BCDV1011 Design Patterns for Blockchain

Server signed transactions

Server signed transactions

- When to server sign
- Create express app
- Connect to web3
- Get contract object
- Create transaction
- Sign and send

When to server sign

- For automated calls to change state in blockchain
  - Enterprise system access to smart contracts
  - CRON job connection
  - Automated oracle
  - Generated payments
  - Exchange transactions
  - Avoid custodial model
- Requires server app to have access to private key
- May require message queue to handle large volumes

Create Express app

- Create a server application in node.js using express.js framework

```
mkdir project-folder
cd project-folder

npx express-generator

npm install

npm start
```

- Check out the app in the browser

    http://localhost:3000/

Connect to Web3

- Run ganache
- Install npm module web3

```
npm install web3
```

- Add to app.js

```
const Web3 = require('web3');
var Tx = require('ethereumjs-tx').Transaction;
const web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:7545"));
web3.eth.getAccounts(console.log);
```

Get contract object

- Need the contract address and ABI - add to app.js

```
const contractAddress = 'YOUR_CONTRACT_ADD';
const ABI = require('YOUR_ABI_FILE');

var TestContract = web3.eth.contract([YOUR_ABI], contractAddress);
```

Create transaction

- Need the contract address, ABI, account, private key and nonce

```
const contractAddress = 'YOUR_CONTRACT_ADD';
const ABI = require('YOUR_ABI_FILE');
const account = '0xACCOUNT_ADDRESS';
const privateKey = Buffer.from('YOUR_PRIVATE_KEY', 'hex');
const newAddress = '0x5aB5E52245Fd4974499aa625709EE1F5A81c8157';
var TestContract = new web3.eth.Contract([YOUR_ABI], contractAddress);
const _data = TestContract.methods.setOwner(_newAddress).encodeABI();

web3.eth.getTransactionCount(account)
.then(nonce => {
   var rawTx = {
      nonce: nonce,
     gasPrice: '0x20000000000',
     gasLimit: '0x27511’,
     to: contractAddress,
     value: 0,
     data: _data
   }
```

Sign and send

- Sign the transaction

```
    var tx = new Tx(rawTx);
    tx.sign(privateKey);

    var serializedTx = tx.serialize();

    web3.eth.sendSignedTransaction('0x' + serializedTx.toString('hex'))
      .on('receipt', console.log);
});
```

Message Queue

- Asynchronous communication
- Queued messages
- Ethereum
- Transaction failures
- Options

Asynchronous communication

- Communication between two parties where responses are not required to be immediate
- Ethereum transactions take time to finalize (5secs-5mins)
- REST API calls are not guaranteed to respond fast
- Handled in Javascript
    - Callback
    - Promise
    - Await

Queued messages

- Producer/consumer mismatch
- Queue up messages and they can be handled when the system is ready
- Hide complexity - just add to queue

Ethereum

- Nonce - transaction count
- Expected that only one transaction per nonce value
- Nonce values increment
- Any mistakes in this and the transaction is rejected
- Does not work well with microservice architecture and scaling

Transaction failures

- Transactions can fail
- Need to requeue
- May need to adjust
- Message queue can avoid losing transactions

Options

- RabbitMQ - good interface
- ActiveMQ - good for scaling
- Kafka - high speed

Key Management

- Custodial/Non-custodial
- Hardware devices
- Backup process
- Key Management
- Social Key Recovery

Custodial/Non-custodial

- Custodial model means the application holds your private key for you
  - Users lose their keys
  - Honeypot
- Non-custodial
  - You control your own - no impersonation
  - You need to manage backups

Hardware keys

- Hardware device that can sign using your private key
- Hardware devices are upgradable to handle different types of signing
- Can be moved around without having to expose your key(s) to the systems you are using
- Clumsy interfaces
- Device can be lost or broken or you lose your device password

Backup process

- Pass phrase
- Paper-based
- Vault
- Biometric access to backup

Key management

- Decentralized
    - MetaMask
    - Hardware keys
    - Wallets
    - Non-custodial
- Centralized
    - Exchanges
    - dApps
    - Custodial
- Distributed
    - Fractional keys

Social key recovery

- Fractional keys shared with others
- Threshold of fractional pieces returned to recover key
- Based on social or business relationships
- Can be used for small data backup too