CodeChain Documentation

CodeChain Team <codechain@kodebox.io>

Mar 19, 2019

Contents

CodeChain is a programmable open source blockchain technology optimal for developing and customizing multi-asset management systems.

CHAPTER 1

Introduction

1.1 What is CodeChain?

CodeChain is a blockchain platform tailored to support regulatory compliance and chain-level privacy for the next generation digital securities. It provides a full-fledged vertical technology stack that covers everything from security token issuance to post-issuance governance. KYC requirements, tier 1 privacy, voting, dividend payouts, vesting, banning users, freezing accounts, transaction restrictions, etc., all these features are already in place, and new requirements can be easily added.

The platform is composed of the CodeChain network, exchange, console, wallet, explorer and a liquidity provider layer. All of these are tooling services which tokenization projects can take advantage from to provide their investors with asset management apps and services with minimal customization.

The CodeChain Asset Exchange Protocol facilitates low friction peer-to-peer exchange of assets on the CodeChain network. The protocol achieves efficiency by relaying orders off-chain and performing settlements on-chain. Security token exchanges can be built on top of the Asset Exchange Protocol.

CodeChain console is a user-friendly portal where token configuration can be easily managed with few clicks even by asset managers with no engineering background. Minimal coding of a non-Turing complete script language will be needed if token requirements are complex.

1.2 Key features:

- **Legal compliance**: Token-level built-in support for regulatory compliant issuance and transactions (KYC/AML)
- **Programmable assets**: Instant dividend or interest payouts, buybacks, voting, vesting, asset composition/decomposition/recomposition
- **Increased liquidity**: Native support for algorithm-based autonomous liquidity reserve mechanism
- **Public blockchain**: Blockchain offers decentralized, secure, transparent, and immutable transactions
- **Privacy protection**: Chain-level control over privacy for assets and owners
- **No issuance or trading fees**: The platform only charges network transaction fees

On CodeChain, issuers can issue and investors can trade asset-backed tokens in a regulatory compliant way while enjoying chain-level privacy protection.

For more information, please visit our website or read the white paper.

1.3 What features does CodeChain offer?

Key features of CodeChain include a built-in multi-asset management solution, which enable users to issue, transfer and manage currencies, tokens and/or digital items on a blockchain network, all without the need of smart contracts.

CodeChain offers programmability. More specifically, transactions are programmable. Asset evolution, fusion, etc, which are features difficult to implement with existing blockchain technologies, can now be done more efficiently and effortlessly. Through programmability, CodeChain offers a scalable blockchain platform where digital tokens can be issued and traded in a regulatory compliant way while protecting the personal information of the investors.

In addition, the console service allows you to easily issue and transfer tokens, set vesting periods, manage votes, freeze illegal transactions or accounts and payout dividends.

Finally, the CodeChain platform includes the wallet, the explorer, the exchange, and other software in one complete package so that you can provide richer services.

1.4 Learn More About CodeChain

1.4.1 Documentations

SDK API Documentation

1.5 Community

Please choose the appropriate forum when you wish to start a discussion or ask a question. The resources below serve as great starting points. We recommend using *Gitter Rooms* for quick feedback from the devs.

1.5.1 Gitter Rooms

Gitter serves as the coworking space where devs share feedback, ask questions, or just hang out. Devs are usually online, so Gitter is probably the best place to get a quick hold of someone with something important.

To go directly to CodeChain's Gitter rooms, click here. We currently have a room for each of the ongoing projects:

- codechain: CodeChain engine related.
- codechain-explorer: for looking up information, such as accounts or assets that belong in CodeChain.
- codechain-sdk-js: JavaScript SDK for CodeChain.
- codechain-agent-hub: responsible for retrieving information from the agent, such as information of CodeChain's current status, or other hardware information, such as CPU and memory usage.
- codechain-wallet: the wallet that CodeChain provides for the user.
- codechain-gateway: the component that can pay the transaction fees in place of another user. The gateway also verifies whether transactions are following specific rules.
- codechain-exchange: the exchange where various trades of assets can be made.

- codechain-helicopter: tool created for providing random airdrops of CCC or assets.

If you are not sure of which room is suitable for your topic, go to codechain and make your inquiries there.

1.6 Contributors

- Seulgi Kim
- Kwang Yul Seo
- JinGyeong Jeong
- Joon-Mo Yang
- Hyunsik Jeong
- Juhyung Park
- Seung Woo Kim
- Geunwoo Kim
- SeongChan Lee
- LEE Dongjun
- Misun Cho

CHAPTER 2

Setup

2.1 Using Docker

CodeChain supports the use of Docker to provide an easy and seamless installation process by providing a single package that gives the user everything he/she needs to get CodeChain up and running. In order to get the installation package, run the following command after installing Docker:

```
docker build -f docker/ubuntu/Dockerfile --tag codechain-io/codechain:branch_or_tag_
↪name .
```

WSL users may find difficulty in using Docker, and thus, it is highly recommended to use Ubuntu, or install Docker for Windows. When using Docker for Windows, it is necessary to enable Hyper-V in BIOS settings.

To see the Docker images created, run the following:

```
docker images
```

It will result in something like this:

```
REPOSITORY                TAG                 IMAGE ID            CREATED                 ␣
↪ SIZE
codechain-io/codechain    branch_or_tag_name  6f8474d9bc7a        About a minute ago      ␣
↪ 1.85GB
ubuntu                    14.04               971bb384a50a        6 days ago              ␣
↪ 188MB
```

If you want to run the first image file, run the following command:

```
docker run -it codechain-io/codechain:branch_or_tag_name
```

This should result in CodeChain running.

2.2 Building From Source

2.2.1 Build Dependencies

CodeChain requires Rust version 1.32.0 to build. Using rustup is recommended.

- For Linux Systems:
  - Ubuntu

  Note: `gcc`, `g++` and `make` are required for installing packages.

```
$ curl https://sh.rustup.rs -sSf | sh
```

- For Mac Systems:
  - MacOS 10.13.2 (17C88) tested

  Note: `clang` is required for installing packages.

```
$ curl https://sh.rustup.rs -sSf | sh
```

- For Windows Systems:
  - Currently not supported for Windows. If on a Windows system, please install WSL to continue as Ubuntu.

Please make sure that all of the binaries above are included in your `PATH`. These conditions must be fulfilled before building CodeChain from source.

Download CodeChain's source code and go into its directory.

```
git clone git@github.com:CodeChain-io/codechain.git
cd codechain
```

2.2.2 Build as Release Version

```
cargo build --release
```

This will produce an executable in the ./target/release directory.

2.3 Using CodeChain SDK

Before starting to use the CodeChain SDK, please install node.js by going to this page.

Next, install the package with the following command:

`npm install codechain-sdk` or `yarn add codechain-sdk`

CHAPTER 3

Configuration

3.1 How To Configure

CodeChain can be configured with either CLI options or a config file. When it comes to which options take precedence, it goes from CLI, user's own config.toml file, and config.dev.toml in that order.

CLI options can be listed by running the command `$codechain --help`. By using the CLI options, or custom config files, the user can overwrite config.dev.toml's configurations.

3.1.1 Config File

The default preset `config.dev.toml` file can be located in `codechain/config/presets/config.dev.toml`.

Config files can be customized by the user and its location can be designated by using the CLI command `--config`. Custom config files created by the user must have the proper custom path.

3.1.2 Default config.dev.toml

The following represents the default configuration values of `config.dev.toml`.

```
[codechain]
quiet = false
db_path = "db"
keys_path = "keys"
chain = "solo"

[mining]

[network]
disable = false
port = 3485
```

(continues on next page)

(continued from previous page)

```
max_peers = 30
min_peers = 10
bootstrap_addresses = []
sync = true
transaction_relay = true
discovery = true
discovery_type = "unstructured"
discovery_refresh = 60000
discovery_bucket_size = 10

[rpc]
disable = false
interface = "127.0.0.1"
port = 8080

[ipc]
disable = false
path = "/tmp/jsonrpc.ipc"

[snapshot]
disable = false
path = "snapshot"
```

CodeChain is set to use the Solo consensus algorithm by default. Tendermint is not suitable for solo testing purposes, since it requires a minimum of 4 users to function properly.

In order to test CodeChain alone, you may want to change chain to Solo. To do this, use `--chain solo`.

3.1.3 CLI Options for CodeChain client

`--config=[PATH]` Specify the certain config file path that you want to use to configure CodeChain to your needs.

`--port=[PORT]` Listen for connections on PORT. (default: 3485)

`--bootstrap-addresses=[BOOTSTRAP_ADDRESSES]` Bootstrap addresses to connect.

`--no-network` Do not open network socket.

`--min-peers=[NUM]` Set the minimum number of connections the user would like. (default: 10)

`--max-peers=[NUM]` Set the maximum number of connections the user would like. (default: 30)

`--instance-id=[ID]` Specify instance id for logging. Used when running multiple instances of CodeChain.

`--quiet` Do not show any synchronization information in the console.

`--chain=[CHAIN]` Set the blockchain type out of solo, simple_poa, tendermint or a path to chain scheme file. (default: solo)

`--db-path=[PATH]` Specify the database directory path.

`--keys-path=[PATH]` Specify the path for JSON key files to be found.

`--snapshot-path=[PATH]` Specify the snapshot directory path.

`--no-sync` Do not run block sync extension.

`--no-tx-relay` Do not relay transactions.

--jsonrpc-interface=[INTERFACE] Specify the interface address for rpc connections

--jsonrpc-port=[PORT] Listen for rpc connections on PORT. (default: 8080)

--no-ipc Do not run JSON-RPC over IPC service.

--ipc-path=[PATH] Specify custom path for JSON-RPC over IPC service

--no-jsonrpc Do not run jsonrpc.

--author=[ADDRESS] Specify the block's author (aka "coinbase") address for sending block rewards from sealed blocks.

--engine-signer=[ADDRESS] Specify the address which should be used to sign consensus messages and issue blocks.

--mem-pool-fee-bump-shift=[INTEGER] A value which is used to check whether a new transaciton can replace a transaction in the memory pool with the same signer and seq. If the fee of the new transaction is *new_fee* and the fee of the transaction in the memory pool is *old_fee*, then *new_fee* > *old_fee* + *old_fee* >> *mem_pool_fee_bump_shift* should be satisfied to replace. Local transactions ignore this option.

--mem-pool-mem-limit=[MB] Maximum amount of memory that can be used by the mem pool. Setting this parameter to 0 disables limiting.

--mem-pool-size=[LIMIT] Maximum amount of transactions in the queue (waiting to be included in next block).

--notify-work=[URLS] URLs to which work package notifications are pushed.

--force-sealing Force the node to author new blocks as if it were always sealing/mining.

--reseal-min-period=[MS] Specify the minimum time between reseals from incoming transactions. MS is time measured in milliseconds.

--reseal-max-period=[MS] Specify the maximum time since last block to enable force-sealing. MS is time measured in milliseconds.

--work-queue-size=[ITEMS] Specify the number of historical work packages which are kept cached lest a solution is found for them later. High values take more memory but result in fewer unusable solutions.

--no-discovery Do not use discovery. No automated peer finding.

--discovery="kademlia" | "unstructured" Decide which p2p discovery extension to use. Options are kademlia and unstructured. In a testing environment, an unstructured p2p network is desirable because it is more than sufficient when there are a few nodes(< 100). (default: unstructured)

--discovery-bucket-size=[NUM] Bucket size for discovery. Choose how many addresses to exchange at a time during discovery.

--discovery-refresh=[ms] Refresh timeout of discovery (ms). It may conflict with:‘‘ –no-discovery‘‘.

--no-snapshot Disable snapshots

3.2 Logging

For logging, run the following to configure: `$ RUST_LOG=<level> codechain`

3.2.1 Log Levels

CodeChain currently offers five different `<level>`. They are error, warn, info, debug, and trace.

For example, the log level will be set to debug, if you run the following:

```
$ RUST_LOG="debug" codechain
```

- The **error** level represents an event where something can be dangerous, but can still run. In the case in which it cannot run anymore, it must crash ASAP instead of logging.
- The **warn** level represents an event which can be potentially dangerous.
- The **info** level represents an event which is not dangerous, but can be useful information to the users.
- The **debug** level represents an event that is useful for the developers, but not for the users.
- The **trace** level is used for tracing.

3.2.2 Log Targets

Log levels can be set differently for each log targets. For example, you can set `tx`'s log level as `trace` and `parcel`'s log level as `info` with the following code:

```
$ RUST_LOG="tx=trace,parcel=info" codechain
```

The possible log targets are as follows:

```
"blockchain"
"client"
"discovery"
"engine"
"external_parcel"
"io"
"mem_pool"
"miner"
"net"
"netapi"
"own_parcel"
"poa"
"shutdown"
"snapshot"
"solo_authoirty"
"spec"
"state"
"state_db"
"stratum"
"sync"
"test_script"
"trie"
"tx"
```

CHAPTER 4

Basic Usage

4.1 Run Built Executable

To get started, you must first run the built executable of CodeChain.

In order to run CodeChain, run

```
./target/release/codechain
```

You can create a block by sending a transaction through JSON-RPC. In order to utilize JSON-RPC, you can use Curl or JavaScript SDK.

4.2 Blockchain Configuration

When configuring CodeChain's blockchain type, you can set it to either `Solo` or `Tendermint`.

4.2.1 Solo Configuration

CodeChain uses this configuration as default. In order to change it into another configuration, such as tendermint, run:

```
--chain tendermint
```

4.2.2 Tendermint Configuration

In order to properly get Tendermint to get going, you need to have 4 nodes up and running. To do this, first run a single node by running the following:

```
codechain --db-path db/db0 --port 3485 --jsonrpc-port 8080 --engine-signer␣
↪tccqzzpxln6w5zrhmfju3zc53w6w4y6s95mf5hw0n62 -c tendermint
```

This creates a node in db0 (database 0) at port 3485 (used for nodes to communicate with each other) and jsonRPC port 8080 (port used for external access) with engine signer of tccqzzpxln6w5zrhmfju3zc53w6w4y6s95mf5hw0n62 (used to sign the block).

Then create more nodes, and allocate each node with a secret key that corresponds to one of the four public keys listed in Tendermint's validator property. When creating new nodes, the db, port and jsonRPC port all must be configured as a different value. So for example, the next node should be set up like this:

```
codechain --db-path db/db1 --port 3486 --jsonrpc-port 8081 --engine-signer␣
↪tccqz03jn96q0kvwqzxgeq5u72e2l8v5vkdyq4cll9x -c tendermint
```

Once each public key has a corresponding node with a corresponding secret key, use the boostrap address command to interlink all the nodes together. The way each node is connected does not matter, as long as each node is connected to another node. For example, in order to make a certain node connect to the node with a secret key of 1, use this command:

```
codechain --db-path db/db1 --port 3486 --jsonrpc-port 8081 --engine-signer␣
↪tccqr8a9rqj09j9l6ahe7yq9xfjj8h5xw3p7vpcgner -c tendermint --bootstrap-addresses 127.
↪0.0.1:3485
```

4.3 Checking if CodeChain is Configured Properly

JSON-RPC is a stateless, light-weight remote procedure call (RPC) protocol. Primarily this specification defines several data structures and the rules around their processing. It is transport agnostic in that the concepts can be used within the same process, over sockets, over HTTP, or in many various message passing environments. It uses JSON (RFC 4627) as data format.

4.3.1 Using Curl

First, check whether CodeChain's RPC port is listening for RPC connections. By default it should be PORT 8080.

In order to check whether CodeChain is configured properly or not, send a ping to check whether CodeChain's RPC server is actually responding. To do this, do the following:

```
curl \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc": "2.0", "method": "ping", "params": [], "id": null}' \
    localhost:8080
```

You should get the following response, or something similar:

```
{"jsonrpc":"2.0","result":"pong","id":null}
```

4.3.2 Using JavaScript SDK

In order to use this method, first install the sdk by running the following:

```
npm install codechain-sdk
```

or

```
yarn add codechain-sdk
```

Then, make sure that your CodeChain RPC server is listening. In the examples, we assume it is localhost:8080

If you run the following code, your should receive a ping response:

```
// ping.js (javascript)
var SDK = require("codechain-sdk");

var sdk = new SDK({ server: "http://localhost:8080" });

sdk.rpc.node.ping().then(function (response) {
    console.log("Ping response:", response);
}).catch(console.error);
```

If you want to run the above example in the command line, first install `nvm` by running the following:

```
wget -qO- https://raw.githubusercontent.com/creationix/nvm/v0.33.11/install.sh | bash
```

Then run the following:

```
node -e 'var SDK = require("codechain-sdk"); var sdk = new SDK({ server: "http://
↪localhost:8080" });sdk.rpc.node.ping().then(function (response) {console.log("Ping␣
↪response:", response); }).catch(console.error);'
```

You should receive the following response:

```
Ping response: pong
```

CHAPTER 5

Account Management

5.1 Subcommands

CodeChain has `account` subcommand. It is used to manage accounts and has subcommands of its own, which are the following:

`create` Create a new account in the `keys` file directory. Upon creation, the user is asked to enter a passphrase.

`import <JSON_FILE_PATH>` Import a key in the format of a JSON file. Enter the directory that holds the JSON file to import.

`import-raw <RAW_KEY>` Import a private key(64 hexadecimal characters) directly.

`remove <ADDRESS>` Remove an account from the `keys` file directory. Use `list` to get the ADDRESS.

`list` List the managed accounts.

`change-password <ADDRESS>` Change the password of the account linked with the given ADDRESS.

5.1.1 Creating an Account

You can create a new account with the `create` command. This command will ask for the user to create a password that goes along with the newly created account.

```
./target/release/codechain account create
```

Note: Password can be left blank by simply pressing the enter key twice after using the `create` command.

After creating an account with `create`, you should see files created under `/codechain/keys` directory. These files should look something like this:

```
UTC--2018-06-21T03-24-11Z--0995f73c-ddba-d65f-a6e5-083be0df4bbb
```

Upon closer inspection, the created accounts contain the following contents:

```
{"id":"0995f73c-ddba-d65f-a6e5-083be0df4bbb","version":1,"crypto":{"cipher":"aes-128-
↪ctr","cipherparams":{"iv":"e0b2af9a7f7676b547fae2c9e6b57694"},
"ciphertext":"681389baba1ca30ba5b5610199168d819d00d318fef251279be0c5a48214c081","kdf":
↪"pbkdf2","kdfparams":
{"c":10240,"dklen":32,"prf":"hmac-sha256","salt":
↪"ddce31fe0610f9d55e0ec1c28c04c11c02c5c19d3a5d64f910a43125a2922b04"},
"mac":"7bc755edea0e64d8a1f14d9d38ebdfeabb791f8dad4f53175ed3c286e40610f7"},"address":
↪"6753f53309a778291f96e339887c1644a8d596db","name":"","meta":"{}"}
```

5.1.2 Changing the Password

You can change your password with the `change-password` command. For instance, if you want to change the password of cccqzn9jjm3j6qg69smd7cn0eup4w7z2yu9myd6c4d7, run the following:

```
./target/release/codechain account change-password␣
↪cccqzn9jjm3j6qg69smd7cn0eup4w7z2yu9myd6c4d7
```

After entering the old password, a new password can be set. If the wrong password is entered, it will throw a KeystoreError.

5.1.3 Importing an Account

Accounts can be imported in two ways. You can either define a certain directory or use a 64 character hexadecimal string. The first method can be done by using the `import` command. Let's try importing a key from the ./keys directory. This can be done as follows:

```
./target/release/codechain account import ./keys/<NAME_OF_KEY>
```

The second method uses the `import-raw` command. Let's say you want to import a private key with the value of `a159aa74f2dc23f560fdc36ad6f7ad597a8e61be4bb9e1a9edb50a9013574910`. Then you would use the following command:

```
./target/release/codechain account import-raw␣
↪a159aa74f2dc23f560fdc36ad6f7ad597a8e61be4bb9e1a9edb50a9013574910
```

The first method asks for the password of the key to import, since it is protected. The second method will ask you to set a new password for the 64 character hexadecimal string of your choice.

5.1.4 Looking Up Accounts

You can list all the accounts that are currently created by using the `list` command.

If you run the following, you should get a list of all the managed accounts' addresses:

```
./target/release/codechain account list
```

5.1.5 Removing Accounts

If you want to remove a certain account, you should first know the address of that account. To do this, simply use the `list` command. Once you found the address of the account you want to remove, simply use the `remove` command. If you want to delete an account with address `0xc3bc9c4bd0020fcc9bd294c379b2eb7284c99de5`, then use the following command:

```
./target/release/codechain account remove 0xc3bc9c4bd0020fcc9bd294c379b2eb7284c99de5
```

Then you will be asked to enter the password. Once the correct password is entered, the account will be removed.

CHAPTER 6

Consensus Algorithms

CodeChain offers a pluggable consensus model, which provides flexibility. You can choose the consensus model that best suits your needs. If the existing consensus models do not meet your business requirements, you can easily create your own consensus model.

Currently, CodeChain supports four consensus algorithms. Each consensus algorithm has its own strengths, which is why a variety is being offered.

6.1 Solo

Used for testing purposes only when there is only one node in the entire network. Solo is not a consensus algorithm.

6.2 Tendermint

Tendermint is a Proof-of-Stake algorithm which is designed to tolerate machines that fail in arbitrary ways, which is also known as Byzantine fault tolerance(BFT). Tendermint claims that even if 1/3 of the machines fail, it will still operate properly, offering a secure and consistent system.

6.3 BlakePoW

BlakePoW follows the Proof-of-Work model of Bitcoin, where a hash is calculated by adding the nonce and the block hash. It is then checked whether this added value is less than or equal to the target value over and over again. If you want an algorithm not bound to forms of processing power, please use Cuckoo.

6.4 Cuckoo

Cuckoo aims to be resistant to Bitcoin style hardware arms-races by making its algorithm memory bound. Thus, solution times should be bound to memory bandwidth instead of other forms of raw processing power. As a result, Cuckoo should be a viable solution for running on most commodity hardware, and require far less energy than other forms of PoW algorithms that are bound to GPU, CPU or ASIC.

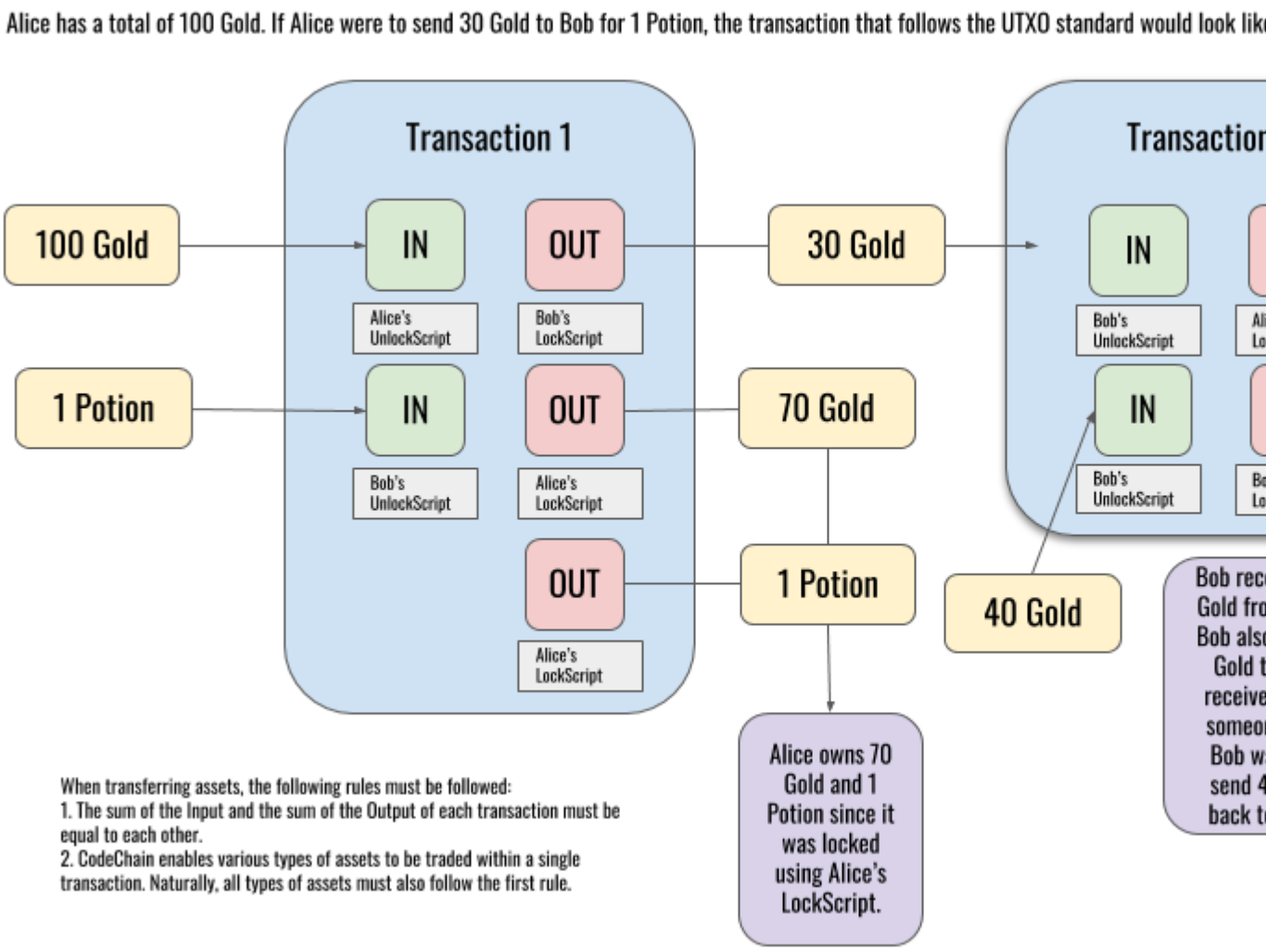
CHAPTER 7

PoW Mining Difficulty

Both BlakePow and Cuckoo are PoW-based consensus models. The mining difficulty is adjusted depending on the timestamp differences of the blocks. If the difference is less than 10 seconds, the difficulty is adjusted upwards. If the timestamp difference is between 10 to 19 seconds, the difficulty is left unchanged. If greater than or equal to 20 seconds, the difficulty is adjusted downwards proportional to the timestamp difference.

CHAPTER 8

Transactions



8.1 Transaction

Transactions can do a variety of things that change the state of various aspects within CodeChain. Obvious features of transactions include trading assets and making payments. However, a less obvious feature involves setting a regular key to accounts so that transactions can be signed with the regular key instead of the private key. Finally, there is also a feature that allows users to create shards, where assets are stored and managed.

CodeChain was developed with multi-asset management in mind, coupled with the ability for the service provider to pay transaction fees for users. Asset transactions are collected at the gateway. These gateways would be the service providers, and can pay the transaction fees for the transactions going through the respective gateways. If users want to add their transactions directly onto the blockchain without the need to go through a gateway, then they must pay their own transaction fees.

A transaction would look something like this:

```
struct Transaction {
    seq: u64,
    fee: u64,
    network_id: NetworkId,
    action: Action,
}

enum Action {
    MintAsset { ..., },
    TransferAsset { ..., },
    ChangeAssetScheme { ..., },
    ComposeAsset { ..., },
    DecomposeAsset { ..., },
    Pay { ..., },
    SetRegularKey { ..., },
    CreateShard,
    SetShardOwners { ..., },
    SetShardUsers { ..., },
    WrapCCC { ..., },
    UnwrapCCC { ..., },
    Store { ..., },
    Remove { ..., },
    Custom { ..., },
}
```

The fee of the transaction would determine its priority, meaning, how quickly it gets processed. In addition, there is also a minimum fee that can be set. The seq property exists for the purpose of preventing replay attacks.

The following is a brief explanation for different actions you can use through a transaction:

8.1.1 Mint Asset

MintAsset issues a new asset. When issuing a new asset, the asset has fields that can be designated, such as metadata, approver, and registrar. There are two types of assets that can be issued:

- A permissioned asset is an asset that has an approver. These kind of assets need permission from the specifically assigned approver in order to be transferred to other addresses.
- A regulated asset is an asset that has an registrar. The registrar can change the asset scheme and is allowed to transfer the asset arbitrarily.

8.1.2 Transfer Asset

TransferAsset transfers assets from one address to another. *TransferAsset* can also be used to make orders on the DEX.

8.1.3 Change Asset Scheme

When minting assets as described above, you create an asset scheme. This scheme defines properties of a specific asset, such as the metadata, and through *ChangeAssetScheme*, the registrar can change an asset's scheme. However, it is important to note that only the registrar has access to *ChangeAssetScheme*.

8.1.4 Compose Asset

ComposeAsset combines multiple assets into a single new package. This new package is called a composed asset, and composed assets can be used as a regular asset. Note that composed assets can be decomposed as well.

8.1.5 Decompose Asset

DecomposeAsset decomposes any composed asset. The original contents that were used as inputs for *ComposeAsset* will be returned as output of *DecomposeAsset*.

8.1.6 Pay

Pay allows a user to make a payment of a certain value of CCC to another user.

8.1.7 Set Regular Key

Regular keys are responsible for taking the place of the master key. Regular keys provide a safe way to verify one's identity and sign transactions, while keeping the original master key safe in cold storage. Regular keys are safe because they can be easily replaced if they are stolen, while allowing you to maintain the original public address of the master key. *SetRegularKey* defines the regular key that will be used by the payer. If one already exists, this transaction will overwrite the existing one with the new regular key.

8.1.8 Wrap CCC

WCCC is a wrapped version of CCC, transforming CCC into an asset. *WrapCCC* converts CCC into WCCC.

8.1.9 Unwrap CCC

UnwrapCCC converts WCCC back into CCC.

8.1.10 Store

Store is a special type of transaction that allows the addition of text onto the blockchain. This added text can also be certified by someone through that person's signature.

8.1.11 Remove

Remove removes the content added by the *Store* transaction.

8.1.12 Custom

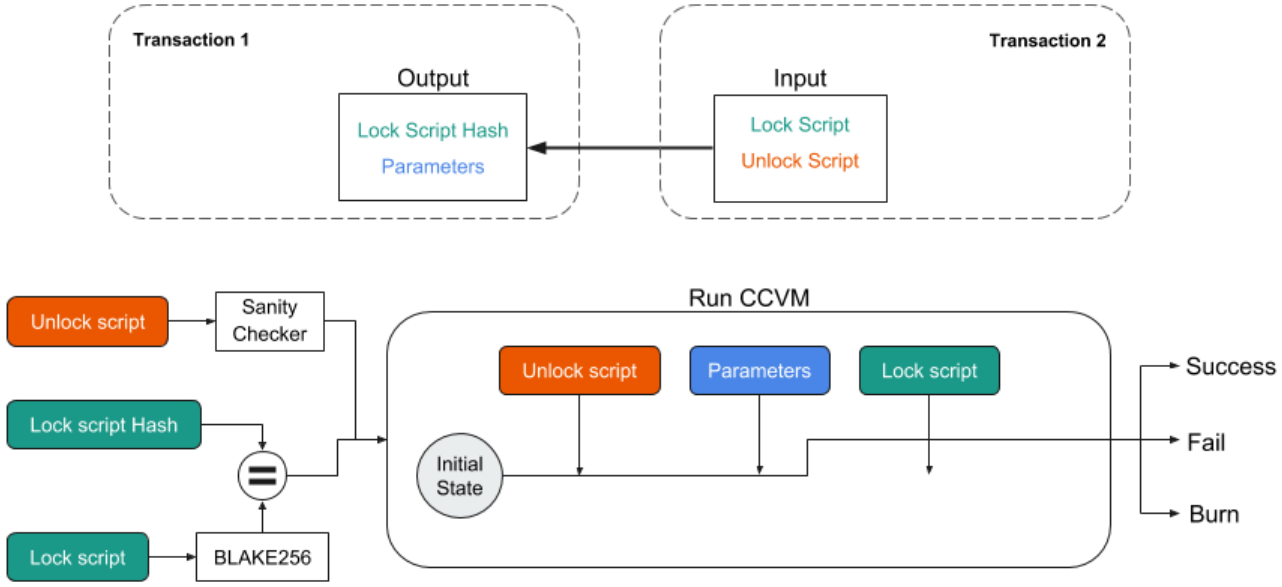
Custom is a special transaction that may have been added or needed when using a custom consensus engine.

8.2 What is UTXO?

UTXO is an acronym for Unspent Transaction Outputs, which always requires users spend their entire balance defined in a UTXO first, and then receive the unspent amount back. For instance, if you have a UTXO that defines that you have 10 potions, and you want to buy something that costs 2 potions, you would make a transaction that would "spend" your entire UTXO balance by sending 2 potions to the other person, and 8 potions back to yourself. Once this transaction is complete, a UTXO would be created, both for the spender and the receiver. In general, the UTXO specifies how much the user got back or received, which basically defines how much the user can spend. The amount the user gets back would be added to his/her account balance. Thus, it is most likely that each user would have more than one UTXOs, and the sum of all the unspent coins in every UTXO would be the user's total account balance.

8.3 Lock Script

Lock scripts are required in CodeChain when making a transaction to a different user. When attempting to make a transaction, the sender must know the receiver's lock script so that the receiver can use his/her private key to use/spend the newly received asset. This is analogous to sending money to someone's bank account. Without knowing the receiver's bank account address, you cannot send money to the proper destination. Lock scripts contain two parts: the lockScriptHash and parameter.



8.3.1 How are Lock Scripts Created?

When the user wants to receive any asset, he/she must create a private and public key pair. The public key is then used to create a lock script that the user needs so that he/she can receive assets. The codechain-sdk allows the lock scripts to be in a form of an address. This address is fundamentally a bank address in the real world. Addresses can be decoded to reveal a user's lockScriptHash and the parameter required to send a transaction.

CHAPTER 9

Gateway

Gateways are responsible for gathering transactions and approving them or paying the fee, which are then added to the blockchain. Gateways must have platform accounts that contain *CodeChain Coin*, since gateways are responsible for paying the transaction fees.

CHAPTER 10

CodeChain Coin

10.1 What is CodeChain Coin?

CodeChain Coin, abbreviated as CCC, is the name of the currency used within CodeChain. The owernship of CodeChain Coins are marked by Platform Accounts.

CHAPTER 11

Asset Management by Example

11.1 Getting Started

If you want to start creating assets that can be transferred amongst users, you can do it with codechain-sdk-js. If you visit this link, you can see an example JavaScript code. This page will guide you along on how to use codechain-sdk-js based on the example provided, called "Mint 10000 Gold and send 3000 Gold using AssetMintTransaction, AssetTransferTransaction".

Before following any examples, please make sure to carefully go through the setup section before starting any examples.

Then, check whether your CodeChain RPC server is up and running. You can read about how that is done in the configure section.

11.2 Setup the Test Account

Before you begin with various examples, you need to setup an account. The given account (cccqzn9jjm3j6qg69smd7cn0eup4w7z2yu9myd6c4d7) holds 100000 CCC at the genesis block. It's a sufficient quantity to pay for the transaction fee. You can setup the account by using this:

```
wget https://raw.githubusercontent.com/CodeChain-io/codechain-sdk-js/master/examples/
↪import-test-account.js
```

If successful, the command line will output the address of the account being used for the transaction fee. In this case, it will output cccqzn9jjm3j6qg69smd7cn0eup4w7z2yu9myd6c4d7.

Then run the downloaded .js file with the following command:

```
node import-test-account.js
```

Note: The initial 100000 CCC is only available in test mode.

11.3 Running the Sample Assets Minting Code

Once you have installed codechain-sdk, go to the installed directory and create a JavaScript file with the example code. For simplicity, we will call this sample script mint-and-transfer.js. To download the .js file, run:

```
wget https://raw.githubusercontent.com/CodeChain-io/codechain-sdk-js/master/examples/
↪mint-and-transfer.js
```

Then, run the following command:

```
node mint-and-transfer.js
```

This should give you the following result:

```
Asset {
    assetType:
    H256 {
        value: '5300000000000000179399be5182ae43b92acbb9de935000f5e33c23e6d4ceba' },
    lockScriptHash:
    H256 {
        value: 'f42a65ea518ba236c08b261c34af0521fa3cd1aa505e1c18980919cb8945f8f3' },
    parameters:
    [ [ 208,
        251,
        253,
        21,
        232,
        131,
        214,
        80,
        73,
        177,
        128,
        232,
        250,
        151,
        108,
        210,
        60,
        69,
        101,
        113,
        113,
        130,
        172,
        17,
        195,
        42,
        207,
        229,
        248,
        152,
        159,
        14 ] ],
    quantity: 3000,
    outPoint:
    AssetOutPoint {
        transactionHash:
```

(continues on next page)

(continued from previous page)

```
        H256 {
            value: '5724c9377508058a27b7fbff10d60255a429ef905792986c07571fcaf0fff980'␣
↪},
        index: 0,
        assetType:
        H256 {
            value: '5300000000000000179399be5182ae43b92acbb9de935000f5e33c23e6d4ceba'␣
↪},
        quantity: 3000,
        lockScriptHash:
        H256 {
            value: 'f42a65ea518ba236c08b261c34af0521fa3cd1aa505e1c18980919cb8945f8f3'␣
↪},
        parameters: [ [Array] ] } }
    Asset {
    assetType:
    H256 {
        value: '5300000000000000179399be5182ae43b92acbb9de935000f5e33c23e6d4ceba' },
    lockScriptHash:
    H256 {
        value: 'f42a65ea518ba236c08b261c34af0521fa3cd1aa505e1c18980919cb8945f8f3' },
    parameters:
    [ [ 174,
        155,
        53,
        229,
        89,
        202,
        36,
        156,
        33,
        75,
        16,
        147,
        201,
        78,
        224,
        71,
        48,
        132,
        174,
        192,
        113,
        187,
        89,
        29,
        225,
        236,
        112,
        109,
        204,
        115,
        84,
        88 ] ],
    quantity: 7000,
    outPoint:
    AssetOutPoint {
```

(continues on next page)

(continued from previous page)

```
        transactionHash:
        H256 {
            value: '5724c9377508058a27b7fbff10d60255a429ef905792986c07571fcaf0fff980'␣
↪},
        index: 1,
        assetType:
        H256 {
            value: '5300000000000000179399be5182ae43b92acbb9de935000f5e33c23e6d4ceba'␣
↪},
        quantity: 7000,
        lockScriptHash:
        H256 {
            value: 'f42a65ea518ba236c08b261c34af0521fa3cd1aa505e1c18980919cb8945f8f3'␣
↪},
        parameters: [ [Array] ] } }
```

In this example, 10000 gold has been minted for Alice. Alice then sends 3000 gold to Bob. Let's see how all of this works specifically by inspecting parts of the code one by one.

11.4 Setting Up Basic Properties

Make sure you are accessing the CodeChain port. In this example, it is assumed that you are using port 8080, since that is the default value.

```
const sdk = new SDK({ server: "http://localhost:8080" });
```

The MemoryKeyStore is created for testing purposes. In real applications, the MemoryKeyStore would be in the form of storage, such as hardware wallets or the key store server, which would hold and manage the key pair (private and public keys). If you want to use the key store server see below *remote key store*. The P2PKH is responsible for locking and unlocking scripts.

```
const keyStore = await sdk.key.createMemoryKeyStore();
const p2pkh = await sdk.key.createP2PKH({ keyStore });
```

Each user needs an address to receive/send assets. Addresses are created by p2pkh. In this example, Bob's address is introduced differently, since Bob's address is recieved from Bob. In real world applications, you would only know the address of the recipient and nothing more.

```
const aliceAddress = await p2pkh.createAddress();
const bobAddress = "ccaqqqap7lazh5g84jsfxccp686jakdy0z9v4chrq4vz8pj4nl9lzvf7rs2rnmc0";
```

If you want to see Alice's address, run the following:

```
console.log(aliceAddress.toString());
```

This will result in showing you an address that is identical to the format of Bob's address shown above.

11.5 Minting/Creating New Assets

In order to create new assets, you must create a new instance of AssetScheme. In this example, we create 10000 gold with the following code:

```
const goldAssetScheme = sdk.core.createAssetScheme({
    shardId: 0,
    metadata: JSON.stringify({
        name: "Gold",
        description: "An asset example",
        icon_url: "https://gold.image/",
    }),
    supply: 10000,
    approver: null,
});
```

Note: You should note that the approver is kept as null. This value is only filled out when there should be an overseer amongst transactions. If not null, the approver must approve of every transaction being made with that newly created Asset. In this case, if the 10000 gold that was minted had a approver, then every time any of those 10000 gold is involved in a transaction, the set approver would have to sign off and approve for the transaction to be successful.

After Gold has been defined in the scheme, the supply that is minted but belong to someone initially. In this example, we create 10000 gold for Alice.

```
const mintTx = sdk.core.createAssetMintTransaction({
    scheme: goldAssetScheme,
    recipient: aliceAddress
```

11.6 Sending/Transferring Assets

Alice then sends 3000 gold to Bob. In CodeChain, users must follow the UTXO standard, and make a transaction that spends an entire UTXO balance, and receive the change back through another transaction.

Next, we create an output which gives 3000 gold to Bob, and returns 7000 gold to Alice.

```
const firstGold = mintTx.getMintedAsset();
const transferTx = sdk.core.createTransferAssetTransaction()
    .addInputs(firstGold)
    .addOutputs({
        recipient: bobAddress,
        quantity: 3000,
        assetType: firstGold.assetType
    }, {
        recipient: aliceAddress,
        quantity: 7000,
        assetType: firstGold.assetType
    });
```

By using Alice's signature, the 10000 gold that was first minted can now be transferred to other users like Bob.

```
await transferTx.sign(0, { signer: p2pkh });
transferTx.getTransferredAssets();
```

The transaction containing the Gold asset is sent to the node. The transaction fee is paid for by the account known as `cccqzn9jjm3j6qg69smd7cn0eup4w7z2yu9myd6c4d7` with the passphrase `satoshi`.

```
await sdk.rpc.chain.sendTransaction(transferTransaction, {
    account: "cccqzn9jjm3j6qg69smd7cn0eup4w7z2yu9myd6c4d7",
```

(continues on next page)

(continued from previous page)

```
    passphrase: "satoshi",
});
```

In order to check if all the transactions were successful, we run the following:

```
// Unspent Bob's 3000 golds
console.log(await sdk.rpc.chain.getAsset(transferTx.hash(), 0));
// Unspent Alice's 7000 golds
console.log(await sdk.rpc.chain.getAsset(transferTx.hash(), 1));
```

This should return the following:

```
[RESULTS WILL BE FIXED AND REUPLOADED]
```

[EXPLANATION WILL BE REVISED]

These are the values of each individual's LockScripts that went through the blake256 hash function. If you run each individual's LockScript under blake256 yourself, you will find that it corresponds to the rightful owners of the assets.

11.7 Address Format

CodeChain adopted Bitcoin's Bech32 Specification. However, there are differences. Codechain does not have a seperator. Also, there are two types of addresses used in CodeChain, which are Platform Address and Asset Address. Platform Addresses are used for CCC, while Asset Addresses are used for mintable assets. These addresses have a human readable part, followed by code. Platform Addresses have a `"ccc"` tag, while Asset Addresses have a `"cca"` tag.

11.7.1 Platform Account Address Format

HRP: `"ccc"` for Mainnet, `"tcc"` for Testnet.

Data Part: `version . body`

Version 0 (0x00) Data body: `Account ID` (20 bytes)

Account ID is a result of ripemd160 of blake256 of a public key (64 bytes uncompressed form).

11.7.2 Asset Transfer Address Format

HRP: `"cca"` for Mainnet, `"tca"` for Testnet.

Data: `version . body`

Version 0 (0x00) Data body: `type . payload`

Type 0 (0x00) Payload: <LockScriptHash> (32 bytes)

Type 0 with given payload represents:

Lock Script Hash: <LockScriptHash> Parameters: [] Type 1 (0x01) Payload: <Public Key Hash> (32 bytes)

Type 1 with given payload represents:

Lock Script Hash: P2PKH Standard Script Hash Parameters: [<Public Key Hash>]

11.8 Use RemoteKeyStore to save Asset Address private key

You should use a key management server to use Asset Address private keys safely. You can use a standalone key management server from this link. In this section, we will install and run the key management server, and use the server in the SDK.

11.8.1 Setup the server

To run the key management server, nodejs and yarn should be installed.

Clone CodeChain-Keystore repository from the below URL.

```
git clone https://github.com/CodeChain-io/codechain-keystore-server.git
```

Move to the directory

```
cd codechain-keystore
```

Install the dependencies

```
yarn install
```

11.8.2 Run the server

Below command will run the server

```
NODE_ENV=production yarn run start
```

11.8.3 Use the SDK's RemoteKeyStore

The SDK can use the key management server through `RemoteKeyStore` class.

```
const keyStore = await sdk.key.createRemoteKeyStore("http://<key-management-server-
↪address>");
```

If you are running the keystore server in the same machine, you can use the `keyStore` object instead of the memory keystore. Refer to the example below:

```
const keyStore = await sdk.key.createRemoteKeyStore("http://127.0.0.1:7007");
```

11.8.4 Example

Here is a sample which uses `RemoteKeyStore` to create and get accounts. If you run this example multiple times, the number of printed keys is increased every time.

```
var { RemoteKeyStore } = require("codechain-sdk/lib/key/classes")
async function main() {
  var keyStore = await RemoteKeyStore.create("http://<key-management-server-address>
↪");
  await keyStore.createKey({ passphrase: "mypassword" });
```

(continues on next page)

(continued from previous page)

```
  var keys = await keyStore.getKeyList();
  console.dir(keys);
}
main().catch(err => console.error(err));
```

CHAPTER 12

Glossary

A

B

Byzantine Fault Tolerance (BFT)

Byzantine failure is when a system loses service due to a Byzantine Fault. Thus, BFT defines the level of immunity of a certain system from those Byzantine faults.

C

D

E

F

G

H

Hot-stuff

I

J

K

L

M

N

O

P

Proof of Stake (PoS)

An alternative to PoW, PoS puts dependency on the amount of resources that someone holds.

Proof of Work (PoW)

A piece of data which is difficult to produce but easy to verify. Producing PoW is a random process, and requires a lot of trial and error.

Q

R

S

T

Tendermint

Software for securely and consistently replicating an application on multiple machines. To learn more about Tendermint, click here.

U

V

W

X

Y

Z

CHAPTER 13

Frequently Asked Questions

13.1 Questions

13.1.1 What is CodeChain?

CodeChain is a programmable open source blockchain technology optimal for developing and customizing multi-asset management systems.

13.1.2 How is CodeChain unique?

CodeChain offers a modular architecture that allows it to be seamlessly incorporated into a variety of different blockchain systems.

13.1.3 How do I report bugs?

If you have questions whether something is a bug or not, please use our *Gitter Rooms* to ask questions first. If you are certain that something is a bug, please report it as an issue at CodeChain's git page. For security issues, please email us at codechain@kodebox.io.

13.1.4 How efficient is CodeChain?

CodeChain aims to solve the scalability issues that many blockchains face as they grow larger. By integrating sharding, CodeChain provides horizontal scaling to achieve higher transaction speeds.

13.1.5 How do I get started?

Check out *Setup* to get started. It should give you the general guidelines required to get everything setup and running.

13.1.6 I still have questions!

No worries. There is a *Community* that is willing to help you.

CHAPTER 14

Security

Different from common bugs, security issues that are an immediate threat to CodeChain's well-being should be reported directly to us at codechain@kodebox.io. When reporting such security issues, it would be of great help if you refer to the following guidelines:

Security issues fall into one of three categories. These three categories are classified as 3 levels: **P1(high)**, **P2(medium)**, and **P3(low)**.

- **P1(high)**: a security vulnerability that will result in loss of value.

  e.g. Steal tokens from someone, mint tokens at your discretion

- **P2(medium)**: a security vulnerability that will not result in loss of value but can result in a loss of function of the CodeChain engine.

  e.g. Block actions for all users

- **P3(low)**: a security vulnerability that will not result in loss of value or function but can cause great inconvenience for some fraction of users.

  e.g. Block a user from transferring tokens

When reporting security issues, please mention the security issue's category in the email's subject/title.