

# Creating your own PKI using Cloudflare’s CFSSL

For those looking for a strait forward PKI, here’s how to get it, using Cloudflare’s CFSSL. Why CFSSL? If you’re looking for a simple solution, this is as simple as it can get: install Go Compiler, compile CFSSL and your done. Drawback? There’s little flexibility in terms or library versions. You get what Go offers. Now, CFSSL isn’t the most well documented application over there and yes, some configuration items aren’t document at all, so see right below on how to create a PKI using CFSSL.

Editor’s Note: This walkthrough was corrected to support the new “ca\_constraint” parameter.

## Installing CFSSL

CloudFlare SSL (CFSSL) is implemented on the Go programming language, which apart from being uncommon today, it quite straitfoward in terms of installing itself and deploy new applications, as the following steps demonstrate:

### 1.Install Go

```
export PATH=$PATH:/usr/local/go/bin

sudo tar -C /usr/local -xzf go1.5.3.linux-amd64.tar.gz
```

### 2.Test Go

```
go
```

### 3. Install CFSSL

```
go get -tags nopkcs11 -u github.com/cloudflare/cfssl/cmd/...

export PATH=$PATH=~/.cfssl/bin/
```

## Setting up you PKI

Now that you have CFSSL application ready we need to understand how it works.

## Relevant configuration files

There are usually two configuration files: CSR\_configuration and signing\_configuration. CSR configuration file contains the configuration for the key pair you’re about to create. Signing configuration as the name goes, sets up the configuration rules.

In order to create a PKI, three steps are involved:<!--nextpage-->

## Create your ROOT CA

For the root CA, check the following CSR configuration file (which we’ll call csr\_ROOT\_CA.json):

- csr\_ROOT\_CA.json

```
{
  "CN": "MY-ROOT-CA",
  "key": {
    "algo": "ecdsa",
    "size": 256
  },
  "names": [
    {
      "C": "UK",
      "L": "London",
      "O": "My Organisation",
      "OU": "My Organisational Unit Inside My Organisation"
    }
  ],
  "ca": {
    "expiry": "262800h"
```

```
}  
  
}
```

A few explanations are in order:

- The configuration file follows the X.509 naming scheme, so the following fields are required:
  - CN (Common Name) – The name of the entity. On the Root CA case, it’s the Root CA Name;
  - C (Country)
  - L (Location)
  - O (Organisation)
  - OU (Organisational Unit)
- Now, a number of specific fields are specific to CFSSL:
  - KEY – Defines the keys characteristics:
    - Algo – Specifies the algorithm. Can be ‘rsa’ or ‘ecdsa’, for RSA or ECDSA algorithms, respectively. Now, ECDSA is always recommended if legacy devices are not relevant, but this only applies to devices less than two or three years old. RSA shall be used otherwise.
    - size – Specifies the key size. 256 shall be used for ecdsa key. For RSA keys, 2048 or 4096 are the recommended values.
  - ca – Defines the CA characteristics and in this case the key validity, in hours, yes, in hours. In this case it’s 30 years (24x356x30), as the root authority shall last as long as you foreseen the security of the root key.

Now, actually create the ROOT CA:

```
cfssl gencert -initca csr_ROOT_CA.json | cfssljson -bare root_ca
```

This command will create a number of files:

- root\_ca.csr – The root ca certificate sign request, which doesn’t make sense for the root ca, and therefore will never be used. As the root CA is self signed.
- root\_ca.pem – The Root CA certificate. This is the file you and to distribute as much as possible.
- root\_ca.key – This is the root CA Key. Keep this file safe and secured, as if your life depends on it. For a public Root CA this is actually truth.

As the Root CA is self signed, we don’t need any other steps or configuration files, apart from the signing configuration files, which will be covered next.<!--nextpage-->

# Create your intermediate CA

This step is not actually mandatory, but corresponds to a best practice. The end goal of having an intermediate CA, is to have an intermediate step in terms of security. In this case, the Root CA key is kept in an offline machine, and only used when you need to sign an intermediate CA certificate.

To create an intermediate CA, we need the following configuration files:

- intermediate\_ca.csr – The certificate sign request for the Intermediate CA

```
{  
  "CN": "My-Intermediate-CA",  
  "key": {  
    "algo": "ecdsa",  
    "size": 256  
  },  
  "names": [  
    {  
      "C": "UK",  
      "L": "London",  
      "O": "My Organisation",  
      "OU": "My Organisational Unit Inside My Organisation"  
    }  
  ],  
  "ca": {  
    "expiry": "42720h"
```

```
}  
  
}
```

- `root_to_intermediate_ca.json` – This is the Root CA signing configuration file

```
{  
  "signing": {  
    "default": {  
      "usages": ["digital signature","cert sign","crl sign","signing"],  
      "expiry": "262800h",  
      "ca_constraint": {"is_ca": true, "max_path_len":0, "max_path_len_zero": true}  
    }  
  }  
}
```

EDIT: Previously the “`is_ca`” was used, but it has been deprecated since.

In this file goes some of the most relevant parameters you can set for a certificate:

- `usages` – Which usages are allowed to be performed by the certificate being signed. Options supported by CFSSL are the following:
  - “digital signature”,
  - “cert sign”,
  - “crl sign”,
  - “signing”
  - etc
- `is_ca` – this field is only applicable to generate intermedia CAs certificates, and allows the generated certificate to sign other certificates. If you leave this field on an end device certificate it will be rejected my most common browsers and operative systems.

Now, actually create your Intermediate CA:

```
cfssl gencert -initca csr_INTERMEDIATE_CA.json | cfssljson -bare intermediate_ca
```

This command will create a number of files:

- `intermediate_ca.csr` – The Intermediate CA certificate sign request.
- `intermediate_ca.pem` – The Intermediate CA certificate, not signed by anyone, and therefore useless.
- `intermediate_ca.key` – This is the Intermediate CA Key. Keep this file safe and secured.

Next, have the Intermediate CA certificate signed by the Root CA:

```
cfssl sign -ca root_ca.pem -ca-key root_ca-key.pem -config root_to_intermediate_ca.json inte
```

A single file, but most important, will be generated:

- `intermediate_ca.pem` – The Intermediate CA certificate, this time signed by the Root CA. Make sure you distribute this file.

This ends the generation of your Intermediate CA. Make sure you keep you Root CA Keys and configurations files safe and secure, as you won’t be needing them anytime soon.

Next, create end (or device certificates).<!--nextpage-->

## Create end certificates

Each device, or website (or server, depending on your needs) will have it’s own certificate, and the steps are mostly the same.

You’ll need the following configuration files:

- `csr_end_device.json` – The certificate sign request for the end device

```
{  
  "CN": "www.my_server.com",  
  "key": {  
    "algo": "ecdsa",  
    "size": 256
```

```
},
"names": [
{
"C": "UK",
"L": "London",
"O": "My Organisation",
"OU": "My Organisational Unit Inside My Organisation"
}
],
"Hosts": ["www1.my_server.com", "www2.my_server.com"]
}
```

The only novelty in this configuration is the “Hosts” directive. This directive allows adding further hostnames in the same certificate, by means of the “Subject Alternative Name” option.

- intermediate\_to\_client\_cert.json

```
{
  "signing": {
    "profiles": {
      "CA": {
        "usages": ["cert sign"],
        "expiry": "8760h"
      }
    },
    "default": {
      "usages": ["digital signature"],
      "expiry": "8760h"
    }
  }
}
```

With those two files, we’re finally ready to generate end certificates, which can be one on one single step, generating keys and signing it:

```
cfssl gencert -ca intermediate_ca.pem -ca-key intermediate_ca-key.pem -config intermediate_t
```

Finally, this generates two important files:

- end\_device.key – The all important end device private key
- end\_device.pem – The end device certificate


You can generate as many end device certificate, as long as you properly set up the csr configuration file.

There are other methods to generate certificates using CFSSL, but that’s for another day.



Also published on Medium.

PREVIOUS POST



NEXT POST

[Authenticated Public NTP server howto](#)

[MicroSD Micro reader USB 3.0 review](#)

