Tutorial used: <https://hyperledger-fabric.readthedocs.io/en/latest/secured_asset_transfer/secured_private_asset_transfer_tutorial.html>

I had to use the Go language because the “shim” API is not available in Javascript and it is useful to e.g., get information about the peers that are executing chaincode. This could be the MSPID of the peer that I as a client interact with.

# Start Network

We have to start the network with the CA flag because we make use of the CA’s to initiate and create our users.

Start command: ./network.sh up createChannel -c mychannel -ca

# Endorsement Policy Upon Chaincode Install on Peers

When we install chaincode on peers, we can determine an endorsement policy as flags to the deploy method. In my case I am working in a test configuration with 2 peers, where I want either peer to endorse transactions because we want one organization to be able to create a contract without the other organization endorsing it. This is done through the -ccep flag, which in this case is "OR('Org1MSP.peer','Org2MSP.peer')". We could modify this in many ways depending on the use case.

Install: ./network.sh deployCC -ccn secured -ccp ../asset-transfer-secured-agreement/chaincode-go/ -ccl go -ccep "OR('Org1MSP.peer','Org2MSP.peer')"

# Create Contract as Admin@Org1

peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"CreateContract","Args":["contract1", "100", "200", "A contract on my energy"]}'

**// WITH PRIVATE DATA**

peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"CreateContract","Args":["contract1", "100", "200", "A contract on my energy"]}' --transient "{\"ownerName\":\"$CONTRACT\_NAME\"}"

Returns: Chaincode invoke successful. result: status:200

# Read Contract as Admin@Org1

peer chaincode query -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"ReadContract","Args":["contract1"]}'

Returns: {"contractID":"asset1","ownerOrg":"Org1MSP","price":"100","amount":"200"}

# Read Contract as Admin@Org2

Same command as above

Returns: {"contractID":"asset1","ownerOrg":"Org1MSP","price":"100","amount":"200"}

This is because I have not implemented any access control. This makes all contracts public to anyone connected to that specific channel (“mychannel” in this case).

Through the use of ctx.clientIdentity.getMSPID() and shim.getMSPID() I could have added a check to ensure that only clients from Org1 are allowed to read the contract.

# Read Contract as Admin@Org2 with Access Control Added

Same code as above

Returns: Error: endorsement failure during query. response: status:500 message:"Org2MSP is not allowed to read contract asset1 owned by Org1MSP"

My implementation checks information about a contract and if anyone else tries to read the contract from another organization, then they are not allowed.

We can also add checks based on attributes of a X.509 certificate, through the use of “ctx.clientIdentity.getAttributeValue(‘department’)” This example would return the value of the department attribute of the certificate of the client interacting with the endpoint. This could be useful if we want to modify access based on departments or something else.

# Lookup Identities Through Admin CA

**Step 1:** Enroll as the admin user

fabric-ca-client enroll -u https://admin:adminpw@localhost:7054 --caname ca-org1 --tls.certfiles "${PWD}/organizations/fabric-ca/org1/tls-cert.pem"

**Step 2:** Call the identity list command

fabric-ca-client identity list --tls.certfiles "${PWD}/organizations/fabric-ca/org1/tls-cert.pem"

Result:

Text

Description automatically generated

We can here see that included in the list is the certificate I created with the email and age attributes connected.

# Create Contract as Admin@Org1 with expanded Access Control Added (Age required to be >= 18)

In this scenario, we read attributes of a certificate and base our access control on those.

To do this, I had to generate a new certificate for the admin of Org1 to include attributes. I did this by editing the registerEnroll.sh file in “/home/jonas/fabric-samples/test-network/organizations/fabric-ca/registerEnroll.sh”, in the createOrg1() method. The certificate generation looks like this:

fabric-ca-client register --caname ca-org1 --id.name org1admin --id.secret org1adminpw --id.type admin --id.attrs 'age=20,email=jonas@gmail.com' --tls.certfiles "${PWD}/organizations/fabric-ca/org1/tls-cert.pem"

To be able to generate the certificates that I wanted, I had to start the test network with Certificate Authorities, otherwise it uses a different configuration. I started it like this: “./network.sh up createChannel -c mychannel -ca”

The user I tried to call the method with, has been assigned an age of 20 on his certificate and since the required age is 18 we succeed.

Result: [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200

# Create Contract as Admin@Org2 with Age < 18

We repeat the same steps as previously but this time on the admin of org2. I have issued a certificate for this user that has attributes where the age is 17. As a result of this, the user should be denied access to creating contracts when attempting to call the createContract endpoint.

Result: endorsement failure during invoke. response: status:500 message:"To create contracts you have to be atleast 18 years old" – The peer does not endorse this transaction and it is denied.

# Sign Contract as Admin@Org2 that has been created by Admin@Org1

This example makes us must interact with the interface that has to do with endorsement policies. If I create a contract, as Admin@Org1 then I must endorse when Admin@org2 tries to sign it. This is different than e.g., CreateContract where anyone can create a contract without endorsement from other peers.

Current issue with this: Unable to sync changes from signContract for some reason

When we want to read updated ledger data, we use query. When we want to update ledger data we use invoke.

peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"SignContract","Args":["contract1"]}'

**Invoke:** Invoke the specified chaincode. It will try to commit the endorsed transaction to the network.

**Query:** Get endorsed result of chaincode function call and print it. It won't generate transaction.

**Endorsement:** **Endorsement in Hyperledger Fabric basically allows users to define policies around the execution of chaincode. These endorsement policies define which peers need to agree on the results of a transaction before it can be added to the ledger.**

**Transient (Private Data): The transient field is where we have ‘private data’ in a transaction. We can retrieve this data on the peer by calling GetTransient(), and it is never included to the public in a transaction.**

* **Private data usually comes with a “salt” value, because if let us say the private data is a dollar amount, then it is straightforward through bruteforce to find a match with the private data hash on the chain, which would cause the private data to be exposed. The “salt” is concatenated with the private data key to make this impossible.**

# **Add Private Data to Transaction**

Add –transient flag to the transaction call

We export the transient data as a variable for ease of use. This must be converted to a base64 string to work.

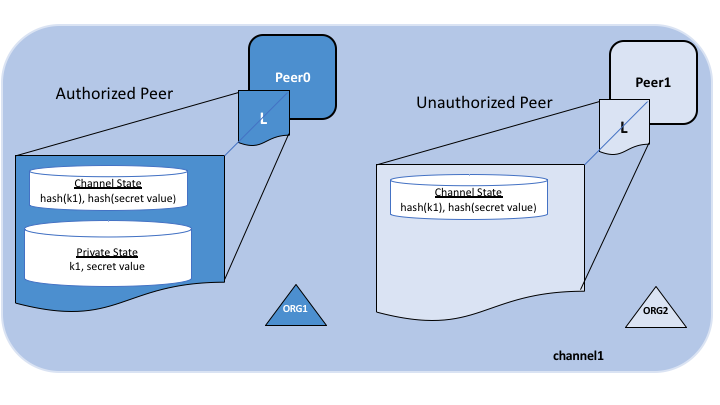
export CONTRACT\_NAME=$(echo -n "{\"Jonas\"}" | base64 | tr -d \\n)

**peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"CreateContract","Args":["contract1", "100", "200", "A contract on my energy"]}' --transient "{\"ownerName\":\"$CONTRACT\_NAME\"}"**

Text

Description automatically generated

Here we can see that the private data I added in the transaction is not visible on the ledger. We can only retrieve this if we have access.



TODO: Add a method that allows us to retrieve the private data again once it has been added. We want this so we can prove it is stored on the ledger.

# Retrieve Private Data as Admin@org1

We created a transaction that has the name of the owner of the contract put in as private data. This command below retrieves the name.

peer chaincode query -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"GetContractPrivateData","Args":["contract1"]}'

Result: Jonas

If we try to do it as Admin@org2, then we get an error "asset private details does not exist in client org's collection: contract1" because the private data is only stored in the ledger of Peer0 and not Peer1.

# Delete Contract as Admin@Org1

Since data is immutable is it not possible to delete data. When we call DelState(), then we flag on that transaction called “is\_delete” will change from false to true, indicating to the blockchain that it should no longer be part of the state database. It will however always remain on the ledger.

Command: peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n secured -c '{"function":"DeleteContract","Args":["contract1"]}'

If we try to retrieve the contract with the ReadContract, then it will say that the contract no longer exists.

# Connect Frontend Application To Blockchain Through Node SDK Framework

It is important than for every time the blockchain network is restarted, the “wallet” folder on the application is deleted. Furthermore, the “organizations” folder from the test-network folder (The blockchain network), has to be copied to the application “test-network” folder. If this is not done, then it is not possible to connect to the blockchain. Additionally, we must run the frontend inside the WSL due to connectivity issues if it is outside.

For each restart/change of blockchain:

1. Delete wallets folder from **/home/jonas/HyperLedger Frontend**
2. Copy from ‘organizations’ folder from **/home/jonas/fabric-samples/test-network/** to **/home/jonas/HyperLedger Frontend/test-network/**

**Vigtigt:** Due to difficulties in generating certificates with attributes (e.g., age and email) from the node SDK API we must disable attribute access control in the chaincode whenever we want to test it from the frontend. Otherwise the chaincode will fail because it can’t find the age and email attributes on the certificate.

To start up we need to first initiate the certificates and we do that by calling the endpoint “initGateway”. If we do not do this then we can’t create or read contracts.

# Fabric Explorer

Det er vigtigt!!!! At man for hver gang man prøver at starte Fabric Explorer specielt med nye certifikater at man husker manuelt via. Docker at fjerne de volumes der bliver oprettet, ellers beholder den de gamle certifikater og kan ikke starte. Dette gøres via. Kommandoen: **docker volume rm fabric-explorer\_pgdata fabric-explorer\_walletstore** in addition we also need to remember to rename the “ca-cert.pem” in the signcerts folder and the file in the keystore folder to “priv\_sk”

We copy the “organizations” folder from the test network to the fabric-explorer folder.

**Rename file** in “/home/jonas/fabric-explorer/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp/keystore” and **file in** /signcerts to Admin@org1.example.com-cert.pem

# For the Evaluation

To evaluate the product and the features a tool called Hyperledger Explorer has been applied. It provides a visual overview of the running blockchain, including elements such as transactions, blocks, chaincode and more.