简介

简单的说，cryptogen是用来生产fabric需要的证书的；这个生产过程是静态的。 和cryptogen工具对等的是CA服务，是一种动态的证书生产环境；在开发和测试阶段，在环境中不需要部署CA，因此可以简单的使用cryptogen工具，当然在运行环境中也可以不使用CA服务器，而继续使用cryptogen。

编译

```
$ cd $GOPATH/src/github.com/hyperledger/fabric
$ make cryptogen
$ cp build/bin/cryptogen /usr/bin
```

证书的生成

cryptogen需要一个crypto-config.yaml配置文件作为参数，它包含网络拓扑，并允许我们为组织和属于这些组织的组件生成一组证书和密钥。每个组织都有一个唯一的根证书（ca-cert），它将特定组件（peer和order service）绑定到该组织。通过为每个组织分配唯一的CA证书，我们正在模仿一个典型的网络，参与的成员将使用其自己的证书颁发机构。Hyperledger Fabric中的事务和通信由实体的私钥（keystore）签名，然后通过公钥进行验证signcerts。

先查看模版

```
$cryptogen showtemplate

# ------------------------------------------------------------------
----------
# "OrdererOrgs" - Definition of organizations managing orderer
nodes
# ------------------------------------------------------------------
----------
OrdererOrgs:
  # ----------------------------------------------------------------
------------
  # Orderer
```

```yaml
  # ---------------------------------------------------------------
------------
  - Name: Orderer
    Domain: example.com
    EnableNodeOUs: false

    # -------------------------------------------------------------
--------------
    # "Specs" - See PeerOrgs below for complete description
    # -------------------------------------------------------------
--------------
    Specs:
      - Hostname: orderer

# -----------------------------------------------------------------
----------
# "PeerOrgs" - Definition of organizations managing peer nodes
# -----------------------------------------------------------------
----------
PeerOrgs:
  # ---------------------------------------------------------------
------------
  # Org1
  # ---------------------------------------------------------------
------------
  - Name: Org1
    Domain: org1.example.com
    EnableNodeOUs: false

    # -------------------------------------------------------------
--------------
    # "CA"
    # -------------------------------------------------------------
--------------
    # Uncomment this section to enable the explicit definition of
the CA for this
    # organization.  This entry is a Spec.  See "Specs" section
below for details.
    # -------------------------------------------------------------
--------------
    # CA:
    #    Hostname: ca # implicitly ca.org1.example.com
    #    Country: US
    #    Province: California
    #    Locality: San Francisco
    #    OrganizationalUnit: Hyperledger Fabric
    #    StreetAddress: address for org # default nil
    #    PostalCode: postalCode for org # default nil
    # ------------------------------------------------------------------
---------------
    # "Specs"
    # ------------------------------------------------------------------
---------------
    # Uncomment this section to enable the explicit definition of
hosts in your
    # configuration.  Most users will want to use Template, below
    #
    # Specs is an array of Spec entries.  Each Spec entry consists
of two fields:
    #   - Hostname:   (Required) The desired hostname, sans the
domain.
    #   - CommonName: (Optional) Specifies the template or explicit
override for
    #                 the CN.  By default, this is the template:
    #
    #                              "{{.Hostname}}.{{.Domain}}"
    #
    #                 which obtains its values from the
Spec.Hostname and
    #                 Org.Domain, respectively.
    #   - SANS:       (Optional) Specifies one or more Subject
Alternative Names
    #                 to be set in the resulting x509. Accepts
template
    #                 variables {{.Hostname}}, {{.Domain}},
{{.CommonName}}. IP
    #                 addresses provided here will be properly
recognized. Other
    #                 values will be taken as DNS names.
    #                 NOTE: Two implicit entries are created for
you:
    #                     - {{ .CommonName }}
    #                     - {{ .Hostname }}
    # ------------------------------------------------------------------
---------------
    # Specs:
    #   - Hostname: foo # implicitly "foo.org1.example.com"
    #     CommonName: foo27.org5.example.com # overrides Hostname-
based FQDN set above
    #     SANS:
    #       - "bar.{{.Domain}}"
    #       - "altfoo.{{.Domain}}"
    #       - "{{.Hostname}}.org6.net"
    #       - 172.16.10.31
    #   - Hostname: bar
    #   - Hostname: baz

    # ---------------------------------------------------------------------------
    # "Template"
    # ---------------------------------------------------------------------------
    # Allows for the definition of 1 or more hosts that are created sequentially
    # from a template. By default, this looks like "peer%d" from 0 to Count-1.
    # You may override the number of nodes (Count), the starting index (Start)
    # or the template used to construct the name (Hostname).
    #
    # Note: Template and Specs are not mutually exclusive.  You may define both
    # sections and the aggregate nodes will be created for you.  Take care with
    # name collisions
    # ---------------------------------------------------------------------------
    Template:
      Count: 1
      # Start: 5
      # Hostname: {{.Prefix}}{{.Index}} # default
      # SANS:
      #   - "{{.Hostname}}.alt.{{.Domain}}"

    # ---------------------------------------------------------------------------
    # "Users"
    # ---------------------------------------------------------------------------
    # Count: The number of user accounts _in addition_ to Admin
    # ---------------------------------------------------------------------------
    Users:
      Count: 1

  # ---------------------------------------------------------------------------
  # Org2: See "Org1" for full specification
  # ---------------------------------------------------------------------------
  - Name: Org2
    Domain: org2.example.com
    EnableNodeOUs: false
    Template:
      Count: 1
    Users:
      Count: 1
```

仿照模板， 编写crypto-config.yaml

```
OrdererOrgs:
  - Name: Orderer
    Domain: example.com
    Specs:
      - Hostname: orderer
PeerOrgs:
  - Name: Org1
    Domain: org1.example.com
    Template:
      Count: 2 #peer数量
    Users:
      Count: 1 #用户数量
  - Name: Org2
    Domain: org2.example.com
    Template:
      Count: 2
    Users:
      Count: 2
```

生成证书和密钥

```
cryptogen generate --config=./crypto-config.yaml
```

执行命令后，将会在crypto-config目录下生成文件，内容如下：

```
tree -L 2 crypto-config
crypto-config
├── ordererOrganizations
│   └── example.com
└── peerOrganizations
    ├── org1.example.com
    └── org2.example.com
```

每一个org生成一个目录(example.com, org1.example.com, org2.example.com),
org目录下面的文件是一致的，如org1.example.com

```
tree -L 2 crypto-config/peerOrganizations/org1.example.com/
crypto-config/peerOrganizations/org1.example.com/
├── ca  # 包含org的根证书和Key
│   ├── ca.org1.example.com-cert.pem
│   └── priv_sk
├── msp # 包含org的根msp信息
│   ├── admincerts
│   ├── cacerts
│   └── tlscacerts
├── peers # 包含org下面的所有peer的证书
│   ├── peer0.org1.example.com
│   └── peer1.org1.example.com
├── tlsca  # 包含org的根tlsca证书和key
│   ├── priv_sk
│   └── tlsca.org1.example.com-cert.pem
└── users # 包含org下面所有用户的证书,配置文件指定1个用户
    ├── Admin@org1.example.com #管理员
    └── User1@org1.example.com
```

查看org2下的User1的证书

```
tree  crypto-
config/peerOrganizations/org2.example.com/users/User1\@org2.example
.com/
crypto-
config/peerOrganizations/org2.example.com/users/User1@org2.example.
com/
├── msp
│   ├── admincerts
│   │   └── User1@org2.example.com-cert.pem # 同org的admin证书
│   ├── cacerts
│   │   └── ca.org2.example.com-cert.pem  # 同org的ca根证书
│   ├── keystore
│   │   └── priv_sk
│   ├── signcerts
│   │   └── User1@org2.example.com-cert.pem # User2的MSP证书
│   └── tlscacerts
│       └── tlsca.org2.example.com-cert.pem # 同org的tlsca根证书
└── tls
    ├── ca.crt # 同org的tlsca根证书
    ├── client.crt # User2的 tls证书
    └── client.key # User2的 tls key
```

证书的扩展

当需要增加新的节点(peer)或者用户(user)的时候，需要为新节点/用户生成新的证书，当然老证书还得继续使用。这个时候需要用到extend命令。 修改./crypto-config.yaml配置文件，将Org2的用户扩展到2，新增Org3

```yaml
OrdererOrgs:
  - Name: Orderer
    Domain: example.com
    Specs:
      - Hostname: orderer
PeerOrgs:
  - Name: Org1
    Domain: org1.example.com
    Template:
      Count: 2
    Users:
      Count: 1
  - Name: Org2
    Domain: org2.example.com
    Template:
      Count: 4
    Users:
      Count: 4
  - Name: Org3
    Domain: org3.example.com
    Template:
      Count: 2
    Users:
      Count: 2
```

执行扩展子命令

```
cryptogen extend --config=./crypto-config.yaml
```

再次查看证书

```
$tree -L 2 crypto-config
crypto-config
├── ordererOrganizations
│   └── example.com
└── peerOrganizations
    ├── org1.example.com
    ├── org2.example.com
    └── org3.example.com


$tree -L 2 crypto-config//peerOrganizations/org2.example.com
crypto-config//peerOrganizations/org2.example.com
├── ca
│   ├── ca.org2.example.com-cert.pem
│   └── priv_sk
├── msp
│   ├── admincerts
│   ├── cacerts
│   └── tlscacerts
├── peers
│   ├── peer0.org2.example.com
│   ├── peer1.org2.example.com
│   ├── peer2.org2.example.com
│   └── peer3.org2.example.com
├── tlsca
│   ├── priv_sk
│   └── tlsca.org2.example.com-cert.pem
└── users
    ├── Admin@org2.example.com
    ├── User1@org2.example.com
    ├── User2@org2.example.com
    ├── User3@org2.example.com
    └── User4@org2.example.com
```

多了org3.example.com目录，org2.example.com下user增加到了4个，是没有更改已经签发的证书(包括根证书)，只是新增的需要的证书。