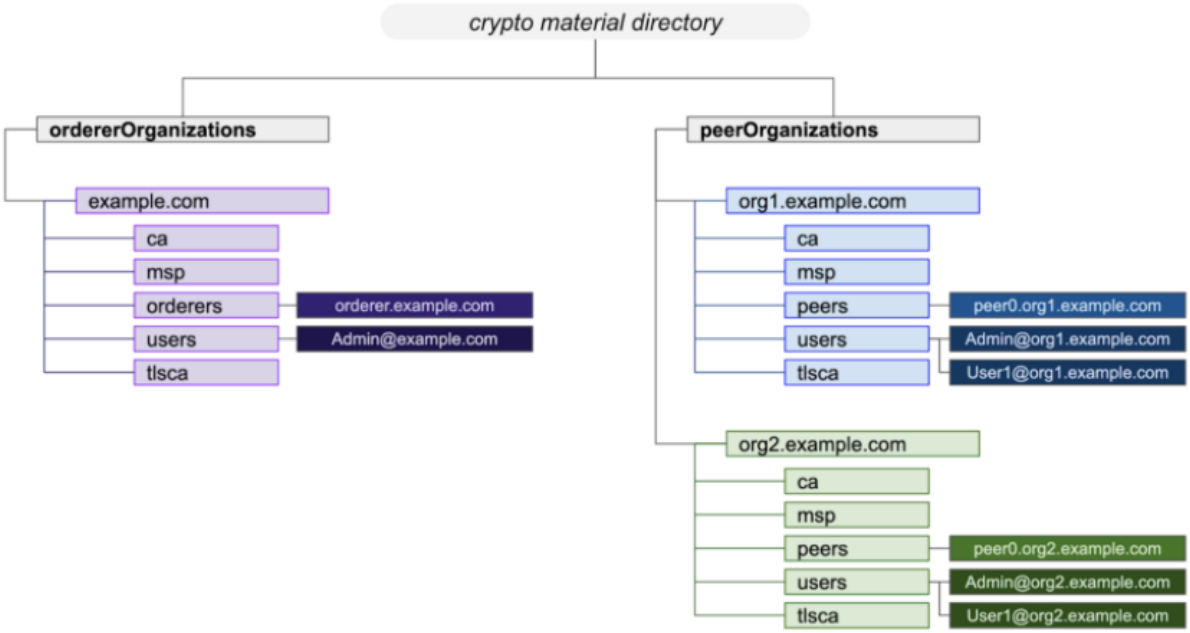
Crypto-config in Hyperledger Fabric

- Ari Sao & T Shee

The following cryptographic material for each node and user in the network:

A private key: This key is used to sign transactions and other communications within the network.
A public key: This key is used to verify the signatures made by the corresponding private key.
A certificate: This is a digital document that binds a public key to a distinguished name (DN) and is used to identify the node or user.
These generated cryptographic materials are used to secure communication between the nodes and users in the network and to authenticate transactions.



There are two kinds of organizations:

1. Orderer organization
2. Peer organization

There are two types of CA servers

1. Normal CA server which controls identity
2. TLS CA server which controls the TLS communication (No separate TLS CA server, network

components like peers and orderers play the TLS server role) (confusing? Will be cleared if you read the document line by line)

Let us investigate the crypto materials of the peer organization, (orderer organization have similar structure)

```
ubuntu@ip-172-31-31-50:~/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com
$ ls -l
total 28
drwxr-xr-x 2 ubuntu ubuntu 4096 May 10 03:03 ca
-rw-rw-r-- 1 ubuntu ubuntu 2959 May 10 03:03 connection-org1.json
-rw-rw-r-- 1 ubuntu ubuntu 2645 May 10 03:03 connection-org1.yaml
drwxr-xr-x 5 ubuntu ubuntu 4096 May 10 03:03 msp
drwxr-xr-x 3 ubuntu ubuntu 4096 May 10 03:03 peers
drwxr-xr-x 2 ubuntu ubuntu 4096 May 10 03:03 tlsca
drwxr-xr-x 4 ubuntu ubuntu 4096 May 10 03:03 users
```

ca/: a directory holding crypto material for CA of Org1
peers/: a directory holding a list of peers under Org1, and each peer has its own crypto material, of both identity and TLS
users/: a directory holding a list of network users under Org1, and each peer has its own crypto material, of both identity and TLS
tlsca/: a directory holding the TLS CA
msp/: it is the material joining a consortium network.

CA folder:

We first take a look at the CA of Org1. It is this CA issuing identity (certificate) to all entities within Org1.

```
ubuntu@ip-172-31-31-50:~/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com
$ ls -l ca/
total 8
-rw-rw-r-- 1 ubuntu ubuntu 863 May 10 03:03 ca.org1.example.com-cert.pem
-rw------- 1 ubuntu ubuntu 241 May 10 03:03 priv_sk
```

So, it contains the public key and private key of CA server of Org1

Peers folder:

Next we will examine the contents under peers folder of org1.

Organizations → PeerOrganizations → Organization Specific folder → peers folder → depending on the number of peers, each peer will have identity(msp) and tls folders

```
ubuntu@ip-172-31-31-50:~/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com
/peers/peer0.org1.example.com$ ls -l
total 8
drwxr-xr-x 7 ubuntu ubuntu 4096 May 10 03:03 msp
drwxr-xr-x 2 ubuntu ubuntu 4096 May 10 03:03 tls
```

Let us look into identity(msp) folder inside it

```
ubuntu@ip-172-31-31-50:~/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com
/peers/peer0.org1.example.com$ ls -l msp/
total 24
drwxr-xr-x 2 ubuntu ubuntu 4096 May 10 03:03 admincerts
drwxr-xr-x 2 ubuntu ubuntu 4096 May 10 03:03 cacerts
-rw-rw-r-- 1 ubuntu ubuntu  488 May 10 03:03 config.yaml
drwxr-xr-x 2 ubuntu ubuntu 4096 May 10 03:03 keystore
drwxr-xr-x 2 ubuntu ubuntu 4096 May 10 03:03 signcerts
drwxr-xr-x 2 ubuntu ubuntu 4096 May 10 03:03 tlscacerts
```

For identity, **the private secret key of this peer is stored in msp/keystore/**, while the **certificate is kept in msp/signcerts/**

Other folders like cacerts, tlscacerts, admincerts will have the public keys of the respective entity so that if a signed tx from that part is coming , then the peer can identify correctly and recognize them.

So, till now we have covered the ca and peers

Users folder:

Now let us look into users folder

```
ubuntu@ip-172-31-31-50:~/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com
$ ls -l users/
total 8
drwxr-xr-x 4 ubuntu ubuntu 4096 May 10 03:03 Admin@org1.example.com
drwxr-xr-x 4 ubuntu ubuntu 4096 May 10 03:03 User1@org1.example.com
```

Let's take Admin as an example. **Again** there are two directories. msp/ is for identity, and tls/ is for TLS material.

```
ubuntu@ip-172-31-31-50:~/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com
/users/Admin@org1.example.com$ ls -l
total 8
drwxr-xr-x 7 ubuntu ubuntu 4096 May 10 03:03 msp
drwxr-xr-x 2 ubuntu ubuntu 4096 May 10 03:03 tls
```

```
ubuntu@ip-172-31-31-50:~/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com
/users/Admin@org1.example.com$ ls -l msp/
total 24
drwxr-xr-x 2 ubuntu ubuntu 4096 May 10 03:03 admincerts
drwxr-xr-x 2 ubuntu ubuntu 4096 May 10 03:03 cacerts
-rw-rw-r-- 1 ubuntu ubuntu  488 May 10 03:03 config.yaml
drwxr-xr-x 2 ubuntu ubuntu 4096 May 10 03:03 keystore
drwxr-xr-x 2 ubuntu ubuntu 4096 May 10 03:03 signcerts
drwxr-xr-x 2 ubuntu ubuntu 4096 May 10 03:03 tlscacerts
```

Here is the private key and certificate for Admin.

```
ubuntu@ip-172-31-31-50:~/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com
/users/Admin@org1.example.com$ ls -l msp/keystore/
total 4
-rw------- 1 ubuntu ubuntu 241 May 10 03:03 priv_sk
ubuntu@ip-172-31-31-50:~/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com
/users/Admin@org1.example.com$ ls -l msp/signcerts/
total 4
-rw-rw-r-- 1 ubuntu ubuntu 810 May 10 03:03 Admin@org1.example.com-cert.pem
```

We inspect both admin and client certificate , one difference is the organization unit.

```
ubuntu@ip-172-31-31-50:~/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com
/users/Admin@org1.example.com$ openssl x509 -in msp/signcerts/Admin@org1.example.com-cert.pem -noout
-subject -issuer
subject=C = US, ST = California, L = San Francisco, OU = admin, CN = Admin@org1.example.com
issuer=C = US, ST = California, L = San Francisco, O = org1.example.com, CN = ca.org1.example.com
```

```
ubuntu@ip-172-31-31-50:~/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com
/users/User1@org1.example.com$ openssl x509 -in msp/signcerts/User1@org1.example.com-cert.pem -noout
-subject -issuer
subject=C = US, ST = California, L = San Francisco, OU = client, CN = User1@org1.example.com
issuer=C = US, ST = California, L = San Francisco, O = org1.example.com, CN = ca.org1.example.com
```

Tlsca folder:

Now, let us inspect the tlsca folder

It has exactly same structure as ca folder

```
ubuntu@ip-172-31-31-50:~/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com
$ ls -l tlsca/
total 8
-rw------- 1 ubuntu ubuntu 241 May 10 03:03 priv_sk
-rw-rw-r-- 1 ubuntu ubuntu 875 May 10 03:03 tlsca.org1.example.com-cert.pem
```

In a TLS communication, network components play a TLS server role. Therefore each network component is installed with a TLS server key and server certificate. As in Org1 we only have one peer, we can inspect the material inside peers/peer0.org1.example.com/tls/

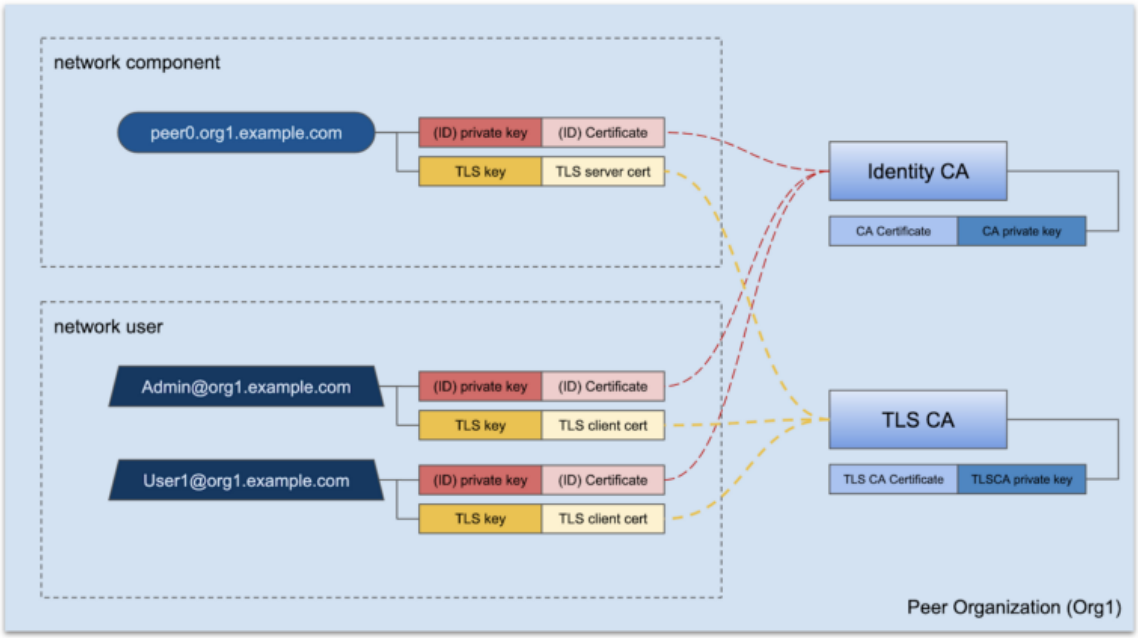
So, THIS IS VERY IMPORTANT tls folder inside peer is having server certificates to enable it act as TLS server.

```
ubuntu@ip-172-31-31-50:~/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com
/peers/peer0.org1.example.com$ ls -l tls/
total 12
-rw-rw-r-- 1 ubuntu ubuntu  875 May 10 03:03 ca.crt
-rw-rw-r-- 1 ubuntu ubuntu 1025 May 10 03:03 server.crt
-rw------- 1 ubuntu ubuntu  241 May 10 03:03 server.key
```

Network user plays a TLS client role. Therefore each network user is installed with a TLS client key and client certificate. We take Admin as an example here.

```
ubuntu@ip-172-31-31-50:~/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com
/users/Admin@org1.example.com$ ls -l tls/
total 12
-rw-rw-r-- 1 ubuntu ubuntu 875 May 10 03:03 ca.crt
-rw-rw-r-- 1 ubuntu ubuntu 834 May 10 03:03 client.crt
-rw------- 1 ubuntu ubuntu 241 May 10 03:03 client.key
```

So, when TLS communication happens then users sign the message using the client key and TLS server aka peers or orderer will validate that as they only have issued the certificates to the client.



Now we got all the concepts regarding crypto materials , now let us look into the yaml file that will help us to generate the certificates.

```
OrdererOrgs:
- CA:
    Country: NY
    Locality: NY-local
    OrganizationalUnit: IT
    PostalCode: "1000000"
    Province: NY-Prov
    StreetAddress: dummy address
  Domain: blockchain.net
  Name: Orderer
```

```
  Specs:
  - SANS:
    - localhost
  Template:
    Count: 1
    Hostname: orderer{{.Index}}
PeerOrgs:
- CA:
    Country: NY
    Hostname: ca
    Locality: NY-local
    OrganizationalUnit: IT
    PostalCode: "1000000"
    Province: NY-Prov
    StreetAddress: dummy address
  Domain: vion.blockchain.net
  EnableNodeOUs: true
  Name: Vion
  Template:
    Count: 1
    SANS:
    - localhost
  Users:
    Count: 1
- CA:
    Country: NY
    Hostname: ca
    Locality: NY-local
    OrganizationalUnit: IT
    PostalCode: "1000000"
    Province: NY-Prov
    StreetAddress: dummy address
  Domain: sim.blockchain.net
  EnableNodeOUs: true
  Name: Sim
  Template:
    Count: 1
    SANS:
    - localhost
  Users:
    Count: 1
```

The file is divided into two main sections: "OrdererOrgs" and "PeerOrgs".

The "OrdererOrgs" section defines the configuration for the orderer organization. Within this section, there are several fields:

"CA": This field defines the configuration for the certificate authority (CA) of the orderer organization. It includes information such as the organization's country, locality, and street address. It will be included in the x.509 certificate details.

"Domain": This field specifies the domain name of the orderer organization. Common name will be “hostname.domain” in the certificate

"Name": This field specifies the name of the orderer organization.

"Specs": This field defines the server name indications (SANs) for the orderer organization. In this case, it specifies "localhost" as the SAN.

"Template": This field defines the number of orderer nodes in the network and the hostname for each node. It is set to "1" and the hostname is "orderer{{.Index}}". The {{.Index}} is a placeholder that cryptogen will replace with an index number when generating the orderer nodes.

The "PeerOrgs" section defines the configuration for the peer organizations. Within this section, there are several fields:

"CA": This field defines the configuration for the certificate authority (CA) of the peer organization. It includes information such as the organization's country, locality, and street address.

"Domain": This field specifies the domain name of the peer organization. Common name will be “hostname.domain” in the certificate

"EnableNodeOUs": This field enables or disables the use of organizational units (OUs) for peer nodes.

"Name": This field specifies the name of the peer organization.

"Template": This field defines the number of peer nodes in the network and the server name indications (SANs) for each node. It is set to "1" and the SANs is "localhost".

"Users": This field defines the number of users in the peer organization. It is set to "1". This is apart of Admin user. So, two users will be created Admin and user1

So, If you want to extend number of organizations you can add in peer section (another struct)

```
- CA:
    Country: NY
    Hostname: ca
    Locality: NY-local
    OrganizationalUnit: IT
    PostalCode: "1000000"
    Province: NY-Prov
    StreetAddress: dummy address
  Domain: sim.blockchain.net
  EnableNodeOUs: true
  Name: Sim
  Template:
    Count: 1
    SANS:
    - localhost
  Users:
    Count: 1
```

Now let us look at fabric ca server.

Fabric ca server should be start with userid and password option. **(present in base.yaml)**

```
ca:
          image: hyperledger/fabric-ca:${CA_TAG}
          environment:
          - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
          - FABRIC_CA_SERVER_TLS_ENABLED=true
          - GODEBUG=netdns=go
          - LICENSE=accept
          command: sh -c 'fabric-ca-server start -b
       admin:adminpw -d'
```

Fabric-ca-server uses a db file to store all the certs.

It is created in the script called “generateartifcats.sh” and then mounted in docker-compose-template.yaml

- ./ca-sim/fabric-ca-server.db:/etc/hyperledger/fabric-ca-server/fabric-ca-server.db

Now we have to point out the certificate for ca created using previous step to this db. This is achieved using add_affilication scripts.

Thats it Tim, 🙂