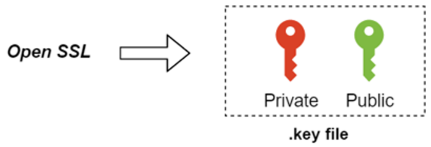
Self Signed Certificates: Create your own Certificate Authority (CA) for local HTTPS sites

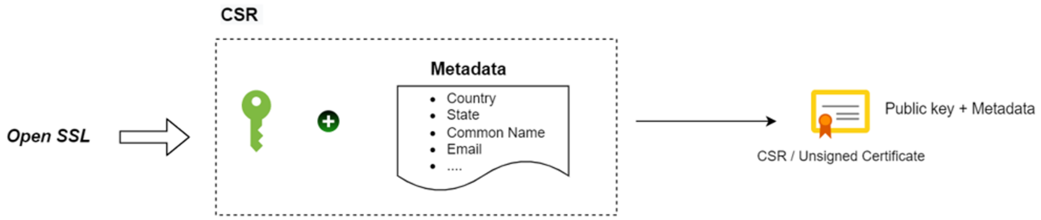
Understanding general concepts

How to create an SSL certificate

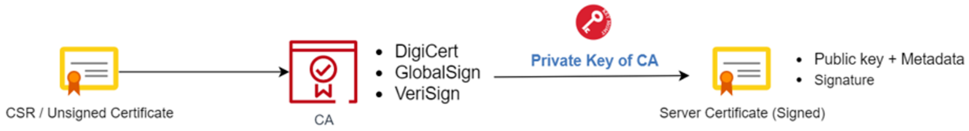
Step 1: Generate RSA key pair using OpenSSL



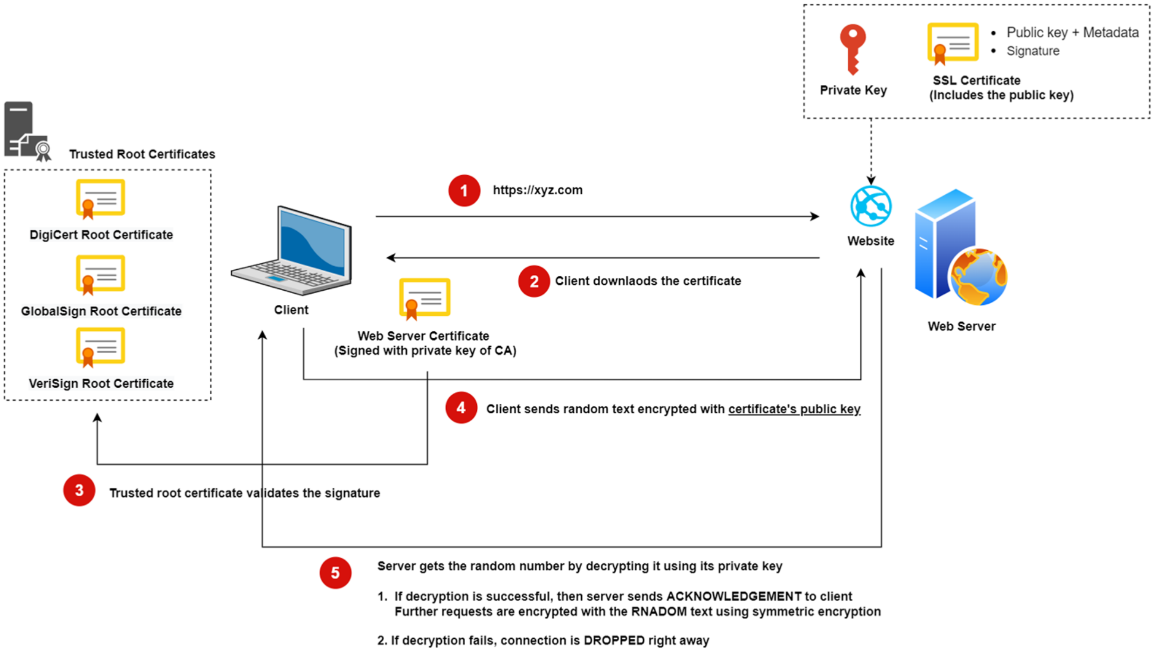
Step 2: Generate CSR (Certificate signing request) / Unsigned certificate



Step 3: Send CSR to CA for signing and get signed SSL certificate



How SSL works - Client/Server flow


Private Key
Public key + Metadata
Signature
SSL Certificate
(Includes the public key)
Trusted Root Certificates
DigiCert Root Certificate
GlobalSign Root Certificate
VeriSign Root Certificate
Client
Web Server Certificate
(Signed with private key of CA)
Website
Web Server

1 https://xyz.com

2 Client downlaods the certificate

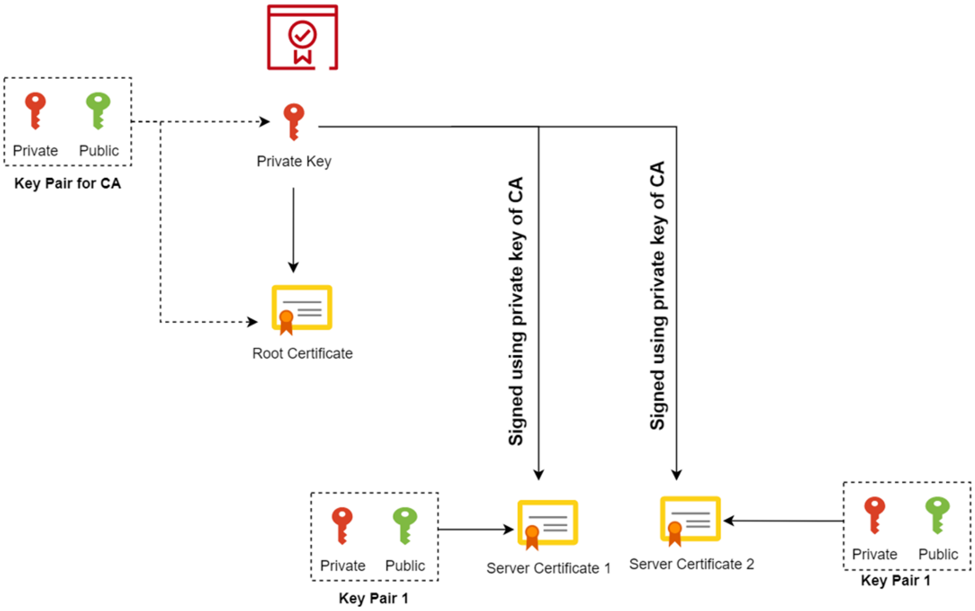
3 Trusted root certificate validates the signature

4 Client sends random text encrypted with certificate's public key

5 Server gets the random number by decrypting it using its private key

1. If decryption is successful, then server sends ACKNOWLEDGEMENT to client
   Further requests are encrypted with the RNADOM text using symmetric encryption
2. If decryption fails, connection is DROPPED right away

High level design



Pre-requisites

1. OpenSSL
    a. Refer installation guide
    b. Add `C:\Program Files\Git\usr\bin` in **Path** environment variable

Key Points

- A `.key` file contains both the **private and public** keys, but often referred as private key.
    - Use `openssl rsa -in private.key -pubout` to see hidden the public key.
    - Use `openssl rsa -in private.key -outform PEM -pubout -out public.pem` to export public key.
    - So, when you generate a `.key` file, you eventually generate a public-private key pair.
- A `.csr` includes the metadata and public key of the RSA key pair.
    - Certificate signing request = Metadata + RSA public key

- Important commands
    - `openssl genrsa` is used to generate RSA private key (public and private RSA key pair).
    - `openssl rsa` is used to process RSA keys. i.e. retrieve public key from private key
    - `openssl req` is used to create certificate requests (CSR), or it can additionally create self signed certificates for use as root CAs for example.
        - `-x509` this option outputs a self signed certificate instead of a certificate request. This is typically used to generate a test certificate or a self signed root CA.
    - `openssl x509` is used to generate certificates by signing certificate requests.

Step by Step Guide

- **Step 1:** Creating Root Certificate or own Certificate Authority (CA)
- **Step 2:** Creating SSL Certificate signed using Root Certificate Private Key
- **Step 3:** Put all together and run a Website using the above certificate

Step 1: Creating Root Certificate or own Certificate Authority (CA)

1. Define a variable **CANAME**

```
# for Linux CANAME=MyOrg-RootCA # for Windows set CANAME=MyOrg-RootCA
```

2. Create private key for CA using OpenSSL

```
# for Linux openssl genrsa -aes256 -out $CANAME.key 4096 # for Windows
openssl genrsa -aes256 -out %CANAME%.key 4096
```

Refer the documentation for `openssl genrsa`.

3. Create Root Certificate for the CA

    ```
    # create certificate, 1826 days = 5 years # the following will ask for
    common name, country, state... # for Linux openssl req -x509 -new -nodes
    -key $CANAME.key -sha256 -days 1826 -out $CANAME.crt # for Windows
    openssl req -x509 -new -nodes -key %CANAME%.key -sha256 -days 1826 -out
    %CANAME%.crt
    ```

    > 💡 For **Root Certificate**, we don't need to 2 step process, where first we generate the CSR, and then sign that CSR to generate the final certificate. Here, we use the same command `openssl req` that we use for CSR, but with an additional `x509` option. This will output a self-signed certificate, which would be signed with the private key (generated in previous step) provided in the command itself.

4. Add the CA certificate to the trusted root certificates

    a. For **Windows,** just double click the `.crt` file

        i. Click **Install Certificate**

        ii. Choose any **Store Location**,

        iii. Select Place all certificates in the following store, choose **Trusted Root Certificate Authorities.**

    Verify it by running `certmgr.msc`.

    b. For **Linux (Ubuntu)**

    ```
    sudo apt install -y ca-certificates sudo cp $CANAME.crt
    /usr/local/share/ca-certificates sudo update-ca-certificates
    ```

    c. For **Linux (Fedora/CentOS)**

    ```
    sudo cp $CANAME.crt /etc/pki/ca-trust/source/anchors/$CANAME.crt sudo
    update-ca-trust
    ```

Step 2: Creating SSL Certificate signed using Root Certificate Private Key

1. Define a variable **MYCERT**

```
# for Linux MYCERT=MyServer # for Windows set MYCERT=MyServer
```

2. Create a SSL certificate for your web server. This will create a new RSA key pair for private and public keys.

```
# for Linux openssl req -new -nodes -out $MYCERT.csr -newkey rsa:4096 -keyout $MYCERT.key # for Windows openssl req -new -nodes -out %MYCERT%.csr -newkey rsa:4096 -keyout %MYCERT%.key
```

- `keyout` will output the private key
- `out` will output the CSR

However, both `.csr` and `.key` include the public key. Refer the documentation for `openssl req`.

3. Create a file to add **Subject Alternative Name (SAN)** to SSL certificate.

   a. Create a new ext file with `$MYCERT.v3.ext` name to add SAN properties.

      i. For windows, run `notepad %MYCERT%.v3.ext`

      ii. For Linux, run `touch $MYCERT.v3.ext`

   b. Copy the below content in that file.

```
authorityKeyIdentifier=keyid,issuer basicConstraints=CA:FALSE keyUsage = digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment subjectAltName = @alt_names [alt_names] DNS.1 = myserver.local DNS.2 = myserver1.local IP.1 = 192.168.1.1 IP.2 = 192.168.2.1
```

4. Sign the CSR using CA private key to generate final certificate

```
# for Linux openssl x509 -req -in $MYCERT.csr -CA $CANAME.crt -CAkey
$CANAME.key -CAcreateserial -out $MYCERT.crt -days 730 -sha256 -extfile
$MYCERT.v3.ext # for Windows openssl x509 -req -in %MYCERT%.csr -CA
%CANAME%.crt -CAkey %CANAME%.key -CAcreateserial -out %MYCERT%.crt -days
730 -sha256 -extfile %MYCERT%.v3.ext
```

Step 3: Put all together and run a Website using the above certificate

1. Create an ASP.NET Core application
2. Generate PFX file using `.key` and `.crt` file

```
# for Linux openssl pkcs12 -export -out $MYCERT.pfx -inkey $MYCERT.key -
in $MYCERT.crt # for Windows openssl pkcs12 -export -out %MYCERT%.pfx -
inkey %MYCERT%.key -in %MYCERT%.crt
```

3. Use the SSL certificate created in previous step

```
builder.WebHost.ConfigureKestrel(serverOptions => {
serverOptions.ConfigureEndpointDefaults(listenOptions => {
listenOptions.UseHttps("MyServer.pfx", "Password"); }); });
```

4. Verify that everything works.

References

1. Get a public key from an RSA private key