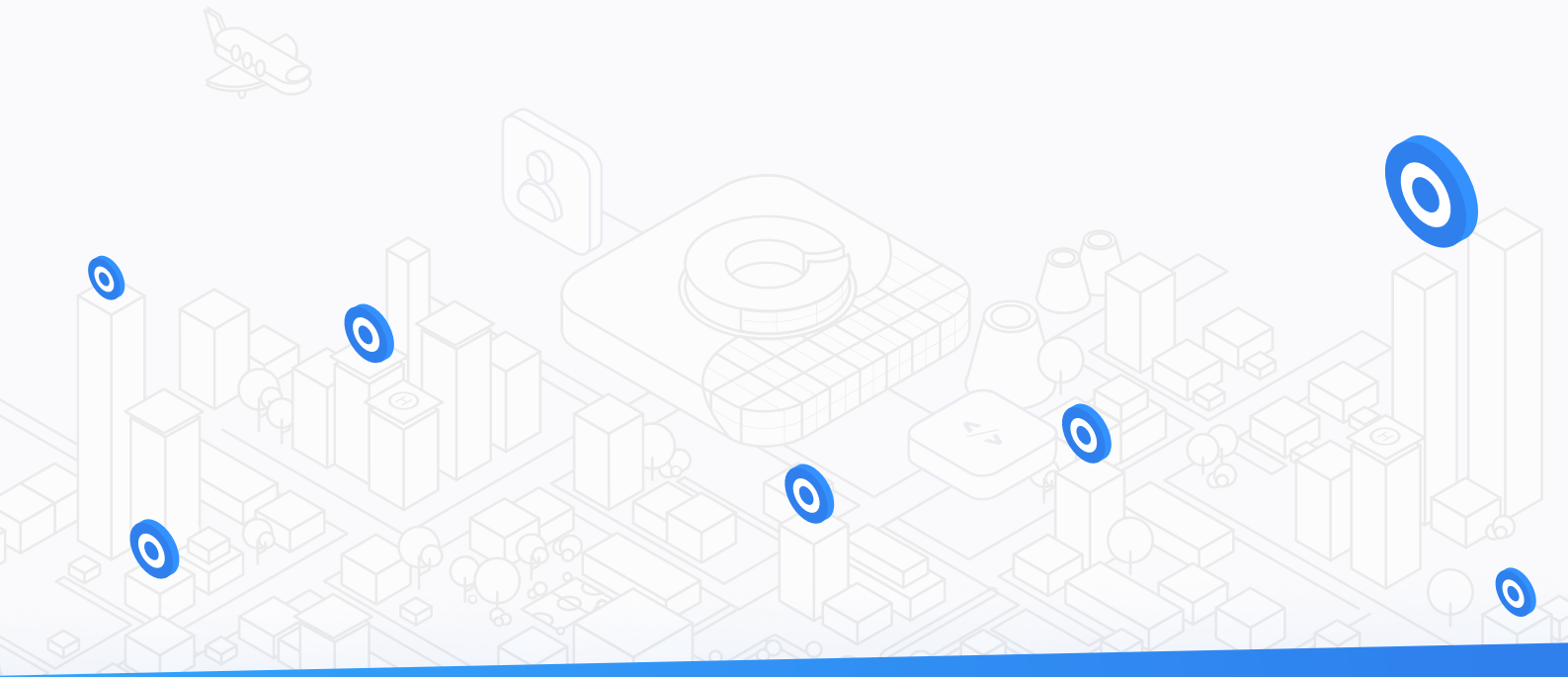
OPEN Chain



YELLOW PAPER

CONTENT

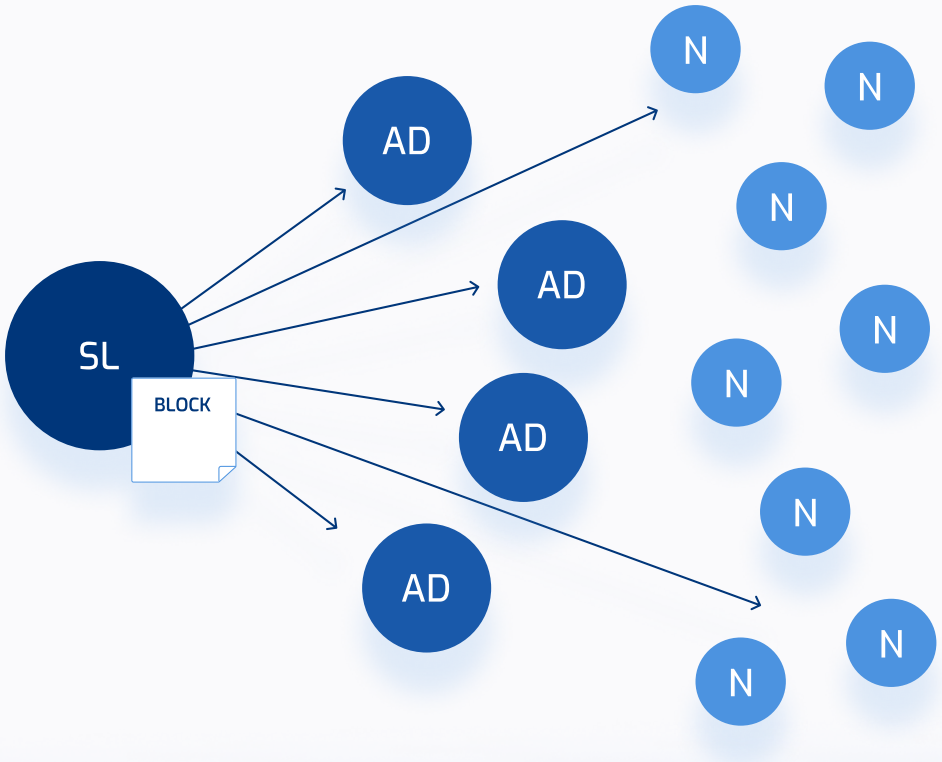
37 REMOTE PROCEDURE CALL

CONSENSUS

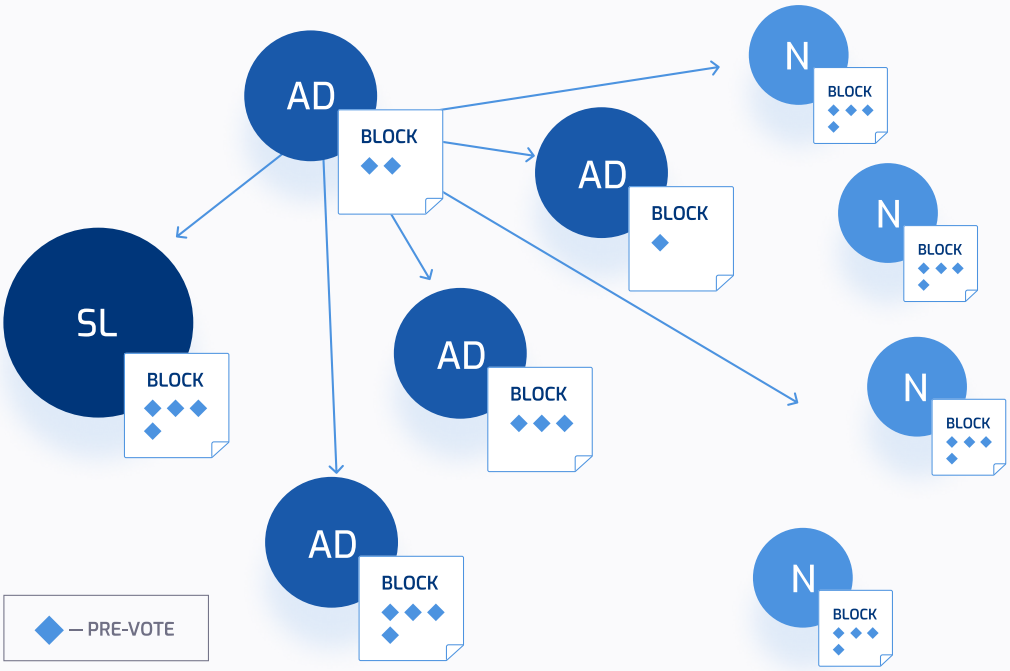
Open Chain consensus is based on the principles of DPoS and BFT algorithms.
In OPEN consensus N nodes are elected to perform a role of Active Delegates for each Epoch. An Epoch is a predetermined interval (e.g. 21 blocks) during which an Active Delegates set is fixed. The duration of each Epoch is limited by Timeslots. Number of Timeslots may differ in accordance with a number of Timeslots required to reach the necessary blocks amount. For each Timeslot, a Timeslot-Leader is defined. Timeslot-Leader is one of the Active Delegates who produces the block. At that time, the rest of the Active Delegates act as validators. Each Timeslot includes five major phases:

- PREPARE – the phase includes an appointment of a Timeslot-Leader, production of a block, block broadcasting
- PREVOTE – at this stage when a node receives a block, it sends a signed pre-vote message
- COMMIT - when a node receives ⅓ of the total amount of pre-vote messages from the Active Delegates, it validates the block and sends a signed commit message
- SAVE - when ⅔ of the total amount of commit messages from the Active Delegates is received, the block is saved
- IDLE – this stage is known as intermission (it takes place when the block processing of the current timeslot is finished but the next timeslot has not been started yet.). This stage is necessary for the chain synchronization.

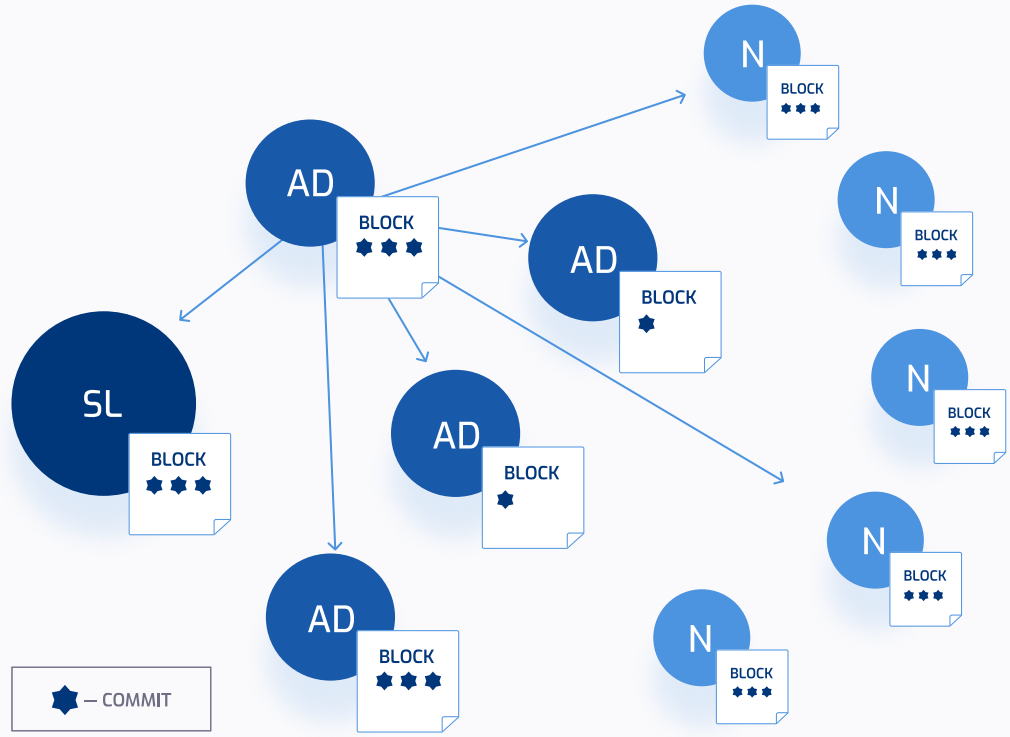
PREPARE stage:



PREVOTE stage:



COMMIT stage:



BLOCK PRODUCTION

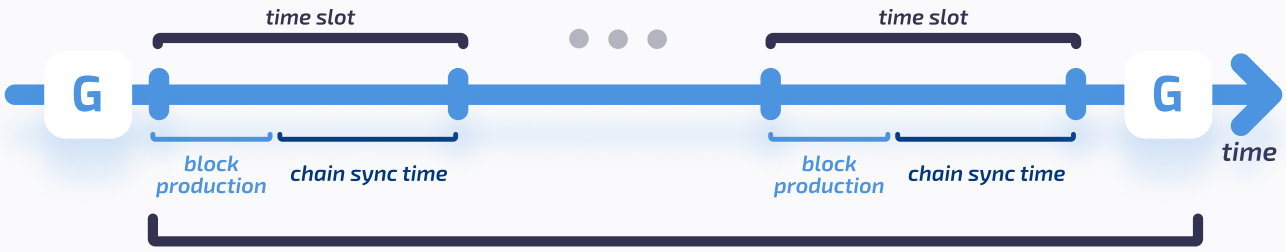
A node has a scheduled job. It is invoked each time, when a timeslot number has changed. The duration of a Timeslot is 18 seconds. Timeslot calculation is conducted based on a timestamp, when the current epoch starts.

Example: Let's consider an epoch started at a timestamp, equal to 38, the current timestamp is 51 and the timeslot duration is 5 seconds. To calculate, when this timeslot will change, the next calculation should be executed:

$$Timeslot_{time\ left} = Timeslot_{duration} - ((Epoch_{start} - Timestamp) \% Timeslot_{duration})$$

e.g. 5 - ((51 - 38) % 5) = 2

The timeslot is checking a sync status of the time and chain. Unless they are synchronized, the block creation won't happen.



EPOCH = 21 Main Block + 1 Genesis Block

GENESIS BLOCK PRODUCTION

A Genesis block is produced when the Epoch contains a sufficient amount of Main blocks. The Genesis block sets rules for the next Epoch. In the current release, it defines a set of Active Delegates. Besides an Active Delegates list Genesis Block also includes a Genesis Nodes list. Genesis nodes need to support the consensus execution in case Active Delegates fail to do it. Genesis block doesn't contain transactions and it is produced by each peer separately. In accordance with vote transactions processed, each node gets a set of Active Delegates for the next epoch, that is contained in the Genesis block. When Genesis blocks have the same height throughout the whole network, it's a proof of consensus mechanism stability.

MAIN BLOCK PRODUCTION

A main block is created by a current Timeslot-Leader. Block has 6 seconds to be created and sent to Active Delegates. The rest 12 seconds are dedicated to voting and in some cases to a synchronization process.
A main block may contain an unlimited amount of vote and delegate transactions plus up to 1000 transfer transactions. Also a block producer creates one more transaction to reflect an amount of the reward granted for the block creation. Processing of transactions results in a set of receipts and states.

State describes a wallet's actual balance at the moment of the creation of the block that is linked to it. Receipt is a result of transaction execution: transaction actually may be directed to multiple wallets, in case there is a contract, which splits the incoming amount among the beneficiaries. Based on transactions, states and receipts, node creates an appropriate merkle hash, which is to be used to form block's hash. At the end, a block hash is signed and a node's public key is attached to the block to provide other Active Delegates with a possibility to validate the signature.

The production order is specified in the Genesis block: from the most voted delegate to the one who has received the smallest amount of votes. There could be situations, when a Delegate hasn't managed to produce a block in its timeslot. And such a problem is crucial for the network, as epoch should contain exactly 21 main blocks. It is not the only threat. It is also possible that more than 34% of all Active Delegates become unavailable. To address the issues of these kinds, a set of boot nodes is attached to each set of Active Delegates. It ensures that if one of Delegates fails production, a boot node will produce a missing block.

BLOCK VALIDATION

Those Active Delegates who do not participate in the block creation, validate a block.
It is necessary to receive messages of three types from network to have a possibility to validate a block and add it to the chain:

- **PendingBlockMessage** – it contains information on the block's content for validation

- **BlockApprovalMessage** – it's a message about the node's transition into either PREPARE or COMMIT stages. It contains the following fields:
  - **stageId** – a digit stage identifier (2 for PREPARE stage, 3 for COMMIT stage)
  - **hash** – a hash of the blocks that were validated
  - **publicKey** – a key of the delegate that sent a message
  - **signature** – a proof that the message was created by a valid delegate

Block validation is executed once the node accepts an incoming block. Active delegates needs to confirms that the following statements are true in regards to the incoming block:

- The block is created by the current timeslot owner
- The transactions merkle root is valid
- The states merkle root is valid
- The seceipts merkle root is valid
- The reward transaction is created correctly
- The transactions are valid
- Resulted receipts and states coincide with the message's ones

If the validation process ends successfully, a node considers such a block to be a valid candidate to be stored. But before it becomes possible, a node should accumulate a specific number of BlockApprovalMessages:

- 34% from the Active Delegates count of PREPARE messages
- 67% from the Active Delegates count of COMMIT messages

When a node has already accumulated a sufficient number of COMMIT messages, a block is persisted to storage

There could be such cases, when a block in a specific timeslot is not valid for the local chain. But the node still tracks the invalid block. It means that if it collects the required amount of COMMIT messages, the node will allow the chain synchronization.

While Active Delegates are responsible for validation, other participants of the network broadcast blocks further and collect only commit messages, without performing any validation functions.

PARAMETERS

The table below describes the parameters used in the consensus mechanism.

Check	Peer	Active Delegate
epochHeight	21	Number of main blocks in a single epoch. When this number is reached, it means that the next block should be agenesis one and it will start new a epoch
delegatesCount	21	Number of active delegates in one epoch that are to be fetched with s rating order
blockCapacity	1000	Maximum amount of transfer transactions to be packed in the main block
timeSlotDuration	6000	Amount of time to create block and broadcast it to network
timeSlotInterval	12000	Amount of time needed to reach consensus and sync local storage
genesisAddress	0x0000000000000000000000000000000000000000	Address of the genesis account, that is a sender of the fixed reward for the block creation
rewardBlock	10	Amount of reward for block creation
feeVoteTxFor	3	Fee for creation of a vote transaction
feeVoteTxAgainst	1	Fee for recall of a vote transaction
feeDelegateTx	3	Fee for creation of a delegate transaction
amountDelegateTx	10	Amount necessary for creation of a delegate transaction

VOTING

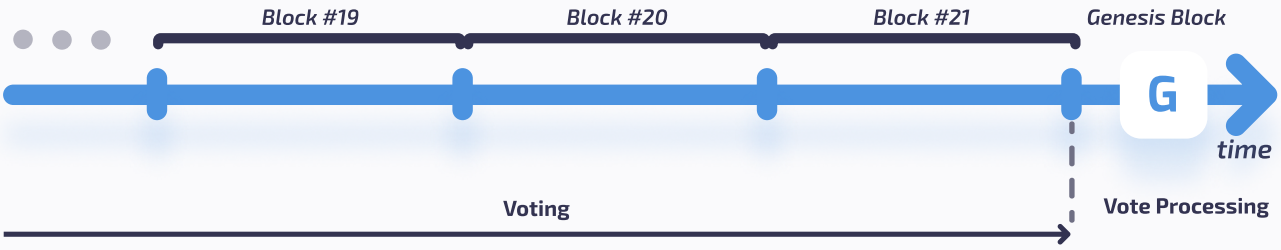
Every node can exist in three states: Peer, Delegate, Active Delegate.

- **Peer** – a node with minimal influence in the Consensus reaching process. It can store a incomplete ledger (light synchronization)

- **Delegate** – a trusted node with a full ledger
- **Active Delegate** – a node that is elected to fulfill a role of one of block producers for a specific epoch

To ensure decentralization of the network, participants who want to become block producers, need to become Delegates and stakeholders of the network have an opportunity vote for the Delegates. In this process stakeholders' voting weight depends on the size of their stake.
To become a Delegate, a Peer should initiate a Delegate-Transaction which bonds a specific Node with a specific Wallet. Each Peer can be bonded with only one Wallet. But one Wallet can be bonded with a lot of Peers. Wallets can vote for Delegates as well as recall their votes. Each Delegate has its rating which is calculated as a sum of the weights of votes in its favor.

Each Wallet has only one vote, which could be given to only one specific Delegate. Thanks to the possibility to recall votes, a stakeholder can choose a Delegate, vote for him and then to change his mind and vote for another Delegate after recalling his vote. There is a fixed vote transaction fee.
Votes are counted on the Genesis blocks creation. Each node chooses Delegates from its local storage in accordance with their rating. The order is stored in the genesis block. In accordance with it, the first block is created by the most voted delegate and the last block by the less voted one.
Voting mechanism is based on Vote-Transaction transmitting.



CORE

This package contains business logic and represents a set of rules, principles, and behavior dependencies of domain objects. This layer consists of business models. The models are represented by domain entities, services that implement the business logic, such as validation of models, repositories designed to transfer business models between the database and the application.

Business entities are stored in an H2 database, this vendor was selected based on the following requirements:

1. Ease of use for the user, no need for additional settings;
2. Support embedded mode;
3. Support in-memory table;
4. Support transaction;
5. Support ACID;
6. Support Hash index;
7. Fast queries executing;
8. Small size DB file.

MODELS

Models are a representation of concepts from the subject area, which are represented by temporary or permanent metadata.

TEMPORARY METADATA

These data are represented by unconfirmed transactions that have not been processed and stored in the ledger. These transactions are the source for creating blocks and are stored in the in-memory table until it gets into a block.
There are three types of unconfirmed transactions that may exist in the application:

- **UnconfirmedTransferTransaction**
  - Amount
  - Recipient Address
  - Data — field for deploy Smart Contract

- **UnconfirmedDelegateTransaction**
  - Delegate address
  - Amount

- **UnconfirmedVoteTransaction**
  - Delegate address
  - Vote type

PERMANENT METADATA

These models represent a ledger. This type of data is based on transactions packed into the blocks which makes it more convenient to work with them and validate them.

TRANSACTIONS

A transaction is a signed data package that is put into blocks. There are four types of transactions:

- **TransferTransaction** — used to transfer currency or deploy a smart contract.
- **DelegateTransaction** — applied when a participant becomes a delegate.
- **RewardTransaction** — contains the reward for creating a block.
- **VoteTransaction** — used to vote for or against the delegate.

All Transactions consist of:

- Timestamp
- Fee
- Sender address
- Hash
- Signature
- Public key
- Payload

Transactions differ in a payload, which is specific to each transaction:

- **TransferTransactionPayload**
  - Amount
  - Recipient Address
  - Data — field for deploy Smart Contract

- **DelegateTransactionPayload**
  - Delegate address
  - Amount

- **RewardTransactionPayload**
  - Wallet address
  - Reward

- **VoteTransactionPayload**
  - Delegate address
  - Vote type

BLOCKS

A block is a collective entity for various transactions, which is used to simplify processing of transactions. There are two types of blocks: GenesisBlock is created before each epoch and determines delegates for its epoch, MainBlock is created by the active delegate and represents a container for processed transactions and their meta information. Blocks differ in their payload:

- GenesisBlockPayload
- MainBlockPayload.

Block consist of:

- Timestamp
- Height
- Previous Hash
- Hash
- Signature
- Public Key
- Payload

GenesisBlockPayload consists of:

- Epoch Index
- Active Delegates — List of active delegates for this epoch

MainBlockPayload consists of:

- Reward Transaction
- Vote Transactions
- Delegate Transactions
- Transfer Transactions
- Delegate States
- Account States
- Receipts
- Transaction Merkle Hash
- State Merkle Hash
- Receipt Merkle Hash

Separate merkle hashes for transactions, receipts and states determine the block content.

CONTRACT

This entity is required to store smart contract information in the application. It contains a source code of the contract in the binary form, it needs to be loaded to JVM for execution. It also contains an address of its owner, its self-address that is used to call the contract by its address , a cost for its execution, ABI interface smart contract that describes the methods available.

RECEIPTS

Receipt refers to a transaction and contains transaction status and all changes which it produces. Receipts are stored in an appropriate block. Receipt contains:

- Transaction hash
- Block
- Set of transaction results

Transaction result contains:

- From
- To
- Amount
- Data

- Error

Simple transfer transaction has two results. The first is the coin transfer between sender and receiver. The second one is fee transfer between sender and block producer. Field Data serves for additional information like an address of a smart-contract.

STATE

Execution of transactions results in state transitions that are based on receipts. For each account, only one final state can be generated per block. While they are stored in a relevant block, the latest state should be considered to be most up-to-date and confirmed one.

State contains common fields:

- Address
- Block Id

AccountState contains:

- Balance
- Vote for
- Storage — it is a serialized state of a smart contract

DelegateState contains:

- Rating
- Wallet Address
- Create Date

STORE SYNCHRONIZATION

Store synchronization takes place based on the synchronization mode. There can be two types of synchronization:

- LIGHT — only carcass of blocks is saved.
- FULL — blocks are saved with entire inner metadata (transactions, receipts, states).

SYNC MESSAGES

Messages in the pre-init synchronization step:

- **BlockAvailabilityRequest** — a message used to clarify whether there is a block with a hash.
  - Hash

- **BlockAvailabilityResponse**
  - Hash
  - Height
  - Genesis Block

Messages in the pre-init synchronization step:

- **EpochRequestMessage**
  - Epoch Index
  - Sync Mode

- **EpochResponseMessage**
  - Delegate Key
  - Is Epoch Exists
  - Genesis Block
  - List of main block

STEP OF SYNCHRONIZATION

Synchronization starts when the node gets a block that meets the following requirements:

- Its height is greater than the height of the node's previous block,
- Previous Hash is not equal to node's last block hash.

INIT

- Set synchronization status to PROCESSING.
- Request *BlockAvailabilityRequest* with last block hash to the randomly chosen last known active delegate, and schedule the request to the next active delegate if the previous one doesn't respond.
- Obtain BlockAvailabilityResponse, cancel the previous scheduled task.
  - If *BlockAvailabilityResponse* contains height = -1, it means that it's not a valid epoch and it should be removed.

        - All data for this epoch should be removed.
        - Repeat the first step with BlockAvailabilityRequest.
    - Else start synchronization by node's last genesis block.

EPOCH SYNCHRONIZATION

In this step SyncSession takes place in accordance with SyncMode (FULL or LIGHT) and it starts to send requests to previous epochs.

SyncSession which is an instance of a session consists of:

- Sync Mode
- Last Local Genesis Block
- Current Genesis Block
- Epoch Quantity - Count of epoch to be downloaded.
- Storage - temp store for received blocks.
- Completion flag
- Epoch Added - Count of epoch added.

Received blocks that are considered to be valid are added to SyncSession.Storage. When received blocks are considered to be invalid, the session is deleted along with the storage and a new session begins.

When all epochs are downloaded, they are saved into the DB and the synchronization status is to be changed to SYNCHRONIZED. Synchronization continues until the last relevant block in the network is received.

DB CHECKER

A DB checker runs before the start of application and it prepares DB for running the application.

PREPARE DB

DB checker's function is to prepare the database for work. This method uses a sync mode parameter. This parameter is necessary to assign a node synchronization type. A sync mode may have the following values:

- Light
- Full

When a 'Light' synchronization takes place, the DB checker validates the following block components:

- Signature
- Hash
- Previous hash
- Timestamp
- Height
- Receipts
- States

In case with synchronization of a 'Full' type, additionally, it needs to validate block transactions.
Delete invalid chain part
In case when the chain has an invalid block, the DB checker removes this part of the chain from this block to its end.
After the validation is complete, the application is started.

CRYPTOGRAPHY

Cryptography is one of the core aspects of blockchain technology.

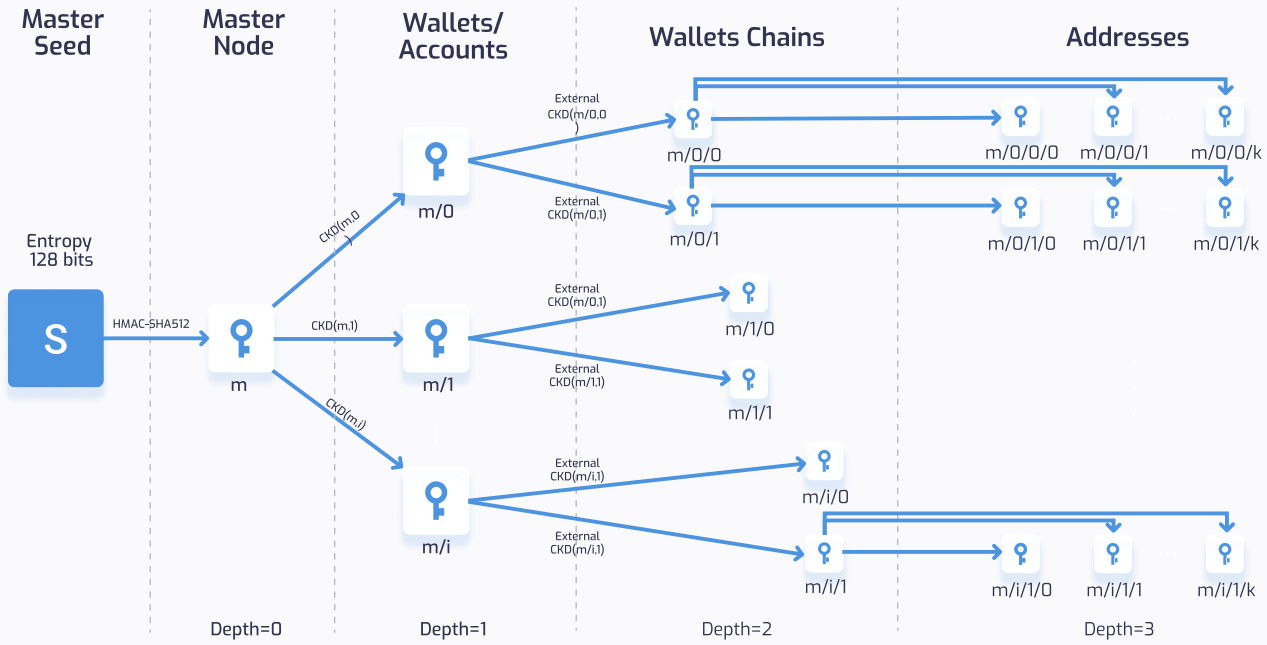
BITCOIN IMPROVEMENT PROPOSAL

BIP32 (HIERARCHICAL DETERMINISTIC WALLETS)

OPEN supports HD Wallets generation. Wallets of this type can be shared partially or entirely with different systems that may presuppose or not presuppose spending coins.

HD Wallets generate a tree-like structure of keys. In such a structure, a parent key can derive a sequence of children keys, each of them can derive a sequence of grandchildren keys, and so on, to an infinite depth.

According to the BIP32 rules, at each level of the hierarchy, the spawning node has three objects: a private key, a public key and a chain code that is used to generate the next level of a hierarchy.



Child Key Derivation Function ~ CKD(x,n) = HMAC-SHA512(Xchain, XpublicKey || n)

Derivation path: **m / account ' / chain ' / address '**, where account, chain, and address are indexes in the path.

The external chain (index = 0) is used to send and receive funds from others (m/0/0/0). The internal chain (index = 1) can be used to generate an address for receiving changes (m/0/1/0).

Wallets are divided into groups (e.g. personal, work) in accordance with the second parameter. E.g. m/0/0/0 - personal account, m/1/0/0 - work account

Derivation path of the default account is m/0/0/0

Examples:

Derivation path: m/0/0/0

Seed phrase	**monkey tissue dream ketchup myth luxury fee plate teach either shadow web**
Private key	**408d2eea0f9acee62d19643822d7084cd9c717a5958fba8b1f63792500a78f9a**
Public key	**03a882e56974a27642f9a97ba9a7936c4cdab86894c9959ef8708fa1f9f7118b1c**
Wallet address	**0xd38f124D7FE13e5693C59e6729B88D482943723C**

Derivation path: m/0/0/1

Seed phrase	**monkey tissue dream ketchup myth luxury fee plate teach either shadow web**
Private key	**0fc23fcf9f035d1dc5823c44cf22e38acab3fe7229251aad2adbd9debda876d9**
Public key	**02277d37802afb983286ca2103d3ee16e8e86b34f000f5364f4c67a25f82d15d2d**
Wallet address	**0x10d32FbBEF1b9A67EE37Da416b187E435dD6CF50**

Hierarchical deterministic wallets are described in BIP32 document

BIP39 (MNEMONIC CODE FOR GENERATING DETERMINISTIC KEYS)

Mnemonic code or mnemonic sentence is a group of easy to remember words. It could be written on paper or spoken over the telephone.

The mnemonic must encode entropy in a multiple of 32 bits. Though with more entropy security is improved, the sentence length increases.

Let:

- ENT - entropy length, bit, ENT [128,256]
- CS - checksum length, bit
- MS - mnemonic sentence length, words count

Generating the mnemonic:

1. Create a random sequence (entropy) of 128 to 256 bits.
2. Create a checksum by taking the first **ENT / 32** bits of its SHA256 hash.
3. Add the checksum to the end of the random sequence.
4. Divide the sequence into sections of 11 bits, using those to index a dictionary of 2048 predefined words.
5. Convert these numbers into words and use the joined words as a mnemonic sentence.

The following table describes the relation between the initial entropy length (ENT), the checksum length (CS) and the length of the generated mnemonic sentence (MS) in words.

ENT	CS = ENT / 32	MS = (ENT + CS) / 11
128	4	12
160	5	15
192	6	18
224	7	21
256	8	24

OPEN uses a 12-length mnemonic sentence to generate a seed.

From mnemonic to seed:

The mnemonic code represents 128 to 256 bits, which are used to derive a longer (512-bit) seed through the use of the key-stretching function PBKDF2. The resulting seed is used to create a deterministic wallet and all of its derived keys.

Examples:

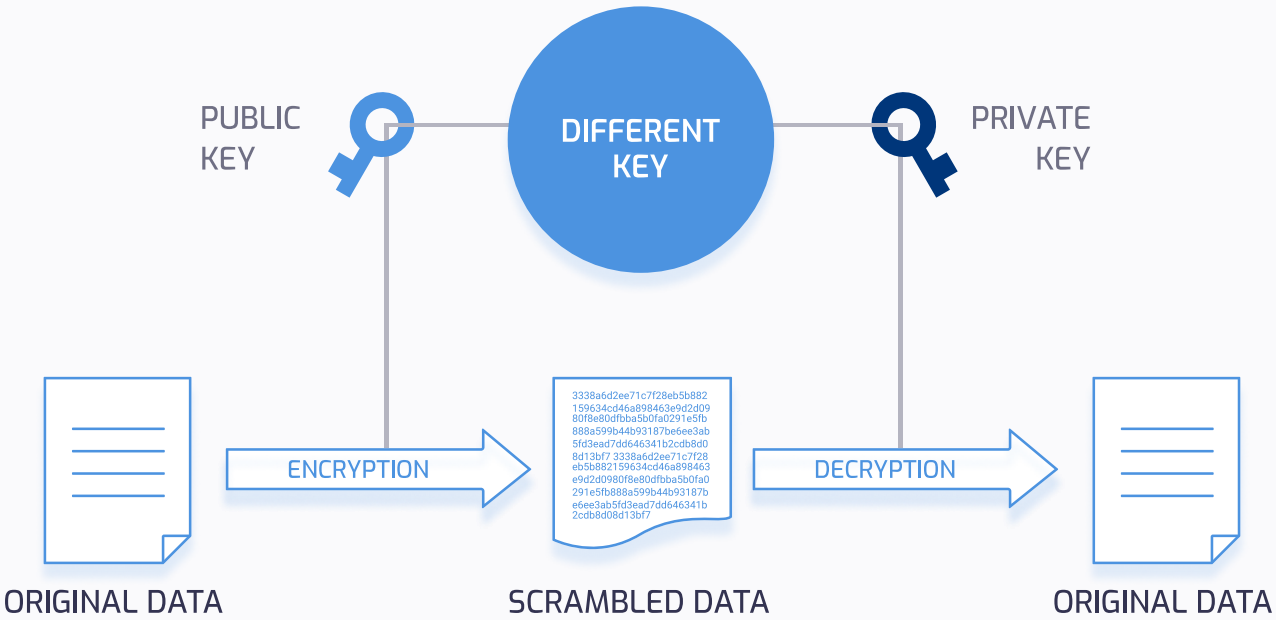
Entropy input (128 bits)	0c1e24e5917779d297e14d45f14e1a1a
Mnemonic (12 words)	army van defense carry jealous true garbage claim echo media make crunch
Seed (512 bits)	3338a6d2ee71c7f28eb5b882159634cd46a898463e9d2d0980f8e80dfbba5b0fa0 291e5fb888a599b44b93187be6ee3ab5fd3ead7dd646341b2cdb8d08d13bf7

Mnemonic code generation is described in BIP39 document

ASYMMETRIC CRYPTOGRAPHY

Asymmetric cryptography is known as public key cryptography. It allows information to be transferred through a public key that can be shared with anyone. Public and private keys both are mathematically related and have a specific role in the operation. Data that is encrypted with a private key can only be decrypted with a public key and vice versa. You cannot encrypt and decrypt data using the same key.

OPEN applies the Elliptic Curve Digital Signature Algorithm, or ECDSA, which is used to create public and private keys.



In OPEN, asymmetric cryptography is used to produce a digital signature of transactions and blocks.

ELLIPTIC-CURVE CRYPTOGRAPHY

Elliptic curve cryptography is a type of asymmetric cryptography based on the discrete logarithm problem as expressed by addition and multiplication on the points of an elliptic curve.

OPEN uses a specific elliptic curve and set of mathematical constants, as defined in a standard called secp256k1, that was established by the National Institute of Standards and Technology (NIST). The secp256k1 curve is defined by the following function, which produces an elliptic curve:

$$y^2 = mod\ p = (x^3+7)\ mod\ p$$

The mod p indicates that this curve is over a finite field of prime order p.

ELLIPTIC-CURVE CRYPTOGRAPHY

The private key is randomly generated from a seed phrase.

When a private key is multiplied by a predetermined point on the curve called **the generator point G**, another point somewhere else on the curve is produced. This is the corresponding public key K.

The generator point is specified as part of the secp256k1 standard and is always the same for all keys.

DIGITAL SIGNATURE

A digital signature is an encrypted electronic stamp used to authenticate digital data or information and securing sensitive information.
OPEN uses digital signatures to verify the identity of the person sending a transaction or a block.

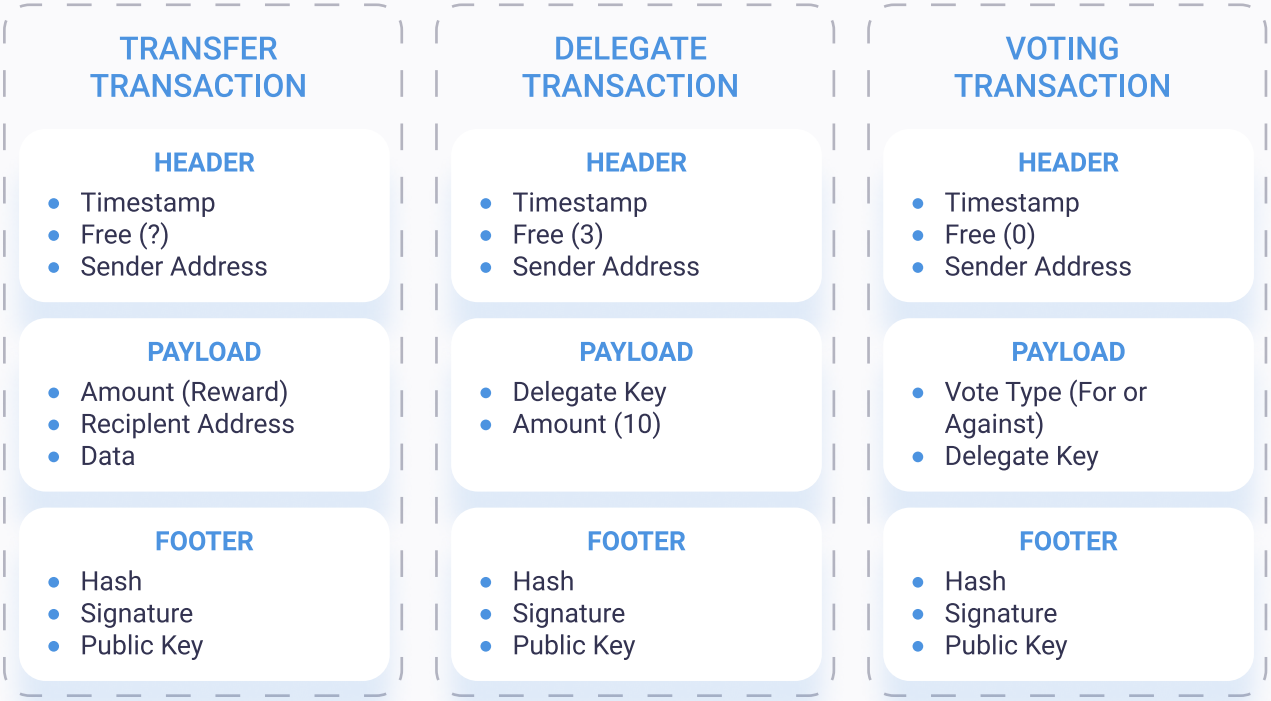
SIGNING TRANSACTION

To sign a transaction a sender encrypts transaction bytes with his private key. The public key of the sender as well as the resulting signature are included in the transaction.

Transaction bytes include:

- Timestamp
- Sender address
- Fee
- Amount

- Recipient address (transfer transaction)
- Delegate Key (delegate and vote transaction)
- Vote type (vote transaction)



SIGNING BLOCK

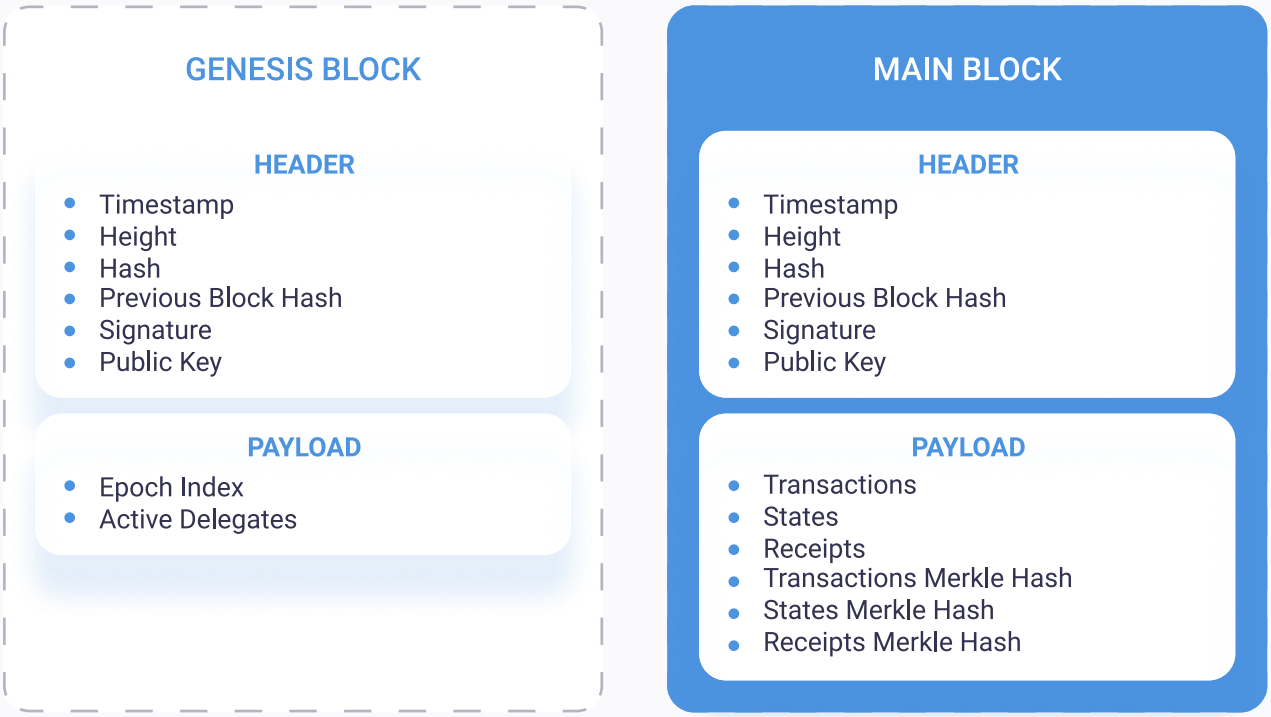
To sign a main block, an Active Delegate encrypts block bytes with his node's private key. Since a Genesis block is generated by each peer independently, the peer uses his node's private key to encrypt the block.
The signer keeps the public key of the node and the resulting signature in the block.

Main block bytes include:

- Timestamp
- Block height
- Previous block height
- Transactions Merkle Hash
- States Merkle Hash
- Receipts Merkle Hash

Genesis block bytes include:

- Timestamp
- Block height
- Previous block height
- Active delegates' public keys
- Epoch index



HASH FUNCTIONS

Hashing is the process of taking the input of any length and turning it into a cryptographic fixed output through a mathematical algorithm.
OPEN uses different types of hash functions for different purposes.

SHA256 is one of the strongest hash functions available. In OPEN it is used to generate a transaction hash, a block hash, a state, and a receipt hash by hashing their bytes. The resulting hash is 265-bit long.

Keccak256 is another hash function used by OPEN. Wallet address is a keccak256 hash of the public key.

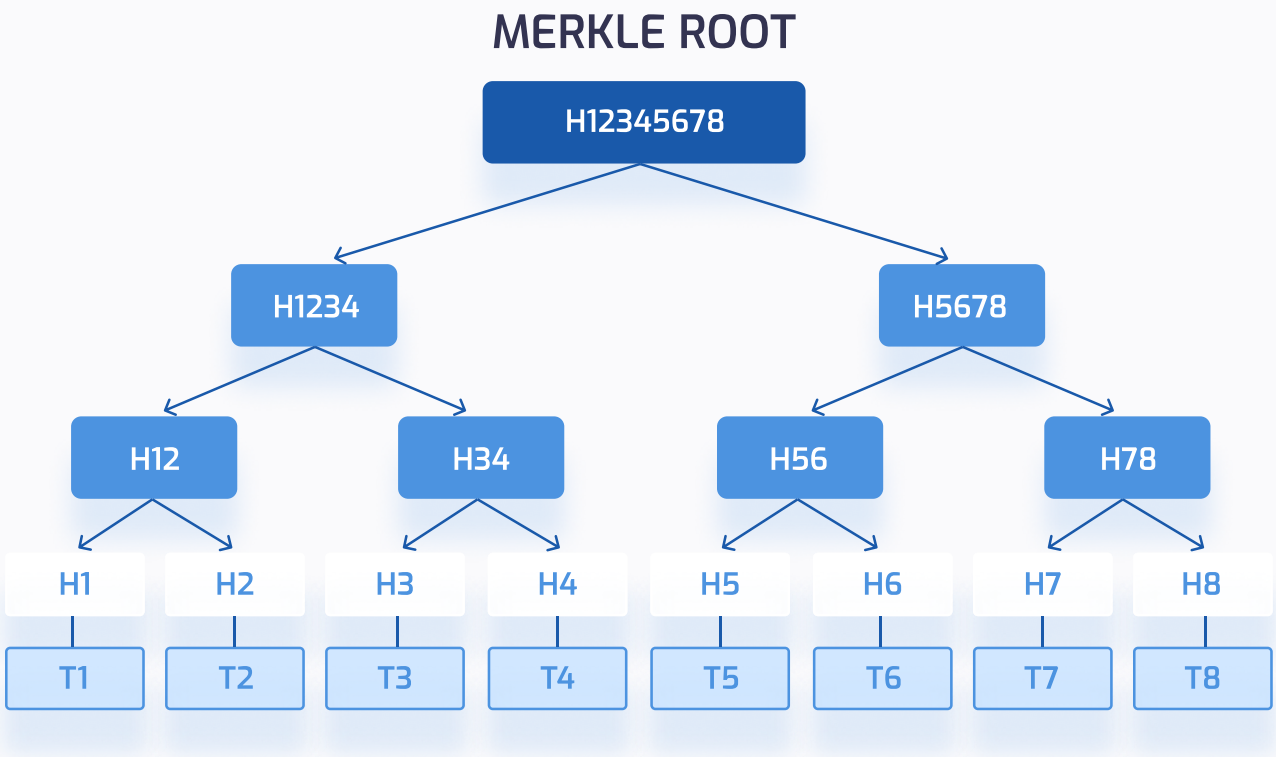
PBKDF2 is a key-stretching function with a sliding computational cost, aimed to reduce the vulnerability of encrypted keys to brute force attacks. PBKDF2 hash is used to generate a seed from a mnemonic sentence.

HMAC is a specific type of a message authentication code (MAC) which includes a cryptographic hash function and a secret cryptographic key. **HMACSHA512** function is used in the key derivation algorithm described in BIP32.

RIPEMD160SHA256 is also used for key derivation. This function presents a combination of RIPEMD160 and SHA256 hashes. RIPEMD160 is applied thanks to its ability to produce the shortest hashes the uniqueness of which is still sufficiently assured.

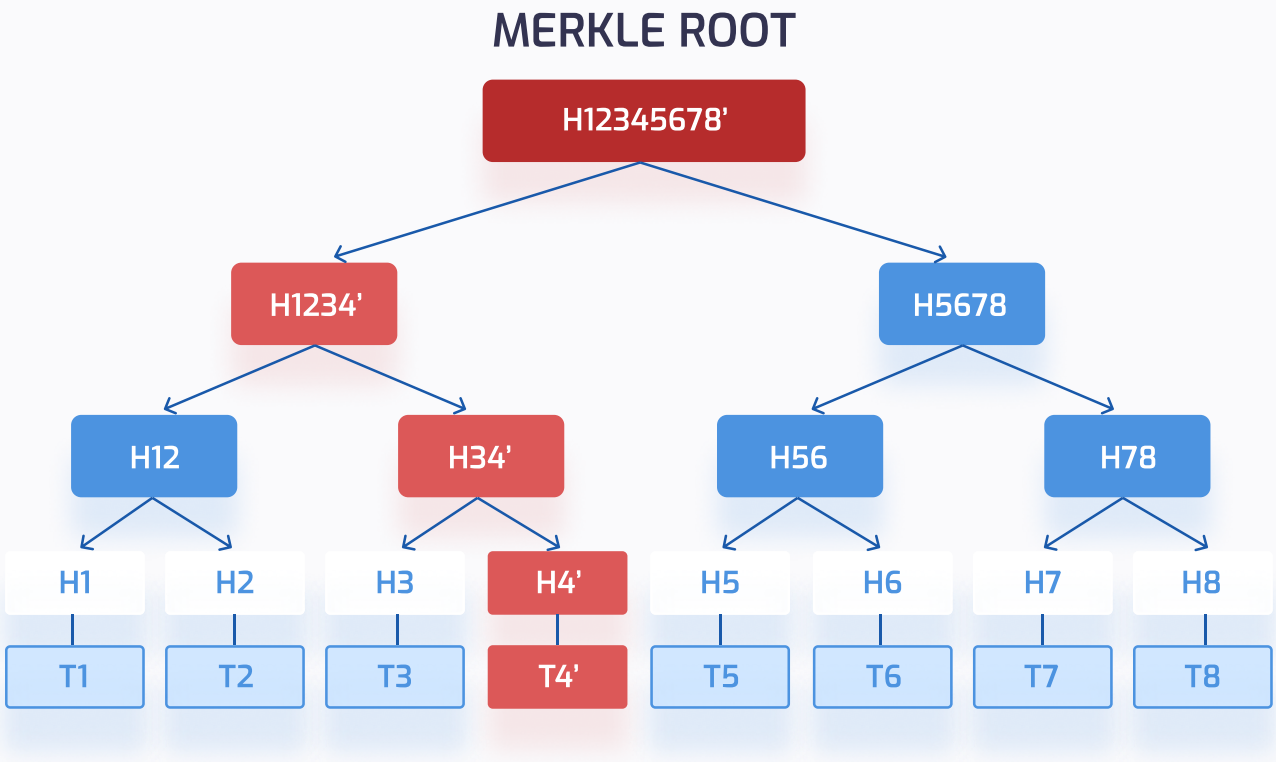
MERKLE ROOT

Merkle trees provide a cryptographically authenticated data structure. A Merkle tree is a tree-like structure in which every leaf node is a hash of a block of data and every non-leaf node is a hash of its children. All these elements are united in a single hash called the Merkle root.



Each OPEN main block has the transaction Merkle root, states Merkle root and receipt Merkle root. Using a Merkle tree provides the integrity and validity of transactions, states and receipts using their hashes that an active delegate has to maintain.

If an attacker changes any parameter in the transaction of the block, transaction Merkle root will change along with the nodes of the Merkle tree. The active delegate will consider this main block to be invalid.



NETWORK

SERIALIZATION PROTOCOL

OPEN Chain nodes communicate via the TCP transport protocol which presupposed a special format of exchanging message. Conventionally, a message has two parts:

- Message header
- Message body

MESSAGE HEADER

Each message strictly contains 3 fields:

- Version - indicates a protocol version. If a nodes protocol version is not equal to the one specified in a header this message will be ignored
- Time - timestamp, when a message was sent. There is a10-second expiry period, when a message is considered to be relevant. If it is transmitted after the allocated period expires, the message will be ignored
- Message Type - indicates type of message

MESSAGE BODY

The message body contains all required fields to create an instance and to handle it by node. Each field is decoded, according to its type. The type of a field pre-determines the amount of bytes that have to be read and stored. The list below includes the primitive types of fields and and the amount of bytes that should be read in each case:

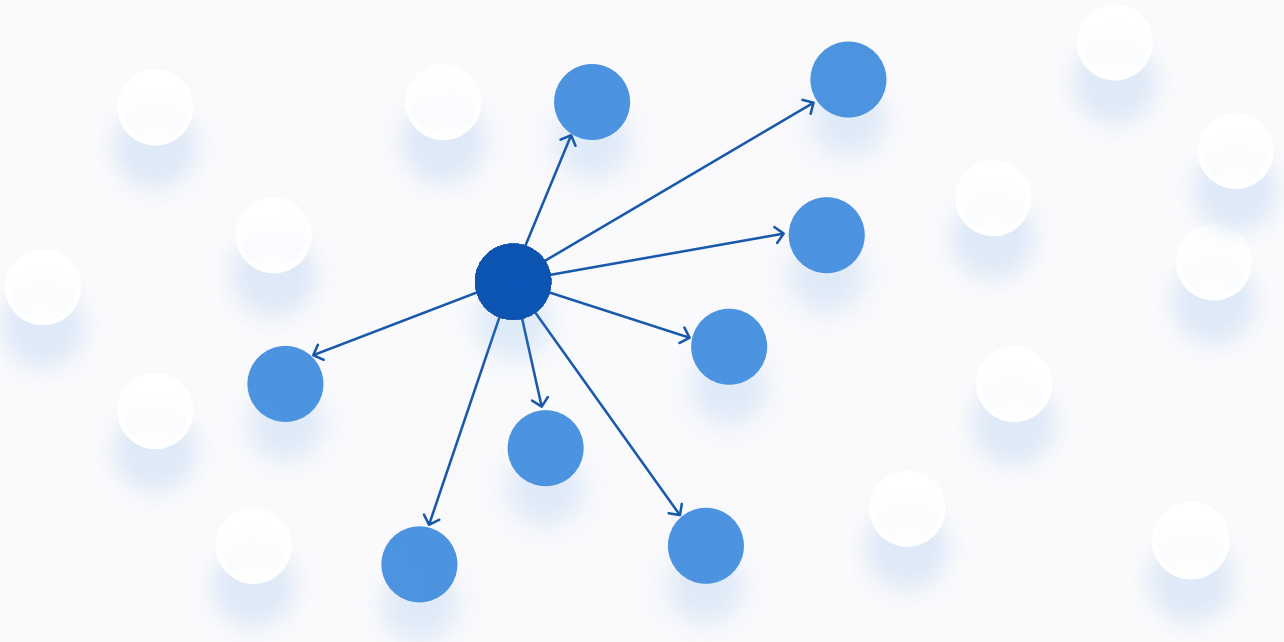
- Byte - 1 byte
- Short - 2 byte
- Integer - 4 byte
- Long - 8 byte
- Float - 4 byte
- Double - 8 byte
- Boolean - 1 byte
- Character - 2 byte

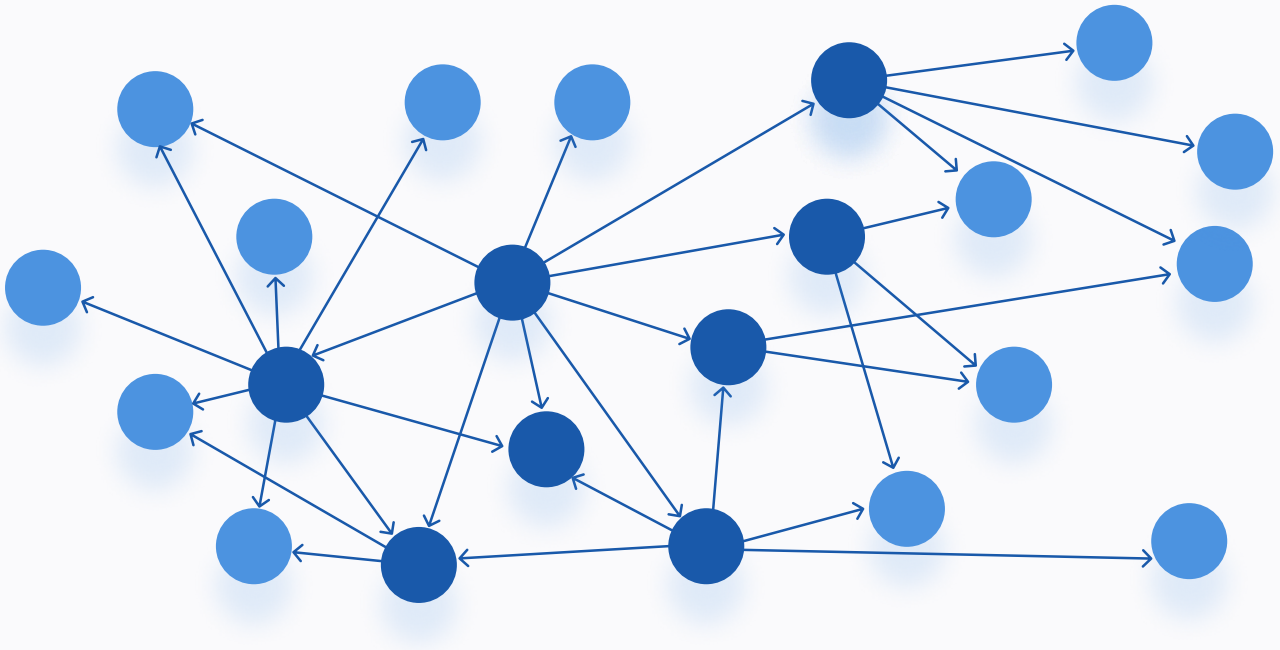
A protocol allows to serialize a string data type. To perform that, an Integer value should be serialized before the set of characters that describe the string's length. The same approach is applied to collections: the amount of elements should be serialized before the relevant content.

GOSSIP PROTOCOL

The gossip protocol describes an approach to build communication in a peer-to-peer network. Let's suppose that we have a peer, and this peer is connected to other nodes. And it sends messages to all other peers. In OPEN Chain node should obtain at least 4 connections to operate in the network (the maximum number of connections allowed is 16). Peers, that have received a message, handle it and pass further in the network. That is a principle according to which a message is broadcasted across that network. A message won't be passed further in two cases:

- If a node that accepted this message had already got such a message previously
- If a message is ignored due to the fact that its period has already expired





TIME SYNCHRONIZATION

For the application to work properly, a correctly synchronized system clock with an accuracy of 100 milliseconds is required.

When the application is being started, a ClockChecker requests time offset from the NTP servers, with a certain time interval. If the received offset is has valid values, the application runs in a normal mode, otherwise, if this parameter is not synchronized, the application will require to set the correct time. If it is not fulfilled, correct functioning is not guaranteed.

SMART CONTRACT

Smart contracts are account holding objects on the Open Platform blockchain. They contain code functions, take decisions, store data, and send ether to others. Contracts are defined by their creators, but their execution, and by extension the services they offer, is provided by the Open Platform network itself. They will exist and be executable as long as the whole network exists, and will disappear only in the case if they were programmed to self destruct.

What can you do with contracts? You can transfer money to other wallets, save different type of information and interact with it. But for our getting started guide let's do some simple things: To start you will create a classic "Hello World" contract, then you can build your own crypto token to send to whomever you like.

CREATION

Before you begin, prepare

1. any jvm environment
2. your patience and curiosity

WRITING A CONTRACT

Any smart contract must extend the superclass **“SmartContract”**, which has an abstract method **“execute”.**

The example will be in Kotlin:

```
class HelloWorld : SmartContract() {

    override fun execute() {
        println("Hello world")
    }

}
```

Then you need to compile the class and get bytecode.

VALIDATION

Before uploading smart contract to the blockchain the **Node** validates it in accordance with a number of restrictions.
You can validate it yourself. In order to do it, there is a special class **“SmartContractValidator”**, which has a method **“validate”**. This method will return **true** if bytecode is right and **false** if not.

Checking by this method runs with the usage of the following criteria:

```
Boolean::class.java.name,
Char::class.java.name,
Byte::class.java.name,
Short::class.java.name,
Int::class.java.name,
Long::class.java.name,
Float::class.java.name,
Double::class.java.name,
Void::class.javaPrimitiveType!!.name,

"java.lang.Boolean",
"java.lang.Character",
"java.lang.Byte",
"java.lang.Short",
"java.lang.Integer",
"java.lang.Long",
"java.lang.Float",
"java.lang.Double",
"java.lang.String",
"java.lang.StringBuilder",
"java.lang.StrictMath",

"java.util.ArrayList",
"java.util.Arrays",
"java.util.HashMap",
"java.util.HashSet"
```

There is a blacklist of the **Object** class methods used to stop a smart contract.

```
"java.lang.Object.getClass",
"java.lang.Object.wait",
"java.lang.Object.notify",
"java.lang.Object.notifyAll"
```

EVALUATION

The cost of the contract is estimated in the framework of the deployment phase. The estimate is based on bytecode, each opcode has its own cost.

Calculation formula

$$contractCost = (countOpcode_{m} * costOpcode_{m}) + ... + (countOpcode_{m+n} * costOpcode_{m+n})$$

LOADING

After all the checks we can begin uploading a smart contract to the blockchain. All you need to dois to insert a ready-made bytecode in the field of **Node** in the browser and send a transfer transaction.

Under the hood, everything is a little bit more complicated. Before you save it to the blockchain this smart contract name is changed in order to make it unique in the JVM. Then the **Node** uploads it to the JVM and initializes object default data. After this, the Node serializes a state and sends it to the blockchain.

EXECUTION

The execution of the contract is triggered when funds are received in the account of the contract. You can do this by sending a transfer transaction which presupposes specifying the address of the contract in the recipients.

Steps of execution:

1. Search for the contract and loading it from the DB into the JVM using its specific address.
2. Creation of the instance of the contract and initialization of the inner state.
3. Execution.
    a. Performance of some of the tasks that are written in it. For example: creation of transfer transactions to a specific address.
4. Saving execution results and transfer of funds to the delegate for execution.
    a. Saving all transfer actions in DB like Receipts.
    b. If the contract fails, it saves the cause of error in Receipt.

5. Serializing of the inner state into the DB if the contract is successfully executed.
6. Transfer of odd money back to sender`s wallet after a successful execution.
    a. If the contract`s wallet amount is not fully spent, it remains on the contract account.

STATE SYNCHRONIZATION

Each contract has an inner state that includes a number of variables, they all are stored in a serialized form in the DB.
Before the contract is executed, its state is loaded from the DB and pushed into the instance.
After a successful execution, the inner state of the contract is serialized and stored in the DB for the next executions. When execution fails, the state does not change.

REMOTE PROCEDURE CALL

The RPC is used to call REST-endpoints.

Endpoint	Description	Explorer	Wallet Client
/rpc/info	Get basic node information.	✓	
/rpc/info/getVersion	Get a node communication protocol version.	✓	
/rpc/info/getUptime	Get node up time.	✓	
/rpc/info/getHardwareInfo	Get node hardware info: CPU and RAM info, total storage size and network interfaces info.	✓	
/rpc/explorer/info	Get information about blockchain	✓	
/rpc/account/doGenerate	Generate a new account		✓
/rpc/account/doRespore	Restore an account with a seed phrase		✓
/rpc/accounts/wallets/{address}/balance	Get a wallet balance by the wallet address		✓
/rpc/accounts/wallets/{address}/delegates	Get wallet votes for delegates		✓
/rpc/accounts/wallets/validateAddress	Validate a wallet address		✓
/rpc/account/keys/doDerive	Create a new account by a derivation path.		✓
/rpc/accounts/keys/doPrivateImport	Get a public key and a wallet address for a private key.		✓
/rpc/accounts/keys/doExtendedImport	Restore an account with a private or public extended key.		✓
/rpc/accounts/keys/doWifImport	Restore an account with a private key the in WIF (Wallet import format) format.		✓

/rpc/blocks/genesis	Get a list of all genesis blocks		✓
/rpc/blocks/genesis/{hash}	Get a genesis block by hash		✓
/rpc/blocks/genesis/{hash}/previous	Get the previous genesis block by hash		✓
/rpc/blocks/genesis/{hash}/next	Get the next genesis block by hash		✓
/rpc/blocks/main	Get a list of all main blocks		✓
/rpc/blocks/main/{hash}	Get the main block by hash		✓
/rpc/blocks/main/{hash}/previous	Get the previous main block by hash		✓
/rpc/blocks/main/{hash}/next	Get the next main block by hash		✓
/rpc/transactions/delegate	Send a delegate transaction		✓
/rpc/transactions/delegate/{hash}	Get a delegate transaction by hash		✓
/rpc/transactions/reward	Get a list of all reward transactions		✓
/rpc/transactions/reward/{address}	Get a reward transaction by a recipient address		✓
/rpc/transactions/transfer	Get a list of transfer transactions		✓
/rpc/transactions/transfer	Send a transfer transaction		✓
/rpc/transactions/transfer/{hash}	Get a transfer transaction by hash		✓
/rpc/transactions/transfer/address/{address}	Get a transfer transaction by a wallet address		✓
/rpc/transactions/vote	Send a vote transaction		✓

/rpc/transactions/vote/{hash}	Get a vote transaction by hash		✓
/rpc/delegates	Get a list of all delegates		✓
/rpc/delegates/active	Get a list of all active delegates		✓
/rpc/delegates/view	Get a list of all delegates rating		✓
/rpc/transactions/{hash}/receipt	Get a receipt by hash of the transaction		✓
/rpc/contracts/estimation	Get a cost of execution/deployment of the contract		✓

These REST-endpoints are used to get current information about the efficiency of the node.

GET INFO

Get basic node information.

HTTP method: GET

Path: /rpc/info

Parameters: None

Response:


```
{
  "timestamp": 1533201403476,
  "version": "1.0.0",
  "payload": {
    "publicKey": "02a54008f5deea06a1bd9ec995ff458d6ffa235ba40be48034a8fab6873c805bfa",
    "host": "127.0.0.1",
    "port": 9190
  }
}
```

Response fields:

Attribute	Type	Description
publicKey	String	Node's public key
host	String	Node's external host
port	String	Node's external port

GET VERSION

Get a node communication protocol version.

Type: GET
Path: /rpc/info/getVersion
Parameters: None
Response:

```
{
    "timestamp": 1533201403476,
    "version": "1.0.0",
    "payload": null
}
```

GET UPTIME

Get node up time.

Type: GET
Path: /rpc/info/getUptime
Parameters: None
Response:

```
{
    "timestamp": 1533201463165,
    "version": "1.0.0",
    "payload": 4007529
}
```


Response fields:

Attribute	Type	Description
payload	Long	Node up time in millis

GET HARDWARE INFO

Get node hardware info: CPU and RAM info, total storage size and network interfaces info.

Type: GET

Path: /rpc/info/getHardwareInfo

Parameters:None

Response:

```
{
  "timestamp": 1533201525728,
  "version": "1.0.0",
  "payload": {
    "cpu": {
      "model": "158",
      "frequency": 3400000000,
      "numberOfCores": 4
    },
    "ram": {
      "free": 2727124992,
      "used": 13984133120,
      "total": 16711258112
    },
    "totalStorageSize": 250059350016,
    "networks": [
      {
        "interfaceName": "veth295f073",
        "addresses": [
          "fe80:0:0:0:d0:c9ff:feb2:4f15%veth295f073"
        ]
      },
      {
        "interfaceName": "br-23bcd0f6ba49",
        "addresses": [
          "fe80:0:0:0:42:14ff:fe98:3253%br-23bcd0f6ba49",
          "172.21.0.1"
        ]
      }
    ]
  }
}
```


Response fields:

Payload:

Attribute	Type	Description
cpu	CPUInfo	Node's CPU information
ram	RAMInfo	Node's CPU information
totalStorageSize	Long	Node's total storage size

CPUInfo:

Attribute	Type	Description
model	String	Processor model
frequency	Long	Processor frequency
numberOfCores	Int	Number of processor's cores

RamInfo:

Attribute	Type	Description
free	Long	Free memory space
used	Long	User memory space
total	Long	Total memory space

NetworkInfo:

Attribute	Type	Description
interfaceName	String	Interface name
addresses	String	List of IP addresses in IPv4 and IPv6 formats

GET BLOCKCHAIN INFO

Get information about blockchain

Type: GET

Path: /rpc/explorer/info

Parameters: None

Response:


```
{
 "timestamp": 1549891484737,
 "version": "1.8.0",
 "payload": {
  "nodesCount": 1,
  "blocksCount": 5435,
  "transactionsCount": 4529,
  "blockProductionTime": 2487,
  "transactionsPerSecond": 0,
  "currentEpochNumber": 906,
  "currentEpochDate": 1549891508021,
  "delegatesCount": 5}
}
```


Response fields:

Attribute	Type	Description
nodesCount	Int	Countrunning nodes
nodesCount	Long	Count created blocks
transactionsCount	Long	Count created transactions
blockProductionTime	Long	Average block production time in millis
transactionsPerSecond	Long	How many transactions are created per second
currentEpochDate	Long	Current epoch date in millis
delegatesCount	Byte	Delegate count dedicated to create and approve blocks

DO GENERATE

Generate a new account

Type: GET

Path: /rpc/account/doGenerate

Parameters: None

Response:


```
{
   "timestamp": 1533200140353,
   "version": "1.0.0",
   "payload": {
      "seedPhrase": "recipe provide own axis bean exotic grocery unit flock barrel bike erosion",
      "masterKeys": {
         "publicKey": "02a54008f5deea06a1bd9ec995ff458d6ffa235ba40be48034a8fab6873c805bfa",
         "privateKey": "bf9d87c65ded8399ce2d4835d386698ae28aaa337c4893956bd867303860347c"
      },
      "defaultWallet": {
         "keys": {
            "publicKey": "0342bf5912bb2dc42e0485a7ef3d3aa054888b04999cd9bc927ff309cfaca8d55d",
            "privateKey": "db962381653a277b5bb5f2b2e76722948a1d361b22e13237c6eb45c889a2eed8"
         },
         "address": "0x62999a450A583A238BFE3B1d50293eb8C74847fD"
      }
   }
}
```


Response fields:

Payload:

Attribute	Type	Description
seedPhrase	String	Seed phrase of 12 words. That is key to restore an account, it should be kept in a secret
masterKeys	KeyDto	Pair of master keys. Keys are serialized
defaultWallet	WalletDto	Default account information that includes a pair of keys and an address. The private key must be kept in a secret. The derivation path of the default account is m/0/0/0

KeyDto:

Attribute	Type	Description
publicKey	String	Public key
privateKey	String	Private key. It must be kept in a secret

WalletDto:

Attribute	Type	Description
keys	KeyDto	Pair of account keys
address	String	Wallet address. It starts with 0x

DO RESTORE

Restore an account with a seed phrase

Type: POST

Path: /rpc/account/doRespore

Parameters: seedPhrase

Attribute	Type	Description
seedPhrase	String	Seed phrase of 12 words

Response:


```
{
   "timestamp": 1533201172635,
   "version": "1.0.0",
   "payload": {
      "seedPhrase": "multiply fault butter script extend exotic luxury ocean matter session version play",
      "masterKeys": {
         "publicKey": "02bb9808c494ff0d7cbf2621d8ba5687819b69a4b9ef64267033369174e56a56a9",
         "privateKey": "15a0dc46df5e4f7285ec8ce1e40c665eddae0b2d8c766b1d3d362889e3a6ee1f"
      },
      "defaultWallet": {
         "keys": {
            "publicKey": "02b28915709de8260a529155fb83bc487fe076d933f864b3e37f953fd8d0cbfd20",
            "privateKey": "7589651497f5dc605be8782d4e21fc63b2b0652c4b99e6790c33f14e0bf9bda3"
         },
         "address": "0x8A1D90a716DB145ef5677553fAc096608416eEE9"
      }
   }
}
```


Response fields:

Payload:

Attribute	Type	Description
seedPhrase	String	Seed phrase of 12 words. That is a key to restore an account. It must be kept in a secret
masterKeys	KeyDto	Pair of master keys. Keys are serialized
defaultWallet	WalletDto	Default account information that includes a pair of keys and an address. The private key must be kept in a secret. The derivation path of the default account is m/0/0/0

GET WALLET BALANCE

Get a wallet balance by a wallet address.

Type: GET

Path: /rpc/accounts/wallets/{address}/balance

Parameters: address

Attribute	Type	Description
address	**String**	**Address of a wallet the balance of which should be received**

Response:

```
{
  "timestamp": 1533200378781,
  "version": "1.0.0",
  "payload": 1500
}
```

GET WALLET VOTES FOR DELEGATES

Get wallet votes for delegates

Type: GET

Path: /rpc/accounts/wallets/{address}/delegates

Parameters: address - wallet address (PageRequest supported)

Response:


```
{
  "timestamp": 1536926823153,
  "version": "1.1.0",
  "payload": {
    "totalCount": 22,
    "list": [
      {
        "address": "0x51c5311F25206De4A9C6ecAa1Bc2Be257B0bA1fb",
        "delegateKey": "02c4aedc4a7e2d8cc0e73e6dfb428e8555adc8a1b74207cb61143babd3e04be63f",
        "rating": 346,
        "votesCount": 1,
        "timestamp": 1532345018021,
        "recalled": true
      }
    ]
  }
}
```


Response fields:

Payload:

Attribute	Type	Description
totalCount	Long	Total votes for delegates count
list	VoteDelegateDto[]	List of vote info

VoteDelegateDto:

Attribute	Type	Description
address	String	Wallet address. It starts with 0x
delegateKey	String	Delegate's public key
rating	Long	Delegate's rating

votesCount	Long	Amount of votes for a delegate
timestamp	Long	Delegate registration time in mills
recalled	Boolean	Flag if a vote was recalled

VALIDATE ADDRESS

Validate a wallet address

Type: POST

Path: /rpc/accounts/wallets/validateAddress

Parameters: address

Response:

```
{
  "timestamp": 1533200378781,
  "version": "1.0.0",
  "payload": {
     "address": "0xD6A196f6E1387343f9939C1989993bAd3273FE28"
  }
}
```

200 OK if valid address

400 Bad request if invalid address

DO DERIVE

Create a new account by a derivation path.

Type: POST

Path: /rpc/account/keys/doDerive

Parameters: seedPhrase, derivationPath

Attribute	Type	Description
seedPhrase	String	A seed phrase of 12 words
derivationPath	String	A path to derive an account

Derivation path
Path - m/*/*/*.

Starts with 'm'
First *: account index
Second *: chain index (0 - external chain, 1 - internal chain)
Third *: wallet account index (default is 0)

The first generated account is derived by m/0/0/0 path. Next wallet is m/0/0/1, second - m/0/0/2, etc.
An external chain is used to spend and receive funds from others (m/0/0/0).
An internal chain can be used to generate an address for receiving changes (m/0/1/0).
To divide your wallets into groups (e.g. personal, work) use the second parameter. E.g. m/0/0/0 - personal account,
m/1/0/0 - work account
Response:


```
{
   "timestamp": 1533201254959,
   "version": "1.0.0",
   "payload": {
      "keys": {
         "publicKey": "02f4cafe456378101d7f8660dda0fa2a3811b183707510030ee36a2e744dd57e77",
         "privateKey": "99c9810b8a84ee4234d01acd8edd866a345f286a7d099c73cdb2937adf54a0c8"
      },
      "address": "0xC2d5a01Cc22295fF4cC49dB5A0013cE911D9A5cb"
   }
}
```

Response fields:

Attribute	Type	Description
publicKey	String	A public key generated by a private key
privateKey	String	The same private key that was sent to the node
address	String	A wallet address generated by a private/public key pair

DO PRIVATE IMPORT

Get a public key and a wallet address for a private key.

Type: POST

Path: /rpc/accounts/keys/doPrivateImport

Parameters: decodedKey

Attribute	Type	Description
decodedKey	String	A private key generated by a seed phrase

Response:


```
{
 "timestamp": 1549895244810,
 "version": "1.8.0",
 "payload": {
  "keys": {
   "publicKey": "02b942e77b68f7ce8c703ff53e09f5e0b75b65c8501b5f46aa0cac073c1bbed5ba",
   "privateKey": "64ce8d8ccd824ffa36e2f85b858cae539a40f7e91a36e5cdee89ed6e8aca2225"
  },
  "address": "0x66a1d6ef9ADf66E4Acd228a0E01ccbB9cEC054bF"
 }
}
```


Response fields:

Attribute	Type	Description
publicKey	String	A public key generated by a private key
privateKey	String	The same private key that was sent to the node
address	String	A wallet address generated by a private/public key pair

EXTENDED IMPORT

Restore an account with a private or public extended key.

Type: POST

Path: /rpc/accounts/keys/doExtendedImport

Parameters: decodedKey

Attribute	Type	Description
decodedKey	String	A serialized private or public key. When a public key is being imported, a private key isn't generated

Response:


```
{
  "timestamp": 1549895244810,
  "version": "1.8.0",
  "payload": {
    "keys": {
      "publicKey": "02b942e77b68f7ce8c703ff53e09f5e0b75b65c8501b5f46aa0cac073c1bbed5ba",
      "privateKey": "64ce8d8ccd824ffa36e2f85b858cae539a40f7e91a36e5cdee89ed6e8aca2225"
    },
    "address": "0x66a1d6ef9ADf66E4Acd228a0E01ccbB9cEC054bF"
  }
}
```


Response fields:

Attribute	Type	Description
publicKey	String	A public key generated by a private key
privateKey	String	The same private key that was sent to the node
address	String	Wallet address. It starts with 0x

DO KEY IMPORT IN WIF FORMAT

Restore an account with a private key in the WIF (Wallet import format) format.

Type: POST

Path: /rpc/accounts/keys/doWifImport

Parameters: decodedKey

Attribute	Type	Description
decodedKey	String	A private key in the WIF format

Response:


```
{
  "timestamp": 1533201349362,
  "version": "1.0.0",
  "payload": {
    "keys": {
      "publicKey": "02243584bd5c6a38278f2ef7018064356611e24cff8a4a4d0702424383bc79d182",
      "privateKey": "5528ea30ee9152e5a026722e8373bae3addb3cd97fdfe7274948b77a2df8cee8"
    },
    "address": "0xb23cDAb48FACF2B5b2Aa57Cd4DFe30fCBAdc9d79"
  }
}
```


Response fields:

Attribute	Type	Description
publicKey	String	A public key generated by a private key
privateKey	String	The same private key that was sent to the node
address	String	Wallet address. It starts with 0x

GET ALL OF GENESIS BLOCKS

Get a list of all genesis blocks of the blockchain.

Type: GET
Path: /rpc/blocks/genesis
Parameters: PageRequest is supported

The Dto of genesis block includes the following fields:

Attribute	Type	Description
timestamp	Long	Block creation time in mullis
height	Long	Block's height
previousHash	String	Previous block hash
reward	Long	Reward for block creation
hash	String	Block hash
signature	String	Creator's signature
publicKey	String	Creator's public key
epochIndex	Long	Epoch index when a block was created
delegatesCount	Long	Number of delegates in block

Response:


```
{
  "timestamp": 1535016027080,
  "version": "1.0.0",
  "payload": {
    "totalCount": 1,
    "list": [
      {
        "timestamp": 1532345018021,
        "height": 1,
        "previousHash": "",
        "reward": 0,
        "hash": "838c84179c7e644cdf2ff0af3055ed45c6f43e0bd7634f8bd6ae7d088b1aaf0a",
        "signature": "MEUCIQCLeQuqCmS037ZfmQNtpUNNli7QIgKNdhszih/PezHkW52q3=",
        "publicKey": "038bbbeeb867b9999934g3f334rf2g3449d1cc7208e32190184b13aaf1b",
        "epochIndex": 1,
        "delegatesCount": 0
      }
    ]
  }
}
```


Response fields:

Payload:

Attribute	Type	Description
totalCount	Long	Total genesis blocks count
list	GenesisBlockDto[]	List of genesis block info

GET GENESIS BLOCK

Get a genesis block by hash

Type: GET

Path: /rpc/blocks/genesis/{hash}

Parameters: hash

Attribute	Type	Description
hash	**String**	**Hash of current block**

Response:


```
{
  "timestamp": 1534841188529,
  "version": "1.0.0",
  "payload": {
    "timestamp": 1532345018021,
    "height": 1,
    "previousHash": "",
    "reward": 0,
    "hash": "838c84179c7e644cdf2ff0af3055ed45c6f43e0bd7634f8bd6ae7d088b1aaf0a",
    "signature": "MEUCIQCLeQuqCrDd8nzih/PezHW52v4/tdsZxaLo4g45vJzDnLvUy98tnsgg=",
    "publicKey": "038bbbeeb867b999991cd5b146bht4g4g45g1cc72g408e32190184b13aaf1b",
    "epochIndex": 1,
    "delegatesCount": 0}
}
```

GET PREVIOUS GENESIS BLOCK

Get the previous genesis block by hash

Type: GET

Path: /rpc/blocks/genesis/{hash}/previous

Parameters: hash

Attribute	Type	Description
hash	**String**	**Hash of current block**

Response:


```
{
 "timestamp": 1534841188529,
 "version": "1.0.0",
 "payload": {
   "timestamp": 1532345018021,
   "height": 1,
   "previousHash": "",
   "reward": 0,
   "hash": "838c84179c7e644cdf2ff0af3055ed45c6f43e0bd7634f8bd6ae7d088b1aaf0a",
   "signature": "MEUCIQCLeQuqCrDd8nzih/PezHW52v4/tdsZxaLo4g45vJzDnLvUy98tnsgg=",
   "publicKey": "038bbbeeb867b999991cd5b146bht4g4g45g1cc72g408e32190184b13aaf1b",
   "epochIndex": 1,
   "delegatesCount": 0}
}
```

GET NEXT GENESIS BLOCK

Get the next genesis block by hash

Type: GET

Path: /rpc/blocks/genesis/{hash}/previous

Parameters: hash

Attribute	Type	Description
hash	**String**	**Hash of current block**

Response:


```
{
 "timestamp": 1534841188529,
 "version": "1.0.0",
 "payload": {
   "timestamp": 1532345018021,
   "height": 1,
   "previousHash": "",
   "reward": 0,
   "hash": "838c84179c7e644cdf2ff0af3055ed45c6f43e0bd7634f8bd6ae7d088b1aaf0a",
   "signature": "MEUCIQCLeQuqCrDd8nzih/PezHW52v4/tdsZxaLo4g45vJzDnLvUy98tnsgg=",
   "publicKey": "038bbbeeb867b999991cd5b146bht4g4g45g1cc72g408e32190184b13aaf1b",
   "epochIndex": 1,
   "delegatesCount": 0}
}
```

GET ALL OF MAIN BLOCKS

Get a list of all main blocks

Type: GET

Path: /rpc/blocks/main

Parameters: PageRequest is supported

Response:


```
{
  "timestamp": 1535015838153,
  "version": "1.0.0",
  "payload": {
    "totalCount": 2,
    "list": [
      {
        "timestamp": 1533813800085,
        "height": 2,
        "previousHash": "838c84179c7e644cdf2ff0af3drhehe3e0bd7634f8bd6ae7d088b1aaf0a",
        "reward": 11,
        "hash": "77bbdc1af09c33c8d6f259dd2085dc8a75ec474d807b500800d5fa3e2358ebd9",
        "signature": "MEUCIEosVTTs8oU/yP58hziUt00od1ggelgtbgrewbav23b7fBt2rz/Tuj5af0=",
        "publicKey": "0278172a4763f9b8f73gre43gf3fg30fd7b151db5b1022760ec21802ff61fbff",
        "merkleHash": "989d4c7126fa30778clket4gw4g4g8ee731637f4d0ed23e8fd121ae540f2",
        "transactionsCount": 1,
        "epochIndex": 1
      },

{
        "timestamp": 1533813836054,
        "height": 3,
        "previousHash": "77bbdc1af09c33c8d6f259c8a75ec474d807b500800d5fa3e2358ebd9",
        "reward": 11,
        "hash": "2f9d98c2f26e01cebb733dded021ba81ffbda4fda6ec9633fc773498db56bb60",
        "signature": "MEQCIFn4Nf1okI5Kf5e7mhH5nw6rEqWeSA1dszal50+M0/JwRckv3w==",
        "publicKey": "03bd2e25db207c727c63f568d3d05e333b45ebaaf68777ede03594155191",
        "merkleHash": "5132b6ffe0da1a802a8d502b7d5ba30bac7ad8e7626a0d83e6cdd83e",
        "transactionsCount": 1,
        "epochIndex": 1}
    ]
  }
}
```

Response fields:

Payload:

Attribute	Type	Description
totalCount	Long	Total main blocks count
list	MainBlockDto[]	List of main block info

MainBlockDto:

Attribute	Type	Description
timestamp	Long	Block creation time in mullis
height	Long	Block's height
previousHash	String	Previous block hash
reward	Long	Reward for block creation
hash	String	Block hash
signature	String	Creator's signature
publicKey	String	Creator's public key
merkleHash	String	Block's merkle hash
transactionsCount	Long	Number of transactions in block
epochIndex	Long	Epoch index when a block was created

GET MAIN BLOCK

Get a main block by hash

Type: GET

Path: /rpc/blocks/main/{hash}

Parameters: hash

Attribute	Type	Description
hash	**String**	**Hash of current block**

Response:


```
{
 "timestamp": 1534841080060,
 "version": "1.0.0",
 "payload": {
  "timestamp": 1533813800085,
  "height": 2,
  "previousHash": "838c84179c7e644cdf2ff0af3055ed45c6fd7634f8bd6ae7d088b1aaf0a",
  "reward": 11,
  "hash": "77bbdc1af09c33c8d6f259dd2085dc8a75ec474d807b500800d5fa3e2358ebd9",
  "signature": "MEUCIEo58hziUt00hEA6Wtdtmf/Uo2J2po5Ut5Hb67av23b7fBt2rz/Tuj5af0=",
  "publicKey": "0278172a4763f9b8f73ee4ae0d24c00151db5b1022760ec21802ff61fbff",
  "merkleHash": "989d4c7126fa30778c31731637hjlkykiyyf4d0ed23e8fd121ae540f2",
  "transactionsCount": 1,
  "epochIndex": 1}
}
```

GET PREVIOUS MAIN BLOCK

Get the previous main block by hash

Type: GET

Path: /rpc/blocks/main/{hash}/previous

Parameters: hash

Attribute	Type	Description
hash	**String**	**Hash of current block**

Response:


```
 {
  "timestamp": 163461080060,
  "version": "1.0.0",
  "payload": {
    "timestamp": 153456630085,
    "height": 3,
    "previousHash": "838c84179c7e644cdf2ff0af3055ed45c6fd7634f8bd6ae7d088b1aaf0a",
    "reward": 11,
    "hash": "fdw3453wb2rbvqeert22t34523d23d23d2a75ec474d807b500800d5fa3e2358ebd9",
    "signature": "MEreg3Ffo58hziUtO0hEA6Wtdtmf/Uo2J2po5Ut5Hb67av23b7fBt2rz/Tuj5af0=",
    "publicKey": "4565472a4763f9b8f73ee4ae0d24c00151db5b1022760ec21802ff61fbff",
    "merkleHash": "989546745g5547547778c31731637hjlkykiyyf4d0ed23e8fd121ae540f2",
    "transactionsCount": 1,
    "epochIndex": 1}
}
```

GET NEXT MAIN BLOCK

Get the next main block by hash

Type: GET

Path: /rpc/blocks/main/{hash}/next

Parameters: hash

Attribute	Type	Description
hash	**String**	**Hash of current block**

Response:


```
{
 "timestamp": 1534841080060,
 "version": "1.0.0",
 "payload": {
  "timestamp": 1533813800085,
  "height": 2,
  "previousHash": "838c84179c7e644cdf2ff0af3055ed45c6fd7634f8bd6ae7d088b1aaf0a",
  "reward": 11,
  "hash": "77bbdc1af09c33c8d6f259dd2085dc8a75ec474d807b500800d5fa3e2358ebd9",
  "signature": "MEUCIEo58hziUt00hEA6Wtdtmf/Uo2J2po5Ut5Hb67av23b7fBt2rz/Tuj5af0=",
  "publicKey": "0278172a4763f9b8f73ee4ae0d24c00151db5b1022760ec21802ff61fbff",
  "merkleHash": "989d4c7126fa30778c31731637hjlkykiyyf4d0ed23e8fd121ae540f2",
  "transactionsCount": 1,
  "epochIndex": 1}
}
```

SEND DELEGATE TRANSACTION

Send a delegate transaction

Type: POST

Path: /rpc/transactions/delegates

Request:


```
{
  "timestamp": 1535983691015,
  "fee": 20,
  "senderAddress": "0x51c5311F25206De4A9C6ecAa1Bc2Be257B0bA1fb",
  "amount": 0,
  "delegateKey": "032f68d94d60b3295715515b8dab106ac452f622ce922d40d0c53",
  "hash": "31b2b36d5f9ff34d3345ada2ecaf39821067a50b1b23ad9b250b09368973dd59",
  "senderSignature": "MEUCIQDcIGhhsBQlwDYCWEjvwGgerABqA7s6CVMVduWPn3cc=",
  "senderPublicKey": "032f68d94d60b329bee515515b8dab106ac452f622ce922d40d0c53",
}
```


Request fields:

Attribute	Type	Description
fee	Long	Transaction fee
delegateKey	String	Delegate's public key
senderAddress	String	Sender's address
amount	Long	Transaction amount
hash	String	Transaction hash
senderSignature	String	Sender's signature
senderPublicKey	String	Sender's public key

Response:


```
{
  "timestamp": 1535983691132,
  "version": "1.0.0",
  "payload": {
    "timestamp": 1535983691015,
    "fee": 3,
    "senderAddress": "0x51c5311F25206De4A9C6ecAa1Bc2Be257B0bA1fb",
    "senderSignature": "MEUCIQDcIGhhsBQlHuCWr5gwABqA7s6CVMVduWPn3cc=",
    "senderPublicKey": "032f68d94d60b329bee59550bf965bac452f622ce922d40d0c53",
    "hash": "31b2b36d5f9ff34d3345ada2ecaf39821067a50b1b23ad9b250b09368973dd59",
    "delegateKey": "032f68d94d60e59550bf96515b8dab106ac452f622ce922d40d0c53",
    "amount": 10,
    "blockHash": null }
}
```


Response fields:

Attribute	Type	Description
timestamp	Long	Transaction creation time in millis
fee	Long	Transaction fee
delegateKey	String	Delegate's public key
senderPublicKey	String	Sender's public key
senderAddress	String	Sender's address
senderSignature	String	Sender's signature
hash	String	Transaction hash
amount	Long	Transaction amount
blockHash	String	Block hash

GET DELEGATE TRANSACTION

Get a delegate transaction by hash

Type: GET

Path: /rpc/transactions/delegates/{hash}

Parameters: hash

Attribute	Type	Description
hash	String	Hash of the current delegate transaction

Response:


```
{
  "timestamp": 1533201843817,
  "version": "1.0.0",
  "payload": {
    "timestamp": 1535983691015,
    "fee": 3,
    "senderAddress": "0x51c5311F25206De4A9C6ecAa1Bc2Be257B0bA1fb",
    "senderSignature": "MEUCIQDclGhhsBQlHua7N2wGger5gwABqA7sVMVduWPn3cc=",
    "senderPublicKey": "032f68d94d60b329bee595515b8da22ce922d40d0c53",
    "hash": "31b2b36d5f9ff34d3345ada2ecaf39821067a50b1b23ad9b250b09368973dd59",
    "delegateKey": "032f68d94d60b329bee5955515b8dab106ac452f622ce922d40d0c53",
    "amount": 10,
    "blockHash": "66dd73d1a54c4f2d69f544178ebd137199db9450f82fbd49f64c5f3cc" }
}
```


Response fields:

Attribute	Type	Description
timestamp	Long	Transaction creation time in millis
fee	Long	Transaction fee
delegateKey	String	Delegate's public key

senderPublicKey	String	Sender's public key
senderAddress	String	Sender's address
senderSignature	String	Sender's signature
hash	String	Transaction hash
amount	Long	Transaction amount
blockHash	String	Block hash

GET ALL REWARD TRANSACTIONS

Get a list of all reward transactions

Type: GET

Path: /rpc/transactions/reward

Parameters: PageRequels is supported

Response:


```
{
    "timestamp": 1550660665058,
    "version": "1.8.0",
    "payload": {
        "totalCount": 32,
        "list": [
            {
                "timestamp": 1550656964072,
                "fee": 0,
                "senderAddress": "0x0000000000000000000000000000000000000000",
                "hash": "8d5507492b08140b13f6435f8dc1f9e968ac062789d9ede0c85f",
                "signature": "MEUCIQCjHjWSlo/sIkkrtgrtyz4y4gtrvrRGR3434vfiQo=",
                "publicKey": "02b04aa1832e7a1cdbb737c167fc829472c726a12ad9a4ccf24eb",
                "payload": {
                    "reward": 10,
                    "recipientAddress": "0x51c5311F25206De4A9C6ecAa1Bc2Be257B0bA1fb"
                },
                "block": {...},
                "id": 1,
                "bytes": " "
            }
        ]
    }
}
```


GET REWARD TRANSACTION

Get a reward transaction by a recipient address

Type: GET

Path: /rpc/transactions/reward/{address}

Parameters: address

Attribute	Type	Description
address	String	Recipient address to get transaction for

Response:


```
{
"timestamp": 1550661020317,
"version": "1.8.0",
"payload": [{
"timestamp": 1550656964072,
        "fee": 0,
        "senderAddress": "0x0000000000000000000000000000000000000000",
        "hash": "8d5507492b08140b13f64353f8d1f9e968ac062789d9ede0c85f",
        "signature": "MEUCIQCjHjWSlo/sIkkBGArrCGbjDwl3T/jiYJiQo=",
        "publicKey": "02b04aa18300a6b8da1cdbb737c167fc829472c726a12ad9a4ccf24eb",
        "payload": {
          "reward": 10,
          "recipientAddress": "0x51c5311F25206De4A9C6ecAa1Bc2Be257B0bA1fb"
        },
        "block": {
          "timestamp": 1550656964072,
          "height": 2,
          "previousHash": "e53bc1dc7389bf8f970b029ef118ad9cc844a86e654b5d5ed611",
          "hash": "fcd1e25f662828525f8dedce617d651669dd8a378f449274516289b",
          "signature": "MEUCIQCcP1tyUv0oVsEpyrl4WsRIJv5Aj8vDIEDVWN9t/231gwt4=",
          "publicKey": "02b04aa1832e799503000a6b8c829472c726a12ad9a4ccf24eb",
          "payload": {
            "transactionMerkleHash": "8d5507492b08140ff9e968ac062789d9ede0c85f",
            "stateMerkleHash": "f9988b959c58e87bb445fabe4b04200754c",
            "receiptMerkleHash": "7a5d82b70289761f0cceb54db08f2d65b85d6c6a",
            "rewardTransactions": [],
            "voteTransactions": [],
            "delegateTransactions": [],
            "transferTransactions": [],
            "delegateStates": [],
            "accountStates": [],
            "receipts": []
          },
          "id": 2,
          "bytes": ""
        },
        "id": 1,
        "bytes": ""
}]
}
```

GET ALL TRANSFER TRANSACTIONS

Get a list of all transfer transactions

Type: GET

Path: /rpc/transactions/transfer

Parameters: PageRequest is supported

Response:


```
{
  "timestamp": 1536059448300,
  "version": "1.0.0",
  "payload": {
    "totalCount": 2,
    "list": [
      {
        "timestamp": 1435203843513,
        "fee": 20,
        "amount" : 1000,
        "recipientAddress": "0xC2d5a01Cc22295fF4cC49dB5A0013cE911D9A5cb",
        "senderPublicKey": "02b28915709de8260a529155fb864b3e37f953fd8d0cbfd20",
        "senderAddress": "0x8A1D90a716DB145ef5677553fAc096608416eEE9",
        "senderSignature": "02f4cafe456378101d7f8660dd07510030ee36a2e744dd57e77",
        "hash": "6709e3516a51fac52a6aa78a63528017147eb2ac56b892011ab9b",
        "blockHash": "ab2c266ab39405d0d75e3ab5abc55a3f14b435cb",
        "data": null
      }
    ]
  }
}
```


Response fields:

Payload:

Attribute	Type	Description
totalCount	Long	Total transfer transactions count
list	TransferTransactionDto[]	List of transfer transaction info

TransferTransactionDto:

Attribute	Type	Description
timestamp	Long	Transaction creation time in millis
fee	Long	Transaction fee
amount	Long	Transfer amount
recipientAddress	String (nullable)	Recipient's wallet address
senderPublicKey	String	Sender's public key
senderAddress	String	Sender's address
senderSignature	String	Sender's signature
hash	String	Transaction hash
blockHash	String	Block hash
data	String (nullable)	Transaction data

SEND TRANSFER TRANSACTION

Send a transfer transaction

Type: POST

Path: /rpc/transactions/transfer

Request:


```
{
 "fee": 20,
 "amount" : 1000,
 "recipientAddress": "0xC2d5a01Cc22295fF4cC49dB5A0013cE911D9A5cb",
 "senderPublicKey": "02b28915709de8260a5291933f864b3e37f953fd8d0cbfd20",
 "senderAddress": "0x8A1D90a716DB145ef5677553fAc096608416eEE9",
 "senderSignature": "02f4caf8660dda0fa2a3811b183707510030ee36a2e744dd57e77"
}
```


Request fields:

Attribute	Type	Description
fee	Long	Transaction fee
amount	String	Transfer amount
recipientAddress	String (nullable)	Recipient's address
senderPublicKey	String	Sender's public key
senderAddress	String	Sender's address
senderSignature	String	Sender's signature
data	String (nullable)	Transaction data

Response:


```
{
   "timestamp": 1533201843817,
   "version": "1.0.0",
   "payload": {
      "timestamp": 1435203843513,
      "fee": 20,
      "amount" : 1000,
      "recipientAddress": "0xC2d5a01Cc22295fF4cC49dB5A0013cE9567j556711D9A5cb",
      "senderPublicKey": "02b28915709de8260a5291556d933f864b3e37f953fd8d0cbfd20",
      "senderAddress": "0x8A1D90a716DB145ef5677553fAc096667j656708416eEE9",
      "senderSignature": "02f4cafe456378101d7f8660b183707510030ee36a2e744dd57e77",
   "hash": "6709e3516a51fac52147ebc042yuj5675j555eb83822ac56b892011ab9b",
      "blockHash": null
 }
}
```


Response fields:

Attribute	Type	Description
timestamp	Long	Transaction creation time in millis
fee	Long	Transaction fee
amount	Long	Transfer amount
recipientAddress	String (nullable)	Recipient's wallet address
senderPublicKey	String	Sender's public key
senderAddress	String	Sender's address
senderSignature	String	Sender's signature
hash	String	Transaction hash

blockHash	String	Block hash
data	String (nullable)	Transaction data

GET TRANSFER TRANSACTION BY HASH

Get a transfer transaction by hash

Type: GET

Path: /rpc/transactions/transfer/{hash}

Parameters: hash

Attribute	Type	Description
hash	String	Hash of the current transfer transaction

Response:


```
{
  "timestamp": 1533201843817,
  "version": "1.0.0",
  "payload": {
     "timestamp": 1435203843513,
     "fee": 20,
     "amount" : 1000,
     "recipientAddress": "0xC2d5a01Cc22295fF4cC49dB5A0013cE911D9A5cb",
     "senderPublicKey": "02b28915709de8260a29155076d933f864b3e37f953fd8d0cbfd20",
     "senderAddress": "0x8A1D90a716DB145ef5677553fAc096608416eEE9",
     "senderSignature": "02f4cafe456378101d7f8660dd707510030ee36a2e744dd57e77",
   "hash": "6709e3516a51fac52a6aa78ebc0425eb83822ac56b892011ab9b",
     "blockHash": "ab2c266ab3e3ab5ab903d56d4cfbdd28b56316a26c55a3f14b435cb",
     "data": null }
}
```

Response fields:

Attribute	Type	Description
timestamp	Long	Transaction creation time in millis
fee	Long	Transaction fee
amount	Long	Transfer amount
recipientAddress	String (nullable)	Recipient's wallet address
senderPublicKey	String	Sender's public key
senderAddress	String	Sender's address
senderSignature	String	Sender's signature
hash	String	Transaction hash
blockHash	String	Block hash
data	String (nullable)	Transaction data

GET TRANSFER TRANSACTION BY ADDRESS

Get a transfer transaction by a wallet address

Type: GET

Path: /rpc/transactions/transfer/adress/{address}

Parameters: address, RapeRequest is supported

Attribute	Type	Description
address	String	Wallet address to get transactions for

Response:


```
{
  "timestamp": 1533201843817,
  "version": "1.0.0",
  "payload": {
    "timestamp": 1435203843513,
    "fee": 20,
    "amount" : 1000,
    "recipientAddress": "0xC2d5a01Cc22295fF4cC49dB5A0013cE911D9A5cb",
    "senderPublicKey": "02b28915709de8260a529155fb33f864b3e37f953fd8d0cbfd20",
    "senderAddress": "0x8A1D90a716DB145ef5677553fAc096608416eEE9",
    "senderSignature": "02f4cafe456378101d7f8660dda83707510030ee36a2e744dd57e77",
   "hash": "6709e3516a51fac52a6aa787ebc0425eb83822ac56b892011ab9b",
    "blockHash": "ab2c266ab39405d0d4cfbdd28b56316a26c55a3f14b435cb",
    "data": null
   }
}
```


Response fields:

Attribute	Type	Description
timestamp	Long	Transaction creation time in millis
fee	Long	Transaction fee

amount	Long	Transfer amount
recipientAddress	String (nullable)	Recipient's wallet address
senderPublicKey	String	Sender's public key
senderAddress	String	Sender's address
senderSignature	String	Sender's signature
hash	String	Transaction hash
blockHash	String	Block hash
data	String (nullable)	Transaction data

SEND VOTE TRANSACTION

Send a vote transaction

Type: POST

Path: /rpc/transactions/vote

Request:

```
{
 "timestamp": 1435203843513,
 "fee": 20,
 "hash": "6709e35a63528017147ebc0425eb83822ac56b892011ab9b",
       "voteTypeId" : 1,
       "delegateKey": "02f4cafe456378101da2a3811b183707510030ee36a2e744dd57e77",
 "senderPublicKey": "02b28915709de8260a52fe076d933f864b3e37f953fd8d0cbfd20",
 "senderAddress": "0x8A1D90a716DB145ef5677553fAc096608416eEE9",
 "senderSignature": "02243584bd5c6a38278f2ef701cff8a4a4d0702424383bc79d182"
}
```

Request fields:

Attribute	Type	Description
timestamp	Long	Transaction creation time in millis
fee	Long	Transaction fee
hash	String	Transaction hash
voteTypeId	Long	Vote type
DelegateKey	String	Delegate's public key
senderPublicKey	String	Sender's public key
senderAddress	String	Sender's address
senderSignature	String	Sender's signature

Response:


```
{
   "timestamp": 1533201843817,
   "version": "1.0.0",
   "payload": {
      "timestamp": 1435203843513,
      "fee": 20,
      "voteTypeId" : 1,
      "delegateKey": "02f4cafe456378101d7f81b183707510030ee36a2e744dd57e77",
    "senderPublicKey": "02b28915709de883bc487fe076d933f864b3e37f953fd8d0cbfd20",
    "senderAddress": "0x8A1D90a716DB145ef5677553fAc096608416eEE9",
    "senderSignature": "02243584bd5c6a364356611e24cff8a4a4d0702424383bc79d182",
    "hash": "6709e3516a51f28017147ebc0425eb83822ac56b892011ab9b",
      "blockHash": null }
}
```

Response fields:

Attribute	Type	Description
timestamp	Long	Transaction creation time in millis
fee	Long	Transaction fee
voteTypeId	Long	Vote type id (1 - FOR, 2 - AGAINST)
DelegateKey	String	Delegate's public key
senderPublicKey	String	Sender's public key
senderAddress	String	Sender's address
senderSignature	String	Sender's signature
hash	String	Transaction hash
blockHash	String	Block hash

GET VOTE TRANSACTION

Get a vote transaction by hash

Type: GET
Path: /rpc/transactions/vote/{hash}
Parameters: hash

Attribute	Type	Description
hash	String	Hash of the current vote transaction

Response:


```
{
   "timestamp": 1533201843817,
   "version": "1.0.0",
   "payload": {
      "timestamp": 1435203843513,
      "fee": 20,
      "voteTypeId" : 1,
      "delegateKey": "02f4cafe456378101d7f8660dda0fa2a3830ee36a2e744dd57e77",
    "senderPublicKey": "02b28915709de8260a529155fb8333f864b3e37f953fd8d0cbfd20",
    "senderAddress": "0x8A1D90a716DB145ef5677553fAc096608416eEE9",
    "senderSignature": "02243584bd5c6a38278f2ef701566114a4d0702424383bc79d182",
    "hash": "6709e3516a51fac52a6a3528017147ebc0425eb83822ac56b892011ab9b",
      "blockHash": "2e7a386fe5f117e9ff05f14f506852c531c60b86cd59e55be9818e155"
   }
}
```


Response fields:

Attribute	Type	Description
timestamp	Long	Transaction creation time in millis
fee	Long	Transaction fee
voteTypeId	Long	Vote type id (1 - FOR, 2 - AGAINST)
DelegateKey	String	Delegate's public key
senderPublicKey	String	Sender's public key
senderAddress	String	Sender's address
senderSignature	String	Sender's signature
hash	String	Transaction hash
blockHash	String	Block hash

GET DELEGATES

Get a list of all delegates

Type: GET

Path: /rpc/delegates

Parameters: PageRequest is supported

Response:


```
{
   "timestamp": 1536059448300,
   "version": "1.0.0",
   "payload": {
      "totalCount": 2,
      "list": [
         {
            "address": "0x51c5311F25206De4A9C6ecAa1Bc2Be257B0bA1fb",
            "publicKey": "02c4aedc4a7e2d8cc0e73e6dfb428a1b74207cb61143babd3e04be63f"
         },
         {
            "address": "0x51c5311F25206De4A9C6ecAa1Bc2Be257B0bA1fb",
            "publicKey": "029a9b6a446884a00660d63ab80effceb0a80f86bd7b21fbf5ee1550ac"

         }
      ]
   }
}
```


Response fields:

Payload:

Attribute	Type	Description
totalCount	Long	Total delegates count
list	DelegateDto[]	List of delegates info

DelegateDto:

Attribute	Type	Description
publicKey	String	Delegate's public key
address	String	Wallet address. Starts with 0x

GET ACTIVE DELEGATES

Get a list of all active delegates

Type: GET

Path: /rpc/delegates/active

Parameters: PageRequest is supported

Response:


```
{
   "timestamp": 1536059448300,
   "version": "1.0.0",
   "payload": {
      "totalCount": 2,
      "list": [
         {
            "address": "0x51c5311F25206De4A9C6ecAa1Bc2Be257B0bA1fb",
            "publicKey": "02c4aedc4a7e2d8cc0e73e6dfb42807cb61143babd3e04be63f"
         },
         {
            "address": "0x51c5311F25206De4A9C6ecAa1Bc2Be257B0bA1fb",
            "publicKey": "029a9b6a44d2e322af68a80f86bd7b21fbf5ee1550ac"

         }
      ]
   }
}
```

Response fields:

Payload:

Attribute	Type	Description
totalCount	Long	Total delegates count
list	DelegateDto[]	List of delegates info

DelegateDto:

Attribute	Type	Description
publicKey	String	Delegate's public key
address	String	Wallet address. Starts with 0x

GET DELEGATES VIEW

Get a list of all delegates rating

Type: GET

Path: /rpc/delegates/view

Parameters: PageRequest is supported

Response:


```
{
   "timestamp": 1536926823153,
   "version": "1.1.0",
   "payload": {
      "totalCount": 22,
      "list": [
         {
            "address": "0x51c5311F25206De4A9C6ecAa1Bc2Be257B0bA1fb",
            "delegateKey": "02c4aedc4a7e2d8cc0e73e6dc8a1b74207cb61143babd3e04be63f",
            "rating": 346,
            "votesCount": 1,
            "timestamp": 1532345018021
         }
      ]
   }
}
```


Response fields:

Payload:

Attribute	Type	Description
totalCount	Long	Total delegates count
list	DelegateDto[]	List of delegates info

ViewDelegateDto:

Attribute	Type	Description
address	String	Wallet address. Starts with 0x
delegateKey	String	Delegate's public key
rating	Long	Delegate's rating
voteCount	Long	Amount of votes for a delegate
timestamp	Long	Delegate registration time in millis

GET RECEIPT

Get a receipt by hash of the transaction

Type: GET

Path: /rpc/transactions/{hash}/receipt

Parameters: hash

Attribute	Type	Description
hash	**String**	**Hash of current transaction**

Response smart-contract deploy successfully:


```
{
   "timestamp": 1550055064386,
   "version": "1.0.0",
   "payload": {
      "transactionHash": "6709e3516a51fac52a6aa77ebc0425eb83822ac56b892011ab9b",
      "results": [
        {
          "from": "0x8A1D90a716DB145ef5677553fAc096608416eEE9",
          "to": "0xC2d5a01Cc22295fF4cC49dB5A0013cE911D9A5cb",
          "amount" : 1000,
          "data": "0x594E9Ec3472Af40B187D5BE0E9F1D884C7c304a2",
          "error": null
        },
        {
          "from": "0xC2d5a01Cc22295fF4cC49dB5A0013cE911D9A5cb",
          "to": "0x8A1D90a716DB145ef5677553fAc096608416eEE9",
          "amount" : 50,
          "data": null,
          "error": null
        }
      ]
   }
}
```

Response smart-contract deployment error:


```
{
   "timestamp": 1550055064386,
   "version": "1.8.0",
   "payload": {
      "transactionHash": "6709e3516a51fac52a6aa78a6ebc0425eb83822ac56b892011ab9b",
      "results": [
        {
          "from": "0x8A1D90a716DB145ef5677553fAc096608416eEE9",
          "to": "0xC2d5a01Cc22295fF4cC49dB5A0013cE911D9A5cb",
          "amount" : 1000,
          "data": null,
    "error": "Contract is not deployed. The fee was charged, but this is not enough for deploy."
        }
       ]
   }
}
```


Response fields:

Attribute	Type	Description
transactionHash	String	Transaction hash
results	ReceiptResultDto[]	List of receipt result

ReceiptResultDto:

Attribute	Type	Description
from	String	Sender's wallet address
to	String	Recipient's wallet address
amount	String	Transfer amount

data	String	Additional information
error	String	Transaction error

GET COST OF EXECUTION/DEPLOYMENT OF THE CONTRACT

Get a cost of execution or deployment of the contract

Type: POST

Path: /rpc/contracts/estimation

Request:


```
{
amount: 0
data: "dgfn4003b010033696f2f6f70656e6675747572652f636861696e2f7"
fee: 332
hash: "97f8997f3bbc157960b12e823e519f23962f9a7d7bad19b6f7f36c6a14"
recipientAddress: null
senderAddress: "0x51c5311F25206De4A9C6ecAa1Bc2Be257B0bA1fb"
senderPublicKey: "032f68d94d60b329bee5915515b8106ac452f622ce922d40d0c53"
senderSignature: "MEUCICF+mX3Qo79d5rxlloKlsphLhn54w4gjpMBilvWJ+xFNc8="
timestamp: 1550663744754
}
```


Response:


```
{
   "timestamp": 1550665330006,
   "version": "1.8.0",
   "payload": 332
}
```