Bachelor's thesis

Information and Communications Technology

2024

Valtteri Äyräs

Authenticating MQTT Messages Using JWT

– Creating a PKI

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Information and Communications Technology

2024 | 47 pages

Valtteri Äyräs

Authenticating MQTT messages using JWT

Creating a PKI

In critical systems, cyber security is an important aspect. By enabling authentication within communication, we can ensure that entities we communicate with are who they claim to be. This can be implemented by creating a public key infrastructure that enables the creation of certificates that are used for authentication.

The remote pilotage project of Brighthouse Intelligence, on which the thesis is based on, is a proof of concept of piloting a cargo vessel from a remote location by using data stream from the vessel as support for decision making. In some use-cases, all data streams are not needed at the remote operation centers, therefore handling the data stream by turning off unnecessary streams can be beneficial for the network load. This data stream control commanding, however, must be conducted in a secure manner so that no malicious actors can tinker the data flow during the remote pilotage operation.

The purpose of this thesis was to create a public key infrastructure using Python cryptography libraries and Docker. As a result, the remote pilot can send authenticated commands to vessels from an interface of choice. This outcome can be used in later implementations of maritime communications, where the authenticity of messages is critical.

Keywords:

MQTT, JWT, PKI, authentication, X.509

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tieto- ja viestintätekniikka

2024 | 47 sivua

Valtteri Äyräs

MQTT viestien autentikointi käyttäen JWT:tä

PKI: n luominen

Kriittisissä järjestelmissä kyberturvallisuus on tärkeä aspekti. Ottamalla autentikoinnin käyttöön viestinnässä voimme varmistaa, että entiteetit, joiden kanssa kommunikoimme, ovat niitä, joita väittävät olevansa. Tämä voidaan toteuttaa luomalla julkisen avaimen infrastruktuuri, joka mahdollistaa autentikointiin käytettävien sertifikaattien luomisen.

Brighthouse Intelligencen etäluotsiprojekti, johon opinnäytetyö perustuu, on konsepti rahtialuksen luotsaamisesta etäpaikalta käyttämällä aluksesta tulevaa datavirtaa päätöksenteon tueksi. Joissakin käyttötapauksissa kaikkia tietovirtoja ei tarvita etäoperaatiokeskuksissa, joten tietovirran säätely sammuttamalla tarpeettomat datavirrat voi olla hyödyllistä verkon kuormituksen kannalta. Nämä tietovirran ohjauskomennot on kuitenkin suoritettava turvallisesti, jottei mitkään pahantahtoiset toimijat voi muokata tietovirtaa etäluotsauksen aikana.

Tämän opinnäytetyön tarkoituksena oli luoda julkisen avaimen infrastruktuuri Python-salauskirjastojen ja Dockerin avulla. Tämän seurauksena etäluotsaaja voi lähettää todennettuja komentoja aluksille haluamastaan käyttöliittymästä. Tätä tulosta voidaan käyttää meriliikenteen myöhemmissä digitaalisissa toteutuksissa, joissa viestien autenttisuus on kriittistä.

Asiasanat:

MQTT, JWT, PKI, autentikointi, X.509

Contents

Figures

Pictures

Tables

List of abbreviations

ASN.1	Abstract Syntax Notation One
DRY	Don’t Repeat Yourself
GCP	Google Cloud Platform
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
ICT	Information and Communication Technology
IEEE	Institute of Electrical and Electronics Engineers
M2M	Machine to Machine
MQTT	MQ Telemetry Transport
OS	Operating System
PEM	Privacy Enhanced Mail
PKI	Public Key Infrastructure.
PKCS	Public Key Cryptography Standards
RSA	Rivest-Shamir-Adleman encryption algorithm
SHA	Secure Hash Algorithm
TLS	Transport Layer Security
UML	Unified Modeling Language
UTF-8	Unicode Transformation Format – 8-bit

1 Introduction

When marine technology and ICT are merged, it is crucial to ensure that the peers within the network are who they claim to be. Sending messages through the public internet exposes them to malicious actors, who may alter the messages causing unwanted and sometimes damaging behavior in software functionality. This can be avoided by signing the messages with the sender's signature, which is unique only for the sender. The recipient can ascertain by verifying the signature whether the message has been altered after it has been sent, thus ensuring that the message is reliable.

The commissioner's remote pilotage project, which the thesis is a part of, covers controlling a cargo vessel from a remote location. The commands governing the data flow between the vessel and remote operation centers were not secure, therefore the goal of this thesis is to make them secure. To prevent any malicious actor from altering these commands, it was essential to ensure their integrity and authenticity, given the critical nature of the system. To address this, a proprietary public key infrastructure was developed to establish trusted communication between all system peers. Following the RFC 5280 guidelines, the system enables the exchange of certificates containing public keys, allowing the sender to sign a JSON web token with their private key, which can later be verified using their public key. [1]

The thesis covers background information on how the technology used works, how the system requirements are set to ensure security and how the right tools are selected and implemented to create a system in which the entities can trust each other. Finally, the system is tested in development to ensure functionality and minimal downtime in production.

2 Methods and technologies

This chapter covers the basics of PKI, MQTT and JWT and explains how they work in real life settings.

2.1 X.509 Public Key Infrastructure

The x.509 proposal is RFC 5280 standard to give a guideline on public key infrastructure certificates and certificate revocation list profiles. In PKI we use applied cryptography to gain several features such as encrypting messages so only intended recipients may read them, creating signed hashes to ensure integrity of data, authenticating entities within network for who they claim to be using digital signatures. [2]

2.1.1 Asymmetric keys and certificates

PKI key is a hash of a chosen length. Security bits change regarding the algorithm and bit length of the key. These keys are generated as a pair using key generators. There exist several different algorithms for generating these keys. Common amongst these are RSA and Elliptic-curve cryptography [3]

A public key is a PKI key which is shared and meant to be public. It is used either in encrypting messages so that only the private key owner can open them like in Figure 1 or in verifying signatures on data which has been signed by using private key.

Figure 1. Asymmetric key encryption process.

A private key is a PKI key which is only known and used by the owner of said key. This key is used for either writing signatures on data of choice or decrypting data. The signature is created by calculating the hash of the message with hashing algorithm for example SHA-256 and then encrypting the hash using RSA private key for example. The signature is then accompanied by plain data and can be used to check whether the message has its integrity by calculating the hash over the plain message and comparing it to the signature that is decrypted using the public key. [4]

A certificate is a file which contains the public key of the subject and information about the certificate authority who issued the certificate. The Picture 1 describes all the possible mandatory or optional fields that are included within the certificate. [1]

```
Certificate  ::=  SEQUENCE  {
        tbsCertificate       TBSCertificate,
        signatureAlgorithm   AlgorithmIdentifier,
        signatureValue       BIT STRING  }
    TBSCertificate  ::=  SEQUENCE  {
        version         [0]  EXPLICIT Version DEFAULT v1,
        serialNumber         CertificateSerialNumber,
        signature            AlgorithmIdentifier,
        issuer               Name,
        validity             Validity,
        subject              Name,
        subjectPublicKeyInfo SubjectPublicKeyInfo,
        issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                             -- If present, version MUST be v2 or v3
        subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                             -- If present, version MUST be v2 or v3
        extensions      [3]  EXPLICIT Extensions OPTIONAL
                             -- If present, version MUST be v3
        }
    Version  ::=  INTEGER  {  v1(0), v2(1), v3(2)  }
    CertificateSerialNumber  ::=  INTEGER
    Validity ::= SEQUENCE {
        notBefore      Time,
        notAfter       Time }
    Time ::= CHOICE {
        utcTime        UTCTime,
        generalTime    GeneralizedTime }
    UniqueIdentifier  ::=  BIT STRING
    SubjectPublicKeyInfo  ::=  SEQUENCE  {
        algorithm            AlgorithmIdentifier,
        subjectPublicKey     BIT STRING  }
    Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension
    Extension  ::=  SEQUENCE  {
        extnID      OBJECT IDENTIFIER,
        critical    BOOLEAN DEFAULT FALSE,
        extnValue   OCTET STRING
                    -- contains the DER encoding of an ASN.1 value
                    -- corresponding to the extension type identified
                    -- by extnID
        }
```

Picture 1. X.509 v3 syntax in ASN.1.

A hash is calculated over the certificate which is then encrypted with the issuer's private key. The encrypted hash can then be decrypted and verified. After this verification process, we can trust the subject of this certificate whether

the hashes match. The certificates have validity periods, outside which the certificate is not to be trusted due to possible key compromises occurring on either other hosts or CAs.

A certificate signing request (CSR) is a means to acquire a certificate. The host creates this document which contains basic information about it so that a certificate can be created. The request holds the public key of the requester which can be used for verifying the signature of said request to ensure integrity. The CA will validate the subject and after that create the x.509 certificate to the repository. [5]

A certificate revocation list (CRL) is generated by the CA to list entities that are not to be trusted. It is periodically issued and contains certificate serial numbers that are no longer valid and thus are not to be trusted. There are 10 reason codes for revocation the certificate. Table 1 shows all possible reason codes for certificate revocation. [2]

Table 1. CRL reason codes.

Code	Reason
0	unspecified
1	keyCompromise
2	cACompromise
3	affiliationChanged
4	superseded
5	cessationOfOperation
6	certificateHold
8	removeFromCRL
9	privilegeWithdrawn
10	aACompromise

2.1.2 Certificate authorities

An intermediate CA is an entity which directly communicates with the end entities(hosts) and performs tasks such as issuing and renewing certificates, writing CRLs and storing/sending them. [2]

Root CA is the base trust in PKI. Its certificate is self-signed and is used for verification of intermediate CA's certificate. It also creates a CRL for the same purposes as intermediate CAs. [2]

A chain of trust is established when all the entities within the same PKI are certified by one entity above their own authority level. This means that root CA will sign itself, and the intermediate CA. The intermediate CA will sign the end-entities. Figure 2 shows how each entity uses the issuers public key to verify their certificates signature. [6]

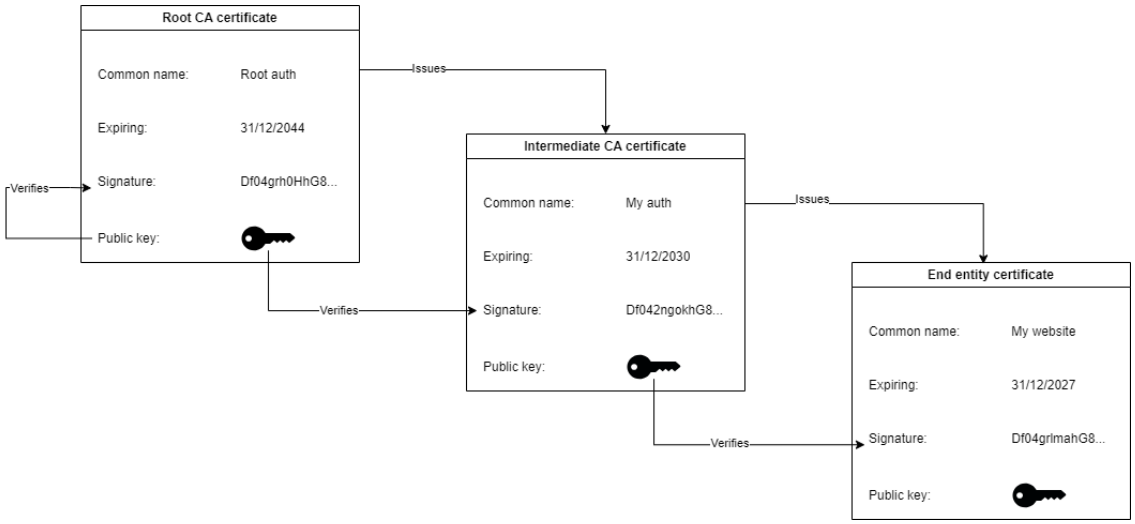


Figure 2. Chain of trust.

2.2 MQTT

MQTT is a messaging protocol used within IoT and IIoT. It allows machines to communicate with each other through a data broker. The machines could be e.g. sensors, actuators or embedded devices. The protocol itself is lightweight

and is scalable for large systems. MQTT clients can use connect, publish, subscribe and disconnect control packets to the broker [7] [8]

During connection, a client sends a request to the broker with a header which contains information such as protocol specifications, several flags which determine what payloads are in the message and connection properties such as maximum packet sizes, and authentication methods. The payload of connection request contains client ID, will info (a message which is published when the client is ungracefully disconnected due to e.g. connectivity problems), username and password. [8]

Publishing message happens with a topic name. This topic name can be any UTF-8 string, however it cannot contain any wildcard characters. The topic level is separated by '/' forward lash. This allows clients to subscribe to multiple topics at once by subscribing to a higher level. For example, in IoT smart house scenarios we could have topics like "home/room1/humidity" and "home/room1/temperature". Subscribing to only one topic can be achieved by including the full topic as a subscribe payload. The payload must match the topic. Additionally, clients can use single-level '+' or multiple-level '#' wildcards in their subscription. Using "home/+" will acquire all messages on only said level such as "home/room1" and "home/room2". This however does not include "home/room1/temperature" meaning the client is subscribing only to the level 2 messages. Using "home/#" however, will include all the levels beyond "home/". This means that the subscriber acquires "home/room1/temperature", "home/room1/humidity" and "home/room2/temperature". Figure 3 shows how clients can publish and subscribe to messages in MQTT communication protocol [8]

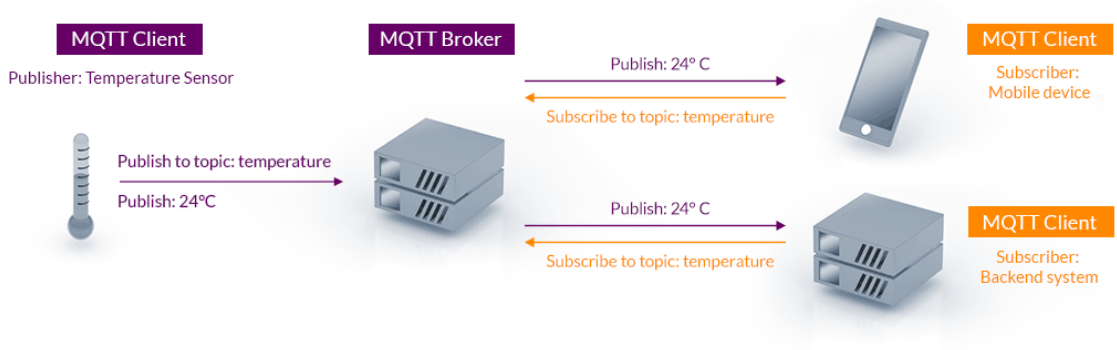


Figure 3. MQTT pub/sub communication [9].

2.3 JWT

JSON web token is a secure way of transmitting messages between peers. The messages are commonly base64Url encoded and contain a signature what is used for verifying the integrity of messages. The JWT token structure parts are separated by dots and have the format of “xxxx.yyyy.zzzz” in the following order.

- Header
- Payload
- Signature

The header has two parts, the type of token and the algorithm that is used for the signature. The header is in JSON format, and it is then Base64Url encoded. [10]

The payload contains the claims of the token. There are three types of claims, registered claims; recommended but not mandatory that determine information about who issued the token, when it’s expiring and who is the subject for the token. Public claims are conventional claims listed in the IANA JWT registry[11]. Private claims are custom claims created to share information between peers. The claims are in JSON format and then are Base64Url encoded. [10]

The signature part of JWT is constructed by taking the base64Url encoded header and payload and passing them into a hash algorithm. Then the hash is

encrypted with a private key to ensure integrity of the message. The signature is then appended to the JWT as a last part. In usual use cases, the JWT is not encrypted so header and payload are easily decodable to human readable form. However, authentication and integrity are achieved by using the sender's public key. We can compare the hashes of header and payload to the decrypted hash from the signature. [10]

3 System requirements

The system must meet certain requirements specified in this chapter to function as intended and have proper security hardenings.

3.1 CA requirements

The CA should reside in a Docker container to ensure that only users with super user rights have access to the source code within. This feature allows deployment to wider range host machines despite which OS or architecture they have if they have Docker installed.

The CA should use RSA private keys with a size sufficiently big enough to be tolerant for attacks throughout the future.

The CA should receive its certificate from a third-party CA that can be used to verify its certificate by the hosts that receive their certificate issued by the intermediate CA thus achieving chain of trust.

The CA should store its instance data within the volume of the host machine. This will create a crash proof persistence where under the circumstance of code failure and Docker container shutdown, we can recover the instance for investigation and debugging of failure point. The instance data includes class properties that are used for identification of the CA, private and public keys of the CA, certificate of the CA issued by the third-party root CA, CRL and CSR of the CA.

The CA should store the issued certificates within the volume of the host machine. This will prevent the loss of certificates under the circumstances of Docker container crash.

The CA should store a human readable JSON file of issued certificates within a volume on the host machine. This enables administrators to view the issued certificates during debugging.

The CA should be able to issue a CRL and append revoked certificates to it during events of key compromise of hosts or intermediate CA. The reason codes that are inserted as extensions to the revoked certificates' list should follow the RFC 5280 proposal. [5]

The CA should have an API interface to manually revoke, renew and issue certificates, this allows the administrator to seize the functionality of the PKI system in circumstances of software failure and cyber-attacks.

The CA should warn the administrator of the events of malicious certificate request attempts. Ensuring the administrator will instantly be notified that the system needs inspection and possible actions, e.g. shutting down or IP blocking.

The CA interface should be accessible from public internet only via port 443 with TLS/HTTPS security on place. This way we can ensure that only secure connections are established. [12]

The CA API should only be accessed with basic authentication credentials, so the interfaces such as certificate issuance, download and renewal can only be accessed with software owner's pre-defined username password -combination.

The CA should have its private keys encrypted with a strong password so, in the event of key compromise or unwanted host machine access, the actor may not be able to open and use the private key in a malicious manner.

The CA API should have a brute force or DOS-attack prevention measures. A simple maximum requests per second per source will ensure that the API will not give out all its resources for a single host.

3.2 Host requirements

The host instance should reside inside a Docker container. This enables deployment to all platforms that have Docker instance installed. Additionally,

only super user accounts can access files and execute commands within the container.

The host instance should save its certificates and key pairs within the volume that resides on the host OS. This ensures that in the circumstances of software failure, the instance data is not lost, and a new container can be started with persistent data.

The host instance should verify JWTs received by using the public key from the sender's certificate. This will ensure that the message received from the MQTT broker has the correct identity and therefore can be trusted.

The host instance should sign outgoing JWTs with its own private key. This will ensure that only the receiving hosts can verify the identity of the sender.

The host instance should publish and subscribe to MQTT topics successfully via MQTT broker residing on the public internet. Additionally, during receiving a message, the host should send an acknowledgement message to the sending party to confirm that the message has been received. This message should also be in JWT format so that the host which sent the command can be sure the right entity has successfully processed the message.

The command sending host should have a written command specification standard which syntax is followed. This can ensure that roque commands can be avoided, preventing system abuse by malicious actors.

During the certificate expiration, the host instance should automatically send a certificate renewal request to the CA to ensure minimal system downtime. The recipient of MQTT messages should also drop the message, whether the sender's certificate has expired.

The host instance should automatically request for new CRL during the expiration of old one. Additionally, always check whether the destination or receiving common name of MQTT messages are not listed on the CRL.

The host instance should always verify the new certificate with the issuers certificate. Intermediate CA's certificate should be verified with root CA public key. Host certificates should be verified with the intermediate CA's public key.

The host instance should have an API interface for sending commands forward. This API should have command validation in place for preventing system failures.

The host instance should connect to the CA using HTTPS connection. Meaning the connection is secure and packets sent within can be trusted.

The host instance should save its private key in a secure location within the host OS with strong password encryption in place. This means gaining access for the key should be difficult for malicious actors without super user privileges and opening and using the key should be computationally time consuming.

The connection established to the MQTT broker residing on the public internet should be implemented with TLS. This adds an extra layer of security by only letting authorized hosts connect to the broker, therefore lowering the probability of roque commands within the network.

4 System topology

The system has been designed by using the technologies, frameworks and standards described in this chapter. The tools have been chosen from industry standard tools, keeping the system requirements in mind.

4.1 Top topology

The solution is using industry standard tools. As seen in figure 4, the hosts are connected to the service on the virtual machine residing in cloud platform using NGINX.

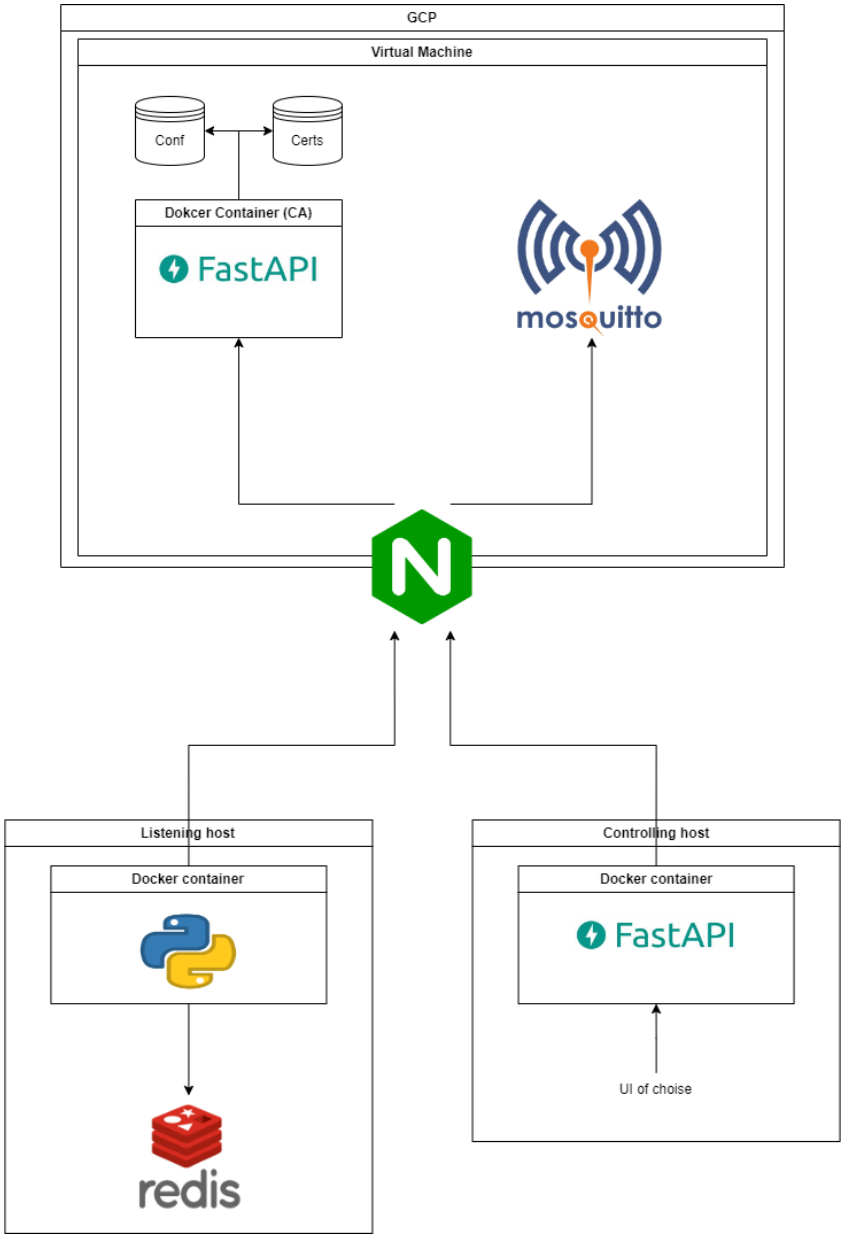


Figure 4. Top topology of the solution.

The solution's computing happened in the cloud. GCP was chosen to be the service provider. This is due to the thesis client's previous usage history of this cloud service provider. GCP is a cloud platform by Google. It provides services used in this solution such as compute engine; an IaaS solution to run MS windows and Linux virtual machines and Cloud DNS; a DNS hosting service. [13]

Linux is going to be used as a host OS for the solution due to being lightweight and personal preference within the company. It has a wide packet repository of software, including those used in this solution. A docker instance was started for hosting the code on the OS without changing python dependencies. The docker can also be connected to directories with host OS for data persistence. These directories linked into docker containers are called volumes. These volumes are going to be used as a PKI entity configuration file storage, plain text JSON file storage and a repository for certificates and CRL. [14]

NGINX was chosen as a server due to its easy configuration and support for TLS. It is used to locate services behind DNS URLs so that API interfaces can be accessed from public internet without using IP addresses. NGINX also supports basic authentication on its servers. This means that to access certain services, an authentication header is needed. [15]

Python is used as a programming language due to its easy readability and large package repository of connectivity and cryptography modules. Additionally, the developer is most experienced using this programming language.

4.2 CA architecture

4.2.1 Python cryptography library

Python cryptography library is going to be used as a core functionality provider in this solution since it has all the cryptographic functionality required for this project. The library provides symmetric encryption, X.509 functionality, asymmetric encryption and hashing. [16]

Cryptography depends on OpenSSL which is an open-source toolkit for cryptography and secure communication. This enables the creation of CRL, and appending entries to it with CRL reason flags.

The python cryptography library can be used to open CSR files. Additional security checks and validations can be implemented to inspect whether the

requesting entity is trusted or not. This validation process can be connected to Microsoft teams webhook function, throwing alerts when suspicious certificate requests occur.

4.2.2 Choosing the cryptographic keys

RSA keys with the modulus size of 1024 are considered not adequate due to their security strength in bits being less than 80. Table 2 shows security strengths of different key sizes which are considered in choosing the key. [17]

Table 2. Security strengths of different RSA key sizes [17].

Security strength	RSA key modulus size
≤ 80	k = 1024
112	k = 2048
128	k = 3072
192	k = 7680
256	k = 15360

The security bit strength affects the time frame through which the cryptographic keys are secure enough. For this solution, the smallest security strength which applies for time frame 2031 and beyond was chosen. The key is RSA with modulus size k = 3072 resulting in security bit strength of 128 thus being acceptable in 2031 and beyond, consulted from Table 3. [17]

Table 3. Security strength time frames [17].

Security Strength		Through 2030	2031 and Beyond
< 112	Applying protection	Disallowed	
	Processing	Legacy use	
112	Applying protection	Acceptable	Disallowed
	Processing		Legacy use
128	Applying protection and processing information that is already protected	Acceptable	Acceptable
192		Acceptable	Acceptable
256		Acceptable	Acceptable

For security reasons, the private key is encrypted with a password. This means only peers who know the password may sign messages or decrypt data. Doing this measure using python cryptography module allows the generating of new keys and certificate revocation during the circumstances of key compromise before malicious actor manages to use the key.

4.2.3 API endpoint

In this solution, fastAPI is going to be chosen as the web framework. It generates interactive documentation for endpoint testing that can be accessed via browser. It also supports type hinting and has built-in functions for file handling and security measures. [18]

The API is going to have endpoints for certificate revocation, renewal and issuance. The endpoints are going to be accessible using any HTTP request tools or the interactive documentation by fastAPI. There is going to be a rate limiter enabled on the endpoints which ensures that a large number of requests cannot be accepted within the pre-defined time.

4.2.4 3rd party root certificate authority

Open-VPN easy-rsa is going to be used as the root CA in this solution. It is a CLI tool for building and managing PKI. The tool runs in our case already

initialized in Linux environment and can be used to create certificates for intermediate certificate authority by using CA certificate request as input.

The certificate from easy-rsa PKI is going to be included in configuration files of all PKI entities of the solution due its public key must be extracted for validation chain verification.

4.3 Controlling host architecture

The host which controls the other host has a paho-MQTT library for python for connecting and communicating with MQTT broker. The library implements MQTT protocol versions 5.0, 3.1.1 and 3.1 so it is suitable for this for this solution since those are currently relevant versions as of writing this in 2024. [19]

The controlling host should also have FastAPI interface for sending the commands though interactive documentation or some other UI that might be developed later. This enables the application to have type validation, since Fast API comes with Pydantic. Therefore, commands sent with this host cannot be injected.

When sending CSR, renewing certificates or downloading certificates and CRL, the hosts use python requests library. It is a library for sending HTTP/1.1 requests with features fulfilling this solution requirements. It allows basic authentication headers for connecting to servers with required authentication headers. Due CA having TLS, the request library is going to be used with HTTPS scheme. [20]

4.3.1 Deployment and production set-up

For easy deployment for production and multi-platform support, Docker was chosen. The images should be built slim so that the host OS won’t lose much storage space.

The container images are going to be stored within a proprietary docker hub which resides within the same virtual machine and behind the same NGINX instance as the MQTT broker and the CA. This eases the deployment process from the development environment to production environment.

The hosts are going to be connected to a Mosquitto broker which is a MQTT broker provided by Eclipse. The Mosquitto broker implements MQTT 5.0, 3.1.1 and 3.1 therefore it is fully compatible with paho-MQTT library. [21]

4.4 Listening host architecture

The listening host functionality resides within a Docker container. This allows deployment to any machine with running Docker instance. The Container is going to have volumes that are connected to it, which reside within the directory that the container has been started in. The volume will have certificates, CRL and key files of said instance.

The MQTT messages from other hosts within the network have JWT tokens as a payload. This allows us to authenticate the commands by comparing the hash behind the signature using public key to decrypt it. An acknowledgement message is going to be sent, that is signed with private key. Doing this ensures that commands sent within the network are successfully received and processed by the right entity. The connection to the MQTT broker is established via TLS.

By using the python cryptography library, the certificates are going to be always checked for expiration during message exchange. If a certificate has expired, a renewal request is issued to the CA and a new certificate will be issued and then fetched. This new certificate goes through a validation process where it is verified by using the issuer’s public key. During this message exchange, the CRL is also inspected, and a new one is requested whether the current one is expired.

5 System development

This chapter describes the development process of the system, including the development tools, software implementation and reasoning.

5.1 Development tools

The development process started with creating a python virtual environment to have package locking system to ensure version updates to modules do not introduce unwanted breaking of the software. Make commands were used to test and deploy code rapidly.

Code version control was conducted using proprietary GitLab instance running on organizations private server. A lightweight CI/CD pipeline was set up to ensure that the software runs on certain python versions. This decision was made due to a problem arising when trying to build the production code on the newest python during that time 3.13.0. Said version has compatibility issues with one of the dependencies of FastAPI library.

VS code was used as an IDE for software development. It provides extensions for python software development from the extensions marketplace, including syntax highlighting and docstring generators which ease the development process.

5.2 Parent class for PKI functionality

The CA and PKI clients share the same core functionality, they needed to save instance data and PKI files such as keys, certificates and CRLs. Obeying DRY principle in programming, the functionality was established using OOP approach. Figure 5 shows the class inheritance that was implemented with public properties and methods. Private methods and properties are omitted from the UML, since they are not called outside the class.

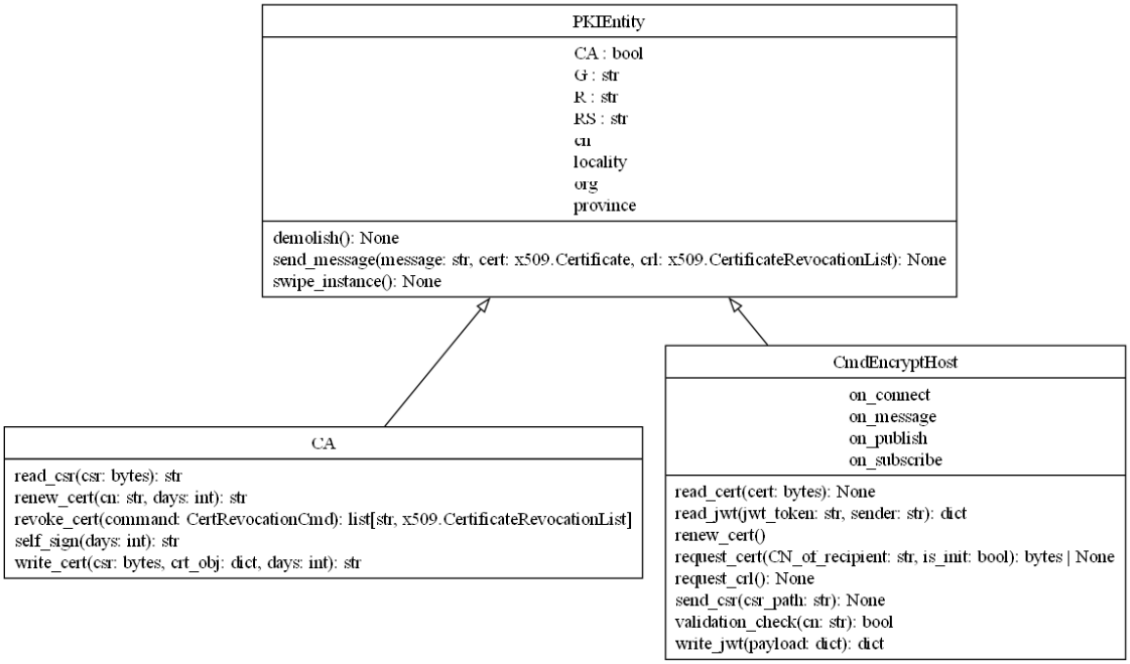


Figure 5. UML diagram of entity classes.

PKI entity is the parent class which is inherited by both CA and hosts. It takes parameters such as common name, locality, organization and province as parameters. These parameters are used in CSR generation and certificate issuance. Demolish and swipe instance methods are used if the entity needs to be deleted. Demolish will delete all data and directories. Swipe will only delete data.

CA class inherits all the properties that are passed from PKI entity and adds proprietary methods. During initialization, the CA bool is set to be true, then the instance data will be saved in a file named ca.json. otherwise, the instance data would be saved in instance_data.json.

Read CSR is used when the API end point has a POST request which contains a CSR. This CSR is then read, and the certificate writing method is called. After that the method will return a status code whether the CSR reading and certificate writing was successful.

Renew certificate method is used when an end-entity wants to extend the validity period of its certificate. This method opens the certificate of said entity

and edits the certificate validity fields and saves the updated certificate into the repository.

Certificate revocation works by giving the reason code of revocation and the common name of certificate holder or the serial number of the certificate. The reason code is mandatory parameter as the reason is listed as an extension to the CRL entry of the certificate. Additionally, reason code 8 can be passed as a parameter to remove the certificate from the CRL. When the process is finished, a new version of the CRL is saved to the repository with updated contents and timestamps.

Self-sign functionality is called in events when the CA is intended to work as a root CA. During self-sign, a certificate is issued to the CA where the issuer and subject fields are identical.

Certificate writing function is called by the CSR reading function. This will create a new certificate based on the information of CSR. The data is saved into the repository as a PEM file.

Properties on_message, on_publish, on_connect and on_subscribe are message handlers for paho MQTT library. They are used for logging information about the connectivity, handling messages that are received and informing about successful publishing or subscription.

Certificate reading method opens a certificate which has been downloaded and validates it using issuers public key. A successfully verified certificate will be saved into a volume of the host.

The JWT reading method will be called on the on_message handle. It takes the JWT from the message and the common name of the sender. The certificate of the sender is then opened from the directory and its public key is used for verifying the signature hash of the JWT message. After which a response acknowledgement JWT message is sent back to the sender.

Certificate renewal method is called when the host requires it due to outdated previous certificate. This method will send a HTTP POST to the CA and the CA will run the certificate renewal method, returning a HTTP status code.

Certificate request method is called when a CSR has been posted or certificate renewal has been requested with a successful status code. This method does a HTTP GET request to the CA.

CRL request does HTTP GET request for a CRL. The CRL is then downloaded into the volume for later usage.

The sending of CSR is during the initialization of the host instance. It will send a HTTP POST request to the CA containing the CSR as a file attachment.

Validation check is implemented on publish and message receiving events. It opens the certificates of the instance and checks their validity. The CRL will be opened, and the contents are compared to the certificates. All expired documents will be replaced with new ones by requesting and installing.

Write JWT takes the command or acknowledgement message as a JSON and then creates and appends a signature of it. The JWT is then returned as a string to be sent further.

The host class inherits most of the attributes from the parent class, however it has proprietary methods during initialization. Figure 6 shows the sequence of initialization happening within a host container.

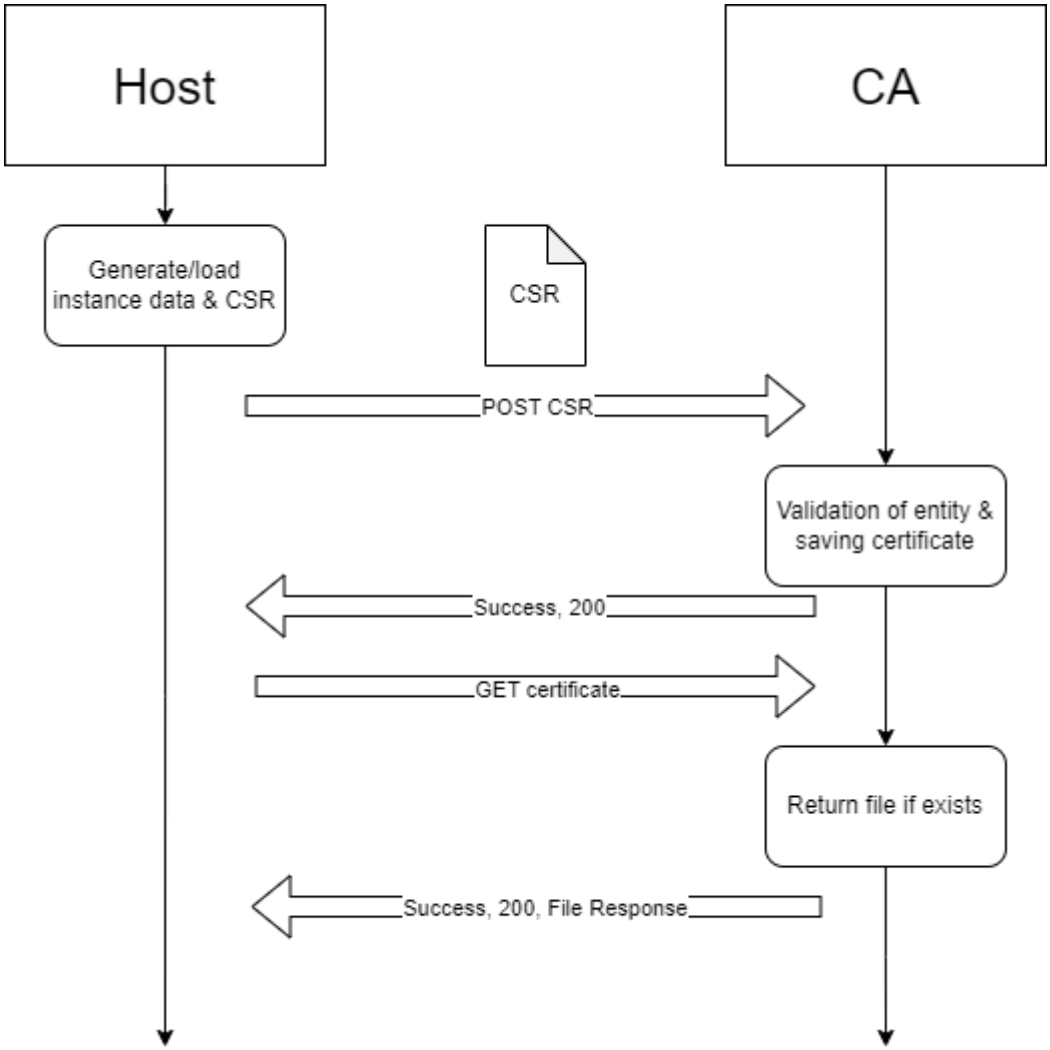


Figure 6. Host initializion process.

5.3 Development of CA

The CA class is imported into an app.py file which has FastAPI imported. There are several endpoints defined for the API that can be called through NGINX reverse proxy.

The root endpoint of this API returns a hello message from the CA. It is used for connectivity tests because there is no functionality within the endpoint what could result in API not responding.

CSR endpoint has HTTP POST as a method, and it takes a CSR file as a parameter. It will call the CSR reading method of the CA class and return a HTTP status code.

Certificate endpoint uses HTTP GET as the method and returns a file response from path that is inputted as a parameter to the URL.

The CRL endpoint uses HTTP GET as the method and returns a file response containing the CRL. The endpoint will check whether the CRL is up to date before sending the file and will update the expiration of the CRL if necessary.

The revocation endpoint takes common name of the certificate subject or serial number of the certificate as a parameter in addition to the reason code which is used for adding the CRL reason code extension to the certificate or de-revoking of the certificate. The endpoint calls the certificate revocation method of the CA class.

The renewal endpoint calls the certificate renewal method of the CA class.

This API will run within a container that has two volumes, one for certificates and one for configuration files. The source code is copied within the container and the dependencies are installed from the pip package registry. After the installation, the API will be started using uvicorn ASGI and the port will be exposed for usage of NGINX.

5.4 Development of controlling host

The root endpoint of controlling host will also return a greeting message as a response. This can be used in connectivity debugging in case other endpoints are not working.

The command endpoint will take the command as an argument. The command is validated by Pydantic. After type validation, an additional block will check whether the command is listed in the accepted commands list. Finally, the endpoint will publish the command to the MQTT broker.

This API will run within a container that has one volume connected on the host OS. This volume contains the configuration files of the PKI entity instance. The applications port which has been set up by running it with uvicorn is forwarded from within the container and then accessed on the local network. Figure 7 shows how the command message is transmitted, and all the validation happening on the background during the message exchange.

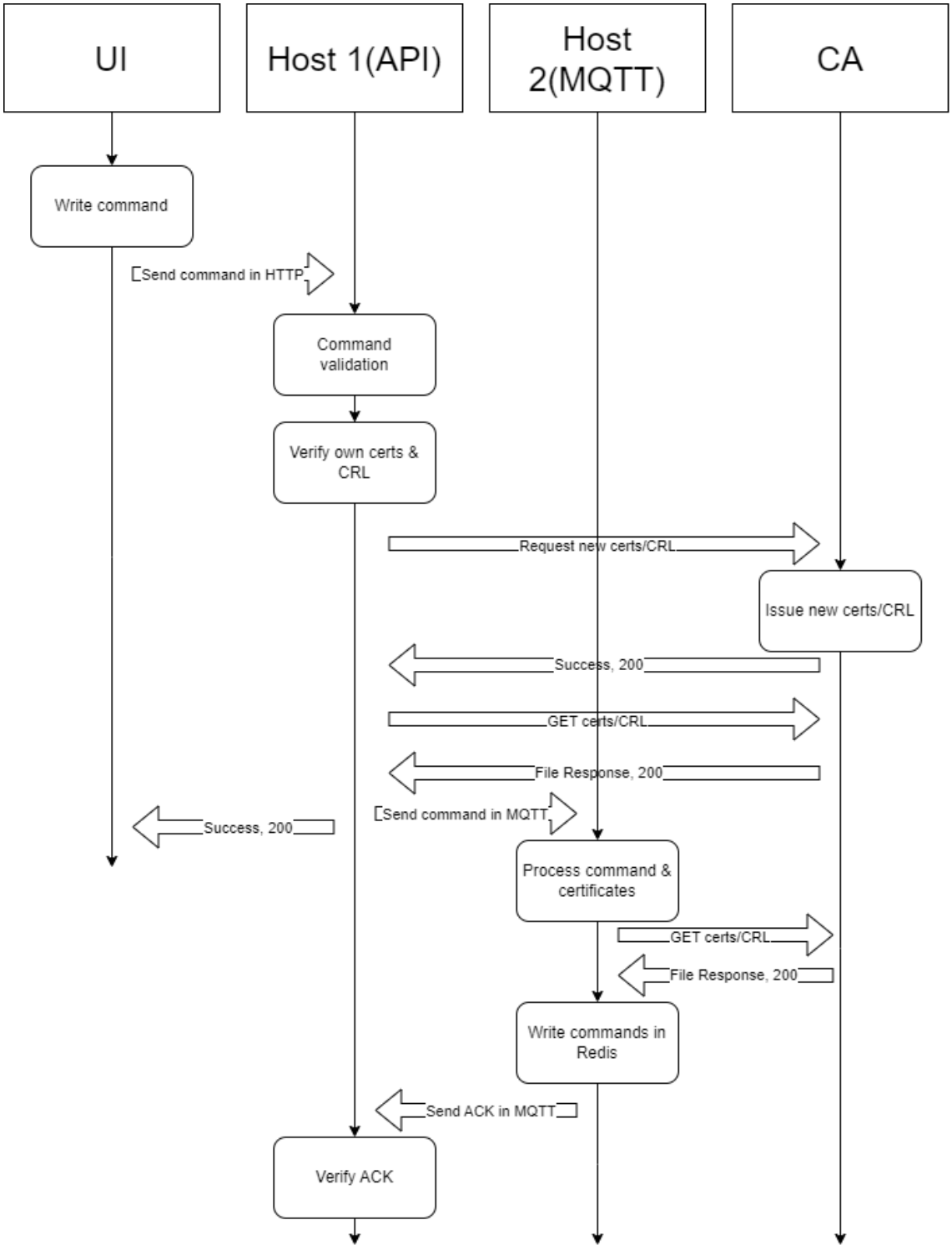


Figure 7. Message exchange.

5.5 Development of listening host

This application is running on a vessel. This means that the software will not have internet connectivity all the time. However, using paho-MQTT's loop forever method, we can re-connect to the broker whether the client gets disconnected for any reason. Slight precautions must be taken when initializing the container at sea. A function which will use python sockets tries to connect to 8.8.8.8 address will return a successful connection flag to the main program if it is safe to connect to CA.

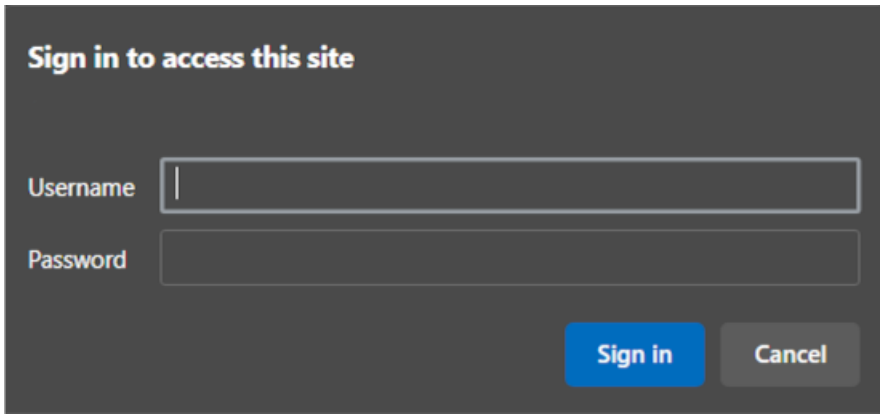
There are no API endpoints and no forwarded ports on this container since the connection on this entity is only for M2M communication. The MQTT messages are parsed by the JWT reading method of host class and the commands are saved into Redis database for further processing of other software.

6 System functionality

The functionality of the system is tested by doing end user tests for each API and reading the logs of docker container during execution.

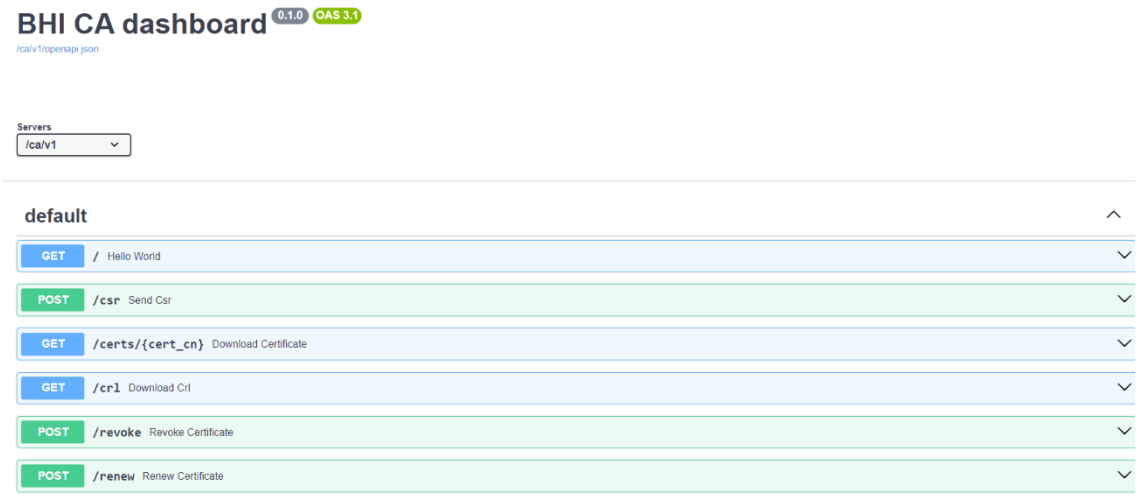
6.1 CA API testing

The CA API dashboard and functionality must be secured for only authorized users. This required usage of basic auth on the NGINX server. Picture 2 Shows the authorization form by Microsoft edge browser on websites with basic auth.



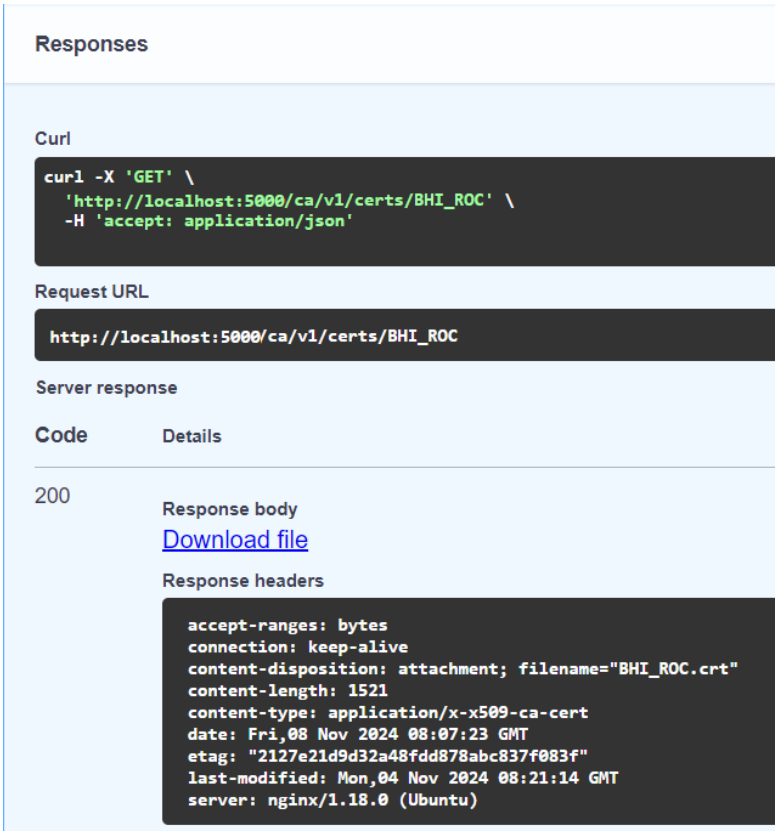
Picture 2. Basic authorization on CA API.

All the endpoints can be tested from the interactive documentation that is automatically generated when the FastAPI application is running. Picture 3 shows the UI of CA dashboard, that can be used to run commands from browser.



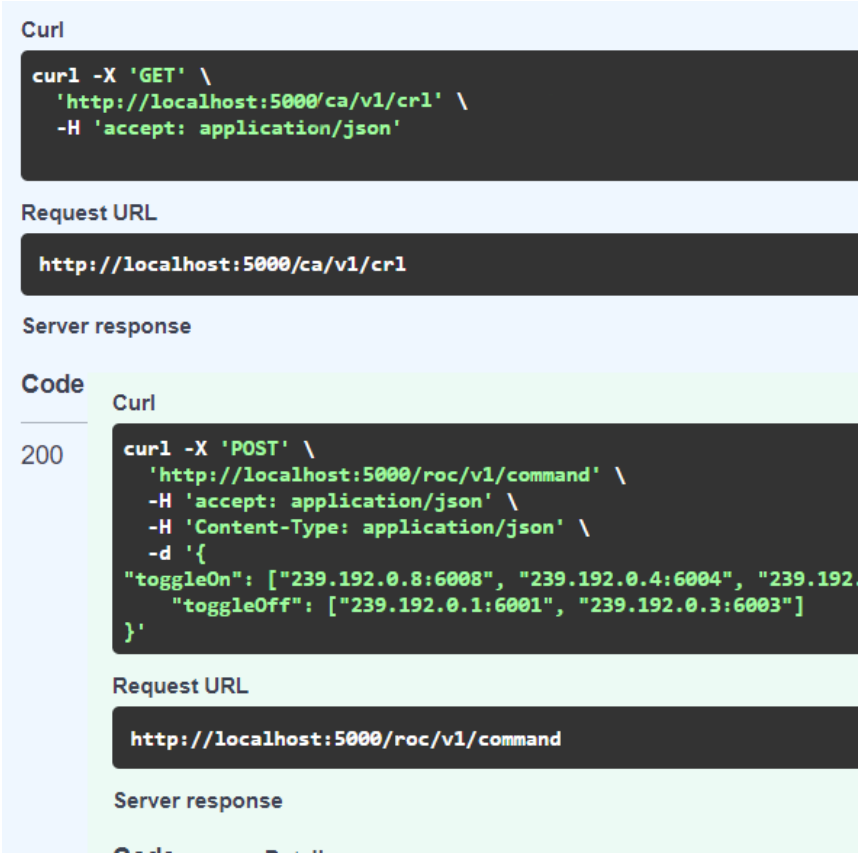
Picture 3. FastAPI interactive docs.

Certificate request endpoint works well by typing in the common name at the end of the endpoint for file response query. Picture 4 shows, by using curl command we receive a file response to our web browser, with correct headers.



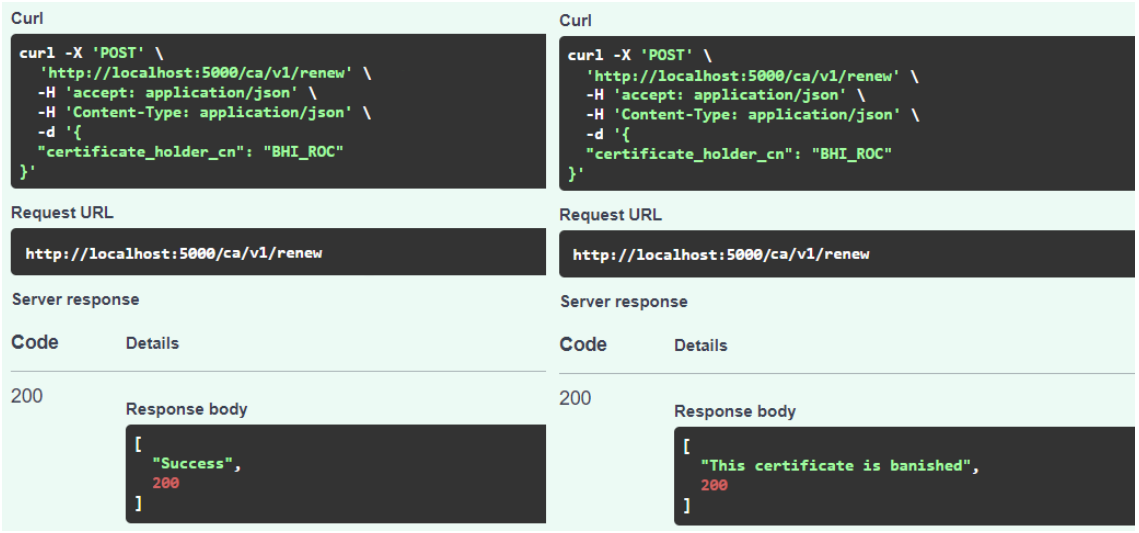
Picture 4. Successful certificate GET request.

The API returns successfully a CRL on request that has been authorized. The CRL file is in PEM format and can be opened with any ASN.1 decoder. Picture 5 shows a 200 HTTP status code and a downloadable file which is the CRL.pem



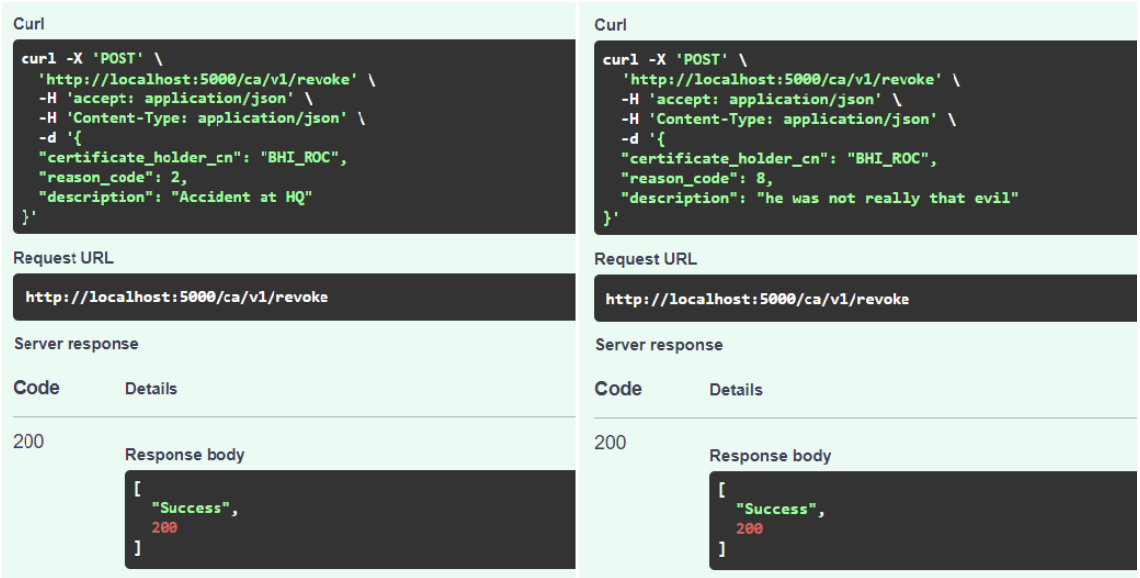
Picture 5. Successful CRL GET request.

Renewal of certificates returns a success code whether the certificate is not listed in the CRL. This is a security mechanism where the certificate will not be updated and therefore not used by entities, even if they have no CRL. Picture 6 shows how the CA will deny the certificate renewal request if the certificate is listed in the CRL.



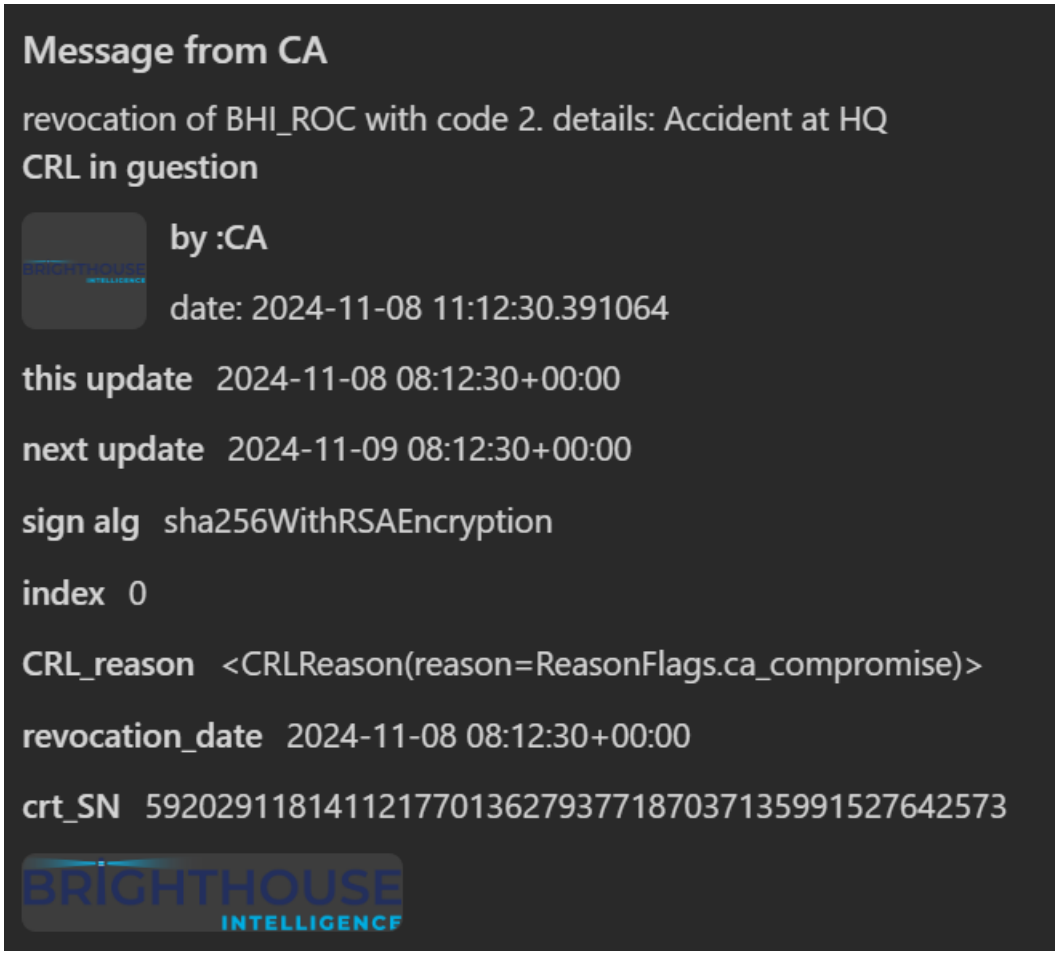
Picture 6. Certificate renewal endpoint tests.

Certificate revocation is successful with any code specified. A detailed description of what has happened before revoking can be added optionally to the request data. The reason code 8 will remove the certificate from the CRL. Picture 7 shows how adding certificate holder's common name with a reason code to the request data will successfully modify the CRL.



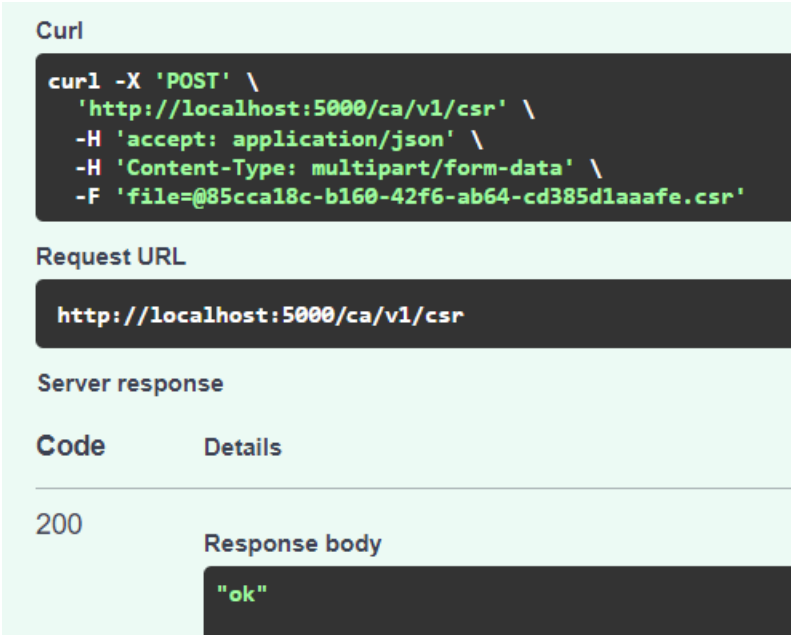
Picture 7. Certificate revocation

The CRL change will be posted to MS teams as a webhook. Picture 8 shows how the whole CRL is spread into a webhook card. The card starts with the detail data that has been passed into the request at /revoke endpoint.



Picture 8. Webhook message in teams.

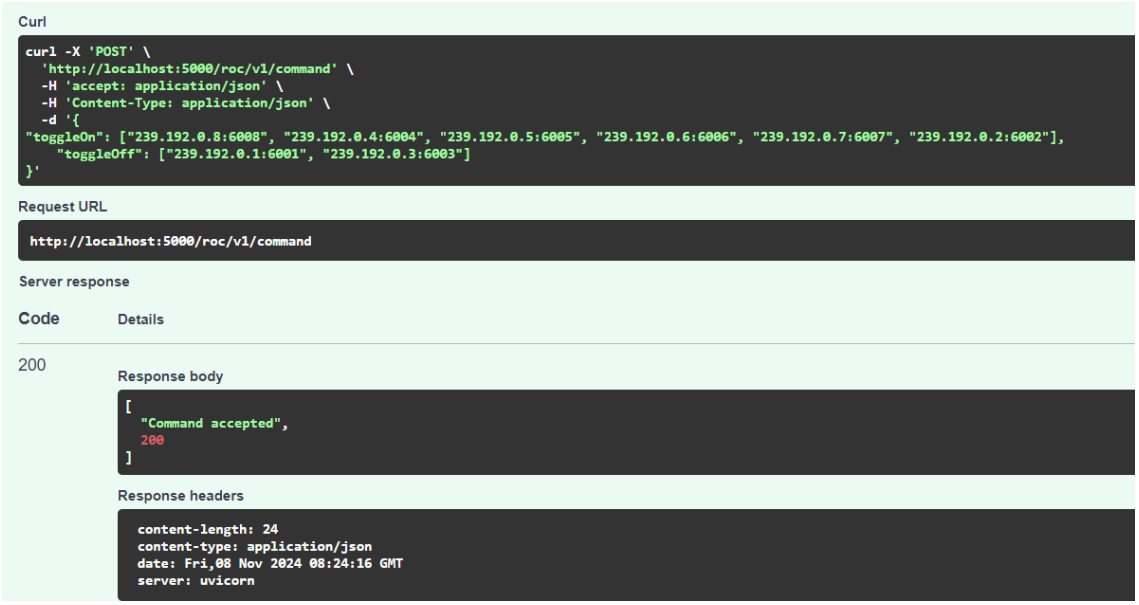
The CSR endpoint takes a file as a parameter and opens it. After which does validation check and parsing and finally saves a certificate to the repository. If all phases were successfully done, HTTP status code 200 will be returned as seen in picture 9.



Picture 9. Successful CSR POST.

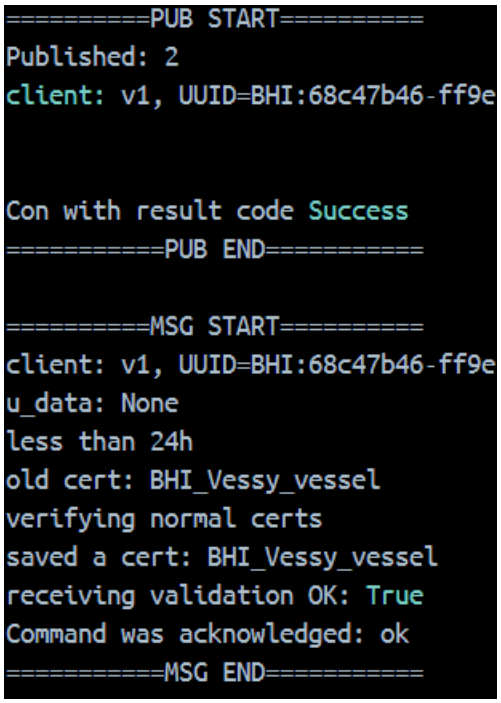
6.2 Host API testing

The controlling host API endpoint is accessible from the local host. Picture 10 shows how a HTTP POST command is successfully processed by the host API when all the parameters are correctly inputted.



Picture 10. Successful host API command.

The Docker container shows the information about what happens within the MQTT client during the message exchange. Picture 11 shows the information about message publishing and receiving the message. During the message receiving process, the certificate has expired so a new one is requested from the CA. The JWT acknowledgement message is verified, and then the status is logged.



Picture 11. Control host docker logs.

The message arrives to the listening host running inside a Docker container. The host has subscribed to the topic and receives the message when the broker publishes it. Picture 12 shows the logging of command processing at the listening host's end. The certificate has expired so the host requests a new one from the CA after which it verifies the message using the public key from the new certificate and saves the command into the Redis database.

```
2024-11-08 08:24:20,747 No.303 ==========MSG START==========
2024-11-08 08:24:20,748 No.304 client: v1, UUID=BHI:cc386086-1bb7-46fb-8e6e-2c0e3cdcbc3a C=FI province=varsinais-suomi
ocality=Turku org=BHI cn=BHI_Vessy_vessel
2024-11-08 08:24:20,748 No.305 u_data: None
2024-11-08 08:24:20,748 No.309 sender: BHI_ROC
2024-11-08 08:24:20,749 No.310 command: eyJhb...

2024-11-08 08:24:20,749 No.312 Reading JWT from BHI_ROC
2024-11-08 08:24:20,751 No.383 less than 24h
2024-11-08 08:24:22,220 No.387 old cert: BHI_ROC
2024-11-08 08:24:23,083 No.177 verifying normal certs
2024-11-08 08:24:23,086 No.142 saved a cert: BHI_ROC
2024-11-08 08:24:23,087 No.393 calc:2024-11-08 08:24:23.086905+00:00 - crl last update =  4 days, 0:05:54.086960
2024-11-08 08:24:25,823 No.229 receiving validation OK: True
2024-11-08 08:24:25,826 No.262 toggleOn: ['6008', '6004', '6005', '6006', '6007', '6002']
2024-11-08 08:24:25,826 No.263 toggleOff: ['6001', '6003']
2024-11-08 08:24:25,829 No.383 less than 24h
2024-11-08 08:24:27,428 No.387 old cert: BHI_Vessy_vessel
2024-11-08 08:24:28,998 No.177 verifying normal certs
2024-11-08 08:24:29,001 No.142 saved a cert: BHI_Vessy_vessel
2024-11-08 08:24:29,001 No.206 sending validation OK: True
2024-11-08 08:24:29,641 No.328 ==========MSG END==========
```

Picture 12. Listening host message processing.

To make sure that our message has successfully been processed, an acknowledgement message is sent. Picture 13 shows the publication of a message. The publication was done with successful status.

```
2024-11-08 08:24:29,643 No.343 ==========PUB START==========
2024-11-08 08:24:29,643 No.344 Published: 6
2024-11-08 08:24:29,643 No.345 client: v1, UUID=BHI:cc386086-1bb7-46fb-8e6e-2c0e3cdcbc3a
ocality=Turku org=BHI cn=BHI_Vessy_vessel
userdata: None
props: []
2024-11-08 08:24:29,644 No.346 Con with result code Success
2024-11-08 08:24:29,644 No.347 ==========PUB END==========
```

Picture 13. Acknowledgement message published on listening host.

The Pydantic type checker and validation function work as seen in Picture 14.

Curl

```
curl -X 'POST' \
  'http://localhost:5000/roc/v1/command' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
"toggleOn": ["239.192.0.8:6008", "239.192.0.4:6002", "239.192.0.5:6005", "239.192.0.6:6006", "239.192.0.7:6007", "239.192.0.2:6002"],
    "toggleOff": ["239.192.0.1:6001", "239.192.0.3:6003"]
}'
```

Request URL

```
http://localhost:5000/roc/v1/command
```

Server response

Code	Details
200	Response body

```
[
  "Command not accepted (specify all ports)",
  400
]
```

Picture 14. Invalid command response body.

7 Conclusion

The purpose of thesis was to create a system for authenticating M2M MQTT messages transferred in a maritime environment using JWT. The connectivity between entities had to be secure for mitigating possible threats in critical infrastructure.

The highlight decision during this system's development process was the autonomous M2M communication approach. This means that administrators don't need to use the system daily. Only critical system parts are handled by humans such as root CA certificates and MQTT client certificates. The system itself is lightweight and does not require much room from the host OS. The base image of python slim combined with DRY programming approach saves storage room.

A system is as secure as its weakest link and in this solution, the weakest link is the basic authentication header required by the CA API. This should be replaced with multifactor authentication technology in the future to ensure that only peers with sufficient rights can access the dashboard for admins. Stricter security can be established by creating a whitelist of devices that can only connect to the service, meaning a software is going to block all the incoming traffic except the ones coming from a predefined source such as trusted IP address or software.

Possible future implementation could support more key encryption algorithms for different systems and have separate key pairs for encryption and message signing, creating an even more secure system of communication. All the methods could be tested using pytest or unittest libraries for python. This could lead to production quality assurance and facilitate the detection of breaking changes, whenever the codebase is updated.

References

[1] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” May 2008, doi: 10.17487/RFC5280.

[2] J. R. . Vacca, “Public key infrastructure : building trusted applications and Web services,” p. 404, 2004, Accessed: Nov. 05, 2024. [Online]. Available: https://books.google.com/books/about/Public_Key_Infrastructure.html?hl=fi&id=3kS8XDALWWYC

[3] K. Moriarty, B. Kaliski, J. Jonsson, and A. Rusch, “PKCS #1: RSA Cryptography Specifications Version 2.2,” Nov. 2016, doi: 10.17487/RFC8017.

[4] “Digital Signatures and PKI | Centre for Digital Public Infrastructure.” Accessed: Nov. 05, 2024. [Online]. Available: https://docs.cdpi.dev/technical-notes/electronic-signature-pki-and-trust-infra/digital-signatures-and-pki

[5] J. Schaad, “Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF),” Sep. 2005, doi: 10.17487/RFC4211.

[6] “How Certificate Chains Work.” Accessed: Nov. 05, 2024. [Online]. Available: https://knowledge.digicert.com/solution/how-certificate-chains-work

[7] “MQTT Essentials - All Core Concepts Explained.” Accessed: Nov. 06, 2024. [Online]. Available: https://www.hivemq.com/mqtt/

[8] “MQTT Version 5.0.” Accessed: Nov. 06, 2024. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

[9] “mqtt-publish-subscribe.png (1024×320).” Accessed: Nov. 06, 2024. [Online]. Available: https://mqtt.org/assets/img/mqtt-publish-subscribe.png

[10] “JSON Web Token Introduction - jwt.io.” Accessed: Nov. 06, 2024. [Online]. Available: https://jwt.io/introduction/

[11] “JSON Web Token (JWT).” Accessed: Nov. 06, 2024. [Online]. Available: https://www.iana.org/assignments/jwt/jwt.xhtml

[12] “Transport Layer Security - Wikipedia.” Accessed: Nov. 01, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Transport_Layer_Security

[13] “Google Cloud Platform - Wikipedia.” Accessed: Nov. 06, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Google_Cloud_Platform

[14] Docker, “Manuals | Docker Docs.” Accessed: Nov. 06, 2024. [Online]. Available: https://docs.docker.com/manuals/

[15] Nginx, “nginx.” Accessed: Nov. 06, 2024. [Online]. Available: https://nginx.org/en/

[16] “Welcome to pyca/cryptography — Cryptography 44.0.0.dev1 documentation.” Accessed: Nov. 06, 2024. [Online]. Available: https://cryptography.io/en/latest/

[17] E. Barker, “NIST Special Publication 800-57 Part 1 Revision 5 Recommendation for Key Management: Part 1-General”, doi: 10.6028/NIST.SP.800-57pt1r5.

[18] “Features - FastAPI.” Accessed: Nov. 06, 2024. [Online]. Available: https://fastapi.tiangolo.com/features/#tested

[19] “paho-mqtt · PyPI.” Accessed: Nov. 06, 2024. [Online]. Available: https://pypi.org/project/paho-mqtt/

[20] “Requests: HTTP for Humans™ — Requests 2.32.3 documentation.” Accessed: Nov. 07, 2024. [Online]. Available: https://requests.readthedocs.io/en/latest/

[21] Eclipse, “Eclipse Mosquitto.” Accessed: Nov. 06, 2024. [Online]. Available: https://mosquitto.org/