Membership Services v1.0 Design

*aka “COP”*

Membership Services in v1.0 of Hyperledger Fabric is renamed to COP. It is not an acronym. The name was selected because:

1. it provides police-like security functionality for Hyperledger Fabric, and
2. it is shorter and easier to say and write “COP” rather than “Membership Services v1.0” :-)

成员管理服务v1.0版本重命名为COP。 它不是一个首字母缩略词。 选择该名称是因为：

a）它为Fabric提供类似警察的安全功能

b）名词更短，更容易说和写

Note: For background information on the previous version of Membership Services (v0.6), see [Membership Services v0.6](https://docs.google.com/document/d/1mQj-MD47EKAN1AqnISqjL5CBiEBpBlZ9nCe3ls-LGTY).

The primary goals of COP are as follows.

1. Make it as pluggable and consumable as possible. It should clear how to completely implement your own COP or to replace some portion of the default implementation. For example, you may use your own CA for generating ECerts but still use TCerts for unlinkability and attributes for access control. In other words, you have the choice of using all, some, or none of the capabilities that COP offers.
2. Do not reinvent the wheel. For example, the default implementation of COP will use CFSSL (see<https://github.com/cloudflare/cfssl>) to generate ECerts, which also supports PKCS11 and is HA via a DB. The code to generate TCerts will be repackaged but will remain mostly the same.
3. Support a decentralized model in which there may potentially be multiple roots of trust. See FAB-359 and FAB 360.
4. Support standard protocols where possible. For example, the ACME protocol may be used to issue server-based certificates.

COP的主要目标如下。

1.使其尽可能可插拔和可使用。 它应该清楚展示如何完全实现自己的COP或替换默认实现的一些部分。 例如，您可以使用您自己的CA生成ECerts，但仍然使用TCerts作为访问控制的属性和不可链接性。 换句话说，您可以选择使用所有，一些或完全不用COP提供的功能。

2.不是新发明。 例如，COP的默认实现将使用CFSSL（请参阅https://github.com/cloudflare/cfssl）生成ECerts，同时支持PKCS11和像DB一样的HA。 生成TCerts的代码将被重新打包，但仍保持不变。

3.支持一种去中心化的模式，其中可能有多个根信任源。 参见FAB-359和FAB 360。

4.在可能的情况下支持标准协议。 例如，ACME协议可以用于发布基于服务器的证书。

In order to accomplish these goals, COP is being built as a reusable go library. The cop executable is a thin wrapper around this go library.

The remainder of this document is divided into three sections.

* COP commands

Describes the usage of the cop executable.

* COP APIs

Describes the APIs provided by the COP library, or references where you can find them.

* Use cases end-to-end flows

Shows how the CLI commands and APIs are used for various use cases, including bootstrap.

为了实现这些目标，COP正在构建为可重复使用的go 库。 可执行文件是一个go库的薄的封装。

本文档的其余部分分为三个部分。

●COP命令

描述COP可执行文件的用法。

●COP API

描述COP库提供的API，或者您可以找到它们的引用。

●端到端过程用例

显示CLI命令和API如何用于各种用例，包括引导。

## COP commands

Let COP\_HOME refer to the COP directory which is $HOME/.cop by default.

This section is divided into two sections: server and client commands.

COP命令

让COP\_HOME引用COP目录，默认为$ HOME / .cop。

本节分为两部分：服务器和客户端命令。

### 1.1 COP server commands

1.1.1) cop server init <csrJsonFile>

This command creates the key material for a root COP server.

The following is a sample <csrJsonFile>. Some fields are optional as indicated below.

1.1 COP服务器命令

1.1.1）cop server init <csrJsonFile>

此命令将创建根COP服务器的密钥材料。

以下是示例<csrJsonFile>。 某些字段是可选的，如下所示。

"CN”: “CA1”, // OPTIONAL

“hosts": [

"mycorp.com",

"www.mycorp.com"

],

"key": { // OPTIONAL

"algo": "ecdsa",

"size": 256

},

"names": [

{

"O": "Hyperledger",

"OU": "Fabric",

"L": "Raleigh",

"ST": "North Carolina",

"C": "US"

}

]

This command creates the following two files:

1. COP\_HOME/server-cert.pem - which contains the server’s certificate
2. COP\_HOME/server-key.pem - which contains the server’s private key

此命令创建以下两个文件：

a）COP HOME / server-cert.pem - 其中包含服务器的证书

b）COP\_HOME / server-key.pem - 其中包含服务器的私钥

NOTES:

1. You may manually initialize the server-cert.pem and server-key.pem files if you want to run a root COP server with a certificate issued by another CA.
2. The server-cert.pem file is used in the “cop client submitJoinRequest” described later to request that another root of trust be added to a blockchain network.
3. This is equivalent to running the following cfssl command in the COP\_HOME directory and renaming server.pem to server-cert.pem

cfssl gencert -initca config.json | cfssljson -bare server

注意：

1）如果要运行具有由另一个CA颁发的证书的根COP服务器，您可以手动初始化server-cert.pem和server-key.pem文件。

2）server-cert.pem文件在稍后描述的“cop客户端submitJoinRequest”中使用，以请求将另一个信任根添加到区块链网络。

3）这相当于在COP\_HOME目录中运行以下cfssl命令，并将server.pem重命名为server-cert.pem

cfssl gencert -initca config.json | cfssljson -bare服务器

1.1.2) cop server init <csrJsonFile> -remote <rootCopServerAddress> -u <user:pass>

This command is used to initialize an intermediate COP server. It is not strictly required that you run root and intermediate COP servers, but is strongly recommended as a best practice. This allows the root COP server’s private key to be offline when not needed, and also allows you to revoke a compromised intermediate COP certificate and issue another one without affecting the root COP server’s certificate.

The <rootCopServerAddress> is the address of the root COP server.

The <user:pass> is a user name and one-time password which is issued to you via the “cop client register” command described below.

As was described under 1.1.1, this also creates server-cert.pem and server-key.pem files in the COP\_HOME directory.

1.1.2）cop server init <csrJsonFile> -remote <rootCopServerAddress> -u <user：pass>

此命令用于初始化中间COP服务器。 并不严格要求运行根服务器和中间COP服务器，但强烈建议作为最佳实践。 这允许根COP服务器的私钥在不需要时脱机，并且还允许您撤销受损的中间COP证书，并发出另一个证书，而不影响根COP服务器的证书。

<rootCopServerAddress>是根COP服务器的地址。

<user：pass>是用户名和一次性密码，通过下面描述的“cop client register”命令发给您。

如1.1.1中所述，这还会在COP\_HOME目录中创建server-cert.pem和server-key.pem文件。

1.1.3) cop server start -config <configFile>

Start the COP server. The following is a sample config file.

启动COP服务器。 以下是一个示例配置文件。

{

“users": {

"admin": { // CUSTOMIZE ADMIN NAME

"pass": "adminpw", // CUSTOMIZE ADMIN ONE-TIME PASSWORD

“registrar”: true

}

}

"ca": { // OPTIONAL

"expiry": "262800h"

},

"signing": { // OPTIONAL

"default": {

"usages": ["cert sign"],

"expiry": "8000h"

} } }

### 1.2 COP client commands

1.2.1) cop client enroll <enrollmentID> <OTP> <cop-server-address> [<csrJsonFile>]

Enroll with a COP server. This returns an ECert for this client (peer, app, or end user).

This assumes that either <enrollmentID> is pre-registered by being in the COP server’s config file, or that “hfc register” is issued by a COP registrar first to generate the <OTP> (one time password) for the peer’s <enrollmentID>.

See section 1.1.1 for a sample <csrJsonFile>.

This command creates the following two files:

1. COP\_HOME/client-cert.pem - which contains the client’s certificate
2. COP\_HOME/client-key.pem - which contains the client’s private key

使用COP服务器注册。 这将返回此客户端（节点，app或最终用户）的ECert。

这假设通过在COP服务器的配置文件中预先注册<enrollmentID>，或者由COP注册器首先发出“hfc register”生成针对节点<enrollmentID>的<OTP>（一次性密码） 。

有关示例<csrJsonFile>，请参见第1.1.1节。

此命令创建以下两个文件：

a）COP\_HOME / client-cert.pem - 其中包含客户端的证书

b）COP\_HOME / client-key.pem - 其中包含客户端的私钥

1.2.2) cop client register <rrJsonFile> <cop-server-address>

Register a new identity with a COP server. The new identity information is contained in the <rrJsonFile> (the “register request” JSON file) which is of the following format:

向COP服务器注册新标识。 新的标识信息包含在<Json文件>（“注册请求”JSON文件）中，它具有以下格式：

{

“user": “user1”,

“group”: “group1”,

“attrs”: [

{“name”: “attr1Name”, “value”: “attr1Value”}

]

}

This returns the one time password in the response.

这将返回一次性密码。

1.2.2) cop client init manual <peer-ID> <keyFile> <certFile>

Manually initialize COP in the peer or orderer. This command may be used when you have a own key and certificate issued by your own CA. It is used instead of “peer/orderer init auto” when the COP server is not being used. For example, if you want to provide your own CA for issuing ECerts and you don’t want to use TCerts, you may use this command to initialize the peer identity information manually.

在peer或order中手动初始化COP。 当您拥有自己的CA颁发的密钥和证书时，可以使用此命令。 当COP服务器没有被使用时，它被用来代替“peer / orderer init auto”。

例如，如果您想要用你自己的CA以发出ECerts，并且您不想使用TCerts，您可以使用此命令手动初始化peer身份信息。

1.2.3) cop client submitJoinRequest <root.pem>

This command is invoked by a “sponser” participant to submit a request for another participant to join the blockchain network.

For example, assume companies C1 and C2 are already in the network and C3 wants to join. C3 runs the “cop server init” command to create the root.pem file. C3 then sends this file out-of-band to C1. In this case, C1 is functioning as C3’s sponsor to join the network.

Assuming that C1 approves of C3 joining, C1 invokes this “cop client submitJoinRequest” command. This invokes the COP system chaincode which places the request on the system ledger and depending on the configured join policy, it may require C2 to also approve. The COP system chaincode emits a custom event to which C2 is listening. Once notified, C2 invokes “cop client listJoinRequests” (which calls the COP system chaincode) to see the domain and certificate of who is requesting to join as well as who else has already approved. Based on this, C2 approves or denies the join request, using “cop client approveJoinRequest” or “cop client denyJoinRequest”, respectively, which again invokes COP system chaincode.

此命令由“sponser”参与者调用以提交对另一个参与者加入区块链网络的请求。

例如，假设公司C1和C2已经在网络中并且C3想要加入。 C3运行“cop server init”命令来创建root.pem文件。 C3然后将此文件安全发送到C1。在这种情况下，C1充当C3的加入网络的赞助商。

假设C1批准C3加入，C1调用这个“cop client submitJoinRequest”命令。这调用了一个将请求放置在系统账本上的COP系统chaincode，并且根据配置的加入策略，它可能需要C2也批准。 COP系统chaincode发出C2正在侦听的自定义事件。一旦通知，C2调用“cop client listJoinRequests”（其调用COP系统chaincode）以查看谁正在请求加入的域和证书以及谁已经批准。基于此，C2分别使用“cop client approveJoinRequest”或“cop client denyJoinRequest”来批准或拒绝加入请求，其再次调用COP系统chaincode.。

1.2.4) cop client listJoinRequests

Lists all current join requests along with a unique <joinRequestId>.

列出所有当前加入请求以及唯一的<join RequestId>。

1.2.5) cop client approveJoinRequest <joinRequestId>

Approves the join request.

1.2.6) cop client denyJoinRequest <joinRequestId>

Denies a join request.

1.2.7) cop client listParticipants

Lists the current participants in the blockchain network.

1.2.8) cop client updateJoinPolicy { add | remove | updatePolicy } <policyString>

Update the join policy associated with a group for a specific operation. The <policyString> may be something like the following:

1. “unanimous” if all current members must agree before the operation is performed
2. “percentage 50” if 50% of the current group members must agree
3. “identity <cert>” if it must be approved by a single identity

更新加入策略，与一组特定操作关联。 <policyString>可能类似于以下内容：

1.“一致”，如果所有当前成员在操作执行之前必须同意

2. “百分比50”如果50％的当前小组成员必须同意，

3.“身份<cert>”，如果它必须由单个身份批准

## **2. COP Library APIs**

The COP library APIs are defined in the “fabric/cop/api/api.go” file in the “feature/ca” branch currently.

### 2.1 Groups and Endorsements

This section provides an example of how the COP library can be used particularly in the collection of endorsements. First, some background.

COP maintains the following hierarchy for a blockchain network:

1) a network may have multiple participants (i.e. customers);

2) a participant may have multiple geographic locations;

3) a location may have multiple servers (i.e. peers or orderers).

本节提供了如何将COP库用于特别是在endorse收集的示例。

背景

COP为区块链网络维护以下层次结构：

1）网络可以具有多个参与者（即，客户）;

2）参与者可以具有多个地理位置;

3）位置可以具有多个服务器（即节点或order）。

It derives this hierarchy by being provided with two list of certificates: orderers and peer certificates, and having a canonical (but configurable) way of extracting the participant and location ID from each certificate. By default,

1. the participant ID is the issuer’s CN, unless self-signed, in which case it is the CN of the cert itself;
2. The location ID is the location from the cert.

它通过提供两个证书列表得到这个层次结构：order和peer证书，并且具有从每个证书提取参与者和位置ID的规范（可配置）方式。 默认，

a）参与者ID是发行者的CN，除非是自签名的，在这种情况下，它是证书本身的CN;

b）位置ID是来自证书的位置。

##### 2.2.1 Get a list of all participants with a peer process

获取peer进程的所有参与者的列表

**participants := cop.GetParticipants('peer')**

**for participant, \_ := range participants {**

**fmt.Println(participant.getName())**

**}**

This would print the CN of bank1, bank2, and bank3 since they all have peers.

##### 2.2.2 Create endorsement with policy and collect responses

使用策略创建endorse并收集回复

**// Create a group including 2 banks. The names must be participant names.**

**group, err := cop.NewGroup(<bank1ParticipantID>,<bank2ParticipanyId>)**

**// Create a new group endorsement which requires at least 51 percent of participants**

**// in this group to endorse the same response.**

**endorsement, err := group.NewEndorsement('{"percentage":51}')**

**// As responses are received from individual servers, call Endorse with a**

**// 'response' and 'signature' of type []byte and the serverName which provided**

**// provided the response. Verify the signature using the certificate associated**

**// with server 'serverId'.**

**complete, err := endorsement.Endorse( response, signature, serverId)**

**if complete {**

**// 51% of participants have agreed with this response.**

**// Return all of the signatures that agreed with this server**

**signatures := endorsement.getSignatures(serverId)**

**}**

## **3 Use Cases**

### 3.1 Bootstrapping orderers

There are several ways to bootstrap the ordering service as shown below. In each scenario, assume 4 orderers: O1, O2, O3, and O4.

3.1引导order

有几种方法来引导排序服务，如下所示。 在每个场景中，假定4个订阅者：O1，O2，O3和O4。

##### 3.1.1 Bootstrapping orderers with a single COP server & single root of trust

1. Start the COP server in O1

# orderer cop server init self-sign orderer1 // Any “cop server init” command will do

# orderer start // The orderer self-registers with the COP server running in process

1. Register O2 through O4. The following is done 3 times, once for each orderer.

# hfc register orderer <ordererEnrollmentID> // outputs a OTP (one-time password)

1. Start the COP client in O2 through O4. The following is done 3 times, once for each orderer.

# orderer cop client init auto <O1-server-address> <ordererEnrollmentID> <OTP>

3.1.1使用单个COP服务器和单个信任根来引导orders

1.在O1中启动COP服务器

＃orderer cop server init self-sign orderer1 //任何“cop server init”命令都会执行

＃orderer start //订阅者向正在运行的COP服务器自注册

2.注册O2到O4。 以下是执行3次，每个订阅者一次。

＃hfc register orderer <ordererEnrollmentID> //输出OTP（一次性密码）

3.在O2到O4中启动COP客户端。 以下是执行3次，每个订阅者一次。

＃orderer cop client init auto <O1-server-address> <ordererEnrollmentID> <OTP>

##### 3.1.2 Bootstrapping orderers with multiple COP servers & multiple roots of trust

1. Initialize the COP server in O1 through O4

# orderer cop server init self-sign <orderer> // Any “cop server init” command will do

1. Manually collect the file from the COP\_HOME/trust directory of O1 through O4 into a single directory. Each of the orderers will have a single uniquely named file based on the <orderer> name used in step #1. Copy all 4 files back to the COP\_HOME/trust directory of O1 through O4.
2. Start O1 through O4. When the orderer starts, the COP system chain code will recognize that there is no trust information on the system ledger, so it will read the COP\_HOME/trust directory and initialize it appropriately.

3.1.2引导具有多个COP服务器和多个信任根的orders

1.在O1到O4中初始化COP服务器

＃orderer cop server init self-sign <orderer> //任何“cop server init”命令都会执行

2.从O1到O4的COP\_HOME / trust目录中手动收集文件到单个目录中。 每个订阅者将具有基于步骤＃1中使用的<orderer>名称的单个唯一命名的文件。 将所有4个文件复制回O1到O4的COP\_HOME / trust目录。

3.启动O1到O4。 当订户启动时，COP系统chaincode将识别系统账本上没有信任信息，因此它将读取COP\_HOME / trust目录并相应地初始化它。