Device certificate authentication

Last updated: 03/03/2026

This content applies to the latest CD version of Cumulocity.

Specifications contained herein are subject to change and these changes will be reported in subsequent versions.

Copyright © 2026 Cumulocity GmbH.

The name Cumulocity GmbH and all Cumulocity GmbH product names are either trademarks or registered trademarks of Cumulocity GmbH and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

This software may include portions of third-party products. Third-party terms are set out in a 3rd-party-licenses file linked to or included with each installation package.

Table of Contents

Table of Contents 3

DEVICE CERTIFICATES 4

PREREQUISITES 4

General requirements for connecting devices with certificates 4

REGISTERING DEVICES USING CERTIFICATES 4

INTRODUCTION TO X.509 CERTIFICATES 5

GENERATING AND SIGNING CERTIFICATES 7

Creating a self-signed CA certificate 7

Creating an intermediate certificate 8

Creating a device certificate signed by CA or intermediate 9

Creating the chain of certificates 9

Creating keystore and truststore 9

KEYSTORE AND TRUSTSTORE 10

UPLOAD YOUR CA CERTIFICATE 10

PERFORM A PROOF OF POSSESSION 11

COMMON SECURITY SCENARIOS AND BEST PRACTICES 13

What to do when a device's certificate is compromised? 13

What to do when the enrollment process has been compromised? 13

What to do when a CA certificate has been compromised? 14

MANAGING TRUSTED CERTIFICATES 16

TO ADD A CERTIFICATE 17

TO EDIT A TRUSTED CERTIFICATE 18

TO RENEW A CA CERTIFICATE PROVIDED BY CUMULOCITY 18

TO DELETE A TRUSTED CERTIFICATE 18

MANAGING TRUSTED CERTIFICATE CRL SETTINGS 19

CRL SETTINGS 19

To enable online revocation 19

To enable offline revocation 20

Offline CRL bulk upload 21

To download or view the offline CRL file 21

CERTIFICATE AUTHORITY 22

PREREQUISITES 22

CREATING A CA CERTIFICATE VIA REST 23

CREATING A CA CERTIFICATE VIA THE UI 24

AUTO-RENEWAL OF CA CERTIFICATES 25

SIMPLE DEVICE ENROLLMENT 27

PREREQUISITES 27

ENROLL A DEVICE CERTIFICATE 27

RE-ENROLL A DEVICE CERTIFICATE 29

PROVISIONED CERTIFICATES 31

TO REVOKE A PROVISIONED CERTIFICATE 31

DEVICE CERTIFICATES

Devices can authenticate to Cumulocity using X.509 certificates over REST (port 8443), MQTT and LWM2M using mTLS.

Each tenant individually defines whom it trusts by uploading the base CA certificate as a trust anchor, these must be unique across the Cumulocity instance. Devices connecting to the platform with certificates do not need to provide the tenant ID, username and password; authentication information is obtained from the certificates.

HTTP

Until recently devices wanting to use mTLS over HTTP were required to generate JWT session tokens by using X.509 certificates for authentication over a defined MQTT endpoint - device access token API. This session token can be used in subsequent HTTP requests to authenticate to Cumulocity. This is still supported but will be deprecated at some point in the future. It is recommended to use the direct HTTP connection.

Core MQTT

Devices can communicate using the Core MQTT interface of the platform, but MQTT over WebSocket is not supported. The Cumulocity platform expects devices to connect using SSL on port 8883.

MQTT Service

Devices can communicate using the MQTT Service. The Cumulocity platform expects devices to connect using SSL on port 9883, and provide the tenant ID as the MQTT username.

LWM2M

For the LWM2M protocol you are required to provide the device endpoint ID in the Common Name of the certificate. This is used to identify the tenant to which the device is registered.

PREREQUISITES

In order to follow this tutorial, check if the following prerequisites are met:

- You have a valid tenant, user and password in order to access Cumulocity.
- The command line tool CURL is installed on your system.

General requirements for connecting devices with certificates

- The CA certificate may also be a self-signed certificate.
- Certificates must be uploaded as X.509 version 3 certificates.
- Uploaded certificates must have set `BasicConstraints:[CA:true]` .
- Devices must trust the Cumulocity server certificate.
- Certificates used by devices must contain the certificate chain that includes the uploaded CA certificate.
- If only the device certificate is provided, then the immediate issuer certificate must be uploaded to the platform's truststore.
- Certificates used by devices must be signed either by uploaded CA certificates or by intermediate certificates signed by uploaded CA certificates.

REGISTERING DEVICES USING CERTIFICATES

> **INFO**
>
> Device registration does not occur when using the MQTT Service.

Cumulocity supports two ways to register devices which will be able to connect using certificates:

Auto registration

The user for the device will be created during the first MQTT call, if a device certificate is derived from a trusted certificate which was uploaded to the Cumulocity platform with a flag *autoRegistrationEnabled* with a value of true. Auto-registration must be activated for the

uploaded certificate. If auto-registration is not activated it is required to use the bulk registration (see below). To manage the auto registration field of uploaded certificates in the UI refer to Managing trusted certificates.

The device_user will be created when the API is called for the first time, provided:

- The trusted certificate is uploaded to the platform.
- The auto registration is enabled.
- The device certificate is derived from a trusted certificate uploaded to the Cumulocity.

Bulk registration

The user for the device can also be created via the standard bulk registration in the Device Management application.

The CSV file used in bulk registration should meet the requirements described in Create a bulk device credentials request in the Cumulocity OpenAPI Specification. Moreover, it is required that the CSV file has an additional column AUTH_TYPE with value CERTIFICATES, and that the column CREDENTIALS is either not present or has an empty value.

Single registration

Single registration is not supported for devices which are going to use certificates for authentication.

> INFO
>
> During device registration, the device user is created, which is necessary for device communication with the platform.

INTRODUCTION TO X.509 CERTIFICATES

X.509 is a standard that defines public key certificates, which are commonly used in the SSL protocol to provide secure connection and data transfer. Version 3 is up-to-date since 1995.

The general purpose of an X.509 certificate is to bind an identity to a pair of keys: the public key is publicly known as a part of the certificate while the private key is known only by the certificate owner. Such pairs of keys should be created with an asymmetric-key algorithm, which is considered secure nowadays. An example of such an algorithm is RSA with at least 2048 bits key size. A size of 1024 bits or lower is no longer considered safe. A private key can be used in two ways:

- To prove that the message was sent by the certificate owner - The owner encrypts some message with his private key and sends it. Then the receiver can decrypt it with the sender's public key. If the decrypted message matches his expectation, then he can be sure that it was sent by the certificate owner.
- To read a message intended only for the certificate owner - If someone encrypts the message with receiver public key, then only the owner of the private key would be able to decrypt it. No third party who somehow intercepted the message will be able to read it.

Every certificate can be self-signed or can be signed by another certificate. To tell if the certificate is self-signed you can look at the "Issuer Name" and "Subject Name" fields of the certificate. If they are the same then it means that it is a self-signed certificate, otherwise the certificate claims to be signed by the issuer. To verify if the issuer really signed the certificate, you must check the "Signature" field of the certificate. After decryption with the issuer's public key, the signature should match the data of the signed certificate. Signing a certificate by another, means that if the issuer's certificate is trusted then the signed certificate also can be trusted.

For example if a platform trusts the customer certificate and that customer has 20 devices with individual certificates, then he does not have to upload each one of them.

If these device certificates are signed by the customer certificate, then the platform should trust them too. In this case, every device should send not only its own certificate, but the whole chain of certificates (so-called chain of trust) during the SSL handshake.

The chain of certificates starts with the one belonging to the device, through all used intermediate certificates until it reaches the CA certificate trusted by the platform. Usually the chain of certificates does not have to contain the trusted CA certificate, so it can end with the certificate signed directly by the CA. However, in the Cumulocity platform it is also required to provide the trusted CA certificate in the chain of certificates.

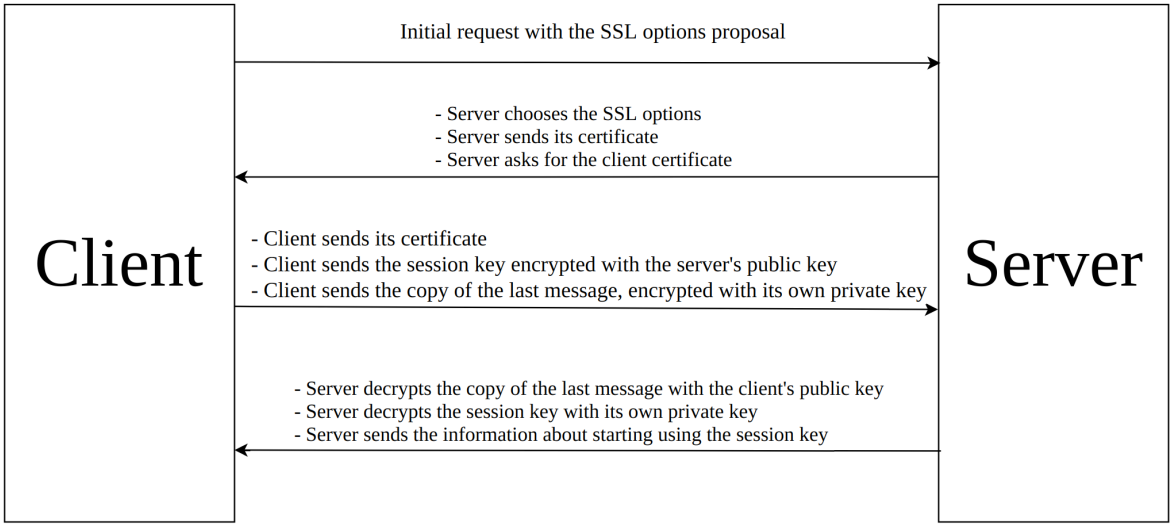
Providing the chain of certificates lets the platform verify the signatures of every certificate in the chain to make sure that the device certificate is signed directly or indirectly by the trusted certificate. The chain of the certificates can differ in length, so if the platform trusts certificate A and certificate B is signed by A, and certificate C is signed by B, then certificate C will also be trusted. However, there are a few things to keep in mind:

- Every certificate which is used to sign another certificate must contain the extension “CA:TRUE”.
- The lengths of the chain of certificates can be restricted by the certificate extension “pathlen”. This extension limits the amount of other CA certificates that can be placed in the chain between the device certificate and the one with that extension. For example the valid chain of the certificates with minimal values of path length would look like this: “A (CA:TRUE, pathlen: 2) -> B (CA:TRUE, pathlen: 1) -> C (CA:TRUE, pathlen: 0) -> D (device with CA:FALSE)””.

The structure of an X.509 certificate in version 3 looks like this:

- Version - The version number of the x.509 certificate,
- Serial Number - Unique serial number that is created for each certificate that is created by an issuer,
- Issuer - The distinguished name of the issuer,
- Not Before - Date since the certificate is valid,
- Not After - Date since the certificate is expired,
- Subject - Distinguished name of the certificate’s owner,
- Public Key Algorithm - Algorithm used to generate the public key,
- Subject Public Key - Public key value,
- Certificate Signature Algorithm - Algorithm used to generate the certificate signature,
- Certificate Signature - Certificate signature generated by encrypting certificate data with the issuer’s private key,
- Extensions - optional; they are responsible for providing various pieces of information, for example if the certificate is CA, which means that it can be used to sign another certificate. Added in version 3 of X.509,
- Issuer Unique Identifier - optional; can be present in the certificate to handle the possibility of reuse of issuer names,
- Subject Unique Identifier - optional; can be present in the certificate to handle the possibility of reuse of subject names.

To show how the authentication with X.509 certificates works, there is an example of the simplified mutual SSL handshake:



A server knows that the client is the owner of the certificate it sent, after the server has decrypted the encrypted copy of the message using the client’s public key. The client knows that the server is the owner of the certificate it sent, if the server uses the correct session key, because that means that the server decrypted it with its private key. In basic authentication with username and password, the password must be sent over the network to authenticate the user. When certificates are used, the private key is never sent, which makes the use of the certificates much more secure than the basic authentication with username and password.

There are 3 important terms related to X.509 certificates, which everyone should know who wants to start working with them:

- Keystore is a file which a device uses to authenticate itself in the mutual SSL handshake. That means that keystore contains the chain of the certificates used by the device and the private key of the device.
- Truststore is a file which contains all trusted certificates. A server or device would only establish a connection with something that is using a certificate from their truststore. When the chain of certificates is used, only one of the certificates in the chain must be trusted to establish the connection.
- Certificate Authority (CA) in the simplest sense is an entity that signs certificates.

Both keystore and truststore can be stored in the same file, but for security reasons we recommended you to keep them separately. The truststore contains public data and the keystore contains the private key, which only to the owner should know. The most popular formats for these files are:

- PKCS12 (Public Key Cryptography Standards, version 12), which can be generated using the OpenSSL toolkit.
- JKS (Java KeyStore), which can be generated with the Java Keytool.

GENERATING AND SIGNING CERTIFICATES

To generate certificates we use the OpenSSL toolkit. If you do not have it already installed, then you can download if from the website: https://www.openssl.org/source/

Creating a self-signed CA certificate

1. Create a directory for the root certificate and the signing configuration, for example:

   ```
   mkdir /home/user/Desktop/caCertificate
   ```
2. Go to the created directory and create a configuration file for your CA certificate:

   ```
   touch caConfig.cnf
   ```
3. Create a database file for keeping the history of certificates signed by the CA:

   ```
   touch database.txt
   ```
4. Create a serial file with initial serial number, which will be used to identify signed certificates. After assigning this serial to the signed certificate, the value in this file will be automatically incremented:

   ```
   echo 1000 > serial
   ```
5. Create subdirectories for signed certificates and the certificate revocation list:

   ```
   mkdir deviceCertificates crl
   ```
6. Fill in the configuration file. This is the example configuration, which can be used for tests after changing the directory `dir` to your own. If you want to use it in the production environment then please consult it first with some security specialist:

   ```
   [ ca ]
   default_ca = CA_default
   [ CA_default ]
   # Directory and file locations.
   dir             = /home/user/Desktop/caCertificate
   certs           = $dir # directory where the CA certificate will be stored.
   crl_dir         = $dir/crl # directory where the certificate revocation list will be stored.
   new_certs_dir   = $dir/deviceCertificates # directory where certificates signed by CA certificate will be stored.
   database        = $dir/database.txt # database file, where the history of the certificates signing operations will be stored.
   serial          = $dir/serial # directory to the file, which stores next value that will be assigned to signed certificate.

   # The CA key and CA certificate for signing other certificates.
   private_key     = $dir/caKey.pem # CA private key which will be used for signing certificates.
   certificate     = $dir/caCert.pem # CA certificate, which will be the issuer of signed certificate.

   default_md      = sha256 # hash function
   default_days    = 375 # default number of days for which the certificate will be valid since the date of its generation.
   preserve        = no # if set to 'no' then it will determine the same order of the distinguished name in every signed certificate.
   policy          = signing_policy # the name of the tag in this file that specifies the fields of the certificate. The fields must be filled in or even match the CA certificate values to be signed.

   # For certificate revocation lists.
   crl             = $crl_dir/caCrl.pem # CA certificate revocation list
   crlnumber       = $crl_dir/crlnumber # serial, but for the certificate revocation list
   crl_extensions  = crl_ext # the name of the tag in this file, which specifies certificates revocation list extensions, which will be added to the certificate revocation by default.
   default_crl_days  = 30 # default number of days for which the certificate revocation list will be valid since the date of its generation. After that date it should be updated to see if there are new entries on the list.

   [ req ]
   default_bits        = 4096 # default key size in bits.
   distinguished_name  = req_distinguished_name # the name of the tag in this file, which specifies certificates fields description during certificate creation and eventually set some default values.
   string_mask         = utf8only # permitted string type mask.
   default_md          = sha256 # hash function.
   x509_extensions     = v3_ca # the name of the tag in this file, which specifies certificates extensions, which will be added to the created certificate by default.

   # descriptions and default values of the created certificate fields.
   [ req_distinguished_name ]
   ```

```
countryName                 = Country Name (2 letter code)
stateOrProvinceName         = State or Province Name
localityName                = Locality Name
organizationName            = Organization Name
organizationalUnitName      = Organizational Unit Name
commonName                  = Common Name
emailAddress                = Email Address

# A default value for each field can be set by adding an extra line with field name and postfix "_default". For example: "countryName_default = PL". If you add this line here, then leaving country name empty during certificate creation will result in the value "PL" being used. If the default value was specified there, but during certificate creation you do not want to use this value, then instead use "." as the value. It will leave the value empty and not use the default.

# default extensions for the CA certificate.
[ v3_ca ]
subjectKeyIdentifier = hash # subject key value will be calculated using hash funtion. It's the recommended setting by PKIX.
authorityKeyIdentifier = keyid:always,issuer # The subject key identifier will be copied from the parent certificate. It's the recommended setting by PKIX.
basicConstraints = critical, CA:true, pathlen:10 # "critical" specifies that the extension is important and must be read by the platform. CA says if it is the CA certificate so it can be used to sign different certificates. "pathlen" specifies the maximum path length between this certificate and the device certificate in the chain of certificates during authentication. Path length is set here only to show how it is done. If you do not want to specify max path length, you can keep only the "basicConstraints = critical, CA:true" part here.
keyUsage = digitalSignature, cRLSign, keyCertSign # specifies permitted key usages.

# Default extensions for the device certificate. This tag is not used directly anywhere in this file, but will be used from the command line to create signed certificate with "-extensions v3_signed" parameter.
[ v3_signed ]
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer
basicConstraints = critical, CA:false
keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# default extensions for certificate revocation list
[ crl_ext ]
authorityKeyIdentifier=keyid:always

# Policy of certificates signing. It specifies which certificate fields must be filled in during certificate creation. There are three possible values here:
# "optional" - field value can be empty
# "supplied" - field value must be filled in
# "match" - signed certificate field value must match the CA certificate value to be created
[ signing_policy ]
countryName            = optional
stateOrProvinceName    = optional
organizationName       = optional
organizationalUnitName = optional
commonName             = supplied # every certificate should have a unique common name, so this value should not be changed.
emailAddress           = optional
```

7. Create a private key with aes256 encryption and a length of at least 2048 bits. You will also be asked to set the password for the key during its creation:

   ```
   openssl genrsa -aes256 -out caKey.pem 4096
   ```

8. Create a self-signed certificate using specifications from the configuration file. The “days” parameter says how long this certificate will be valid since the generation, so set it as you prefer:

   ```
   openssl req -config caConfig.cnf -key caKey.pem -new -x509 -days 7300 -sha256 -extensions v3_ca -out caCert.pem
   ```

9. You can print the created certificate with the command:

   ```
   openssl x509 -noout -text -in caCert.pem
   ```

Creating an intermediate certificate

The intermediate certificate is signed by the CA certificate, but will also be used to sign device certificates. This step is optional. If you are fine with signing all the device certificates with one common CA certificate, then you can skip this step. However, if you need some

certificates between the CA certificate and the device certificate then it is the way to go. Keep in mind that in the Cumulocity cloud the maximum length of the chain of certificates is currently restricted to 10. This behaviour can be changed for dedicated installations by changing a platform wide configuration setting and increasing (or decreasing) the allowed maximum length of the chain of certificates to more (or less) than 10. If you use a chain with a length greater than 2, we strongly recommend you to use the proof of possession feature to protect your service from DOS attacks.

To create the intermediate certificate:

1. Create a new directory for intermediate certificates inside the caCertificate path: `mkdir intermediateCertificate`
2. Go to this directory and create a configuration file for your intermediate certificate: `touch intermediateConfig.cnf`
3. Create a database file for keeping the history of certificates signed by the intermediate certificate: `touch database.txt`
4. Create a serial file with initial serial number, which will be used to identify signed certificates: `echo 1000 > serial`
5. Create subdirectories for the signed certificates and the certificate revocation list: `mkdir deviceCertificates crl`
6. Fill in the configuration file as in the CA configuration, but remember to change the general directory ("dir") to your intermediateCertificate folder. Also don't forget to change private_key, certificate and crl names from current with "ca" prefix to "intermediate" prefix (for example: caKey.pem -> intermediateKey.pem), because files with this prefix will be generated in the next steps.
7. Generate a private key for the intermediate certificate: `openssl genrsa -aes256 -out intermediateKey.pem 4096`
8. Generate a certificate signing request: `openssl req -config intermediateConfig.cnf -new -sha256 -key intermediateKey.pem -out intermediateCsr.pem`
9. Go to the caCertificate directory and generate a signed intermediate certificate. You must use the CA configuration here, because the private key specified in the configuration file will be used to sign the certificate: `openssl ca -config caConfig.cnf -extensions v3_ca -days 3650 -notext -md sha256 -in intermediateCertificate/intermediateCsr.pem -out intermediateCertificate/intermediateCert.pem`
10. Verify if the generated certificate is correctly signed by CA: `openssl verify -CAfile caCert.pem intermediateCertificate/intermediateCert.pem`

Creating a device certificate signed by CA or intermediate

1. Go to the directory of your caCertificate or intermediateCertificate depending on which one is used to sign the device certificate.
2. Generate the private key for the new certificate: `openssl genrsa -aes256 -out deviceCertificates/deviceKey.pem 4096`
3. Generate the certificate signing request (change "caConfig.cnf" to "intermediateConfig.cnf" if you are in the intermediateCertificate directory): `openssl req -config caConfig.cnf -new -sha256 -key deviceCertificates/deviceKey.pem -out deviceCertificates/deviceCsr.pem` Remember that the `commonName` of the device certificate, which you will be asked to provide in the console, must match the ClientId of the device during the connection.
4. Generate the certificate signed by the CA or intermediate (change "caConfig.cnf" to "intermediateConfig.cnf" if you are in the intermediateCertificate directory): `openssl ca -config caConfig.cnf -extensions v3_signed -days 365 -notext -md sha256 -in deviceCertificates/deviceCsr.pem -out deviceCertificates/deviceCert.pem`
5. Verify if the generated certificate is correctly signed by CA or intermediate (change "caCert.pem" to "intermediateCert.pem" if you are in the intermediateCertificate directory): `openssl verify -partial_chain -CAfile caCert.pem deviceCertificates/deviceCert.pem`

Creating the chain of certificates

Go into your caCertificate directory.

If you created the CA certificate, which was used to sign the intermediate certificate and then the intermediate certificate was used to sign the device certificate, then you create your chain with the command:

```
cat intermediateCertificate/deviceCertificates/deviceCert.pem intermediateCertificate/intermediateCert.pem caCert.pem >
intermediateCertificate/deviceCertificates/deviceCertChain.pem
```

If you are not using the intermediate certificate then the command is:

```
cat deviceCertificates/deviceCert.pem caCert.pem > deviceCertificates/deviceCertChain.pem
```

If you are using multiple intermediate certificates between the CA certificate and the device certificate, then remember that you must keep the correct order during the chain creation (Every certificate must be followed by the certificate, which it is signed by).

Creating keystore and truststore

1. Go into your deviceCertificates directory with the device's private key and the generated chain of certificates. If you are using an

intermediate certificate between the CA certificate and the device certificate then it will be the `caCertificate/intermediateCertificate/deviceCertificates` path, otherwise it will be `caCertificate/deviceCertificates` . Create keystore using the generated chain of certificates and the private key of the device: `openssl pkcs12 -export -name devicekeyentry -inkey deviceKey.pem -in deviceCertChain.pem -out deviceKeystore.pkcs12`

2. If you want to convert your keystore to JKS format then you would need the Java Keytool which is usually downloaded together with Java Development Kit: `keytool -importkeystore -srckeystore deviceKeystore.pkcs12 -srcstoretype PKCS12 -destkeystore deviceKeystore.jks -deststoretype JKS`
3. If you do not have the server certificate, get it by the command: `openssl s_client -showcerts -connect <cumulocity url>:<mqtt mutual ssl port (currently 8883, but that can be changed in the future)> | openssl x509 -outform PEM > serverCertificate.pem`
4. Now you can create a truststore, which will contain the server certificate. It must be created with Java Keytool (openssl does not support creating truststore, so if you don't want to use Java keytool then you must keep every trusted certificate in a separate PEM file). Remember that `alias` is the unique identifier for every keystore or truststore entry. It means that if you want to add a second trusted certificate to the same truststore then you must change the alias from `servercertificate` in the command below to some other name:
   - In PKCS12 format: `keytool -importcert -noprompt -keystore deviceTruststore.pkcs12 -alias servercertificate -file serverCertificate.pem`
   - In JKS format: `keytool -import -file serverCertificate.pem -alias servercertificate -keystore deviceTruststore.jks`
5. Optionally, instead of creating a new file for the truststore, you can add the trusted certificates to your created keystore and store everything in one file, which is not the recommended solution:
   - If your keystore is in the PKCS12 format: `keytool -importcert -noprompt -keystore deviceKeystore.pkcs12 -alias servercertificate -file serverCertificate.pem`
   - If your keystore is in the JKS format: `keytool -import -file serverCertificate.pem -alias servercertificate -keystore deviceKeystore.jks`
6. You can check the content of your keystore (or truststore) with the command: `keytool -list -v -keystore deviceKeystore.jks`

KEYSTORE AND TRUSTSTORE

Generate a keystore and a truststore as described in Generating and signing certificates if you didn't do it yet.

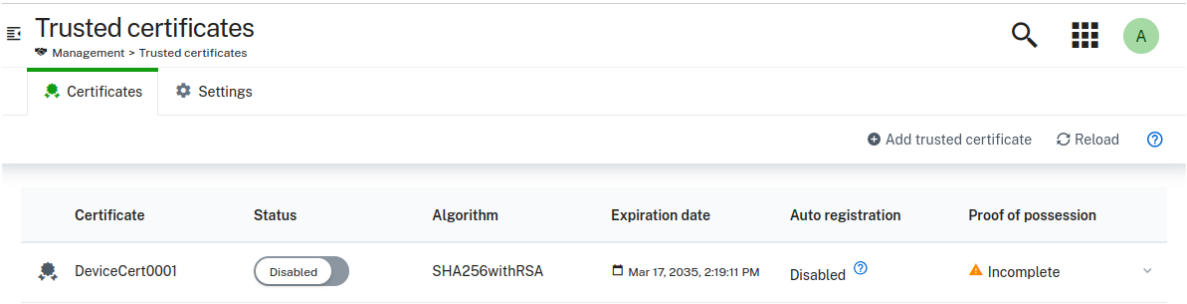
UPLOAD YOUR CA CERTIFICATE

Upload your CA (or intermediate) certificate to the platform. This operation will add your uploaded certificate to the server's truststore. It can be done in two ways, both of which have a role requirement of either ROLE_TENANT_ADMIN or ROLE_TENANT_MANAGEMENT_ADMIN:

Via UI:

1. In the Device Management application, navigate to the **Management** menu in the navigator and select **Trusted certificates**.
2. In the resulting dialog, enter a custom name for the new certificate.
3. Drop your CA certificate (caCert.pem or intermediateCert.pem).
4. Select the **Auto registration** check box.
5. Set the toggle to **Enabled**.
6. Click **Add certificate**.

Then new certificate will be added to the trusted certificates list:



Via REST:

1. Display your CA (or intermediate) certificate, which you want to upload to the Cumulocity platform and copy its PEM value, which starts with "—BEGIN CERTIFICATE—" and ends with "—END CERTIFICATE—" (including the hyphens). Remove new line symbols (`\n`) if they were added automatically at the end of each line: `openssl x509 -in caCert.pem -text`
2. Send it to the platform via POST request:

```
POST /tenant/tenants/<TENANT_ID>/trusted-certificates
Host: https://<TENANT_DOMAIN>/
Authorization: Basic <YOUR_AUTHENTICATION>
Content-Type: application/json
{
 "status" :  "ENABLED",
 "name" : "certificateName",
 "autoRegistrationEnabled" : "true",
 "certInPemFormat" : "<CERT_PEM_VALUE>"
}
```

PERFORM A PROOF OF POSSESSION

Cumulocity platform uses X.509 certificates to authenticate end devices. The certificates work with a chain of trust: you can create trustworthy subcertificates with a trusted certificate. Each certificate consists of a public and a private part. Also see asymmetric encryption.

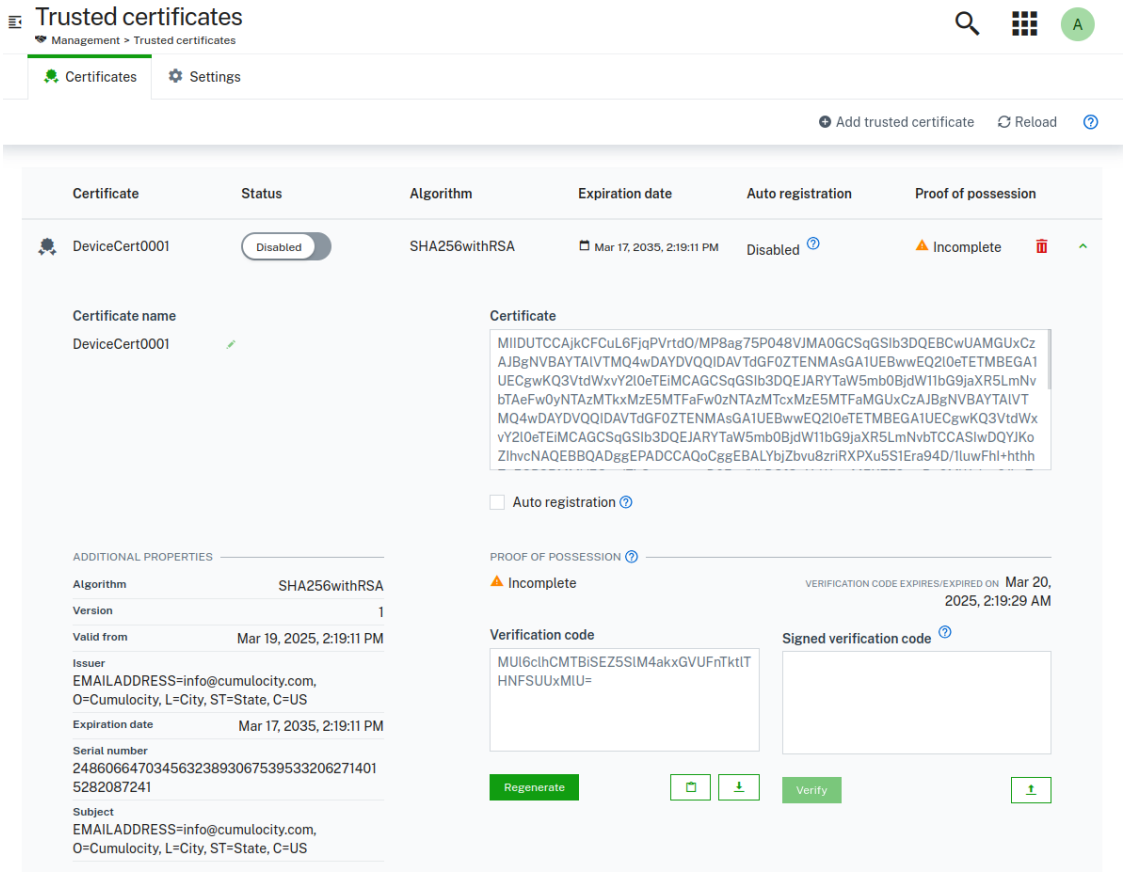
Cumulocity platform receives the public part of each certificate that is to be used for device authentication. The assignment of the device to a tenant is also done by the certificate, since each certificate must be uniquely assigned. Performing the proof of possession steps filters out all certificates without prior proof of possession, giving preference to tenant mappings of certificates with a validated proof of possession.

However, since the public part of a certificate (and the subcertificates) is not secret, anyone on the internet theoretically has access to it. A potential attacker could upload the public part of a certificate to Cumulocity platform even if he does not have access to the private part of the certificate (thus not being the owner of the certificate). In this case the Cumulocity platform cannot decide which uploader is the legitimate one, so the platform does not accept any reference to this certificate as valid, which would result in a DOS scenario.

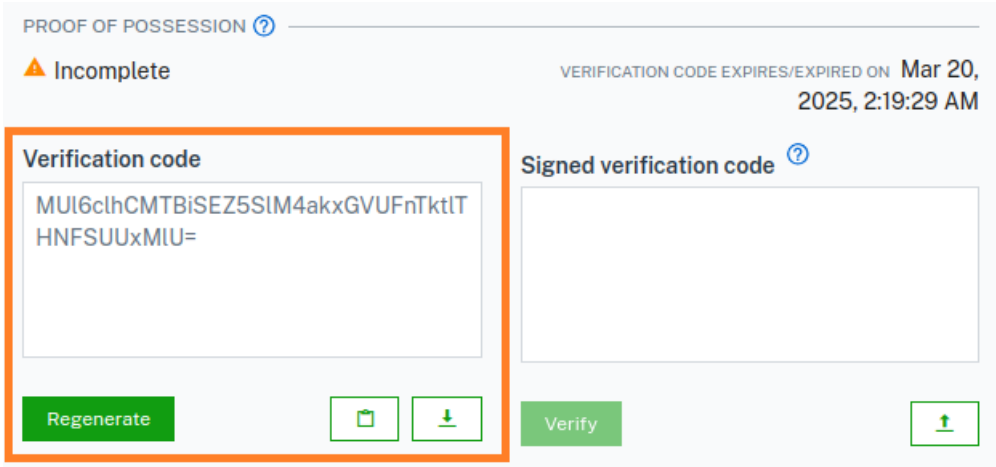
To ensure verification of ownership by the uploader, a proof of possession is required by the platform.

The steps for the proof of possession are as follows:

1. Navigate to **Management** > **Trusted certificates** in the Device Management application and verify that the certificate has been uploaded properly.



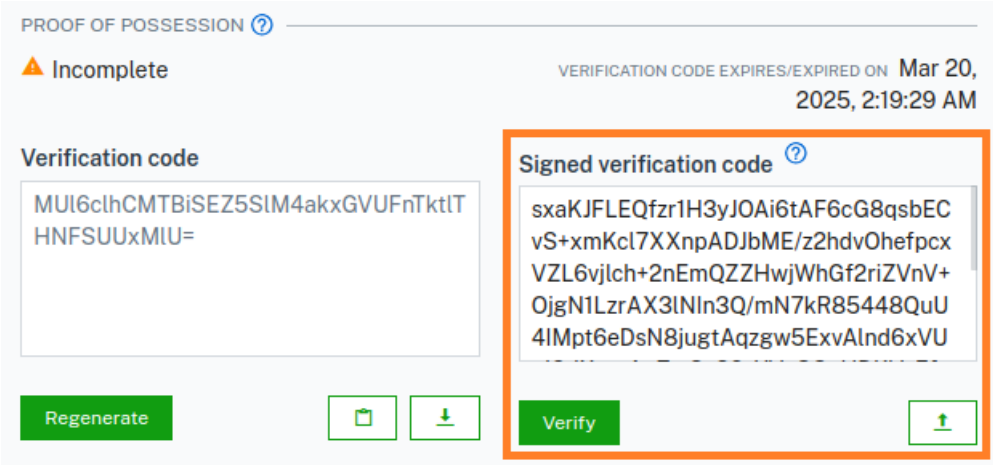
2. In the **Proof of Possession** section of the certificate details, download the verification code.



3. Encrypt the verification code using the private key of the certificate to produce the signed verification code. Use the following OpenSSL command:

```
openssl dgst -sha256 -sign <private.key> <verification_code.txt> | openssl base64 -A
```

4. Upload the signed verification code to the platform.



The proof of possession is confirmed if the uploaded signed verification code matches the signed verification code expected by the platform. This is indicated by switching the state from “Incomplete” to “Complete” in the **Proof of Possession** section.

> **INFO**
>
> If administrators cannot carry out this process on their own for organizational reasons, they can manually request the proof of possession for the corresponding certificate and the Cumulocity Support team can complete the proof of possession through a back end API upon reasonable verification.

COMMON SECURITY SCENARIOS AND BEST PRACTICES

What to do when a device’s certificate is compromised?

If you suspect or confirm that a device certificate has been compromised, follow the steps below:

1. **Revoke the compromised certificate**
   - Add the certificate to the certificate revocation list maintained by Cumulocity.
2. **Disable the device user**
   - This will disconnect the device from the Cumulocity but leave the certificate untouched until an investigation can be done.
   - If not compromised: Re-enable the device user. The device will reconnect using its existing certificate.
   - If compromised: Revoke the certificate to prevent any further use of it.
3. **Provision a new certificate**
   - Generate a new certificate for the device.
   - Using the existing CA certificate which is owned by Cumulocity generate a new certificate for the device.
   - Update the device to use the new certificate for authentication.

What to do when the enrollment process has been compromised?

Enrollment processes may be compromised when insecure practices are used, for example:

- Using a device ID-derived one-time password (OTP)
- Using the same OTP across multiple enrollment requests

If such a situation occurs, follow these steps to mitigate the risk:

1. **Disable auto-registration on the tenant’s CA certificate**
   - This immediately stops any new automatic registrations.
   - Existing registered devices will remain connected and operational.
2. **Communicate and remediate unsafe procedures**
   - Inform all stakeholders about the issue.

- If necessary, disable users who were following unsafe enrollment practices.
- Provide guidance on secure manual enrollment procedures.

3. **Investigate potential misuse of OTPs**

   - When attempting to enroll a new device, you may see the following message:

     > *"No newDeviceRequest found for this ID. It may already be registered, check the audit logs for details."*

   - This indicates that the OTP may have already been used. To verify:

     - **Check the audit logs**
       Look for an audit record with:
       - **Activity:** `Tenant certificate authority (CA) signed certificate for device`
       - **Type:** `TenantCertificateAuthority`
       - **Text:** containing `Certificate serial number hex: '%s'`
     - **Verify the signer**
       Confirm whether the certificate was signed by an authorized device.
       - If it **was not** signed by an authorized user, it may indicate fraudulent or unintended device registration.
     - **Revoke unauthorized certificates**
       Use the certificate revocation list to revoke any certificates that were not properly authorized. This prevents unauthorized devices from being accepted as trusted.

4. **Re-enable auto-registration**

   - Once secure procedures have been confirmed and enforced, re-enable auto-registration.
   - This ensures new enrollment requests are processed safely.

> **INFO**
>
> Consider rotating enrollment credentials periodically and monitoring logs for unusual enrollment activity.

What to do when a CA certificate has been compromised?

If you suspect or confirm that a CA (Certificate Authority) certificate has been compromised, follow the steps below:

1. **Revoke the compromised CA certificate**
   - Add the compromised CA certificate to the certificate revocation list using the CRL API.
   - Note: This action is effective only if the tenant has offline CRL check enabled in the Cumulocity. If offline CRL check is disabled, the recommended solution here is to use the disable CA certificate functionality.
   - The CA certificate itself remains in the system until the next step.
2. **Delete the compromised CA certificate**
   - Remove the compromised CA certificate from the tenant.
   - This action also deletes the associated key pair, ensuring it cannot be reused.
3. **Create a new CA certificate**
   - Generate a new CA certificate.
   - A new key pair will be created automatically during this process.
   - Use this new CA to issue device certificates going forward.
4. **Reprovision devices**
   - All device certificates previously signed by the compromised CA certificate will no longer be trusted by the Cumulocity.
   - Devices using these certificates will be unable to connect via any communication channel, such as MQTT or deviceAccessToken API.
   - Re-enroll affected devices using certificates issued by the new CA.

> **INFO**
>
> After replacing a compromised CA certificate, review and update your certificate management and enrollment policies to ensure stronger security practices, including periodic CA rotation and continuous monitoring for

unusual certificate activity.

MANAGING TRUSTED CERTIFICATES

Cumulocity allows devices to connect via MQTT protocol using a X.509 certificate for authentication. To do so, a certificate must be trusted by Cumulocity. A certificate is trusted when it is added to the trusted certificates and is in activated state.

REQUIREMENTS

ROLES & PERMISSIONS:

To view trusted certificates, you must be the initial admin user (first created user) of the tenant or have one of the following permissions:

- READ permission for the “Tenant management” permission type.

To manage trusted certificates, you must be the initial admin user (first created user) of the tenant or have one of the following permissions:

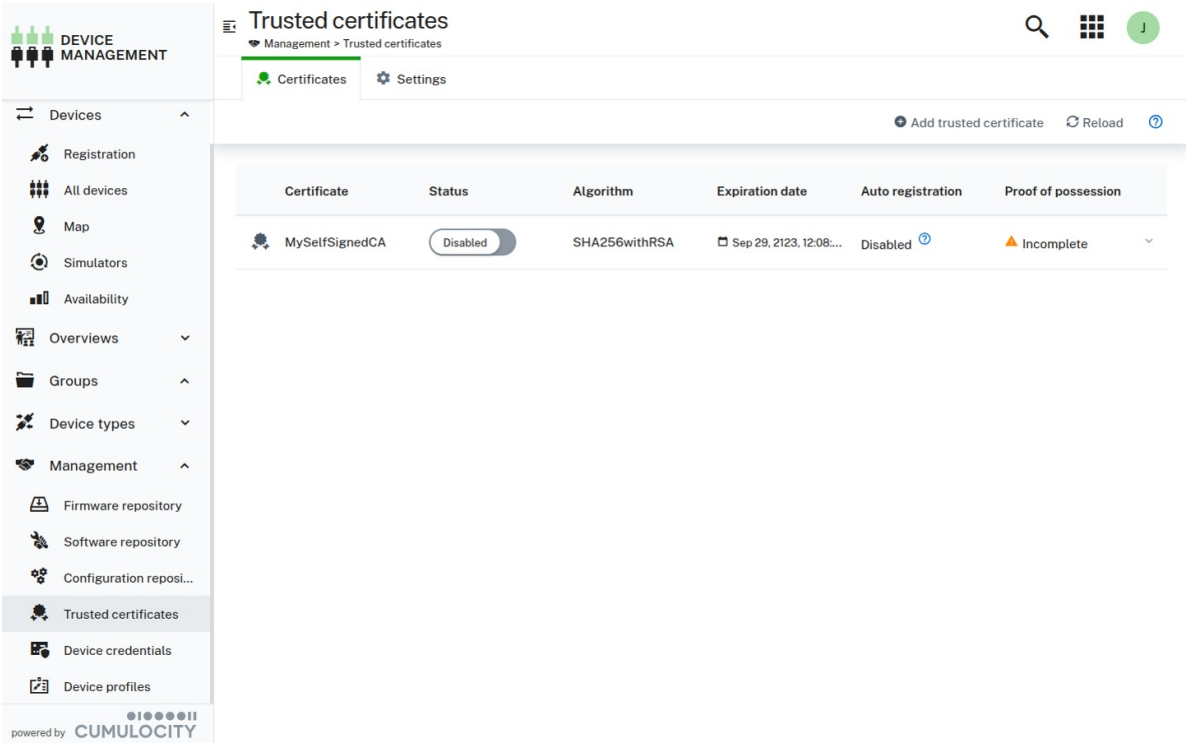
- ADMIN permission for the “Tenant management” permission type.

INFO

This section describes how to manage trusted certificates. For information on connecting devices using certificates refer to Device certificates.

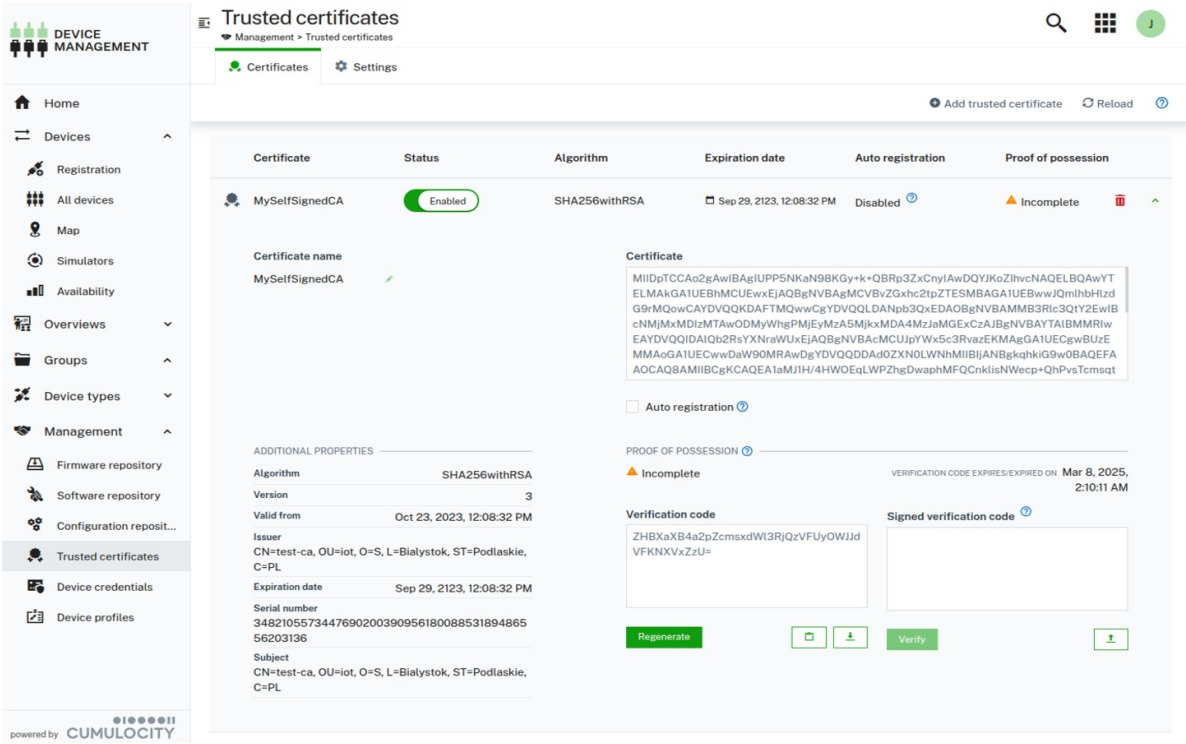
Click **Trusted certificates** in the **Management** menu in the navigator.

All certificates owned by the tenant will be displayed.



The **Status** column indicates if the certificate is enabled or disabled. At any given time a tenant can have any number of enabled or disabled certificates. Expand a certificate by clicking the arrow icon ⌄ at the right to view more details.

The information in the table at the right side is extracted from the provided certificate. The content is read-only and cannot be changed.



TO ADD A CERTIFICATE

Before adding a new trusted certificate, make sure that:

- It is a X.509 certificate in PEM format.
- It is in version 3.
- It contains `BasicConstraints:[CA:true]` .
- It has not already been uploaded to Cumulocity.
- It is still valid (not expired).

To add a certificate perform these steps:

1. Click **Add trusted certificate** at the right of the top menu bar.
2. In the resulting dialog box, provide the following information:

Field	Description
Certificate name	User-provided name for the certificate. This name is not used by Cumulocity and can serve as a description of the certificate.
Certificate	File containing the certificate in PEM format. Add the file by dropping it into this field or browsing for it in your file system.
Auto registration	If selected, new devices which use a certificate signed by the authority owning this trusted certificate will automatically be registered. The option does not support devices using the LWM2M protocol.

Field	Description
Enabled/ Disabled	When disabled, devices which use a certificate signed by the authority owning this certificate, will not be able to connect.

3. Click **Add Certificate** to validate and save the certificate.

INFO

For performance reasons, you shouldn't add the certificates of each device you want to connect, but only add the root certificate or one of the intermediate certificates from the chain which has been used to sign certificates used by devices.

TO EDIT A TRUSTED CERTIFICATE

In the detail view of a certificate you may change the parameters on the left, that is, the certificate name, and the settings for the auto registration and enabled/disabled option.

For details on the fields, see the description on adding certificates above.

TO RENEW A CA CERTIFICATE PROVIDED BY CUMULOCITY

Certificate renewal enables you to generate a new Certificate Authority (CA) certificate provided by Cumulocity before the existing one expires. The connected devices may need a moment to reconnect after the renewal, but it prevents sudden authentication failures in case the current certificate reaches the end of its validity period.

A **Renew certificate** button appears for a certificate in the certificate list when the following conditions are met:

- The certificate is a CA certificate.
- The certificate expires within 2 years from the current date.

To renew a CA certificate perform these steps:

1. In the **Trusted certificates** list, find the certificate you want to renew.
2. Click the refresh icon next to its current expiration date.

INFO

When you renew a certificate, a new certificate with updated expiration dates is generated and it becomes active immediately, and can be used for device authentication.

TO DELETE A TRUSTED CERTIFICATE

To permanently delete a certificate from the trusted certificates list, click the delete icon at the right of the respective entry and in the context menu click **Delete**. The certificate will be permanently deleted.

MANAGING TRUSTED CERTIFICATE CRL SETTINGS

Cumulocity allows administrators to fine-tune various certificate configurations. Currently, Certificate Revocation List (CRL) is supported.

REQUIREMENTS

ROLES & PERMISSIONS:

To view trusted certificates, you must be the initial admin user (first created user) of the tenant or have one of the following permissions:

- READ permission for the “Tenant management” permission type.

To manage trusted certificates, you must be the initial admin user (first created user) of the tenant or have one of the following permissions:

- ADMIN permission for the “Tenant management” permission type.

INFO

This section targets at advanced users for granular control over certificate attributes like CRL, OCSP, Certificate Policy and PKI. If you are unfamiliar with these terms it’s advisable to leave the defaults. Bad configuration might result in changes which cannot be reversed.

CRL SETTINGS

In general, Certificate Revocation List (CRL) contains a list of serial numbers of revoked certificates. These are issued by Certificate Authorities (CAs) periodically and published through an endpoint called CRL Distribution Point (CDP). Revocation reasons and dates are included in the CRL.

In Cumulocity terms, if there is a breach at device certificates signed by a trust anchor, users can inform the platform about the revoked certificates in two ways:

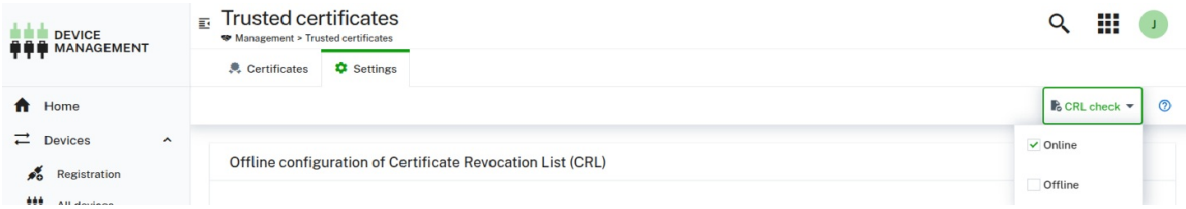
1. Online revocation : To perform online revocation checks, the device administrator is expected to upload a trusted CA which maintains the CRLs with the list of revoked certificate serial numbers and should provide this information in its CDP attribute.
2. Offline revocation : The trusted CA doesn’t maintain the revocation details of the certificates, in which case, the platform users can manually upload the serial numbers of the revoked certificates.

Note that both online and offline CRL checks are by default **unchecked**.

IMPORTANT

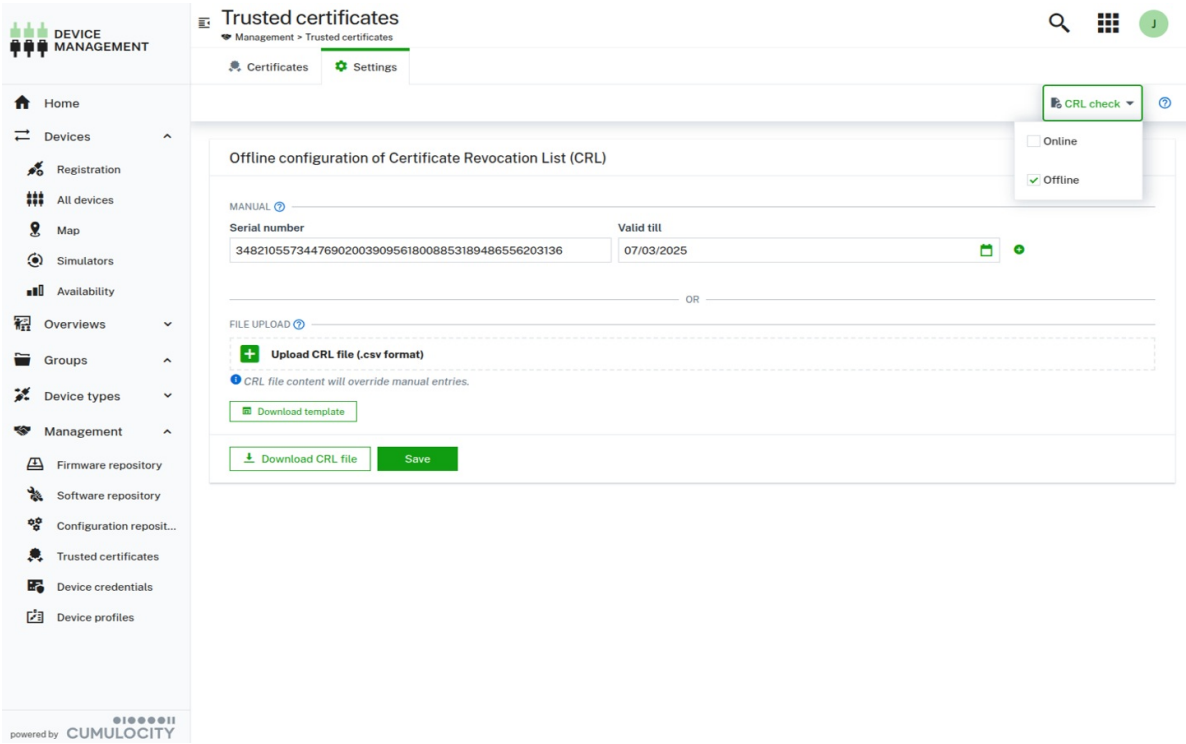
Certificate revocation is an irreversible process. So, offline entries once made cannot be removed.

To enable online revocation



1. Click **CRL check** in the upper right corner of the screen.
2. Enable the **Online** option.

To enable offline revocation



1. Click **CRL check** in the upper right corner to enable the offline revocation of the screen.
2. Check the **Offline** checkbox.
3. To add revoked certificate serial numbers manually, enter the serial number and date in the **Revoked certificates list** panel.

   Click ⊕ to add new entries in the format below:

Field	Description	Example
Serial number	Must be a hexadecimal value.	0b8a5b9dd501a88775399b9a048811a3
Date (optional)	Date format: `yyyy-MM-dd'T'HH:mm:ss.SSS'Z'` .	2024-01-17T10:48:51.000Z

4. 4. Click **Save** to confirm your entries.

INFO

If the date field is empty or a future date, then the current date is used.

Offline CRL bulk upload

INFO

Revoked certificate serial numbers can be added in bulk. Each file can hold at maximum 5000 revocation entries. If the date is in the future then it defaults to the current date. In case of a duplicate, the existing entry is retained.

To bulk upload, follow these steps:

1. Download the CSV template from the **Revoked certificates list** panel.
2. Fill in all revoked certificate serial numbers and revocation dates.
3. Upload the filled CSV file using file upload.

To download or view the offline CRL file

In the **Revoked certificates list** panel, click **Download CRL file**.

Device certificate authentication › Managing trusted certificate CRL settings

CERTIFICATE AUTHORITY

Cumulocity includes a Certificate Authority (CA), providing the following capabilities:

- Manage signing certificates
- Accept Certificate Signing Requests (CSR)
- Perform legitimacy checks as defined by each tenant
- Issue signed X.509 certificates trusted by the device tenant

The Cumulocity CA service is based on the EST protocol due to its simple interactions between devices and the CA service. The following REST API endpoints support the provisioning and renewal of device certificates.

- `/.well-known/est/simpleenroll` to be used by a device to get a fresh new certificate. The device has to authenticate itself using its tenant, identifier and security token as the BasicAuth realm, user and password. These tenant, identifier and security token must be shared with Cumulocity.
- `/.well-known/est/simplereenroll` to be used by a device to renew its certificate or to substitute for a certificate. The device has to authenticate itself using its password or a JWT token (obtained using its certificate over MQTT).

> ⚠ CAUTION
>
> Migration from Public Preview to General Availability - action required.
>
> As part of the move to General Availability you need to remove all the devices you have registered under Public Preview and re-register them. All such devices will continue to be able to connect, but none of the other capabilities of the certificate lifecycle management will be available until they are re-registered.

The Cumulocity certificate management allows Cumulocity to sign and issue certificates.

The Cumulocity signed certificates are shown in the list of certificate authority (CA) certificates for a tenant along with the trust anchor certificates. In this list, the Cumulocity signed certificates are identifiable by the words TENANT CA.

This section outlines how to create a Certificate Authority (CA) for a tenant with Cumulocity.

PREREQUISITES

To use the Certificate Authority API, this feature must be enabled at the tenant level. By default, it is enabled. You can verify whether the feature is enabled in your tenant using the following API:

```
GET /features/certificate-authority
Content-Type: application/json
Authorization: Basic <<Base64 encoded bootstrap credentials>>
```

If you get 200 with `active: false` then the feature is disabled for the tenant.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: ...
{
  "phase":"PUBLIC_PREVIEW",
  "active":false,
  "strategy":"TENANT",
  "key":"certificate-authority"
}
```

There are two ways to enable the feature in your tenant.

Tenant Administrators who have the role `ROLE_TENANT_MANAGEMENT_ADMIN` can use the following API:

```
PUT /features/certificate-authority/by-tenant
Content-Type: application/json
Authorization: Basic <<Base64 encoded bootstrap credentials>>
...
{
  "active": true
}
```

Operations team personnel, who have access to the Management tenant using a user with the role `ROLE_TENANT_MANAGEMENT_ADMIN`, can enable the feature in any subtenant using the following API:

```
PUT /features/certificate-authority/by-tenant/{{tenantId}}
Content-Type: application/json
Authorization: Basic <<Base64 encoded bootstrap credentials>>
...
{
  "active": true
}
```

CREATING A CA CERTIFICATE VIA REST

To create a new CA certificate for a tenant the Certificate Authority API must be called. This triggers the following actions:

- A new key pair is created and a certificate is self-signed with `tenantID` as the Common Name (CN).
- The private key is stored in an encrypted tenant option.
- The certificate is stored in the trusted certificate repository with auto-registration turned on by default. Devices can only be registered automatically when the administrator turns this option on.
- If a CA is already present, a respective message is returned.
- If the CA is removed from the trusted certificate list, the corresponding private key is removed from the database collection.

This is an example of a REST request:

```
POST /certificate-authority
Content-Type: application/json
Authorization: Basic <<Base64 encoded bootstrap credentials>>
```

The following response is returned:

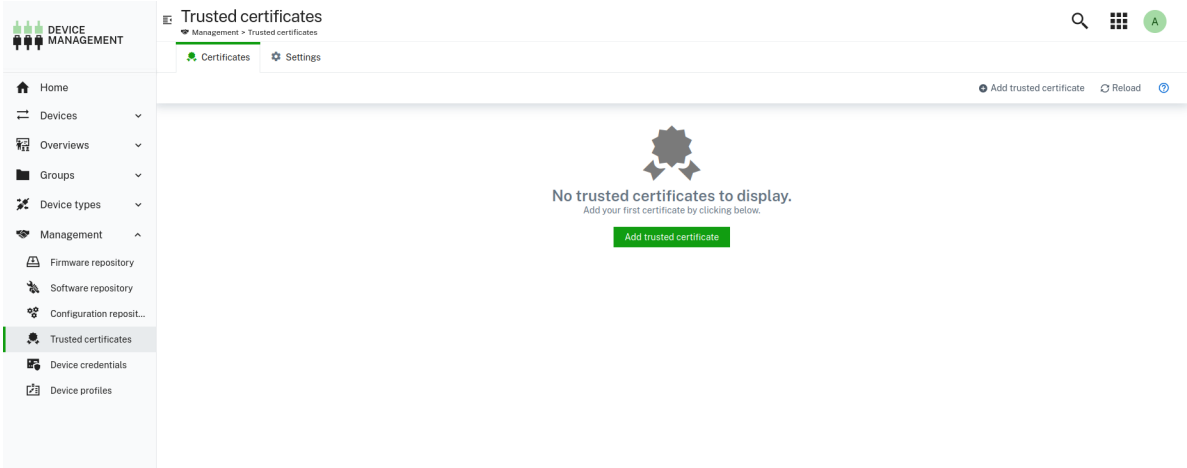
```
HTTP/1.1 201 OK
Content-Type: application/json
Content-Length: ...
{
   ......
   "subject":"CN={tenantId}, O=Cumulocity",
   "tenantCertificateAuthority":true,
   "autoRegistrationEnabled":true,
   "status":"ENABLED"
   ....
}
```

This certificate is identified as a TENANT CA and it has the attribute `"tenantCertificateAuthority":true`.

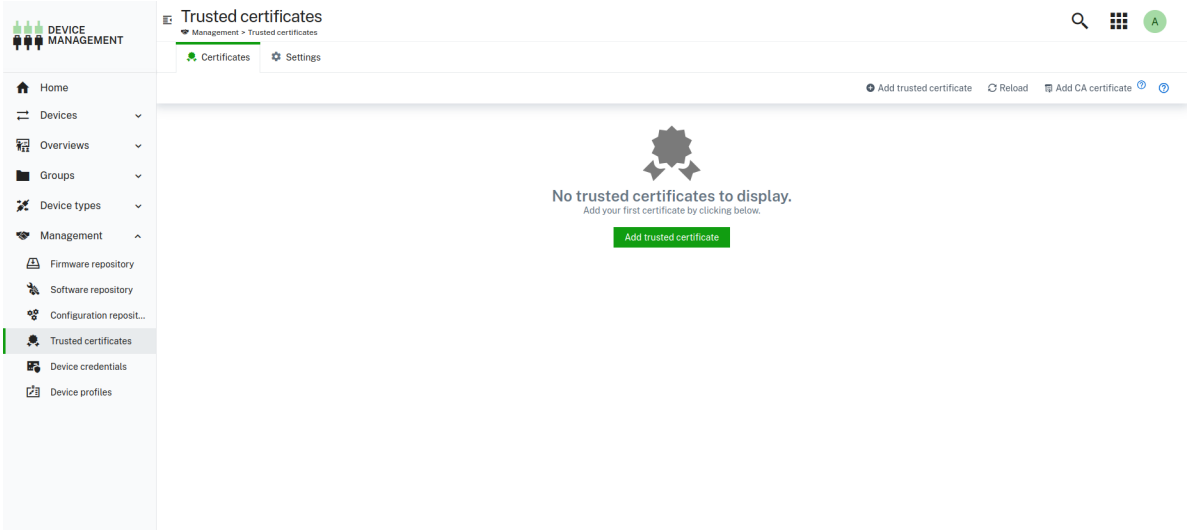
> **INFO**
>
> To call `/certificate-authority`, your role must include ADMIN permission for “Tenant” (API string = `ROLE_TENANT_ADMIN`) or “Tenant management” (API string = `ROLE_TENANT_MANAGEMENT_ADMIN`). Without this, the request will fail (HTTP 403 error).

CREATING A CA CERTIFICATE VIA THE UI

If the Certificate Authority feature is disabled the button **Add CA certificate** is not visible.

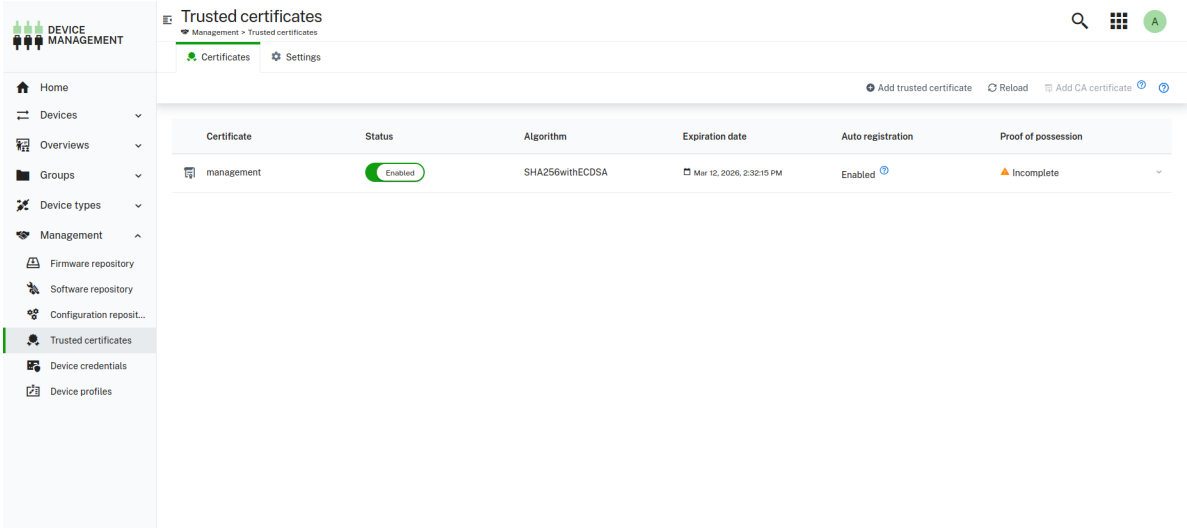


Follow the instructions in Prerequisites to enable it. After enabling the feature, the UI will show the button **Add CA certificate**.



1. In the Device Management application, navigate to the **Management** menu in the navigator and select **Trusted certificates**.
2. Click **Add CA certificate** at the top right to create the CA certificate.
3. Once the CA has been created the **Add CA certificate** button and the API are both disabled as only one CA is allowed per tenant. If you want to replace the certificate you need to remove the current CA.

The new CA certificate will be added to the trusted certificates list:



AUTO-RENEWAL OF CA CERTIFICATES

Tenant Certificate Authority (CA) is automatically renewed on 2 October at 02:00 AM every year. The renewal process ensures that existing device certificates remain valid until their expiration. This auto-renewal mechanism ensures uninterrupted certificate management while maintaining security and compliance. If automatic renewal fails, the renewal can also be performed via API, but only if the current Certificate Authority (CA) has less than 24 months remaining before expiration.

This is an example of a REST request:

```
POST /certificate-authority/renew
Content-Type: application/json
Authorization: Basic <<Base64 encoded bootstrap credentials>>
```

The following response is returned:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: ...
{
  ......
  "subject":"CN={tenantId}, O=Cumulocity",
  "tenantCertificateAuthority":true,
  "autoRegistrationEnabled":true,
  "status":"ENABLED"
  ....
}
```

> **INFO**
>
> In order to call `/certificate-authority/renew` one of the following roles is required: ROLE_TENANT_MANAGEMENT_ADMIN or ROLE_TENANT_ADMIN. Otherwise, an HTTP response 403 is returned.

- Each CA certificate has a validity of 1095 days (3 years) and undergoes automatic renewal in the background.
- All CA metadata, private keys, and public keys remain unchanged, ensuring a seamless renewal process. Only `NotAfter` and `NotBefore` wil be changed.
- Device certificates issued by the CA continue to have 1 year validity from issuance date, and new device certificates can be

issued without disruption.

- If a CA certificate has a near expiration date, then an error banner will be displayed in the UI for this certificate.
- An audit log is generated both when the CA certificate is refreshed and when a refresh is determined to be not yet required.

Device certificate authentication › Certificate Authority

SIMPLE DEVICE ENROLLMENT

In this tutorial, you will learn how to create a device certificate which is signed by a tenant's Certificate Authority (CA) with Cumulocity using the existing bulk device registration.

To connect larger amounts of devices, Cumulocity offers the option to bulk-register devices, that means, to register larger amounts of devices by uploading a CSV file. The CSV files must contain at least the **ID** as device identifier, **AUTH_TYPE** as `CERTIFICATES` and **ENROLLMENT_OTP** as one time password of the devices. In addition to these columns the file can also contain other columns like ICCID, NAME, TYPE as shown in the following example:

```
ID;TYPE;NAME;ICCID;IDTYPE;PATH;SHELL;AUTH_TYPE;ENROLLMENT_OTP
006064ce800a;c8y_Device;Sample_Device1;+491555555;c8y_Serial;bulk group/subgroup1;1;CERTIFICATES;somePassword@123
006064ce8077;c8y_Device;Sample_Device2;+491555555;c8y_Serial;bulk group/subgroup2;1;CERTIFICATES;password@345
```

- These EST Enrollment Over Secure Transport devices will be added to the Bulk devices registration list, with their status set to Accepted.
- These enrollment OTPs (value of column `ENROLLMENT_OTP`) act as temporary device user credentials, which are used for authentication when calling the `/.well-known/est/simpleenroll` endpoint.
- This enhancement simplifies the enrollment process for multiple devices, ensuring a seamless and secure onboarding experience.

PREREQUISITES

In order to follow this tutorial, check the following prerequisites:

- You have a valid tenant, a user and a password in order to access Cumulocity.
- A Certificate Authority (CA) has already been created for this tenant.

ENROLL A DEVICE CERTIFICATE

On creating a new device certificate the Device enroll API is called. This triggers the following actions:

- The Device enroll API is used by a device to get a fresh new certificate.
- If no CA is available an error occurred with message `Tenant CA certificate is either missing, expired, or has a validity of less than one year` .
- If tenant's keypair is not found then an error occurred with message `Failed to retrieve tenant keypair` .
- The Common Name (CN) field in the Certificate Signing Request (CSR) subject must exactly match the device ID supplied during the device registration process. If the request does not contain a valid Certificate Signing Request an error will be returned.

> **INFO**
>
> As per EST standards, the certificate in response is in `PKCS7` format by default. Clients can optionally request for a `PKCS10` format by sending `Accept: application/pkcs10` in the request header.

This is an example of a REST request without `Accept: application/pkcs10` (default):

```
POST /.well-known/est/simpleenroll
Authorization: Basic <<Base64 encoded bootstrap credentials>>
Content-Type: application/pkcs10
Content-Transfer-Encoding: base64
...
{
  MIICVjCCAT4CAQAwETEPMA0GA1UEAwwGSWRfMTA0MIIBIjANBgkqhkiG9w0BAQEF
  AAOCAQ8AMIIBCgKCAQEAtz41ucrtb28Hk4FDkGK19DRHD6Z3oyZ6CtS1xycWkf9h
  za+6DI6ugBLRsxgRQi+VcBSQ8BRgPQTj3eEKi6s+7ySLpzuA4HlihPWo7dWhioXM
  3lZ4jiyPlCAZALYHNWbgk6Pehk6eNKMz8998Kq2GnzUQ65grAjjr4Q+HQJGOUYx/
  gsEQoCdhKRjG0xcI75OE+CW9Wg/VU4RMboMke5LUNHwnHIDy8Ie2C3VozVS6mRgE
  1VIBXWDm9fjaGEzYf25yWk0fzo/d/osLxReJgBq3FOz1TM4m9c3CtJUiedVXmeuf
  OPwNlTjGDOWgQfeMmXaPC0jHp0YNfV+txdWwEPev6QIDAQABoAAwDQYJKoZIhvcN
  AQELBQADggEBAHIZu0WR6hfiIsPxhBrdo5w0bt/2X6gy+tHdTI017rf1FUww6OLs
  wKtpPC0km4BS20hxbD9NLb3FNQEUCf80YQFSGbi0ziY0okVN7gaes6XiNofZbYx9
  TQF0oo+QWTa+otjoXpw8lLY9Ak6T9MppYh2GlRIiio2VzFu4Vg+FEoyNw9jvQwLj
  LP0eYTIQ/2SX1DnxMBzCm4MzieXJ7DJPHAdqADUfKbFNuaVjdxG9uRZdP3LRL90g
  6YxfFb8c+RcOL/lAKdSP5/rIUI05z0agzGMajsEnqxRXSk+CwlZo2D02++STStY9
  pELozQsItNjEVrfWta6353kOguYYqjB1rNY=
}
```

The following response is returned:

```
HTTP/1.1 201 OK
Content-Type: application/pkcs7-mime;smime-type=certs-only
Content-Length: ...
{
  MIIBpQYJKoZIhvcNAQcCoIIBljCCAZICAQExADALBgkqhkiG9w0BBwGgggF6MIIB
  djCCARygAwIBAgIDCnqWMAoGCCqGSM49BAMCMBcxFTATBgNVBAMTDGVzdEV4YW1w
  bGVDQTAeFw0yNTA1MjIwNjUwMzJaFw0yNjA1MjIwNjUwMzJaMBQxEjAQBgNVBAMM
  CWV4YW1wbGUwMTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABI882KOUAoc5DCKF
  u0zansWbtL84zKJa7SnconIF3AYjvHl3zBUnkv+/7+tSA13mBbYdq4oGh6Yg83zj
  zhEJ0eujWjBYMAkGA1UdEwQCMAAwCwYDVR0PBAQDAgeAMB0GA1UdDgQWBBTsMXNm
  ZMWaR8B74Imk6XckmUvOXTAfBgNVHSMEGDAWgBQa3zmEwlbmbM8qtCal/QzSQ/U9
  PjAKBggqhkjOPQQDAgNIADBFAiEAhrhYYDGcvsFkJpg9U4fr8d7LuqearEnMn+TF
  RWWgCmICIGPUvHHKzSc8GbgwuKddhe0Ef+HTamyEmqQBX3z/iE3GMQA=
}
```

And an example of a REST request with `Accept: application/pkcs10` (optional):

```
POST /.well-known/est/simpleenroll
Authorization: Basic <<Base64 encoded bootstrap credentials>>
Content-Type: application/pkcs10
Content-Transfer-Encoding: base64
Accept: application/pkcs10
...
{
  MIICVjCCAT4CAQAwETEPMA0GA1UEAwwGSWRfMTA0MIIBIjANBgkqhkiG9w0BAQEF
  AAOCAQ8AMIIBCgKCAQEAtz41ucrtb28Hk4FDkGK19DRHD6Z3oyZ6CtS1xycWkf9h
  za+6DI6ugBLRsxgRQi+VcBSQ8BRgPQTj3eEKi6s+7ySLpzuA4HlihPWo7dWhioXM
  3lZ4jiyPlCAZALYHNWbgk6Pehk6eNKMz8998Kq2GnzUQ65grAjjr4Q+HQJGOUYx/
  gsEQoCdhKRjG0xcI75OE+CW9Wg/VU4RMboMke5LUNHwnHIDy8Ie2C3VozVS6mRgE
  1VIBXWDm9fjaGEzYf25yWk0fzo/d/osLxReJgBq3FOz1TM4m9c3CtJUiedVXmeuf
  OPwNlTjGDOWgQfeMmXaPC0jHp0YNfV+txdWwEPev6QIDAQABoAAwDQYJKoZIhvcN
  AQELBQADggEBAHIZu0WR6hfiIsPxhBrdo5w0bt/2X6gy+tHdTI017rf1FUww6OLs
  wKtpPC0km4BS20hxbD9NLb3FNQEUCf80YQFSGbi0ziY0okVN7gaes6XiNofZbYx9
  TQF0oo+QWTa+otjoXpw8lLY9Ak6T9MppYh2GlRIiio2VzFu4Vg+FEoyNw9jvQwLj
  LP0eYTIQ/2SX1DnxMBzCm4MzieXJ7DJPHAdqADUfKbFNuaVjdxG9uRZdP3LRL90g
  6YxfFb8c+RcOL/lAKdSP5/rIUI05z0agzGMajsEnqxRXSk+CwlZo2D02++STStY9
  pELozQsItNjEVrfWta6353kOguYYqjB1rNY=
}
```

The following response is returned:

```
HTTP/1.1 201 OK
Content-Type: application/pkcs10
Content-Length: ...
{
   -----BEGIN CERTIFICATE-----
   MIIBdjCCARygAwIBAgIDCnqWMAoGCCqGSM49BAMCMBcxFTATBgNVBAMTDGVzdEV4
   YW1wbGVDQTAeFw0yNTA1MjIwNjUwMzJaFw0yNjA1MjIwNjUwMzJaMBQxEjAQBgNV
   BAMMCWV4YW1wbGUwMTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABI882KOUAoc5
   DCKFu0zansWbtL84zKJa7SnconIF3AYjvHl3zBUnkv+/7+tSA13mBbYdq4oGh6Yg
   83zjzhEJ0eujWjBYMAkGA1UdEwQCMAAwCwYDVR0PBAQDAgeAMB0GA1UdDgQWBBTs
   MXNmZMWaR8B74Imk6XckmUvOXTAfBgNVHSMEGDAWgBQa3zmEwlbmbM8qtCal/QzS
   Q/U9PjAKBggqhkjOPQQDAgNIADBFAiEAhrhYYDGcvsFkJpg9U4fr8d7LuqearEnM
   n+TFRWWgCmICIGPUvHHKzSc8GbgwuKddhe0Ef+HTamyEmqQBX3z/iE3G
   -----END CERTIFICATE-----
}
```

RE-ENROLL A DEVICE CERTIFICATE

The Device re-enroll API is used by a device to renew its certificate or to substitute the current credentials for a certificate (a password or a certificate generated by another CA). The device has to authenticate itself using its password or a JWT token.

```
POST /.well-known/est/simplereenroll
Authorization: Basic <<Base64 encoded bootstrap credentials>>
Content-Type: application/pkcs10
Content-Transfer-Encoding: base64
...
{
   MIICVjCCAT4CAQAwETEPMA0GA1UEAwwGSWRfMTA0MIIBIjANBgkqhkiG9w0BAQEF
   AAOCAQ8AMIIBCgKCAQEAtz41ucrtb28Hk4FDkGK19DRHD6Z3oyZ6CtS1xycWkf9h
   za+6DI6ugBLRsxgRQi+VcBSQ8BRgPQTj3eEKi6s+7ySLpzuA4HlihPWo7dWhioXM
   3lZ4jiyPlCAZALYHNWbgk6Pehk6eNKMz8998Kq2GnzUQ65grAjjr4Q+HQJGOUYx/
   gsEQoCdhKRjG0xcl75OE+CW9Wg/VU4RMboMke5LUNHwnHIDy8Ie2C3VozVS6mRgE
   1VIBXWDm9fjaGEzYf25yWk0fzo/d/osLxReJgBq3FOz1TM4m9c3CtJUiedVXmeuf
   OPwNlTjGDOWgQfeMmXaPC0jHp0YNfV+txdWwEPev6QIDAQABoAAwDQYJKoZIhvcN
   AQELBQADggEBAHIZu0WR6hfiIsPxhBrdo5w0bt/2X6gy+tHdTI017rf1FUww6OLs
   wKtpPC0km4BS20hxbD9NLb3FNQEUCf80YQFSGbi0ziY0okVN7gaes6XiNofZbYx9
   TQF0oo+QWTa+otjoXpw8lLY9Ak6T9MppYh2GlRIiio2VzFu4Vg+FEoyNw9jvQwLj
   LP0eYTIQ/2SX1DnxMBzCm4MzieXJ7DJPHAdqADUfKbFNuaVjdxG9uRZdP3LRL90g
   6YxfFb8c+RcOL/lAKdSP5/rIUI05z0agzGMajsEnqxRXSk+CwlZo2D02++STStY9
   pELozQsItNjEVrfWta6353kOguYYqjB1rNY=
}
```

The following response is returned:

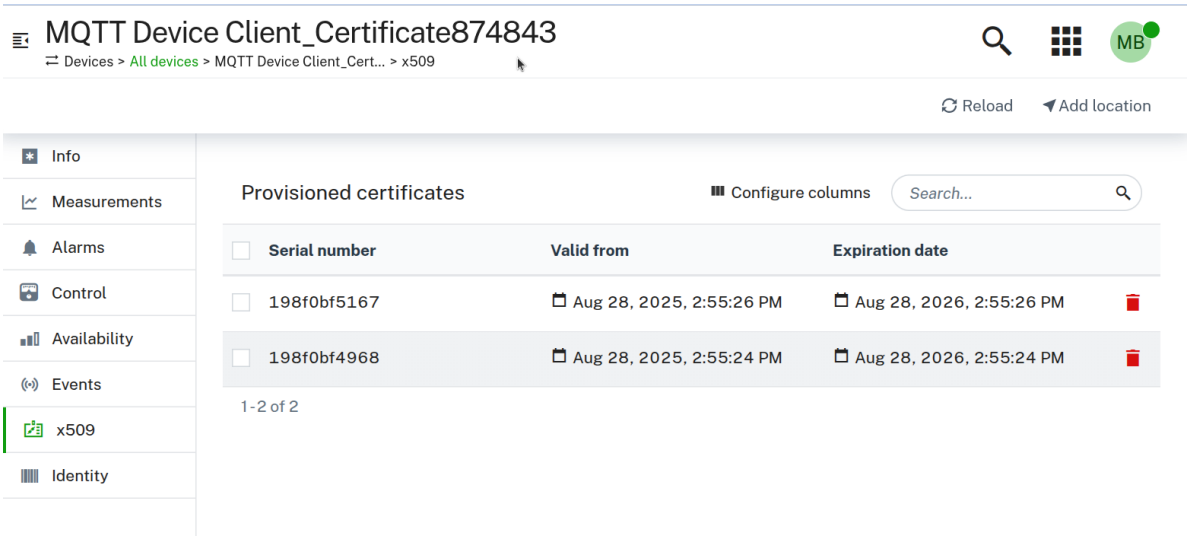
```
HTTP/1.1 201 OK
Content-Type: application/pkcs7-mime;smime-type=certs-only
Content-Length: ...
{
   -----BEGIN CERTIFICATE-----
   MIIEGjCCAwKgAwIBAgIEEgEgJDANBgkqhkiG9w0BAQsFADBSMQswCQYDVQQGEwJF
   VTELMAkGA1UECAwCUEwxGzAZBgNVBAoMEklvdCBEZXZpY2UgRmFjdG9yeTEZMBcG
   A1UEAwwQSW90RGV2RmFjdG9yeUludDAeFw0yNDA5MDIxMjA1MDlaFw0yNTA5MTIx
   MjA1MDlaME8xCzAJBgNVBAYTAkVVMQswCQYDVQQIDAJQTDEbMBkGA1UECgwSSW90
   IERldmljZSBGYWN0b3J5MRYwFAYDVQQDDA1Jb3REZXZpY2UyMDAwMIIBIjANBgkq
   hkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAumzj99zMfC9wyTHun3dEIsL/wk3VyWTw
   dtgdFx/lt5uPfvTAJ62GLWzYsCe2Q54G13IG/S1SJTfsNURlvUrPvXed97/yhe/o
   g0ljLz5VGNfsNUw+51KI7Tcc1SunHvUKD7TtgZ4rVga5q1DFxmzZ/dFXJpG5VAgu
   pwojACMv+T6qKGlUsF5F/1coOVrUo26aby4mg7C6ZsbwzXj2PbIvNXHCcrBrUU7S
   L2EI89RTgFZCVnZtW1SozDBn8S+WafKFBSgD0GwNtAkkffNWji1fW645gDc80u7w
   bQuA/xtPXK6+giGB8jN3daVjp6pzbzrYzDTYoC2vyL2A4F4zUedBhwIDAQABo4H6
   MIH3MAkGA1UdEwQCMAAwEQYJYIZIAYb4QgEBBAQDAgWgMDMGCWCGSAGG+EIBDQQm
   FiRPcGVuU1NMIEdlbmVyYXRlZCBDbGllbnQgQ2VydGlmaWNhdGUwHQYDVR0OBBYE
   FONhE6IdItEwn95lmf7uKjVLYpJ1MB8GA1UdIwQYMBaAFFSopCQWaJLzT3DYQi8n
   +nA1S86RMA4GA1UdDwEB/wQEAwIF4DAdBgNVHSUEFjAUBggrBgEFBQcDAgYIKwYB
   BQUHAwEwMwYDVR0fBCwwKjAooCagJIYiaHR0cDovL2xvY2FsaG9zdDoxNTAwMC9v
   bmxpbmUtY3JsczANBgkqhkiG9w0BAQsFAAOCAQEAJmPPKVSR4nAf8TZG3dPeGRX4
   V6a8mT52HKQYTilicq0cwec1zSyjayX6lsvqbENkA/L8fN64fpOSmb/+HUbSWP7Y
   F3ZYKalZVS9XlDUz6TWSA1LiWiDx1E28W2ubCOzUBRg7ux+59hoSGldtGrpxZ1Ox
   +H9kemart76xC+l85EYys7YARL5vk5Jwyr/f1/FpXasaGBFbC4aJ+2fNJkn1LJal
   II8Fl3GElheLpYM20VNw2J0PoD8I17htLfT+j7IwPyJ+uZcNxl60GiqDoBafRWuq
   oTL1SYqewv/dTU98aZUXG9yFLPgldQ2YfMli6vOC2gcjW2vun+IP7T/5ZU/xtA==
   -----END CERTIFICATE-----
}
```

Device certificate authentication › Simple device enrollment

PROVISIONED CERTIFICATES

This section describes how to view and revoke provisioned certificates.

Open the **x509** tab in the details of a specific device to view currently issued certificates.



Field	Description	Example
Serial number	Serial number of the certificate in hexadecimal format.	1989853f938
Not after	The date and time after which the certificate is no longer valid. This is effectively the certificate's **expiry date**, and any connection attempt using it after this time will fail the TLS/SSL handshake.	Aug 11, 2027, 2:21:36 PM
Not before	The date and time before which the certificate is not yet valid. This is typically the **issuance date**, and a certificate cannot be used before this point in time.	Aug 11, 2026, 2:21:36 PM

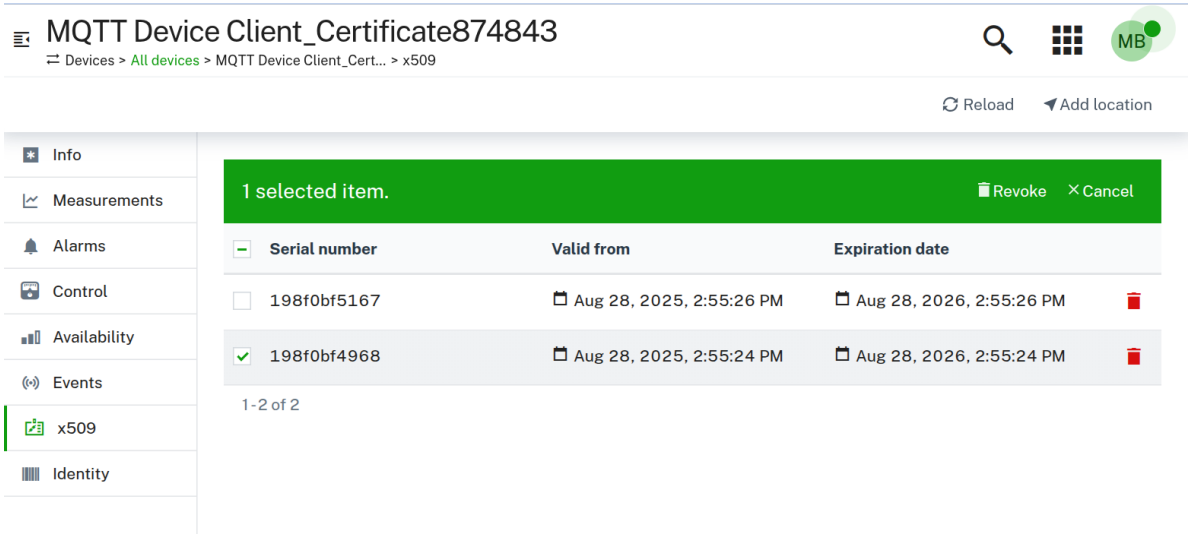
> **INFO**
>
> Issued certificates have a validity of 1 year, which is not configurable.

Only certificates that are used during a successful handshake with the platform are recorded in the provisioned certificates list.

Although only the last 5 entries are stored (based on the issue date), there is no restriction on the number of certificates that can be issued for a given device.

TO REVOKE A PROVISIONED CERTIFICATE

Provisioned certificates can be revoked by clicking the adjacent delete icon 🗑 for each entry, or by using multi-select.



Alternatively, revocation can also be performed from the existing CRL settings.