part of eex group

epexspot

ETS API Certificates

16.05.2022
Paris

Rel. V5.53

Table of Contents

1. Introduction

1.1 Audience

This document is intended for customers who will use the ETS API, EPEX SPOT Market Operators and EPEX SPOT IT Team.

1.2 Purpose

This documentation provides information about certificates needed to connect to ETS API Server.
It provides the technical information about certificates, the certificate management process and the process to obtain a certificate.

1.3 Changes History Table

Date	Version	Change description
03.02.2021	V5.5	TLS and cipher suites changes postponed from ETS API 3.5 to 3.6
10.06.2021	V5.51	ETS API 3.5 postponed to Q4 2021, ETS API 3.6 postponed to Q2 2022
17.02.2022	V5.52	Section §2.3 Technical Info update with last TLS and cipher suites announcements
18.05.2022	5.53	Security updates: TLS and cipher suites decommission TLS v1.0 and 1.1 and “old” cipher suites will be decommissioned as of ETS 3.5.5

2. Certification rules and process

2.1 General information

The certificate needed for the ETS API is a signed public key:

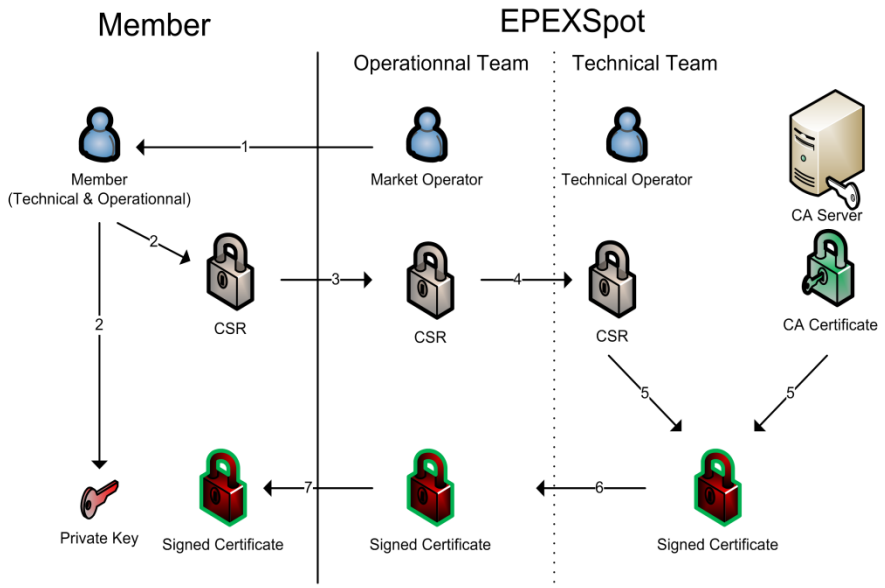
- generated by a trusted Certificate Authority (CA),
- based on a certificate signing request customers send to EPEX,
- which is created by the customer using the private key

API Customers have to generate the private key. Once the private key has been generated it can be used to generate the “Certificate Signing Request” file (CSR), as explained below.
Your private key should never be provided to anyone. Should for any reason the private part of the certificate be shared outside of the member, EPEX SPOT will not be able to guarantee the member identity.

The below sections will guide you through the required steps to obtain a signed certificate from EPEX and generate the mandatory technical file (keystore) each of your API application needs to establish a secure connection with an ETS API server (section 3).

2.2 Certification process

The following schema describes the certification process for customers (Members and Market Data Customers):



Steps:

1. Market Operators **validate the Member/Market Data customer identity** (referred below as the “customer”).
2. **The customer first generates the private key,** and once the key has been generated **the CSR can be generated using** that private key.
3. **The customer sends by email the CSR file** to Market Operators **but does NOT share the private key.**
4. Market Operators transfer the CSR file to a Technical Operator in order to get the certificate signed.

5. The Technical Operator uses the CSR file and the Certificate Authority (CA) certificate to **generate the Signed certificate** (.pem file).
6. The Technical Operator transfers the signed certificate (.pem file) to Market Operators.
7. **Market Operators send by email the signed certificate (.pem file) to the customer.**
8. **What to do with the signed certificate (.pem file)? :** please refer to section 3 explaining how to build the Key Store your API application needs to be able to connect to the ETS API server, using both the private key generated at step #2 and the signed certificate (.pem file).

Notes:

- The exchanged files (CSR File and Signed Certificate file) are public and can be exchanged by email.
- You can double check your CSR content by using a CSR decoder such as https://certlogik.com/decoder/
- The only valid certificates that can be used with the ETS API will be signed by the EPEX API Certificate Authority.

2.3 Technical information

In order to ensure a secure communication with ETS API, the following solutions have been implemented:

- All communications between ETS API Clients (member application) and ETS API Server are encrypted, using HTTPS
- Bi-lateral authentication system which requires a Client certificate to connect

The following solutions are supported by ETS API:

- Protocol TLS 1.2
  - TLS v1.0 and 1.1 will be decommissioned as of ETS 3.5.5,
  - TLS v1.3 will NOT be introduced as previously announced,
  - We will stick to TLS v1.2 until further notice.
- SHA2 cryptographic hash function with RSA encryption for public key (sha256WithRSAEncryption)
  - Note: other signing algorithms are not supported in this version.

List of supported cipher suites: (*): decommissioned as of ETS 3.5.5, first in SIMU2, then PROD and SIMU1.

Cipher suites	SIMU2 status (*)	SIMU1 and PROD status (*)
• TLS_RSA_WITH_AES_128_CBC_SHA256	Decommissioned	Decommissioned
• TLS_RSA_WITH_AES_256_CBC_SHA256	Decommissioned	Decommissioned
• TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	Supported	Supported
• TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	Supported	Supported
• TLS_RSA_WITH_AES_256_GCM_SHA384	Decommissioned	Decommissioned
• TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	Supported	Supported
• TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	Supported	Supported
• TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256	Decommissioned	Decommissioned
• TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	Supported	Supported
• TLS_RSA_WITH_AES_128_GCM_SHA256	Decommissioned	Decommissioned
• TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	Supported	Supported

2.4 CSR Creation Rules

Each certificate is unique and is identified by a combination of Country Name / Organization Name / Common Name.
This information is used by EPEX SPOT IT for the validation of the CSR and the generation of the signed certificate.
The following conditions must be met for the creation of the CSR:

- **Only ASCII characters are accepted**
- **Country Name** (2 letter code) : Must respect the country of the company (Validated by Operators),
- **Organization Name** (eg, company): Must meet the company short name. The company short name is the one set by EPEX SPOT at member registration; please contact Market Operation if you do not remember your company short name,
- **Common Name** (e.g. server FQDN or YOUR name): Client name must start with OrganizationName _ (OrganizationName is the same as previous field). If you have several certificates, this name must be unique.
    - Example:
        - For test environments SIMU1 and SIMU2: *[your company Short Name]ApiClientSimu*
        - For Production: *[your company Short Name]ApiClientProd*

2.5 Revocation of certificate

In case the member would like to revoke a certificate (e.g. if the private key got exposed), the process is as follows:

- The member contacts Market Operators, who validate the member identity
- The member provides the **Country Name / Organization Name / Common Name** combination which identifies the certificate to be revoked
- The member is contacted (by email) by Market Operators to confirm the revocation of his certificate

Once revoked, a certificate cannot be used anymore.

2.6 Expiration and Renewal of a certificate

The certificates are valid for 2 years in PRODUCTION and 5 years in SIMULATION.
EPEX Market operators monitor PRODUCTION certificates expiry dates and informs customers one month before the expiry date.

The validity period of a certificate cannot be extended and a new CSR should be provided to EPEXSPOT to generate a new signed certificate.
Note: The same **"Country Name / Organization Name / Common Name"** combination can be used for the new CSR.

3. From getting your signed certificate (.pem file) to connecting your API app

Once you received your signed certificate from EPEX (.pem file, signed by GlobalSign RSA OV SSL CA 2018) there are 2 steps you need to follow to get ready:

1. Install the root certificate
2. Build a Key store

These concepts and related tools are described the below sections.

3.1 How to install your certificate so you can start your implementation

- EPEX signed the certificate and send you a .pem file.
- Please generate a .cer file out of the signed certificate (.pem file) signed sent by EPEX
- double click on the “EPEX_customer.cer” file to install the CA root certificate:

- once the certificate is installed and added to the Trusted Root you should be able to retrieve WSDL/XSD files via the Browser itself (favored browser : please use Internet Explorer if you experience difficulties with Chrome),
- this is a pre-requisite before being able to import the WSDL in your development application (for this you might need to generate a .jks certificate for Java or a .pkcs12 certificate, like explained below.

The second step is the key store as described below.

3.2 Why do you need a Key Store?

Your API application cannot directly use the signed certificate sent by EPEX. It needs in addition your certificate private key (generated at the same time as your CSR), combined in an appropriate container called a key store.

Key Store = your signed certificate + its private key

This KeyStore file (with any file extension) is required when the application wants to communicate over TLS through a secure channel.
The most popular keyStore files are:

- JKS (*Java KeyStore*), a Java proprietary format;
- PKCS12, one of the *Public-Key Cryptography Standards*, not Java specific

3.3 JKS AND PKCS12 (PFX) Key Store File formats

The biggest difference between **JKS** and **PKCS12** is that:

- **JKS** is a **format specific to Java** which stores private keys and certificates.
- **PKCS12** is a standardized and language-neutral way of storing encrypted private keys and certificates.

JKS Key store file format
The default and most widely used format for these files is JKS (Java Keystore) for a Java based application until Java 8. With Java 9, the default Keystore format changed from JKS to PKCS12.
That is until Java 8 your keystore format will be JKS if you don't specify the -storetype while creating your keystore with the *keytool* command. However, the default keystore type will be changed to PKCS12 in Java 9 because of its enhanced compatibility compared to JKS.

PKCS12 Key store file format
PKCS#12 is a file format (often called .p12 or .pfx) where you can store a private key and certificates. It's used for converting/transporting keys and certificates.

PKCS12, is a standard keystore type is not Java specific. It is portable and can be operated with libraries written in languages such as Java, C, C++ or C#.

3.4 Using JKS or PKCS12 (PFX) files

Java applications typically expect to get the keys they need from JKS, and **it's easy to access from your own Java apps**. JKS is not accessible from outside Java though.

PKCS12 (aka PFX) files, on the other hand, are more secure language-neutral files that have been around long enough to be supported just about everywhere.

At the end of the day, the decision on what Keystore type to use should be based on how you plan to use the private key-that is: what applications will need to use the private key and what format(s) of key store do they already handle.

3.5 Using JKS or PKCS12 with SoapUI

With the Soap UI application, one can use any key store file format (.p12, .pfx, .jks) for a secure communication channel over TLS. Soap UI supports all these file formats and works in the same way with all of them.

4. Examples of CSR and KeyStore files generation

4.1 Example of CSR generation using OpenSSL

Pre-requisite: OpenSSL must be installed to be able to generate a CSR.

- Command to generate a CSR and Private Key associated **with password protection for private key**:
  openssl req -new -keyout [PrivateKeyPath] -out [CSRPath]
- Command to generate a CSR and Private Key associated **without** password protection for private key:
  openssl req -new -nodes -keyout [PrivateKeyPath] -out [CSRPath]

Generation example on a Linux server (same command line when OpenSSL is installed on Windows):

a) With Password protection for private key

```
root@pluton:~# openssl req -new -keyout /tmp/MyPrivateKey.key -out /tmp/MyCsr.key
Generating a 2048 bit RSA private key
.........................+++
.....................................................................................+++
writing new private key to '/tmp/MyPrivateKey.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [FR]:FR
State or Province Name (full name) [PARIS]:PARIS
Locality Name (eg, city) [PARIS]:PARIS
Organization Name (eg, company) [POWERNEXTSA]:POWERNEXTSA
Organizational Unit Name (eg, section) [DSI]:DSI
Common Name (e.g. server FQDN or YOUR name) []:POWERNEXTSA_Client001
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

b) Without Password:

```
root@pluton:~# openssl req -new -nodes -keyout /tmp/MyPrivateKey.key -out /tmp/MyCsr.k
Generating a 2048 bit RSA private key
.......+++
..............................+++
writing new private key to '/tmp/MyPrivateKey.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [FR]:FR
State or Province Name (full name) [PARIS]:PARIS
Locality Name (eg, city) [PARIS]:PARIS
Organization Name (eg, company) [POWERNEXTSA]:POWERNEXTSA
Organizational Unit Name (eg, section) [DSI]:DSI
Common Name (e.g. server FQDN or YOUR name) []:POWERNEXTSA_Client002
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

4.2 Example of PKCS12 keystore generation using OpenSSL

Pre-Requisites:

4.3 OpenSSL must be installed

- you need to have the private Key file and the signed certificate (.pem file)

Command to generate a PKCS12 with private key and certificate:

openssl pkcs12 -in [CertitifactePath] -inkey [PrivateKeyPath] -export -out [Pkcs12Path]

Generation example on a Linux server (same command line when OpenSSL is installed on Windows):

```
root@scolca001:~# openssl pkcs12 -in /tmp/ApiExemple.pem -inkey /tmp/ApiExe
mple.key -export -out /tmp/ApiExemple.p12
Enter pass phrase for /tmp/ApiExemple.key:
Enter Export Password:
Verifying - Enter Export Password:
root@scolca001:~# ls /tmp | grep p12
ApiExemple.p12
root@scolca001:~#
```

4.4 Example of JKS keystore generation using Java keytool.exe

Pre-Requisites:

- Java must be installed
- you need to have a PKCS12 file.

Command to generate a JKS from PKCS12:

keytool.exe -importkeystore -deststorepass [WantedKeystorePassword] -destkeystore [DestinationKeystoreName.jks] -srckeystore [SourceKeyStore -srcstoretype PKCS12

Example:

5. Troubleshooting

5.1 How to debug a handshake issue

You have your signed certificate (.pem file) and private key, but you do not success in connecting to our server, because of an handshake issue you can use one of the two below alternative approaches

- Option 1: Command to check your TLS connection to our server. Simulation example:
  - *openssl s_client -showcerts -connect ets-simu2.api.epexspot.com:4444 -tls1_2 -state -key <privateKey> -cert <PEMFile>*
  - In case of a successful connection you shoud get the following message: TO BE COMPLETED
  - In case of failure:

- Option 2: Command to download the WSDL file from our API server:
  - *curl --tlsv1.2 --tls-max 1.2 --key <privateKey> private.key --cert <PEMFile> -v* https://ets-simu2.api.epexspot.com:4444/OpenAccess/3.4?wsdl
  - In case of a success the whole WSDL file content should be displayed.

Note: In case you are using PYTHON and unable to establish the TLS connection:

1) Force the use of the TLS version 1.2.
2) Don't use a pkcs12 keystore –just pass the .pem file and your private key .key file as separate arguments TO WHICH COMMAND/WHERE?