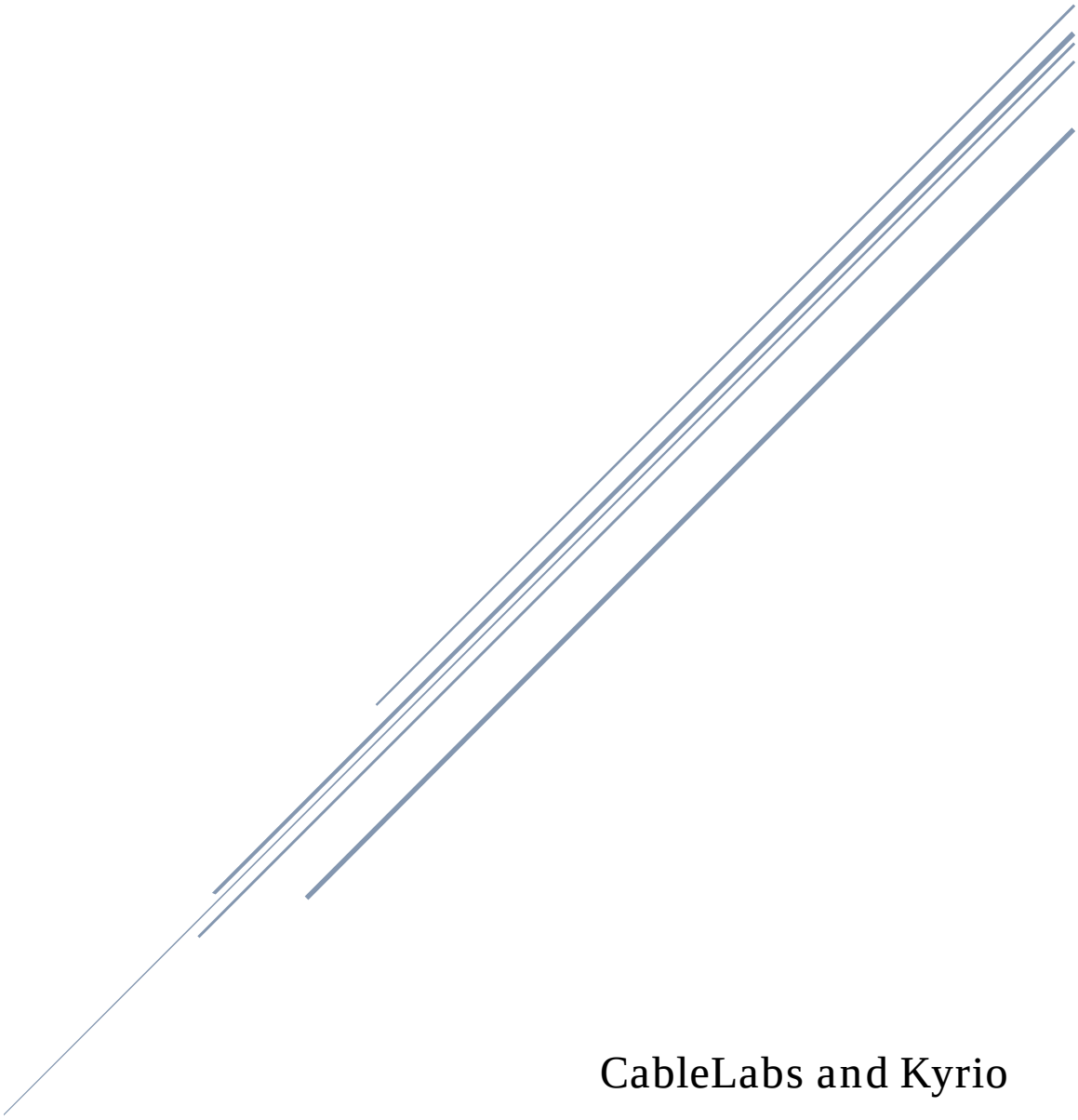
CERTIFICATE REQUESTING OPERATIONS MANUAL

Practical Information for PKI subscribers



CableLabs and Kyrio

REFERENCES

[1] WiFi Alliance – Certification WiFi Test Suite. Available at https://www.wi-fi.org/certification/wi-fi-test-suite.

[2] The Internet Engineering Task Force (IETF) – IETF RFC 2315. PKCS #7: Cryptographic Message Syntax Version 1.5, edited by B. Kaliski, March 1998. Also available at https://www.rfc-editor.org/rfc/rfc2315.txt.

[3] The Internet Engineering Task Force (IETF) – IETF RFC 2986. PKCS #10: Certification Request Syntax Specification Version 1.7, edited by M. Nystrom et al., November 2000. Also available at https://www.rfc-editor.org/rfc/rfc2986.txt.

[4] The Internet Engineering Task Force (IETF) – IETF RFC 5208. Public-Key Cryptography Standards (PKCS) #8: Private-Key Information Syntax Specification Version 1.2, edited by B. Kaliski, May 2008. Also available at https://www.rfc-editor.org/rfc/rfc5208.txt.

[5] The Internet Engineering Task Force (IETF) – IETF RFC 5652. Cryptographic Message Syntax (CMS), edited by R. Housley, September 2009. Also available at https://www.rfc-editor.org/rfc/rfc5652.txt.

[6] The Internet Engineering Task Force (IETF) – IETF RFC 7292. PKCS #12: Personal Information Exchange Syntax v1.1, edited by K. Moriarty et al., July 2014. Also available at https://www.rfc-editor.org/rfc/rfc7292.txt.

Certificate Requesting Operations User Manual

1 Kyrio and CableLabs

Kyrio and CableLabs partner in providing high-quality secure certification services for the Cable Industry and other Ecosystems that have become part of the family of offered products. Through Kyrio, you can access a variety of different ecosystems that cover many types of PKIs – from the DOCSIS® family of products to IoT related infrastructures, Kyrio and CableLabs provide a one-stop shop for all of your needs.

2 Public Key Infrastructures (PKIs)

Public Key Infrastructures or PKIs provide a technical solution for scaling trust. In fact, instead of having to enable trust in every single communication, devices can trust one or more Root Certification Authorities or Trust Anchors (TAs) and use cryptography and digital signatures to extend trust to all entities with a valid certificate from the same infrastructure.

Selecting your ecosystem is, therefore, the key to your success – the selection criteria can depend on several factors such as functional or protocol-driven requirements. Please contact us for questions related to specific needs and/or ecosystems' details.

2.1 Public Key Algorithms

There are many different algorithms and acronyms used in PKIs. In particular when it comes to public key algorithms, today, there are two main ones that are actively deployed: RSA and EC.

RSA is the oldest public-key algorithm and it has been commonly used for decades. This algorithm exploits the difficulties in providing efficient factorization of large numbers as the underlying mathematical property that guarantees the security of the system. Typical key sizes that are considered secure are in the range of 2048-4096 bits.

EC or Elliptic-Curves cryptography is relatively new (compared to RSA) and offers efficiency in terms of computation and crypto-messages size over RSA. However, differently from RSA, the key size depends on the selected curve and, therefore, to choose a different key-size a different curve must be picked (while in RSA no additional parameters are needed but the key size). EC keys are used with the ECDSA algorithms to produce signatures[1]. Typical key sizes that are considered secure are in the range of 128-521 bits.

[1] The algorithm does not directly support data encryption and requires the use of a key-exchange algorithm like Diffie-Hellman to support secure derivation of content-encryption keys.

2.2 Digest (or Hashing) Algorithms

Digest (or Hashing) Algorithms are used in PKI to create unique "summary" of data (or messages) – these can be used instead of the original message to, for example, calculate a signature. As long as the hashing algorithm provide the required properties for uniqueness and pre-calculation resistance, the digest (or hash) of a message can be used in its place. Typical digest algorithms are SHA-1, SHA-256, SHA-384, and SHA-512.

3 Introduction to PKIs, Data Types, and Encodings

Submitting Certificate Request involves several procedural steps that have been designed to guarantee the integrity of the ecosystems we serve. Please contact us at ***pkiops@kyrio.com*** for a detailed description of the registration and request submission procedures.

In this section we provide a brief description of the encoding and standards used when generating CSRs. In the next sections we will cover the basic requirements and procedures for generating Certificate Signing Requests (or CSRs) for the different ecosystems.

3.1 What is Certificate Signing Request (.csr)?

In public key infrastructure, a certificate signing request (CSR) is a piece of data that binds the information about a subscriber (e.g., the Organization Name or the MAC address of a device) to a Public Key. The standard data format used to carry this information is called PKCS#10.

3.2 Binary and Text Encoding (DER vs. PEM)

In PKIs, there are two main encodings for the different data structures used (e.g., CSRs or Certificates). Specifically, when a binary and more compact encoding is preferred, the ***Distinguished Encoded Rules (DER)*** is used. When, instead, a text version is more suitable (e.g., easier for copy and paste), the ***Privacy Enhanced Mail (PEM)*** encoding is used instead. It is possible to convert from the DER (binary format) to/from the PEM (text format) because the PEM

```
-----BEGIN CERTIFICATE-----
... DER encoded certificate in Base 64
```

Figure 1 - Example of PEM format (certificate)

format is obtained from the DER format by applying the Base64 encoding to it and then adding a "`-----BEGIN <TYPE>-----`" and a "`-----END <TYPE>-----`" lines to the beginning and end of the data[2]. The following is an example of a PEM formatted certificate:

[2] Example values for <TYPE> are "CERTIFICATE", "CERTIFICATE REQUEST", "PRIVATE KEY", or "X509 CRL"

All cryptographic library and tools are capable of handling both the binary (DER) and the text (PEM) encodings of PKI data types.

3.3 Common PKI Data Types

In this section we provide a quick summary of the different data types used when dealing with PKIs and generating new certificates requests. Use this section as a starting point for further research.

3.3.1 Private Keys (PKCS#8)

In order to be able to securely transport keys, the PCKS#8 data structure is defined in [4] and provides the required syntax for storing private key information for all algorithms. A previous version of private key storage format was defined in PKCS#1, however because it is algorithm-specific and can handle only RSA, the PKCS#8 is today used to provide secure transport for private keys of any algorithm (e.g., RSA, ECDSA, etc.)

3.3.2 Certificate Requests or CSRs (PKCS#10)

In PKIs there are many different protocols for submitting and renewing certificates. The most used format for submitting certificate requests is PKCS#10 which is defined in [3]. This standard specifies the syntax for binding the subscriber's data (e.g., the name of the organization or the mac address of a device) to a public key. When generating the request in PKCS#10, the private key of the subscriber is used to sign the certificate request itself as a "Proof of Possession" (or PoP) for the private key (required by our Certification Authorities). The PKCS#10 type is commonly referred to as CSR or Certificate Signing Request.

3.3.3 PKI Credentials (PKCS#12)

When credentials are generated by the provider (i.e., both the Key Pair and the Certificate), the PKCS#12 format is used to deliver the key pair and the issued certificate together. This format it is also used to export and import PKI credentials across applications and operating systems[3]. The PKCS#12 is defined in [6] as a standard for archive format that is used for storing many cryptographic objects (private keys, certificates) as a single file.

3.3.4 Certificate Revocation Lists (CRLs)

Should a compromise occur for a specific key pair, the PKI administrators need to communicate to any third party trying to validate a certificate that that particular one is not to be trusted – even if it is not expired yet.

In order to provide all relying parties with a list of compromised PKI credentials, PKIs use the concept of Certificate Revocation Lists (CRLs). These lists are signed by the Certification Authority and contain all the serial numbers of certificates that, for one reason or another, are not to be considered trusted. CRLs can be provided in binary (DER) or text format (PEM).

---

[3] When exporting or importing a certificate and a key pair on Windows platform, the PKCS#12 format is used to protect the credentials – the filename extension used by Windows is ".pfx"

3.3.5 Trust Stores (PKCS#7 and CMS)

The PKCS#7 format is defined in [2] which described general syntax for cryptographic data such as digital signature and digital envelopes. Another common use of the PKCS#7 format is to distribute Trust Anchors.

Because the PKCS#7 format suffers from some limitations when it comes to the algorithms, a new version of the specification is available: The Cryptographic Message Syntax (CMS) [5]. This format supports all the new algorithm (algorithm agility) and can be used in a backward-compatible way when it comes to digital signatures.

4 Submitting Certificate Requests

4.1 Software Requirements

In order to be able to generate a Certificate Signing Request (or CSR), you will need to use a tool that is capable of Key Generation. OpenSSL is a popular opensource cryptographic library for TLS/SSL protocols that provides many Command Line Tools that can help in key and CSR generation.

With OpenSSL, one can generate certificates and key pairs or convert cryptographic objects in different formats. All examples in this user manual use OpenSSL to generate certificate signing request. It is strongly suggested that you install the latest stable OpenSSL version on your computer before proceeding.

Please find more information about how to install the OpenSSL version for your platform from the official OpenSSL website directly[4].

Conventions used throughout the document:

- The command-line prompt is represented by a dollar sign (“$”). On Windows platforms this is usually represented by a letter and other symbols (e.g., “C:\ >”)
- The backslash (“\”) symbol is used to break lines for better readability. When typing the command on your computer, please ignore the backslash symbols at the end of lines and continue typing as if the new-line is not there (i.e., assume the command is provided on a single line)

4.2 First Step: Generate a Keypair

The OpenSSL project provides several command-line tools that allow the user to generate keys, requests, and other data types. In order to generate a request, the user must first generate a keypair which contains the private key (your secret) and the public key that is to be included in the certificate request that is sent to Kyrio.

To generate a new keypair, the “openssl” command takes different arguments depending on the algorithm used for the key generation. Although the majority of the ecosystem handled at Kyrio use the RSA algorithm, this section provides the instructions for both RSA and ECC key generation.

[4] https://www.openssl.org

4.2.1 RSA Keypair Generation

In order to generate an RSA keypair, you need to know which size is needed. This information is usually provided as a number of bits. Typical values of RSA key sizes are 2048, 3072, or 4098 bits. To generate an RSA key, the following command can be used:

```
$ openssl genrsa -out <KeyFileName> <bit_number>
```

Where:

- `<bit_number>` is the number of bits for the generated key (e.g., 2048, 3072, 4096, etc.)
- `<KeyFileName>` is the name of the file where the Keypair is stored

For example, to generate a 2048 bit keypair, you would use the following command:

```
$ openssl genrsa -out <KeyFileName> 2048
```

The generated file is your “Private Key” or “Secret” file that should NOT be shared with anybody else – a compromise of the private key must be reported to Kyrio for proper revocation of the associated PKI credentials.

4.2.2 EC/ECDSA Keypair Generation

In order to generate an EC (or ECDSA) keypair, you need to know which curve you want to use. This information is usually provided as a name or an identifier. For example, commonly values for this field are the NIST curves that are indicated with “secp256r1”, “secp384r1”, or “secp521r1” curve names. To generate an EC key, the following command can be used:

```
$ openssl ecparams -noout -genkey -curve <CurveName> -out <KeyFileName>
```

Where:

- `<CurveName>` is the name of the Elliptic Curve that is used for key generation. The name of the curve determines the number of bits in the keypair. For example, when using the (popular) “secp256r1” curve, the number of bits for the key is 256 which is equivalent to an RSA 3072 bit key.
- `<KeyFileName>` is the name of the file where the Keypair is stored

For example, to generate a P-256 (or secp256r1) keypair, you would use the following command:

```
$ openssl ecparam -noout -genkey -curve “secp256r1” -out “ecc.pem”
```

The generated file is your “Private Key” or “Secret” file that should NOT be shared with anybody else – a compromise of the private key must be reported to Kyrio for proper revocation of the associated PKI credentials.

4.3 Second Step: Generate the Request

Now that you have the keypair (i.e., the private and public key), you need to generate the certificate request to tie your information to the public key. The request is then submitted to Kyrio for the certificate to be issued and their contents varies depending on the specific ecosystem you are submitting your request to.

To generate a PKCS#10 request (or CSR) from the private key you just generated, you can use the following command:

```
$ openssl req -new -sha256 -key <KeyFileName> \
              -subj “<Certificate_Subject>” -out <ReqFileName>
```

Where:

- <KeyFileName> is the name of the private key (the file that was generated in the First Step)
- <Certificate_Subject> is the Subject Information that will be set in the issued certificate. The <Certificate_Subject> data MUST match the data contained in the DCAA that should have already be sent to (and approved by) Kyrio. The format of this field is simple – the ‘/’ symbol is used to separate the fields of the Subject. The double quotes (“”) are used to make sure that if spaces are present, the whole value is used correctly.
- <ReqFileName> is the name of the file where the request (or CSR) is saved.

For example, to generate a request for a DOCSIS 3.1 device certificate, the following command can be used:

```
$ openssl req -new -sha256 -key myKeyFile.pem \
    -subj “/C=US/O=CableLabs/OU=DOCSIS/CN=00:00:00:00:00:00” -out “req.pem”
```

5 Ecosystems and Enabled Products

CableLabs and Kyrio provide certification and registration services for a multitude of ecosystems. Each of these ecosystems have different requirements when it comes to generating compliant requests. In this section we provide a practical guide to generate compliant requests for the specific ecosystem.

Wherever not specified differently, this document assumes that the information provided in the certificate request matches the information provided in the corresponding DCAA document – this requirement stem from the need to provide extra validation steps when approving certificate requests to ensure that all the data that is embedded in issued certificates are correct.

5.1 DOCSIS® 3.1 and Remote PHY

5.1.1 Introduction

The DOCSIS 3.1 ecosystem provides the trust infrastructure for the Cable industry. In particular, device certificates are used on Cable Modems, Remote PHY devices, Infrastructure Servers, etc. to provide verifiable credentials for network authentication.

5.1.2 Available Products

The DOCSIS ecosystem supports the following types of certificates that can be requested through Kyrio:

- **DOCSIS® 3.1 and Remote PHY Device Certificate** – This profile is used for providing device certificates. This profile is used for Remote PHY devices.

- **DOCSIS® 3.1 and Remote PHY CVC Certificate** – This profile is used for firmware signing (Code Verification Certificate). This profile is used for CVCs to sign firmware for Cable Modems and Remote PHYs devices.
- **DOCSIS® 3.1 Service Provider and Remote PHY Server Certificate** – This profile is used to provide certificates for Server-Side authentication. This profile is used for Operators' services and Remote PHY Server certificates.

There are different requirements to be able to request the different types of certificates – please contact your customer representative to make sure you qualify to request the type of certificates before submitting the requests.

5.1.3 Requesting a DOCSIS 3.1 or Remote PHY Device Certificate

Ecosystem Name(s):	**Supported Algorithms:**
DOCSIS 3.1 ("New PKI"), Remote PHY	RSA (2048 bit)

In order to generate the request for DOCSIS 3.1 devices, you need to have the exact MAC address of the device and the organization name that will be used in the certificate (must match the submitted DCAA). Supported Key Sizes can be found in the DOCSIS® 3.1 SEC (See Appendix III – DOCSIS 3.1 SEC document) and Remote PHY specifications when generating the Keypair for the request (See 4.2). To generate the request, you can use the following syntax:

```
$ openssl req -new -sha256 -key rsa.pem \
    -subj "/C=<CC>/O=<Company Name>/OU=<Facility>/CN=<MAC_Address>" \
    -out <ReqFileName>
```

Where:

- <CC> is the two-letter country code[5].
- <Company_Name> is the name of the Company or Organization as submitted in the Exhibit B of the Digital Certificate Authorization Agreement (DCAA) (max 64 chars)
- <Facility> is the identifier of the Manufacturing Facility (max 64 chars)
- <MAC_Address> is the MAC address of the device the request is for. Each octet of the MAC address should be provided in its Hexidecimal representation separated by the colon (":") symbol.
- <ReqFileName> is the name of the file where the request (or CSR) is saved.

An example for generating a valid request (assuming CableLabs as the device manufacturer):

```
$ openssl req -new -sha256 -key rsa.pem \
    -subj "/C=US/O=CableLabs/OU=Louisville, CO/CN=00:00:00:00:00:00" \
    -out "req.pem"
```

[5] "US" (without the quotes) is the country code for United States of America.

5.1.4 Requesting a DOCSIS 3.1 and Remote PHY CVC Certificate

Ecosystem Name(s):	**Supported Algorithms:**
DOCSIS 3.1 (“New PKI”), Remote PHY	RSA (2048 bit)

In order to generate the request for DOCSIS 3.1 and Remote PHY Code Verification Certificates, you just need the 2-letters country code and the organization name that will be used in the certificate (must match the submitted DCAA). Supported Key Sizes can be found in the DOCSIS® 3.1 SEC (See Appendix III – DOCSIS 3.1 SEC document) and Remote PHY specifications when generating the Keypair for the request (See 4.2). To generate the request, you can use the following syntax:

```
$ openssl req -new -sha256 -key rsa.pem \
    -subj “/C=<CC>/O=<Company Name>/CN=Code Verification Certificate” \
    -out <ReqFileName>
```

Where:

- <CC> is the two-letter country code[6].
- <Company_Name> is the name of the Company or Organization as submitted in the Exhibit E of the Digital Certificate Authorization Agreement (DCAA) (max 64 chars)
- <ReqFileName> is the name of the file where the request (or CSR) is saved.

For example, to generate a valid request (assuming CableLabs as the Manufacturer):

```
$ openssl req -new -sha256 -key “rsa.pem” \
    -subj “/C=US/O=CableLabs/CN=Code Verification Certificate” \
    -out “req.pem”
```

5.1.5 Requesting a DOCSIS 3.1 Service Provider and Remote PHY Server Certificate

Ecosystem Name(s):	**Supported Algorithms:**
DOCSIS 3.1 (“New PKI”), Remote PHY	RSA (2048 bit)

To submit a request for a server-side certificate for DOCSIS 3.1 or Remote PHY environments, you need the 2-letters country code, the organization name, and the Fully Qualified Domain Name (i.e., the server’s name) or FQDN that will be used in the certificate (must match the submitted DCAA). Supported Key Sizes can be found in the DOCSIS® 3.1 SEC (See Appendix III – DOCSIS 3.1 SEC document) and Remote PHY specifications when generating the Keypair for the request (See 4.2). To generate the request, you can use the following syntax:

[6] “US” (without the quotes) is the country code for United States of America.

```
$ openssl req -new -sha256 -key rsa.pem \
    -subj "/C=<CC>/O=<Company Name>/CN=<FQDN>" \
    -out <ReqFileName>
```

Where:

- <CC> is the two-letter country code[7].
- <Company_Name> is the name of the Company or Organization as submitted in the Exhibit B of the Digital Certificate Authorization Agreement (DCAA) (max 64 chars)
- <FQDN> is the fully qualified domain name (i.e., the server's full DNS name) as submitted in the Exhibit F of the Digital Certificate Authorization Agreement (DCAA) (max 64 chars)
- <ReqFileName> is the name of the file where the request (or CSR) is saved.

For example, to generate a valid request (assuming CableLabs as the Manufacturer):

```
$ openssl req -new -sha256 -key rsa.pem \
    -subj "/C=US/O=CableLabs/CN=Code Verification Certificate" \
    -out "req.pem"
```

5.2 WiFi Alliance Hotspot 2.0 (Online Sign-Up)

5.2.1 Introduction

Within the WiFi Alliance ecosystem, Online sign-up (OSU) is the process by which a mobile device registers with a service provider, enabling a user to select a plan with which to obtain network access, and is then provisioned with the credentials necessary to securely connect to an access network. The user's intent to connect to a selected SP is indicated by the user's selection of a Friendly Name and/or icon displayed on the mobile device's UI.

5.2.2 Available Products

The Hotspot 2.0 OSU ecosystem supports the following types of certificates that can be requested through Kyrio:

- **Passpoint 2.0 / Hotspot 2.0 OSU Server Certificate** – This profile is used for providing OSU production certificates.
- **Passpoint 2.0 / Hotspot 2.0 Test Bundle** – This profile is used for providing test bundles.

There are different requirements to be able to request the different types of certificates – please contact your customer representative to make sure you qualify to request the type of certificates before submitting the requests.

5.2.3 Requesting a Passpoint 2.0 / Hotspot 2.0 OSU Server Certificate

Ecosystem Name(s):	**Supported Algorithms:**
Passpoint 2.0 / Hotspot 2.0	RSA (2048 bits, 3072 bits, 4096 bits, +)

[7] "US" (without the quotes) is the country code for United States of America.

In order to generate a valid CSR for an OSU server, you need the 2-letters country code, the organization name, and the list of Fully Qualified Domain Names (FQDNs) that will be used in the certificate (must match the submitted DCAA). One of the FQDNs for the server should be used in the Common Name (CN) of the Subject of the Certificate. Supported Key Sizes can be found in Section 3.4 of the Hotspot 2.0 Online Sign-Up Certificate Policy Specification when generating the Keypair for the request (See 4.2). To generate the request, you can use the following syntax:

```
$ openssl req -new -sha256 -key rsa.pem \
    -subj "/C=<CC>/O=<Company Name>/OU=Hotspot 2.0 Online Sign Up Server/CN=<FQDN>" \
    -out <ReqFileName>
```

Where:

- <CC> is the two-letter country code[8].
- <Company_Name> is the name of the Company or Organization as submitted in the Exhibit B of the Digital Certificate Authorization Agreement (DCAA) (max 64 chars)
- <FQDN> is the fully qualified domain name (i.e., the server's full DNS name) as submitted in the Exhibit B of the Digital Certificate Authorization Agreement (DCAA) (max 64 chars)
- <ReqFileName> is the name of the file where the request (or CSR) is saved.

For example, to generate a valid request (assuming CableLabs as the Company Name):

```
$ openssl req -new -sha256 -key "rsa.pem" \
    -subj "/C=US/O=CableLabs/OU=Hotspot 2.0 Online Sign Up Server/CN=ap.cablelabs.com" \
    -out "req.pem"
```

It is important to notice that the information related to the Friendly Name and the Icon(s) are directly taken from the DCAA – any value that might be present as extensions in the CSR are ignored and replaced with the values from the submitted DCAA.

5.2.4 Requesting a Passpoint 2.0 / Hotspot 2.0 Test Bundle

Ecosystem Name(s):	**Supported Algorithms:**
Passpoint 2.0 / Hotspot 2.0	RSA (2048 bits, 3072 bits, 4096 bits, +)

The Test Bundle for Passpoint 2.0 provides the user with a set of different certificates (some of which are already revoked) to be used in TEST environments only.

The procedures for generating a valid set of CSRs for the Test Bundle is described in [1]. Please refer to the referenced documentation for the technical details. The submitted requests set should include the following files:

- IDQ-req-ARU.pem
- IDQ-req-RKS.pem

[8] "US" (without the quotes) is the country code for United States of America.

- `IDR2-req-ARU.pem`
- `IDR2-req-RKS.pem`

For more information about this procedure, please contact your customer representative with the specific questions you might want to get clarifications about.

5.3 OpenADR

5.3.1 Introduction

The OpenADR Alliance was created to standardize, automate, and simplify Demand Response (DR) and Distributed Energy Resources (DER). OpenADR offers RSA and ECC certificates for both Virtual Top Nodes (VTNs) and Virtual End Nodes (VENs).

5.3.2 Available Products

The OpenADR ecosystem supports the following types of certificates that can be requested through Kyrio:

- **OpenADR Server Certificate (Virtual Top Node or VTN)** – This profile is used for providing server-side authentication.
- **OpenADR Client Certificate (Virtual End Node or VEN)** – This profile is used for providing device or client certificates.

There are different requirements to be able to request the different types of certificates – please contact your customer representative to make sure you qualify to request the type of certificates before submitting the requests.

5.3.3 Requesting an OpenADR Server Certificate (VTN)

Ecosystem Name(s):	**Supported Algorithms:**
OpenADR (VTN or Server Certificates)	RSA (2048 bits) ECC (Secp256r1)

This profile is used in OpenADR to offer server-side authentication capabilities. In order to generate a valid CSR for Virtual Top Node (VTN), you need the 2-letters country code, the organization name, and the list of Fully Qualified Domain Names (FQDNs) that will be used in the certificate (must match the submitted DCAA). One of the FQDNs for the server should be used in the Common Name (CN) of the Subject of the Certificate. Supported Key Sizes and Algorithms are specified in the OpenADR CP – please verify with the CP the right values to use when generating the Keypair for the request (See 4.2). To generate the request, you can use the following syntax:

```
$ openssl req -new -sha256 -key MyPrivateKey.pem \
    -subj "/C=<CC>/O=<Company Name>/OU=<Additional Identifying Information>/CN=<FQDN>" \
    -out <ReqFileName>
```

Where:

- <CC> is the two-letter country code[9].
- <Company_Name> is the name of the Company or Organization as submitted in the Exhibit B of the Digital Certificate Authorization Agreement (DCAA) (max 64 chars)
- <Additional Identifying Information> contains additional data related to the organization (max 64 characters). This identifier does not have a specific format other than being a printable string.
- <FQDN> is the fully qualified domain name (i.e., the server's full DNS name) as submitted in the Digital Certificate Authorization Agreement (DCAA) (max 64 chars).
- <ReqFileName> is the name of the file where the request (or CSR) is saved.

For example, to generate a valid request (assuming CableLabs as the Company Name):

```
$ openssl req -new -sha256 -key "MyPrivateKey.pem" \
    -subj "/C=US/O=CableLabs/OU= OpenADR Alliance RSA VEN Certificate/CN=A33B1K01" \
    -out "req.pem"
```

It is important to notice that the OpenADR ecosystem supports both RSA and ECC algorithms through two separate Root CAs. When generating the request, please verify that the algorithm used to generate the corresponding private key is the one supported by the device the certificate will be installed on – i.e., RSA or ECC.

5.3.4 Requesting an OpenADR Device Certificate (VEN)

Ecosystem Name(s):	**Supported Algorithms:**
OpenADR (VEN or Device Certificates)	RSA (2048 bits) ECC (Secp256r1)

This profile is used in OpenADR to offer client-side authentication capabilities for devices. In order to generate a valid CSR for a Virtual End Node (VEN), you need the 2-letters country code, the organization name, and the device unique identifier (Unique Id) for the Common Name (e.g., a unique serial number, a MAC address, etc.). Supported Key Sizes can be found in Section 3.4 of the Hotspot 2.0 Online Sign-Up Certificate Policy Specification when generating the Keypair for the request (See 4.2). To generate the request, you can use the following syntax:

```
$ openssl req -new -sha256 -key rsa.pem \
   -subj "/C=<CC>/O=<Company Name>/OU=OpenADR Alliance RSA VEN Certificate/CN=<Unique_Id>" \
    -out <ReqFileName>
```

Where:

- <CC> is the two-letter country code[10].
- <Company_Name> is the name of the Company or Organization as submitted in the Exhibit B of the Digital Certificate Authorization Agreement (DCAA) (max 64 chars)

[9] "US" (without the quotes) is the country code for United States of America.

[10] "US" (without the quotes) is the country code for United States of America.

- <Unique_Id> is a unique identifier for the device. This identifier does not have a specific format (e.g., a MAC address, a DNS name, or a Serial Number are all valid examples) other than being a printable string.
- <ReqFileName> is the name of the file where the request (or CSR) is saved.

For example, to generate a valid request (assuming CableLabs as the Company Name):

```
$ openssl req -new -sha256 -key “ecc.pem” \
    -subj “/C=US/O=CableLabs/OU= OpenADR Alliance RSA VEN Certificate/CN=A33B1K01” \
    -out “req.pem”
```

The OpenADR ecosystem supports both RSA and ECC algorithms through two separate Root CAs. When generating the request, please verify that the algorithm used to generate the corresponding private key is the one supported by the device the certificate will be installed on – i.e., RSA or ECC.

DOCUMENT REVISIONS

Version	Author	Changes	Date
0.9	Yuan Tian	Initial Version of the Document	May 2020
1.0	Massimiliano Pala	Re-Structuring of the Document	June 2020