区块链密码节点机
使用说明书

鼎链数字科技（深圳）有限公司

2021 年 01 月

版本历史

版本	修订内容	修订	审核	日期
V1.0	初次修订	刘雄	周辉、孙子文	2021.1

目录

1. 区块链密码节点机概述 - 5 -
1.1. 简介 - 5 -
1.2. 组成部件 - 6 -
1.2.1. 硬件模块 - 6 -
1.2.2. 软/固件模块 - 7 -
1.3. 主要功能 - 7 -
1.3.1. 鼎链 Node 服务 - 7 -
1.3.2. 鼎链 Consenter 服务 - 8 -
1.3.3. 鼎链群组管理 - 8 -
1.3.4. 鼎链智能合约管理 - 8 -
1.4. 技术指标 - 9 -
2. 设备外观 - 9 -
3. 设备安装上架 - 10 -
3.1. 机箱导轨安装 - 11 -
3.2. 机架导轨安装 - 12 -
3.3. 节点机上架 - 12 -
4. 区块链密码节点机操作使用说明 - 12 -
4.1. 使用默认配置使用密码节点机 - 13 -
4.1.1. 更新软件许可文件 - 13 -
4.1.2. 启动密码节点机密钥管理工具 - 14 -
4.1.3. 登录区块链密码节点机 - 15 -
4.1.4. Consenter 服务启动 - 16 -
4.1.5. Node 服务启动 - 16 -
4.1.6. Jetty 容器服务启动 - 17 -
4.2. 密码节点机更多参数设置 - 17 -
4.2.1. 安装向导 - 17 -
4.2.2. 增加管理员 - 18 -
4.2.3. 增加操作员 - 19 -
4.2.4. 产生 ECC 密钥对 - 21 -
4.2.5. 产生对称密钥 - 22 -
4.2.6. 群组创建 - 23 -
4.2.7. 加入群组 - 24 -
4.2.8. 智能合约部署 - 25 -
4.2.9. 智能合约调用 - 26 -
4.2.10. 智能合约查询 - 26 -
4.2.11. 智能合约升级 - 27 -
5. 接口说明（客户端） - 28 -
5.1. 申请授权 - 28 -
5.1.1. 准备工作 - 28 -
5.1.2. 申请 token - 29 -
5.1.3. 获取 token - 30 -
5.1.4. 使用 token 发送请求 - 31 -
5.2. 业务调用 - 34 -

5.2.1. 调用智能合约（数据上链）.....- 34 -
5.2.2. 智能合约查询（链上数据查询）..... - 36 -

1. 区块链密码节点机概述

区块链密码节点机是将鼎链（聚龙链商业版）软件与密码硬件集成在一起的区块链软硬件一体化设备，内嵌密码功能并提供完备的区块链节点软件功能，支持快速构建基于鼎链的联盟区块链应用。该设备使用内置的 PCI-E 密码卡提供数据加解密、签名、验签、杂凑和真随机数等基础密码功能，通过构建区块链密码服务接口层，为区块链功能层实现节点认证、签名背书、账本生成、节点共识等服务提供密码支撑。

1.1. 简介

在一个完整的鼎链网络中，一般由若干 Node 节点、若干 Consenter 节点组成鼎链服务集群对应用系统服务器提供区块链服务。

鼎链 Node 服务、Consenter 服务可依据实际需要，部署在同一个或不同的区块链密码节点机上。当部署于不同的密码节点机上时，多个鼎链节点之间通过 RPC 进行通信。

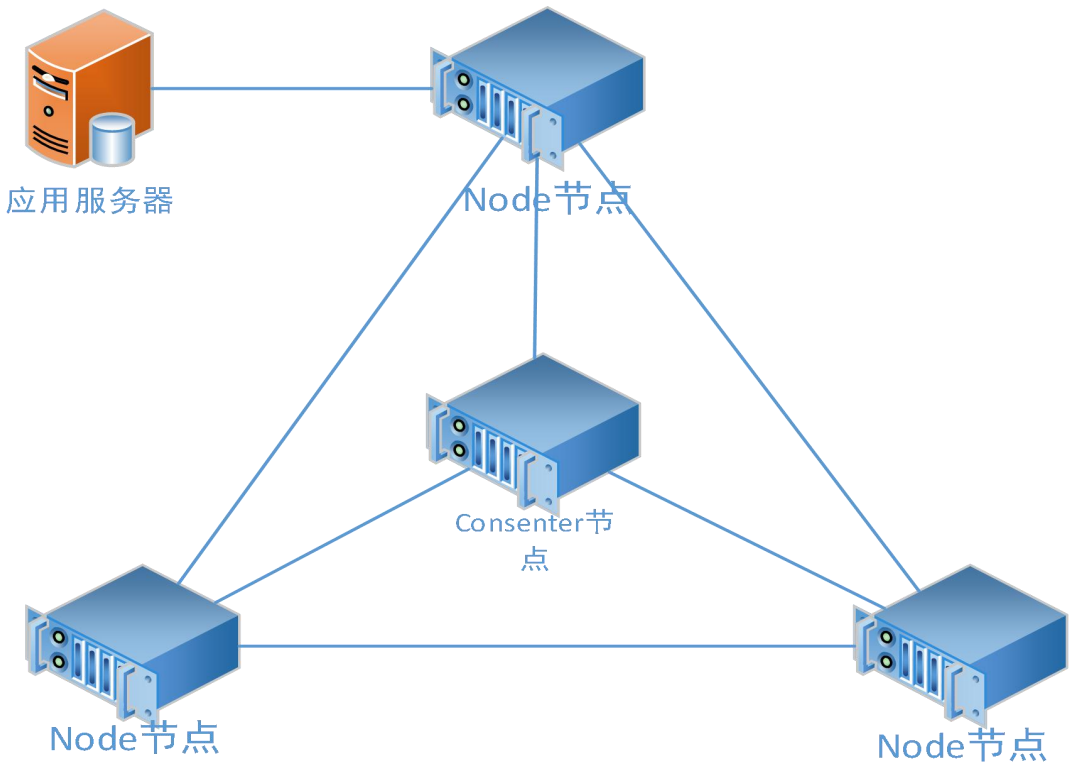


图 1 区块链系统部署结构图

1.2. 组成部件

区块链密码节点机由硬件、软/固件等模块部件组成。

1.2.1. 硬件模块

硬件模块由 PCI-E 密码卡、Ukey（智能密码钥匙）、服务器三大部分组成：

序号	部件组成	功能	备注
1	PCI-E 密码卡	SM2、SM3、SM4 密码算法，随机数产生，密钥管理等功能	采用带商用密码产品型号的产品
2	Ukey（智能密码钥匙）	提供管理员、用户认证和密钥分量备份数据存储功能	采用带商用密码产品型号的产品

3	主机	CPU	提供网络接口、电源、存储、信息运算、程序执行能力	Intel Xeon E5-2650V4 * 2
4		内存		32GB DDR4RECC * 4
5		硬盘		2.5 寸 600GSAS * 6
6		机箱		2U 机架式/ D×W×H：550mm x 438mm x 87mm
7		电源		550W 冗余电源
8		网卡		LSI9361-8I 万兆双口多模网卡 *1；板载千兆网口 * 2

1.2.2. 软/固件模块

软/固件模块由节点机密钥管理工具、鼎链软件和鼎链浏览器管理系统三大部分组成：

序号	部件组成	版本号	功能
1	节点机密钥管理工具	V3.7.3	PCI-E 密码卡初始化、密钥管理等功能
2	鼎链软件	V1.0	调用 PCI-E 密码卡，提供区块链密码服务，包括 Consenter 服务、Node 服务、智能合约、API 接口等
3	鼎链浏览器管理系统	V1.0	为区块链系统提供可视化的管理页面，包括链管理、节点管理、智能合约管理等模块

1.3. 主要功能

1.3.1. 鼎链 Node 服务

区块链主要服务由 Node 提供，包括分布式存储账本、背书、智能合约执行、权限策略检查、数据安全加密、身份验证等。

1.3.2. 鼎链 Consenter 服务

区块链 Consenter 服务，提供基于单例排序的区块链共识服务。

1.3.3. 鼎链群组管理

- 创建鼎链群组，即创建一条独立的链。处于同一群组中的节点可以共享账本数据，执行交易等。
- 加入鼎链群组，使当前 Node 节点加入鼎链群组中，完成与群组中成员的账本同步，并在接下来的时间内可以与群组成员进行交互。
- 列出已加入的群组，列出当前 Node 节点已经加入的群组。
- 查询群组信息，查询群组的相关信息。

1.3.4. 鼎链智能合约管理

- 部署智能合约，部署智能合约并将部署结果写入账本。
- 执行智能合约，执行智能合约中定义的逻辑，并将执行结果写入账本。
- 查询智能合约，执行智能合约中定义的查询逻辑，但不将执行结果写入账本。
- 升级智能合约，在智能合约修改后升级智能合约，并将结果写入账本。

1.4. 技术指标

- 使用的密码算法：SM2、SM3、SM4；
- 体积：438mm × 550mm × 87mm（宽×深×高）；
- 重量：23KG；
- 供电：交流 220V±10% 50~60HZ；
- 功率：550W；
- 环境适应性符合 GB/T 9813 规范要求；
- 储存温度：-50-65℃；
- 工作温度：0-50℃；
- 平均无故障工作时间（>=10000h）；
- 交易发送吞吐量：>1000TPS（不小于 3 台节点机）；
- 交易上链吞吐量：>1000TPS（不小于 3 台节点机）；
- SM4 密码算法：支持 ECB/CBC 工作模式，多任务加解密速率可达 200Mbps；
- SM3 密码算法：高速功能处理速率可达 600Mbps；
- SM2 密码算法：256 位 SM2 密钥对生产速度可达 10000 次/秒；256 位 SM2 算法数字签名速度可达 20000 次/秒；256 位 SM2 算法数字签名验证速度可达 12000 次/秒。

2. 设备外观

区块链密码节点机外观如下图所示（以实物为准）。

图 2 密码节点机外观实物图-1

图 3 密码节点机外观实物图-2

3. 设备安装上架

区块链密码节点机在安装使用前，请先检查对照实物与配件清单是否一致，并应检查密码节点机主机的拆卸存迹封条是否有拆卸过的痕迹。在确保整机完整的情况下完成设备安装、上电使用。

区块链密码节点机配件清单：

序号	部件名称	数量	备注
1	主机（预装操作系统和区块链节点）	1 台	

2	电源线	2 根	
3	导轨	2 条	
4	智能密码钥匙-管理员 UKey	3 个	
5	智能密码钥匙-操作员 UKey	1 个	
6	光盘	1 张	

3.1. 机箱导轨安装

滑轨内衬用单侧用 4 颗 M4*4 盘头螺钉固定在机箱侧面。

3.2. 机架导轨安装

前后各用两颗 M5 大扁头螺钉将滑轨外衬固定在机柜立柱上，左右各装一条。

3.3. 节点机上架

滑轨外衬固定好后，将装好内衬的机箱抬平，对准外衬白色胶件口缓慢推入，完成上架。

4. 区块链密码节点机操作使用说明

密码节点机出厂时已经对设备进行过初始化，客户使用默认配置直接启动相关服务即可使用密码节点机，也可以按照实际情况进行更

多参数的设置。

密码节点机出厂配置：

序号	配置项	参数
1	鼎链安装主目录	/opt/topchain
2	JDK	/usr/lib/jvm
3	密码模块管理员	默认 3 个管理员用户，口令 12345678
4	密码模块操作员	默认 1 个操作员用户，口令 12345678
5	对称密钥	预生成 5 个索引为 1-5 的对称密钥
6	非对称密钥	预生成 5 个索引为 1-5 的非对称密钥对
7	Consenter 服务	已默认部署了 Consenter 服务
8	Node 服务	已默认部署了 Node 服务
9	群组	已默认创建/加入名为 myGroup 的群组
10	智能合约	已默认部署名为 mycc、版本是 1.0.0 的智能合约

4.1. 使用默认配置使用密码节点机

4.1.1. 更新软件许可文件

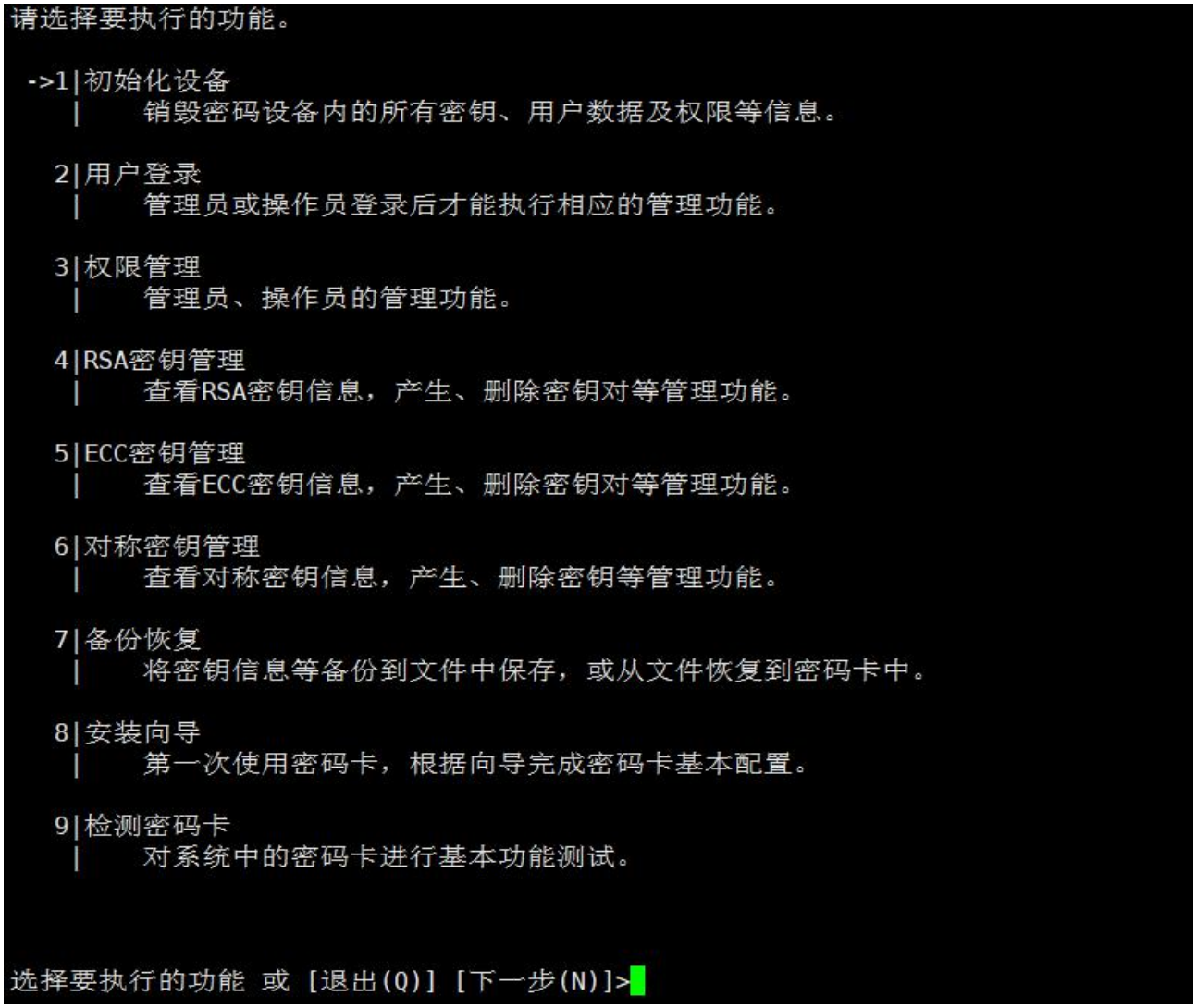
设备上电后进入系统，客户根据实际使用环境，修改 IP 地址，并将实际使用的 IP 地址发送给鼎链数科技术支持人员，我们会重新生成许可文件，客户需要替换鼎链安装主目录/node/msp/sample.julongchain.org/msp/lic 目录下的同名文件。

<!>提示：如果客户能提前规划使用环境的 IP，可以预先发给我们，在出厂初始化时我们将许可文件生成，省去客户的操作。

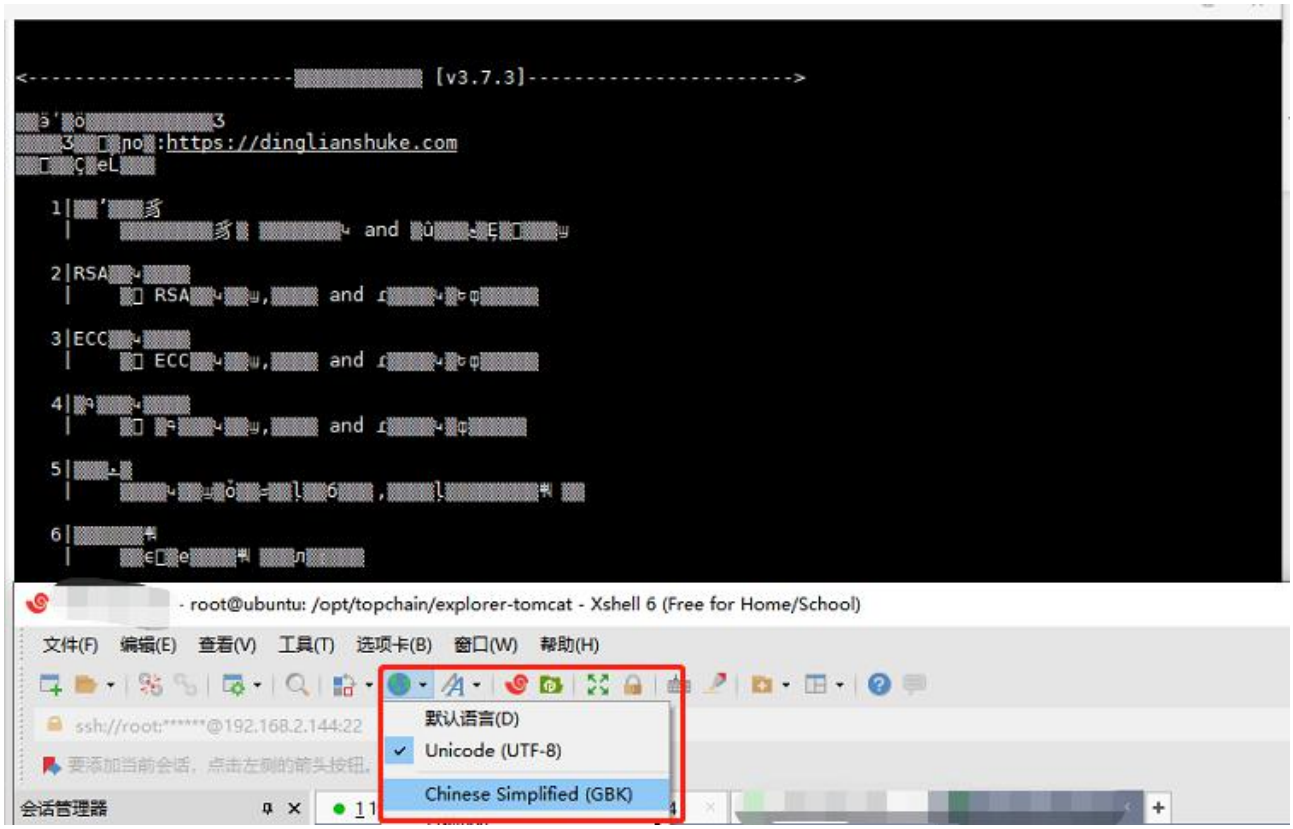
<!>提示：该步骤只有首次使用节点机时才需要，后续密码机关机后重启使用，从步骤 4.1.2 开始。

4.1.2. 启动密码节点机密钥管理工具

进入鼎链安装主目录/pcie/mngmnt，运行启动密码节点机密钥管理工具可执行文件（./swcsmmgmt），运行后终端显示密码节点机密钥管理工具界面如下：



<!>提示：如果打开密钥管理工具页面出现乱码，修改编码格式为 GBK。

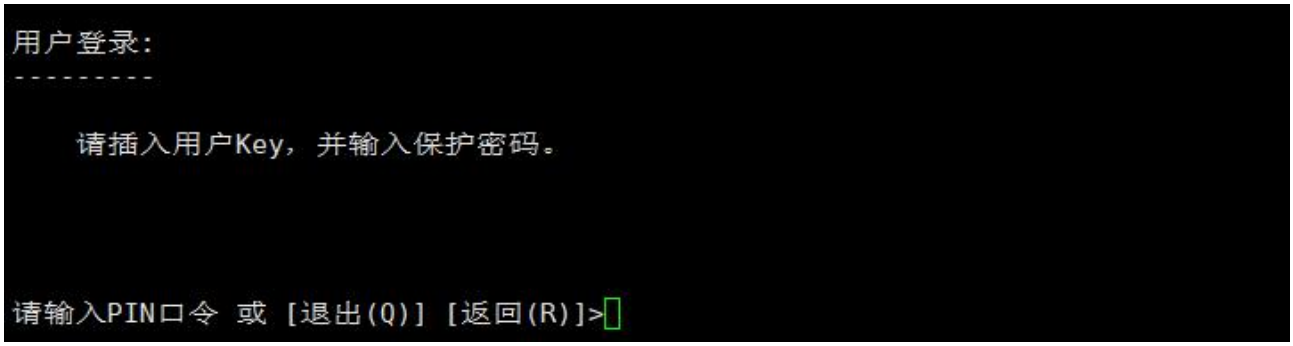


4.1.3. 登录区块链密码节点机

前置条件：4.1.2 启动密码节点机密钥管理工具。

✓ 在密码节点机密钥管理主页选择“2 用户登录”功能，进入用户登陆；

✓ 插入用户 U-KEY 并输入正确的口令后，即可完成用户登录



4.1.4. Consenter 服务启动

前置条件：4.1.3 用户已经登录区块链密码节点机。

- ✓ 进入鼎链主目录/consenter
- ✓ 执行命令

  java -jar topchain-1.0.jar consenter start

- ✓ 得到类似日志则 Consenter 启动成功

```
[CspManager][INFO] Initialize CSP dxctgmt0018
[Msp][INFO] signingIdentityInfo publicSigner size:688
[Registrar][INFO] Starting system group 'systemGroup' with g
be singleton
[Consumer][INFO] Consumer start
[StartCmd][INFO] Arg: start
[ConsenterServer][INFO] consenter service start, port:7050
```

<!>提示：如果当前节点只是 node 节点，则不需要启动 Consenter。

4.1.5. Node 服务启动

前置条件：4.1.3 用户已经登录区块链密码节点机。

- ✓ 进入鼎链主目录/node
- ✓ 执行命令

  java -jar topchain-1.0.jar node start

- ✓ 得到类似日志则 Node 启动成功

```
[NodeGrpcServer][INFO] NodeGrpcServer start, port: 7051
[EventGrpcServer][INFO] EventGrpcServer start, port: 7053
[SmartContractGrpcServer][INFO] SmartContractGrpcServer start, port: 7052
[NodeServer][INFO] Process name: 8518@ubuntu
[NodeServer][INFO] Process id: 8518
[Node][INFO] Node Command end
```

<!>提示：如果当前节点只是 Node 节点，需要进入鼎链主目录/node/config，修改 node.yaml 的 consenterAddress 参数改为实际的 Consenter 地址，完成 Node 和 Consenter 的组网过程。

4.1.6. Jetty 容器服务启动

- ✓ 进入鼎链主目录/node/jetty
- ✓ 执行命令

  java -jar start.jar

- ✓ 得到类似日志则 Jetty 容器服务启动成功

4.2. 密码节点机更多参数设置

4.2.1. 安装向导

在第一次使用密码节点机或需要对密码节点机初始化时，可以使用密码节点机密钥管理工具的安装向导功能，逐步完成对密码卡的基本配置。具体操作步骤可参考本章节其他管理操作说明。

前置条件：4.1.2 启动密码节点机密钥管理工具；4.1.3 使用管理员登录成功。

安装向导提供以下主要配置功能：

- ✓ 初始化密码设备：清空所有密钥及管理信息。
- ✓ 增加管理员：U-KEY 访问控制模式下，为保证设备的安全性、可靠性，及正常使用所有功能，建议设置 3 个管理员。
- ✓ 增加操作员：U-KEY 访问控制模式下，用于启动密码服务。
- ✓ 密钥管理：产生签名密钥对或加密密钥对并保存在密码设备内部。

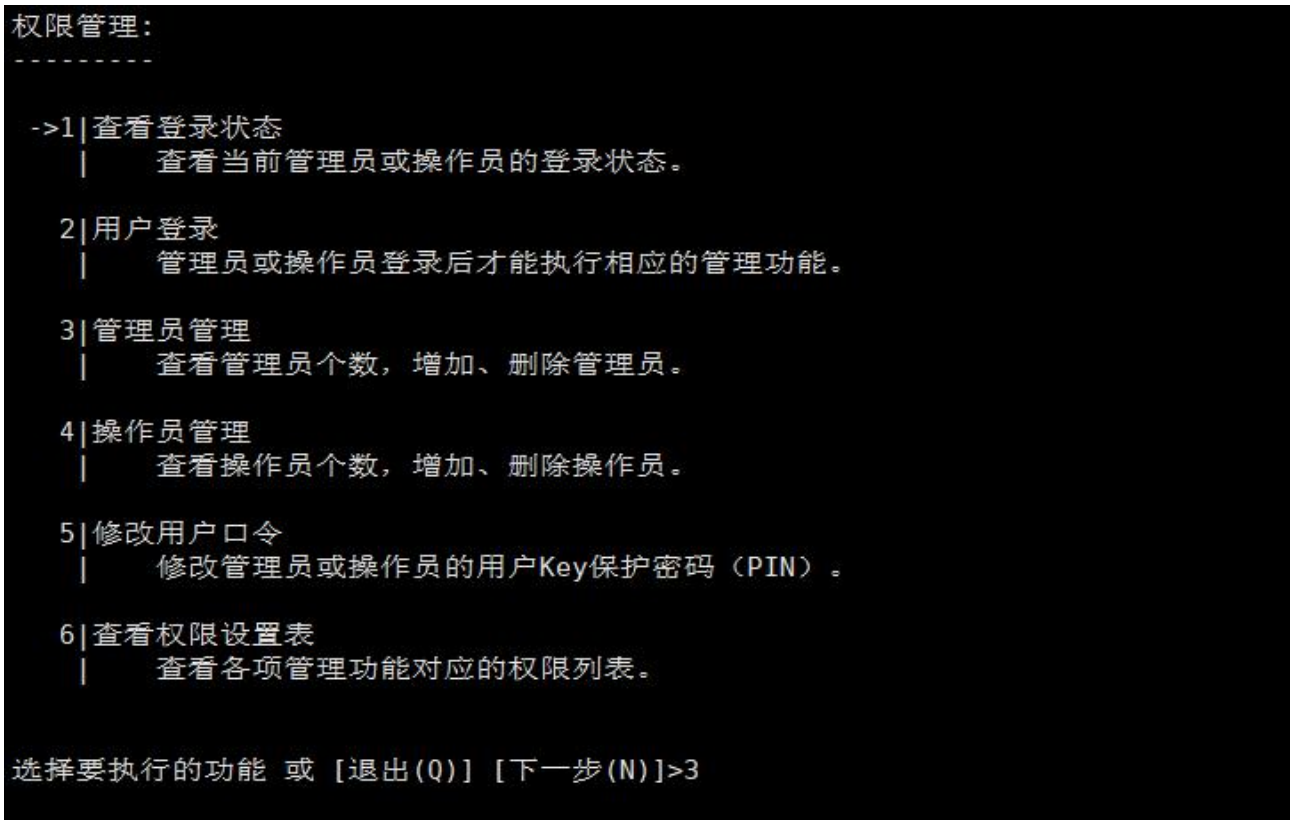
✓ 备份密钥信息：将密钥等重要信息加密后备份到文件中并妥善保管。

<!>安全提示：执行安装向导及初始化设备会清空密码模块的数据，包括密钥信息、密码用户信息等，请谨慎操作。

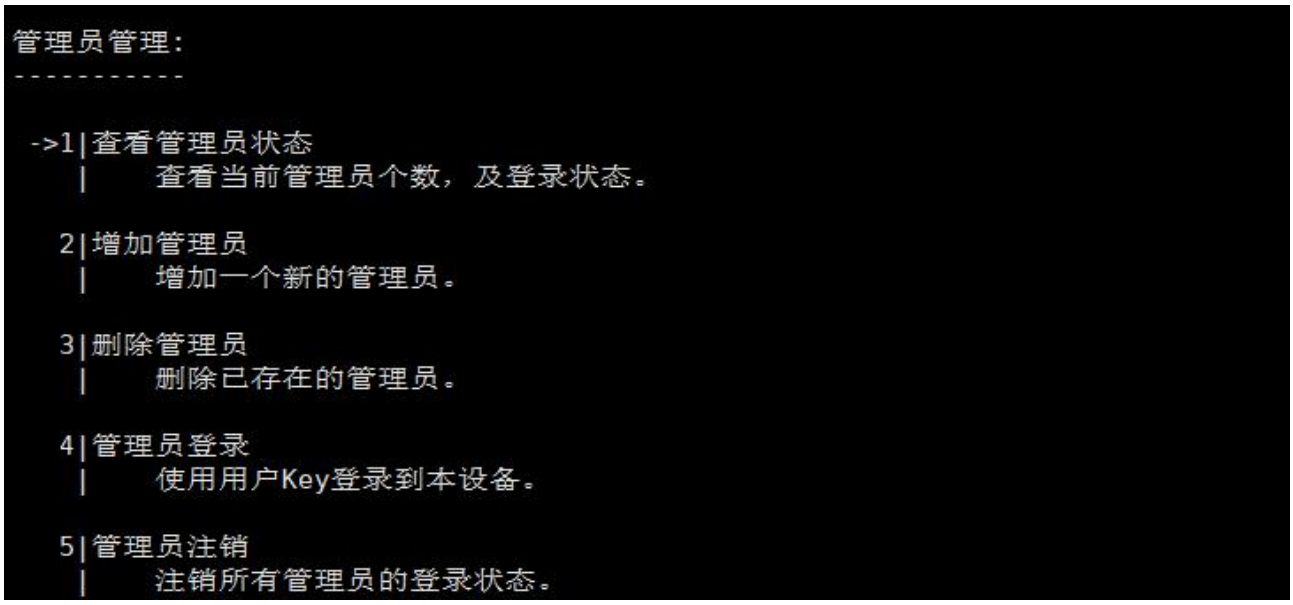
4.2.2. 增加管理员

前置条件：4.1.2 启动密码节点机密钥管理工具；4.1.3 使用管理员登录成功。

✓ 在密码节点机密钥管理主页选择“3 权限管理”功能，进入权限管理；



✓ 在权限管理界面选择“3 管理员管理”功能进入管理员管理；



✓ 在管理员管理界面选择“2 增加管理员”功能，插入管理员 UKEY 输入正确的口令后，即可完成新增管理员功能；

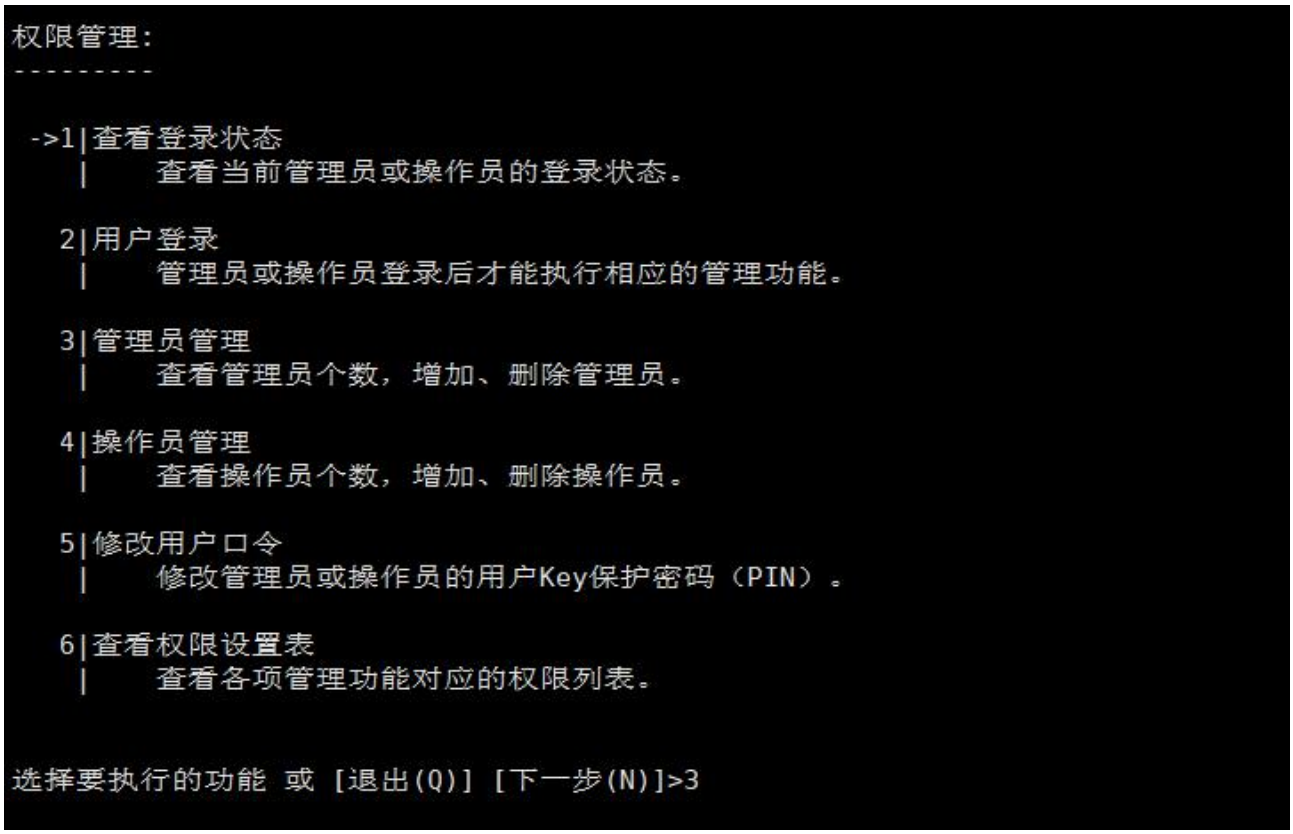
<!>安全提示：该密码设备在设计时支持 1 到 5 个管理员，为保证密码设备的安全性及可靠性，建议设置 3 个管理员。

<!>安全提示：在出厂时所有用户 U-KEY 的保护口令均被初始化成出厂值，为保证系统的安全性，请及时通过“修改用户口令”功能修改该口令。

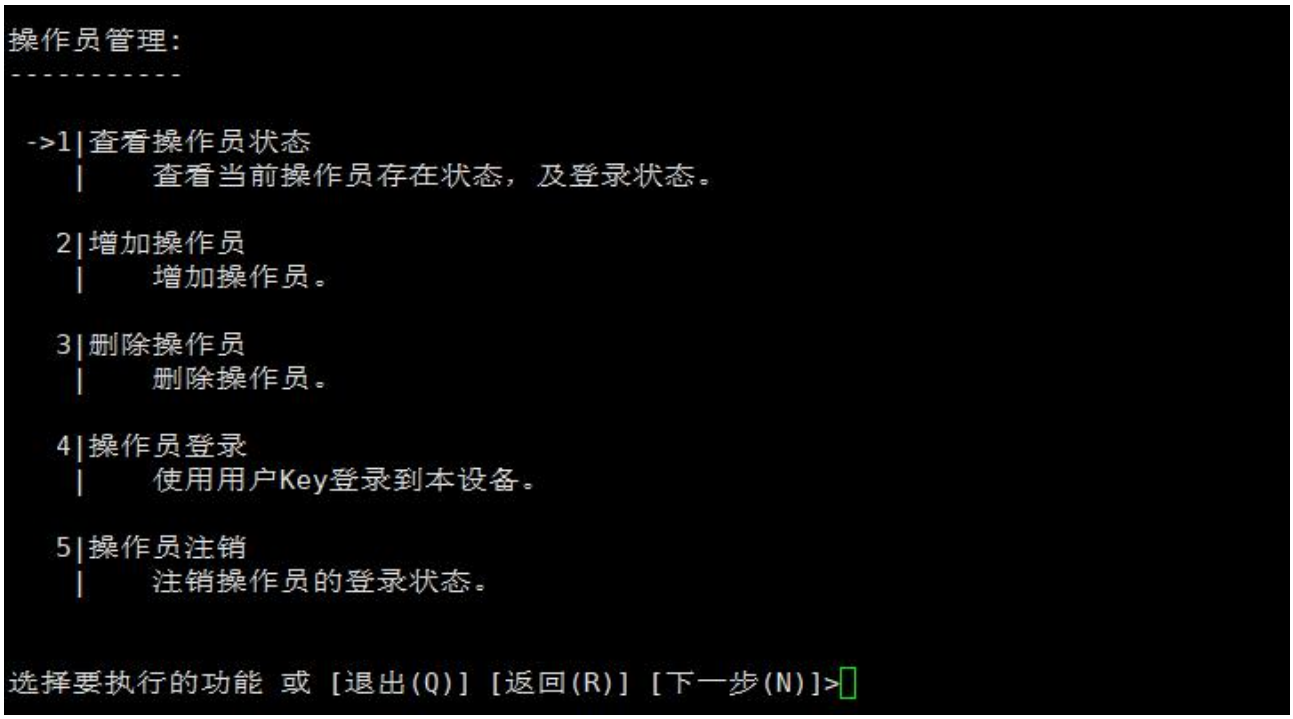
4.2.3. 增加操作员

前置条件：4.1.2 启动密码节点机密钥管理工具；4.1.3 使用管理员登录成功。

✓ 在密码节点机密钥管理主页选择“3 权限管理”功能，进入权限管理；



✓ 在权限管理界面选择“4 操作员管理”功能进入操作员管理；



✓ 在操作员管理界面选择“2 增加操作员”功能，插入操作员 U-KEY 输入正确的口令后，即可完成新增操作员功能；

<!>安全提示：为保证设备的可扩展性，本设备支持 1 个以上的操作员，但正常情况下建议只设置 1 个操作员。

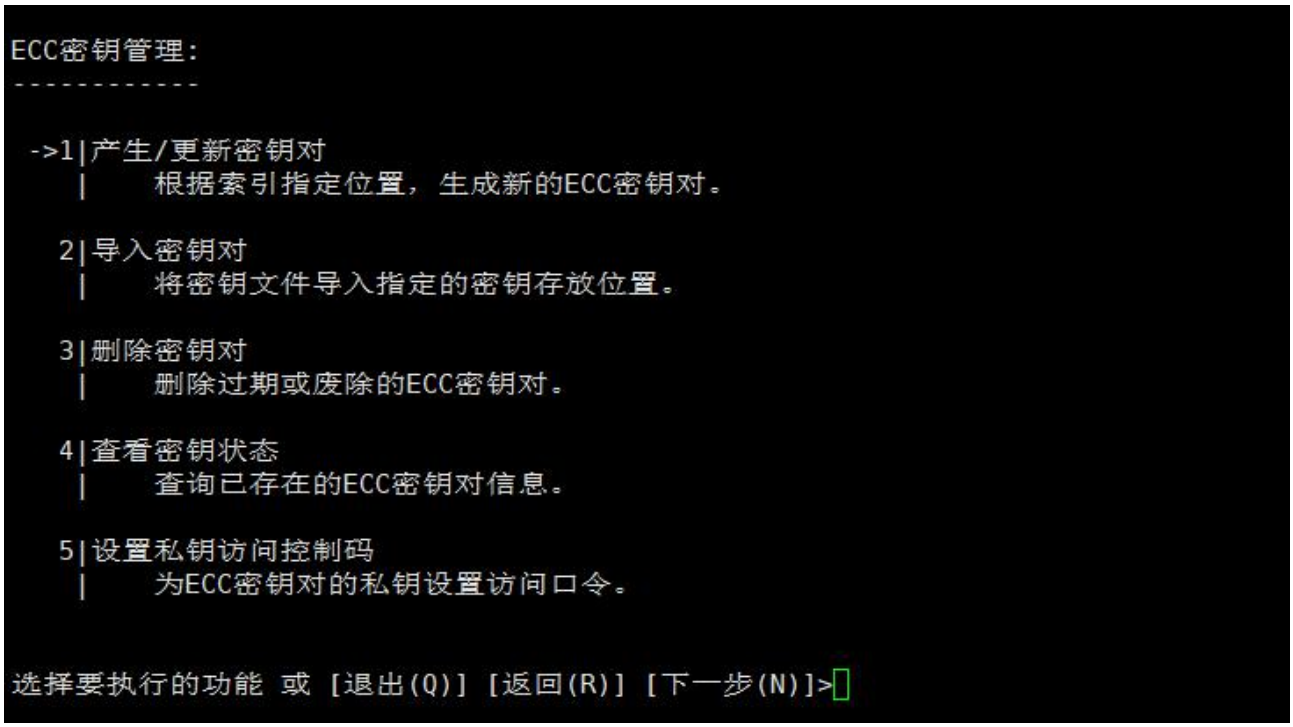
<!>安全提示：在出厂时所有用户 U-KEY 的保护口令均被初始化成出厂值，为保证系统的安全性，请及时通过“修改用户口令”功能修改该口令。

4.2.4. 产生 ECC 密钥对

前置条件：4.1.2 启动密码节点机密钥管理工具；4.1.3 使用管理员登录成功。

本设备支持 ECC 双密钥体制，每个索引位置对应两对 ECC 密钥对，分别是签名密钥对和加密密钥对，签名密钥对主要用于数字签名，加密密钥对一般用于数字信封或保护会话密钥的安全。具体的产生步骤如下：

✓ 在密码节点机密钥管理主页选择“5 ECC 密钥管理”功能，进入 ECC 密钥管理；



✓ 在 ECC 密钥管理界面选择“1 产生/更新密钥对”，根据提示的密钥索引范围，指定密钥位置；

```
产生 ECC 密钥对：
---------------
根据索引指定位置，生成新的 ECC 密钥对。

输入密钥索引<1~500>，或 [退出(Q)] [返回(R)] [下一步(N)]>
```

✓ 选择密钥用途，也可以选择仅产生签名密钥对或加密密钥对，产生密钥成功。

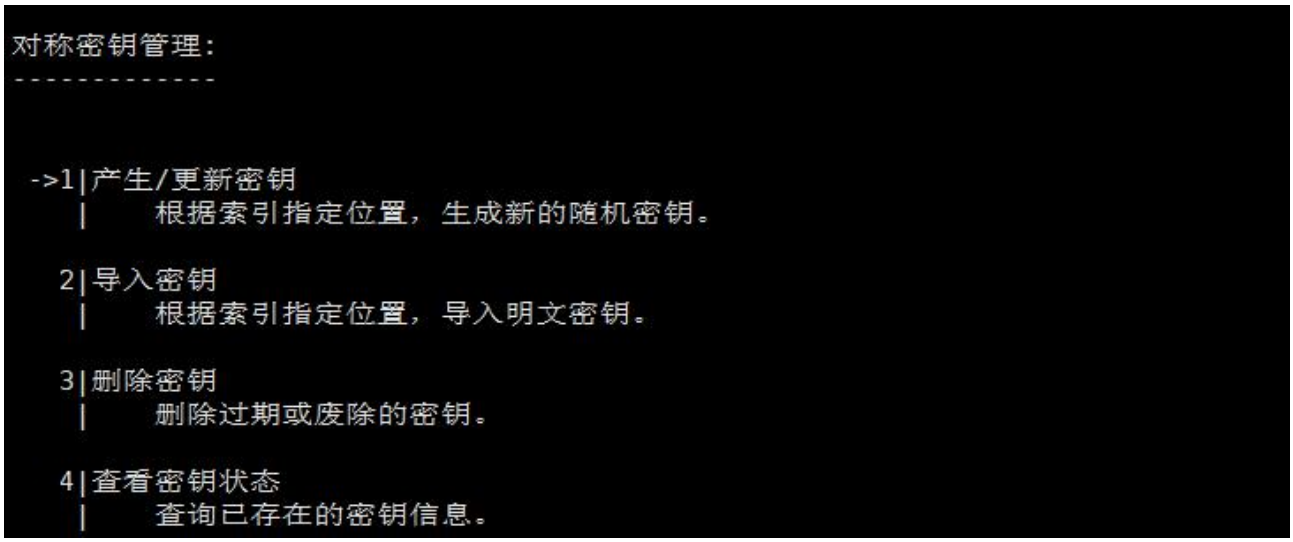
```
产生 ECC 密钥对：
---------------
请选择要产生的密钥对的用途，并在指定位置生成密钥对。
  _|_____________________________________
  1|  签名密钥
  2|  加密密钥
  3|  签名密钥及加密密钥

选择密钥用途，或 [退出(Q)] [返回(R)] [上一步(P)] [下一步(N)]>
```

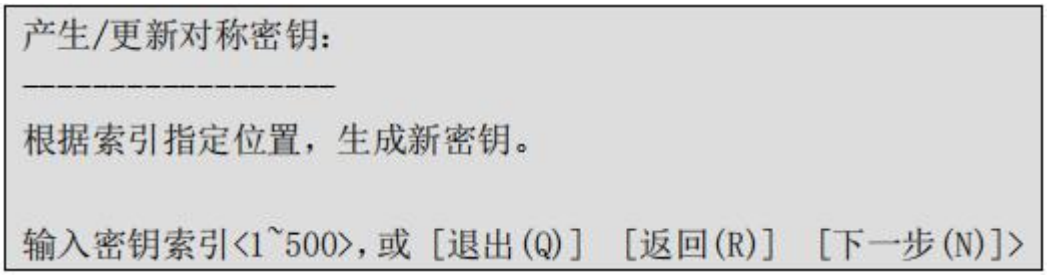
4.2.5. 产生对称密钥

前置条件：4.1.2 启动密码节点机密钥管理工具；4.1.3 使用管理员登录成功。

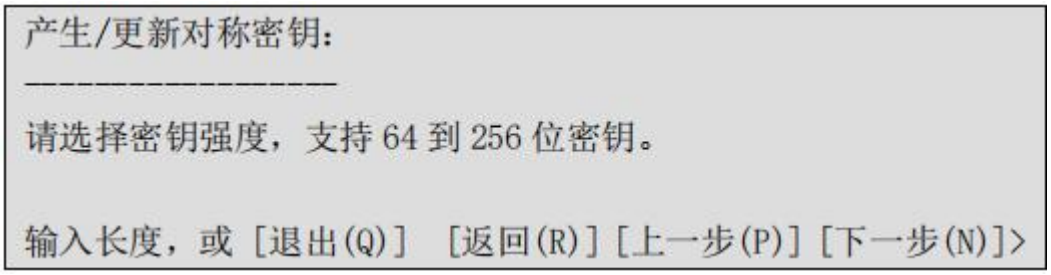
✓ 在密码节点机密钥管理主页选择“6 对称密钥管理”功能，进入对称密钥管理；



✓ 在 ECC 密钥管理界面选择“1 产生/更新密钥”，根据提示的密钥索引范围，指定密钥位置；



✓ 选择密钥强度，支持 64 到 256 位密钥长度，产生密钥成功。



4.2.6. 群组创建

前置条件：4.1.3 使用管理员登录成功；4.1.4 Consenter 服务已启动；4.1.5 Node 服务已启动。

✓ 进入鼎链主目录

✓ 执行命令

java -jar topchain-1.0.jar group create -g myGroup -c 127.0.0.1:7050

g：群组名称

c：Consenter 地址

✓ 得到文件 myGroup.block，群组创建完成

4.2.7. 加入群组

前置条件：4.1.3 使用管理员登录成功；4.1.4 Consenter 服务已启动；4.1.5 Node 服务已启动；4.2.6 新的群组已经创建成功。

✓ 进入鼎链主目录

✓ 执行命令

java -jar topchain-1.0.jar group join -b myGroup.block -t 127.0.0.1:7051

b：创世区块文件名称

t：Node 地址

✓ Node 得到如下日志，群组加入完成

```
[PvtDataStoreImpl][INFO] Saved 0 private data write sets for block [0]
[KvLedger][INFO] Group myGroup: Committed block 0 to storage
[KvLedger][INFO] Group myGroup: Saving current ledger height:1 at:1564714075670
```

<!>提示：如果当前节点只是 node 节点，要从有 Consenter 的 node 节点复制 myGroup.block 文件并且参数 c 要改成实际的 Consenter 地址。

4.2.8. 智能合约部署

前置条件：4.1.3 使用管理员登录成功；4.1.4 Consenter 服务已启动；4.1.5 Node 服务已启动；4.2.7 节点已加入群组成功。

- ✓ 进入鼎链主目录
- ✓ 执行命令

```
java -jar topchain-1.0.jar contract restDeploy -t 127.0.0.1:7051
-n mycc -v 1.0.0 -p Smartcontract-1.0-topchain.war -c 127.0.0.1:7050
-g myGroup -i "{'args':['init','a','init']}" -P
"OR('Org1MSP.member','Org2MSP.member')"
```

c：Consenter 地址

t：Node 地址

n：智能合约名称

v：智能合约版本

p：智能合约 war 包

g：群组名称

i：智能合约 init 方法所需参数

P：智能合约执行权限控制策略

- ✓ Node 得到如下日志，智能合约 mycc1.0.0 部署成功。

```
generate Key Success.
generate Key Success.
01-20 16:59:01,608 [NodeSmartContract][INFO][grpc-default-executor-1] Deploy success
01-20 16:59:01,609 [Node][INFO][main] Node Command end
01-20 16:59:01,609 [NodeSmartContract][INFO][grpc-default-executor-1] Broadcast completed
01-20 16:59:01,609 [BroadcastClient][INFO][grpc-default-executor-1] BroadcastClient close
```

4.2.9. 智能合约调用

前置条件：4.1.3 使用管理员登录成功；4.1.4 Consenter 服务已启动；4.1.5 Node 服务已启动；4.2.8 智能合约部署成功。

- ✓ 进入鼎链主目录
- ✓ 执行命令

java -jar topchain-1.0.jar contract invoke -t 127.0.0.1:7051 -c 127.0.0.1:7050 -g myGroup -n mycc -i "{'args':['save','test0107','valueFromInvoke0107']}"

c：Consenter 地址

t：Node 地址

g：群组名称

n：智能合约名称

i：智能合约 invoke 方法所需参数

- ✓ Node 得到类似如下日志，智能合约调用完成

```
[PvtDataStoreImpl][INFO] Saved 0 private data write sets for block [2]
[KvLedger][INFO] Group myGroup: Committed block 2 to storage
[KvLedger][INFO] Group myGroup: Saving current ledger height:3 at:1564716172598
```

4.2.10. 智能合约查询

前置条件：4.1.3 使用管理员登录成功；4.1.4 Consenter 服务已启动；4.1.5 Node 服务已启动；4.2.8 智能合约部署成功。

- ✓ 进入鼎链主目录
- ✓ 执行命令

java -jar topchain-1.0.jar contract query -t 127.0.0.1:7051 -g myGroup -n mycc -i "{'args':['query','test0107']}"

t：Node 地址

g：群组名称

n：智能合约名称

i：智能合约 invoke 方法所需参数

✓ CLI 得到类似如下日志，智能合约查询完成。

```
08-02 11:28:35,712 [ContractQueryCmd][INFO] TargetAddress: 127.0.0.1:7051
08-02 11:28:35,713 [ContractQueryCmd][INFO] GroupId: myGroup
08-02 11:28:35,713 [ContractQueryCmd][INFO] Smart contract id: mycc
08-02 11:28:35,713 [AbstractNodeContractCmd][INFO] InputStr: {'args':['invoke','b']}
08-02 11:28:35,812 [AbstractNodeContractCmd][INFO] Input.getArg: invoke
08-02 11:28:35,812 [AbstractNodeContractCmd][INFO] Input.getArg: b
08-02 11:28:36,939 [ContractQueryCmd][INFO] Query result: , <ByteString@22356acd size=0>
08-02 11:28:36,939 [Node][INFO] Node Command end
```

4.2.11. 智能合约升级

前置条件：4.1.3 使用管理员登录成功；4.1.4 Consenter 服务已启动；4.1.5 Node 服务已启动；4.2.7 节点已加入群组成功。

✓ 进入鼎链主目录

✓ 执行命令

java -jar topchain-1.0.jar contract restDeploy -t 127.0.0.1:7051 -n mycc -v 1.0.1 -p Smartcontract-1.0-topchain.war -c 127.0.0.1:7050 -g myGroup -i "{'args':['init','a','init']}" -P "OR('Org1MSP.member','Org2MSP.member')"

c：Consenter 地址

t：Node 地址

n：智能合约名称

v：智能合约版本

p：智能合约 war 包

g：群组名称

i：智能合约 init 方法所需参数

P：智能合约执行权限控制策略

✓ 得到如下日志，智能合约 mycc 成功升级到 1.0.1。

```
[PvtDataStoreImpl][INFO] Saved 0 private data write sets for block [1]
[KvLedger][INFO] Group myGroup: Committed block 1 to storage
[KvLedger][INFO] Group myGroup: Saving current ledger height:2 at:1564716003471
```

<!>提示：和 4.2.8 部署智能合约基本一致，只需升级 v：智能合约版本号参数即可。

5. 接口说明（客户端）

5.1. 申请授权

5.1.1. 准备工作

(1) 在调用接口之前，需要准备好客户端的密钥对。可采用以下方式：联系区块链节点管理员申请密钥对，管理员离线发放密钥对或者客户端自主生成密钥对，将公钥发送给区块链节点管理员。

(2) 区块链节点管理员将客户端的公钥经过签名处理后，保存至 node 节点的目录“config/restapi/certs/appID/pubKey”下。（其中，目录名称“appID”为具体的客户端 appID，不可重名，由字母、数字、@符号组成”。）

示例：

文件名：config/restapi/certs/Server1@org1/pubKey

内容：

04f2be298e4d9159f0b57a279b22210f5dca391a426666993103784fa2e69fbb2080d93cd5cd4f8f81fe11371a6c6dc8c410030ac08a155f98cb19ab75e50653ad

(3)重启 Node 节点，命令：java -jar topchain-1.0.jar node start

5.1.2. 申请 token

Token 是与 RESTful API 交互的通信密钥，通过通信密钥对消息进行加密，在 RESTful API 服务端对密文进行解密，比对解密消息和请求消息，从而判断请求是否由合法用户发送。

客户端通过 SM2 算法对一串 32 位的随机数进行签名，并将随机数放入 requestBody 中的 data 字段，在 requestBody 中携带 appID、签名、data，访问“http://ip 地址:7055/restfulapi/token/get_token”。

访问 URI	/restfulapi/token/get_token
访问方法	POST
参数	**说明**
appID	客户端应用名称
signature	签名
data	一段 32 位的随机数
返回值	**说明**
data	经过客户端的公钥加密的 token
success	请求是否成功，若成功，为 true；否则，为 false

注意：token 具有有效期，默认为 60 分钟，超过有效期需要重新发送申请 token 的请求。

示例：

请求：http://127.0.0.1:7055/restfulapi/token/get_token

Body：


```
{
    "appID":"Server1@org1",
    "signature":"304402207d4c88cc6e021a0f6897bdf624c31c43dcb183110075fb1feacecfc74210ef0b02200575081042b67c1e16f0d20beace0029ed7f109eaa6e419948081ea719dd2996",
    "data":"request token"
}
```


响应：


```
{
  "data": "04d4e1b88257293e84d19587e8cb6158570acc39ec85c2820f05ac35dadede50401b353a47596cc704c68e1f079c04175978844293480eafd9df2c7890724a0a9b1235676e3ef5121ae47f1077efb56347e40b77b91875eddb481eb76b5ed779da55fbd163d91acdca3863f4634aab1f39b36bc22f2df35ca51bb4675edba1525b",
  "message": "",
  "success": true,
  "code": 1,
  "currentPage": 0,
  "totalPage": 0,
  "totalCount": 0
}
```


5.1.3. 获取 token

通过步骤 2 得到的是使用公钥经过 SM2 算法加密后的 token，用户使用客户端的私钥通过 SM2 算法对这个被加密的 token 进行解密，获取 token。

上述示例取得的加密后的 token 为：

04d4e1b88257293e84d19587e8cb6158570acc39ec85c2820f05ac35dadede50

401b353a47596cc704c68e1f079c04175978844293480eafd9df2c7890724a0a9b12
35676e3ef5121ae47f1077efb56347e40b77b91875eddb481eb76b5ed779da55fbd1
63d91acdca3863f4634aab1f39b36bc22f2df35ca51bb4675edba1525b

使用私钥经过 SM2 解密后的 token 为：

19AfE3Ba0Ee7965755Cf931ddeFfEC35

5.1.4. 使用 token 发送请求

使用步骤 3 获得的 token，通过 SM4 算法对请求的参数中的其中一个参数的值进行加密，访问相应的 URI。

示例：

- ✓ GET 请求

  请求：http://127.0.0.1:7055/restfulapi/node/details

  Params: {/** 按 API 所需的实际情况填充，例如：key:value */}

  Header：{

  Content-Type:application/json,

  tokenSign:/** 填充使用 token 对 Params 中的参数值（例如：value）加密的结果 */,

  appID:Server1@org1

  }

其中，tokenSign 是通过 SM4 算法，使用 token 对 Params 中的参数进行加密的结果(如果 API 接口不要求提供 Params,则对一段 24 位的随机数进行加密，

并将随机数附带在参数 Params 中一同发送，以实现请求方认证的目的）。建议对重要的参数进行加密，以达到在区块链平台接收到的数据中能够校验重要的参数是真实的目的。

响应:

```
{
  "data": {/** 相应的消息 */},
  "message": "",
  "success": true,
  "code": 1,
  "currentPage": 0,
  "totalPage": 0,
  "totalCount": 0
}
```

✓ POST 请求

请求：http://127.0.0.1:7055/restfulapi/smart_contract/query

Header：{

tokenSign:/** 填充使用 token 对 Body 中的参数值加密的结果 */,

appID:Server1@org1

}

Body：{/** 根据具体的 API 填充不同的内容。 */}

Header 中的 tokenSign 为使用 token 通过 SM4 算法对 Body 中的一个参数值进行加密后的密文。建议对重要的参数进行加密，以达到在区块链平台接收到的数据中能够校验重要的参数是真实的目的。

举例如下：

requestBody 为：

```
{
"scArgs":"{'args':['save','key1','value1']}",
"scName":"mycc",
"groupID":"myGroup",
"consenterIP":"127.0.0.1",
"consenterPort":7050,
"tlsEnable":false
}
```

其中，scArgs 是一项重要的参数，{'args':['save','key1','value1']}为其参数值，对该参数值进行加密即可。此外，进行加密的数据不能包含\n\t 等转义符，否则 SM4 加密会抛出异常。

响应：

密文与 body 不匹配：token is incorrect.

密文与 body 匹配：

```
{
  "data": {},
  "message": "",
  "success": false,
  "code": 0,
  "currentPage": 0,
  "totalPage": 0,
  "totalCount": 0
}
```

5.2. 业务调用

5.2.1. 调用智能合约（数据上链）

访问 URI	/restfulapi/smart_contract/invoke
访问方法	POST
参数	**说明**
consenterIP	共识节点 IP 地址。
consenterPort	共识节点端口号。
scName	智能合约名称。
scArgs	智能合约入参。
groupID	群组名。
tlsEnable	默认使用 false
返回值	**说明**
message	消息。
success	升级是否成功。

示例：

请求：（请求类型：POST，头部信息见 5.1.4）

http://127.0.0.1:7055/restfulapi/smart_contract/invoke

requestBody:

```
{
    "scArgs":"{'args':['save','key','value']}",
    "scName":"testcc",
```

"groupID":"myGroup",
"consenterIP":"127.0.0.1",
"consenterPort":7050,
"tlsEnable":false
}

返回：

成功：

{
"data": null,
"message":"invoke success!",
"success": false,
"code": 0,
"currentPage": 0,
"totalPage": 0,
"totalCount": 0
}

失败：

{
"data": null,
"message":"Invoke fail:org.bcia.julongchain.commons.exception.SmartContractException: [SmartContract]Execute smart contract fail: \b_x0012__x0016_unknown function: save",
"success": false,
"code": 0,
"currentPage": 0,
"totalPage": 0,
"totalCount": 0
}

5.2.2. 智能合约查询（链上数据查询）

访问 URI	/restfulapi/smart_contract/query
访问方法	POST
参数	**说明**
scName	智能合约名称。
scArgs	智能合约入参。
groupID	群组名。
返回值	**说明**
data	查询结果。
message	消息。
success	升级是否成功。

示例：

请求：（请求类型：POST，头部信息见前文 4.1.4 小节）

http://127.0.0.1:7055/restfulapi/smart_contract/query

requestBody:

```
{
    "scArgs":"{'args':['query','key1']}",
    "scName":"mycc",
    "groupID":"myGroup"
}
```

返回：

成功：

{
"data": "[{\"name\":\"key1\",\"value\":80}]",
"message":"query success.",
"success": true,
"code": 0,
"currentPage": 0,
"totalPage": 0,
"totalCount": 0
}

失败：

{
"data": null,
"message":"org.bcia.julongchain.commons.exception.SmartContractException: [SmartContract]Execute smart contract fail: _x0012_unknown function: query",
"success": false,
"code": 0,
"currentPage": 0,
"totalPage": 0,
"totalCount": 0
}