If you followed previous post, you should now have working *cfssl* configuration. Next step is to setup OCSP and API servers for revoking and issuing certificates with REST endpoints.

Update
If you don't set `auth_key` *in config.json, then any client that knows you API server address, can sign and issue certificates... config.json below has been updated.*

First, dependencies... I first choosed PostgreSQL database backend, but came to conclusion that it's huge overkill, so I'll use SQLite backend.

By official (https://github.com/cloudflare/cfssl/tree/master/certdb) certdb guide, you need to install goose (https://bitbucket.org/liamstask/goose/) for using their database migration script. If you follow their guide, skip next few lines/commands...

Now, you need to create SQLite database with required tables. This next lines are copied from **$GOPATH/src/github.com/cloudflare/cfssl/certdb/sqlite/migrations/001_CreateCertificates.sql**, from first section, and if they are different in your case, use ones from provided file

```
CREATE TABLE certificates (
  serial_number            blob NOT NULL,
  authority_key_identifier blob NOT NULL,
  ca_label                 blob,
  status                   blob NOT NULL,
  reason                   int,
  expiry                   timestamp,
  revoked_at               timestamp,
  pem                      blob NOT NULL,
  PRIMARY KEY(serial_number, authority_key_identifier)
);

CREATE TABLE ocsp_responses (
  serial_number            blob NOT NULL,
  authority_key_identifier blob NOT NULL,
  body                     blob NOT NULL,
  expiry                   timestamp,
  PRIMARY KEY(serial_number, authority_key_identifier),
  FOREIGN KEY(serial_number, authority_key_identifier) REFERENCES certificates(serial_number,
 authority_key_identifier)
);
```

Save in *sqlite.sql* file and execute following command

```
cat sqlite.sql | sqlite3 certdb.db
```

Now create JSON file with database connection config.
sqlite_db.json

```
{"driver":"sqlite3","data_source":"certdb.db"}
```

So, now you can run *cfssl* API server, but... If you want OCSP function also, you need to create certificate specifically for OCSP and include OSCP profile in *config.json*...

Sample *ocsp.csr.json*

```
{
  "CN": "OCSP signer",
  "key": {
    "algo": "ecdsa",
    "size": 256
  },
  "names": [
    {
      "C": "US",
      "ST": "CA",
      "L": "San Francisco"
    }
  ]
}
```

Now, you must expand section `default` from *config.json* to define OCSP server URL and also include OCSP profile into `profiles` section.

Modified *config.json*

```
{
  "signing": {
    "default": {
      "auth_key": "key1",
      "ocsp_url": "http://cfssl.lan.amos:8889",
      "crl_url": "http://cfssl.lan.amos:8888/crl",
      "expiry": "26280h"
    },
    "profiles": {
      "intermediate": {
        "auth_key": "key1",
        "expiry": "43800h",
        "usages": [
          "signing",
          "key encipherment",
          "cert sign",
          "crl sign"
        ],
        "ca_constraint": {
          "is_ca": true,
          "max_path_len": 1
        }
      },
      "ocsp": {
        "auth_key": "key1",
        "usages": [
          "digital signature",
          "ocsp signing"
        ],
        "expiry": "26280h"
      },
      "serverCA": {
        "auth_key": "key1",
        "expiry": "43800h",
        "usages": [
          "signing",
          "key encipherment",
          "server auth",
          "cert sign",
          "crl sign"
        ]
      },
      "server": {
        "auth_key": "key1",
        "expiry": "43800h",
        "usages": [
          "signing",
          "key encipherment",
          "server auth"
        ]
      },
      "client": {
        "auth_key": "key1",
        "expiry": "43800h",
        "usages": [
          "signing",
          "key encipherment",
          "client auth",
          "email protection"
        ]
      }
    }
  },
  "auth_keys": {
    "key1": {
      "key": "<16 byte hex private key>",
      "type": "standard"
    }
  }
}
```

And generate certificate for OCSP.

```
cfssl gencert -ca=server/server.pem -ca-key=server/server-key.pem -config=config.json -profile
="ocsp" ocsp.csr.json |cfssljson -bare ocsp/ocsp
```

server.pem is 'CA' for certificates. It can be rootCA or intermediateCA, or in my case server certificate that has `cert sign` and `crl sign` attributes.

Now, you are ready to run *cfssl* API server with OCSP endpoint...

```
cfssl serve -db-config=sqlite_db.json -ca=server/server.pem -ca-key=server/server-key.pem -con
fig=config.json -responder=ocsp/ocsp.pem -responder-key=ocsp/ocsp-key.pem
```

And that's it!

NOTE
If you need more issuers, then you need to run cfssl/OCSP server for each one. Also, you need different OCSP certificates, because each needs to be signed with issuer that you are using. In my case, that means that if I have two servers which I want to use for issuing client certificates, I need to run two *cfssl* servers/services...

Now, you can test it from localhost. First you need to create CSR with openssl and for that, first step is to create private key and CSR (certificate signing request). Here I'm using ECDSA algorithm for private key (for which you first need to create params file)...

```
mkdir ~/client
cd ~/client
openssl ecparam -name prime256v1 -out prime256v1.pem
openssl req -new -newkey ec:prime256v1.pem -nodes -keyout client.key -out client.csr
```

And now you can request sign for your CSR...

```
cfssl sign -remote "localhost:8888" -profile "client" client.csr |cfssljson -bare my-client
```

and if you don't have any output, you should have *my-client.pem* (certificate) and *my-client.csr* (which should be identical to one generated before).

Or, you can have *cfssl* server to generate certificate, private key and sign it.

client.csr.json should be like

```
{
  "CN": "client_username",
  "key": {
    "algo": "ecdsa",
    "size": 256
  },
  "names": [
    {
      "C": "US",
      "L": "San Francisco",
      "OU": "Some location",
      "ST": "CA"
    }
  ]
}
```

And *client.config.json*

```
{
   "auth_keys" : {
      "key1" : {
         "type" : "standard",
         "key" : ""<16 byte hex private key>"
      }
   },
   "signing" : {
      "default" : {
         "auth_remote" : {
            "remote" : "cfssl_server",
            "auth_key" : "key1"
         }
      }
   },
   "remotes" : {
      "cfssl_server" : "localhost:8888"
   }
}
```

```
cfssl gencert -config client.config.json -profile client client.csr.json |cfssljson -bare my-c
lient
```

As usual, you can check certificate with

```
openssl x509 -in my-client.pem -noout -text
```

And check Issuer and Subject fields...

And verify certificate

```
cat ../cfssl/rootCA/rootCA.pem ../cfssl/intermediateCA/intermediateCA.pem ../cfssl/server/serv
er.pem > server_chain.pem
openssl verify -CAfile server_chain.pem my-client.pem
```

And you should get OK result

Next post is about revoking certificates and running OCSP responder.