOTES | None | | | |
| ABSTRACT | <p>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.</p> <p>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.</p> | | | |



Document History

| VERSION | ISSUE DATE | STAGE | DESCRIPTION | CONTRIBUTOR |
|---------|-------------|-----------------------|--|-------------|
| 0.1 | 12-May-2022 | ToC | ToC created and sent for review | IKERLAN |
| 0.2 | 10-Jun-2022 | Working Version | 1 st input to all sections | IKERLAN |
| 0.3 | 17-Jun-2022 | 1 st Draft | First draft sent for internal review | IKERLAN |
| 0.4 | 20-Jun-2022 | Internal review | Internal review | FBA, FIDIA |
| 0.5 | 24-Jun-2022 | 2 nd Draft | Addressing the comments from the internal review. Updated draft sent to the coordinator. | IKERLAN |
| 1.0 | 30-Jun-2022 | Final doc | Final quality check and issue of final document | CERTH |

Disclaimer

Any dissemination of results reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains.

Copyright message

© i4Q Consortium, 2022

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both. Reproduction is authorised provided the source is acknowledged.



TABLE OF CONTENTS

| | |
|---|-----------|
| Executive summary | 7 |
| Document structure | 8 |
| 1. General Description | 9 |
| 1.1 Overview | 9 |
| 1.2 Features..... | 9 |
| 1.2.1 Authentication..... | 10 |
| 1.2.2 Privacy..... | 10 |
| 1.2.3 Integrity..... | 10 |
| 1.2.4 Non-repudiation | 11 |
| 2. Technical Specifications | 12 |
| 2.1 Overview | 12 |
| 2.2 Certificate Life Cycle Management | 12 |
| 2.2.1 Certificate Handler..... | 12 |
| 2.2.2 Certificate Issuer..... | 12 |
| 2.2.3 Certificate Status Manager..... | 13 |
| 2.2.3.1 Status Checking | 13 |
| 2.2.3.2 Certificates Renewal | 13 |
| 2.2.3.3 Certificate Revocation | 14 |
| 2.2.4 Certificate Holder | 14 |
| 2.3 OPC-UA..... | 15 |
| 2.3.1 Connection Security | 15 |
| 2.3.2 Security Configuration..... | 16 |
| 2.3.3 User Authentication..... | 16 |
| 2.3.4 Certificate handling..... | 16 |
| 2.3.5 Certificate Management..... | 19 |
| 3. Implementation Status | 21 |
| 3.1 Current implementation..... | 21 |
| 3.1.1 Virtualbox | 21 |
| 3.1.2 HSM | 21 |
| 3.1.3 EJBCA..... | 22 |
| 3.1.3.1 CA creation..... | 23 |



| | | |
|-----------|---------------------------------|-----------|
| 3.1.3.2 | User Certificate Creation | 23 |
| 3.1.4 | Connection API's..... | 24 |
| 3.1.4.1 | API-WS Interface | 24 |
| 3.1.4.2 | API-Rest Interface | 25 |
| 3.2 | Next developments..... | 25 |
| 3.3 | History..... | 25 |
| 4. | Conclusions..... | 26 |
| | References | 27 |
| | Appendix I..... | 28 |

LIST OF FIGURES

| | | |
|-------------------|---|----|
| Figure 1. | Generic certificate issuance view..... | 12 |
| Figure 2. | New certificate creation flow | 13 |
| Figure 3. | Certificate status checker flow | 13 |
| Figure 4. | Certificates renew flow | 14 |
| Figure 5. | Certificates revocation flow..... | 14 |
| Figure 6. | Certificate deal with..... | 17 |
| Figure 7. | CA Certificate handling | 19 |
| Figure 8. | CardContact USB-Token HSM | 21 |
| Figure 9. | Main view of Ejbca PKI manager | 23 |
| Figure 10. | 'i4Q One CA' certificate information..... | 23 |
| Figure 11. | User certificate information | 24 |

LIST OF TABLES

| | | |
|-----------------|--------------|----|
| Table 1. | History..... | 25 |
|-----------------|--------------|----|



ABBREVIATIONS/ACRONYMS

| | |
|----------------|---|
| API | Application Programming Interfaces |
| CA | Certification Authority |
| CIA | Confidentiality, Integrity and Availability |
| COTS | Commercial Off-The-Shelf |
| CRL | Certificate Revocation List |
| CSR | Certificate Signing Request |
| EST | Enrollment over Secure Transport |
| HSM | Hardware Security Module |
| HTTP | HyperText Transfer Protocol |
| ICS | Industrial Control Systems |
| IIoT | Industrial Internet of Things |
| IP | Internet Protocol |
| IT | Information Technology |
| JAX-WS | Java API for XML Web Services |
| JSON | JavaScript Object Notation |
| OCSP | Online Certificate Status Protocol |
| OPC | Open Platform Communications |
| OPC-UA | OPC Unified Architecture |
| OpenSSL | Open Secure Sockets Layer |
| OS | Operational System |
| OT | Operational Technology |
| PHP | Hypertext Preprocessor |
| PKI | Public Key Infrastructure |
| RA | Registration Authority |
| REST | Representational State Transfer |
| SDK | Software Development Kit |
| SOAP | Simple Object Access protocol |
| TCP | Fieldbus communication |
| TLS | Transport Layer Security |
| URI | Uniform Resource Identifier |



| | |
|-------------|-----------------------------------|
| URL | Uniform Resource Locator |
| VA | Validation Authority |
| WSDL | Web Services Description Language |



Executive summary

This deliverable provides the specification and design of the **i4Q Security handler** (i4Q^{SH}) to provide trust in Industrial control system (ICS) by means of a Public Key Infrastructure, that is being developed by IKERLAN in i4Q.

An ICS is a collection of control systems that work together to achieve a certain industrial goal. The integration of operational technology (OT) and information technology (IT) in ICS has boosted operational and financial efficiency, but it has also opened the door to greater risks, including system outages and espionage. Today a control system is vulnerable to cyber-attacks in many ways and control system engineers need to be well aware of subjects such as ICS security and SCADA security. This document presents a proposal to use X.509 certificates to provide trust in i4Q ICS ecosystem.



Document structure

Section 1: Contains a brief introduction of the **i4Q IIoT Security Handler**, 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**, 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**, 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** 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 [1]. Almost 22,000 vulnerabilities were published in 2021 [2]. Organizations use OT to manage physical industrial equipment, assets, processes, and events using a combination of technology, software, and hardware. In these OT environments, Industrial Control Systems (ICS) are widely used to monitor and control industrial processes, such as those in the manufacturing, transportation, and pharmaceutical sectors, as well as critical infrastructures like electricity, water treatment plants, and oil and gas refineries [3].

Traditionally, ICS is designed to function on customized hardware and/or software that is physically separated from the outside world, this is no longer the case. ICSs have used a variety of information technology (IT) solutions, such as commercial off-the-shelf (COTS) elements, remotely enabled connections, standardized OS, and cloud-based solutions. This development, combined with the usage of insecure industrial protocols like DNP3, OPC, and MODBUS, raises the risk of security flaws and incidents [4][5]. 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 [6].

Cryptography is one of the most important strategies used by organizations to protect the systems that store their most precious asset or information, whether it is in transit or at rest. Data can include client, worker, intellectual property, ICS business policies, and any other classified information. As a result, cryptography is critical infrastructure, as cryptographic solutions are becoming increasingly reliant on the security of sensitive data. It can help protect secret information and sensitive material, and also increase client-server communication security. In other words, even if an unwanted person or entity has access to your data, they will be unable to read it.

Public Key Infrastructure (PKI) is used to create a trust chain that allows a person, service, machine, or application to be authorized, a secure connection to be formed, or the provenance of software or documents to be validated. This is accomplished through certificates, which a PKI creates, manages, and distributes while also having the ability to revoke. The public key in a certificate must be kept safe and secret, and the private key must be kept safe and hidden as well. It'll have to be stored in a hardware security module (HSM).

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 by the X.509 standard.



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 than 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

In information security, the three most critical concepts are confidentiality, integrity, and availability. The relevance of the CIA triad security model speaks for itself, with each letter expressing a core premise in cybersecurity.

1.2.1 Authentication

Identifying who you are dealing with is one of the most difficult parts of doing business on the Internet today.

The PKI allows trading partners to be identified online through trustworthy authorities that are in charge of issuing digital certificates and providing procedures for identifying individuals who hold those certificates on behalf of a company.

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?

The technique for protecting information is provided by digital certificates. Messages can be encrypted to reduce the danger of them being intercepted or read by someone other than the intended receiver while in transit. This creates a digital version of a registered letter sent through the mail.

PKI can be used to make messages more private.

1.2.3 Integrity

When conducting business over the Internet, a company needs to know that any transaction conducted or information provided will not be altered or interfered with during transmission.



As a fundamental component, the PKI ensures transaction security. The recipient of a communication can use PKI to verify that the message is still the same as it was when it was transmitted.

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.

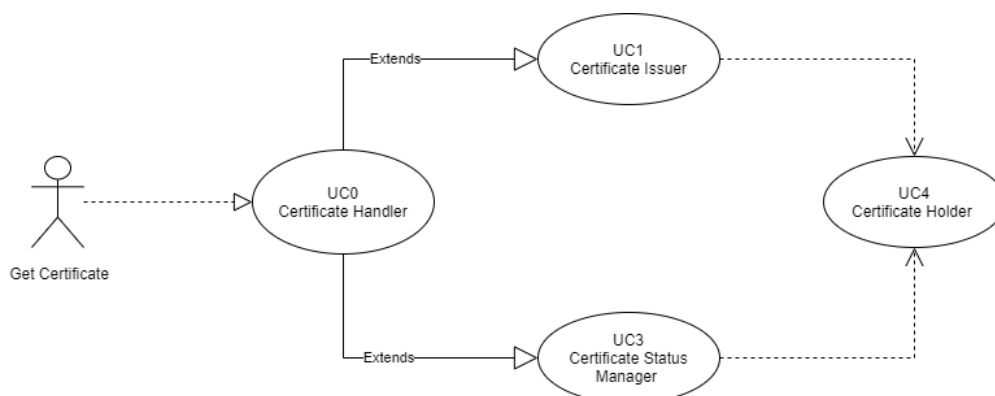


Figure 1. Generic certificate issuance view

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.

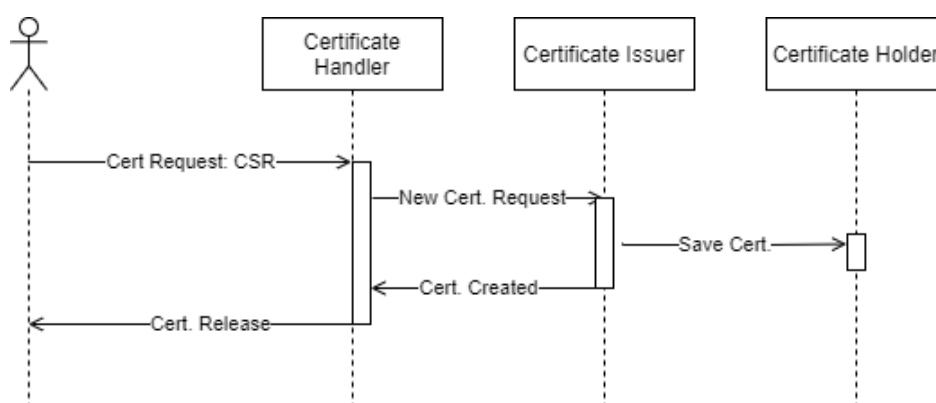


Figure 2. New certificate creation flow

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 inconsistency 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 (OCSP) request.

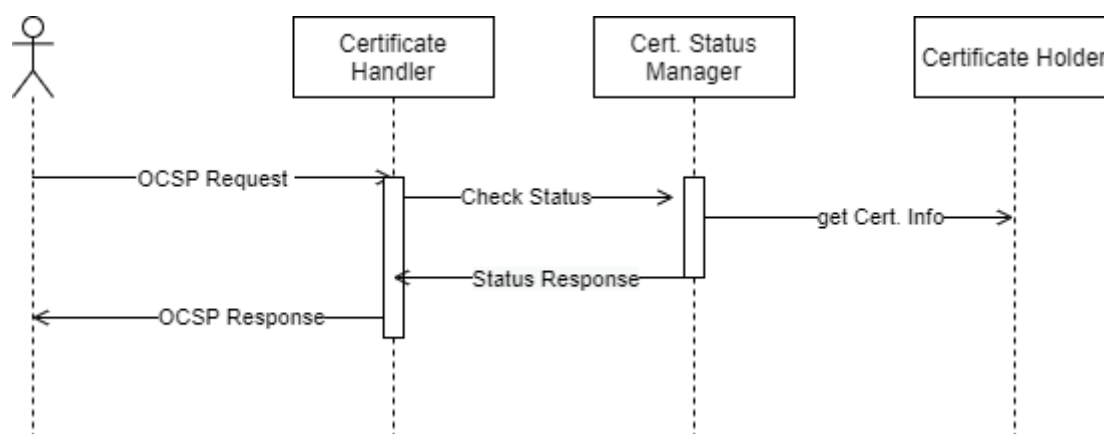


Figure 3. Certificate status checker flow

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.

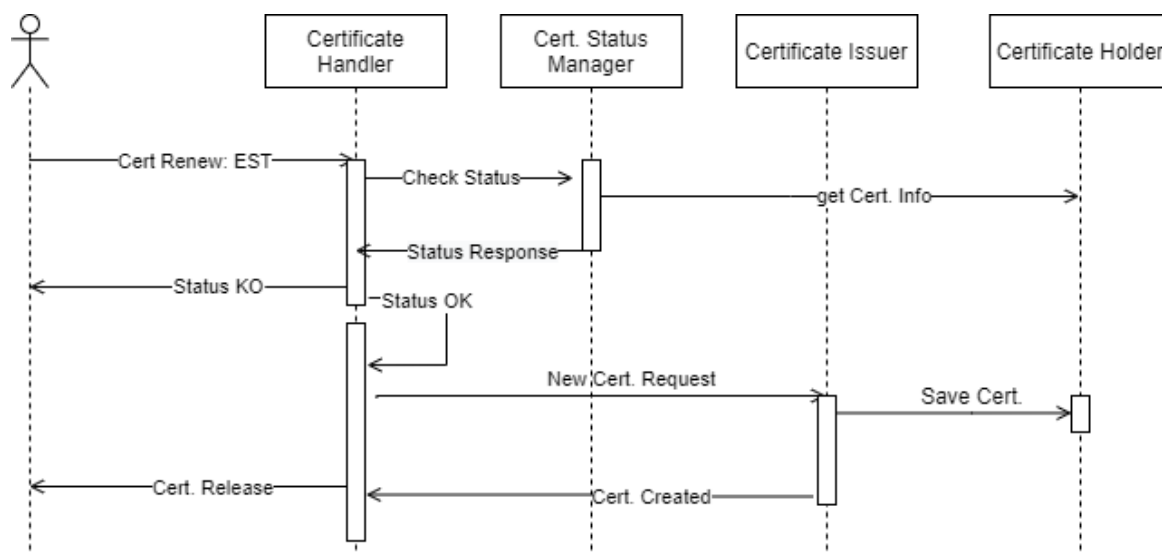


Figure 4. Certificates renew flow

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.

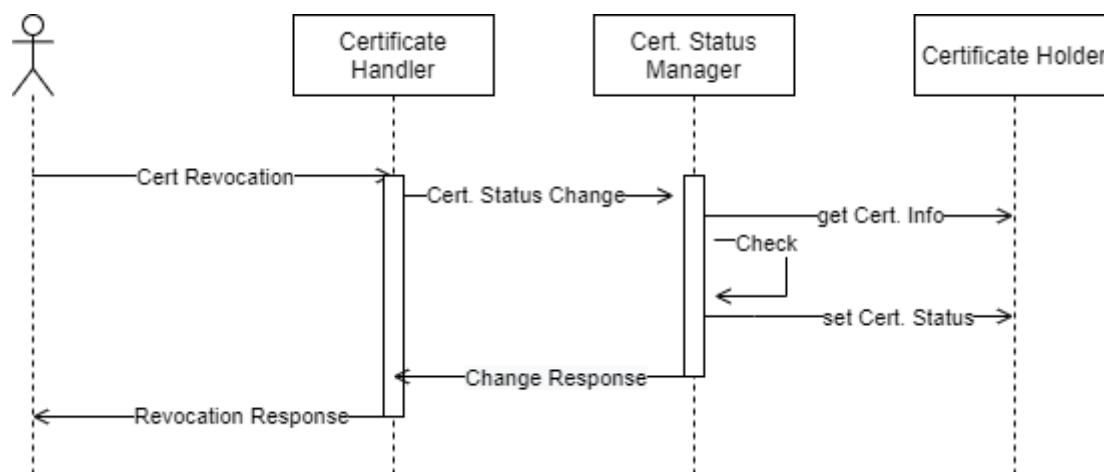


Figure 5. Certificates revocation flow

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 [7]. It is a protocol for transferring data in an object-oriented fashion rather than as discrete data points. This makes your plant floor data more accessible by allowing you to reuse information stored in a shared object. OPC-UA also has a service-oriented model, which improves security and interoperability with other platforms.

OPC UA's transport protocol provides a stable and dependable communication infrastructure with techniques for handling lost messages, heartbeats, and failover, among other things. OPC UA is a high-performance data exchange technology that uses binary-encoded data. Commercial SDK are available for C, C++, Java, and .NET. On the other hand, open-source stacks are also available at least for C, C++, Java, Javascript(node) and Python.

Security is built into OPC UA and has been a design goal from the beginning of the development. OPC UA offers different technology mappings basing either on TCP / IP or on SOAP based web services. A secure channel is used on top of the transport layer to protect messages from unwanted alterations and eavesdropping by using encryption and digital signatures. Furthermore, this layer uses digital certificate-based authentication procedures to authenticate and authorize particular instances of OPC UA applications. This allows administrators to establish a fine-grained access control in critical infrastructures such as production facilities and power plants. A session, that's represents a connection between a client application and a server, it is used for exchanging payload representing plant information (e.g., valve status, temperature, status of level indicators), settings, and commands. Session messages are therefore secured by the secure channel mentioned above. Users intending to establish sessions on the client side need to be authenticated and authorized by OPC UA servers. The specification allows three different mechanisms: Username/password combinations, digital certificates, and WS compliant user tokens.

The OPC UA security is based on three layers: an application layer that is on top of a second communication layer that's relays on the third transport layer. Data transfer is always done over TCP. TLS is used to encrypt transport layer traffic if the HTTPS protocol is utilized. The Communication Layer consists of a Secure Channel which can optionally perform message signing and encryption. This layer maintains the confidentiality and integrity of the exchanged messages when a secure communication policy is selected. It also allows applications to communicate with each other to be authenticated. The session, which is used to authenticate and authorize users, is part of the Application Layer. Because all activities are done in a session, unauthenticated users cannot view or edit data in the target system. The session always connects using a secure channel, which is renewed on a regular basis.

2.3.1 Connection Security

The establishment of OPC UA connection ensures the authentication and authorization of applications using standard security techniques. An Application Instance Certificate is a regular X.509v3 certificate with certain extra fields for enhanced OPC UA validation for each application instance. The application developer is responsible for deciding which certificate store the UA application is tied to. It is feasible, for example, to use an Active Directory's PKI. APIs for UA are



available in a variety of computer languages. When programs build the Secure Channel between them, the appropriate RSA public and private keys are utilized to complete a secure handshake. In the handshake, that in reality is an Open Secure Channel service message, both programs will perform the authentication of the other side. A symmetric encryption key is also exchanged by the applications during the handshake, and this key is then used to encrypt all future Secure Channel messages. AES-128 or AES-256 are used for symmetric encryption.

2.3.2 Security Configuration

Between each connection, OPC UA allows for a free option of the security mode to be employed. None, Sign, and SignAndEncrypt are the three Message Security Modes defined by the OPC UA specification. These determine the amount of security that each message receives. Basic128Rsa15, Basic256, and Basic256Sha256 are some of the other Security Policies defined. As security requirements grow in the future, new policies can be developed, and old ones can be phased out. These define the Application Instance Certificates' enabled key sizes, as well as the signature algorithm and symmetric encryption algorithm that are used.

All OPC UA servers that implement the Standard Server Profile must provide signing and encryption capabilities with at least the Basic128Rsa15 policy level. Applications implementing only the Micro or Nano Embedded Server Profile don't need to provide security capabilities at all. These policies are available can be configured by the server administrator. The client application always makes the selection, which security mode is used for each connection. This makes sure that security is always available and can be easily switched on as necessary.

2.3.3 User Authentication

The authentication of users takes place at the session level. Alternative authentication mechanisms defined by OPC UA include Anonymous, Username and Password, X.509 certificates, and external systems for user authentication like Kerberos via outer tokens. The server applications must again support different alternatives depending on the Server Profile that they implement. And the client application again selects the authentication type to use for the connection from the alternatives implemented and configured for the server.

2.3.4 Certificate handling

From the security perspective [8], 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.

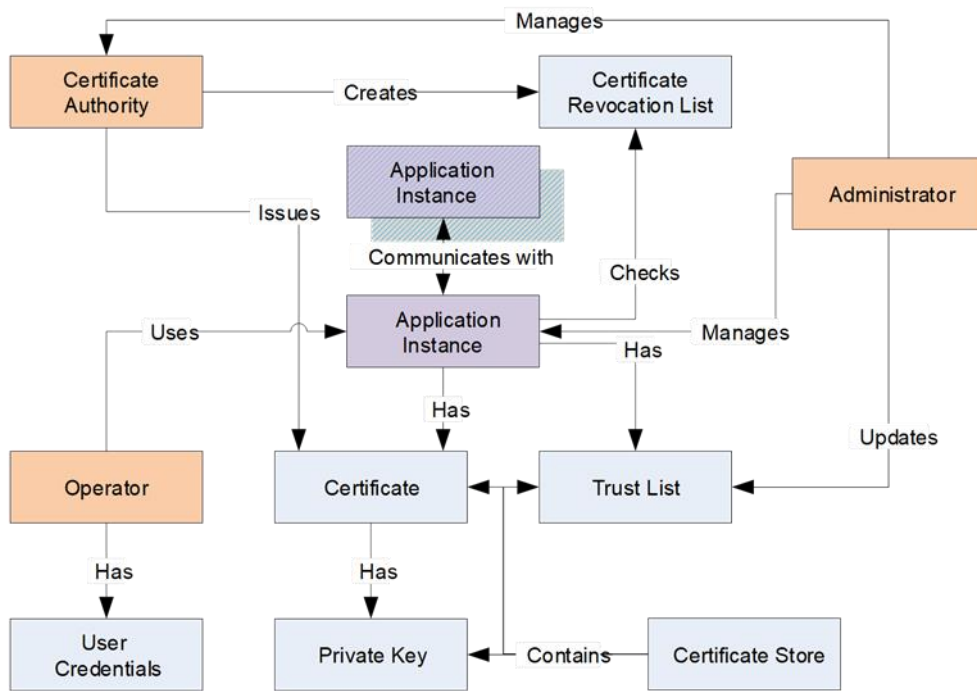


Figure 6. Certificate deal with

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 the security of OPC UA is based on X.509 certificates, the main concern in practice will be the certificate management. OPC UA specification does not enforce any specific strategy for managing the certificates.

All OPC UA applications maintain certificates in their own Trust Store. All encountered Application Instance Certificates from the connecting applications are classified either as trusted or rejected. This selection can be done on a certificate basis, which is usable, when the number of connections is small. The applications typically use self-signed certificates by default, which can only be trusted individually. In a standard security environment, trust is based on CA, which sign certificates and maintain CRL. CA helps to maintain the trust chain between applications: all certificates signed by a trusted CA can be trusted automatically, until they are revoked.

The establishment of Public Keys in Trust Lists can turn into burdensome in systems with several Servers and Clients. The adoption of a proper CA in these situations can substantially ease the installation and configuration challenges. Additional services provided by the CA include certificate expiration management and CRL management. **Figure 7** gives an example of what this activity entails [9].

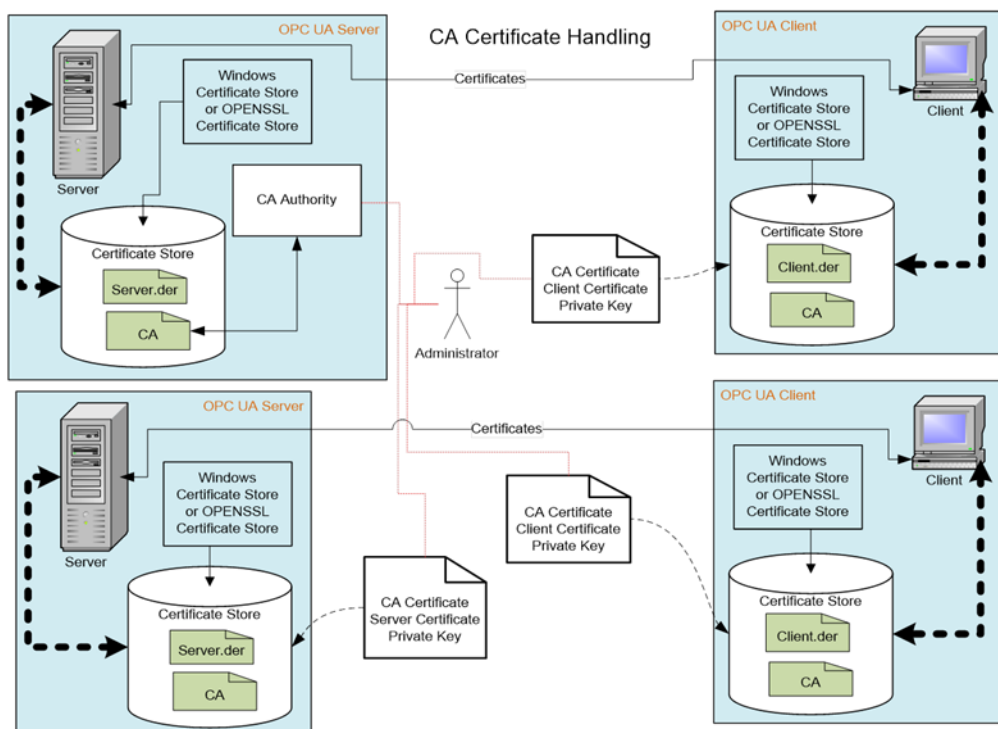


Figure 7. CA Certificate handling

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 [10] shown below in **Figure 8**.



Figure 8. CardContact USB-Token HSM

To install the procedure described in the manufacturer's site has been followed <https://www.smartcard-hsm.com/support.html>. Once the drivers have been installed, it can be checked that the device is present and ready running different commands:

```
[ikerlan@localhost ~]$ pkcs11-tool -l
```

```
Cryptoki version 2.20
```



```
Manufacturer: OpenSC Project
```

```
Library: OpenSC smartcard framework (ver 0.19)
```

```
Using slot 0 with a present token (0x0)
```

```
[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 [11] community edition has been installed following the official installation guides[12]. 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 **Figure 9** a view is shown.

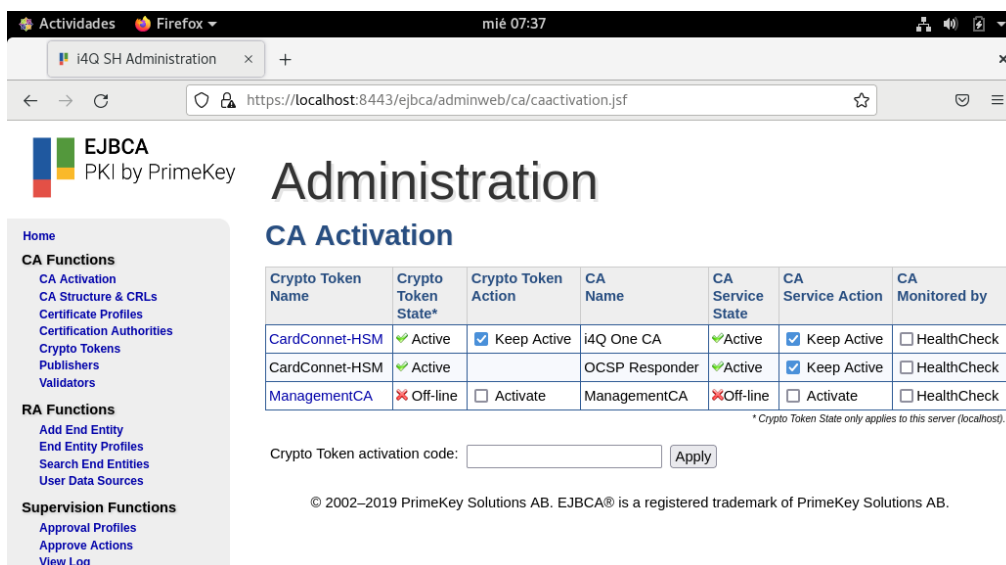


Figure 9. Main view of Ejbca PKI manager

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.

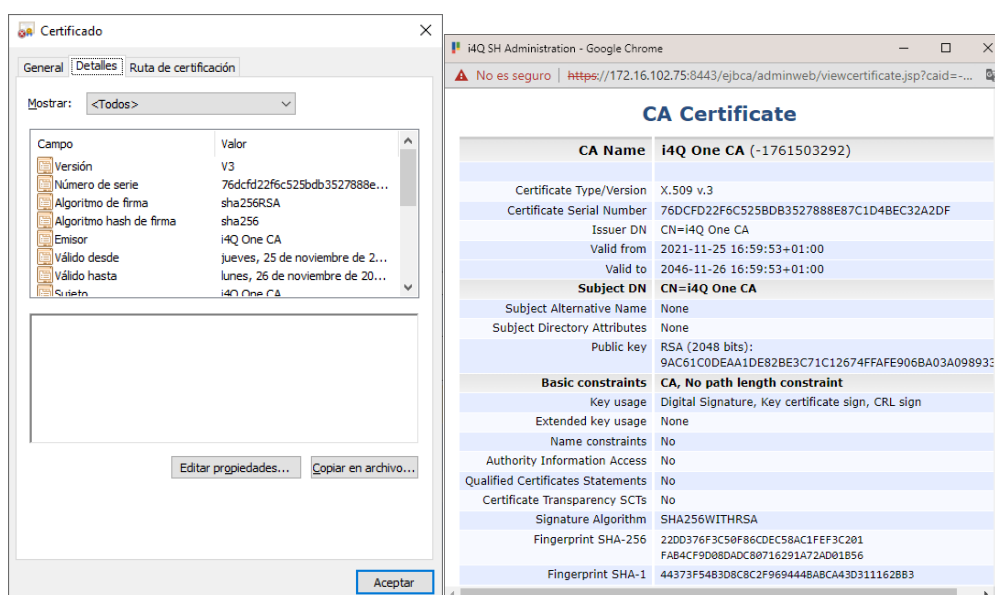


Figure 10. ‘i4Q One CA’ certificate information

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.

View Certificates

| | |
|---|--|
| Username | aitor.uribarren |
| Certificate number | 1 of 3 |
| <input type="button" value=" < View Older"/> | |
| 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 |
| <input type="button" value="Republish"/> | <input type="text" value="Unspecified"/> <input type="button" value="Revoke"/> |

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 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.

3.1.4.1 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.



3.1.4.2 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 Next developments

This document described the first approach to the envisaged **i4Q** PKI architecture.

It is expected to add and complete it, which will serve as an example of integration, with the OP-UA Security scheme as base of the integration in ICS. A use case for OPC-UA will be explained for the last release.

Integration with other **i4Q** solution and pilots is also planned to show and describe in here for the last release. We identified collaboration opportunities between the **i4Q** Trusted Network (**i4Q^{TN}**), **i4Q** Block Chain (**i4Q^{BC}**) module and **i4Q** Message Broker (**i4Q^{MB}**) service, but they are in a very early stage of integration, therefore the evolution of the integration will be described in the next release.

3.3 History

| Version | Release date | New features |
|---------|--------------|--|
| V0.0.1 | 10/02/2022 | EJBCA PKI manager installation in CentOS virtual machine |
| V0.0.2 | 14/02/2022 | HSM cardContact installation and configuration |
| V0.0.3 | 14/02/2022 | HSM cardContact integrated and added in EJBCA PKI |
| V0.0.4 | 07/03/2022 | Creation of the first x509 certificate in the PKI. |
| V0.0.5 | 22/03/2022 | Creation of crypto tokens in PKI using CardContact HSM |
| V0.0.6 | 29/03/2022 | Creation of CA in PKI |
| V0.0.7 | 29/03/2022 | Creation of the first x509 certificate in the PKI. |
| V0.0.8 | 03/05/2022 | Compact installation guide. Virtualbox to distribute. |
| | | |

Table 1. 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.



References

- [1] 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
- [2] <https://www.comparitech.com/blog/information-security/cybersecurity-vulnerability-statistics>.
- [3] 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>
- [4] 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>
- [5] “Communication network dependencies for ICS/SCADA Systems”, 19-Dec-2016. [Online]. Available: <https://www.enisa.europa.eu/publications/ics-scada-dependencies>.
- [6] 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>
- [7] https://en.wikipedia.org/wiki/OPC_Unified_Architecture#UA_security
- [8] <https://reference.opcfoundation.org/v104/Core/docs/Part2/8/>
- [9] <https://reference.opcfoundation.org/v104/Core/docs/Part2/8.1.3/>
- [10] <https://www.smartcard-hsm.com/features.html#usbstick>
- [11] EJBICA® Community Open Source PKI Software. <https://www.ejbca.org/>
- [12] <https://doc.primekey.com/ejbca743/ejbca-installation>



Appendix I

The PDF version of the **i4Q IloT Security Handler** web documentation can be accessed online at: http://i4q.upv.es/6_i4Q_SH/index.html

i4Q Solution IloT Security Handler (i4QSH)

General Description

This document presents general description and technical application **i4Q IloT Security Handler (i4Q^{SH})** which is a proposal of an implementation of a Public Key Infrastructure (PKI) to provide trust in the **i4Q Industrial Control System (ICS)** ecosystem.

An ICS is a collection of control systems that work together to achieve a certain industrial goal. The integration of operational technology (OT) and information technology (IT) in ICS has boosted operational and financial efficiency, but it has also opened the door to greater risks, including system outages and espionage. Today a control system is vulnerable to cyber-attacks in many ways and control system engineers need to be well aware of subjects such as ICS security and SCADA security. This document presents a proposal to use X.509 certificates to provide trust in **i4Q ICS** ecosystem. This document presents a proposal to use X.509 certificates to provide trust in **i4Q ICS** ecosystem.

PKI is used to create a trust chain that allows a person, service, machine, or application to be authorized, a secure connection to be formed, or the provenance of software or documents to be validated. This is accomplished through certificates, which a PKI creates, manages, and distributes while also having the ability to revoke. The public key in a certificate must be kept safe and secret, and the private key must be kept safe and hidden as well. It'll have to be stored in a hardware security module (HSM).

Features

The main aspects considered in **i4QSH** are as follows:

1. **Authentication:** The PKI allows trading partners to be identified online through trustworthy authorities that are in charge of issuing digital certificates and providing procedures for identifying individuals who hold those certificates on behalf of a company.
2. **Privacy:** The technique for protecting information is provided by digital certificates. Messages can be encrypted to reduce the danger of them being intercepted or read by someone other than the intended receiver while in transit.
3. **Integrity:** When conducting business over the Internet, a company needs to know that any transaction conducted or information provided will not be altered or interfered with during transmission. As a fundamental component, the PKI ensures transaction security. The recipient of a communication can use PKI to verify that the message is still the same as it was when it was transmitted.
4. **Non-repudiation:** As a fundamental component, the PKI ensures transaction security. The recipient of a communication can use PKI to verify that the message is still the same as it was when it was transmitted.



ScreenShots

Getting HSM information from console

```
[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
pin min/max      : 6/15
[ikerlan@localhost ~]$ pkcs11-tool -I
Cryptoki version 2.20
Manufacturer      OpenSC Project
Library           OpenSC smartcard framework (ver 0.19)
Using slot 0 with a present token (0x0)
[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
pin min/max      : 6/15
[ikerlan@localhost ~]$
```

Bootstrapping Ejbca PKI instace from console

```
root@localhost:~
Archivo Editar Ver Buscar Terminal Ayuda
[root@localhost ~]# /opt/wildfly-10.1.0/bin/standalone.sh -b=0.0.0.0 -bmanagement=0.0.0.0
=====
JBoss Bootstrap Environment

JBOSS_HOME: /opt/wildfly-10.1.0

JAVA: /usr/lib/jvm/jre/bin/java

JAVA_OPTS: -server -Xms64m -Xmx512m -XX:MetaspaceSize=96M -XX:MaxMetaspaceSize=256m -Djava.net.prefer
IPv4Stack=true -Djboss.modules.system.pkgs=org.jboss.byteman -Djava.awt.headless=true
=====

11:43:59,186 INFO [org.jboss.modules] (main) JBoss Modules version 1.5.2.Final
11:44:00,516 INFO [org.jboss.msc] (main) JBoss MSC version 1.2.6.Final
11:44:00,672 INFO [org.jboss.as] (MSC service thread 1-2) WFLYSRV0049: WildFly Full 10.1.0.Final (WildF
ly Core 2.2.0.Final) starting
```



View of Ejbca and HSM configured to use

EJBCA
PKI by PrimeKey

Administration

CA Activation

| Crypto Token Name | Crypto Token State* | Crypto Token Action | CA Name | CA Service State | CA Service Action | CA Monitored by |
|-------------------|---------------------|---|----------------|------------------|---|--------------------------------------|
| CardConnet-HSM | ✔ Active | <input checked="" type="checkbox"/> Keep Active | i4Q One CA | ✔ Active | <input checked="" type="checkbox"/> Keep Active | <input type="checkbox"/> HealthCheck |
| CardConnet-HSM | ✔ Active | | OCSP Responder | ✔ Active | <input checked="" type="checkbox"/> Keep Active | <input type="checkbox"/> HealthCheck |
| ManagementCA | ✘ Off-line | <input type="checkbox"/> Activate | ManagementCA | ✘ Off-line | <input type="checkbox"/> Activate | <input type="checkbox"/> HealthCheck |

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

Crypto Token activation code:

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

View of an i4Q CA certificate


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): 9AC61C0DEAA1DE82BE3C71C12674FFAFE906BA03A098933... |
| 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 |
| Revoked | No |



Commercial Information

Authors

| Company | Website | Logo |
|---------|--|--|
| IKERLAN | www.ikerlan.es/en/ |  |

Associated i4Q Solutions

Required

Currently, it can operate without the need for another i4Q solution.

Optional

None. However, the i4Q^{SH} is expected to be used with almost any other i4Q solution in the pilots, where security is need providing SSL certificates to trust data and communications, for example in the i4Q^{DR} solution.

System Requirements

1. Windows or Linux powered PC
2. VMware or VirtualBox CentOS image
3. CardContact HSM.
 - CentOS requirements:
 - 4 GB Ram
 - 2 CPU
 - 64-bit operating system
 - Hardware virtualisation support

Additionally, it requires the following dependencies to be already installed:

1. *OpenSC*, required security library to interact with HSM.
:> `wget http://mirror.centos.org/centos/7/os/x86_64/Packages/opensc-0.19.0-3.el7.x86_64.rpm`
2. *EJBCA*, PKI software administrator.
:> `wget https://sourceforge.net/projects/ejbca/files/ejbca6/ejbca_6_15_2_6/ejbca_ce_6_15_2_6.zip`



API Specification

Since i4Q^{SH} is expected to be used or integrated with other i4Q solutions, the plan is to use Web Service API as a mechanism to interact with the i4Q^{SH}. This WS API offers the following endpoints:

<https://HOST-IP:8443/ejbca/ejbcaws/ejbcaws?wsdl>

Providing links to access to the required functions to interact with PKI.

```

▼<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:
xmlns:ns1="http://schemas.xmlsoap.org/soap/http" name="EjbcaWSService" targetNamespace="http://ws.protocol.core.e:
  ▼<wsdl:types>
    ▼<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://ws.protocol.core.ejbca.org/" attribut
      targetNamespace="http://ws.protocol.core.ejbca.org/">
        <xs:element name="addSubjectToRole" type="tns:addSubjectToRole"/>
        <xs:element name="addSubjectToRoleResponse" type="tns:addSubjectToRoleResponse"/>
        <xs:element name="caCertResponse" type="tns:caCertResponse"/>
        <xs:element name="caCertResponseForRollover" type="tns:caCertResponseForRollover"/>
        <xs:element name="caCertResponseForRolloverResponse" type="tns:caCertResponseForRolloverResponse"/>
        <xs:element name="caCertResponseResponse" type="tns:caCertResponseResponse"/>
        <xs:element name="caRenewCertRequest" type="tns:caRenewCertRequest"/>
        <xs:element name="caRenewCertRequestResponse" type="tns:caRenewCertRequestResponse"/>
        <xs:element name="certificateRequest" type="tns:certificateRequest"/>
        <xs:element name="certificateRequestResponse" type="tns:certificateRequestResponse"/>
        <xs:element name="checkRevokationStatus" type="tns:checkRevokationStatus"/>
        <xs:element name="checkRevokationStatusResponse" type="tns:checkRevokationStatusResponse"/>
        <xs:element name="createCA" type="tns:createCA"/>
        <xs:element name="createCAResponse" type="tns:createCAResponse"/>
        <xs:element name="createCRL" type="tns:createCRL"/>
        <xs:element name="createCRLResponse" type="tns:createCRLResponse"/>
        <xs:element name="createCryptoToken" type="tns:createCryptoToken"/>
        <xs:element name="createCryptoTokenResponse" type="tns:createCryptoTokenResponse"/>
        <xs:element name="crmfRequest" type="tns:crmfRequest"/>
        <xs:element name="crmfRequestResponse" type="tns:crmfRequestResponse"/>
        <xs:element name="customLog" type="tns:customLog"/>
        <xs:element name="customLogResponse" type="tns:customLogResponse"/>
        <xs:element name="cvcRequest" type="tns:cvcRequest"/>
        <xs:element name="cvcRequestResponse" type="tns:cvcRequestResponse"/>
        <xs:element name="deleteUserDataFromSource" type="tns:deleteUserDataFromSource"/>
        <xs:element name="deleteUserDataFromSourceResponse" type="tns:deleteUserDataFromSourceResponse"/>
        <xs:element name="editUser" type="tns:editUser"/>
        <xs:element name="editUserResponse" type="tns:editUserResponse"/>
        <xs:element name="existsHardToken" type="tns:existsHardToken"/>
        <xs:element name="existsHardTokenResponse" type="tns:existsHardTokenResponse"/>
        <xs:element name="fetchUserData" type="tns:fetchUserData"/>
        <xs:element name="fetchUserDataResponse" type="tns:fetchUserDataResponse"/>
        <xs:element name="findCerts" type="tns:findCerts"/>
        <xs:element name="findCertsResponse" type="tns:findCertsResponse"/>
        <xs:element name="findUser" type="tns:findUser"/>
        <xs:element name="findUserResponse" type="tns:findUserResponse"/>
        <xs:element name="genTokenCertificates" type="tns:genTokenCertificates"/>
        <xs:element name="genTokenCertificatesResponse" type="tns:genTokenCertificatesResponse"/>

```

WS-API functions provided by PKI

Installation Guidelines

| Resource | Location |
|------------------------|-------------------------|
| Last release (v.1.0.0) | Website |

Installation CardContact HSM

Latest installation information at:

<https://www.smartcard-hsm.com/opensource.html#starterkit>



Command to initial installation of HSM: sc-hsm-tool:

```
[root@localhost ~]# sc-hsm-tool -initialize -so-pin xxxxxx -pin yyyyyy
```

Installation EJBCA

Latest installation information at:

<https://www.ejbca.org/documentation/>

Need to adapt web.properties file in ejbca/conf to use CardContact HSM:

- cryptotoken.p11.lib.60.name=OpenSC
- cryptotoken.p11.lib.60.file=/usr/lib64/opensc-pkcs11.so

User Manual

How to use

The resulting solution is based on the external program Ejbca to configure and manage the PKI infrastructure. Latest guidelines in how to operate will be find at <https://download.primekey.com/docs>

WS-API

The SOAP API is compatible with all development languages that can handle SOAP messages. This includes programming languages such as Java, C# and PHP. Client stubs are generated from the WSDL file, which includes all information to use the WS API. The WSDL can be accessed from your installed EJBCA at <https://HOST-IP:8443/ejbca/ejbcaws/ejbcaws?wsdl>.