(/)

Creating a Self-Signed Certificate With OpenSSL

Last modified: October 10, 2022

Written by: baeldung (https://www.baeldung.com/author/baeldung)

Security (https://www.baeldung.com/category/security)

SSL (https://www.baeldung.com/tag/ssl)

1. Overview

OpenSSL is an open-source command-line tool that allows users to perform various SSL-related tasks.

In this tutorial, we'll learn **how to create a self-signed certificate with OpenSSL**.

Further reading:

Trusting a Self-Signed Certificate in OkHttp (/okhttp-self-signed-cert)

Learn how to configure an OkHttpClient to trust self-signed certificates

Read more (/okhttp-self-signed-cert) →

Converting a PEM File to Java KeyStore Format (/convert-pem-to-jks)

Learn how to convert certificates from a PEM (Privacy Enhanced Email) file to JKS (Java KeyStore) format using the openssl and keytool command-line utilities.

Read more (/convert-pem-to-jks) →

HTTPS using Self-Signed Certificate in Spring Boot (/spring-boot-https-self-signed-certificate)

Explore how to generate a self-signed certificate to enable HTTPS in a Spring Boot application.

Read more (/spring-boot-https-self-signed-certificate) →

2. Creating a Private Key

First, we'll create a private key. A private key helps to enable encryption, and is the most important component of our certificate.

Let's create a password-protected, 2048-bit RSA private key (*domain.key*) with the *openssl* command:

```
openssl genrsa -des3 -out domain.key 2048
```

We'll enter a password when prompted. The output will look like:

```
Generating RSA private key, 2048 bit long modulus (2 primes)
.....................+++++
.........+++++
e is 65537 (0x010001)
Enter pass phrase for domain.key:
Verifying - Enter pass phrase for domain.key:
```

If we want our private key unencrypted, we can simply remove the *-des3* option from the command.

3. Creating a Certificate Signing Request

If we want our certificate signed, we need a certificate signing request (CSR). The CSR includes the public key and some additional information (such as organization and country).

Let's create a CSR (*domain.csr*) from our existing private key:

```
openssl req -key domain.key -new -out domain.csr
```

We'll enter our private key password and some CSR information to complete the process. The output will look like:

```
Enter pass phrase for domain.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) [AU]:AU
State or Province Name (full name) [Some-State]:stateA
Locality Name (eg, city) []:cityA
Organization Name (eg, company) [Internet Widgits Pty Ltd]:companyA
Organizational Unit Name (eg, section) []:sectionA
Common Name (e.g. server FQDN or YOUR name) []:domain
Email Address []:email@email.com

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

An important field is "*Common Name,*" which should be the exact Fully Qualified Domain Name (FQDN) of our domain.

"*A challenge password*" and "*An optional company name*" can be left empty.

We can also create both the private key and CSR with a single command:

```
openssl req -newkey rsa:2048 -keyout domain.key -out domain.csr
```

If we want our private key unencrypted, we can add the *-nodes* option:

```
openssl req -newkey rsa:2048 -nodes -keyout domain.key -out
domain.csr
```

4. Creating a Self-Signed Certificate

A self-signed certificate is **a certificate that's signed with its own private key**. It can be used to encrypt data just as well as CA-signed certificates, but our users will be shown a warning that says the certificate isn't trusted.

Let's create a self-signed certificate (*domain.crt*) with our existing private key and CSR:

```
openssl x509 -signkey domain.key -in domain.csr -req -days 365 -out
domain.crt
```

The *-days* option specifies the number of days that the certificate will be valid.

We can create a self-signed certificate with just a private key:

```
openssl req -key domain.key -new -x509 -days 365 -out domain.crt
```

This command will create a temporary CSR. We still have the CSR information prompt, of course.

We can even create a private key and a self-signed certificate with just a single command:

```
openssl req -newkey rsa:2048 -keyout domain.key -x509 -days 365
-out domain.crt
```

5. Creating a CA-Signed Certificate With Our Own CA

We can be our own certificate authority (CA) by creating a self-signed root CA certificate, and then installing it as a trusted certificate in the local browser.

5.1. Create a Self-Signed Root CA

Let's create a private key (*rootCA.key*) and a self-signed root CA certificate (*rootCA.crt*) from the command line:

```
openssl req -x509 -sha256 -days 1825 -newkey rsa:2048 -keyout
rootCA.key -out rootCA.crt
```

5.2. Sign Our CSR With Root CA

First, we'll create a configuration text-file (*domain.ext*) with the following content:

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
subjectAltName = @alt_names
[alt_names]
DNS.1 = domain
```

The "*DNS.1*" field should be the domain of our website.

Then we can sign our CSR (*domain.csr*) with the root CA certificate and its private key:

```
openssl x509 -req -CA rootCA.crt -CAkey rootCA.key -in domain.csr
-out domain.crt -days 365 -CAcreateserial -extfile domain.ext
```

As a result, the CA-signed certificate will be in the *domain.crt* file.

6. View Certificates

We can use the *openssl* command to view the contents of our certificate in plain text:

```
openssl x509 -text -noout -in domain.crt
```

The output will look like:

```
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:

64:1a:ad:0f:83:0f:21:33:ff:ac:9e:e6:a5:ec:28:95:b6:e8:8a:f4
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = AU, ST = stateA, L = cityA, O = companyA, OU =
sectionA, CN = domain, emailAddress = email@email.com
        Validity
            Not Before: Jul 12 07:18:18 2021 GMT
            Not After : Jul 12 07:18:18 2022 GMT
        Subject: C = AU, ST = stateA, L = cityA, O = companyA, OU =
sectionA, CN = domain, emailAddress = email@email.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:a2:6a:2e:a2:17:68:bd:83:a1:17:87:d8:9c:56:
                    ab:ac:1f:1e:d3:32:b2:91:4d:8e:fe:4f:9c:bf:54:
                    aa:a2:02:8a:bc:14:7c:3d:02:15:a9:df:d5:1b:78:
                    17:ff:82:6b:af:f2:21:36:a5:ad:1b:6d:67:6a:16:
                    26:f2:a9:2f:a8:b0:9a:44:f9:72:de:7a:a0:0a:1f:
                    dc:67:b0:4d:a7:f4:ea:bd:0e:83:7e:d2:ea:15:21:
                    6d:8d:18:65:ed:f8:cc:6a:7f:83:98:e2:a4:f4:d6:
                    00:b6:ed:69:95:4e:0d:59:ee:e8:3f:e7:5a:63:24:
                    98:d1:4b:a5:c9:14:a5:7d:ef:06:78:2e:08:25:3c:
                    fd:05:0c:67:ce:70:5d:34:9b:c4:12:e6:e3:b1:04:
                    6a:db:db:e9:47:31:77:80:4f:09:5e:25:73:75:e4:
                    57:36:34:f8:c3:ed:a2:21:57:0e:e3:c1:5c:fc:d9:
                    f2:a3:b1:d9:d9:4f:e2:3e:ad:21:77:20:98:ed:15:
                    39:99:1b:7e:29:60:14:eb:76:8b:8b:72:16:b1:68:
                    5c:10:51:27:fa:41:49:c5:b7:c4:79:69:5e:28:a2:
                    c3:55:ac:e8:05:0f:4b:4a:bd:4b:2c:8b:7d:92:b0:
                    2d:b3:1a:de:9f:1a:5b:46:65:c6:33:b2:2e:7a:0c:
                    b0:2f
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha256WithRSAEncryption
         58:c0:cd:df:4f:c1:0b:5c:50:09:1b:a5:1f:6a:b9:9a:7d:07:
         51:ca:43:ec:ba:ab:67:69:c1:eb:cd:63:09:33:42:8f:16:fe:
         6f:05:ee:2c:61:15:80:85:0e:7a:e8:b2:62:ec:b7:15:10:3c:
         7d:fa:60:7f:ee:ee:f8:dc:70:6c:6d:b9:fe:ab:79:5d:1f:73:
         7a:6a:e1:1f:6e:c9:a0:ae:30:b2:a8:ee:c8:94:81:8e:9b:71:
         db:c7:8f:40:d6:2d:4d:f7:b4:d3:cf:32:04:e5:69:d7:31:9c:
         ea:a0:0a:56:79:fa:f9:a3:fe:c9:3e:ff:54:1c:ec:96:1c:88:
         e5:02:d3:d0:da:27:f6:8f:b4:97:09:10:33:32:87:a8:1f:08:
         dc:bc:4c:be:6b:cc:b9:0e:cf:18:12:55:17:44:47:2e:9c:99:
```

```
        99:3c:96:60:12:c6:fe:b0:ee:01:97:54:20:b0:13:51:4f:ee:
        1d:c0:3d:1a:30:aa:79:30:12:e2:4f:af:13:85:f8:c8:1e:f5:
        28:7c:55:66:66:10:f4:0a:69:c0:55:8a:9a:c7:eb:ec:15:f0:
        ef:bd:c1:d2:47:43:34:72:71:d2:c3:ff:f0:a3:c1:2c:63:56:
        f2:f5:cf:91:ec:a1:c0:1f:5d:af:c0:8e:7a:02:fe:08:ba:21:
        68:f2:dd:bd
```

7. Convert Certificate Formats

Our certificate (*domain.crt*) is **an X.509 certificate that's ASCII PEM-encoded**. We can use OpenSSL to convert it to other formats for multi-purpose use.

7.1. Convert PEM to DER

The DER format is usually used with Java. Let's convert our PEM-encoded certificate to a DER-encoded certificate:

```
openssl x509 -in domain.crt -outform der -out domain.der
```

7.2. Convert PEM to PKCS12

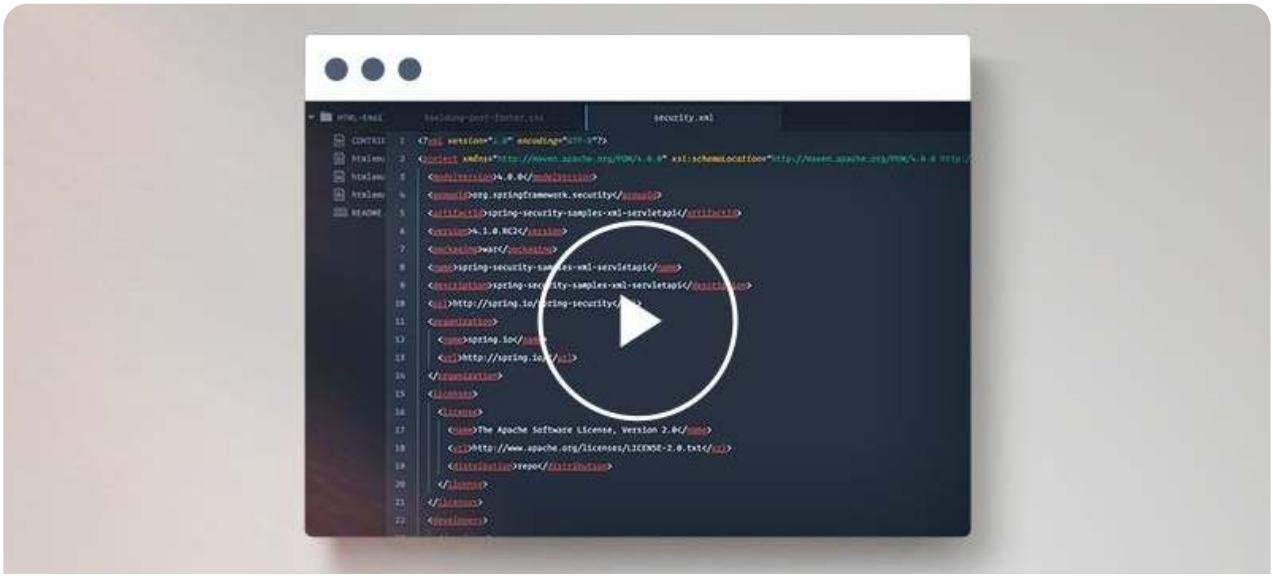
PKCS12 files, also known as PFX files, are usually used for importing and exporting certificate chains in Microsoft IIS.

We'll use the following command to take our private key and certificate, and then combine them into a PKCS12 file:

```
openssl pkcs12 -inkey domain.key -in domain.crt -export -out
domain.pfx
```

8. Conclusion

In this article, we learned how to **create a self-signed certificate with OpenSSL** from scratch, **view this certificate, and convert it to other formats**. We hope these things help with your work.



Comments are closed on this article!