Digital Asset Custody Service

: Project Progress Report

Haechan Lee
Nguyen, Van Tu
Minji Choi
Jeongheon Kim

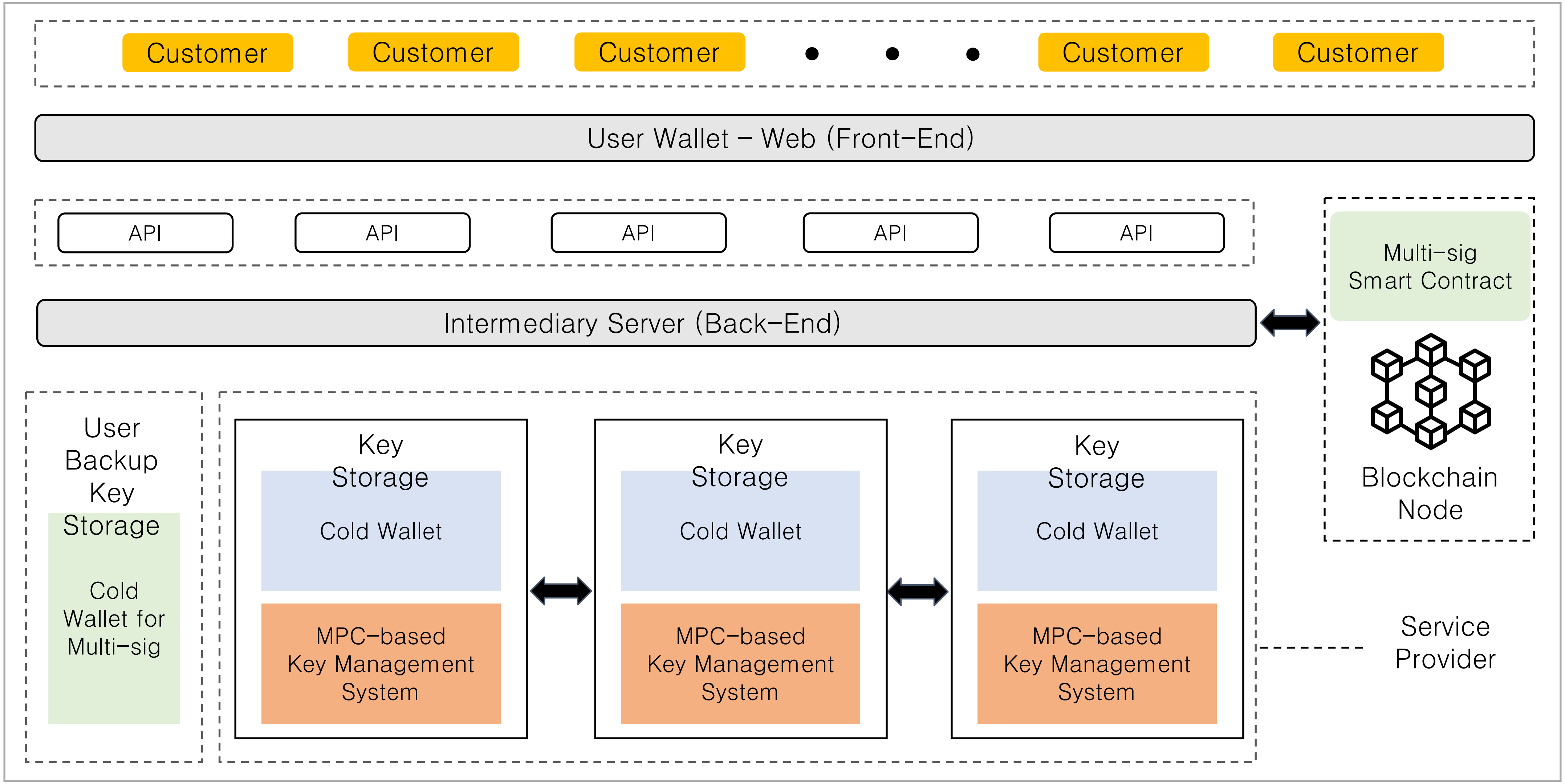
Project Overview

- Digital Asset Custody Service (DACS)
  - Keep users private digital asset (Cryptocurrency, NFT) safely
  - Digital asset's ownership can be proved by only the owner's private key

- Two kinds of DACS
  - Digital asset consignment management service
  - Private key management service

- In this project
  - Implement digital asset consignment management service using Multi-party computation (MPC)
  - Implement private key management service using Multi-sig

Architecture



Customer

Customer

Customer

● ● ●

Customer

Customer

User Wallet – Web (Front–End)

API

API

API

API

API

Intermediary Server (Back–End)

Multi–sig Smart Contract

Blockchain Node

User Backup Key Storage

Cold Wallet for Multi–sig

Key Storage

Cold Wallet

MPC–based Key Management System

Key Storage

Cold Wallet

MPC–based Key Management System

Key Storage

Cold Wallet

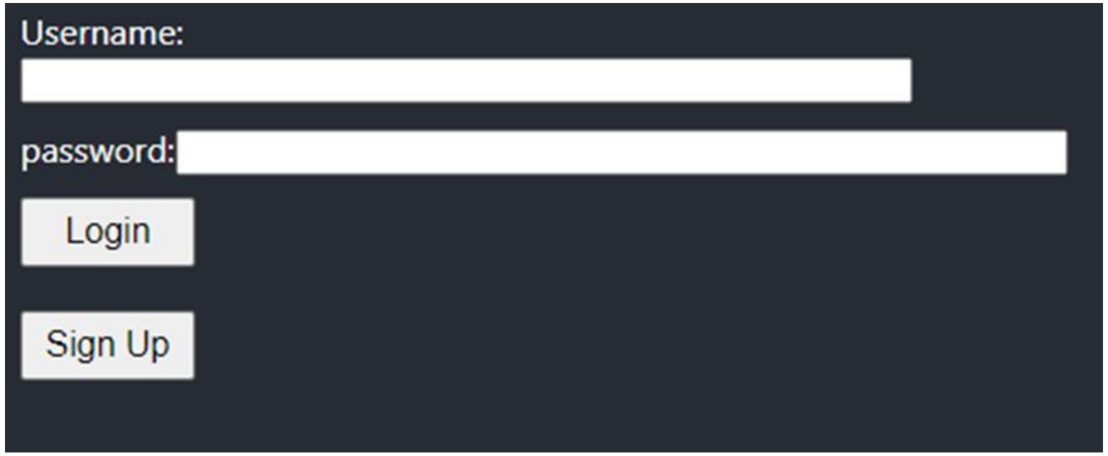
MPC–based Key Management System

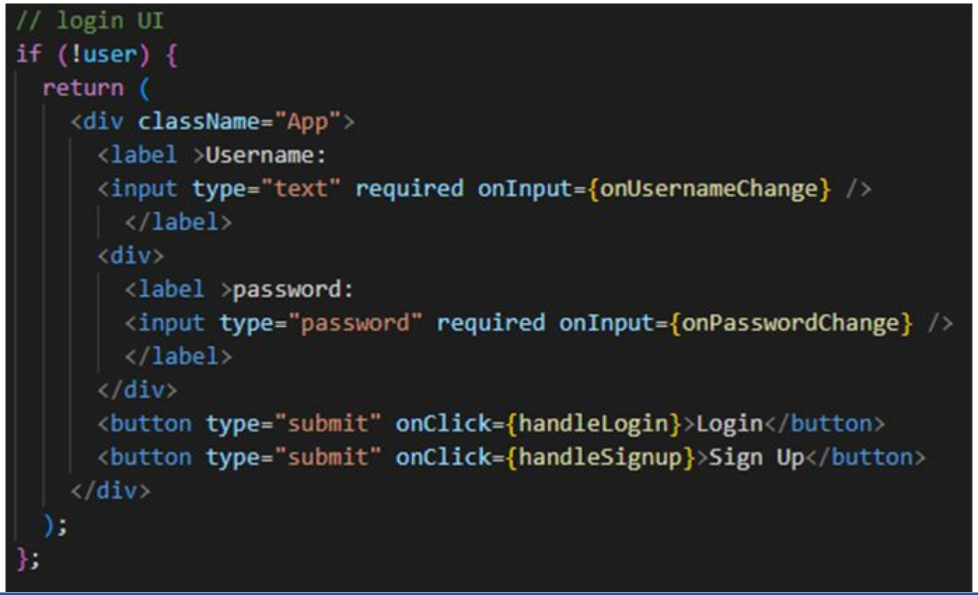
Service Provider

User GUI (frontend, 1/2)

- Front-end wallet for user
  - Implemented with Node.js and React.

- Features
  - Sign Up : Enroll Username and Password for using the wallet.
  - Login : Connect to the server using the previously registered account and password.



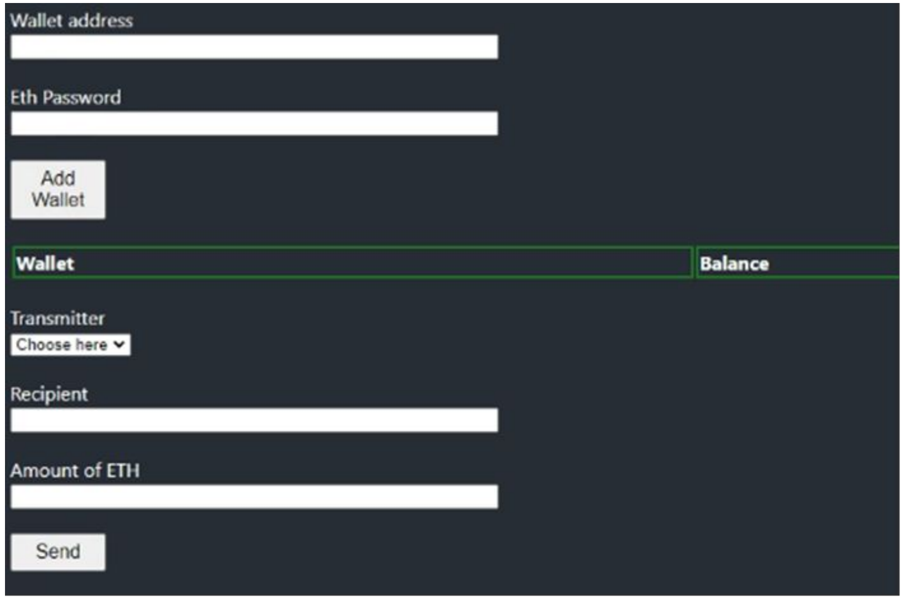


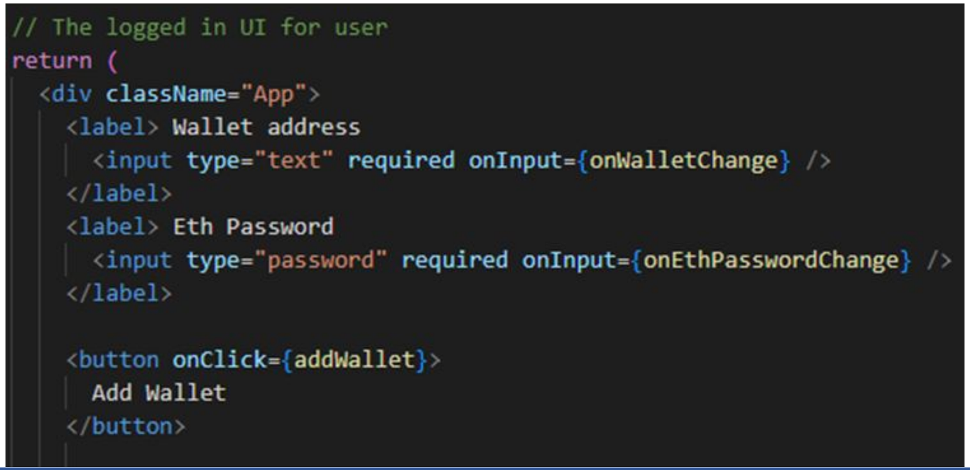
```
// login UI
if (!user) {
  return (
    <div className="App">
      <label >Username:
      <input type="text" required onInput={onUsernameChange} />
        </label>
      <div>
        <label >password:
        <input type="password" required onInput={onPasswordChange} />
        </label>
      </div>
      <button type="submit" onClick={handleLogin}>Login</button>
      <button type="submit" onClick={handleSignup}>Sign Up</button>
    </div>
  );
};
```

User GUI (frontend, 2/2)

- Front-end wallet for user
  - Implemented with Node.js and React.

- Features
  - Add Wallet: allow user to add their User’s Ethereum wallet address and password.
  - Send ETH: Send ETH to other Ethereum Account (not implemented yet)



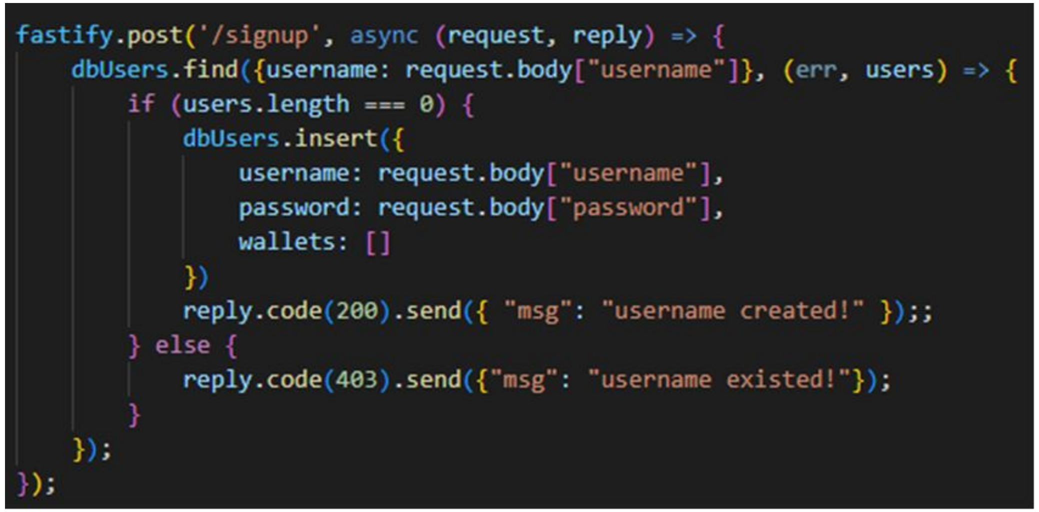


```
// The logged in UI for user
return (
  <div className="App">
    <label> Wallet address
      <input type="text" required onInput={onWalletChange} />
    </label>
    <label> Eth Password
      <input type="password" required onInput={onEthPasswordChange} />
    </label>

    <button onClick={addWallet}>
      Add Wallet
    </button>
```

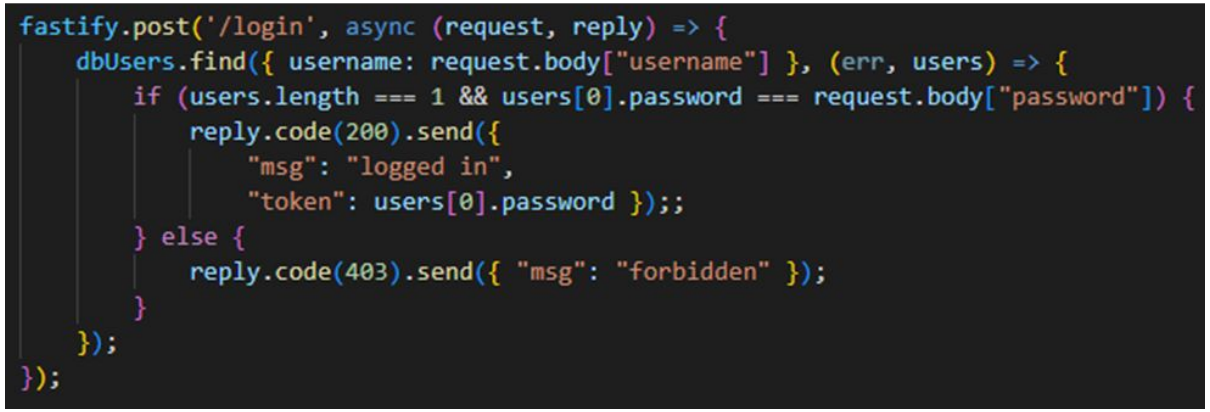
Server side (backend, 1/3)

- Manage user information and transactions
  - Provides REST APIs
  - Implemented using Fastify (node.js) and web3

- APIs
  - **Signup**: Get user's input of username and password then record it to database file
  - **Login**: Check whether username and password matches correctly, then reply front-end



```
fastify.post('/signup', async (request, reply) => {
    dbUsers.find({username: request.body["username"]}, (err, users) => {
        if (users.length === 0) {
            dbUsers.insert({
                username: request.body["username"],
                password: request.body["password"],
                wallets: []
            })
            reply.code(200).send({ "msg": "username created!" });;
        } else {
            reply.code(403).send({"msg": "username existed!"});
        }
    });
});
```

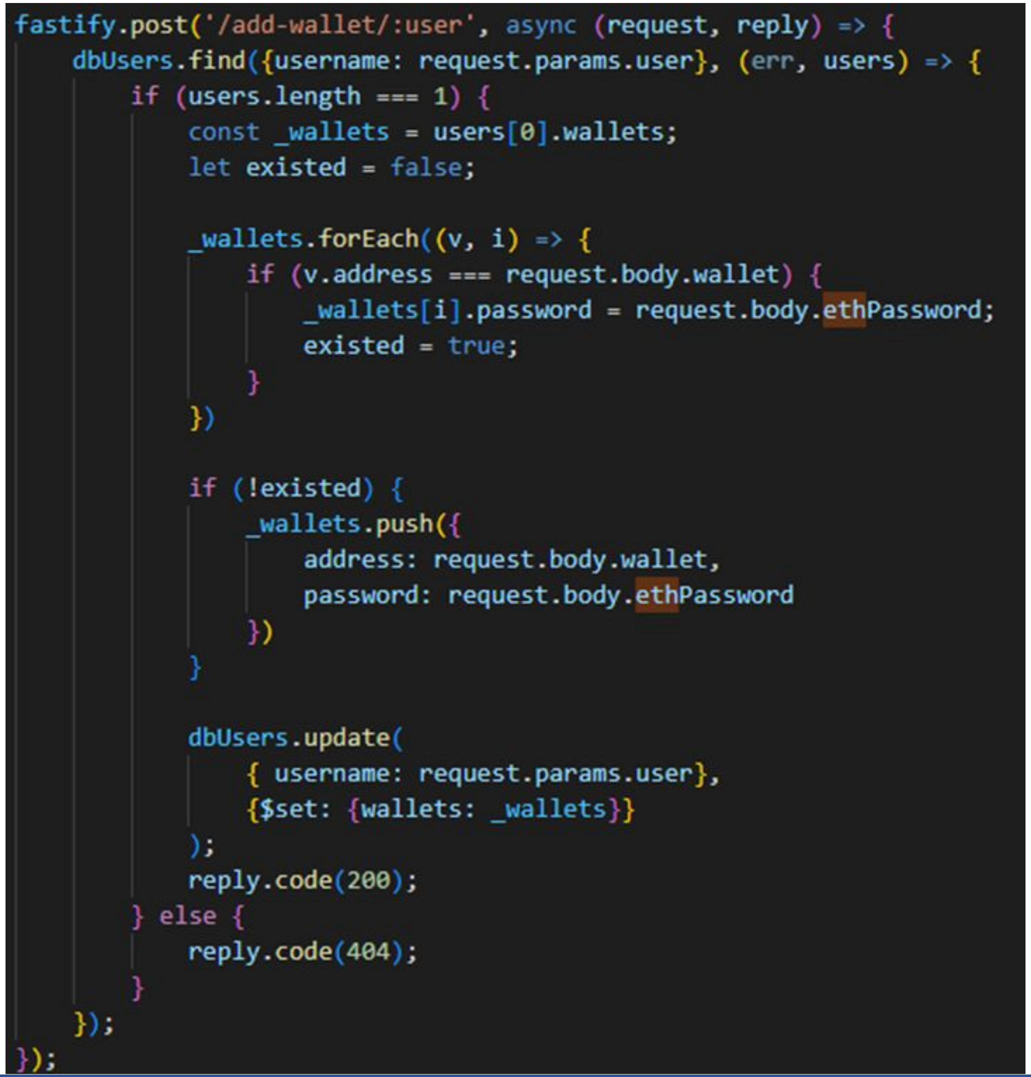


```
fastify.post('/login', async (request, reply) => {
    dbUsers.find({ username: request.body["username"] }, (err, users) => {
        if (users.length === 1 && users[0].password === request.body["password"]) {
            reply.code(200).send({
                "msg": "logged in",
                "token": users[0].password });;
        } else {
            reply.code(403).send({ "msg": "forbidden" });
        }
    });
});
```

Server side (backend, 2/3)

- Handle transactions and managements
  - Provides REST APIs
  - Implemented using Fastify (node.js) and web3

- APIs
  - **Add wallet**: allow user to add new wallet with password

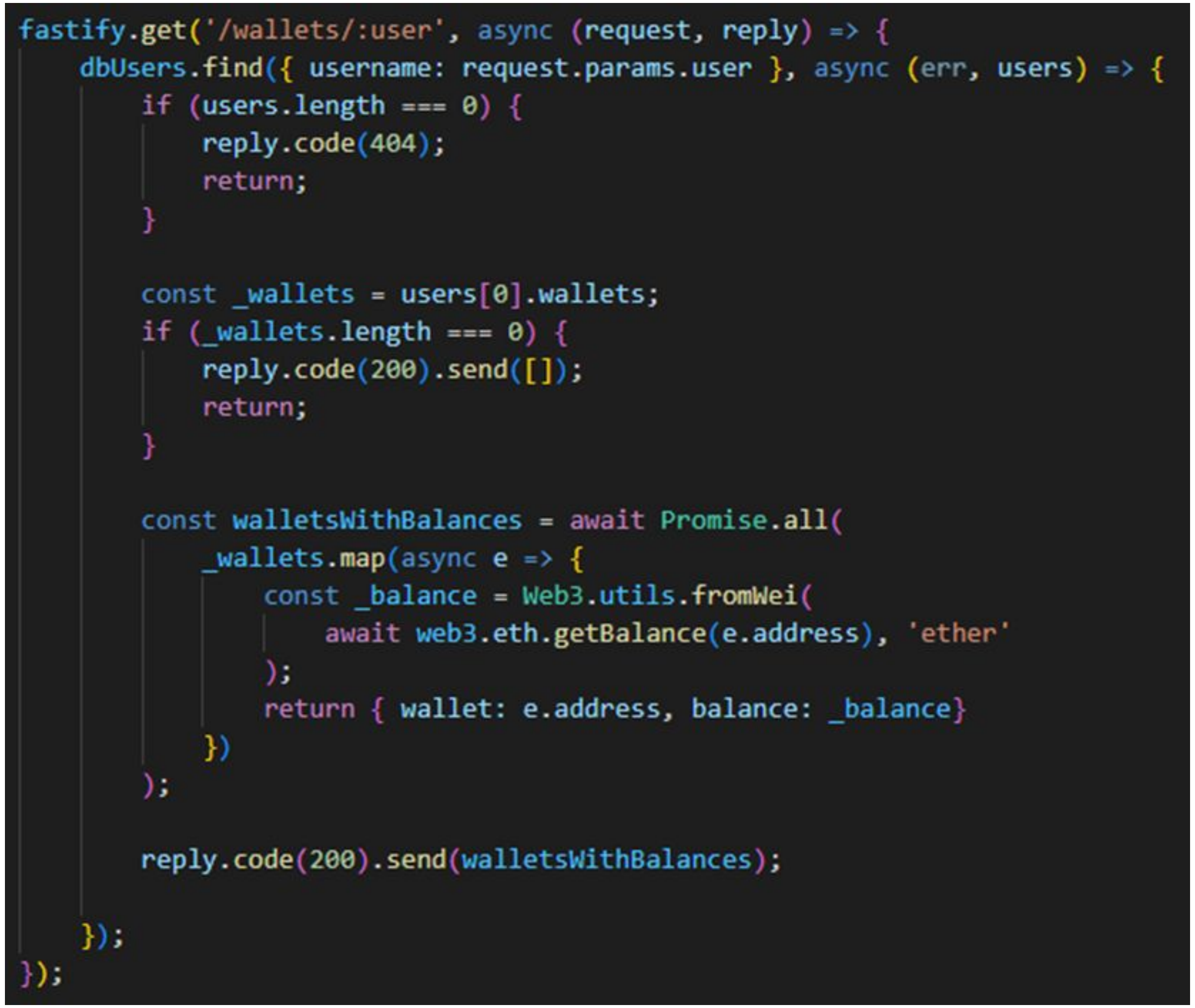


```
fastify.post('/add-wallet/:user', async (request, reply) => {
    dbUsers.find({username: request.params.user}, (err, users) => {
        if (users.length === 1) {
            const _wallets = users[0].wallets;
            let existed = false;

            _wallets.forEach((v, i) => {
                if (v.address === request.body.wallet) {
                    _wallets[i].password = request.body.ethPassword;
                    existed = true;
                }
            })

            if (!existed) {
                _wallets.push({
                    address: request.body.wallet,
                    password: request.body.ethPassword
                })
            }

            dbUsers.update(
                { username: request.params.user},
                {$set: {wallets: _wallets}}
            );
            reply.code(200);
        } else {
            reply.code(404);
        }
    });
});
```

Server side (backend, 3/3)

- Handle transactions and managements
  - Provides REST APIs
  - Implemented using Fastify (node.js) and web3

- APIs
  - **Get wallet**: Get wallets of an user and return the wallet address with balances
  - **Send**: (not implemented yet, need to integrate with MPC & multisig)



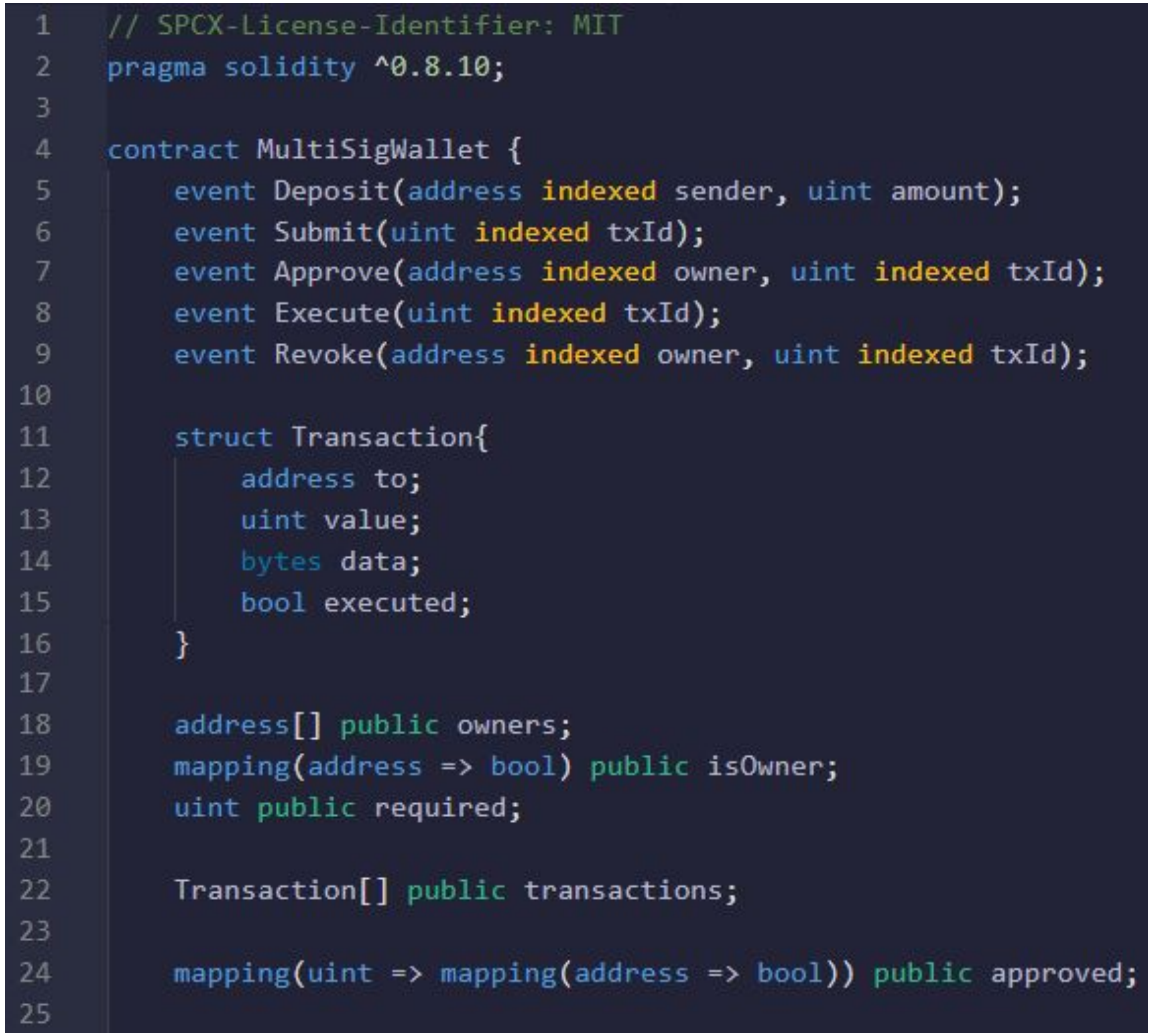
```
fastify.get('/wallets/:user', async (request, reply) => {
    dbUsers.find({ username: request.params.user }, async (err, users) => {
        if (users.length === 0) {
            reply.code(404);
            return;
        }

        const _wallets = users[0].wallets;
        if (_wallets.length === 0) {
            reply.code(200).send([]);
            return;
        }

        const walletsWithBalances = await Promise.all(
            _wallets.map(async e => {
                const _balance = Web3.utils.fromWei(
                    await web3.eth.getBalance(e.address), 'ether'
                );
                return { wallet: e.address, balance: _balance}
            })
        );

        reply.code(200).send(walletsWithBalances);

    });
});
```

Multi-Signature Smart Contract (1/5)

- Event
  - Deposit : Deposit ETH in Multisig wallet
  - Submit : Submit transaction, waiting for approval
  - Approve : Owners approve transactions
  - Execute : Implement transactions
    when certain amount of approvals exists
  - Revoke : Owners can revoke approvals
    before implementing transaction

- Transaction Structure
  - Address, value, data, execution

- Define public address & values
  - Address of Owners
  - Value of required
  - Array of Transactions
  - Boolean of Approved

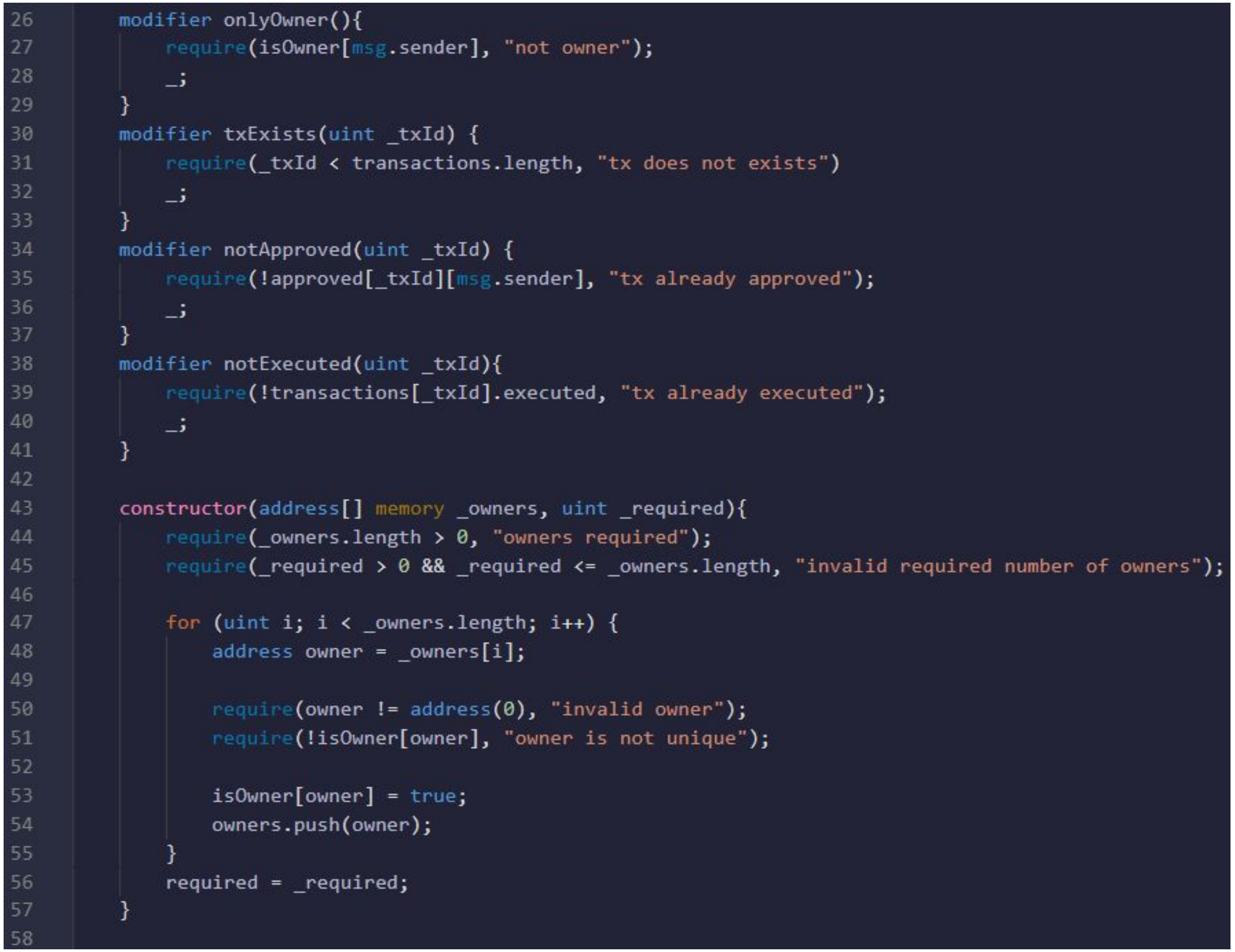


```
// SPCX-License-Identifier: MIT
pragma solidity ^0.8.10;

contract MultiSigWallet {
    event Deposit(address indexed sender, uint amount);
    event Submit(uint indexed txId);
    event Approve(address indexed owner, uint indexed txId);
    event Execute(uint indexed txId);
    event Revoke(address indexed owner, uint indexed txId);

    struct Transaction{
        address to;
        uint value;
        bytes data;
        bool executed;
    }

    address[] public owners;
    mapping(address => bool) public isOwner;
    uint public required;

    Transaction[] public transactions;

    mapping(uint => mapping(address => bool)) public approved;

```

Multi-Signature Smart Contract (2/5)

- Modifier for functions
    - onlyOwner : Check if the address is owner
    - txExists : Check if the transaction exists
    - notApproved : Check transaction if it is approved or not
    - notExecuted : Check transaction if it is executed or not.

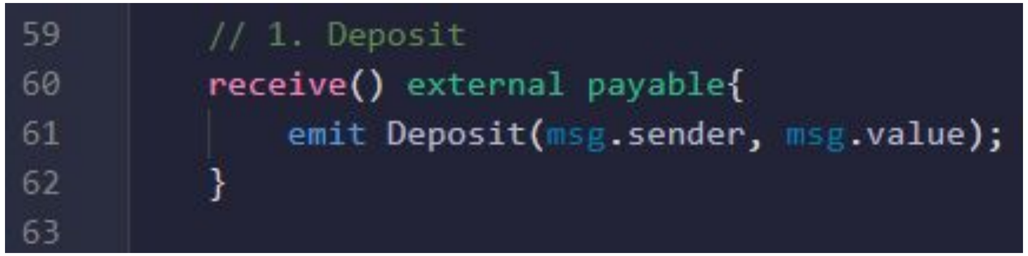
- Constructor
    - # of owner has to be more than one.
    - 'required' value should be more than zero, less than # of owner.



```
    modifier onlyOwner(){
        require(isOwner[msg.sender], "not owner");
        _;
    }
    modifier txExists(uint _txId) {
        require(_txId < transactions.length, "tx does not exists")
        _;
    }
    modifier notApproved(uint _txId) {
        require(!approved[_txId][msg.sender], "tx already approved");
        _;
    }
    modifier notExecuted(uint _txId){
        require(!transactions[_txId].executed, "tx already executed");
        _;
    }

    constructor(address[] memory _owners, uint _required){
        require(_owners.length > 0, "owners required");
        require(_required > 0 && _required <= _owners.length, "invalid required number of owners");

        for (uint i; i < _owners.length; i++) {
            address owner = _owners[i];

            require(owner != address(0), "invalid owner");
            require(!isOwner[owner], "owner is not unique");

            isOwner[owner] = true;
            owners.push(owner);
        }
        required = _required;
    }
```

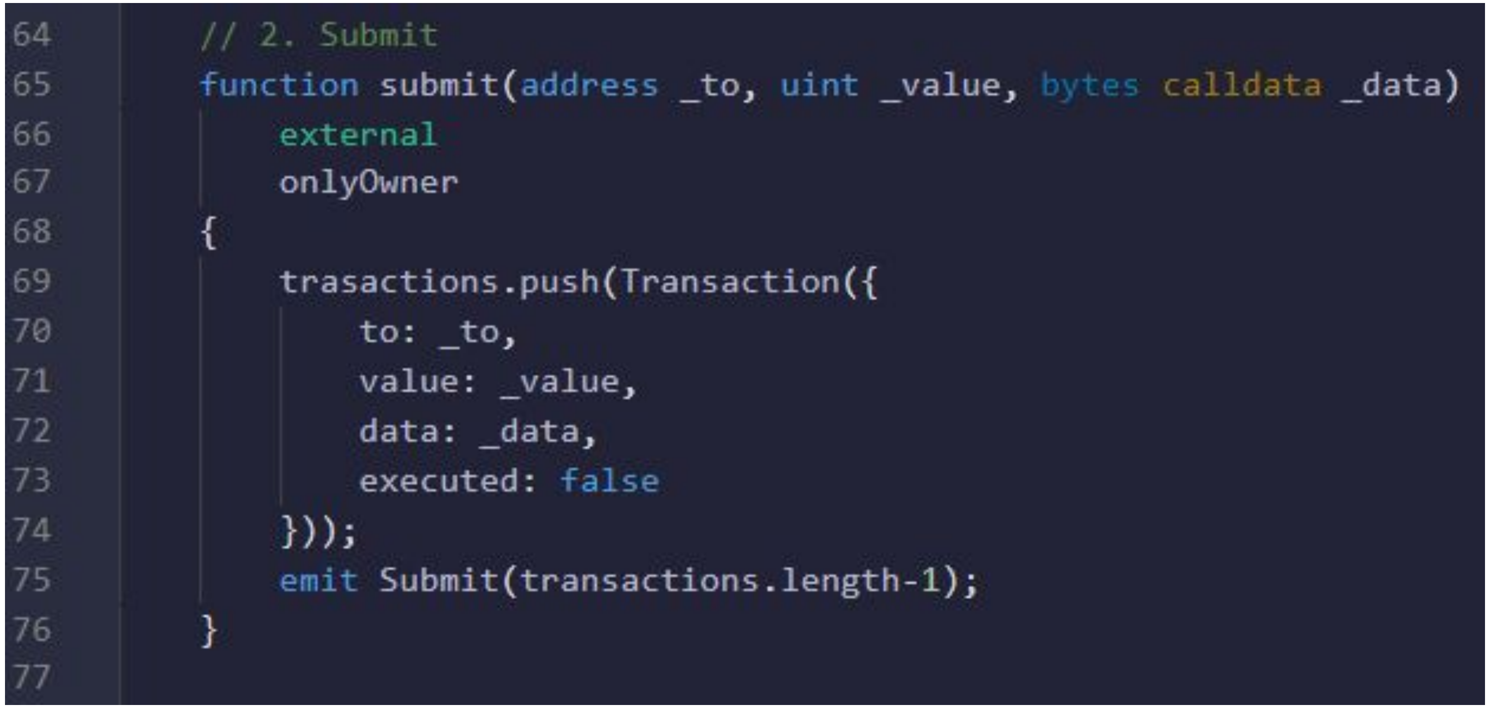
Multi-Signature Smart Contract (3/5)

- Event1. Deposit
  - Make Multisig wallet able to receive ETH.



```
    // 1. Deposit
    receive() external payable{
        emit Deposit(msg.sender, msg.value);
    }

```

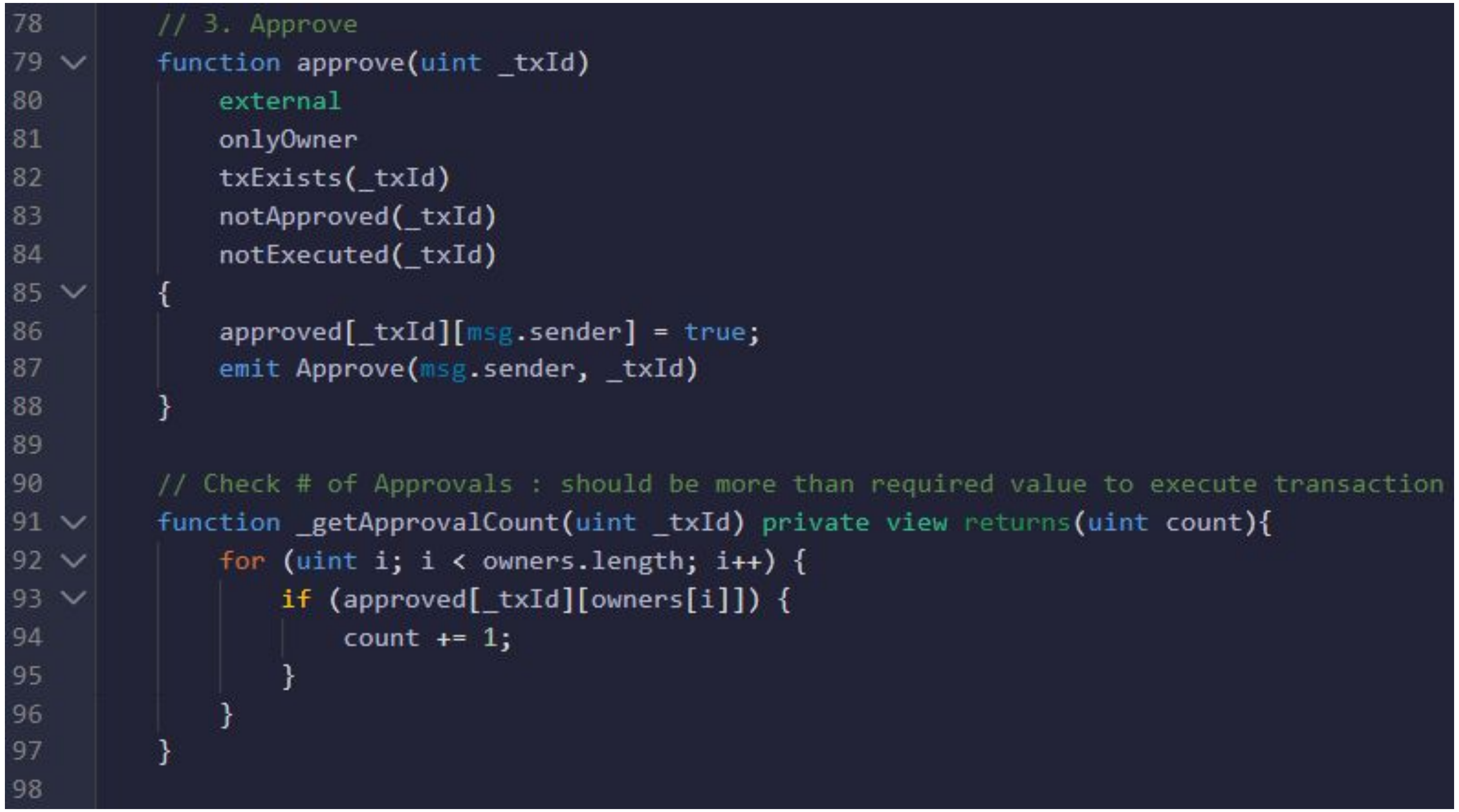
- Event2. Submit
  - Only owner can submit transaction
  - When the transaction is submitted, and the transaction has received sufficient amount of approvals, the owner can execute transaction.



```
    // 2. Submit
    function submit(address _to, uint _value, bytes calldata _data)
        external
        onlyOwner
    {
        trasactions.push(Transaction({
            to: _to,
            value: _value,
            data: _data,
            executed: false
        }));
        emit Submit(transactions.length-1);
    }

```

Multi-Signature Smart Contract (4/5)

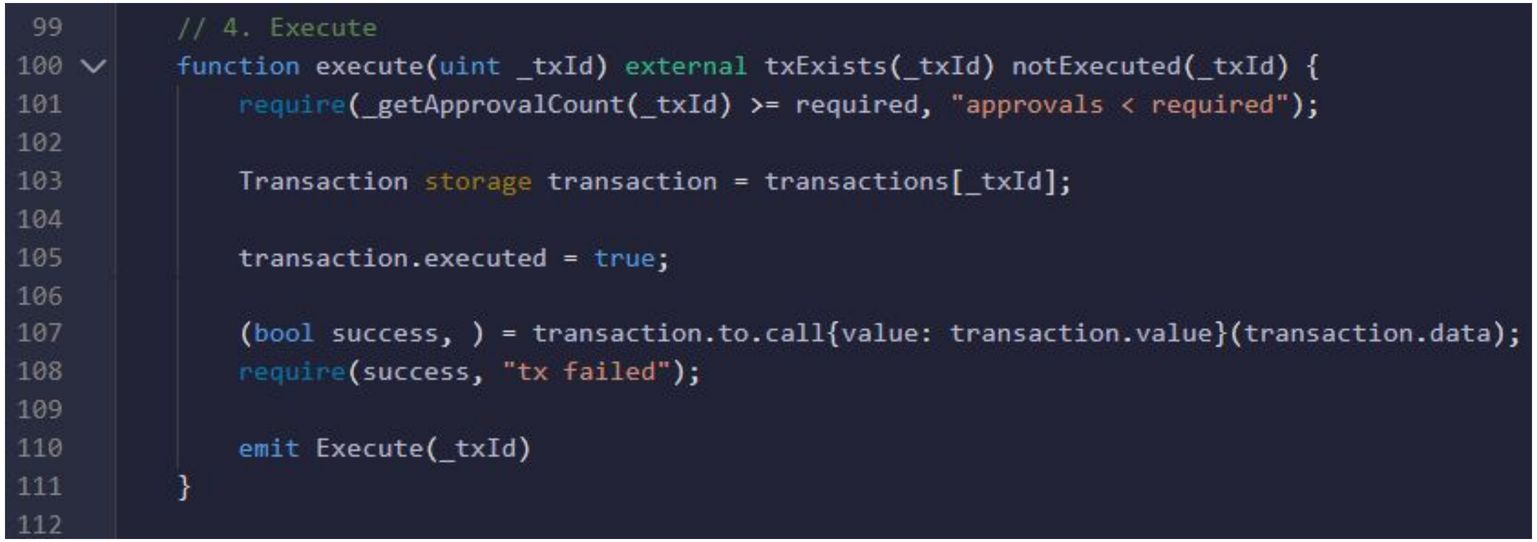
- Event3. Approve
  - After the transaction is submitted,
  - other owners can approve the transaction



```
    // 3. Approve
    function approve(uint _txId)
        external
        onlyOwner
        txExists(_txId)
        notApproved(_txId)
        notExecuted(_txId)
    {
        approved[_txId][msg.sender] = true;
        emit Approve(msg.sender, _txId)
    }

    // Check # of Approvals : should be more than required value to execute transaction
    function _getApprovalCount(uint _txId) private view returns(uint count){
        for (uint i; i < owners.length; i++) {
            if (approved[_txId][owners[i]]) {
                count += 1;
            }
        }
    }

```

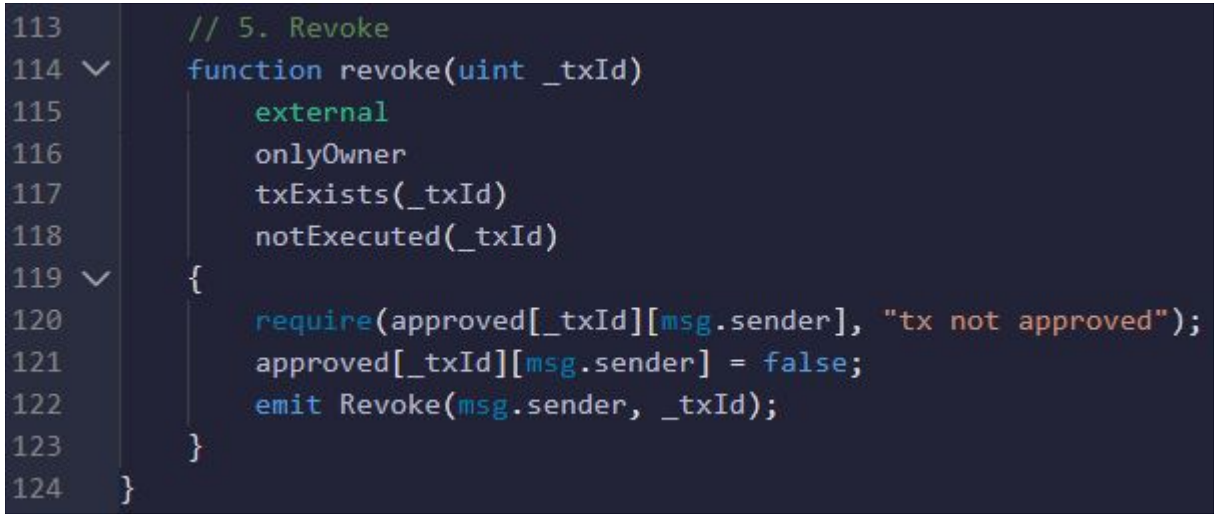
Multi-Signature Smart Contract (5/5)

- Event4. Execute
  - When the number of approval is more than required value, the transaction can be executed



```
    // 4. Execute
    function execute(uint _txId) external txExists(_txId) notExecuted(_txId) {
        require(_getApprovalCount(_txId) >= required, "approvals < required");

        Transaction storage transaction = transactions[_txId];

        transaction.executed = true;

        (bool success, ) = transaction.to.call{value: transaction.value}(transaction.data);
        require(success, "tx failed");

        emit Execute(_txId)
    }

```

- Event5. Revoke
  - Even if the owner had approved the transaction, the owner can revoke the transaction before the transaction is executed



```
    // 5. Revoke
    function revoke(uint _txId)
        external
        onlyOwner
        txExists(_txId)
        notExecuted(_txId)
    {
        require(approved[_txId][msg.sender], "tx not approved");
        approved[_txId][msg.sender] = false;
        emit Revoke(msg.sender, _txId);
    }
}
```

MPC (1/4)

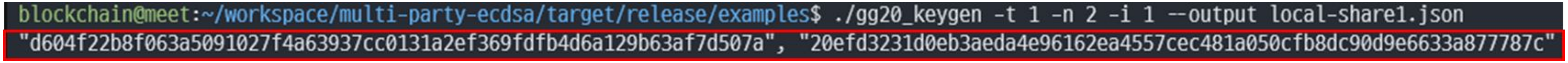
- Select MPC protocol
  - GG20[1] – Full Threshold (t,n) ECDSA protocol
  - https://github.com/ZenGo-X/multi-party-ecdsa/tree/master/src/protocols/multi_party_ecdsa/gg_2020

- MPC can be divided to two process
  - Key generation protocol
    - Generate the share of private key, public key
    - The public key should be translated to Ethereum address
  - Signing protocol
    - Generate the ECDSA signature (r,s) by signing the hash of unsigned transaction
    - Then by sending ECDSA signature, the unsigned transaction could be signed and send

[1] GENNARO, Rosario; GOLDFEDER, Steven. One round threshold ECDSA with identifiable abort. *Cryptology ePrint Archive*, 2020.

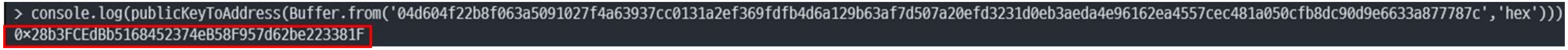
MPC (2/4)

- Key generation
    - Save share of private key in local-share.json
    - Output the ECDSA public key K = (x,y)



```
blockchain@meet:~/workspace/multi-party-ecdsa/target/release/examples$ ./gg20_keygen -t 1 -n 2 -i 1 --output local-share1.json
"d604f22b8f063a5091027f4a63937cc0131a2ef369fdfb4d6a129b63af7d507a", "20efd3231d0eb3aeda4e96162ea4557cec481a050cfb8dc90d9e6633a877787c"
```

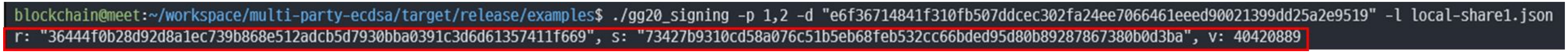
- Ethereum public key translation
    - “04 | x | y” by Standard for Efficient Cryptography (SEC1)
    - ex) “04d604f22b8f063a5091027f4a63937cc0131a2ef369fdfb4d6a129b63af7d507a20efd3231d0eb3aeda4e96162ea4557cec481a050cfb8dc90d9e6633a877787c”
- Ethereum address translation (Public Key -> Address)
    - “(keccak-256(x | y))[-20:]”



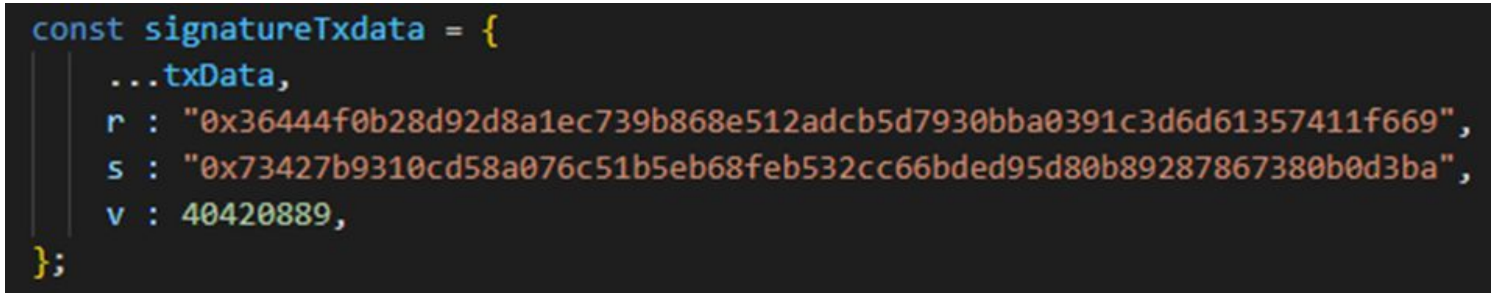
```
> console.log(publicKeyToAddress(Buffer.from('04d604f22b8f063a5091027f4a63937cc0131a2ef369fdfb4d6a129b63af7d507a20efd3231d0eb3aeda4e96162ea4557cec481a050cfb8dc90d9e6633a877787c','hex')))
0×28b3FCEdBb5168452374eB58F957d62be223381F
```

MPC (3/4)

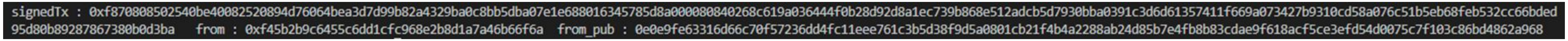
- Signing
  - Generate the ECDSA signature (r,s) and recid value by signing hashed unsigend transaction
  - Using recid value and Etheruem network id, it can calculate v
  - Send the r, s, v to unsigned transaction



```
blockchain@meet:~/workspace/multi-party-ecdsa/target/release/examples$ ./gg20_signing -p 1,2 -d "e6f36714841f310fb507ddcec302fa24ee7066461eeed90021399dd25a2e9519" -l local-share1.json
r: "36444f0b28d92d8a1ec739b868e512adcb5d7930bba0391c3d6d61357411f669", s: "73427b9310cd58a076c51b5eb68feb532cc66bded95d80b89287867380b0d3ba", v: 40420889
```



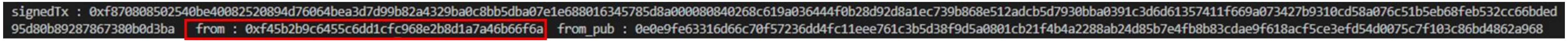
```
const signatureTxdata = {
    ...txData,
    r : "0x36444f0b28d92d8a1ec739b868e512adcb5d7930bba0391c3d6d61357411f669",
    s : "0x73427b9310cd58a076c51b5eb68feb532cc66bded95d80b89287867380b0d3ba",
    v : 40420889,
};
```



```
signedTx : 0xf870808502540be40082520894d76064bea3d7d99b82a4329ba0c8bb5dba07e1e688016345785d8a000080840268c619a036444f0b28d92d8a1ec739b868e512adcb5d7930bba0391c3d6d61357411f669a073427b9310cd58a076c51b5eb68feb532cc66bded
95d80b89287867380b0d3ba   from : 0xf45b2b9c6455c6dd1cfc968e2b8d1a7a46b66f6a   from_pub : 0e0e9fe63316d66c70f57236dd4fc11eee761c3b5d38f9d5a0801cb21f4b4a2288ab24d85b7e4fb8b83cdae9f618acf5ce3efd54d0075c7f103c86bd4862a968
```

MPC – troubleshooting (4/4)

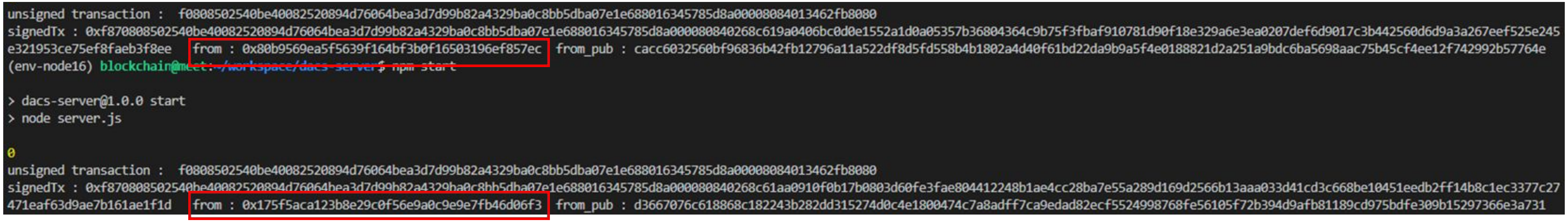
- Trouble 1
    - By extracting sender address in signed transaction, it's different with our address



```
signedTx : 0xf870808502540be40082520894d76064bea3d7d99b82a4329ba0c8bb5dba07e1e688016345785d8a000080840268c619a036444f0b28d92d8a1ec739b868e512adcb5d7930bba0391c3d6d61357411f669a073427b9310cd58a076c51b5eb68feb532cc66bded
95d80b89287867380b0d3ba   from : 0xf45b2b9c6455c6dd1cfc968e2b8d1a7a46b66f6a  from_pub : 0e0e9fe63316d66c70f57236dd4fc11eee761c3b5d38f9d5a0801cb21f4b4a2288ab24d85b7e4fb8b83cdae9f618acf5ce3efd54d0075c7f103c86bd4862a968
```

    - Our ethereum address : "0x28b3FCEdBb5168452374eB58F957d62be223381F"
    - Extracted sender address : "0xf45b2b9c6455c6dd1cfc968e2b8d1a7a46b66f6a"

- Trouble 2
    - If I create another signature of same transaction (r, s is different), then the server shows different sender address every time.



```
unsigned transaction :  f0808502540be40082520894d76064bea3d7d99b82a4329ba0c8bb5dba07e1e688016345785d8a000080840134 62fb8080
signedTx : 0xf870808502540be40082520894d76064bea3d7d99b82a4329ba0c8bb5dba07e1e688016345785d8a000080840268c619a0406bc0d0e1552a1d0a05357b36804364c9b75f3fbaf910781d90f18e329a6e3ea0207def6d9017c3b442560d6d9a3a267eef525e245
e321953ce75ef8faeb3f8ee   from : 0x80b9569ea5f5639f164bf3b0f16503196ef857ec  from_pub : cacc6032560bf96836b42fb12796a11a522df8d5fd558b4b1802a4d40f61bd22da9b9a5f4e0188821d2a251a9bdc6ba5698aac75b45cf4ee12f742992b57764e
(env-node16) blockchain@meet:~/workspace/dacs-server$ npm start

> dacs-server@1.0.0 start
> node server.js

0
unsigned transaction :  f0808502540be40082520894d76064bea3d7d99b82a4329ba0c8bb5dba07e1e688016345785d8a000080840134 62fb8080
signedTx : 0xf870808502540be40082520894d76064bea3d7d99b82a4329ba0c8bb5dba07e1e688016345785d8a000080840268c61aa0910f0b17b0803d60fe3fae804412248b1ae4cc28ba7e55a289d169d2566b13aaa033d41cd3c668be10451eedb2ff14b8c1ec3377c27
471eaf63d9ae7b161ae1f1d   from : 0x175f5aca123b8e29c0f56e9a0c9e9e7fb46d06f3  from_pub : d3667076c618868c182243b282dd315274d0c4e1800474c7a8adff7ca9edad82ecf5524998768fe56105f72b394d9afb81189cd975bdfe309b15297366e3a731
```

Finish