SSL Overview

These are just some notes which help me organize How Things Work. These are things which were unclear to me when I started and I now (hopefully) understand better. They may or may not be true.

Flow

1. Create an Internal CA
    1. Generate private key for Certificate of Authority (CA), encrypted (-des3)
    2. Generate Public Certificate for CA using *openssl req -x509*. Use long -days parameter (like 10 years)
2. Install CA on workstations
3. for each server/service, Create an SSL Configuration File
    1. Generate new private key, if needed
    2. Generate Certficate Signing Request (csr) using -days somewhere between 30 and 365 days
    3. Generate Server Certificate combining private key, CSR and signing with CA
    4. Combine .key and .crt files into .pem
    5. Copy .key, .crt and .pem to server and configure/restart services, see Deploy Server Certificate
4. Test
5. Prior to Server Certificate expiry
    1. Generate new private key (not required, but better security)
    2. Generate new csr
    3. combine .key and .crt into .pem
    4. copy .key, .crt and .pem to server, restart services

Definition and Notes

First, note we use only a small subset of the openssl commands. Also, note that there is significant overlap in what the various commands can do. For example, the *openssl req* command **can** create a key, or can use an existing key created with the *openssl genpkey* command. I chose to create the key with one command, then use it later.

Many commands have the *-noout -text* parameters which allow you to view an existing file. For example, a certificate created with the *openssl x509* command can be viewed with *openssl x509 -noout -text -in filename*

The file name suffixes are pretty arbitrary for Unix systems, but Microsoft products rely on them and it is good practice to use the suffixes so you know what they are. Thus, if you see a file with a .crt suffix, it is likely a Certificate.

Note that a .pem file is a special file which can contain the combination of other files. So, you can append a key file to a certificate file and store it in a PEM file. However, there are some programs which do not correctly handle that, so I generally keep all three (.key, .crt and .pem) on the server.

Key Files

A **key** is the private key of a public key pair. It will generally include the key data (a really big number), some Algorithm information (ie, I was created using RSA), and some metadata like a comment, label, or passphrase. Usually stored as base64 encoded data.

It can be generated by a lot of different programs, but is generally created using *openssl genpkey*. Generally, I encrypt the keys for a Certificate of Authority, but not for a Server Certificate.

Certificate

A certificate file contains the public counterpart to the private key. It will also include a Distinguised Name (DN) containing Common Name, Organization, etc...), a validity period (start and end dates). It may also include a digital signature from a Certificate of authority. Generally has a .crt suffix, or is part of a PEM along with its private key.

Certificate of Authority

A **CA** is a Certificate of Authority. This is a self-signed Certificate, just like a Server Certificate, but not signed by anyone. Instead, the **public portion** of the certificate (.crt file) is stored on all machines which want verified access to services signed by this key.

Server Certificate

This is my name for a signed certificate. A Server Certificate and its private key are generally copied to a server, and services are configured to use it for SSL connections (like https or smpts). Since the workstations all have the public CA information on them, they accept this certificate, since it is signed by the CA they know about.

Certificate Signing Request

A **csr** or a Certificate Signing Request is a special format file that is used to request a certificate which will be signed by a CA. It has the public key, the Distinguished Name, and a signature from the private key (to prove ownership)

Configuration Files

A **cnf** or **ext** file are basically configuration files used for processing. They actually have separate purposes, but can be combined into the same file. In my documents, I use *openssl.cnf* as a base configuration, then copy and modify that into an ext file for each service/server I'm creating a certificate for.

openssl.cnf contains the information for creating the public key, like all the Distinguished Name stuff, and the key usage and constraints sections.

There are different constraints and key usage parameters for CA's and Server Certificates. Instead of two files, I created my openssl.cnf file to have sections specifically designed for each function. For CA's, I use a section tailored to it (CA_default) and add the *-extensions CA_default* to the openssl command that creates a CA.

For Server Certificates, I copy openssl.cnf to servername.ext, then edit the two sections that need it. First, I need my DN to be different, so I set my CN to the be actual server name. Then, I remove the alt_names section and replace it with all of the names needed for the certificate. Now, when I create a cert, I do **not** use -config openssl.cnf, but instead use *-config servername.ext*

opnSense

I use opnSense a lot, so here are some notes about integrating it with your opnSense router.

Using opnSense to manage Certificates

I did not realize it when I started, but the opnSense router firmware can likely replace all of this stuff. You can create CA's, and use those CA's to sign Server Certificates. All in a nice little GUI. It can even reissue and replace a Server Certificate which is expiring.

Signing an opnSense Server Cert

opnSense can also issue a Certificate Signing Request (csr), allowing you to use certificates signed by your CA within opnSense, like for the Web GUI. Took me a minute to wrap my head around this, but is fairly simple.

1. Go to System | Trust | Certificates
2. Click the plus sign to add a new Certificate
    1. For Method, choose *Create a Certificate Signing Request*
    2. Enter a description and all the rest of the information
    3. Enter the DNS name of the router (from inside) into *Common Name*
    4. Open *Alternative Names* and fill in DNS domain names and IP's, if needed
    5. Click Save button
3. Edit the certificate
    1. Copy *Private key data* to a file with a .key suffix
    2. Copy *Certificate signing request* to a file with a .csr suffix
    3. Create an ext file
    4. Run the following command

```
openssl x509 -req -in csrFileName -CA CACertFileName -CAkey
CAKeyFileName -out crtFileName -days 365 -extfile extFileName -
extensions req_ext
```

    5. copy the contents of the newly created crtFile into the *Certificate data* block in opnSense
    6. Save

At this point, you have a new, signed certificate. You can add it as the Cert for your Web GUI by going

to **System | Settings | Administration** and changing *SSL Certificate*

From:
https://kb.unixservertech.com/ - **Unix Server Tech Knowledge Base**

Permanent link:
https://kb.unixservertech.com/software/openssl/internalca/overview

Last update: **2025/10/25 08:25**