



Technical description of the Waves Enterprise platform

Release master

<https://wavesenterprise.com>

Jan 14, 2021

BLOCKCHAIN-PLATFORM WAVES ENTERPRISE

FEATURES OVERVIEW

Waves Enterprise Blockchain Platform — is a universal solution for scalable digital infrastructures that combines features of public and private blockchain for corporate and government use. The enterprise blockchain platform solves the problem of trust between the parties not at the level of business logic, but at the level of the platform operation protocol. Supported *PoS* and *PoA* consensus guarantee correctness of data added to the blockchain, while decentralization provides counterparty independence in terms of data access.

1.1 Blockchain Waves Enterprise

- Built on Scala programming language.
- Includes technologies and best use practices of use proven on the Waves public blockchain platform.
- Adapted for corporate and government use.
- Supports algorithms of *PoS* and *PoA* consensus - you can choose the most fitting one during the network deployment.
- Ensures high throughput rate.
- Supports two types of *smart contracts*: Turing-incomplete RIDE contracts and Turing-complete Docker contracts.
- Delivered as a set of microservices.
- Uses cryptographic algorithms certified by state regulators.
- Supports confidential and direct data exchange via private groups without loading data into external networks.
- Permission management system implemented at the consensus level.
- Waves Enterprise web client features *transactions* explorer, wallet, creation of transactions, smart contract development, blockchain status monitoring, permission management.

1.1.1 Waves Enterprise network deployment options

1. Operating in the main public network.
2. Operating in a private network anchored to the main network.
3. Operating in an independent private network.

1.2 Main network

Main network is supported by consortia of companies from various economy sectors including banking, industrial, real estate, logistics, etc. Companies which run main network may be using public blockchain for their projects or for supplying blockchain processes, e.g. banking enterprises delivering fiat *gateways* and state registrars granting access to cloud GOST-cryptography.

1.3 Independent private network

There are cases when company is not ready to share its processes publicly. Waves Enterprise box version allows such companies to deploy a stand-alone private network and configure it in accordance to business needs.

Following features are configurable:

- Consensus type.
- Cryptography provider.
- Number of nodes.
- Blockchain operating parameters.

1.4 Private network with block hashes broadcasted to main network

This solution combines advantages of the preceding concepts and can be relevant for small companies and/or partners of companies supporting main network. Private networking allows concealing private information from public network. Broadcasting of private block hashes into the main network ensures reliability of the broadcasted information due to scalability effect of main network.

OFFICIAL RESOURCES

- Official site of the blockchain-platform [Waves Enterprise](#)
- [Github](#) project
- Official site of the blockchain-platform [Waves](#)

ARCHITECTURE

The Waves Enterprise platform is based on distributed ledger technology and represents a fractal network consisting of a master blockchain (Waves Enterprise Mainnet), which secures the operation of the network as a whole, serving as a global arbiter and a reference chain, and a number of custom, separated sidechains that can be easily tuned according to a specific business task. The implementation of such construction principle allows to achieve optimization for higher speeds or large volumes of calculations, consistency and availability of data, as well as resistance to malicious changes in information.

Anchoring mechanism uses strengths of both consensus algorithms for creating a net configuration. The main blockchain Waves Enterprise is based on the Proof-of-Stake consensus algorithm, since it is supported by independent participants. At the same time, enterprise sidechains do not need to stimulate miners by means of transaction fees and can use the Proof-of-Authority algorithm. Sidechains are embedded in the main blockchain using the anchoring mechanism (by placing cryptographic proof of transactions in the main blockchain network).

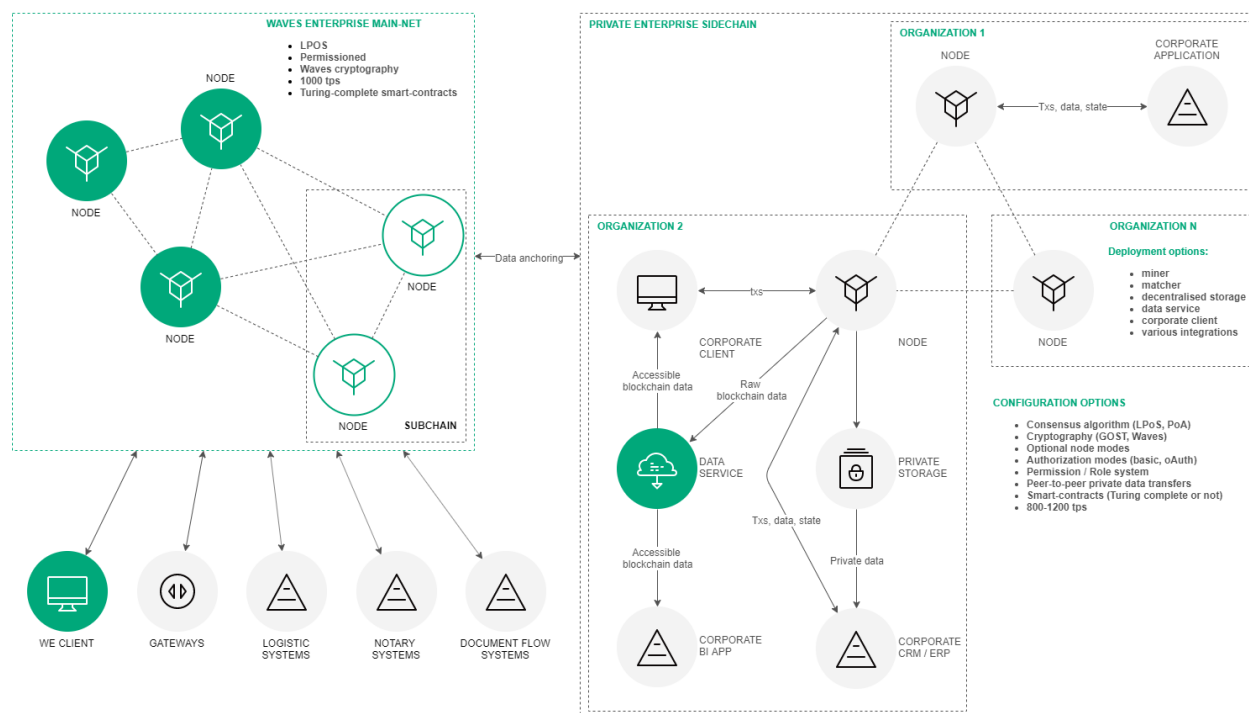


Fig. 1: Network topology including Waves Enterprise and sidechains

3.1 Node architecture and additional services

Only the node component is mandatory, since it ensures the functioning and interaction within the blockchain network. Other components serve auxiliary purposes, but significantly simplify user interaction with the blockchain platform. The Waves Enterprise platform instance consists of five basic modules and several additional microservices. The main modules include:

- Node - the main software which is installed on the computer and is set for work with the blockchain according with any scenario.
- Waves Enterprise corporate client – *web-application* that provides contemporary and multifunctional user interface for blockchain platform.
- Smart-contracts module – an environment for deployment and execution of Turing-complete *Docker smart-contracts*. Docker containers with smart-contracts are deployed on remote virtual machine for additional security.
- Data service – *the service* aggregates data from the blockchain in RDBMS (PostgreSQL) storage and provides full-text search on any information containing in blockchain via RESTful webservice.
- Private store - DB PostgreSQL provides private information processing and storing mechanisms, along with encrypted p2p communication service.

Additional services include:

- Authorization service – a single authorization service for system components.
- Data crawler - the service extracts data from blockchain node and loads it into Data-service component.
- Generator - the service generates key pairs for new accounts and creates *api-key-hash*.
- Custom microservice plugins - a set of plugins for processing and customization of the data transferred and accepted from external systems.
- Monitoring Service – an external monitoring service that uses a database (InfluxDB) for storing time rows with application data and metrics. The InfluxDB database is an open source software and is installed by the client separately.

Node components

The node includes the following internal components:

- Node API – interface of the REST API node which allows receiving the data from the blockchain, sign and send transactions, sending private data, creating and calling smart contracts, and so on.
- Node storage – a system component that provides key-value storage (based on LevelDB) for a full set of validated and confirmed transactions and blocks, same as the current state of objects.
- Unconfirmed transaction pool – a component that provides temporary storage and queue service for validated transactions until they are included into a block.
- Consensus and cryptolibraries – configurable and customizable logical components responsible for achieving the agreement between nodes and cryptographic algorithms.
- Key store - a component used to store key pairs for node itself and node users (optional), all keys are secured by passwords.
- Miner – a component responsible for the creating of transaction blocks for the recording in the blockchain. Also miner component is in charge for interaction with docker-smart contracts.
- Network layer – a logic layer that provides interaction between nodes on the application level via network protocol over the TCP.

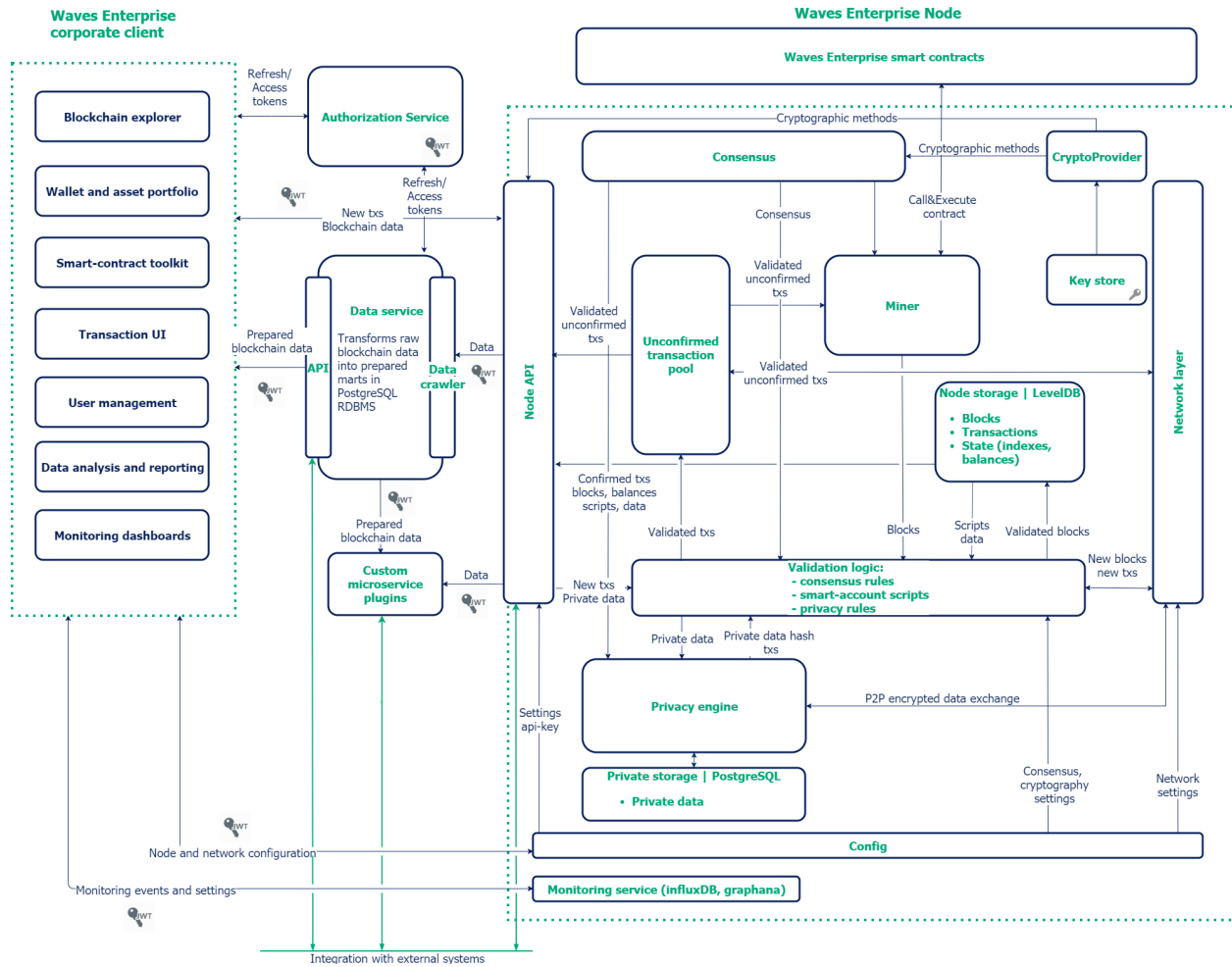


Fig. 2: A detailed diagram of the node architecture and additional microservices

- Validation logic – a logic layer containing such transaction verification rules as basic sign verification and advanced scripted verification.
- Config – node configuration parameters specified in the `node-name.conf` file.
- Monitoring Service – an external monitoring service that uses a database (InfluxDB) for storing time rows with application data and metrics. The InfluxDB database is an open source software and is installed by the client separately.

WAVES-NG PROTOCOL

Description of Waves Enterprise Operation Protocol which provides performance advantage relative to other blockchains.

4.1 Terms

- Block — a set of transactions registered in the blockchain signed by the miner and containing a link to the proof of the previous block. Limited to 1MB or 6000 transactions.
- Round — a period of time between issuance of key blocks. A floating value is controlled by the consensus algorithm depending on the load on the network, averaging 40 seconds.
- Proof of ownership — acquisition of mining right in the PoS consensus.
- Node — network host with the Waves Enterprise blockchain application running.
- Miner — a node whose address has sufficient balance and a “mining” permission.
- Key block — contains no transactions, only service information such as:
 - Miner public key — to verify proof of microblocks.
 - Amount of miner’s fee for the previous block.
 - Miner’s proof.
 - Link to previous key block.
- Liquid Block — a service term to describe the state of a block before issuing the next key block, i.e. completing its mining.
- Microblocks — a service term, sets of transactions applied to the state of blockchain every 5 seconds. Limited to 500 transactions. Each microblock is signed by the miner’s private key.

4.2 Protocol description

Waves-NG - developed by Waves Platform based on [Bitcoin-NG](#) to increase the throughput of the Waves blockchain based on whose architecture Waves Enterprise is implemented. The idea of the protocol is to create not 1 large block in each round of mining but continuously create microblocks. Small blocks are faster to forward and check.

Mining rounds begin with generation of the key block. The moment of occurrence of each key block and the address of the miner identified in it are determined by consensus, for more details see *Consensus*. A key block containing no transactions, but only proof, is generated quickly. Further, before the next block is generated, once in 5 seconds microblocks with transactions are generated without proof of a stake which

also increases the speed of processing. Each block of the microblock is linked to the previous one. The key block is added to the blockchain as soon as the next miner generates its key block with a link to it.

This approach reduces the time to confirm a transaction compared to other blockchains.

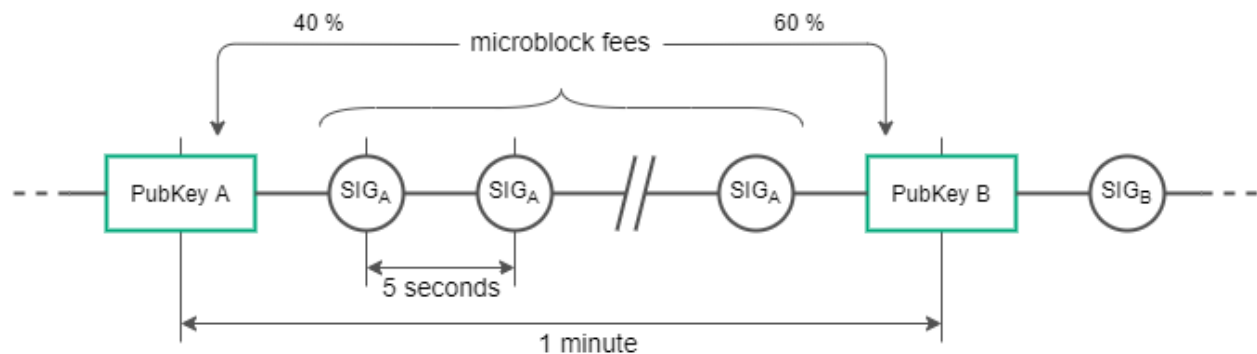
4.2.1 1. Creating a Liquid Block

1. The mining address is determined by consensus.
2. A miner creates and distributes a key block on the network.
3. Every 5 seconds, the miner creates and sends out to the network a microblock which contains transactions. It must be linked to the previous microblock or key block.
4. The process continues until a new valid key block appears on the network.

4.2.2 2. Miner reward mechanism in Waves-NG

The protocol has a provision for financial incentive for participants' compliance with the rules. The fee from transactions in the block is distributed in the proportion of 40% for miner who created the block and 60% for the miner of the following block. The fee credit transaction is performed after 100 blocks to ensure a trust interval of checks.

Fee distribution diagram



4.2.3 3. Conflict resolution

If the miner continues the chain creating two microblocks with the same parent, it is punished by cancellation of income from fees; the one who discovers the fraud receives the miner's award for the block. Blockchain is a distributed system and each node stores a copy of the state of the entire network. When the next microblock appears, the node applies the received changes to its copy of the state of the network and checks against other nodes of the network. At this point, the transaction inconsistency can be detected.

CONSENSUS ALGORITHMS

Blockchain is a decentralized system with no central authority. This makes the system non-corrupt, but creates difficulties with the final decision making and organization of work. These problems could be solved by a consensus mechanism, which is a way to reach agreement in a group of participants. Voting takes in account the majority opinion without interests of the minority, but on the other hand, it guarantees an agreement that benefits the entire network.

You can choose the consensus mechanism during the initial configuration of the network. The description of available mechanisms, as well as their pros and cons, are described in the relevant sections.

5.1 LPoS consensus algorithm

Proof of ownership with the right to lease. In PoS systems, the creation of a block does not require energy-intensive calculations, the miner's task is to create a digital block proof.

5.1.1 Proof of Stake

The mechanism for allocating block creation rights is based on the number of tokens in the user's account. The more tokens a user has, the more likely he or she can create a block.

In Proof of Stake consensus the right to generate a block is determined by pseudo-random way, because by knowing the previous miner and balances of all users in the system the following miner can be identified. To do this, calculate the generating signature of the next block as sha256 hash from the generating signature of this block and the public account key. The first 8 bytes of the resulting hash will be a pointer to the following miner. The time of block generation for account i is calculated as:

$$T_i = T_{min} + C_1 \log\left(1 - C_2 \frac{\log \frac{X_n}{X_{max}}}{b_i A_n}\right)$$

where:

- b_i - is a stake (stake of participant's balance of overall balance of the system);
- A_n - baseTarget, the adaptive ratio, regulating the average time of issue of the block;
- X_n - generating signature;
- T_{min} - 5 seconds, it is a constant defining the minimum time interval between blocks;
- C_1 - a constant, which is equal 70 and adjusting the form of allocation of the interval between blocks;
- C_2 - a constant which is equal 5E17 and adjusting the baseTarget value (complexity).

Based on the given formula it is easy to see that the probability of selecting the participant depends on the participant's stake of assets in the system: the bigger the stake the higher the chance. The minimum number of tokens on the balance for mining is 10000 WEST. BaseTarget is a computational complexity, a parameter that maintains the block generation time within a given range. BaseTarget in its turn is calculated as:

$$(S > R_{max} \rightarrow T_b = T_p + \max(1, \frac{T_p}{100})) \wedge (S < R_{min} \wedge \wedge T_b > 1 \rightarrow T_b = T_p - \max(1, \frac{T_p}{100}))$$

where

- $R_{max} = 90$ - is a maximum reduction of complexity when the block generation time in the network exceeds 40 seconds;
- $R_{min} = 30$ - minimal increase of complexity when the block generation time in the network is less than 40 seconds;
- S - average generation time, at least for the last three blocks;
- T_p - previous baseTarget value;
- T_b - computed baseTarget value.

For advanced description of technical features and enhancements of the classic PoS algorithm see [article](#).

Advantages Over Proof of Work

The absence of complex calculations allows PoS networks to lower the hardware requirements for participants of the system, which reduces the cost of deploying private networks. Also, no additional emission is required, which in PoW systems is used for rewarding miners for finding a new block. In PoS-Systems a miner receives a reward in the form of fees for transactions which appeared in its block.

5.1.2 Leased Proof of Stake

For a user who has a stake insufficient for effective mining, it is possible to transfer his or her balance for lease to other participants, and receive a portion of the income from mining. Thus you can increase the likelihood of choosing a miner, for which you can receive a portion of the fees for transactions which this miner has placed in its blocks. Leasing is a completely safe operation. Tokens do not leave your wallet, you delegate the right to consider your balance when you draw the right of mining to another member of the network.

5.2 Proof of Authority Consensus Algorithm

In a private blockchain tokens are not always needed - for example, a blockchain can be used to store hashes of documents exchanged by organizations. In this case, in the absence of tokens and fees from transactions, a solution based on the PoS consensus algorithm is redundant. The Waves Enterprise platform gives a possibility to choose the Proof of Authority consensus algorithm. The mining permission is issued centrally in the PoA algorithm. Compared to the PoS algorithm this simplifies the decision-making. The Proof of Authority model is based on a limited number of block validators, which is making it a scalable system. Blocks and transactions are verified by pre-approved participants who act as moderators of the system.

5.2.1 Algorithm description

An algorithm determining the miner of the current block is formed based on the described below parameters. The parameters of the consensus are specified in the `consensus` block of the node configuration file.

- t - duration of a round in seconds (the parameter of the node configuration file: `round-duration`).
- t_s - duration of a synchronization period, calculated as $t*0.1$, but not more than 30 seconds (the parameter of the node configuration file: `sync-duration`).
- N_{ban} - a number of missed consecutive rounds for issuing the ban for the miner (the parameter of the node configuration file: `warnings-for-ban`);
- P_{ban} - a share of the maximum number of banned miners, in percentage from 0 to 100 (the parameter of the node configuration file: `max-bans-percentage`);
- t_{ban} - duration of miner ban in blocks (the parameter of the node configuration file: `ban-duration-blocks`).
- T_0 - unix time for generation of Genesis block.
- T_H - unix time for generation of H Block - a key block for NG.
- r - round number, calculated as $(T_{\text{Current}} - T_0) \text{ div } (t + t_s)$.
- A_r - leader of round r , which has the right to create key blocks and microblocks for NG in round r .
- H - height of the chain in which the key block and microblocks for NG are created. The leader of round A_r has the right to generate a block at height H .
- M_H - miner issuing block at height H .
- Q_H - queue of miners active at height H .

The Q_H queue is generated using addresses which are given the mining permission by a permission transaction which was not revoked until height H , and did not expire until the time T_H .

The queue is sorted by the time stamp of the mining rights transaction - the node which was granted the rights earlier will be higher in the queue. For a consistent network, this queue will be the same on each node.

A new block is created at each round r . A round lasts t seconds. After each round, t_s seconds count down to complete data synchronization in the network. During the synchronization period, microblocks and key blocks are not generated. For each round there is a single leader A_r , which has the right to create a block in this round. A leader can be defined on each node of the network with the same result. The leader of the round is defined as follows:

1. Miner M_{H-1} is defined, which created the previous key block at height $H-1$.
2. The Q_H queue of active miners is calculated.
3. Inactive miners are excluded from the queue (see more in *Exclusion of inactive miners*).
4. If the $H-1$ block miner (M_{H-1}) is in the Q_H queue, the following miner becomes the leader A_r .
5. If the $H-1$ block miner (M_{H-1}) is not in the Q_H queue the miner following the $H-2$ block miner (M_{H-2}) becomes the leader A_r and so on.
6. If no miners of blocks ($H-1..1$) are in the queue, the first miner in the queue becomes the leader.

This algorithm allows to identify and check deterministically the miner, which had to create each block of the chain by means of the ability to calculate the list of authorized miners for each moment of time. If the block was not created by the designated leader within the allotted time, no blocks are generated within the current round, and the round is skipped. Leaders who skip block generation are temporarily excluded from the queue by the algorithm described in paragraph *Exclusion of inactive miners*.

Valid is the block generated by the leader A_r with the time of block T_H from the half-interval $(T_0 + (r-1) \cdot (t + t_s); T_0 + (r-1) \cdot (t + t_s) + t]$. The block created by the miner out of its turn or not in time, is not considered valid. After a round of t duration, the network synchronizes the data for t_s . The leader A_r has time t_s to propagate the validation block over the network. If any node of the network during t_s has not received a block from the leader A_r , this node recognizes the round as “skipped” and expects a new H block in the next round $r+1$, from the following leader A_{r+1} .

Consensus parameters: type (PoS or PoA), t , t_s are specified in the configuration file of the host network. *The parameter T should be the same for all network participants*, otherwise the network will fork.

5.2.2 Synchronization of time between network hosts

Each host should synchronize the application time with a trusted NTP server at the beginning of each round. The server address and port are specified in the node configuration file. The server must be available to each network node.

5.2.3 Exclusion of inactive miners

If any miner has missed the block creation N_{ban} times in a row, this miner is excluded from the queue at t_{ban} subsequent blocks (`ban-duration-blocks` parameter in the configuration file). The exception is made by each node on its own based on the calculated queue Q_H and information about block H and miner M_H . The P_{ban} parameter specifies the maximum allowable share of excluded miners in the network relative to all active miners at any given time. If at achievement of N_{ban} round passes it is known that the maximum share of the excluded miners P_{ban} is reached, the exception of the next miner is not made.

5.2.4 Monitoring

The PoA consensus monitoring helps to identify the facts of creation and distribution of non-valid blocks, as well as skipping queue by the miners. Further troubleshooting and blocking of malicious nodes are performed by network administrators.

In order to monitor the process of generation of blocks for the PoA algorithm the following details are entered in InfluxDB:

- Active list of miners sorted in order of granting mining rights.
- Scheduled round timestamp.
- Actual round timestamp.
- Current miner.

5.2.5 Changing consensus settings

Changing consensus parameters (time of round and synchronization period) is performed on the basis of the node configuration file (see the insert) at the height “from-height”. If one of the nodes fails to specify new parameters, the transaction will fork.

Sample configuration:

```
// specifying inside of the blockchain parameter
consensus {
  type = poa
  sync-duration = 10s
```

(continues on next page)

(continued from previous page)

```
round-duration = 60s
ban-duration-blocks = 100
changes = [
  {
    from-height = 18345
    sync-duration = 5s
    round-duration = 60s
  },
  {
    from-height = 25000
    sync-duration = 10s
    round-duration = 30s
  }
]
```


CRYPTOGRAPHY

To ensure security, the Waves Enterprise platform uses cryptographically resistant algorithms complying with GOST requirements, as well as qualified electronic signatures.

The platform provides a choice of cryptography to be used based on one of the standards:

- GOST-cryptography - for implementation of projects in government structures and departments;
- Elliptical curve Curve25519 - for projects which do not require strict compliance with GOST (commercial companies, organizations outside the Russian Federation).

6.1 Data Pre-processing

All byte sequences before hashing or signing operations are converted using Base58 or Base64 algorithm.

Hint: Base64 is a binary data encoding standard using 64 ASCII characters. The encoding alphabet contains Latin symbols and numbers A-Z, a-z and 0-9 (62 characters) and 2 service symbols. Every 3 source bytes are encoded with 4 characters ($1/3$ magnification).

Hint: BASE58 is a variant of encoding binary data in the form of alphanumeric text based on the Latin alphabet. It contains 58 characters, the characters used in Base64, such as 0, O, I, L, +, / are excluded. It is used for data transmission in heterogeneous networks (transport coding).

6.2 GOST-Cryptography

Hashing algorithm is implemented according to GOST R 34.11 2012 ‘Information Technology. Cryptographic Protection of Information. Hashing Function’. The Stribog function is used with 256 bits output blocks.

The encryption algorithm is implemented according to GOST ‘Information processing systems. Cryptographic protection. Algorithm of cryptographic transformation’.

EDS generation and verification algorithms are implemented according to GOST R 34.10-2012 ‘Information Technology. Cryptographic Protection of Information. Electronic Digital Signature Generation and Verification Processes’. The key length is 256 bits.

6.3 Using an elliptical curve Curve25519 (Waves cryptography)

The hashing algorithm is implemented by successive hashing functions Blake2b256 and Keccak256. The size of output blocks is 256 bits.

EDS generation and verification algorithms are implemented on the basis of elliptical curve Curve25519 (ED25519 with X25519 keys). The key length is 256 bits.

6.4 Encrypting text data in transactions

The Waves Enterprise platform allows to encrypt/decrypt the text data using the session keys. The option is used to encrypt any kind of text information, for example, Docker contract parameters or data from *Data Transactions*. Encryption of text data can be performed both individually for each recipient, with forming of a unique instance of the encrypted text, or with forming of a single encrypted text for a group of recipients. Limit size to encrypt data is 165 Kbytes. Hashing algorithms correspond to the chosen cryptography scheme (GOST or Waves).

6.4.1 Encryption algorithm

Symmetric CEK and KEK keys are used to encrypt/decrypt data. CEK (Content Encryption Key) is the key for the encrypting text data, KEK (Key Encryption Key) is the key for encrypting the CEK. The CEK key is generated by a node randomly using the appropriate hashing algorithms. The KEK key is generated by a node based on Diffie-Hellman algorithm, using public and private keys of sender and recipients, and is used to encrypt the CEK key.

The symmetric CEK key is unreachable and does not appear in the encryption process. It is transmitted from the sender to the recipient in the encrypted form (wrappedKey) via open communication channels along with the encrypted message. One of such channels can be a record to the blockchain — a *DataTransaction* or a smart contract state. The KEK key does not transmit from the sender to recipients, it is restored by the recipient based on its private key and the known public key of the sender (Diffie-Hellman key exchange algorithm).

Text data encryption/decryption is performed using the *crypto* method. This method allows to encrypt the text individually for each recipient or for all recipients together.

Encryption/decryption process includes the following actions:

1. Use the *POST /crypto/encryptSeparate* method to encrypt data for each recipient separately. Specify the following parameters inside the JSON request:
 - **sender** - the sender address.
 - **password** - a key pair password of the sender, which is generated at the same time as the account itself.
 - **encryptionText** - the text for the encryption.
 - **recipientsPublicKeys** - an array with recipients public keys list inside.
2. Use the *POST /crypto/encryptCommon* method to encrypt data for all recipients with a single CEK key. Specify the same parameters as inside the *POST /crypto/encryptSeparate* method into the JSON request.
3. Use the *POST /crypto/decrypt* method for the decryption. Specify the following parameters inside the JSON request:

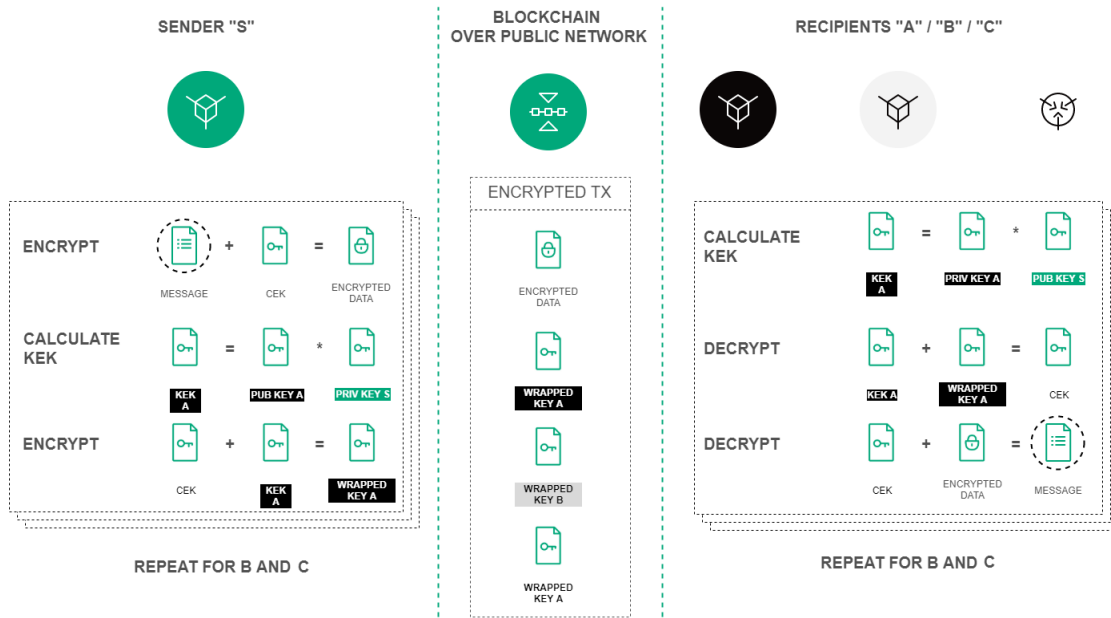


Fig. 1: Encryption procedure of the text data based on the Diffie-Hellman algorithm

- **recipient** - the recipient address.
- **password** - a key pair password of the recipient, which is generated at the same time as the account itself.
- **encryptedText** - the encrypted text data.
- **wrappedKey** - the wrapped key obtained by encoding the data.
- **senderPublicKey** - the sender public key.

ROLE MODEL

The blockchain platform implements a mechanism limiting actions of participants based on the role model which allows the platform owner to protect participants from threats, such as:

- attacks of unscrupulous miners on blockchain network;
- unauthorized issue of tokens;
- unauthorized access to confidential information;
- other illegal actions of intruders.

The procedure for issuing and revoking permissions is given in module *Role management*.

7.1 Roles list

The following table provides a list of possible platform roles:

Role name	Authority
permissioner	Add transactions to modify the permission list
blacklister	Add transactions to modify the black list
miner	Create new blocks
issuer	Add transactions for issuing, reissuing, and burning tokens
dex	Add the exchange transaction (deprecated)
contract_developer	Add the transaction to create a docker contract
connection-manager	Add the transaction for registering/deleting node in the blockchain network
banned	It is forbidden to send any transactions to the blockchain. A group of all participants with this role forms a blacklist

7.2 Permission model

Permission model describes a mechanism for applying different types of permissions when validating operations in a blockchain.

Hint: The node with the **permissioner** role can assign to itself any existing role in the system.

Action	Action permission condition
Assign or remove a role	Available permissioner role
Add or Remove from blacklist	Available blacklister role
Registration of the new node to the net	Available contract_developer role
Generation and issue of blocks	Available miner role
Token operations (issue, reissue, burn)	Available issuer role
Token transfer (transfer, mass transfer)	User not in the blacklist
Token leasing (lease, lease cancel)	User not in the blacklist
Creating an alias (alias)	User not in the blacklist
Create a docker contract	Available contract_developer role
Execution of docker contract	User not in the blacklist

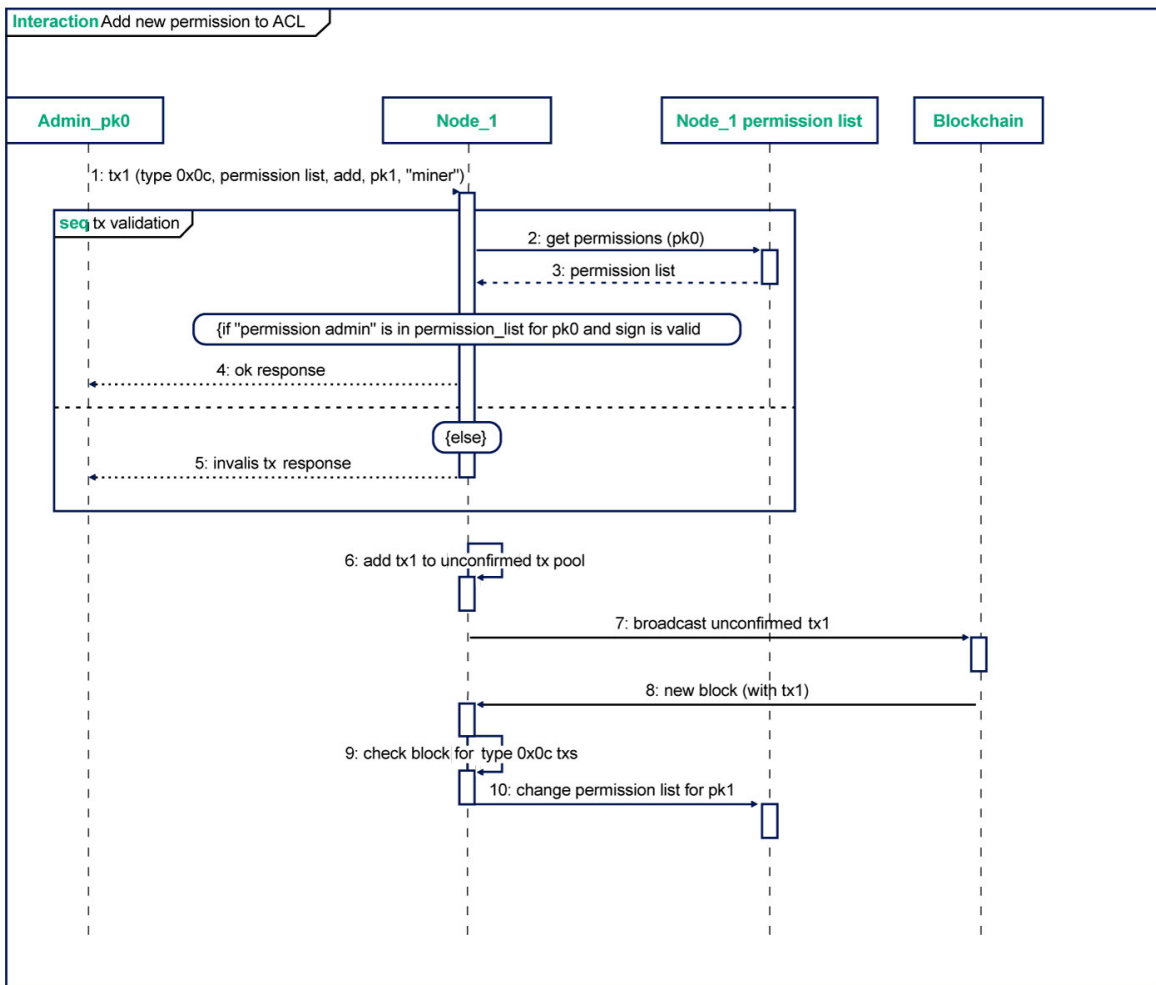
7.3 Update the permission list

A permission transaction is used to modify the permission list.

JSON description:

- Transaction Type
- Version
- Sender PublicKey
- Target Address or Alias
- Timestamp
- Operation Byte
- Role Byte
- Timestamp
- Due Timestamp Defined Byte (0 - None, 1 - Defined)
- Due Timestamp Bytes

The following diagram shows the sequence of actions when updating a permission list.



When modifying the permission list, the platform performs the following checks:

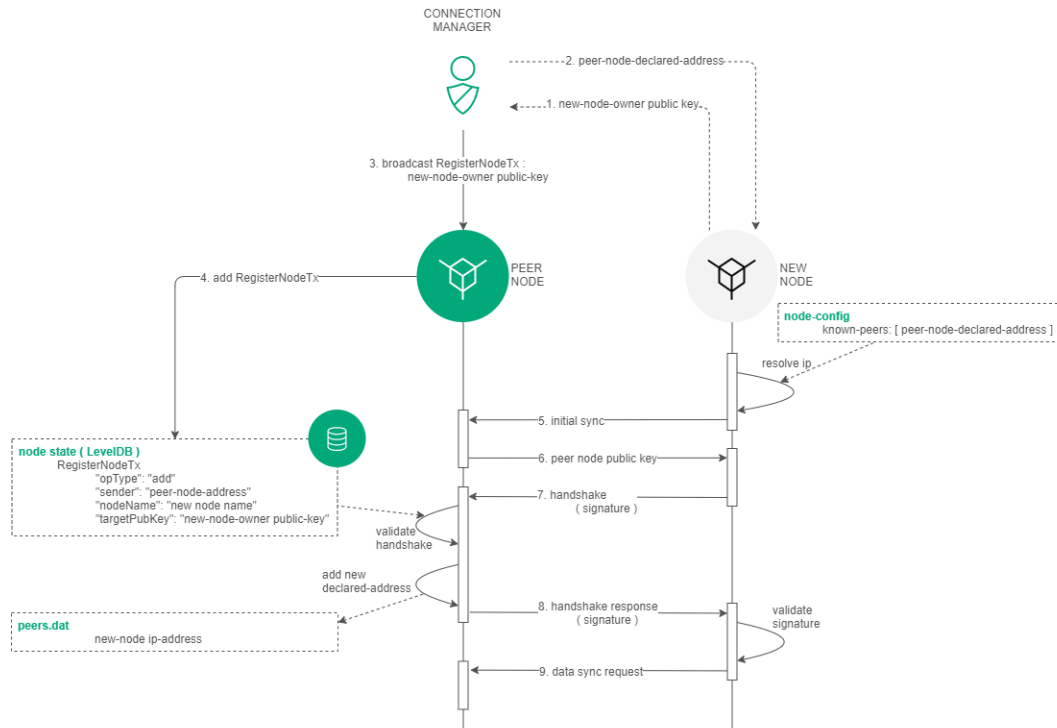
1. Sender is not in the blacklist.
2. Sender has the role of permissioner.
3. DueTimestamp (role duration) > Timestamp (current time).
4. This role is not active (if added) or active (if removed).

ACCESS MANAGING

The Waves Enterprise platform implements the closed model of blockchain where the new participants adding is under control of an individual user with the authority. This model of blockchain is also supports the restriction for the data access for all participants. The advantage of this model is its increased security compared to open blockchains, as well as the ability to flexibly configure access levels and distribution of rights.

Only a user with the “Connection Manager” role can add new participants to the Waves Enterprise blockchain. The *111 RegisterNode* transaction is used to connect a new node to the network. This transaction contains the credentials of the connected node. As a result of all such transactions each node is creating and updating the table which includes all approved network participants.

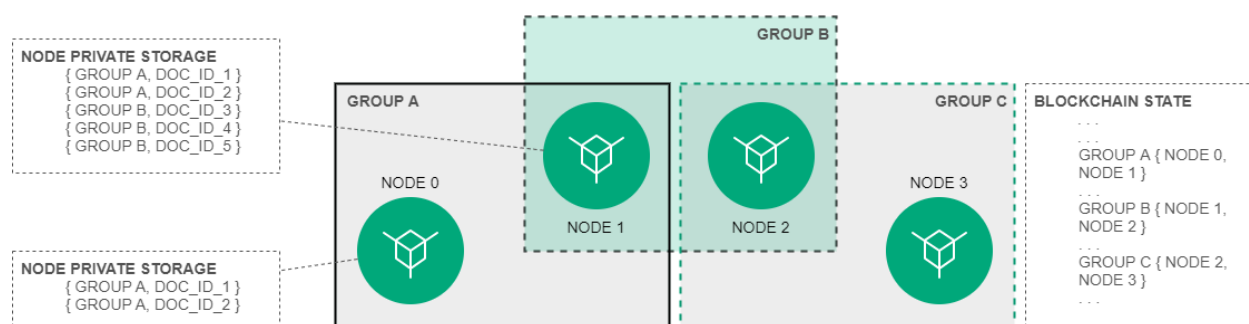
Each attempt of the participant connection is accompanied by *handshake-message*, which in addition to service information specifies the data area with proof of belonging to the connected network - in a simplified way it is a set of public key with the electronic signature of the participant. Since the public key of the connected participant is already stored in the storage of other peers, the participant who received the handshake request verifies the signature and the public key provided earlier in the blockchain. If the check is successful, the participant generates a response handshake request, the success of which establishes a connection between the parties. After successful connection participants perform the network synchronization as well as synchronization of the table of corresponding of blockchain and network addresses of nodes, which is necessary in the future in the process of sending *private data*.



The process of disconnecting a participant from the network is similar to the process of connection, except that the “Connection Manager” user sends the `111 RegisterNode` transaction with the `opType: "remove"` parameter. Since the handshake request is executed at a frequency of 1 time every 30 seconds, the next request after the participant is removed from the network will be denied, due to the lack of credentials of the connected participant in the blockchain node table.

DATA PRIVACY

Blockchain platform Waves Enterprise provides the confidential data transfer and storage between the participants of network interaction. Protection of confidential data during its transfer and storage is provided by a set of groups, which contain a list of participants for the interaction with private data.



9.1 Access groups

Usually the access group is created by the net participants who need to arrange the private data exchange. Any participant can create an access group and add into it any number of other participants. Only nodes can exchange information within a group.

The group contains the following parameters:

- name (policyName);
- description (policyDescription);
- duration (policyDueDate);
- the list of confidential data recipients (policyRecipients);
- the list of the policy owners with editing rights (policyOwners).

The access group is created by sending transaction *CreatePolicy* (type = 112, group creation) into the blockchain.

Owners can change the access group. To make this it is necessary to send the *UpdatePolicy* (type = 113, group editing) transaction into the blockchain.

For external access and getting the information about groups there are using specified *API Node* requests: GET /privacy/{policy}/recipients, GET /privacy/{policy}/getHashes, GET /privacy/getInfo/{hash}.

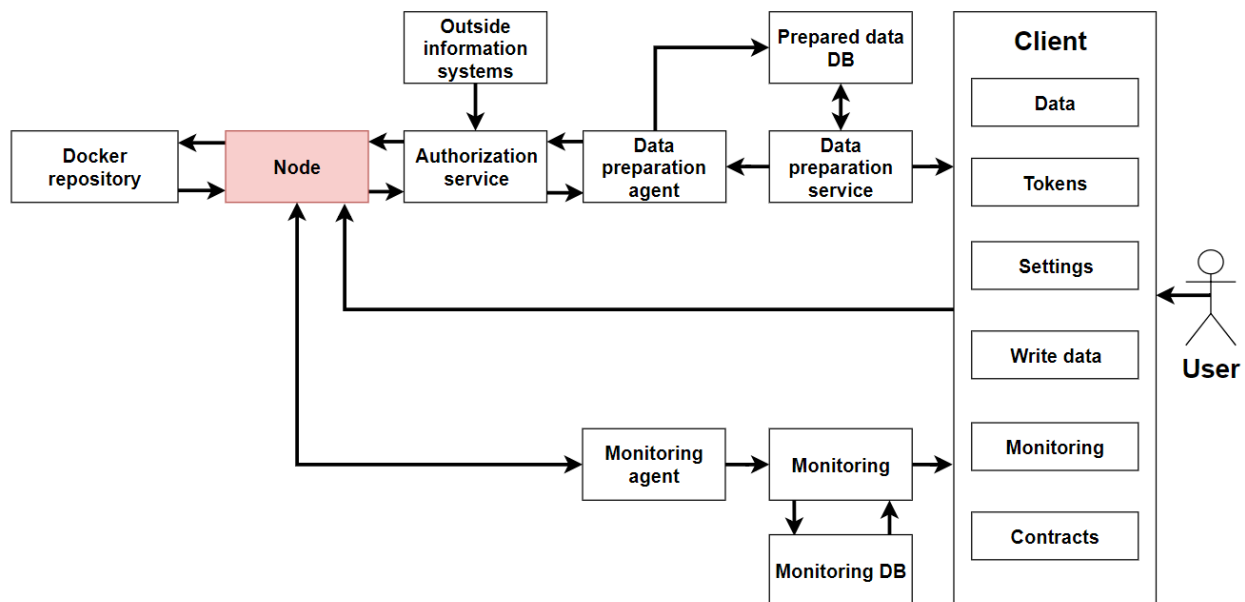
9.2 Sending and receiving the data

The data is sending via POST `/privacy/sendData` request through its own node of the organization, which checks the sender's belonging to the specified group. If the check is successful, the data is written to the node store, and the *PolicyDataHash* (type = 114, sending the data hash) transaction is initiated with the calculated hash sum of the data. The size of the transferred data to the network is up to 20 MB.

When receiving a transaction with the hash sum from the transmitted data, the receiving party checks whether the blockchain node is involved in the group specified in the transaction. If the participant is belong to the group, the `getPrivateData` request for confidential data is executed. The request is executed at the network address of the group participant via P2P connection. To ensure the security of data transmission over an unprotected communication channel, a set of encryption algorithms on a symmetric key and the creation of session keys, as well as the Diffey-Hellman protocol are used.

10.1 General Description of the Client

Waves Enterprise client is a convenient way to manage your blockchain. <https://client.wavesenterprise.com> is intended for operations in the Waves Enterprise public network.



The client includes sections for use of all blockchain features:

- “Data” — allows to find information about transactions or users through flexible search and advanced filter system.
- “Tokens” — allows to transfer, issue, lease tokens.
- “Contracts” — provides tools for publishing and calling docker contracts. Contracts are available for publishing from the repository, the address of which was specified when the client was built.
- “Enter Data” — allows sending data transactions and files from the interface.
- “Settings” — allows managing permissions for user actions in the blockchain.

The client supports the following browsers:

- Google Chrome.
- Mozilla Firefox.

- Opera.
- Apple Safari.
- Microsoft Edge.

If the client web interface does not work properly, or if you see any errors during loading pages, please, update your browser to the latest version.

Data

This section contains information about blockchain transactions. For information, use the filter and the search string to specify the transaction fields to search for.

Available transaction filters:

- All transactions - displays of all transactions.
- Data transactions - displays of the data transactions.
- Tokens - a selection of transactions with tokens. When this value is selected, an additional option of contextual filtering by types of token operations (for example, transfer, lease or issue of tokens) appears.
- Permissions - a transactions selection by operations with aliases and by user permissions. When selected, context filters are available by permission type (for example, mining, contract publishing, or access control).
- Groups - a selection of privacy data access groups transactions. When this value is selected, an additional option of contextual filtering by operation types (for example, a creation or an update of the access group) appears.
- Contracts - a selection of the contracts transactions. When this value is selected, an additional option of contextual filtering by contracts types (for example, Docker or RIDE) appears.
- Unconfirmed transactions - a selection of the unconfirmed transactions.
- Users - users info. When this value is selected, an additional option of contextual filtering by permissions types (for example, mining, publish smart-contract or access control) appears.

Tokens

This section shows the balance of authorized account. Allows transferring tokens to other network participants, transfer tokens for lease and manage tokens. Token management requires the “Token Management” permission.

Contracts

The section displays information on existing contracts in the network and allows you to run the selected contracts. You can use the search string with transaction parameters for the filtration. Contract publishing requires the “contract-developer” role.

Data transactions

The section allows to create data transactions and view information about existing data transactions.

Settings

The section contains basic information about the user’s account (public and private keys, secret phrase), also the current version of the client and allows you to change the language of the interface. Also you can add permissions to another users. This option requires the “permissioner” role.

10.2 Data Preparation Service

This service aggregates data from a blockchain into a relational database and provides an API to access that data. Service features are designed to meet the needs of the Waves Enterprise client. Specifying parameters are available for requests.

Deploy your client and node using the delivery set for service usage. Currently, access to the Data Preparation Service API is limited in the public network. The data service REST API is represented in the *Data service REST API* service.

BLOCKS, TRANSACTIONS, MESSAGES

11.1 Blocks

This module contains the structure of block storage in the Waves Enterprise blockchain.

Field order number	Field	Type	Field size in bytes
1	Version (0x02 for Genesis block, 0x03 for common block)	Byte	1
2	Timestamp	Long	8
3	Parent block signature	Bytes	64
4	Consensus block length (always 40 bytes)	Int	4
5	Base target	Long	8
6	Generation signature*	Bytes	32
7	Transactions block length (N)	Int	4
8	Transaction #1 bytes	Bytes	M1
...
8 + (K - 1)	Transaction #K bytes	Bytes	MK
9 + (K - 1)	Generator's public key	Bytes	32
10 + (K - 1)	Block's signature	Bytes	64

Generation signature is calculated based on the hash (Blake2b256) of the following fields:

Field order number	Field	Type	Field size in bytes
1	Previous block's generation signature	Bytes	32
2	Generator's public key	Bytes	32

The block signature is calculated based on the following data:

Field order number	Field	Type	Field size in bytes
1	Version (0x02 for Genesis block,, 0x03 for common block)	Byte	1
2	Timestamp	Long	8
3	Parent block signature	Bytes	64
4	Consensus block length (always 40 bytes)	Int	4
5	Base target	Long	8
6	Generation signature*	Bytes	32
7	Transactions block length (N)	Int	4
8	Transaction #1 bytes	Bytes	M1
...
8 + (K - 1)	Transaction #K bytes	Bytes	MK
9 + (K - 1)	Generator's public key	Bytes	32

11.2 Transactions

In this section we can see the structure of transaction storage in the blockchain platform of Waves Enterprise. For some types of transactions, versioning is introduced.

Important: All transactions use the `timestamp` field containing a time stamp in the **Unix Timestamp** format in milliseconds.

Table 1: Transaction types

№	Transaction type	Fee	Description
1	<i>Genesis transaction</i>	no fee	Initial binding of the balance to the addresses of nodes created at the start of the blockchain
3	<i>Issue Transaction</i>	1WEST	Tokens issue
4	<i>Transfer Transaction</i>	0.01WEST	Tokens transfer
5	<i>Reissue Transaction</i>	1WEST	Tokens reissue
6	<i>Burn Transaction</i>	0.05WEST	Tokens burn
8	<i>Lease Transaction</i>	0.01WEST	Tokens lease
9	<i>Lease Cancel Transaction</i>	0.01WEST	Cancel of the tokens lease
10	<i>Create Alias Transaction</i>	1WEST	Alias creation
11	<i>MassTransfer Transaction</i>	0.05WEST	Mass tokens transfer. Minimum commission is specified
12	<i>Data Transaction</i>	0.05WEST	Transaction with the data in the key-value pairs format. Minimum commission is specified
13	<i>SetScript Transaction</i>	0.5WEST	Transaction which is binding a script with a RIDE contract to an account
15	<i>SetAssetScript</i>	1WEST	Transaction which is binding a script with a RIDE contract to an asset
101	<i>Genesis Permission Transaction</i>	no fee	Assignment of the first network administrator for further distribution of rights
102	<i>Permission Transaction</i>	0.01WEST	Issuance/withdrawal of rights from the account
103	<i>CreateContract Transaction</i>	1WEST	Docker-contract creation
104	<i>CallContract Transaction</i>	0.1WEST	Docker-contract call
105	<i>ExecutedContract Transaction</i>	no fee	Docker-contract execution
106	<i>DisableContract Transaction</i>	0.1WEST	Docker-contract disable
107	<i>UpdateContract Transaction</i>	1WEST	Docker-contract update
110	<i>GenesisRegisterNode Transaction</i>	no fee	Node registration in the genesis block with the blockchain start
111	<i>RegisterNode Transaction</i>	0.01WEST	TA new node registration
112	<i>CreatePolicy Transaction</i>	1WEST	Access group creation
113	<i>UpdatePolicy Transaction</i>	0.5WEST	Update the access group
114	<i>PolicyDataHash Transaction</i>	0.05WEST	TA data hash sending to the net

11.2.1 1. Genesis transaction

Field	Broadcasted JSON	Blockchain state	Type
type	•	•	Byte
id	•		Byte
fee	•		Long
timestamp	•	•	Long
signature	•		ByteStr
recipient	•	•	ByteStr
amount	•	•	Long
height	•		

11.2.2 3. Issue Transaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	•	•	•	Byte
id		•		Byte
sender	•	•		PublicKeyAccount
sender's public key		•	•	PublicKeyAccount
fee	•	•	•	Long
timestamp	• (opt)	•	•	Long
proofs		•	•	List[ByteStr]
version	•	•	•	Byte
assetId		•		ByteStr
name	•	•	•	Array[Byte]
quantity	•	•	•	Long
reissuable	•	•	•	Boolean
decimals	•	•	•	Byte
description	•	•	•	Array[Byte]
chainId		•	•	Byte
script	• (opt)	•	•	Bytes
password	• (opt)			String
height		•		

JSON to sign

```
{
  "type": 3,
```

(continues on next page)

(continued from previous page)

```
"version":2,  
"name": "Test Asset 1",  
"quantity": 100000000000,  
"description": "Some description",  
"sender": "3FSCKyfFo3566zwiJjSFLBwKvd826KXUaqR",  
"password": "",  
"decimals": 8,  
"reissuable": true,  
"fee": 100000000  
}
```

Broadcasted JSON

```
{  
  "type": 3,  
  "id": "DnK5Xfi2wXUJx9BjK9X6ZpFdTLdq2GtWH9pWrcxcmrhB",  
  "sender": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",  
  "senderPublicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",  
  "fee": 100000000,  
  "timestamp": 1549378509516,  
  "proofs": [  
↔ "NqZGcbcQ82FZrPh6aCEjuo9nNnkPTvyhrNq329YWydaYcZTywXUwDxFaknTMEGuFrEndCjXBtrueLWaqbJhpeiG" ],  
  "version": 2,  
  "assetId": "DnK5Xfi2wXUJx9BjK9X6ZpFdTLdq2GtWH9pWrcxcmrhB",  
  "name": "Token Name",  
  "quantity": 10000,  
  "reissuable": true,  
  "decimals": 2,  
  "description": "SmarToken",  
  "chainId": 84,  
  "script": "base64:AQa3b8tH",  
  "height": 60719  
},
```

11.2.3 4. Transfer Transaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	•	•	•	Byte
id		•		Byte
sender	•	•		PublicKeyAccount
sender's public key		•	•	PublicKeyAccount
fee	•	•	•	Long
timestamp	• (opt)	•	•	Long
proofs		•	•	List[ByteStr]
version	•	•	•	Byte
recipient	•	•	•	ByteStr
assetId	• (opt)	•	•	ByteStr
fee assetId	• (opt)	•	•	Bytes
amount	•	•	•	Long
attachment	• (opt)	•	•	Bytes
password	• (opt)			String
height		•		

JSON to sign

```
{
  "type": 4,
  "version": 2,
  "sender": "3M6dRZXaJY9oMA3fJKhMALyYKt13D1aimZX",
  "password": "",
  "recipient": "3M6dRZXaJY9oMA3fJKhMALyYKt13D1aimZX",
  "amount": 4000000000,
  "fee": 100000
}
```

Broadcasted JSON

```
{
  "senderPublicKey": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
  "amount": 200000000,
  "fee": 100000,
  "type": 4,
  "version": 2,
  "attachment": "3uaRTtZ3taQtRSmquqeC1DniK3Dv",
  "sender": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
  "feeAssetId": null,
  "proofs": [
    "2hRxJ2876CdJ498UCpErNfDSYdt2mTK4XUnmZNgZiq63RupJs5WTrAqR46c4rLQdq4toBZk2tSYCeAQWEQyi72U6"
  ],
  "assetId": null,
  "recipient": "3GPtj5osoYqHpyfmsFv7BMiyKsVzbG1ykfL",
  "id": "757aQzJiQZRfVRuJNnP3L1d369H2oTjUEazwtYxGngCd",
  "timestamp": 1558952680800
}
```

11.2.4 5. Reissue Transaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	•	•	•	Byte
id		•		Byte
sender	•	•		PublicKeyAccount
sender's public key		•	•	PublicKeyAccount
fee	•	•	•	Long
timestamp	• (opt)	•	•	Long
proofs		•	•	List[ByteStr]
version	•	•	•	Byte
chainId		•	•	Byte
assetId	•	•	•	ByteStr
quantity	•	•	•	Long
reissuable	•	•	•	Boolean
password	• (opt)			String
height				

JSON to sign

```
{
  "type": 5,
  "version": 2,
  "quantity": 10000,
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "password": "",
  "assetId": "7bE3JPwZC3QcN9edctFrLAKYysjfMEk1SDjZx5gitSGg",
  "reissuable": true,
  "fee": 100000001
}
```

Broadcasted JSON

```
{
  "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
```

(continues on next page)

(continued from previous page)

```

"quantity": 10000,
"sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
"chainId": 84,
"proofs": [
↪ "3gmgGM6rYpxuuR5QvJkugPsERG7yWYF7JN6QzpUGJwT8Lw6SUHkzzk8R22A7cGQz7TQQ5NifKxvAQzwPyDQbwmBg" ],
"assetId": "7bE3JPwZC3QcN9edctFrLAKYysjfMEk1SDjZx5gitSGg",
"fee": 100000001,
"id": "GsNvk15Vu4kqtRmMSpYW21WzgJpZrLBwjCREHWuwnvh5",
"type": 5,
"version": 2,
"reissuable": true,
"timestamp": 1551447859299,
"height": 1190
}

```

11.2.5 6. Burn Transaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	•	•	•	Byte
id		•		Byte
sender	•	•		PublicKeyAccount
sender's public key		•	•	PublicKeyAccount
fee	•	•	•	Long
timestamp	• (opt)	•	•	Long
proofs		•	•	List[ByteStr]
version	•	•	•	Byte
chainId		•	•	Byte
assetId	•	•	•	ByteStr
quantity	•		•	Long
amount		•		Long
password	• (opt)			String
height				

JSON to sign

```
{
  "type": 6,
  "version": 2,
  "sender": "3MtrNP7AkTRuBhX4CBti6iT21pQpEnmHtyw",
  "password": "",
  "assetId": "7bE3JPwZC3QcN9edctFrLAKYysjfMEk1SDjZx5gitSGg",
  "quantity": 1000,
  "fee": 100000,
  "attachment": "string"
}
```

Broadcasted JSON

```
{
  "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
  "amount": 1000,
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "chainId": 84,
  "proofs": [
    ↪ "kzTwsNXjJkzk6dpFFZZXyeimYo6iLTVbCnCXBD4xBtyrNjysPqZfGKk9NdJUTP3xeAPhtEgU9hsdwzRVo1hKMgS" ],
  "assetId": "7bE3JPwZC3QcN9edctFrLAKYysjfMEk1SDjZx5gitSGg",
  "fee": 100000,
  "id": "3yd2HZq7sgun7GakisLH88UeKcpYMUEL4sy57aprAN5E",
  "type": 6,
  "version": 2,
  "timestamp": 1551448489758,
  "height": 1190
}
```

11.2.6 8. Lease Transaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	•	•	•	Byte
id		•		Byte
sender	•	•		PublicKeyAccount
sender's public key		•	•	PublicKeyAccount
fee	•	•	•	Long
timestamp	• (opt)	•	•	Long
proofs		•	•	List[ByteStr]
version	•	•	•	Byte
amount	•	•	•	Long
recipient	•	•	•	ByteStr
status		•		
password	• (opt)			String
height		•		

JSON to sign

```
{
  "type": 8,
  "version": 2,
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "password": "",
  "recipient": "3N1ksBqc6uSksdiYjCzMtvEpiHhS1JjkbPh",
  "amount": 1000,
  "fee": 100000
}
```

Broadcasted JSON

```
{
  "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
  "amount": 1000,
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
```

(continues on next page)

(continued from previous page)

```

"proofs": [
↪ "5jvmWkMU89HnxXFXNAd9X41zmiB5fSGoXMirsaJ9tNeyiCAJmjm7MR48g789VucckQw2UExaVXfhdsEBuUrchvrq" ],
"fee": 100000,
"recipient": "3N1ksBqc6uSksdiYjCzMtvEpiHhS1JjkbPh",
"id": "6Tn7ir9MycHW6Gq2F2dGok2stokSwXJadPh4hW8eZ8Sp",
"type": 8,
"version": 2,
"timestamp": 1551449299545,
"height": 1190
}
    
```

11.2.7 9. Lease Cancel Transaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	•	•	•	Byte
id		•		Byte
sender	•	•		PublicKeyAccount
sender's public key		•	•	PublicKeyAccount
fee	•	•	•	Long
timestamp	• (opt)	•	•	Long
proofs		•	•	List[ByteStr]
version	•	•	•	Byte
chainId		•	•	Byte
leaseId	• (txId)	•	•	Byte
leaseId		•		
password	• (opt)			String
height		•		

JSON to sign

```

{
  "type": 9,
    
```

(continues on next page)

(continued from previous page)

```
"version": 2,  
"fee": 100000,  
"sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",  
"password": "",  
"txId": "6Tn7ir9MycHW6Gq2F2dGok2stokSwXJadPh4hW8eZ8Sp"  
}
```

Broadcasted JSON

```
{  
  "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",  
  "leaseId": "6Tn7ir9MycHW6Gq2F2dGok2stokSwXJadPh4hW8eZ8Sp",  
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",  
  "chainId": 84,  
  "proofs": [  
↪ "2Gns72hraH5yay3eiWeyHQEA1wTqiiAztaLjHinEYX91FEv62HFW38Hq89GnsEJFHUvo9KHYtBBrb8hgTA9wN7DM" ],  
  "fee": 100000,  
  "id": "9vhxB2ZDQcqiuhQbCPnAoPBLuir727qgJhFeBNmPwmu",  
  "type": 9,  
  "version": 2,  
  "timestamp": 1551449835205,  
  "height": 1190  
}
```

11.2.8 10. Create Alias Transaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	•	•	•	Byte
id		•		Byte
sender	•	•		PublicKeyAccount
sender's public key		•	•	PublicKeyAccount
fee	•	•	•	Long
timestamp	• (opt)	•	•	Long
proofs		•	•	List[ByteStr]
version	•	•	•	Byte
alias	•	•	•	Bytes
password	• (opt)			String
height		•		

JSON to sign

```
{
  "type": 10,
  "version": 2,
  "fee": 100000,
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "password": "",
  "alias": "hodler"
}
```

Broadcasted JSON

```
{
  "type": 10,
  "id": "DJTtaiMpb7eLuPW5GcE4ndeE8jWsWPjx8gPYmbZPJjpag",
  "sender": "3N65yEf31oJBZUvpu4LCo7n8D73juFtheUJ",
  "senderPublicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "fee": 0,
  "timestamp": 1549290335781,
  "signature":
  ↪ "2qYepod9DhpxVad1yQDbv1QzU4KLKcbjjdtGY7De2272K76nbQfaXsRnyd31hUE8bhvLjjpHRdtoLVzbDDzRZYEY",
  "proofs": [
  ↪ "2qYepod9DhpxVad1yQDbv1QzU4KLKcbjjdtGY7De2272K76nbQfaXsRnyd31hUE8bhvLjjpHRdtoLVzbDDzRZYEY" ]
}
```

(continues on next page)

(continued from previous page)

```

"version": 1,
"alias": "testperson4",
"height": 59245
}
    
```

11.2.9 11. MassTransfer Transaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	•	•	•	Byte
id		•		Byte
sender	•	•		PublicKeyAccount
sender's public key		•	•	PublicKeyAccount
fee	•	•	•	Long
timestamp	• (opt)	•	•	Long
proofs		•	•	List[ByteStr]
version	•	•	•	Byte
assetId	• (opt)	•	•	ByteStr
attachment	• (opt)	•	•	
number of transfers	•	•	•	List[Transfer]
transferCount		•	•	
totalAmount		•		
password	• (opt)			String
height		•		

JSON to sign

```

{
  "type": 11,
    
```

(continues on next page)

(continued from previous page)

```

"sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
"password": "",
"fee": 2000000,
"version": 1,
"transfers":
[
  { "recipient": "3MtHszoTn399NfsH3v5foeEXRRrchEVtTRB", "amount": 100000 },
  { "recipient": "3N7BA6J9VUBfBRutuMyjF4yKTUEtrRFfHMc", "amount": 100000 }
],
"height": 1190
}

```

Broadcasted JSON

```

{
  "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
  "fee": 2000000,
  "type": 11,
  "transferCount": 2,
  "version": 1,
  "totalAmount": 200000,
  "attachment": "",
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "proofs": [
    ↪ "2gWpMwdgZCjbygCX5US3aAfftKtGPRSK3aWGJ6RDnWJf9hend5sBFAgY6u3Mp4jN8cqwaJ5o8qrKNedGN5CPN1GZ" ],
  "assetId": null,
  "transfers":
  [
    {
      "recipient": "3MtHszoTn399NfsH3v5foeEXRRrchEVtTRB",
      "amount": 100000
    },
    {
      "recipient": "3N7BA6J9VUBfBRutuMyjF4yKTUEtrRFfHMc",
      "amount": 100000
    }
  ],
  "id": "D9jUSHHcJqVAvkFMiRfDBhQbUzoSfQqd9cjaunMmtjdu",
  "timestamp": 1551450279637
}

```

11.2.10 12. Data Transaction

Warning: The transaction has limits:

1. "key": "value" pairs count no more than 100,

```

"data": [
  {
    "key": "objectId",
    "type": "string",
    "value": "obj:123:1234"
  }, {...}
]

```

2. The byte composition of the signed transaction should not exceed more than 150 KB.

Hint: You do not need to specify the `senderPublicKey` parameter if you are signing a transaction where the author and the sender are the same.

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type	Size (Bytes)
type	•	•	•	Byte	1
id		•		Byte	1
sender	•	•		PublicKeyAccount	3264
sender's public key	• (opt)	•	•	PublicKeyAccount	3264
fee	•	•	•	Long	8
timestamp	• (opt)	•	•	Long	8
proofs		•	•	List[ByteStr]	32767
version	•	•		Byte	1
authorPublicKey		•	•	PublicKeyAccount	3264
author	•	•			3264
data	•	•	•		3264
password	• (opt)			String	32767
height		•			8

JSON to sign

```
{
  "type": 12,
  "version": 1,
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "password": "",
  "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
  "author": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "data": [
```

(continues on next page)

(continued from previous page)

```
{
  "key": "objectId",
  "type": "string",
  "value": "obj:123:1234"
},
"fee": 100000
}
```

Broadcasted JSON

```
{
"senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
"authorPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
"data":
[
  {
    "type": "string",
    "value": "obj:123:1234",
    "key": "objectId"
  }
],
"sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
"proofs": [
  ↪ "2T7WQm5XW8cFHfiFkdDEic9oNiT7aFiH3TyKkARERopr1VJvzRKqHAVnQ3eiYZ3uYN8uQnPopQEH4XV8z5SgSwsf" ],
"author": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
"fee": 100000,
"id": "7dMMCQNTusahZ7DWtNGjCwAhRYpjaH1hsepRmbpn2BkD",
"type": 12,
"version": 1,
"timestamp": 1551680510183
}
```

11.2.11 13. SetScript Transaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	•	•	•	Byte
id		•		Byte
sender	•	•		PublicKeyAccount
sender's public key		•	•	PublicKeyAccount
fee	•	•	•	Long
timestamp	• (opt)	•	•	Long
proofs		•	•	List[ByteStr]
chainId		•	•	Byte
version	•	•	•	Byte
script	• (opt)	•	•	Bytes
name	•	•	•	Array[Byte]
description	• (opt)	•	•	Array[Byte]
password	• (opt)			String
height		•		

JSON to sign

```
{
  "type": 13,
  "version": 1,
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "password": "",
  "fee": 1000000,
  "name": "faucet",
  "script": "base64:AQQAAAAHJG1hdGNoMAUAAAACdHgG+RXSzQ=="
}
```

Broadcasted JSON


```
{
  "type": 13,
  "id": "HPDypnQJHJskN8kwszF8rck3E5tQiuim1fEN42w6PLmt",
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
  "fee": 1000000,
  "timestamp": 1545986757233,
  "proofs": [
    ↪ "2QiGYS2dqh8QyN7Vu2tAYaioX5WM6rTSDPGbt4zrWS7QKTzobjmR2kjppvGNj4tDPsYPbcDunqBaqhaudLyMeGFgG" ],
  "chainId": 84,
  "version": 1,
  "script": "base64:AQQAAAAHJG1hdGNoMAUAAAAACdHgG+RXSszQ==",
  "name": "faucet",
  "description": "",
  "height": 3805
}
```

11.2.12 15. SetAssetScriptTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	•	•	•	Byte
id		•		Byte
sender	•	•		PublicKeyAccount
sender's public key		•	•	PublicKeyAccount
fee	•	•	•	Long
timestamp	• (opt)	•	•	Long
proofs		•	•	List[ByteStr]
version	•	•	•	Byte
chainId		•	•	Byte
assetId	•	•	•	ByteStr
script	• (opt)	•	•	Bytes
password	• (opt)			String
height		•		

JSON to sign

```
{
  "type": 15,
  "version": 1,
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "password": "",
  "fee": 100000000,
  "script": "base64:AQQAAAAHJG1hdGNoMAUAAAACdHgG+RXSszQ==",
  "assetId": "7bE3JPwZC3QcN9edctFrLAKYysjfMEk1SDjZx5gitSGg"
}
```

Broadcasted JSON

```
{
  "type": 15,
  "id": "CQpEM9AEDvgxKfgWlH2HxE82iAzpXrtqsDDcgZGPAF9J",
  "sender": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",
  "senderPublicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "fee": 100000000,
  "timestamp": 1549448710502,
  "proofs": [
    ↪ "64eodpuXQjaKQQ4GJBaBrqiBtmkjSxseKC97gn6EwB5kZtMr18mAUHPRkZaHJeJxaDyLzGEZKqhYoUknWfNhXnkf" ],
  "version": 1,
  "chainId": 84,
  "assetId": "DnK5Xfi2wXUJx9BjK9X6ZpFdTLdq2GtWH9pWrcxcmrhB",
  "script": "base64:AQQAAAAHJG1hdGNoMAUAAAACdHgG+RXSszQ==",
  "height": 61895
}
```

11.2.13 101. GenesisPermitTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	•	•	Byte	
id	•		Byte	
fee	•		Long	
timestamp	•	•	Long	
signature	•		ByteStr	
target	•	•	ByteStr	
role	•	•	String	
height				

11.2.14 102. PermissionTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	•	•	•	Byte
id		•		Byte
sender	•	•		PublicKeyAccount
sender's public key		•	•	PublicKeyAccount
fee		•		Long
timestamp	• (opt)	•	•	Long
proofs		•	•	List[ByteStr]
version			•	Byte
target	•	•	•	ByteStr
PermissionOp			•	PermissionOp
opType	•	•		String
role	•	•		String
dueTimestamp	• (opt)	•		Option[Long]
password	• (opt)			String
height		•		

JSON to sign

```
{
  "type": 102,
  "sender": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
  "password": "",
  "senderPublicKey": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
  "fee": 0,
  "proofs": [],
  "target": "3GPtj5osoYqHpyfmsFv7BMiyKsVzbG1ykfL",
  "opType": "add",
  "role": "contract_developer",

```

(continues on next page)

(continued from previous page)

```
"dueTimestamp":null
}
```

Broadcasted JSON

```
{
  "senderPublicKey": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
  "role": "contract_developer",
  "sender": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
  "proofs": [
    "5ABJCRTKGo6jmDZCRWcLQc257CCeczmcjmtfJmbBE7TP3KsVkwvisH9kEkfYPckVCzEMKZTCd3LKAPcN8o4Git3j"
  ],
  "fee": 0,
  "opType": "add",
  "id": "8zVUH7nsDCcpwyfxiq8DCTgqL7Q23FW1KWepB9EZcFG6",
  "type": 102,
  "dueTimestamp": null,
  "timestamp": 1559048837487,
  "target": "3GPtj5osoYqHpyfmsFv7BMiyKsVzbG1ykfL"
}
```

11.2.15 103. CreateContractTransaction

Warning: The byte composition of the signed transaction should not exceed more than 150 KB.

The `contractVersion` field specifies the contract version, the 1 value is for the new contract, and the 2 value is for the updated contract. The contract is updated by using the `107` transaction. When you create a contract, the `104` transaction is automatically created, this transaction is calling the contract to validate it. If the contract fails or runs with error, transactions 103 and 104 will be discarded and will not fall into the block.

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type	Size(Bytes)
type	•	•	•	Byte	1
id		•		Byte	1
sender	•	•		PublicKeyAccount	3264
sender's public key		•	•	PublicKeyAccount	3264
password	• (opt)			String	32767
fee	•	•	•	Long	8
timestamp	• (opt)	•	•	Long	8
proofs		•	•	List[ByteStr]	32767
version		•	•	Byte	1
contractVersion	•	•	•	Byte	1
image	•	•	•	Array[Bytes]	32767
imageHash	•	•	•	Array[Bytes]	32767
contractName	•	•	•	Array[Bytes]	32767
params	•	•	•	List[DataEntry[...]]	32767
height		•			8

JSON to sign

```
{
  "fee": 100000000,
  "image": "stateful-increment-contract:latest",
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
  "contractName": "stateful-increment-contract",
  "sender": "3PudkbvjV1nPj1TkuuRahh4sGdgfr4YAUUV2",
  "password": "",
  "params": [],
  "type": 103,
  "version": 1
  "contractVersion": 1
}
```

Broadcasted JSON

```
{
  "type": 103,
  "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
  "sender": "3N3YTjtNwn8XUJ8ptGKbPuEFNa9GFnhqew",
  "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsJMVT2M",
  "fee": 500000,
  "timestamp": 1550591678479,
  "proofs": [
    ↪ "yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxfj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
  "version": 1,
  "image": "stateful-increment-contract:latest",
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
  "contractName": "stateful-increment-contract",
  "params": [],
  "height": 1619
}
```

11.2.16 104. CallContractTransaction

Warning: The byte composition of the signed transaction should not exceed more than 150 KB.

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type	Size(Bytes)
type	•	•	•	Byte	1
id		•		Byte	1
sender	•	•		PublicKeyAccount	3264
sender's public key		•	•	PublicKeyAccount	3264
fee	•	•	•	Long	8
timestamp	• (opt)	•	•	Long	8
proofs		•	•	List[ByteStr]	32767
version		•	•	Byte	1
contractId	•	•	•	ByteStr	32767
params	•	•	•	List[DataEntry[...]]	32767
height		•			8
password	• (opt)			String	32767

JSON to sign

```
{
  "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
  "fee": 10,
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
  "password": "",
  "type": 104,
  "params":
  [
    {
      "type": "integer",
      "key": "a",
      "value": 1
    },
    {
      "type": "integer",
      "key": "b",
      "value": 100
    }
  ],
}
```

(continues on next page)

(continued from previous page)

```
}
  "version": 1
}
```

Broadcasted JSON

```
{
  "type": 104,
  "id": "9fBrL2n5TN473g1gNfoZqaAqAsAJCuHRHYxZpLexL3VP",
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
  "senderPublicKey": "2YvzcVLRqLCqouVrFZynjfoTEuPNV9GrdauNpgdWXLsq",
  "fee": 10,
  "timestamp": 1549365736923,
  "proofs": [
    ↪ "2q4cTBhDkEDkFxr7iYaHPAv1dzaKo5rDaTxPF5VHryyYTxTPvN9Wb3YrsDYixKiUPXBnAyXzEcnKPFRCW9xVp4v" ],
  "version": 1,
  "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
  "params":
  [
    {
      "key": "a",
      "type": "integer",
      "value": 1
    },
    {
      "key": "b",
      "type": "integer",
      "value": 100
    }
  ]
}
```

11.2.17 105. ExecutedContractTransaction

Warning: The byte composition of the signed transaction should not exceed more than 150 KB.

Field	Broadcasted JSON	Blockchain state	Type
type	•	•	Byte
id	•		Byte
sender	•		PublicKeyAccount
sender's public key	•	•	PublicKeyAccount
fee	•		Long
timestamp	•	•	Long
proofs	•	•	List[ByteStr]
version	•	•	Byte
tx	•	•	ExecutableTransaction
results	•	•	List[DataEntry[_]]
height	•		
password	• (opt)		String

Broadcasted JSON

```
{
  "type": 105,
  "id": "38GmSVC5s8Sjeybzfe9RQ6p1Mb6ajb8LYJDcep8G8Umj",
  "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
  "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsJMVT2M",
  "password": "",
  "fee": 500000,
  "timestamp": 1550591780234,
  "proofs": [
    ↪ "5whBipAWQgFvm3myNZe6GDd9Ky8199C9qNxBHQDNmVAUJW9gLf7t9LBQDi68CKT57dzmnP JpJkrwKh2HBSwUer6" ],
  "version": 1,
  "tx": {
    {
      "type": 103,
      "id": "ULc9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLZVj4Ky",
      "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
      "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsJMVT2M",
      "fee": 500000,
      "timestamp": 1550591678479,
      "proofs": [
        ↪ "yeCRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxfj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],

```

(continues on next page)

(continued from previous page)

```

    "version": 1,
    "image": "stateful-increment-contract:latest",
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
    "contractName": "stateful-increment-contract",
    "params": [],
    "height": 1619
  },
  "results": [],
  "height": 1619
}

```

11.2.18 106. DisableContractTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	•	•	•	Byte
id		•		Byte
sender	•	•		PublicKeyAccount
sender's public key		•	•	PublicKeyAccount
fee	•	•	•	Long
timestamp	• (opt)	•	•	Long
proofs		•	•	List[ByteStr]
version		•	•	Byte
contractId	•	•	•	ByteStr
height		•		
password	• (opt)			String

JSON to sign

```

{
  "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
  "password": "",
  "contractId": "Fz3wqAWWcPMT4M1q6H7crLKtToFJvbeLSvqjaU4ZwMpg",
  "fee": 500000,
  "timestamp": 1549474811381,
  "type": 106
}

```

Broadcasted JSON

```
{
  "type": 106,
  "id": "8Nw34YbosEVhCx18pd81HqYac4C2pGjyLKck8NhSoGYH",
  "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
  "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsJMVT2M",
  "fee": 500000,
  "timestamp": 1549474811381,
  "proofs": [
    ↪ "5GqPQkuRvG6LPXgPoCr9FogAdmhAaMbyFb5UfjQPUKdSc6BLuQSZ75LAWix1ok2Z6PC5ezPpjzqnr15i3RQmaEc" ],
  "version": 1,
  "contractId": "Fz3wqAwwcPMT4M1q6H7crLKtToFJvbeLSvqjaU4ZwMpg",
  "height": 1632
}
```

11.2.19 107. UpdateContractTransaction

Warning:

- The byte composition of the signed transaction should not exceed more than 150 KB.

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type	Size(Bytes)
type	•	•	•	Byte	1
id		•		Byte	1
sender	•	•		PublicKeyAccount	3264
sender's public key		•	•	PublicKeyAccount	3264
image	•	•	•	Array[Bytes]	32767
imageHash	•	•	•	Array[Bytes]	32767
fee	•	•	•	Long	8
timestamp		•	•	Long	8
proofs		•	•	List[ByteStr]	32767
version	•	•	•	Byte	1
contractId	•	•	•	ByteStr	32767
height		•			8
password	• (opt)			String	32767

JSON to sign

```
{
  "image" : "registry.wvservices.com/we-sc/tdm-increment3:1028.1",
  "sender" : "3Mxxz9pBYS5fJMARJNQmzYUHxiWAtvMzSRT",
  "password": "keyPass",
  "fee" : 100000000,
  "contractId" : "EnsihTUHSNAB9RcWXJbiWT98X3hYtCw3SBzK8nHQRcWA",
  "imageHash" : "0e5d280b9acf6efd8000184ad008757bb967b5266e9ebf476031fad1488c86a3",
  "type" : 107,
  "version" : 1
}
```

Broadcasted JSON

```
{
  "senderPublicKey":
  ↪ "5qBRDm74WKR5xK7LPs8vCy9QjzzqK4KCb8PL36fm55S3kEi2XZETHFgMgp3D13AwgE8bBkYrzvEvQZuabMfEyJwW",
  "tx":
```

(continues on next page)

(continued from previous page)

```

{
  "senderPublicKey":
  ↪ "5qBRDm74WKR5xK7LPs8vCy9QjzzqK4KCb8PL36fm55S3kEi2XZETHFgMgp3D13AwgE8bBkYrzvEvQZuabMfEyJwW",
  "image": "registry.wservices.com/we-sc/tdm-increment3:1028.1",
  "sender": "3Mxxz9pBYS5fJMARJNQmzYUHxiWAtvMzSRT",
  "proofs": [
  ↪ "3tNsTyteeZrxEbVSv5zPT6dr247nXsVWR5v7Khx8spypgZQUdorCQZV2guTomutUTCyhxJUjNkQW4VmSgbCtgm1Z"],
  "fee": 0,
  "contractId": "EnsihTUHSNAB9RcWXJbiWT98X3hYtCw3SBzK8nHQRcWA",
  "id": "HdZdhXVveMT1vYzGTviCoGQU3aH6ZS3YtFpYujWeGCH6",
  "imageHash": "17d72ca20bf9393eb4f4496fa2b8aa002e851908b77af1d5db6abc9b8eae0217",
  "type": 107, "version": 1, "timestamp": 1572355661572},
  "sender": "3HfRBedCpWi3vEzFSKEZDFXkyNWbWLWQmmG",
  "proofs": [
  ↪ "28ADV8miUVN5EFjhqeFj6MADSXYjbxA3TsxSwFVs18jXAsHVAbczvnyoUSaYJsJRnmaWgXbpbduccRxpKGTs6tro"],
  "fee": 0, "id": "7niVY8mjzeKqLBePvhTxFRfLu7BmcwVfqaqtbWAN8AA2",
  "type": 105,
  "version": 1,
  "results": [],
  "timestamp": 1572355666866
}
}

```

11.2.20 110. GenesisRegisterNodeTransaction

Field	Broadcasted JSON	Blockchain state	Type
type	•	•	Byte
id	•		Byte
fee	•		Long
timestamp	•	•	Long
signature	•		Bytes
version		•	Byte
targetPubKey	•	•	
height	•		

11.2.21 111. RegisterNodeTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	•	•	•	Byte
id		•		Byte
sender	•	•		PublicKeyAccount
sender's public key		•	•	PublicKeyAccount
fee	•	•		Long
timestamp	• (opt)	•	•	Long
proofs		•	•	List[ByteStr]
version			•	Byte
targetPubKey	•	•	•	PublicKeyAccount
nodeName	•	•	•	String
opType	•	•	•	
height		•		
password	• (opt)			String

JSON to sign

```
{
  "type": 111,
  "opType": "add",
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUGeytUUz",
  "password": "",
  "targetPubKey": "apgJP9atQccdBPAgJPwH3NBVqYXrapgJP9atQccdBPAgJPwHapgJP9atQccdBPAgJPwHDKkh6A8",
  "nodeName": "Node #1",
  "fee": 500000,
  "timestamp": 111111111
}
```

11.2.22 112. CreatePolicyTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	•	•	•	Byte
id		•	•	Byte
sender	•	•	•	PublicKeyAccount
sender's public key		•	•	PublicKeyAccount
policyName	•	•	•	String
recipients	•	•	•	Array[Byte]
owners	•	•	•	Array[Byte]
fee	•	•	•	Long
timestamp	• (opt)	•	•	Long
proofs		•	•	List[ByteStr]
height			•	Long
description	•	•	•	String
password	• (opt)			String

JSON to sign

```
{
  "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SsqJycqv8d",
  "policyName": "Policy# 7777",
  "password": "sfgKYBFCF@#$fsdf()*%",
  "recipients": [
    "3NkZd8Xd4KsuPiNVsuphRNCZE3SsqJycqv8d",
    "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
    "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
    "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
    "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx"
  ],
  "fee": 15000000,
  "description": "Buy bitcoin by 1c",
  "owners": [
    "3NkZd8Xd4KsuPiNVsuphRNCZE3SsqJycqv8d",
    "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
  ]
}
```

(continues on next page)

(continued from previous page)

```

    "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
  ],
  "type": 112
}

```

11.2.23 113. UpdatePolicyTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	•	•	•	Byte
id		•	•	Byte
sender	•	•	•	PublicKeyAccount
sender's public key		•	•	PublicKeyAccount
policyName	•	•	•	String
recipients	•	•	•	Array[Byte]
owners	•	•	•	Array[Byte]
fee	•	•	•	Long
timestamp	• (opt)	•	•	Long
proofs		•	•	List[ByteStr]
height			•	Long
opType	•	•	•	
description	•	•	•	String
password	• (opt)			String

JSON to sign

```

{
  "policyId": "7wphGbhqbmUgzun5wzggwqtViTiMdFezSa11fxRV58Lm",
  "password": "sfgKYBFCF@#$fsdf()*%",
  "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "proofs": [],

```

(continues on next page)

(continued from previous page)

```

"recipients": [
  "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
  "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
    "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
    "3NxAoohUoLsAQvxBSqjE91WK3LwWGjiiCxx",
    "3NwJfjG5RpaDfxEhkwXgwD7oX21NMFCxJHL"
],
"fee": 15000000,
"opType": "add",
"owners": [
  "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
  "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
],
"type": 113,
}

```

11.2.24 114. PolicyDataHashTransaction

When the user sends confidential data to the network using *POST /privacy/sendData*, the node automatically will create the 114 transaction.

Field	Broadcasted JSON	Blockchain state	Type
type	•	•	Byte
id	•	•	Byte
sender	•	•	PublicKeyAccount
sender's public key	•	•	PublicKeyAccount
policyId	•	•	String
dataHash	•	•	String
fee	•	•	Long
timestamp	•	•	Long
proofs	•	•	List[ByteStr]
height		•	Long

11.3 Network messages

This section describes the structure of network messages in the Waves Enterprise blockchain platform.

11.3.1 Network message

All network messages, except Handshake, are based on the following structure:

Field order number	Field	Type	Field size in bytes
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Payload	Bytes	N

Magic Bytes are 0x12, 0x34, 0x56, 0x78. Payload checksum is first 4 bytes of `_FastHash_` of Payload bytes. `FastHash` is hash function `Blake2b256(data)`.

11.3.2 Handshake message

Handshake message is intended for primary data exchange between two nodes. An authorized handshake contains the node owner's blockchain address and signature. Unsigned handshakes are not accepted.

Authorized Handshake

Field order number	Field	Type	Field size in bytes
1	HandshakeType	byte	1
2	Application name length (N)	Byte	1
3	Application name (UTF-8 encoded bytes)	Bytes	N
4	Application version major	Int	4
5	Application version minor	Int	4
6	Application version patch	Int	4
7	Consensus name length (P)	Byte	1
8	Consensus name length (UTF-8 encoded bytes)	Bytes	P
9	Node name length (M)	Byte	1
10	Node name (UTF-8 encoded bytes)	Bytes	M
12	Node nonce	Long	8
13	Declared address length (K) or 0 if no declared address was set	Int	4
14	Declared address bytes (if length is not 0)	Bytes	K
15	Peer port	Int	4
16	Node owner address	Bytes	26
17	Signature	Bytes	64

11.3.3 GetPeers message

GetPeers message is sent to request network addresses of network participants.

Field order number	Field	Type	Field size in bytes
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x01)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4

11.3.4 Peers message

Peers message is a response to a GetPeers request.

Field order number	Field	Type	Field size in bytes
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x02)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Peers count (N)	Int	4
7	Peer #1 IP address	Bytes	4
8	Peer #1 port	Int	4
...
6 + 2 * N - 1	Peer #N IP address	Bytes	4
6 + 2 * N	Peer #N port	Int	4

11.3.5 GetSignatures message

Field order number	Field	Type	Field size in bytes
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x14)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Block IDs count (N)	Int	4
7	Block #1 ID	Bytes	64
...
6 + N	Block #N ID	Bytes	64

11.3.6 Signatures message

Field order number	Field	Type	Field size in bytes
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x15)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Block signatures count (N)	Int	4
7	Block #1 signature	Bytes	64
...
6 + N	Block #N signature	Bytes	64

11.3.7 GetBlock message

Field order number	Field	Type	Field size in bytes
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x16)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Block ID	Bytes	64

11.3.8 Block message

Field order number	Field	Type	Field size in bytes
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x17)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Block bytes (N)	Bytes	N

11.3.9 Score message

Field order number	Field	Type	Field size in bytes
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x18)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Score (N bytes)	BigInt	N

11.3.10 Transaction message

Field order number	Field	Type	Field size in bytes
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x19)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Transaction (N bytes)	Bytes	N

11.3.11 Checkpoint message

Field order number	Field	Type	Field size in bytes
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x64)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Checkpoint items count (N)	Int	4
7	Checkpoint #1 height	Long	8
8	Checkpoint #1 signature	Bytes	64
...
6 + 2 * N - 1	Checkpoint #N height	Long	8
6 + 2 * N	Checkpoint #N signature	Bytes	64

SMART CONTRACTS

12.1 RIDE Smart Contracts

Smart Contract is a script that checks transactions for compliance with conditions. Scripts extend the logic of blockchain to meet your business tasks. The fee for smart contract is fixed. The script can be published for both an account and token assets issued by you.

For an account, all transactions originating from this address will be checked. An account with a published script is called a Smart Account. For token assets, all transactions with these token assets will be checked. A token asset with a published script is called a Smart Asset. Only 1 script can be assigned to one account. Accordingly, any installed script replaces the previous one, including the “default script”.

12.1.1 RIDE

The RIDE language is used for creating a script in the Waves Enterprise blockchain (about RIDE language you can read on the WAVES portal <https://docs.wavesplatform.com>). Scripts written in RIDE use the following data when checking conditions:

- Outgoing transaction details.
- Details of account on behalf of which transactions are made.
- Details of third accounts balance.
- Details of blockchain height.

The principle of the script operation is pattern matching. The script specifies transaction types and checks them for compliance with conditions under which corresponding transactions can be executed. Also, the following features are available:

- ban transaction regardless of conditions,
- permit regardless of conditions.

Operations with permissions and bans per transaction types are possible both by specifying specific transaction types and using the “everything but” principle. The script is set by the Setscript transaction, so permission, prohibition or verification for compliance with conditions must be explicitly specified.

Important: The script does not modify the transaction, it only verifies that the conditions are met.

12.1.2 Complexity of scripts

RIDE is not a Turing-complete language, which imposes limitations on the available complexity of logic. Computational complexity is constrained from the top to guarantee network performance. For complex business processes, the mechanics of which does not fit into one script, a combination of several scripts (for several addresses, respectively), or a combination of scripts for token assets and address can be used. We are actively developing RIDE features, and in the near future nested functions expanding its capabilities in terms of complexity of tasks to be implemented will appear in the language.

12.1.3 Signatures and default script

Each transaction in the blockchain has a cryptographic proof of integrity based on the signature of the transaction by the sender's private key. This also guarantees that transaction authorship is unalienable. For a better understanding of the mechanism, imagine that “by default” a script is installed on each address, which verifies the only condition for each outgoing transaction — the signature of the sender's address.

Example of a default script code:

```
sigVerify(tx.bodyBytes, tx.proofs[0], tx.senderPk)
```

The script mechanics enhances proof verification capabilities. A transaction can be signed by multiple users or other than on behalf of the address from which it was sent. This is necessary because the contract checks only transactions originating from its address. Accordingly, the user generates a transaction on behalf of the contract, signs it with his proof and successfully passes the script test.

Important: If the proof verification is not explicitly specified in your script, it is not executed. Accordingly, when the transaction body is generated manually, it is possible to send transactions on behalf of an address with a script with another address proof.

12.1.4 Account data

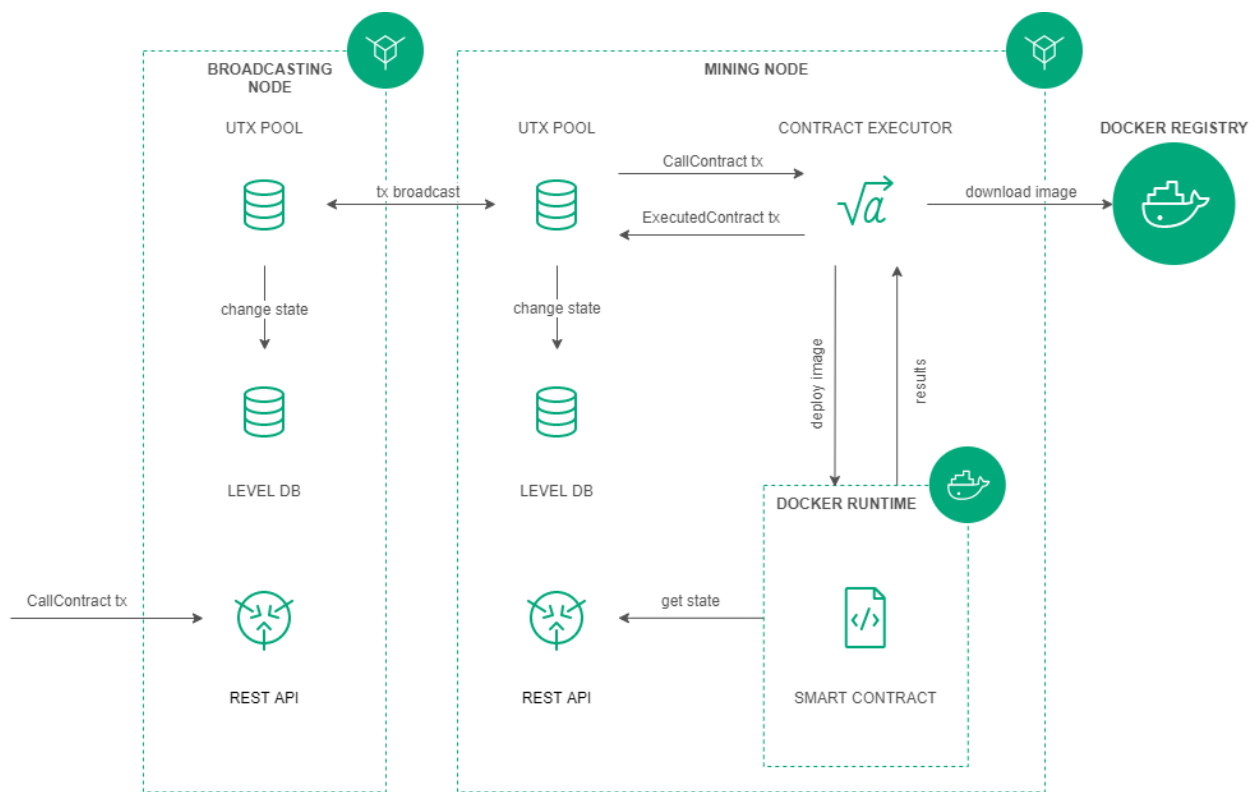
Data can be stored in the key-value format on addresses in the Waves Enterprise blockchain. The data stored on the address is available for viewing at the request [return data from address by key](#). The data is placed on the address when sending a data transaction. Since RIDE scripts are stateless, data transactions form an updated data storage which the script addresses. Configuring proof verification on a smart account allows multiple users to collaborate on data on a smart account. For example, with document flow statuses.

Important: Keys are unique for the address. Only one value corresponds to one address key. When publishing a new value for an existing key, it will be overwritten. The history and the author of changes can be tracked in the blockchain.

12.2 Docker Smart Contracts

In addition to contracts implemented on the basis of *RIDE* scripts for smart accounts and smart-assets, the Waves Enterprise platform provides the option to develop and use Turing-complete smart contracts. For implementation of Turing-complete contracts an approach is used where programs are launched in an isolated Docker container environment. Application development can be performed without restrictions in terms of the programming language to be used. Each application is launched in a Docker container to ensure isolation and manage resources available to a particular application. To store smart contracts, Docker Registry with read access only (Docker images) of contracts is used for machines with nodes. The node state can be accessed through a REST API node.

Important: It is a must to run the Docker-engine and the Docker-daemon simultaneously on the node which is processing the Docker smart-contracts.



12.2.1 Creating a contract

Creating a smart contract starts with preparation of a Docker image, which consists of the contract program code, the required environment and the special scenario Dockerfile. A prepared Docker image is assembled (built) and sent to Docker Registry.

Example of Dockerfile:

```
FROM python:alpine3.8
ADD contract.py /
ADD run.sh /
```

(continues on next page)

(continued from previous page)

```
RUN chmod +x run.sh
RUN apk add --no-cache --update iptables
CMD exec /bin/sh -c "trap : TERM INT; (while true; do sleep 1000; done) & wait"
```

The contract is created by publishing a special (CreateContractTransaction) transaction containing a link to the image in Docker Registry. After the transaction is received, the node downloads the image using the link specified in the “image” field, the image is checked and launched as a Docker container.

12.2.2 Executing a Contract

Smart contract execution is initiated by a special (CallContractTransaction) transaction containing the contract ID and call parameters. The transaction ID defines the Docker container. The container is executed unless it has been launched before. The contract launch parameters are transferred to the container. | Smart contracts change their state by updating the key-value pairs.

12.2.3 Updating Contract

Only the developer of the Docker smart contract can update this contract. The developer should keep the `contract_developer` role during the contract update and should be the *103* transaction creator. *107* transaction is using for the contract update. And it is necessary that the contract is active.

All the mining nodes download the contract image and run it for the checking after the *107* transaction includes into the block. Then the *105* transaction is issued within the *107* transaction inside it.

12.2.4 Contract Call Disabling

If necessary, the contract developer can disable calling the contract. To do this, a special (DisableContractTransaction) transaction is published specifying the Contract ID. The contract becomes unavailable after its disconnection, but you can get information about the contract from the the blockchain later.

12.2.5 Description of Transactions

The following transactions are implemented to ensure the interaction between the blockchain and the Docker Contract:

Code	Transaction type	Purpose
103	<i>CreateContractTransaction</i>	Initiates the Contract. Transaction is signed by a user with the role “ <i>contract_developer</i> ”
104	<i>CallContractTransaction</i>	Calls the Contract. Transaction is signed by the initiator of contract execution
105	<i>ExecutedContractTransaction</i>	Records the contract execution result in the contract state. br Transaction is signed by the block generating node
106	<i>DisableContractTransaction</i>	Disables calling a contract. Transaction is signed by a user with the role “ <i>contract_developer</i> ”
107	<i>UpdateContractTransaction</i>	Updating a contract. Transaction is signed by a user with the role “ <i>contract_developer</i> ” Only the contract developer and <i>103</i> transaction issuer can update the contract

12.2.6 Node configuration

Downloading and execution of Docker Contracts initiated by transactions with codes 103-107 are performed on nodes with enabled option `docker-engine.enable = yes` (for details see module “*Node configuration*” > “*Docker configuration*”).

12.2.7 REST API

The REST API methods description for the Docker contract usage is represented on the *API methods available to smart contract* page.

12.2.8 Implementation examples

- *Creating a simple contract*

ANCHORING

One of the main ideas behind private blockchain is that transactions are processed by a certain number of participants known in advance. In a private blockchain there is a threat of information spoofing, because the number of participants is quite small if we compare it to the public blockchain where there are no restrictions to join the network. When using PoS consensus algorithm, the threat of overwriting a blockchain becomes real.

To increase the confidence of the private blockchain participants in data placed there, the anchoring mechanism was developed. Anchoring allows checking the data for invariability. The guarantee of invariability is achieved by publishing data from a private blockchain to a larger network, where data spoofing is unlikely due to a larger number of participants and blocks. Published data represent a signature and a height of blocks in a private network. Mutual connectivity of two or more networks increases their resistance, because as a result of a **long-range attack** as forgery or alteration of data resulting from a long-range attack would require attacking all connected networks.

13.1 How does anchoring work in the Waves Enterprise blockchain

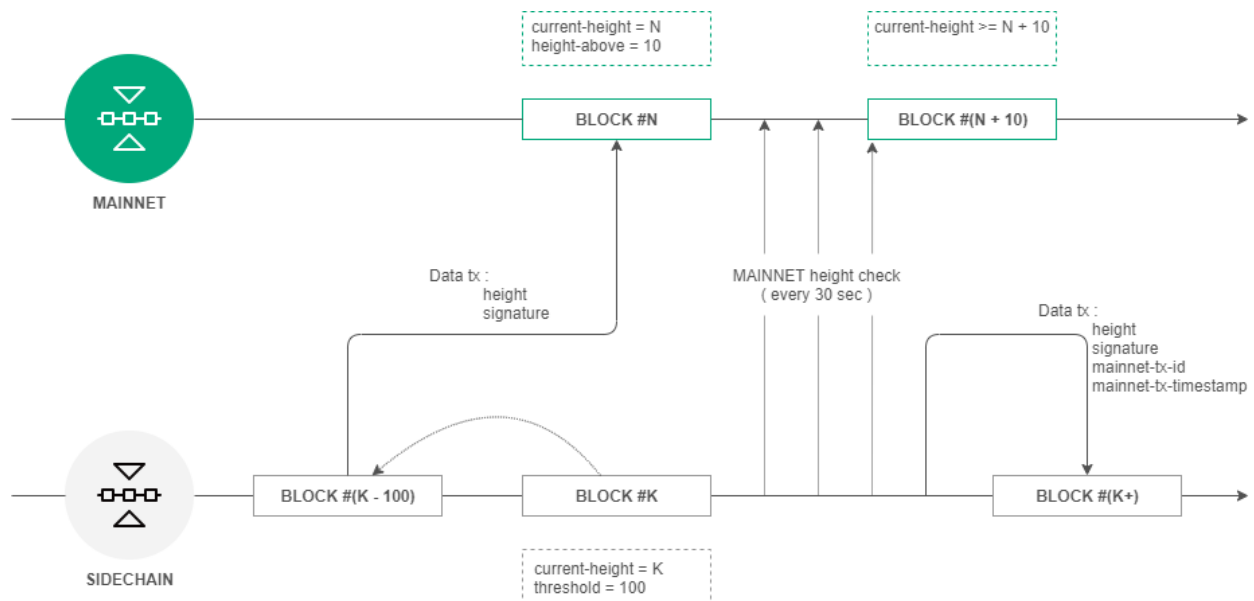


Fig. 1: Mainnet anchoring scheme

Anchoring process is shown below:

1. *Anchoring configurations* are set in the configuration file of the private blockchain node. You should use recommended values for the configurations to avoid anchoring malfunctioning.
2. Each **height-range** an anchoring transaction, that contains data of the block at **current-height - threshold**, is broadcasted to the Mainnet by the anchoring node. The *Data Transaction* with a **key-value** list is used as *an anchoring transaction*. The node then requests height of the broadcasted transaction.
3. The node then checks the Mainnet height each 30 seconds until its height reaches **the height of the created transaction + height-above**.
4. When required height is reached and the presence of previously created data transaction is confirmed, another anchoring data transaction is created in the private blockchain.

13.2 Transaction structure for anchoring

Mainnet transaction consists of the following fields:

- **height** - the height of the chosen block from the private blockchain.
- **signature** - the signature of the chosen block from the private blockchain.

The private blockchain transaction consists of the following fields:

- **height** - the height of the chosen block from the private blockchain.
- **signature** - the signature of the chosen block from the private blockchain.
- **mainnet-tx-id** - the Mainnet anchoring transaction ID.
- **mainnet-tx-timestamp** - the timestamp of the Mainnet anchoring transaction.

13.3 Errors during the anchoring

Errors during the anchoring can occur at any step. In case of any error in the private blockchain the *Data Transaction* containing the error code and the description is always published. The error transaction includes the following data:

- **height** - the height of the chosen block from the private blockchain.
- **signature** - the signature of the chosen block from the private blockchain.
- **error-code** - the error code.
- **error-message** - the error message.

Table 1: Error types

Code	Message	Possible cause
0	Unknown error	An unknown error occurred during the send of the transaction to the Mainnet
1	Fail to create data transaction for Mainnet	Creating of the transaction to be sent to the Mainnet failed
2	Fail send transaction to Mainnet	The transaction publication to the Mainnet failed (it could be a JSON request error)
3	Invalid http status of response from mainnet transaction broadcast	The Mainnet has returned an HTTP code other than 200 after the transaction publication
4	Fail to parse http body of response from mainnet transaction broadcast	The Mainnet has returned an unknown JSON after the transaction publication
5	Mainnet return transaction with id='\$mainnetTxId' but it differ from transaction that we sent id='\$sentTxId'	The Mainnet has returned mismatched ID after the transaction publication
6	Mainnet didn't respond on transaction info request	The Mainnet has not responded to the request about the transaction info
7	Fail to get current height in Mainnet	Failed to get current Mainnet height
8	Anchoring transaction in mainnet disappeared after height rise enough	The anchoring transaction has disappeared from the Mainnet after its height evened height-above value
9	Fail to create sidechain anchoring transaction	Fail to public the anchoring transaction in the private blockchain
10	Anchored transaction in sidechain was changed during mainnet height arise await, looks like a rollback has happened	Anchored transaction in sidechain was changed during mainnet height arise await, looks like a rollback has happened

AUTHORIZATION SERVICE

The authorization service is an external service and it provides authorization for all components of the blockchain network. Service is built on the basis of ‘OAuth 2.0 <<https://en.wikipedia.org/wiki/OAuth>> ‘_ authorization protocol. OAuth 2.0 is the open framework for realization of the authorization mechanism, allowing to give to the third part the limited access to the protected resources without credentials transfer to the third part. The data flow scheme between participants of information interaction on the OAuth 2.0 basis is presented below.

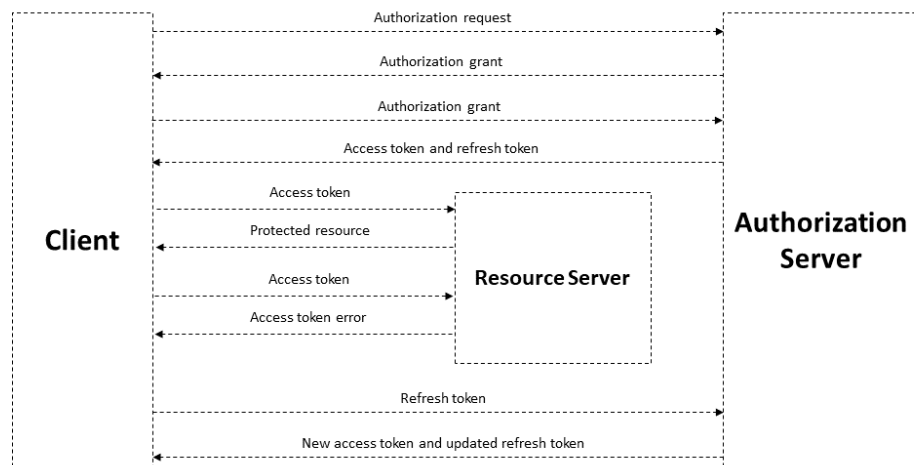


Fig. 1: Basic authorization scheme based on OAuth 2.0 protocol

JSON Web Token is the authorization unit. Tokens are used to authorize each request from the client to the server and have a limited lifetime. The client receives two types of token - access and refresh. Access token is used to authorize requests for access to protected resources and to store additional information about the user. The refresh token is used to get a new access token and to refresh the refresh token.

In general, the authorization scheme includes the following operations:

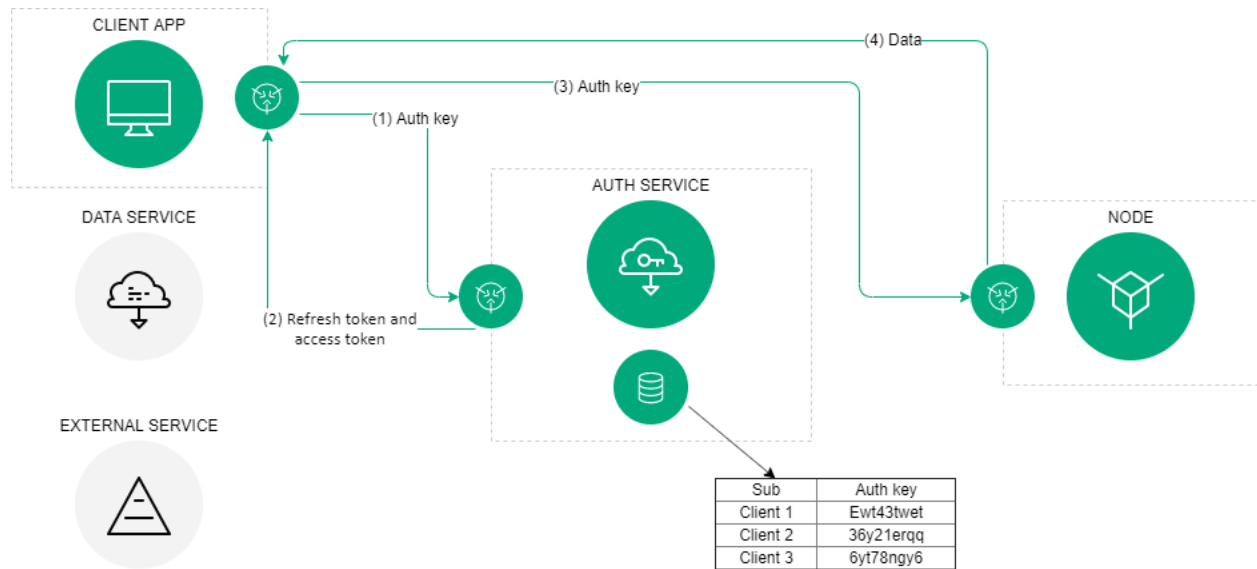


Fig. 2: The authorization scheme of the Waves Enterprise blockchain platform

1. The client (it could be any blockchain network component like the web client, data service or an external application) once provides its authentication data to the authorization service.
2. If the initial authentication procedure is successful, the authorization service stores the client's authentication data in the database, generates and sends signed access and refresh tokens to the client. Tokens include the lifetime info and basic customer data, such as an ID and a role. Client authentication data are stored in the authorization service configuration file. The client checks the lifetime of the access token each time before sending a request to a third-party service. In case of expiration of the token, the client refers to the authorization service to obtain a new access token. The refresh token is used for requests to the authorization service.
3. The client sends a request of receiving data from a third-party service using the current access token.
4. The external application checks the lifetime of the access token, its integrity, and compares the previously obtained public key of the authorization service with the key contained in the signature of the access token. In case of successful verification, this service provides the requested data to the client.

MAINNET AND PARTNERNET CONNECTION

15.1 Working inside the “Waves Enterprise Mainnet”

15.1.1 Connection of the node to the “Waves Enterprise Mainnet”

Warning: The account balance must be at least **10 000 WEST** if you want to connect your node to the network “Waves Enterprise Mainnet” and do mining!

Follow these steps for the node connection to the “Waves Enterprise Mainnet”:

1. Go to the website <https://client.wavesenterprise.com/> and create an account following the web-interface hints.
2. Transfer tokens to the “Waves Enterprise Mainnet” network.
3. Transfer for leasing any number of tokens to the `3NrKDuHjUG7vSCiMMD259msBKcPRm4MvaJu` address and keep the transaction ID. Further you can withdraw tokens from the lease, because this operation is necessary to verify your ownership of this address and the balance.
4. *Install* the node software.
5. Perform the node *configuration*. An example of a node configuration file can be found on the project page on [GitHub](#). To add a node to the Mainnet network, the name of the configuration file is `mainnet-example.conf`.
6. Go to the website <https://support.wavesenterprise.com/servicedesk/customer/portal/3> and perform the registration.
7. Select the type of request “Participant connection” for legal or natural person.
8. Register on the resource by filling in all the required fields of the form. If you want to mine, check the box **Please grant mining rights**.
9. Enter the transaction ID of the token lease transfer in the **Proof of WEST token ownership** field.
10. Please, wait for the connection application consideration. You can start working in the “Waves Enterprise Mainnet” after successful registration.
11. *Run* the node after obtaining permission to connect to the network “Waves Enterprise Mainnet”, public key of which you specified in the application.
12. Transfer or lease tokens to the address of the connected node for the mining and work in the network.

15.1.2 Fees in the “Waves Enterprise Mainnet”

#	Transaction type	Fee	Description
1	<i>Genesis transaction</i>	no fee	Initial binding of the balance to the addresses of nodes created at the start of the blockchain
2	Payment Transaction (not used)		
3	<i>Issue Transaction</i>	1WEST	Tokens issue. The fee is charged only in WEST
4	<i>Transfer Transaction</i>	0.01WEST	Tokens transfer
5	<i>Reissue Transaction</i>	1WEST	Tokens reissue
6	<i>Burn Transaction</i>	0.05WEST	Tokens burn
8	<i>Lease Transaction</i>	0.01WEST	Tokens lease
9	<i>Lease Cancel Transaction</i>	0.01WEST	Cancel of the tokens lease
10	<i>Create Alias Transaction</i>	1WEST	Alias creation
11	<i>Mass Transfer Transaction</i>	0.05WEST	Mass tokens transfer. Minimum commission is specified, the fee depends on the number of addresses in the transaction
12	<i>Data Transaction</i>	0.05WEST	Transaction with the data in the key-value pairs format. The fee is always charged to the transaction author. Minimum commission is specified, the fee depends on data volume
13	<i>SetScript Transaction</i>	0.5WEST	Transaction which is binding a script with a RIDE contract to an account
14	SponsorFee Transaction (not used)		
15	<i>SetAssetScript</i>	1WEST	Transaction which is binding a script with a RIDE contract to an asset
101	<i>Genesis Permission Transaction</i>	no fee	Assignment of the first network administrator for further distribution of rights
102	<i>Permission Transaction</i>	0.01WEST	Grantance/withdrawal of rights from the account
103	<i>CreateContract Transaction</i>	1WEST	Docker-contract creation
104	<i>CallContract Transaction</i>	0.1WEST	Docker-contract call
105	<i>Executed-Contract Transaction</i>	no fee	Docker-contract execution
106	<i>Disable-Contract Transaction</i>	0.01WEST	Docker-contract disable
107	<i>Update-Contract Transaction</i>	1WEST	Docker-contract update
110	<i>Genesis RegisterNode Transaction</i>	no fee	Node registration in the genesis block with the blockchain start
111	<i>RegisterNode Transaction</i>	0.01WEST	New node registration

15.1.3 Examples of the “Waves Enterprise Mainnet” configuration files

You can read *here* about the node configuration.

The `accounts.conf` file example

```
// accounts.conf listing

accounts-generator {
  waves-crypto = yes
  chain-id = V
  amount = 1
  wallet = ${user.home}"/node/keystore.dat"
  wallet-password = "some string as password"
  reload-node-wallet {
    enabled = false
    url = "http://localhost:6869/utills/reload-wallet"
  }
}
```

The `chain-id` parameter contains the identification network byte, for the “Waves Enterprise Mainnet” in is V.

The `api-key-hash` file example

```
// api-key-hash.conf listing

apikeyhash-generator {
  waves-crypto = no
  api-key = "some string"
}
```

The node configuration file example

```
node {
  # Type of cryptography
  waves-crypto = yes

  # Node owner address
  owner-address = ""

  ntp {
    fatal-timeout = "1 minute"
    server = "pool.ntp.org"
  }

  # Node "home" and data directories to store the state
  # directory = ${user.home}"/node"
  # data-directory = ${node.directory}"/data"

  # Settings for Privacy Data Exchange
  # Uncomment and fill to enable
  # privacy {
  #   storage {
  #     url = "jdbc:postgresql://"${POSTGRES_ADDRESS}":"${POSTGRES_PORT}"/"${POSTGRES_DB}"
```

(continues on next page)

(continued from previous page)

```
# driver = "org.postgresql.Driver"
# profile = "slick.jdbc.PostgresProfile$"
#
# user = ${POSTGRES_USER}
# password = ${POSTGRES_PASSWORD}
# connectionPool = HikariCP
# connectionTimeout = 5000
# connectionTestQuery = "SELECT 1"
# queueSize = 10000
# numThreads = 20
# schema = "public"
# migration-dir = "db/migration"
# }
# }

# Blockchain settings
# Mainnet blockchain settings (should match on all nodes for consistency)
blockchain {
  type = CUSTOM
  consensus.type = pos

  custom {
    address-scheme-character = "V"
    functionality {
      feature-check-blocks-period = 15000
      blocks-for-feature-activation = 10000
      pre-activated-features = {
        2 = 0
        3 = 0
        4 = 0
        5 = 0
        6 = 0
        7 = 0
        9 = 0
        10 = 0
      }
    }
  }
}

# Mainnet genesis settings
genesis {
  average-block-delay: 40s
  initial-base-target: 10000000000
  block-timestamp: 1559320391040
  initial-balance: 10000000000000000
  genesis-public-key-base-58: "D7tDsKd7DQ7H9m6fPRyk1GsNQxjAQXsETtuVgqSaaXDs"
  signature:
↪ "P7kwe3dWSWgUYL8FZu5kccPfPzoxGgLuKjTCkeapTxoDbdp06EtcqndXoSjqUUVS67xXfogGmaNroLgNocWcBg
↪ "
  transactions = [
    {recipient: "3Nnq14SGqeYETSd1SJ6z8LsgBRYB2ya1yRC", amount: 9999000000000000},
    {recipient: "3Nrystx7J1TN6vB1eYdHgug2nfxA7um918zy", amount: 1000000000000},
    {recipient: "3NuiCzDhmeSKL5QFa5sqZzzm9zTL4max4fZ", amount: 1500000000000},
    {recipient: "3NqaDwdEgGsqqJ1HjzndQMtk6v5KVxmRceg", amount: 2000000000000},
    {recipient: "3Nckru7f8Y8vS3PXGyy5iwoheRrKvqW5u8x", amount: 2500000000000},
    {recipient: "3NmHrYoC8S2SUosy6UJp47bBwq2Cr2X6Yq1", amount: 3000000000000}
  ]
}
```

(continues on next page)

(continued from previous page)

```

]
network-participants = [
  {public-key: "GasRtAUXMhifrUUmgU66rRZPii68tE4QxdQmtCcrV3xL", roles: [permissioner,
↪ connection_manager]},
  {public-key: "Er29kgV3yeumEAtPxBAk5fXPERYYa1wmAcPgzWw4mxHi", roles: [miner]},
  {public-key: "9eoVBycnr2m8bgu1WvYySoFJ1QqFLPAMzhnmErp291f6", roles: [miner]},
  {public-key: "9ngXJ3d1XSQgXcYbgZm2wH4QHS8Tc5mtf9M4XDoz5db", roles: [miner]},
  {public-key: "2cvrBT6jePt6mjinE1EdLLymoqRHFhWwepM3E5gRuSeL", roles: [miner]},
  {public-key: "87ZVwBTeBiKYdF2Q5hxGazwhR1pKy9VYgun8rLFMEmoW", roles: [miner]}
]
}

fees {
  genesis = 0
  genesis-permit = 0
  issue = 100000000
  transfer = 1000000
  reissue = 100000000
  burn = 5000000
  exchange = 500000
  lease = 1000000
  lease-cancel = 1000000
  create-alias = 100000000
  mass-transfer = 5000000
  data = 5000000
  set-script = 50000000
  sponsor-fee = 100000000
  set-asset-script = 100000000
  permit = 1000000
  create-contract = 100000000
  call-contract = 10000000
  executed-contract = 0
  disable-contract = 1000000
  update-contract = 100000000
  register-node = 1000000
  create-policy = 100000000
  update-policy = 50000000
  policy-data-hash = 5000000
  additional {
    mass-transfer = 1000000
    data = 1000000
  }
}
}
}

# Application logging level. Could be DEBUG | INFO | WARN | ERROR. Default value is 
↪ INFO.
logging-level = DEBUG

features {
  supported = [] # NG
}

# P2P Network settings
network {

```

(continues on next page)

(continued from previous page)

```

# Network address
bind-address = "0.0.0.0"
# Port number
port = 6864

# Peers network addresses and ports
# Example: known-peers = ["node-0.wavesenterprise.com:6864", "node-1.wavesenterprise.
↪com:6864"]
known-peers = [ ]

# Node name to send during handshake. Comment this string out to set random node name.
# node-name = "node"

# String with IP address and port to send as external address during handshake. Could ↪
↪be set automatically if uPnP is enabled.
declared-address = "0.0.0.0:6864"
}

wallet {
# Path to keystore. In case of GOST cryptography keys stored in a './keystore/' folder. ↪
↪In case of Waves-cryptography keys stored in a 'keystore.dat' file.
file = ${user.home}"/node/keystore.dat"
# Access password
password = ""
}

# Node's REST API settings
rest-api {
enable = yes
bind-address = "0.0.0.0"
port = 6862

# Hashed secret Api-Key to access node's REST API
api-key-hash = ""

# Api-key hash for Privacy Data Exchange REST API methods
privacy-api-key-hash = ""
}

# New blocks generator settings
miner {
enable = no
quorum = 2
interval-after-last-block-then-generation-is-allowed = 35d
micro-block-interval = 5s
min-micro-block-age = 3s
max-transactions-in-micro-block = 500
minimal-block-generation-offset = 200ms
}

# Anchoring settings
scheduler-service.enable = no

# Docker smart-contracts engine config
docker-engine {
enable = no
    
```

(continues on next page)

(continued from previous page)

```

execution-limits {
  timeout = 10s
  memory = 512
  memory-swap = 512
}
}
}

```

15.2 Working inside the “Waves Enterprise Partnetnet”

15.2.1 Connection of the node to the “ Waves Enterprise Partnetnet”

Follow these steps for the node connection to the “Waves Enterprise Partnetnet”:

1. *Create* the `accounts.conf` configuration file before the generator start.
2. Download the `current` release of the node and generator in the jar format.
3. *Generate* a key pair for the connected node using the generator. For your convenience it is recommended to create one key pair for one node, please, specify the number of nodes 1 in the `amount` field of the `accounts.conf` configuration file. Enter the node address password during the key pair creation and keep it for the following steps. Press `enter` key if you do not want to use this password.
4. Create the node configuration file using the template from the project [GitHub](#). Please, fill all the fields marked with `#FILL` string. If you want the node to be a miner specify the value `yes` of the `enable` parameter of the `miner` block and request the miner rights inside the connection application. Otherwise specify the `no` value. Also specify the PostgreSQL DB address as a value of the `url` parameter of the `privacy {storage {}}` block.
5. If you do not want to enter the password each time when node is starting, create the `WE_NODE_OWNER_PASSWORD` and `WE_NODE_OWNER_PASSWORD_EMPTY` global variables in your OS.
6. Go to the website ‘<<https://support.wavesenterprise.com/servicedesk>>_ and perform the registration.
7. Select the type of request “Participant connection” for legal or natural person.
8. Register on the resource by filling in all the required fields of the form. If you want to mine, check the box **Please grant mining rights**.
9. Please, wait for the connection application consideration. You can start working in the “Waves Enterprise Partnetnet” after successful registration.
10. *Run* the node after getting the application approve.

15.2.2 Examples of the “Waves Enterprise Partnetnet” configuration files

You can read *here* about the node configuration.

The accounts.conf file example

```
// accounts.conf listing

accounts-generator {
  waves-crypto = yes
  chain-id = P
  amount = 1
  wallet = "${user.home}"/node/keystore.dat"
  wallet-password = "some string as password"
  reload-node-wallet {
    enabled = false
    url = "http://localhost:6869/utils/reload-wallet"
  }
}
```

The `chain-id` parameter contains the identification network byte, for the “Waves Enterprise Partnernet” in is P. If you want to use the GOST cryptography specify the no value of the `waves-crypto` parameter inside all the configuration files. Also install the [CryptoPro JCP 2.0.40035](#) software before the node configuration. You can find full info about installation [here](#).

The api-key-hash file example

```
// api-key-hash.conf listing

apikeyhash-generator {
  waves-crypto = yes
  api-key = "some string"
}
```

The node configuration file example

```
node {
  waves-crypto = yes
  # Blockchain settings
  blockchain {
    type: CUSTOM
    consensus.type = PoS
    custom {
      address-scheme-character: "P"
      functionality {
        feature-check-blocks-period = 1
        blocks-for-feature-activation = 1
        pre-activated-features { 1 = 0, 2 = 0, 3 = 0, 4 = 0, 5 = 0, 6 = 0, 7 = 0, 8 = 0, 9 = 0, 10 = 0 }
        double-features-periods-after-height = 100000000
      }
    }
    genesis {
      average-block-delay: 60s
      initial-base-target: 153722867
      block-timestamp: 1559260800000
      initial-balance: 1625000000000000
      genesis-public-key-base-58: "8RbU8qKWWxLuVk49LgeE39y83LUTVp1zHEJwMM7zKaMC"
      signature:
        "2dKzdL9bdWz1B9wBPnGALfowrPDSidEoGAQEoRogGuBB4sQanCr4JySXvWoAmpu1EgcH8MsC0TL3TzSMoEyc2U"
    }
  }
}
```

(continues on next page)

(continued from previous page)

```

transactions = [
  { recipient: "3LWg4n6VmN6DKBSwGF1hwNaCzXdjMkQCFrn", amount: 1250000000000000 },
  { recipient: "3LPPZNhakdm9ZPiGShNvWGCshFqsQXFjUQ1", amount: 3000000000000000 },
  { recipient: "3LEpXfh7XmCRias92swo6LUJqyo9MA7SaFc", amount: 7500000000000000 }
]
network-participants = [
  {public-key: "CaFrRzAv7B3DrECR4i2Los1DwxHj4yKAEKCT3zEke9U4", roles: [permissioner,
↪miner, connection_manager]},
  {public-key: "Vxb6LQ8Qt9Afs6VJuyiMbMN5qM2pm1EEcWdoZo3WmkN", roles: [miner,
↪permissioner]},
  {public-key: "FmzyByBePwbKDjSdnYjwF9G12zGrQc7Gcr8WvQ5ybejC", roles: [miner]}
]
}
}
}
# Application logging level. Could be DEBUG | INFO | WARN | ERROR. Default value is
↪INFO.
logging-level = DEBUG
# P2P Network settings
network {
# Network address
bind-address = "0.0.0.0"
# Port number
port = 6864
known-peers = [
"node0-partnernet.wavesenterprise.com:6864",
"node1-partnernet.wavesenterprise.com:6864",
"node2-partnernet.wavesenterprise.com:6864"
]
# Node name to send during handshake. Comment this string out to set random node name.
# String with IP address and port to send as external address during handshake. Could
↪be set automatically if uPnP is enabled.
declared-address = "0.0.0.0:6864"
}
wallet {
file = "" #FILL
password = "" #FILL
}
# Privacy network settings: node owner address is used to sign handshakes
owner-address = "" #FILL

ntp {
fatal-timeout = "1 minute"
server = "pool.ntp.org"
}

# Matcher settings
matcher.enable = no
# Node's REST API settings
rest-api {
enable = yes
bind-address = "0.0.0.0"
port = 6862
api-key-hash = "" #api-key for all api #FILL
privacy-api-key-hash = "" #api-key for SendData api #FILL
}

```

(continues on next page)

(continued from previous page)

```

# New blocks generator settings
miner {
  enable = yes
  interval-after-last-block-then-generation-is-allowed = 15d
  quorum = 1
  minimal-block-generation-offset = 200ms
}
# Anchoring
scheduler-service.enable = no

# For docker smart-contracts
docker-engine {
  enable = yes
  # Optional connection string to docker host
  # docker-host = "unix:///var/run/docker.sock"
  # Optional string to node REST API if we use remote docker host
  # node-rest-api = "https://clinton.weservices.com/node-0"
  execution-limits {
    timeout = 10s
    memory = 512
    memory-swap = 512
  }
  allow-net-access = yes
}

privacy {
  # DB connection config
  storage {
    url = "" #FILL insert DB connection string here, example "jdbc:postgresql://db_
    ↪hostname:5432/____?user=_____&password=____"
    driver = "org.postgresql.Driver"
    profile = "slick.jdbc.PostgresProfile$"
    connectionPool = HikariCP
    connectionTimeout = 5000
    connectionTestQuery = "SELECT 1"
    queueSize = 10000
    numThreads = 10
    schema = "public"
    migration-dir = "db/migration"
  }
}
}
}

```


REST API

The Waves Enterprise blockchain platform provides an opportunity to interact with blockchain both in terms of receiving data (transactions, blocks, balances, etc.) and in terms of writing information to blockchain (signing and sending transactions) via RESTful API of the node. REST API allows users to interact remotely with the node using requests and responses in JSON format. HTTPS protocol is used to work with API and as an interface it is utilized the Swagger framework.

16.1 Node REST API methods

16.1.1 Activation

Hint: The rules for generating requests to the node are given in module *How to use REST API*.

GET /activation/status

Returns the activation status of the new functionality in the node(s).

Method Response:

```
{ "height": 47041,
  "votingInterval": 1,
  "votingThreshold": 1,
  "nextCheck": 47041,
  "features": [
    { "id": 1,
      "description": "Minimum Generating Balance of 1000 WEST",
      "blockchainStatus": "ACTIVATED",
      "nodeStatus": "IMPLEMENTED",
      "activationHeight": 0 },
    { "id": 2,
      "description": "NG Protocol",
      "blockchainStatus": "ACTIVATED",
      "nodeStatus": "IMPLEMENTED",
      "activationHeight": 0 },
    { "id": 3,
      "description": "Mass Transfer Transaction",
      "blockchainStatus": "ACTIVATED",
      "nodeStatus": "IMPLEMENTED",
      "activationHeight": 0 },
```

(continues on next page)

(continued from previous page)

```
{
  "id": 4,
  "description": "Smart Accounts",
  "blockchainStatus": "ACTIVATED",
  "nodeStatus": "IMPLEMENTED",
  "activationHeight": 0 },
  {"id": 5,
  "description": "Data Transaction",
  "blockchainStatus": "ACTIVATED",
  "nodeStatus": "IMPLEMENTED",
  "activationHeight": 0 },
  {"id": 6,
  "description": "Burn Any Tokens",
  "blockchainStatus": "ACTIVATED",
  "nodeStatus": "IMPLEMENTED",
  "activationHeight": 0 },
  {"id": 7,
  "description": "Fee Sponsorship",
  "blockchainStatus": "ACTIVATED",
  "nodeStatus": "IMPLEMENTED",
  "activationHeight": 0 },
  {"id": 8,
  "description": "Fair PoS",
  "blockchainStatus": "ACTIVATED",
  "nodeStatus": "IMPLEMENTED",
  "activationHeight": 0 },
  {"id": 9,
  "description": "Smart Assets",
  "blockchainStatus": "VOTING",
  "nodeStatus": "IMPLEMENTED",
  "supportingBlocks": 0 },
  {"id": 10,
  "description": "Smart Account Trading",
  "blockchainStatus": "ACTIVATED",
  "nodeStatus": "IMPLEMENTED",
  "activationHeight": 0 } ]
}
```

16.1.2 Addresses

Hint: The rules for generating queries to the node are given in module *How to use REST API*.

GET/addresses

Get all addresses of participants whose key pairs are stored in the node keystore.

Method Response:

```
[
  "3NBVqYXrapgJP9atQccdBP AgJPwHDKkh6A8",
  "3Mx2afTZ2KbRrLNbytyzTtXukZvqEB8SkW7"
]
```


GET/addresses/seq/{from}/{to}

Gets all addresses of participants whose key pairs are stored in node keystore in the specified range.

Method Response:

```
[
  "3NBVqYXrapgJP9atQccdBP AgJPwHDKkh6A8",
  "3Mx2afTZ2KbRrLNbytyzTtXukZvqEB8SkW7"
]
```

GET/addresses/balance/{address}

Get the balance for the address {address}.

Method Response:

```
{
  "address": "3N3keodUiS8WLEw9W4BKDNxgNdUpwSnpb3K",
  "confirmations": 0,
  "balance": 100945889661986
}
```

POST/addresses/balance/details

Get balances for the address list.

Method Query:

```
{
  "addresses": [
    "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ", "3N11u447zghwj9MemYkrkt9v9xDaMwTY9nG"
  ]
}
```

GET/addresses/effectivebalance/{address}/{confirmations}

Get the balance for the address {address} after a number of confirmations \geq value {confirmations}. Returns the total balance of the participant, including assets transferred to the participant for the leasing.

Method Response:

```
{
  "address": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",
  "confirmations": 1,
  "balance": 0
}
```

GET /addresses/effectiveBalance/{address}

Get the effective balance of the specified address.

Method Response

```
{
  "address": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
  "confirmations": 0,
  "balance": 1240001592820000
}
```

GET /addresses/balance/details/{address}

Returns detailed information about balance of address {address}.

Method Query:

```
{
  "addresses": [
    "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ"
  ]
}
```

Method Response:

```
[
  {
    "address": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",
    "regular": 0,
    "generating": 0,
    "available": 0,
    "effective": 0
  }
]
```

Response Options

- Regular - total balance of participant, including assets transferred for leasing
- Available - total balance of participant, except for assets transferred for leasing
- Effective — total balance of participant, including assets transferred to participant for leasing (Available + assets transferred to you for leasing)
- Generating - minimum balance of participant, including assets transferred to participant for leasing, for the last 1000 blocks (used for mining)

GET/addresses/scriptInfo/{address}

Get information about the script installed on the address {address}.

Method Response:

```
{
  "address": "3N3keodUiS8WLEw9W4BKDNxgNdUpwSnpb3K",
  "script":
  ↪ "3rbFDtbPwAvSp2vBvqGfGR9nRS1nBVnfuSCN3HxSZ7fVRpt3tuFG5JSmyTmvHPxyf34So cMRkRKFgzTtXXnnv7upRHXJzZrLSQo8tUW6yMtEiZ
  ↪",
  "scriptText": "ScriptV1(BLOCK(LET(x,CONST_LONG(1)),FUNCTION_CALL(FunctionHeader(=,List(LONG,
  ↪LONG)),List(FUNCTION_CALL(FunctionHeader(+,List(LONG, LONG)),List(REF(x, LONG), CONST_LONG(1)),
  ↪LONG), CONST_LONG(2)),BOOLEAN),BOOLEAN))",
  "complexity": 11,
  "extraFee": 10001
}
```

Response Options

- “address” - address in Base58 format
- “script” - Base64 representation of the script
- “scriptText” - source code of the script
- “complexity” - complexity of the script
- “extraFee” - fee for outgoing transactions set by the script

POST/addresses/sign/{address}

Returns the message encoded in BASE58 format signed by address private key {address}, stored in node keystore. The message is first signed and then converted.

Method Query:

```
{
  "message": "mytext"
}
```

Method Response:

```
{
  "message": "wWshKhJj",
  "publicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
  ↪ "62PFG855ThsEHUZ4N8VE8kMyHCK9GWnvtTZ3hq6JHYv12BhP1eRjegA6nSa3DAoTTMammhamadvizDUYZAZtKY9S"
}
```

POST /addresses/verify/{address}

Validates signature of a message executed by address {address}, including the one created through POST method /addresses/sign/{address}.

Method Query:

```
{
  "message": "wWshKhJj",
  "publicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
  ↪ "5kwwE9sDZzss0NaoBSJnb8RLqfYGt1NDGbTWWXUeX8b9amRRJN3hr5fhs9vHBq6VES5ng4hqbCUoDEsoQNauRRts"
}
```

Method Response:

```
{
  "valid": true
}
```

POST /addresses/signtext/{address}

Returns a message signed by address private key {address} stored in the node keystore.

Method Query:

```
{
  "message": "mytext"
}
```

Method Response:

```
{
  "message": "message",
  "publicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
  ↪ "5kVZfWfFmoYn38cJfNhkdct5WCyksMgQ7kjjwHK7Zjnrzs9QYRwo6HuJoGc8WRMozdYcAVJvojJnPPArqPvu2uc3u"
}
```

POST /addresses/verifytext/{address}

Validates signature of a message executed by address {address}, including the one created through the POST method /addresses/signtext/{address}.

Method Query:

```
{
  "message": "message",
  "publicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
  ↪ "5kVZfWfFmoYn38cJfNhkdct5WCyksMgQ7kjjwHK7Zjnrzs9QYRwo6HuJoGc8WRMozdYcAVJvojJnPPArqPvu2uc3u"
}
```

Method Response:

```
{
  "valid": true
}
```

GET /addresses/validate/{addressOrAlias}

Validates correctness of specified address or its alias {addressOrAlias} in a network blockchain of operating node.

Method Response:

```
{
  addressOrAlias: "3HSVTtjim3FmV21HWQ1LurMhFzjut7Aa1Ac",
  valid: true
}
```

POST /addresses/validateMany

Checks the validity of addresses or aliases.

Method Query:

```
{
  addressesOrAliases: [
    "3HSVTtjim3FmV21HWQ1LurMhFzjut7Aa1Ac",
    "alias:T:asdfghjk",
    "alias:T:1nvAlidAl1ass99911%~&$$$ "
  ]
}
```

Method Response:

```
{
  validations: [
    {
      addressOrAlias: "3HSVTtjim3FmV21HWQ1LurMhFzjut7Aa1Ac",
      valid: true
    },
    {
      addressOrAlias: "alias:T:asdfghjk",
      valid: true
    },
    {
      addressOrAlias: "alias:T:1nvAlidAl1ass99911%~&$$$ ",
      valid: false,
      reason: "GenericError(Alias should contain only following characters: -.0123456789@_
↵abcdefghijklmnopqrstuvwxyz)"
    }
  ]
}
```

GET /addresses/publicKey/{publicKey}

Returns participant address based on its public key.

Method Response:

```
{
  "address": "3N4WaaaNAVLMQgVKTRSePgWbuAKvZTjAQbq"
}
```

GET /addresses/data/{address}

Returns all data recorded to address account {address}.

Method Response:

```
[
  {
    "key": "4yR7b6Gv2rzLrhYBHpgVCmLH42raPGTF4Ggi1N36aWnY",
    "type": "integer",
    "value": 1500000
  }
]
```

GET /addresses/data/{address}/{key}

Returns data recorded to address account {address} by key {key}.

Method Response:

```
{
  "key": "4yR7b6Gv2rzLrhYBHpgVCmLH42raPGTF4Ggi1N36aWnY",
  "type": "integer",
  "value": 1500000
}
```

16.1.3 Alias

Hint: The rules for generating queries to the node are given in module *How to use REST API*.

GET /alias/by-alias/{alias}

Gets participant address by its alias {alias}.

Method Response:

```
{
  "address": "address:3Mx2afTZ2KbRrLNbytyzTtXukZvqEB8SkW7"
}
```

GET /alias/by-address/{address}

Gets alias {alias} of participant by its address {address}.

Method Response:

```
[
  "alias:HUMANREADABLE1",
  "alias:HUMANREADABLE2",
  "alias:HUMANREADABLE3",
]
```

16.1.4 Anchoring

GET /anchoring/config

Hint: Rules of the creating requests to a node, see *How to use REST API* section.

Get the *anchoring* section of the node configuration file.

Method answer

```
{
  "enabled": true,
  "currentChainOwnerAddress": "3FWwx4o1177A4oeHAEW5EQ6Bkn4Lv48quYz",
  "mainnetNodeAddress": "https://clinton-pool.vostokservices.com:443",
  "mainnetSchemeByte": "L",
  "mainnetRecipientAddress": "3JzVWCSV6v4ucSxtGSjZsvdiCT1FAzwpqrP",
  "mainnetFee": 8000000,
  "currentChainFee": 666666,
  "heightRange": 5,
  "heightAbove": 3,
  "threshold": 10
}
```

16.1.5 Assets

Hint: The rules for generating queries to the node are given in module *How to use REST API*.

GET/assets/balance/{address}

Returns balance of all address {address} assets.

Method Response:

```
{
  "address": "3Mv61qe6egMSjRDZiiuvJDnf3Q1qW9tTZDB",
  "balances": [
    {
      "assetId": "Ax9T4grFxx5m3KPUEKjMdnQkCKtBktf694wU2wJYvQUD",
```

(continues on next page)

(continued from previous page)

```

    "balance": 4879179221,
    "quantity": 48791792210,
    "reissuable": true,
    "minSponsoredAssetFee" : 100,
    "sponsorBalance" : 1233221,
    "issueTransaction" : {
      "type" : 3,
      ...
    }
  },
  {
    "assetId": "49KfHPJcKvSAvNKwM7CTofjKHZL87SaSx8eyADBjv5Wi",
    "balance": 10,
    "quantity": 10000000000,
    "reissuable": false,
    "issueTransaction" : {
      "type" : 3,
      ...
    }
  }
]
}

```

Method Parameters:

- “Address” - participant address
- “balances” - object with participant balance
- “assetId” - asset ID
- “balance” - asset balance
- “quantity” - number of issued assets
- “reissuable” - indicator whether asset can be reissued or not
- “issueTransaction” - asset creation transaction
- “minSponsoredAssetFee” - minimum value of fee for sponsorship transactions
- “sponsorBalance” - assets allocated for payment of sponsored asset transactions

GET /assets/balance/{address}/{assetId}

Returns address {address} balance by asset {assetId}.

Method Response:

```

{
  "address": "3Mv61qe6egMSjRDZiiuvJDnf3Q1qW9tTZDB",
  "assetId": "Ax9T4grFxx5m3KPUEKjMdnQkCKtBktf694wU2wJYvQUD",
  "balance": 4879179221
}

```


GET /assets/details/{assetId}

Returns description of asset {assetId}.

Method Response:

```
{
  "assetId" : "8tdULCMr598Kn2dUaKwHkvsNyFbDB1Uj5NxvVRTQRnMQ",
  "issueHeight" : 140194,
  "issueTimestamp" : 1504015013373,
  "issuer" : "3NCBMxgdghg4tUhEEffSYy11L6hUi6fcBpd",
  "name" : "name",
  "description" : "Sponsored asset",
  "decimals" : 1,
  "reissuable" : true,
  "quantity" : 1221905614,
  "script" : null,
  "scriptText" : null,
  "complexity" : 0,
  "extraFee": 0,
  "minSponsoredAssetFee" : 100000 // null assume no sponsorship, number - amount of assets for
  ↪ minimal fee
}
```

GET /assets/{assetId}/distribution

Returns distribution of asset {assetId}.

Method Response:

```
{
  "3P8GxcTEyZtG6LEfnn9knp9wu8uLKrAFHCb": 1,
  "3P2voHxcJg79csj4YspNq1akepX8TSmGhTE": 1200
}
```

POST /assets/balance

Returns the assets balance for one or few addresses.

Method Response

```
{
  "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG": [],
  "3GRLFi4rz3SniCuC7rbd9UuD2KUZyNh84pn": []
}
```

16.1.6 Blocks

Hint: The rules for generating queries to the node are given in module *How to use REST API*.

The last block may contain a different number of transactions during the period of its creation. It depends on the fact that while the block is not accepted by the nodes-miners, the number of transactions in it can constantly change. Therefore, when using methods that provide information about the last block, it should be kept in mind that the number of transactions in the last block may change.

GET /blocks/height

Returns block number of current blockchain state.

Method Response:

```
{
  "height": 7788
}
```

GET /blocks/height/{signature}

Returns height (number) of block by its signature.

GET /blocks/first

Returns contents of first block (genesis block).

GET /blocks/last

Returns contents of last block.

Method Response:

```
{
  "version": 2,
  "timestamp": 1479313809528,
  "reference":
  ↪ "4MLXQDbARiJDEAoy5vZ8QYh1yNnDhdGhGwkDKna8J6QXb7agVpFEi16hHBGUxxnq8x4myG4w66DR4Ze8FM5dh8Gi",
  "nxtconsensus": {
    "basetarget": 464,
    "generationsignature": "7WUV2TufarAyjicPFdnAWbn2Q7Jk7nBmWbnnDXKDEeJv"
  },
  "transactions": [
    {
      "type": 2,
      "id":
      ↪ "64hxaxZvB9iD1cfRf1j8KPTXs4qE7SHaDWTZKoUvgfVZotaJUtSGa5Bxi86ufAfp5ifoNAGknBqS9CpxBKG9RNVR",
      "fee": 100000,
      "timestamp": 1479313757194,
      "signature":
      ↪ "64hxaxZvB9iD1cfRf1j8KPTXs4qE7SHaDWTZKoUvgfVZotaJUtSGa5Bxi86ufAfp5ifoNAGknBqS9CpxBKG9RNVR",
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

    "sender": "3NBVqYXrapgJP9atQccdBP AgJPwHDKkh6A8",
    "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMhM3Uki7pLw",
    "recipient": "3N8UPtqiy322NVr1fLP7SaK1AaCU7oPaVuy",
    "amount": 100000000
  }
],
"generator": "3N5GRqzDBhjVXnCn44baHcz2GoZy5qLxtTh",
"signature":
↪"4ZhZdLAvaGneLU4K4b2eTgRQvbBjEZrtwo1qAhM9ar3A3weGEutbfNKM4WJ9JZnV8BXenx8JRGVNwpfxf3prGaxd",
"fee": 100000,
"blocksize": 369
}

```

GET /blocks/at/{height}

Returns contents of block at height {height}.

GET /blocks/seq/{from}/{to}

Returns contents of blocks ranging from {from} to {to}.

GET /blocks/seqext/{from}/{to}

Returns contents of blocks with additional transactions info ranging from {from} to {to}.

GET /blocks/signature/{signature}

Returns contents of block by its signature {signature}.

GET /blocks/address/{address}/{from}/{to}

Returns all blocks generated (mined) by address {address}.

GET /blocks/child/{signature}

Returns block inherited from block with signature {signature}.

GET /blocks/headers/at/{height}

Returns block header at height {height}.

GET /blocks/headers/seq/{from}/{to}

Returns block headers ranging from {from} to {to}.

GET /blocks/headers/last

Returns header of last block in the blockchain.

16.1.7 Consensus

Hint: The rules for generating queries to the node are given in module *How to use REST API*.

GET /consensus/algo

Returns type of consensus algorithm used on the network.

Method Response:

```
{
  "consensusAlgo": "Fair Proof-of-Stake (FairPoS)"
}
```

GET /consensus/settings

Returns consensus settings specified in node configuration file.

Method Response:

```
{
  "consensusAlgo": "Proof-of-Authority (PoA)",
  "roundDuration": "25 seconds",
  "syncDuration": "5 seconds",
  "banDurationBlocks": 50,
  "warningsForBan": 3
}
```

GET /consensus/minersAtHeight/{height}

Returns miner queue at height {height}.

Method Response:

```
{
  "miners": [
    "3Mx5sDq4NXef1BRzJRAofa3orYFfxLanxmd7",
    "3N2EsS6hJPYgRn7WfJHLJNnrsm92sUKcXkd",
    "3N2cQFfUDzG2iujBrFTnD2TAsCNoHDxYu8w",
    "3N6pfQJyqjLcMmbU7G5sNABLmSF5aFT4KTF",
    "3NBbipRYQmZFudFCoVJXg9JMkkyZ4DEdZNS"
  ],
}
```

(continues on next page)

(continued from previous page)

```
"height": 1
}
```

GET /consensus/miners/{timestamp}

Returns miner queue at timestamp {timestamp}.

Method Response:

```
{
  "miners": [
    "3Mx5sDq4NXef1BRzJRAofa3orYFxFxLanxmd7",
    "3N2EsS6hJPYgRn7WFJHLJNnrsm92sUKcXkd",
    "3N2cQFfUDzG2iuJBrFTnD2TAsCNohDxYu8w",
    "3N6pfQJyqjLCmMbU7G5sNABLmSF5aFT4KTF",
    "3NBbipRYQmZFudFCoVJXg9JMkkyZ4DEdZNS"
  ],
  "timestamp": 1547804621000
}
```

GET /consensus/bannedMiners/{height}

Returns a list of blocked miners at height {height}.

Method Response:

```
{
  "bannedMiners": [],
  "height": 1000
}
```

GET /consensus/basetarget/{blockId}

Returns value of 'base complexity' _ (basetarget) of creating block {blockId} .

GET /consensus/basetarget

Returns value of 'base complexity' _ (basetarget) of creating last block.

GET /consensus/generatingbalance/{address}

Returns generating balance available for minning node {address} - minimum participant balance including assets transferred to participant for leasing, for last 1000 blocks.

GET /consensus/generationsignature/{blockId}

Returns value of 'generation signature' of generating block {blockId}.

GET /consensus/generationsignature

Returns value of 'generation signature' of last block.

16.1.8 Contracts

Hint: The rules for generating queries to the node are given in module *How to use REST API*.

GET /contracts

Returns the contracts info.

Method Response

```
[
  {
    "contractId": "dmLT1ippM7tmfSC8u9P4wU6sBgHXGYy6JYxCq1CCh8i",
    "image": "registry.wvservices.com/wv-sc/may14_1:latest",
    "imageHash": "ff9b8af966b4c84e66d3847a514e65f55b2c1f63afcd8b708b9948a814cb8957",
    "version": 1,
    "active": false
  }
]
```

POST /contracts

Returns some parameters for the one or more contract IDs specified in the query.

Method Response

```
{
  "8vBJhy4eS8oEwCHC3yS3M6nZd5CLBa6XNt4Nk3yEEExG": [
    {
      "type": "string",
      "value": "Only description",
      "key": "Description"
    },
    {
      "type": "integer",
      "value": -9223372036854776000,
      "key": "key_may"
    }
  ]
}
```

GET /contracts/status/{id}

Returns the contract execution transaction status.

Method Response

```
[
  {
    "sender": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
    "senderPublicKey": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
    "txId": "4q5Q8vLeGBpcdQofZikyrrjHUS4pB1AB4qNEn2yHRKWU",
    "status": "Success",
    "code": null,
    "message": "Smart contract transaction successfully mined",
    "timestamp": 1558961372834,
    "signature":
    ↪"4gXy7qtzkaHHH6NkksnZ5pvn8juF65MvjQ9JgVztpgNwLNwuyyr27Db3gCh5YyADqZeBH72EyAkBouUoKvwJ3RQJ"
  },
  {
    "sender": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
    "senderPublicKey": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
    "txId": "4q5Q8vLeGBpcdQofZikyrrjHUS4pB1AB4qNEn2yHRKWU",
    "status": "Success",
    "code": null,
    "message": "Smart contract transaction successfully mined",
    "timestamp": 1558961376012,
    "signature":
    ↪"3Vhqc9DvNhMvFFtWnBuV4XwQ62ZcTAvLNZYmeGc7mGzMcncGZ3RLshDs393fnQu1WTh8CmL58YnvnjyULEEi5yorV"
  }
]
```

GET /contracts/{contractId}

Returns result of smart contract execution by its ID (contract creation transaction ID).

Method Response:

```
[
  {
    "key": "avg",
    "type": "string",
    "value": "3897.80146957"
  },
  {
    "key": "buy_price",
    "type": "string",
    "value": "3842"
  }
]
```

GET /contracts/executed-tx-for/{id}

Returns result of smart contract execution by ID of contract execution transaction.

Method Response:

```
{
  "type": 105,
  "id": "2UAHvs4KsfBbRVpM2dCigWtqUHuaNqou83CXy6DGDiRa",
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
  "senderPublicKey": "2YvzcVLrqLCqouVrFZynjfoTEuPNV9GrdauNpgdWXLsq",
  "fee": 500000,
  "timestamp": 1549365523980,
  "proofs": [
    "4BoG6wQnYyZWyUKzAwH5n1184tsEWUqUTWmXMExvvcu95xgk4UFB8iCnHJ4GhvJm86REB69hKM7s2WLAwTSXquAs"
  ],
  "version": 1,
  "tx": {
    "type": 103,
    "id": "ULc9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsJMVT2M",
    "fee": 500000,
    "timestamp": 1550591678479,
    "proofs": [
      ↪ "yeCRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxfj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
    "version": 1,
    "image": "stateful-increment-contract:latest",
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
    "contractName": "stateful-increment-contract",
    "params": [],
    "height": 1619
  },
  "results": []
}
```

GET /contracts/{contractId}/{key}

Returns smart contract execution value by its ID (contract creation transaction ID) and key {key}.

Method Response:

```
{
  "key": "updated",
  "type": "integer",
  "value": 1545835909
}
```


16.1.9 Crypto

Hint: The rules for generating queries to the node are given in module *How to use REST API*.

The description of text data encryption is presented in the subsection *Encrypting text data in transactions*.

Attention: Nodes of the **1.0** version and below use **Base58** encoding to encrypt/decrypt text data. Starting from the **1.0.2** version the **Base64** encoding is used. If there are different versions of nodes in the network, use the **crypto** methods as follows:

- In case of accepting data from newer version nodes by the older version node you need to convert data from the **encryptedText** string from **Base64** to **Base58**.
- In case of accepting data from older version nodes by the newer version node you need to convert data from the **encryptedText** string from **Base58** to **Base64**.

POST /crypto/encryptSeparate

Encrypts the text separately for the each recipient with the unique key.

Method Query

```
{
  "sender": "3MCUfX4P4U56hoQwSqXnLJenB6cDkxBjisL",
  "password": "some string as a password",
  "encryptionText": "some text to encrypt",
  "recipientsPublicKeys": [
    ↪ "5R65oLxp3iwPekwirA4VwwUXaySz6W6YKXBKBRl352pwwcpsFcjRHJ1VVHLp63LkrkxsNod64V1pffeizZ5i2qXc",
    ↪ "9LopMj2GqWxBYgnZ2gxaNxxXqxXHuWd6ZAdVqkprR1fFMNvDUHYUCwFxsB79B9sefgxNdqwNtqzuDS8Zmm48w3S"]
}
```

Method Response

```
{
  "encryptedText": "IZ5Kk5YNspMw1/jmlTizVxD6Nik=",
  "publicKey":
  ↪ "5R65oLxp3iwPekwirA4VwwUXaySz6W6YKXBKBRl352pwwcpsFcjRHJ1VVHLp63LkrkxsNod64V1pffeizZ5i2qXc",
  "wrappedKey":
  ↪ "uWVoxJAzrwwTDDSbphDS31TjSQX6CSWXivp3x34uE3XtnMqK9swoaZ3LyAgFDR7o6CfkgzFkWmTen4qAZewPfbBwR"
},
{
  "encryptedText": "F9u010RGvSEDe6dWm1pzJQ+3xqE=",
  "publicKey":
  ↪ "9LopMj2GqWxBYgnZ2gxaNxxXqxXHuWd6ZAdVqkprR1fFMNvDUHYUCwFxsB79B9sefgxNdqwNtqzuDS8Zmm48w3S",
  "wrappedKey":
  ↪ "LdздоKadUzBTmwcZGYgu1AM4YrbblR9Uh1MvQ3MPcLZUhCD9herz4dv1m6ssaVHPiBNUggqKnLZ6Si4Cc64UvhXBbG"
}
```

POST /crypto/encryptCommon

Encrypts the data with a single CEK key for all recipients and the CEK wraps into a unique KEK for the each recipient.

Method Query

```
{
  "sender": "3MCUfX4P4U56hoQwSqXnLJenB6cDkxBjisL",
  "password": "some string as a password",
  "encryptionText": "some text to encrypt",
  "recipientsPublicKeys": [
    ↪ "5R65oLxp3iwPekwirA4VwwUXaySz6W6YKXBKBRLL352pwwcpsFcjRHJ1VVHLp63LkrkxsNod64V1pffeizZ5i2qXc",
    "9LopMj2GqWxBYgnZ2gxaNxxXqxXHuWd6ZAdVqkprR1fFMNvDUHYUCwFxsB79B9sefgxNdqwNtqzuDS8Zmn48w3S"
  ]
}
```

Method Response

```
{
  "encryptedText": "NpCCig2i3jzo0xBnfqjfedbti8Y=",
  "recipientToWrappedStructure": {
    "5R65oLxp3iwPekwirA4VwwUXaySz6W6YKXBKBRLL352pwwcpsFcjRHJ1VVHLp63LkrkxsNod64V1pffeizZ5i2qXc":
    "M8pAe8HnKiWLE1HsC1ML5t8b7giWxiHfvagh7Y3F7rZL8q1tqMCJMYJo4qz4b3xjcuuUiV57tY3k7oSig53Aw1Dkkw",
    "9LopMj2GqWxBYgnZ2gxaNxxXqxXHuWd6ZAdVqkprR1fFMNvDUHYUCwFxsB79B9sefgxNdqwNtqzuDS8Zmn48w3S":
    "Doqn6gPvBBesSu2vdwgFYmbDHM4knEGMbqPn8Np76mNRRoZXLDioofyVbSSaTTEr4cvXwzEwVMugiy2wuzFWk3zCiT3"
  }
}
```

POST /crypto/decrypt

Decrypts the data. The decryption is available only if the message recipient's key is in the node's keystore.

Method Query

```
{
  "recipient": "3M5F8B1qxSY1W6kA2ZnQiDB4JTGz9W1jvQy",
  "password": "some string as a password",
  "encryptedText": "oiKFJijfid8HkjsjdhKKhud987d",
  "wrappedKey": "M5F8B1qxSY1W6kA2ZnQiDB4JTGzA2ZnQiDB4JTGz9W1jvQy"
  "senderPublicKey": "M5F8B1qxSY1W6kA2ZnQiDB4JTGzA2ZnQiDB4JTGz9W1jvQy",
}
```

Method Response

```
{
  "decryptedText": "some string for encryption",
}
```

16.1.10 Leasing

Hint: The rules for generating queries to the node are given in module *How to use REST API*.

GET /leasing/active/{address}

Returns list of lease creation transactions, in which {address} was involved as sender or recipient.

Method Response:

```
[
  {
    "type": 8,
    "id": "2jWhz6uGYsgvfoMzNR5EEGdi9eafyCA2zLFfkM4NP6T7",
    "sender": "3PP6vdkEwoif7AZDtSeSDtZcwiqSfhmwttE",
    "senderPublicKey": "DW9NKLyeyoEWDqJKhWv87EdFfTqpFtJBWoCqfCVwRhsY",
    "fee": 100000,
    "timestamp": 1544390280347,
    "signature":
    ↪"25kpwh7nYjRUtfbAbWYRyMDPCUCoyMoUuWTJ6vZQrXsZYXbdiWHa9iGscTTGnPfyegP82sNSfM2bXNX3K7p6D3HD",
    "version": 1,
    "amount": 31377465877,
    "recipient": "3P3RD3yJW2gQ9dSVwVVDVCQiFWqaLtZcyzH",
    "height": 1298747
  }
]
```

```
[
  {
    "type": 8,
    "id": "2jWhz6uGYsgvfoMzNR5EEGdi9eafyCA2zLFfkM4NP6T7",
    "sender": "3PP6vdkEwoif7AZDtSeSDtZcwiqSfhmwttE",
    "senderPublicKey": "DW9NKLyeyoEWDqJKhWv87EdFfTqpFtJBWoCqfCVwRhsY",
    "fee": 100000,
    "timestamp": 1544390280347,
    "signature":
    ↪"25kpwh7nYjRUtfbAbWYRyMDPCUCoyMoUuWTJ6vZQrXsZYXbdiWHa9iGscTTGnPfyegP82sNSfM2bXNX3K7p6D3HD",
    "version": 1,
    "amount": 31377465877,
    "recipient": "3P3RD3yJW2gQ9dSVwVVDVCQiFWqaLtZcyzH",
    "height": 1298747
  }
]
```

16.1.11 Node

Hint: The rules for generating queries to the node are given in module *How to use REST API*.

GET /node/config

Returns main node configuration parameters.

Method Response:

```
{
  "version": "0.6.6",
  "waves-crypto": false,
  "chainId": "D",
  "consensus": "POS",
  "minimumFee": {
    "1": 0,
    "3": 100000000,
    "4": 100000,
    "5": 100000000,
    "6": 100000,
    "7": 300000,
    "8": 100000,
    "9": 100000,
    "10": 100000,
    "11": 100000,
    "12": 100000,
    "13": 100000,
    "14": 100000000,
    "15": 100000000,
    "102": 0
  }
}
```

POST /node/stop

Query stops node.

GET /node/status

Returns main node configuration parameters.

Method Response:

```
{
  "blockchainHeight": 47041,
  "stateHeight": 47041,
  "updatedTimestamp": 1544709501138,
  "updatedAt": "2018-12-13T13:58:21.138Z"
}
```

GET /node/version

Returns version of application.

Method Response:

```
{
  "version": "Waves Enterprise v0.9.0"
}
```

16.1.12 Peers

Hint: The rules for generating queries to the node are given in module *How to use REST API*.

POST /peers/connect

Request to connect a new host to the node.

Method Query:

```
{
  "host": "127.0.0.1",
  "port": "9084"
}
```

Method Response:

```
{
  "hostname": "localhost",
  "status": "Trying to connect"
}
```

GET /peers/connected

Returns a list of connected nodes.

Method Response:

```
{
  "peers": [
    {
      "address": "52.51.92.182/52.51.92.182:6863",
      "declaredAddress": "N/A",
      "peerName": "zx 182",
      "peerNonce": 183759
    },
    {
      "address": "ec2-52-28-66-217.eu-central-1.compute.amazonaws.com/52.28.66.217:6863",
      "declaredAddress": "N/A",
      "peerName": "zx 217",
      "peerNonce": 1021800
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```
}  
]  
}
```

GET /peers/blacklisted

Returns a list of blocked nodes (at the network level).

GET /peers/all

Returns a list of all known nodes.

Method Response:

```
{  
  "peers": [  
    {  
      "address": "/13.80.103.153:6864",  
      "lastSeen": 1544704874714  
    }  
  ]  
}
```

POST /peers/clearblacklist

Clears the list of blocked nodes.

Method Query:

No need to transfer object of query.

Method Response:

```
{  
  "result": "blacklist cleared"  
}
```

GET /peers/suspended

Returns a list of suspended nodes.

Method Response:

```
[  
  {  
    "hostname": "/13.80.103.153",  
    "timestamp": 1544704754619  
  }  
]
```

POST /peers/identity

Gets the public key of the peer which is used by the node for the connection and the confidential data transfer.

Method Query:

```
{
  "address": "3NBVqYXrapgJP9atQccdBP AgJPwHDKkh6A8",
  "signature":
  ↪ "6RwMUQcwrxtKDgM4ANes9Amu5EJgyfF9Bo6nTpXyD89ZKMAcpCM97igbWf2MmLXLdqNxdUc68fd5TyRBEB6nqf"
}
```

Parameters:

- address - the blockchain address corresponding to the “privacy.owner-address” parameter in the node configuration file;
- signature - electronic signature of the “address” field value.

Method Response:

```
{
  "publicKey": "3NBVqYXrapgJP9atQccdBP AgJPwHDKkh6A8"
}
```

Parameters:

- publicKey - the peer public key associated with “privacy.owner-address” parameter in the configuration file. This parameter does not appear if the mode of the handshake checking turned off.

GET /peers/hostname/{address}

Gets the hostname and IP Address of the node by its address in the Waves Enterprise net.

Method Response:

```
{
  "hostname": "node1.we.io",
  "ip": "10.0.0.1"
}
```

GET /peers/allowedNodes

Gets the actual list of allowed participants at the request moment.

Method Response:

```
{
  "allowedNodes": [
    {
      "address": "3JNLQYuHYSHZiHr5KjJ89wwFJpDMdrAEJpj",
      "publicKey": "Gt3o1ghh2M2TS65UrHZCTJ82LLcMcBrxuaJyrgsLk5VY"
    },
    {
      "address": "3JLp8wt7rEUdn4Cca5Hp9jZ7w8T5XDAKicd",
      "publicKey": "J3ffCciVu3sustgb5vxmEHczACMR89Vty5ZBLbPn9xyg"
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```
  },
  {
    "address": "3JRY1cp7atRMBd8QQoswRpH7DLawM5Pnk3L",
    "publicKey": "5vn4UcB9En1XgY6w2N6e9W7bqFshG4SL2RLFqEWEbWxG"
  }
],
"timestamp": 1558697649489
}
```

16.1.13 Permissions

Hint: The rules for generating queries to the node are given in module *How to use REST API*.

GET /permissions/{address}

Returns roles (permissions) assigned to specified address {address} which are valid at the moment.

Method Response:

```
{
  "roles": [
    {
      "role": "miner"
    },
    {
      "role": "permissioner"
    }
  ],
  "timestamp": 1544703449430
}
```

GET /permissions/{address}/at/{timestamp}

Returns roles (permissions) assigned to specified address {address} which are valid at the moment {timestamp}.

Method Response:

```
{
  "roles": [
    {
      "role": "miner"
    },
    {
      "role": "permissioner"
    }
  ],
  "timestamp": 1544703449430
}
```


POST /permissions/addresses

Returns roles (permissions) assigned to specified address list which are valid at the moment.

Method Query:

```
{
  "addresses": [
    "3N2cQFfUDzG2iujBrFTnD2TAsCNoHDxYu8w", "3Mx5sDq4NXef1BRzJRAofa3orYFxFanxmd7"
  ],
  "timestamp": 1544703449430
}
```

Method Response:

```
{
  "addressToRoles": [
    {
      "address": "3N2cQFfUDzG2iujBrFTnD2TAsCNoHDxYu8w",
      "roles": [
        {
          "role": "miner"
        },
        {
          "role": "permissioner"
        }
      ]
    },
    {
      "address": "3Mx5sDq4NXef1BRzJRAofa3orYFxFanxmd7",
      "roles": [
        {
          "role": "miner"
        }
      ]
    }
  ],
  "timestamp": 1544703449430
}
```

16.1.14 PKI

Warning: The PKI methods can be used only with GOST cryptography.

Digital signature formats listed in the table below is used in PKI. The digital signature number in the table is consistent for the `sigType` field value.

Table 1: Digital signature formats

#	Digital signature format
1	CAdES-BES
2	CAdES-X Long Type 1
3	CAdES-T

POST /pki/sign

Hint: The rules for generating queries to the node are given in module *How to use REST API*.

This method creates a detached digital signature. `inputData` is data for generating a digital signature as an array of bytes in the **Base64** coding, `keystoreAlias` is a name of the key container of the digital signature private key. Also you need to specify a password in the `password` string.

Request example

```
{
  "inputData" : "SGVsbG8gd29ybGQh",
  "keystoreAlias" : "key1",
  "password" : "password",
  "sigType" : "CAES_X_Long_Type_1",
}
```

Answer example

```
{
  "signature" :
  ↪ "c2RmZ3NkZmZoZ2ZkZ2hmZGpkZ2ZoamhnZmtqaGdmamtKZmdoZmdk c2doZmQj s ndj f vnks dnj fn="
}
```

GET /pki/keystoreAliases

This method returns all the keystore aliases based on the GOST cryptography.

Answer example

```
{
  [
    "3Mq9crNkTFf8oRPyisgtf4TjBvZxo4BL2ax",
    "e19a135e-11f7-4f0c-9109-a3d1c09812e3"
  ]
}
```

POST /pki/verify

This method checks the detached digital signature for the sent data. The `extendedKeyUsageList` is optional and may contain an array of object identifiers - OID. It is useful for the determination of the scope of the certificate. Any node with query parameters can check the certificate.

Request example

```
{
  "inputData" : "SGVsbG8gd29ybGQh",
  "signature" : "c2RmZ3NkZmZoZ2ZkZ2hmZGpkZ2ZoamhnZmtqaGdmamtKZmdoZmdk c2doZmQ=",
  "sigType" : "CAES_X_Long_Type_1",
  "extendedKeyUsageList": [
    "1.2.643.7.1.1.1.1",
    "1.2.643.2.2.35.2"
  ]
}
```

Answer example

```
{
  "sigStatus" : "true"
}
```

Working with POST /pki/verify method

Using API *Post /pki/verify* method you can verify qualified digital signature. You need to install the root certificate on the node for proper using of API *Post /pki/verify*. The CA root certificate uniquely identifies the certification authority and is the basis in the chain of trust.

How to install a root certificate on a node

The root certificate is installing into the following Java directory:

```
-keystore /Library/Java/JavaVirtualMachines/jdk1.8.0_191.jdk/Contents/Home/jre/lib/
↳security/cacerts
```

The default password for the Java cacerts certificate store is **changeit**. You can change the password if you wish. Install certificates using the following command:

```
sudo keytool -import -alias testAliasCA_cryptopro -keystore /Library/Java/
↳JavaVirtualMachines/jdk1.8.0_191.jdk/Contents/Home/jre/lib/security/cacerts -file ~/
↳Downloads/cert.cer
```

16.1.15 Privacy

Hint: Rules of the creating requests to a node, see *How to use REST API* section.

POST /privacy/sendData

Writing the confidential data to the node store.

Method request:

```
{
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "password": "apgJP9atQccdBPA",
  "policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "type": "file",
  "info": {
    "filename": "Service contract #100/5.doc",
    "size": 2048,
    "timestamp": 1000000000,
    "author": "AIvanov@org.com",
    "comment": "some comments"
  }
},
```

(continues on next page)

(continued from previous page)

```
  "data":  
  ↪ "TWFuIGlziGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyByZWZzb24sIGJ1dCBieSB0aGlzIHhpbmd1bGFyIHBhc3Npb24gZnJvbSBvdGhl  
  ↪ ",  
    "hash": "FRog42mmzTA292ukng6PHoEK9Mpx9GZnrEHecfvpwmta"  
}
```

Parameters:

- sender - blockchain address for data broadcast (corresponds the “privacy.owner-address” parameter value in the node configuration file);
- password - access password to the private key of the node keystore;
- policyId - the group ID managing data forwarding;
- type - the type of the data;
- info - the information about the data;
- data - binary data;
- hash - data hash.

Method answer:

```
{  
  "senderPublicKey": "Gt3o1ghh2M2TS65UrHZCTJ82LLcMcBrxuaJyrsgLk5VY",  
  "policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",  
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",  
  "dataHash": "FRog42mmzTA292ukng6PHoEK9Mpx9GZnrEHecfvpwmta",  
  "proofs": [  
    "2jM4tw4uDmspuXUBt6492T7opuZskYhFGW9gkbq532BvLYRF6R.Jn3hVGNLMLK8JSM61GkVgYvYJg9UscAayEYfc"  
  ],  
  "fee": 110000000,  
  "id": "H3bdFTatppjnMmUe38YWh35Lmf4XDYrgsDK1P3KgQ5aa",  
  "type": 114,  
  "timestamp": 1571043910570  
}
```

GET /privacy/{policy-id}/recipients

Getting all addresses of participants, signed to the access group {policy-id}.

Method answer:

```
[  
  "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",  
  "3Mx2afTZ2KbRrLNbytyzTtXukZvqEB8SkW7"  
]
```

GET /privacy/{policy-id}/getHashes

Getting all addresses of participants, signed to the access group {policy-id}.

Method answer:

```
[
  "3GCFaCWtvLDnC9yX29YftMbn75gwf dwGsBn",
  "3GGxcmNyq8ZAHzK7or14Ma84khwW8peBohJ",
  "3GRlFi4rz3SniCuC7rbd9UuD2KUzyNh84pn",
  "3GKpShrQRtddF1yYhQ58ZnKMTnp2xdEzKqW"
]
```

GET /privacy/{policy-id}/getHashes

Getting the array of identified hashes which are written with association to the {policy-id}.

Method answer:

```
[
  "FdfdNBVqYXrapgJP9atQccdBP AgJPwHDKkh6A8",
  "eedfdNBVqYXrapgJP9atQccdBP AgJPwHDKkh6A"
]
```

GET /privacy/{policyId}/getData/{policyItemHash}

Getting the confidential data package by its identified hash.

Method answer:

```
c29tZV9iYXN1NjRfZW5jb2RlZF9zdHJpbmc=
```

GET /privacy/{policyId}/getInfo/{policyItemHash}

Getting the metadata for the confidential data package by the identified hash.

Method answer:

```
{
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "policy": "4gZnJvbSBvdGhlciBhbmltYWxzLzCB3aGljaC",
  "type": "file",
  "info": {
    "filename": "Contract №100/5.doc",
    "size": 2048,
    "timestamp": 1000000000,
    "author": "AIvanov@org.com",
    "comment": "Comment"
  },
  "hash": "e67ad392ab4d933f39d5723aeed96c18c491140e119d590103e7fd6de15623f1"
}
```

POST /privacy/forceSync

Forced getting the confidential data package by the identified hash.

Method answer:

```
{
  "result": "success" // or "error"
  "message": "Address '3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8' not in policy 'policyName'"
}
```

POST /privacy/getInfos

Getting the meta information array about private data according with the provided group ID and data hash.

Request example:

```
{ "policiesDataHashes":
  [
    {
      "policyId": "somepolicyId_1",
      "datahashes": [ "datahash_1","datahash_2" ]
    },
    {
      "policyId": "somepolicyId_2",
      "datahashes": [ "datahash_3","datahash_4" ]
    }
  ]
}
```

Method answer:

```
{
  "policiesDataInfo": [
    {
      "policyId": "somepolicyId_1",
      "datasInfo": [
        {
          "hash": "e67ad392ab4d933f39d5723aeed96c18c491140e119d590103e7fd6de15623f1",
          "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
          "type": "file",
          "info": {
            "filename": "Contract №100/5.doc",
            "size": 2048,
            "timestamp": 1000000000,
            "author": "AIvanov@org.com",
            "comment": "Comment"
          }
        }
      ],
    },
    {
      "hash": "e67ad392ab4d933f39d5723aeed96c18c491140e119d590103e7fd6de15623f1",
      "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
      "type": "file",
      "info": {
```

(continues on next page)

(continued from previous page)

```

        "filename": "Contract №101/5.doc",
        "size": "2048",
        "timestamp": 1000000000,
        "author": "AIvanov@org.com",
        "comment": "Comment"
    }
}
]
}

```

16.1.16 Transactions

Hint: The rules for generating node queries are given in module *How to use REST API*.

GET /transactions/info/{id}

Query transaction information by its ID.

Query Parameters:

"id" - Transaction ID

Method Response:

```

{
  "type": 4,
  "id": "52GG9U2e6foYRkp5vAzsTQ86aDAABfRJ7synz7ohBp19",
  "sender": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
  "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMhM3Uki7pLw",
  "recipient": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
  "assetId": "E9yZC4cVhCDfbjFJc9CqkAtkoFy5KaCe64iaxHM2adG",
  "amount": 100000,
  "fee": 100000,
  "timestamp": 1479313236091,
  "attachment": "string",
  "signature":
  ↪ "GknccUA79dBcwWgKjqB7vYHcnsj7caYETfncJhRkkaetbQon7DxbpMmvK9LYqUkirJp17geBJCRTNkHEoAjsUm",
  "height": 7782
}

```

GET /transactions/address/{address}/limit/{limit}

Returns latest {limit} transactions from address {address}.

Method Response:

```
[
  [
    {
      "type": 2,
      "id":
      ↪ "4XE4M9eSoVWVdHwDYXqZsXhEc4q8PH9mDMUBegCSBBVHJyP2Yb1ZoGi59c1Qzq2TowLmymLNkFQjWp95CddnyBW",
      "fee": 100000,
      "timestamp": 1479313097422,
      "signature":
      ↪ "4XE4M9eSoVWVdHwDYXqZsXhEc4q8PH9mDMUBegCSBBVHJyP2Yb1ZoGi59c1Qzq2TowLmymLNkFQjWp95CddnyBW",
      "sender": "3NBVqYXrapgJP9atQccdBPAGJPwHDKkh6A8",
      "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMhM3Uki7pLw",
      "recipient": "3N9iRMou3pgmyPbFZn5QZQvBTQBkL2fR6R1",
      "amount": 1000000000
    }
  ]
]
```

GET /transactions/unconfirmed

Returns all unconfirmed transactions from node utx-pool.

Method Response:

```
[
  {
    "type": 4,
    "id": "52GG9U2e6foYRkp5vAzsTQ86aDAABfRJ7synz7ohBp19",
    "sender": "3NBVqYXrapgJP9atQccdBPAGJPwHDKkh6A8",
    "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMhM3Uki7pLw",
    "recipient": "3NBVqYXrapgJP9atQccdBPAGJPwHDKkh6A8",
    "assetId": "E9yZC4cVhCDfbjFJCc9CqkAtkoFy5KaCe64iaxHM2adG",
    "amount": 100000,
    "fee": 100000,
    "timestamp": 1479313236091,
    "attachment": "string",
    "signature":
    ↪ "GknccUA79dBcwWgKjqB7vYHcnsj7caYETfncJhRkkaetbQon7DxbpMmvK9LYqUkirJp17geBJCRTNkHEoAjsUm"
  }
]
```


GET /transactions/unconfirmed/size

Return the number of transactions available in UTX pool.

GET /unconfirmed/info/{id}

Query transaction details from UTX pool by its ID.

POST /transactions/calculateFee

Calculates fee amount for transferred transaction.

Query Parameters

```
"type" - Transaction type
"senderPublicKey" - Public key of sender
"sender" is ignored
"fee" is ignored
and all the other parameters appropriate for a transaction of the given type.
```

Method Query

```
{
  "type": 10,
  "timestamp": 1516171819000,
  "sender": "3MtrNP7AkTRuBhX4CBti6iT21pQpEnmHtyw",
  "alias": "ALIAS",
}
```

or

```
{
  "type": 4,
  "sender": "3MtrNP7AkTRuBhX4CBti6iT21pQpEnmHtyw",
  "recipient": "3P8JYPHrnXSfsWP1LVXySdzU1P83FE1ssDa",
  "amount": 1317209272,
  "feeAssetId": "8LQW8f7P5d5PZM7GtZEBgaqRPGSszS3DfPuiXrURJ4AJS",
  "attachment": "string"
}
```

Method Response

```
{
  "feeAssetId": null,
  "feeAmount": 10000
}
```

or

```
{
  "feeAssetId": "8LQW8f7P5d5PZM7GtZEBgaqRPGSszS3DfPuiXrURJ4AJS",
  "feeAmount": 10000
}
```

POST /transactions/sign

Signs a transaction with sender's private key stored in node keystore. After signing, method response must be sent to method input *Broadcast*.

It is necessary to enter the password into the `password` field in order to sign requests with the key from keystore node.

Sample queries

ID	Transaction type
3	<i>Issue</i>
4	<i>Transfer</i>
5	Reissue
6	Burn
7	Exchange
8	Lease
9	Lease Cancel
10	<i>Alias</i>
11	Mass Transfer
12	<i>Data</i>
13	<i>Set Script</i>
14	Set Sponsorship
101	Permission (for Genesis block)
102	<i>PermissionTransaction</i>
103	<i>CreateContractTransaction</i>
104	<i>CallContractTransaction</i>
105	<i>ExecutedContractTransaction</i>
106	<i>DisableContractTransaction</i>
110	<i>GenesisRegisterNode Transaction</i>
111	<i>RegisterNode Transaction</i>
112	<i>CreatePolicy Transaction</i>
113	<i>UpdatePolicy Transaction</i>
114	<i>PolicyDataHash Transaction</i>

3. Issue

```
{
  "type": 3,
  "version":2,
  "name": "Test Asset 1",
  "quantity": 100000000000,
  "description": "Some description",
  "sender": "3FSCKyfFo3566zwiJjSFLBwKvd826KXUaqR",
  "decimals": 8,
  "reissuable": true,
  "fee": 100000000
}
```

4. Transfer

```
{
  "type": 4,
  "version": 2,
  "sender": "3M6dRZXaJY9oMA3fJKhMALyYKt13D1aimZX",
  "password": "",
  "recipient": "3M6dRZXaJY9oMA3fJKhMALyYKt13D1aimZX",
  "amount": 40000000000,
  "fee": 100000
}
```

10. Alias

```
{
  "type": 10,
  "version": 2,
  "fee": 100000,
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "alias": "hodler"
}
```

12. Data

```
{
  "type": 12,
  "version": 1,
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
  "author": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "data":
  [
    {
      "key": "objectId",
      "type": "string",
      "value": "obj:123:1234"
    }
  ],
  "fee": 100000
}
```

13. Set Script

```
{
  "type": 13,
  "version": 1,
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "fee": 100000,
  "name": "faucet",
  "script": "base64:AQQAAAAHJG1hdGNoMAUAAAAcDHgG+RXSszQ=="
}
```

102. PermissionTransaction

Sample query

```
{
  "type": 102,
  "sender": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",

```

(continues on next page)

(continued from previous page)

```

"senderPublicKey": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
"fee": 0,
"proofs": [],
"target": "3GPtj5osoYqHpyfmsFv7BMiyKsVzbG1ykfL",
"opType": "add",
"role": "contract_developer",
"dueTimestamp": null
}

```

103. CreateContractTransaction

Sample query

```

{
  "type": 103,
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
  "image": "localhost:5000/sum-contract-kv",
  "params": [],
  "imageHash": "930d18dacb4f49e07e2637a62115510f045da55ca16b9c7c503486828641d662",
  "fee": 500000
}

```

Sample response

```

{
  "type": 103,
  "id": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
  "senderPublicKey": "2YvzcVLrQLCqouVrFZynjfoTEuPNV9GrdauNpgdWXLsq",
  "fee": 500000,
  "timestamp": 1549443811183,
  "proofs": [
    "YSomSCKBhQWHKHR8f8ZMp7EzuA6Uouu1oq5WA5VDiZ8o2adL4XMQP3jgcckctjGCEpnTnCjm5bABZG486CVR5ZM"
  ],
  "version": 1,
  "image": "localhost:5000/sum-contract-kv",
  "imageHash": "930d18dacb4f49e07e2637a62115510f045da55ca16b9c7c503486828641d662",
  "params": []
}

```

104. CallContractTransaction

Sample query

```

{
  "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
  "fee": 10,
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
  "type": 104,
  "version": 1,
  "params": [
    {
      "type": "integer",
      "key": "a",
      "value": 1
    },
    {

```

(continues on next page)

(continued from previous page)

```

        "type": "integer",
        "key": "b",
        "value": 100
    }
]
}

```

Sample response

```

{
  "type": 104,
  "id": "9fBrL2n5TN473g1gNfoZqaAqAsAJCuHRHYxZpLexL3VP",
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
  "senderPublicKey": "2YvzcVLrqlCqouVrFZynjfotEuPNV9GrdauNpgdWXLsq",
  "fee": 10,
  "timestamp": 1549365736923,
  "proofs": [
    "2q4cTBhdKEDkFxr7iYaHPAv1dzaKo5rDaTxPF5VHryyYTXxTPvN9Wb3YrsDYixKiUPXBnAyXzEcnKPFRCW9xVp4v"
  ],
  "version": 1,
  "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
  "params": [
    {
      "key": "a",
      "type": "integer",
      "value": 1
    },
    {
      "key": "b",
      "type": "integer",
      "value": 100
    }
  ]
}

```

105. ExecutedContractTransaction

Sample response

```

{
  "type": 105,
  "id": "2UAHvs4KsfBbRVPm2dCigWtqUHuaNQou83CXy6DGDiRa",
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
  "senderPublicKey": "2YvzcVLrqlCqouVrFZynjfotEuPNV9GrdauNpgdWXLsq",
  "fee": 500000,
  "timestamp": 1549365523980,
  "proofs": [
    "4BoG6wQnYyZWyUKzAwh5n1184tsEWUqUTWmXMExvvCU95xgk4UFB8iCnHJ4GhvJm86REB69hKM7s2WLAwTSXquAs"
  ],
  "version": 1,
  "tx": {
    "type": 103,
    "id": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
    "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
    "senderPublicKey": "2YvzcVLrqlCqouVrFZynjfotEuPNV9GrdauNpgdWXLsq",
    "fee": 500000,

```

(continues on next page)

(continued from previous page)

```

    "timestamp": 1549365501462,
    "proofs": [
      "2ZK1Y1ecfQXeWsS5sfcTLM5W1KA3kwi9Up2H7z3Q6yVzMeGxT9xWJT6jREQsmuDBcvk3DCCiWBdFHaxazU8pbo41"
    ],
    "version": 1,
    "image": "localhost:5000/contract256",
    "imageHash": "930d18dadb4f49e07e2637a62115510f045da55ca16b9c7c503486828641d662",
    "params": []
  },
  "results": []
}

```

106. DisableContractTransaction

Sample query

```

{
  "senderPublicKey":
  ↪ "42jj4GA89Z2SnegzpxoocmWZChrpqhGVKcJUctAGWJB2oSTQrZCQyzbvr iDSFu5ZmCBsFutDyg9ES6WqqULyV5e",
  "contractId": "Fz3wqAwwcPMT4M1q6H7crLkTToFJvbeLSvqjaU4ZwMpg",
  "fee": 0,
  "timestamp": 1549474811381,
  "proofs": [
    "4Dny2XwkXmoLN7emoqdFdvvKdgnCBuA3XwGgBiWNkZBFXDPrfz36Cyp2CbpjrLBadCnuobbkK5wyM41FGU6yp6h"
  ],
  "type": 106
}

```

Sample response

```

{
  "type" : 106,
  "id" : "BwcVQeC9CdmeYxiWydc5NK1MSgqPqQmWYy4PJ6eqZDtP",
  "sender" : "3HhXnbMuZAaCRr9L9hWSKwfnrcDR6CThJVB",
  "senderPublicKey" :
  ↪ "42jj4GA89Z2SnegzpxoocmWZChrpqhGVKcJUctAGWJB2oSTQrZCQyzbvr iDSFu5ZmCBsFutDyg9ES6WqqULyV5e",
  "fee" : 0,
  "timestamp" : 1549474811381,
  "proofs" : [
  ↪ "4Dny2XwkXmoLN7emoqdFdvvKdgnCBuA3XwGgBiWNkZBFXDPrfz36Cyp2CbpjrLBadCnuobbkK5wyM41FGU6yp6h" ],
  "version" : 1,
  "contractId" : "Fz3wqAwwcPMT4M1q6H7crLkTToFJvbeLSvqjaU4ZwMpg"
}

```

110. GenesisRegisterNode

Sample query

```

{
  "type": 110,
  "id": "2Xgbsqgfbbp5fiq4nsaAoTkQsXc399tXdnKom8prEZqPW2Q7xZKNKCCqpkymtmJMgYlPvwynbxHPTFPFEfFdyLpJ",
  "fee": 0,
  "timestamp": 1489352400000,
  "signature":
  ↪ "2Xgbsqgfbbp5fiq4nsaAoTkQsXc399tXdnKom8prEZqPW2Q7xZKNKCCqpkymtmJMgYlPvwynbxHPTFPFEfFdyLpJ",
  "targetPublicKey": "3JNLQYUHYSHZiHr5KjJ89wwfJpDmdrAEJpj",
  "target": "3JNLQYUHYSHZiHr5KjJ89wwfJpDmdrAEJpj"
}

```

Sample response

```
{
  "signature":
  ↪ "2Xgbsqgfbp5fiq4nsaAoTkQsXc399tXdnKom8prEZqPW2Q7xZKNKCCqpkymtmJMgYLpvwynbxHPTFPFEfFdyLpJ",
  "fee": 0,
  "id": "2Xgbsqgfbp5fiq4nsaAoTkQsXc399tXdnKom8prEZqPW2Q7xZKNKCCqpkymtmJMgYLpvwynbxHPTFPFEfFdyLpJ",
  "type": 110,
  "targetPublicKey": "3JNLQYyHYSHZiHr5KjJ89wwFJpDMdrAEJpj",
  "timestamp": 1489352400000,
  "target": "3JNLQYyHYSHZiHr5KjJ89wwFJpDMdrAEJpj",
  "height": 1
}
```

111. RegisterNode

Sample query

```
{
  "type": 111,
  "opType": "add",
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUGeytUUz",
  "targetPubKey": "apgJP9atQccdBPAgJPwH3NBVqYXrapgJP9atQccdBPAgJPwHapgJP9atQccdBPAgJPwHDKkh6A8",
  "nodeName": "Node #1",
  "fee": 500000,
  "timestamp": 1557239100
}
```

112. CreatePolicy

Sample query

```
{
  "type": 112,
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUGeytUUz",
  "description": "Policy for rural internal nodes",
  "policyName": "Policy name",
  "timestamp": 1000000000,
  "recipients": [ "3HsvTtjim3FmV21HWQ1LurMhFzjut7Aa1Ac", "3HYW75PpAeVukmbYo9PQ3mzSHdKUGeytUUz" ],
  "owners": [ "3HYW75PpAeVukmbYo9PQ3mzSHdKUGeytUUz", "3HYW75PpAeVukmbYo9PQ3mzSHdKUGeytUUz" ]
}
```

113. UpdatePolicy

Sample query

```
{
  "type": 113,
  "policyId": "45n2BC8TmobhH7zbog8Zsr1mcHSd1uU84UvWEoSbqQBH", // id of the existing policy
  ↪ otherwise it occurs the error "Object with policyId = <request id> does not exist"
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUGeytUUz",
  "timestamp": 1000000000,
  "opType": "add", // or "remove" when removing participants from policy
  "recipients": [ "3HsvTtjim3FmV21HWQ1LurMhFzjut7Aa1Ac", "3HYW75PpAeVukmbYo9PQ3mzSHdKUGeytUUz" ],
  "owners": [ "3HYW75PpAeVukmbYo9PQ3mzSHdKUGeytUUz", "3HYW75PpAeVukmbYo9PQ3mzSHdKUGeytUUz" ]
}
```

114. PolicyDataHash

Sample query

```
{
  "type": 114,
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "timestamp": 1000000000,
  "policyId": "45n2BC8TmobhH7zbog8ZsR1mcHSd1uU84UvWEoSbqQBH",
  "hash": "ad2a814482df0dd0d2cf6321f535be720caa7b3aa1289b0575f60d7a5e109631",
}
```

POST /transactions/broadcast

Sends a signed transaction to blockchain.

Method Query

```
{
  "type": 10,
  "senderPublicKey": "G6h72icCSjdW2A89QWDb37hyXJoYKq3XuCUJY2joS3EU",
  "fee": 100000000,
  "timestamp": 46305781705234713,
  "signature":
  ↪ "4gQyPXzJFEzMbsCd9u5n3B2WauEc4172ssyrXCL882oNa8NfNihnpKianHXrHWnZs1RzDLbQ9rcRYnSqxKWfEPJG",
  "alias": "dajzmj6gfuzmbfnhamsbuxivc"
}
```

Method Response

```
{
  "type": 10,
  "id": "9q7X84wFuVvKqRdDQeWbtBmpsHt9SXFbvPptUuKBVxxr",
  "sender": "3MtrNP7AkTRuBhX4CBti6iT21pQpEnmHtyw",
  "senderPublicKey": "G6h72icCSjdW2A89QWDb37hyXJoYKq3XuCUJY2joS3EU",
  "fee": 100000000,
  "timestamp": 46305781705234713,
  "signature":
  ↪ "4gQyPXzJFEzMbsCd9u5n3B2WauEc4172ssyrXCL882oNa8NfNihnpKianHXrHWnZs1RzDLbQ9rcRYnSqxKWfEPJG",
  "alias": "dajzmj6gfuzmbfnhamsbuxivc"
}
```


16.1.17 Utils

Hint: The rules for generating queries to the node are given in module *How to use REST API*.

POST /utils/hash/secure

Returns secure (double) hash of specified message.

Method query:

```
ridethewaves!
```

Method response:

```
{
  "message": "ridethewaves!",
  "hash": "H6nsiifwYKYEx6YzYD7woP1XCn72RVvx6tC1zjjLXqsu"
}
```

POST /utils/hash/fast

Returns hash of specified message.

Method query:

```
ridethewaves!
```

Method response:

```
{
  "message": "ridethewaves!",
  "hash": "DJ35ymschUFDmqCnDJewjcnVExVkWgX7mJDxhFy9X8oQ"
}
```

POST /utils/script/compile

Response parameters:

```
"script" - скрипт в формате Base64
"complexity" - сложность скрипта
"extraFee" - комиссия за исходящие транзакции, установленные скриптом
```

Method query:

```
let x = 1
(x + 1) == 2
```

Method response:

```
{
  "script":
  ↪ "3rbFDtbPwAvSp2vBvqGfGR9nRS1nBVnfuSCN3HxSZ7fVRpt3tuFG5JSmyTmvHPxYf34So cMRkRKFgzTtXXnnv7upRHXJzZrLSQo8tUW6yMtEiZ
  ↪",
  "complexity": 11,
  "extraFee": 10001
}
```

or

Method query:

```
x == 1
```

Method response:

```
{
  "error": "Typecheck failed: A definition of 'x' is not found"
}
```

POST /utils/script/estimate

Decoding base64 script.

Method query:

```
AQQAAAABeAAAAAAAAAAAAAQkAAAAAAAAACCQAAZAAAAIFAAAAAXgAAAAAAAAAAAAEAAAAAAAAAAAAAJdecYi
```

Method response:

```
{
  "script":
  ↪ "3rbFDtbPwAvSp2vBvqGfGR9nRS1nBVnfuSCN3HxSZ7fVRpt3tuFG5JSmyTmvHPxYf34So cMRkRKFgzTtXXnnv7upRHXJzZrLSQo8tUW6yMtEiZ
  ↪",
  "scriptText": "FUNCTION_CALL(FunctionHeader(=,List(LONG, LONG)),List(CONST_LONG(1), CONST_
  ↪LONG(2)),BOOLEAN)",
  "complexity": 11,
  "extraFee": 10001
}
```

GET /utils/time

Returns current node time.

Method response:

```
{
  "system": 1544715343390,
  "NTP": 1544715343390
}
```


POST /utils/reload-wallet

Reloads node keystore. Runs if new key pair was created in keystore without restarting node.

Method response:

```
{
  "message": "Wallet reloaded successfully"
}
```

16.2 Authorization service REST API methods

You can read more about working with REST API in *this* section. The authorization service REST API methods are accessed via HTTPS protocol. Methods are closed by authorization and are marked with the  icon.

16.2.1 GET /status

Getting the authorization service status.

Method answer

```
{
  "status": "OK"
}
```

16.2.2 POST /v1/user

Registering a new user.

Method request

```
{
  "username": "string",
  "password": "string",
  "locale": "string"
}
```

Method answer

```
{
  "access_token": "string",
  "refresh_token": "string",
  "token_type": "string"
}
```

16.2.3 GET /v1/user/profile



Getting user data.

Method answer

```
{
  "id": "string",
  "name": "string",
  "locale": "en",
  "addresses": [
    "string"
  ],
  "roles": [
    "string"
  ]
}
```

16.2.4 POST /v1/user/address



Getting an user address.

Method request

```
{
  "address": "string",
  "type": "string"
}
```

Method answer

```
{
  "addressId": "string"
}
```

16.2.5 GET /v1/user/doesEmailExist

Checking an user email address.

Method answer

```
{
  "exist": true
}
```

16.2.6 POST /v1/user/password/restore

Restoring an user account password.

Method request

```
{
  "email": "string"
}
```

Method answer

```
{
  "email": "string"
}
```

16.2.7 POST /v1/user/password/reset

Resetting an user password.

Method request

```
{
  "token": "string",
  "password": "string"
}
```

Method answer

```
{
  "userId": "string"
}
```

16.2.8 GET /v1/user/confirm/{code}

Entering a confirmation code to reset an user account password.

16.2.9 POST /v1/user/resendEmail

Resending a password recovery code to the specified email address.

Method request

```
{
  "email": "string"
}
```

Method answer

```
{
  "email": "string"
}
```

16.2.10 POST /v1/auth/login

Registering a new user in the authorization service.

Method request

```
{
  "username": "string",
  "password": "string",
  "locale": "string"
}
```

Method answer

```
{
  "access_token": "string",
  "refresh_token": "string",
  "token_type": "string"
}
```

16.2.11 POST /v1/auth/token



Registering external services and applications in the authorization service.

Method request

```
{
  "token": "string"
}
```

Method answer

```
{
  "access_token": "string",
  "refresh_token": "string",
  "token_type": "string"
}
```

16.2.12 POST /v1/auth/refresh

Getting a new refresh token.

Method request

```
{
  "token": "string"
}
```

Method answer

```
{
  "access_token": "string",
  "refresh_token": "string",
  "token_type": "string"
}
```

16.2.13 GET /v1/auth/publicKey

Getting the authorization service public key.

Method answer

```
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAQEAt7d90j/ZQTkkjf4UuMfUu
QIFDTYxYf6QBKMVJnq/wXyPYYkV8HVFFyzCaEciv3CXmBH77sXnuTlrEtV7zHB
KvV870HmZuazjIgzVSk0n0Y7F8UUUVNxn1zVD1dPs0GJ6orM41DnC1W65mCrP3bjn
fV4RbmykN/lk7McA6EsMcLEGbKkFhmeq2Nk4hn2CQvoTkupJU0CP1dh04bq1lQ7
Ffj9K/FJq73wSXDoH+qqdRG9sfrtgrhtJHerruhv3456eOzyAcD08+sJUQFKY80B
SZMEndVzFS2ub9Q8e7BfcNxmQPM4PhH05wuTqL32qt3uJBx20I41u30ND44ZrDJ
BbVog73oPjRYXj+kTbwUZI66SP4aLcQ8syPQyLwqKk5DtLRozSN00IrupJJ/pwZs
9zPEggL91T0rirbEhG1f5U8/6XN8GVXX4iMk2fD8FHLFJuXCD70j4JC2iWfFDC6a
uUkwUfqfjJB8BzIHkncoq0ZbpideE21TW1+svuEu/wyP5rNlyMiE/e/fZQqM2+o0
cH5Qow6HH35BrloCSZciutUcd1U7YPqESJ5tryy1xn9bsMb+On1ocZTtvec/ow4M
RmnJwm0j1nd+cc19OKLG5/boeA+2zqWu0jCbWR9c0oCmgbhucZChaHTBEAKDwCsC
VRz5qD6FPpePpTQDb6ss3bkCAwEAAQ==
-----END PUBLIC KEY-----
```

16.3 REST API methods for the data service

16.3.1 Transactions

GET /transactions

Returns a list of transactions matching the search query criteria and filters applied.

Important: It is returned a maximum of 500 transactions for the API **GET /transactions** method request.

Method Response:

```
[
  {
    "id": "string",
    "type": 0,
    "height": 0,
    "fee": 0,
    "sender": "string",
    "senderPublicKey": "string",
    "signature": "string",
    "timestamp": 0,
    "version": 0
  }
]
```

(continues on next page)

(continued from previous page)

```
}  
]
```

GET /transactions/count

Returns the number of transactions matching the search query criteria and filters applied.

Method Response:

```
{  
  "count": "string"  
}
```

GET /transactions/id/{id}

Returns transaction by ID {id}.

Method Response:

```
{  
  "id": "string",  
  "type": 0,  
  "height": 0,  
  "fee": 0,  
  "sender": "string",  
  "senderPublicKey": "string",  
  "signature": "string",  
  "timestamp": 0,  
  "version": 0  
}
```

16.3.2 Token assets

GET /assets

Returns a list of token assets available in the blockchain (as token issue transactions).

Method Response:

```
[  
  {  
    "id": "string",  
    "type": 0,  
    "height": 0,  
    "fee": 0,  
    "sender": "string",  
    "senderPublicKey": "string",  
    "signature": "string",  
    "timestamp": 0,  
    "version": 0,  
    "assetId": "string",  
    "name": "string",  
  }  
]
```

(continues on next page)

(continued from previous page)

```

    "description": "string",
    "quantity": 0,
    "decimals": 0,
    "reissuable": true
  }
]

```

16.3.3 Users

GET /users

Returns a list of users matching the search query criteria and filters applied.

Method Response:

```

[
  {
    "address": "string",
    "aliases": [
      "string"
    ],
    "registration_date": "string",
    "permissions": [
      "string"
    ],
    "balances": [
      {
        "assetId": "string",
        "amount": 0
      }
    ]
  }
]

```

GET /users/{userAddress}

Returns information about the user as per user's address.

Method Response:

```

{
  "address": "string",
  "aliases": [
    "string"
  ],
  "registration_date": "string",
  "permissions": [
    "string"
  ],
  "balances": [
    {
      "assetId": "string",
      "amount": 0
    }
  ]
}

```

(continues on next page)

(continued from previous page)

```
]
}
```

16.3.4 Blocks

GET /blocks/{height}

Returns the block at the specified height.

Method Response:

```
{
  "version": 0,
  "timestamp": 0,
  "reference": "string",
  "nxt-consensus": {
    "base-target": 0,
    "generation-signature": "string"
  },
  "features": [
    0
  ],
  "generator": "string",
  "signature": "string",
  "blocksize": 0,
  "transactionCount": 0,
  "fee": 0,
  "height": 0,
  "transactions": [
    {
      "id": "string",
      "type": 0,
      "height": 0,
      "fee": 0,
      "sender": "string",
      "senderPublicKey": "string",
      "signature": "string",
      "timestamp": 0,
      "version": 0
    }
  ]
}
```

16.3.5 Data transactions

GET /api/v1/txIds/{key}

Returns a list of data transaction ID's containing the specified key.

Method Response:

```
[
{
```

(continues on next page)

(continued from previous page)

```
[
  "id": "string"
]
```

GET /api/v1/txIds/{key}/{value}

Returns a list of data transaction ID's containing the specified key and value.

Method Response:

```
[
  {
    "id": "string"
  }
]
```

GET /api/v1/txData/{key}

Returns data transaction bodies containing the specified key.

Method Response:

```
[
  {
    "id": "string",
    "type": "string",
    "height": 0,
    "fee": 0,
    "sender": "string",
    "senderPublicKey": "string",
    "signature": "string",
    "timestamp": 0,
    "version": 0,
    "key": "string",
    "value": "string",
    "position_in_tx": 0
  }
]
```

GET /api/v1/txData/{key}/{value}

Returns data transaction bodies containing the specified key and value.

Method Response:

```
[
  {
    "id": "string",
    "type": "string",
    "height": 0,
    "fee": 0,
    "sender": "string",
    "senderPublicKey": "string",
```

(continues on next page)

(continued from previous page)

```

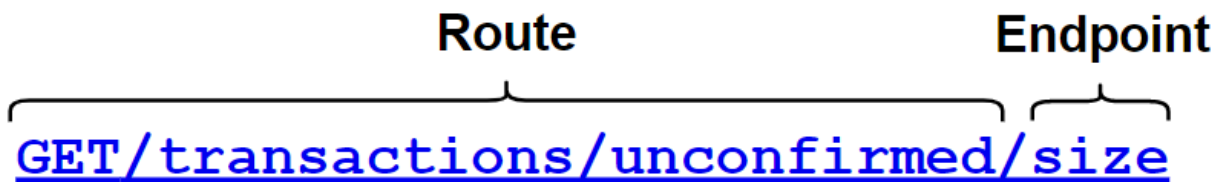
"signature": "string",
"timestamp": 0,
"version": 0,
"key": "string",
"value": "string",
"position_in_tx": 0
}
]

```

16.4 How to use REST API

All API methods are including GET, POST or DELETE HTTPS requests to URL <https://yournetwork.com/node-N/api-docs/swagger.json> using the set of parameters. The requests groups with routes and endpoints are selected in the Swagger interface. The route is the URL of the HTTP method, and the endpoint is the final part of the route, this is the access to the method. Example:

URL to the HTTP-method



For requests requiring the following actions, mandatory authorization by `api-key-hash` is required. The authorization type is specified in the node configuration file. If `api-key-hash` authorization type is selected, it is necessary to specify the value of the secret phrase, the hash of which is wrote in the node configuration file (`rest-api.api-key-hash` field).

- access to the node keystore (for example, sign method);
- access to operations with confidential data access groups;
- access to the node configuration.

When authorized by token, the value of `access` token is specified in the corresponding field. If token authorization is selected, then all REST API methods for node access are closed.

16.5 Authorization methods

Depending on the authorization method, different values are specified to get the access to the node REST API.

Available authorizations

X

OAuth2 Bearer (apiKey)

Name: Authorization

In: header

Value:

Authorize

Close

ApiKey or PrivacyApiKey (apiKey)

Name: X-API-Key

In: header

Value:

Authorize

Close

- OAuth2 Bearer (apiKey) - an **access** token value.
- ApiKey or PrivacyApiKey (apiKey) - **api-key-hash** value for both access to the node REST API and *privacy* methods.

16.5.1 api-key-hash authorization

The **api-key-hash** generation is happening during the *node configuration*. The value of the field **rest-api.api-key-hash** can be also generated using the */utils/hash/secure* method of node REST API. It is required to specify the access password to the keystore in the **password** field of the POST */transaction/sign* request for signing requests by the node keystore key.

Sample query:

```
curl -X POST
--header 'Content-Type: application/json'
--header 'Accept: application/json'
--header 'X-API-Key: 1' -d '1' 'http://2.testnet-pos.com:6862/transactions/calculateFee'
```

16.5.2 Token authorization

If the *authorization service* is used, the client receives a pair of tokens - **refresh** and **access** - for the node and other services access. Tokens can be obtained via the authorization service REST API.

DOCKER SMART-CONTRACTS

17.1 Example of starting a contract

Hint: Technical description of contracts implementation is given in module *Docker Smart Contracts*.

17.1.1 Description of program logic

This module reviews an example of how to create and run a simple smart contract. The contract performs increment the number transferred to the contract entry in *call-transactions*.

Program listing `contract.py` on Python:

```
import json
import os
import requests
import sys

def find_param_value(params, name):
    for param in params:
        if param['key'] == name: return param['value']
    return None

def print_success(results):
    print(json.dumps(results, separators=(',', ':')))

def print_error(message):
    print(message)
    sys.exit(3)

def get_value(contract_id):
    node = os.environ['NODE_API']
    if not node:
        print_error("Node REST API address is not defined")
    token = os.environ["API_TOKEN"]
    if not token:
        print_error("Node API token is not defined")
    headers = {'X-Contract-Api-Token': token}
```

(continues on next page)

(continued from previous page)

```

url = '{0}/internal/contracts/{1}/sum'.format(node, contract_id)
r = requests.get(url, verify=False, timeout=2, headers=headers)
data = r.json()
return data['value']

if __name__ == '__main__':
    command = os.environ['COMMAND']
    if command == 'CALL':
        contract_id = json.loads(os.environ['TX'])['contractId']
        value = get_value(contract_id)
        print_success([{"key": "sum",
                        "type": "integer",
                        "value": value + 1}])
    elif command == 'CREATE':
        print_success([{"key": "sum",
                        "type": "integer",
                        "value": 0}])
    else:
        print_error("Unknown command {0}".format(command))

```

Description of operation

- The program expects to get the data structure in json format with the field “params”.
- It reads the values of the “a” fields.
- Returns the result as a value of field “{a} + 1” in json format.

Example of incoming parameters

```

"params": [
  {
    "key": "a",
    "type": "integer",
    "value": 1
  }
]

```

17.1.2 Installing a smart contract

1. Download and install [Docker for Developers](#) for your operating system.
2. Prepare a contract image. In the `stateful-increment-contract` folder, create the following files:
 - `contract.py`
 - `Dockerfile`
 - `run.sh`

Listing of `run.sh` file

```

#!/bin/sh

python contract.py

```


Dockerfile File Listing

```
FROM python:alpine3.8
ADD contract.py /
ADD run.sh /
RUN chmod +x run.sh
RUN apk add --no-cache --update iptables
CMD exec /bin/sh -c "trap : TERM INT; (while true; do sleep 1000; done) & wait"
```

Important: It is required to install `iptables` into the smart contract container.

3. Install the image in Docker registry. Execute the following commands in the terminal:

```
docker run -d -p 5000:5000 --name registry registry:2
cd contracts/stateful-increment-contract
docker build -t stateful-increment-contract .
docker image tag stateful-increment-contract localhost:5000/stateful-increment-contract
docker start registry
docker push localhost:5000/stateful-increment-contract
```

4. Run the following command in the terminal to get the information about the container:

```
docker inspect 57c2c2d2643d
[
{
  "Id": "sha256:57c2c2d2643da042ef8dd80010632ffdd11e3d2e3f85c20c31dce838073614dd",
  "RepoTags": [
    "wenode:latest"
  ],
  "RepoDigests": [],
  "Parent": "sha256:d91d2307057bf3bb5bd9d364f16cd3d7eda3b58edf2686e1944bcc7133f07913",
  "Comment": "",
  "Created": "2019-10-25T14:15:03.856072509Z",
  "Container": "",
  "ContainerConfig": {
    "Hostname": "",
    "Domainname": "",
    "User": "",
    "AttachStdin": false,
    "AttachStdout": false,
    "AttachStderr": false,
```

The smart contract identifier `Id` is the value of the `imageHash` field and it is used in transactions with the created smart contract.

5. Sign a transaction to create a smart contract. In this example, the transaction is signed with the key stored in the node keystore.

Hint: To create a key pair and the participant address, use the utility `generators.jar`. The procedure for creating a key pair is given in item 1 of the module “Connecting to the Network”. The rules for generating queries to the node are given in the module *Node REST API*.

Query Body

```
{
  "fee": 100000000,
  "image": "stateful-increment-contract:latest",
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
  "contractName": "stateful-increment-contract",
  "sender": "3PudkbvjV1nPj1TkuuRahh4sGdgfr4YAUUV2",
  "password": "",
  "params": [],
  "type": 103,
  "version": 1
}
```

Sample query

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --
↳header 'X-Contract-API-Token' -d '{ \
  "fee": 100000000, \
  "image": "stateful-increment-contract:latest", \
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65", \
  "contractName": "stateful-increment-contract", \
  "sender": "3PudkbvjV1nPj1TkuuRahh4sGdgfr4YAUUV2", \
  "password": "", \
  "params": [], \
  "type": 103, \
  "version": 1 \
}' 'http://localhost:6862/transactions/sign'
```

Sample response

```
{
  "type": 103,
  "id": "ULcq9R7PvUB2yPmrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
  "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
  "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhAv38Dws5skQDsJMVT2M",
  "fee": 500000,
  "timestamp": 1550591678479,
  "proofs": [
↳"yeCRFZm9iBLyDy93BDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxfj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
  "version": 1,
  "image": "stateful-increment-contract:latest",
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
  "contractName": "stateful-increment-contract",
  "params": [],
  "height": 1619
}
```

6. Send the signed transaction to the blockchain. The response from the sign method must be transferred to the input for the broadcast method.

Sample query

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --
↳header 'X-Contract-API-Token' -d '{ \
{
  "type": 103, \
  "id": "ULcq9R7PvUB2yPmrmBdxoTi3bcRmQPT3JDLLLZVj4Ky", \
  "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew", \
```

(continues on next page)

(continued from previous page)

```

"senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsJMVT2M", \
"fee": 500000, \
"timestamp": 1550591678479, \
"proofs": [
↪"yecrFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxfj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ], \
"version": 1, \
"image": "stateful-increment-contract:latest", \
"imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65", \
"contractName": "stateful-increment-contract", \
"params": [], \
"height": 1619 \
}
}' 'http://localhost:6862/transactions/broadcast'

```

7. Use the transaction ID to check that the contract initiation transaction is placed in the blockchain.

Sample response

```

{
  "type": 103,
  "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLLZVj4Ky",
  "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
  "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsJMVT2M",
  "fee": 500000,
  "timestamp": 1550591678479,
  "proofs": [
↪"yecrFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxfj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
  "version": 1,
  "image": "stateful-increment-contract:latest",
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
  "contractName": "stateful-increment-contract",
  "params": [],
  "height": 1619
}

```

17.1.3 Smart Contract Execution

1. Sign a call-transaction to call (execute) the smart contract.

In the "contractID" field, specify the contract initialization transaction ID.

Query Body

```

{
  "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
  "fee": 10,
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
  "password": "",
  "type": 104,
  "version": 1,
  "params": [
    {
      "type": "integer",
      "key": "a",
      "value": 1
    }
  ]
}

```

(continues on next page)

(continued from previous page)

```

]
}

```

Sample query

```

curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --
↳header 'X-Contract-API-Token' -d '{ \
  "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2", \
  "fee": 10, \
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58", \
  "password": "", \
  "type": 104, \
  "version": 1, \
  "params": [ \
    { \
      "type": "integer", \
      "key": "a", \
      "value": 1 \
    } \
  ] \
}' 'http://localhost:6862/transactions/sign'

```

Sample response

```

{
  "type": 104,
  "id": "9fBrL2n5TN473g1gNfoZqaAqAsAJCuHRHYxZpLexL3VP",
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
  "senderPublicKey": "2YvzcVLRqLCqouVrFZynjfoTEuPNV9GrdauNpgdWXLsq",
  "fee": 10,
  "timestamp": 1549365736923,
  "proofs": [
    "2q4cTBhdKEDkFxr7iYaHPAv1dzaKo5rDaTxPF5VHryyYTXxTPvN9Wb3YrsDYixKiUPXBnAyXzEcnKPFRCW9xVp4v"
  ],
  "version": 1,
  "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
  "params": [
    {
      "key": "a",
      "type": "integer",
      "value": 1
    }
  ]
}

```

2. Send the signed transaction to the blockchain. The response from the sign method must be transferred to the input for the broadcast method.

Sample query

```

curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --
↳header 'X-Contract-API-Token' -d '{ \
  "type": 104, \
  "id": "9fBrL2n5TN473g1gNfoZqaAqAsAJCuHRHYxZpLexL3VP", \
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58", \
  "senderPublicKey": "2YvzcVLRqLCqouVrFZynjfoTEuPNV9GrdauNpgdWXLsq", \
  "fee": 10, \

```

(continues on next page)

(continued from previous page)

```

"timestamp": 1549365736923, \
"proofs": [ \
  "2q4cTBhDkEDkFxr7iYaHPAv1dzaKo5rDaTxPF5VHryyYTXxTPvN9Wb3YrsDYixKiUPXBnAyXzEcnKPFRCW9xVp4v" \
], \
"version": 1, \
"contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2", \
"params": [ \
  { \
    "key": "a", \
    "type": "integer", \
    "value": 1 \
  } \
] \
}' 'http://localhost:6862/transactions/broadcast'
    
```

3. Get the result of smart contract execution by its ID.

Sample response

```

[
  {
    "key": "1+1",
    "type": "integer",
    "value": 2
  }
]
    
```

17.2 API methods available to smart contract

Docker container-based smart contracts can use node *REST API*. Smart contract developers can use limited list of REST API methods. This list is represented below, these methods are available directly from the container.

Addresses methods

- *GET /addresses*
- *GET /addresses/publicKey/{publicKey}*
- *GET /addresses/balance/{address}*
- *GET /addresses/data/{address}*
- *GET /addresses/data/{address}/{key}*

Crypto methods

- *POST /crypto/encryptCommon*
- *POST /crypto/encryptSeparate*
- *POST /crypto/decrypt*

Privacy methods

- *GET /privacy/{policy-id}/getData/{policy-item-hash}*
- *GET /privacy/{policy-id}/getInfo/{policy-item-hash}*
- *GET /privacy/{policy-id}/hashes*

- *GET /privacy/{policy-id}/recipients*

Transactions methods

- *GET /transactions/info/{id}*
- *GET /transactions/address/{address}/limit/{limit}*

Contracts methods

A smart contract can use *Contracts* methods implementing the separated `/internal/contracts/` route, which is totally identical to the regular *Contracts* methods.

- *GET /internal/contracts/{contractId}/{key}*
- *GET /internal/contracts/executed-tx-for/{id}*
- *GET /internal/contracts/{contractId}*
- *GET /internal/contracts*

PKI methods

- *PKI /verify*

17.2.1 Docker contract authorization

A smart contract requires an authorization to use the node *REST API*. There are following steps for the correct REST API methods usage by the smart contract:

1. The following variables should be defined in the Docker contract environment:
 - `NODE_API` - an URL address to the node *REST API*.
 - `API_TOKEN` - an authorization token of the Docker contract.
 - `COMMAND` - commands for the Docker contract creation and call.
 - `TX` - a transaction which is required to the Docker contract for work (*103 - 107* codes).
2. The Docker contract developer assigns the value of the variable `API_TOKEN` to the request header `X-Contract-Api-Token`. The node specifies JWT authorization token into the variable `API_TOKEN` for the contract creation and execution.
3. The contract code should pass the received token in the request header (`X-Contract-Api-Token`) each time the node API is accessed.

ROLE MANAGEMENT

The list of possible roles in the blockchain platform is given in module *“Authorization of participants”*.

Important: The prerequisite for changing permissions of participants (adding or deleting roles) is the availability of the participant’s private key with the “permissioner” role in the node keystore from which the query is made.

18.1 Option 1 (through REST API)

Participant permissions are managed by signing (sign method) and broadcasting (broadcast method) of permission transactions through *Node REST API*.

Query object for sign method:

```
{
  "type":102,
  "sender":3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG,
  "senderPublicKey":4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g,
  "fee":0,
  "proofs":[""],
  "target":3GPtj5osoYqHpyfmsFv7BMiyKsVzbG1ykfL,
  "opType":"add",
  "role":"contract_developer",
  "dueTimestamp":null
}
```

Query fields:

- type - the type of the transaction for the participant permission management (type = 102);
- sender - the participant address with the permission to issue permission transactions;
- proofs - the transaction signature;
- target - the participant address, for which permissions are required to be assigned or deleted;
- role - participant permissions to be assigned or removed. Possible values: “miner”, “issuer”, “dex”, “permissioner”, “blacklister”, “banned”, “contract_developer”, “connection_manager”;
- opType - the type of the operation “add” (add permissions) or “remove” (delete permissions);
- dueTimestamp - the permission validity date in the timestamp format. The field is optional.

Transfer the response from the node to the broadcast method.

18.2 Option 2 (using the utility)

Using the Generators utility the process can be automated.

Example of console launching:

```
java -jar generators.jar GrantRolesApp [configfile]
```

Example of configuration:

```
permission-granter {
waves-crypto = no
chain-id = T
account = {
  addresses = [
    "3N2cQFfUDzG2iujBrFTnD2TAsCNohDxYu8w"
  ]
  storage = ${user.home}"/node/keystore.dat"
  password = "some string as password"
}
send-to = [
  "devnet-aws-fr-2.we.wavesnodes.com:6864"
]
grants = [
  {
    address: "3N2cQFfUDzG2iujBrFTnD2TAsCNohDxYu8w"
    assigns = [
      {
        permission = "miner",
        operation = "add",
        due-timestamp = 1527698744623
      },
      {
        permission = "issuer",
        operation = "add",
        due-timestamp = 1527699744623
      },
      {
        permission = "blacklister",
        operation = "add"
      },
      {
        permission = "permissioner",
        operation = "remove"
      }
    ]
  }
]
txs-per-bucket = 10
}
```

The field “due-timestamp” limits the role validity; Fields “nodes”, “roles” are mandatory.

If the node is already assigned any of the roles specified in the config, then the case is handled in accordance with the rules:

Current node status	Status received from transaction	Processing result
No role assigned	New role	Success - role assigned
Role assigned without dueDate	Role with dueDate	Checking dueDate; if less than current, then IncorrectDateTime, otherwise Success - role assigned with dueDate
Role assigned with dueDate	Role with dueDate	Checking dueDate; if less than current, then IncorrectDateTime, otherwise Success - updating dueDate
Role assigned with dueDate	Role without dueDate	Success - role assigned without dueDate
Role assigned with/without dueDate	Role removal	Checking node address; if <> for genesis address, then Success - role removed

PARTICIPANTS CONNECTION TO THE NETWORK

The moment of the first node *running* is the beginning of the new blockchain net creation. You can create the blockchain net from the starting only one node, further you can add new nodes as required.

- *Connect* a new node into the existing network.
- *Delete* unnecessary nodes from the network.

19.1 Connection of a new node to the existing net

You can add new nodes into the net at any time. The configuration files setting is described in the section *Node configuration*. Perform all these actions and *run* the node. The following steps are making:

1. The new node user gives the public key and the node description to the net administrator.
2. The network administrator (the node with “Connection-manager” role) uses the received public key and description for the *111 RegisterNode* transaction creation with the "opType": "add" parameter.
3. Transaction falls to the block and further into the nodes states of network participants. As a result of the transaction among the stored data, each participant of the network stores the public key and the address of the new node.
4. If necessary, the network administrator can add additional roles to the new node using the transaction *102 Permit*.
5. The user *runs* the node.
6. After starting, the node sends *handshake-message* with its public key to the participants from the “peers” list of its configuration file.
7. Network participants compare the public key from the *handshake message* and the key from transaction *111 RegisterNode* sent earlier by the network administrator. If the check is successful, the network participant updates its database and sends the Peers Message message to the network.
8. Having successfully connected, the new node synchronizes with the network and receives the address table of the network participants.

19.2 Deleting the node

1. The network administrator creates the *111 RegisterNode* transaction with the parameter "opType": "remove" and the public key of the removed node within.
2. This transaction is fell into the block and approved by other nodes.
3. After accepting the transaction the nodes find the public key specified in the transaction *111 RegisterNode* in their state and delete it from there.
4. Then nodes delete the network address of the removed node from the `network.known-peers` of the node configuration file.

CONFIDENTIAL DATA EXCHANGE

Before you can share the confidential data, you need to create access groups. Using transactions, you can *add* or *change* access groups to the confidential data.

20.1 Creation of the confidential data access group

The confidential data access group can be created by any network participant. You need to specify the range of participants, which will get the data. Then any of participant will perform the following actions:

1. The network participant, the future owner of the group, is creating the *112 CreatePolicy* with the following parameters:
 - sender - the public key of the access group creator.
 - description - the description of the access group.
 - policyName - the name of the access group.
 - recipients - public keys of access group participants, which will have the access to the confidential data.
 - owners - public keys of access group participants, which, in addition to the data access, can change the lineup of the group participants.
2. This transaction is fell into the block and approved by other nodes.
3. After accepting the transaction the nodes which are the access group participants will get the access to the confidential data.

20.2 Changing the access group

Access groups can only be changed by their owners. The following actions are performed to change the list of participants in the access group:

1. The group owner creates the *113 UpdatePolicy* transaction with the following parameters:
 - policyId - identifier of the access group.
 - sender - the public key of the access group owner.
 - opType - the option of the adding (**add**) or the removing (**remove**) the group participants.
 - recipients - public keys of access group participants, which are added or removed from the access group.
 - owners - public keys of access group participants, which are added or removed from the access group.
2. This transaction is fell into the block and approved by other nodes.

3. After accepting the transaction the information about participants of the changed access group will update.

20.3 Exchanging the confidential data

Important: The size of the transferred data via API method *POST /privacy/sendData* to the network is up to 20 MB.

1. Using the API *POST /privacy/sendData* tool the client sends the data to the network (API parameters: sender, password, policy ID, data type, data information, data and hash).
2. Access group participants use the *GET /privacy/getData/{hash}* tool for getting information about data and its further download.

Follow these steps for the values creation of the **data** and **hash** fields:

1. Translate the data byte sequence into the **Base64** encoding.
2. Place the result of the data conversion to the "data": "29sCt...RgdC60LL" field of the API *POST /privacy/sendData*.
3. Specify the data hash sum according to the **SHA-256** algorithm in the "hash": "9wetTB...SU2zr1Uh" field. You need to specify the hash result in the **Base58** encoding.
4. Send the data to the network by pressing the **Try it out!** button.
5. Node automatically will create the *114 PolicyDataHash* transaction as a result of the data sending.

SYSTEM REQUIREMENTS

System and hardware requirements are given below.

Optional	CRASH	JVM Operation Mode
Minimum requirements	2 + 2G	50G java -Xmx2048M -jar
Recommended requirements	2 + 4 + 50G	10G java -Xmx4096M -jar

Hint: “Xmx” - flag defining maximum size of memory available for JVM.

Node environment requirements for the Waves cryptography usage

- JRE 1.8 (64-bit) or OpenJDK 12.0.1

Node environment requirements for the CryptoPro JCP cryptography usage

- Oracle JRE 1.8 (64-bit)
- CryptoPro JCP 2.0.40035

Warning: The Waves Enterprise platform only supports 2.0.40035 version for the CryptoPro JCP software. You need to register on the site before downloading the installation package. Also you need to register on the Oracle site to get the Oracle JRE 1.8 distributive.

Corporate client and data service environment requirements

- Docker CE
- Docker-compose
- node.js LTS version 10+
- npm 6+

Corporate client and data service environment requirements

- node.js LTS version 10+
- npm 6+
- PostgreSQL version 11

Authorization service environment requirements

- PostgreSQL version 11

Hint: There must be separate PostgreSQL databases for the data service and the authorization service.

SANDBOX

We offer to test the already configured of three nodes Waves Enterprise blockchain platform, which includes the authorization service and Docker contracts.

Attention: This version is not intended for commercial use and is provided for demonstration purposes only. The demo version can be run on Linux and MacOS operating systems.

You need the following software to use the sandbox version of the Waves Enterprise platform:

- Docker CE
- Docker-compose

Perform the following commands to run the sandbox:

1. Create a working directory and navigate there using the terminal.
2. Download the `docker-compose.yml` configuration file from our [GitHub](#) page and copy it into the working directory.
3. Log in as an administrator using the `sudo` command, and you will be asked to enter your password after it.
4. Run the following command and wait for the results:

```
docker run --name generator -v $(pwd)/nodes/:/opt/generator/nodes/ wavesenterprise/generator:demo
```

5. Run the sandbox with the following command:

```
docker-compose up -d
```

22.1 Sending transactions from the web client

Follow these steps after the blockchain platform full start:

1. Open a browser and enter the `http://localhost` address.
2. Register in the web client using any valid email address and log in to the web client.
3. Open the Choose address -> Add address manually page.
4. Fill in the fields below. You can take the values from the `accounts.conf` configuration file of the first node in the `nodes/node-1` directory.
 - *Node network address* - specify the `http://localhost/nodeAddress` value.

- *Address* - specify the node address. See the **Address** field marked in the picture below.
- *Key pair password* - specify the key pair password of the node. See the **Key-pair password** field marked in the picture below.

```
Please enter password for 1 account (empty password means no password):
2019-12-10 13:06:54,243 INFO [ecution-context-global-10]
c.w.g.AccountsGeneratorApp$ - 1 Address: 3FoVTCJGATgZE6VSnLXYw9t889yEFH4kP5f;
public key: GNmi9q8iJXC6P9kMNB1aTtvN4pLuKP8XGQjYbynYeRbZ
2019-12-10 13:06:54,245 INFO [ecution-context-global-10]
c.w.g.AccountsGeneratorApp$ - Generator future done
Key-pair password: Eiyae3aeBabiePhoo4iedahk
```

5. You can also simply create a new custom blockchain address using the **Choose address** -> **Add address** manually page and following the prompts of the web interface.

It is now possible to send transactions from the web client from the node address.

NODE INSTALLATION

Currently we support Unix-like systems (for example, popular Linux distributives and MacOS). However Waves Enterprise platform can be run under the Windows natively in experimental mode. Also you can you Unix virtual machines or the Docker environment for the installation and running the platform under the Windows.

The node allows to use the GOST cryptography based on the CryptoPro software as well as the embedded Waves cryptography module. If you want to use the GOST cryptography, please, contact Waves Enterprise support for more information.

Important: Waves Enterprise nodes installation must be performed on a separate machine from the Waves blockchain platform nodes.

The process of deploying and launching a node for Linux and MacOS systems is the same. The additional services set needs the apps [Docker CE](#) and [Docker-compose](#) for the fully running. Follow these steps:

1. Download and install the [Docker CE](#) and [Docker-compose](#) installation packages. Registration on the site is need for downloading.
 - 1.1. After installation check if applications [Docker CE](#) and [Docker-compose](#) (the part of the Docker CE installation package) have been successfully installed:

```
root@Eterna:/home/angela# docker-compose --version
docker-compose version 1.24.0, build 0aa59064
root@Eterna:/home/angela#
```



```
Last login: Wed May 29 17:48:43 on ttys000
➔ ~ docker --version
Docker version 18.09.2, build 6247962
➔ ~ docker-compose --version
docker-compose version 1.23.2, build 1110ad01
➔ ~
```

You can use the commands `docker --version` and `docker-compose --version` for macOS and Linux OS.

2. Download the [latest release](#) of the node and the config file template from the [GitHub](#). The following files are included into the release:
 - `generators-x.x.x.jar` utility which is used for the accounts creating and the genesis block signature;

- docker-compose configuration files for node deployment and all services.
3. Perform the node configuration according with the *Node configuration* section.
 4. Create the `node` directory. Copy created `node.conf` and `keystore.dat` files in this `node` folder.
 5. The node is run using docker-compose. Run one of the following commands:

```
docker-compose up -d
docker-compose -f docker-compose-name.yml up -d
```

The first command is used if you deploy a node using the `docker-compose.yml` default configuration file. In any other case use the second command.

Attention: When a node is run, the available resources are checked. If the available resources are less than the minimum, the node will not be able to start. See *System requirements* for checking of minimum and recommended hardware specifications.

6. Stop the node using the command:

```
docker-compose down -d
docker-compose -f docker-compose-name.yml down -d
```

The first command is used if you stop a node using the `docker-compose.yml` default configuration file. In any other case use the second command.

NODE CONFIGURATION

The node configuration includes the following steps:

24.1 Preparation of configuration files

These following configuration files are used for the configuration:

- `accounts.conf` – the configuration file for the accounts creation.
- `api-key-hash.conf` – the configuration file for the `api-key-hash` and `privacy-api-key-hash` values creation when you choose the `api-key` string hash authorization.
- `node.conf` – the main node configuration file defining the operational principals and an option list.

24.1.1 `accounts.conf` configuration file for the accounts creation

When specifying a path, use the “forward slash” - / as a delimiting character for directory hierarchy levels. During Linux using the value `wallet` must match the directory structure of the operating system, for example `/home/contract/we/keystore.dat`. During node setting it is prohibited to use cyrillic symbols for specifying paths to the working directory, `keystore`, etc.

```
// accounts.conf listing

accounts-generator {
  waves-crypto = yes
  chain-id = V
  amount = 1
  wallet = "${user.home}"/node/keystore.dat"
  wallet-password = "some string as password"
  reload-node-wallet {
    enabled = false
    url = "http://localhost:6862/utils/reload-wallet"
  }
}
```

The description of the configuration file parameters is represented below.

- `waves-crypto` – the choice of a cryptographic algorithm (“yes” - use cryptography *Waves*, “no” - use *GOST-cryptography*);
- `chain-id` – an identifying byte of the network, the value will be necessary further on for entry in parameter `address-scheme-character` of the node configuration file;
- `amount` – a number of generated key pairs;

- `wallet` – the path to the key storage directory on the node, the value will be required further on for entry in parameter `wallet > file` of the node configuration file. For the Waves cryptography, the path to file `keystore.dat` is specified (example, `${user.home}/nodeName/keystore.dat`), for the GOST-cryptography - the path to directory (`${user.home}/nodeName/keystore/`);
- `wallet-password` – a password for access to closed node keys, the value will be necessary further for entry into the parameter `wallet > password` of the node configuration file;
- `reload-node-wallet` – an option to update the node keyStore without restarting the application, by default it is turned off (`false`). `url` parameter specifies the path to the `/utils/reload-wallet` method of the REST API node.

24.1.2 api-key-hash.conf configuration file

`api-key-hash.conf` configuration file is intended only for the `api-key-hash` and `privacy-api-key-hash` values creation when you choose the `api-key` string authorization.

```
// api-key-hash.conf listing

apikeyhash-generator {
  waves-crypto = no
  api-key = "some string for api-key"
}
```

Parameters description

- `waves-crypto` – the choice of a cryptographic algorithm (“yes” - use cryptography *Waves*, “no” - use *GOST-cryptography*);
- `api-key` – the key you need to come up with. The value of this key will need to be specified in requests to REST API node (for more details see page *REST API*).

24.1.3 node.conf node configuration file

If you are planning to connect the new node to the existing network, it will be more easy to request full configuration file from your network administrator or from any of net participants. When you are creating the configuration file from a scratch or connecting to the “Waves Enterprise Mainnet”, you can get the example of the file from our [GitHub](#) page. You can read on the *Changes in the node configuration file* page about changes in the node configuration file.

Warning: If your node’s version is 1.0 and higher you need to specify the following parameter in the `node` section of the node configuration file:

```
"features": {
  "supported": [100]
}
```

This option becomes active when the total quantity of blocks from `feature-check-blocks-period = 15000` and `blocks-for-feature-activation = 10000` parameters is achieved (25 000 of blocks). These parameters are stored in the `blockchain` section and can not be changed during Mainnet or Partnetnet connection. Nodes will not be able to connect to the network without activation of this option.

The example of the node configuration file is represented below. This file does not include such options like *anchoring*, *Docker* smart contracts and private data access *groups*. Also there are `api-key` authorization and Waves cryptography. You can find the fields description *here*.

Note: If you want to use additional options, set the `enable` field of the selected option to `yes` or `true` and configure the option section according to the description of its setting.

Warning: Please, fill **ONLY** the fields with the `/FILL/` word inside as a value.

```

node {
  # Type of cryptography
  waves-crypto = yes

  # Node owner address
  owner-address = " /FILL/ "

  # NTP settings
  ntp {
    server = "pool.ntp.org"

    # Maximum time without synchronization. Required for PoA consensus.
    fatal-timeout = 5 minutes
  }

  # Node "home" and data directories to store the state
  directory = "/node"
  data-directory = "/node/data"

  wallet {
    # Path to keystore.
    file = "/node/keystore.dat"

    # Access password
    password = " /FILL/ "
  }

  # Blockchain settings
  blockchain {
    type = CUSTOM
    fees.enabled = false
    consensus {
      type = "poa"
      round-duration = "17s"
      sync-duration = "3s"
      ban-duration-blocks = 100
      warnings-for-ban = 3
      max-bans-percentage = 40
    }
    custom {
      address-scheme-character = "E"
      functionality {
        feature-check-blocks-period = 1500
        blocks-for-feature-activation = 1000
        pre-activated-features = { 2 = 0, 3 = 0, 4 = 0, 5 = 0, 6 = 0, 7 = 0, 9 = 0, 10 = 0, 100 = 0 }
      }
    }

    # Mainnet genesis settings

```

(continues on next page)

(continued from previous page)

```
genesis {
  average-block-delay: 60s
  initial-base-target: 153722867

  # Filled by GenesisBlockGenerator
  block-timestamp: 1573472578702

  initial-balance: 1625000000000000

  # Filled by GenesisBlockGenerator
  genesis-public-key-base-58: ""

  # Filled by GenesisBlockGenerator
  signature: ""

  transactions = [
    # Initial token distribution:
    # - recipient: target's blockchain address (base58 string)
    # - amount: amount of tokens, multiplied by 10e8 (integer)
    #
    # Example: { recipient: "3HQSr3VFCiE6JcWwV1yX8attYbAGKTLV3Gz", amount:
↪3000000000000000 }
    #
    # Note:
    # Sum of amounts must be equal to initial-balance above.
    #
    { recipient: " /FILL/ ", amount: 1000000000000000 },
    { recipient: " /FILL/ ", amount: 1500000000000000 },
    { recipient: " /FILL/ ", amount: 5000000000000000 },
  ]

  network-participants = [
    # Initial participants and role distribution
    # - public-key: participant's base58 encoded public key;
    # - roles: list of roles to be granted;
    #
    # Example: {public-key: "EPxkVA9iQejsjQikovyakkY8iHnbXsR3wjkgE7ZW1It", roles:
↪[permissioner, miner, connection_manager, contract_developer, issuer]}
    #
    # Note:
    # There has to be at least one miner, one permissioner and one connection_manager for
↪the network to start correctly.
    # Participants are granted access to the network via GenesisRegisterNodeTransaction.
    # Role list could be empty, then given public-key will only be granted access to the
↪network.
    #
    { public-key: " /FILL/ ", roles: [permissioner, miner, connection_manager, contract_
↪developer, issuer]},
    { public-key: " /FILL/ ", roles: [miner]},
    { public-key: " /FILL/ ", roles: []},
  ]
}
}

# Application logging level. Could be DEBUG | INFO | WARN | ERROR. Default value is INFO.
logging-level = DEBUG
```

(continues on next page)

(continued from previous page)

```

features.supported = [100]

# P2P Network settings
network {
    # Network address
    bind-address = "0.0.0.0"
    # Port number
    port = 6864

    # Peers network addresses and ports
    # Example: known-peers = ["node-1.com:6864", "node-2.com:6864"]
    known-peers = [ /FILL/ ]

    # Node name to send during handshake. Comment this string out to set random node name.
    # Example: node-name = "your-we-node-name"
    node-name = " /FILL/ "

    # How long the information about peer stays in database after the last communication with it
    peers-data-residence-time = 2h

    # String with IP address and port to send as external address during handshake. Could be set
    ↪ automatically if uPnP is enabled.
    # Example: declared-address = "your-node-address.com:6864"
    declared-address = "0.0.0.0:6864"
}

# New blocks generator settings
miner {
    enable = yes
    # Important: use quorum = 0 only for testing purposes, while running a single-node network;
    # In other cases always set quorum > 0
    quorum = 0
    interval-after-last-block-then-generation-is-allowed = 10d
    micro-block-interval = 5s
    min-micro-block-age = 3s
    max-transactions-in-micro-block = 500
    minimal-block-generation-offset = 200ms
}

# Nodes REST API settings
rest-api {
    # Enable/disable REST API
    enable = yes

    # Network address to bind to
    bind-address = "0.0.0.0"

    # Port to listen to REST API requests
    port = 6862

    auth {
        type: "api-key"

        # Hash of API key string
        # You can obtain hashes by running ApiKeyHash generator
    }
}
    
```

(continues on next page)

(continued from previous page)

```

api-key-hash: " /FILL/ "

# Hash of API key string for PrivacyApi routes
privacy-api-key-hash: " /FILL/ "
}
}

#Settings for Privacy Data Exchange
privacy {
  storage {
    enabled = false
    # url = "jdbc:postgresql://postgres:5432/node-1?user=postgres&password=wenterprise"
    # driver = "org.postgresql.Driver"
    # profile = "slick.jdbc.PostgresProfile$"

    # user = "postgres@postgres&password=wenterprise"
    # password = "wenterprise"

    # connectionPool = HikariCP
    # connectionTimeout = 5000
    # connectionTestQuery = "SELECT 1"
    # queueSize = 10000
    # numThreads = 20
    # schema = "public"
    # migration-dir = "db/migration"
  }
}

# Docker smart contracts settings
docker-engine {
  # Docker smart contracts enabled flag
  enable = no

  # Basic auth credentials for docker host
  #docker-auth {
  #  username = "some user"
  #  password = "some password"
  #}

  # Optional connection string to docker host
  docker-host = "unix:///var/run/docker.sock"

  # Optional string to node REST API if we use remote docker host
  # node-rest-api = "node-0"

  # Execution settings
  execution-limits {
    # Contract execution timeout
    timeout = 10s
    # Memory limit in Megabytes
    memory = 512
    # Memory swap value in Megabytes (see https://docs.docker.com/config/containers/resource_
    ↪constraints/)
    memory-swap = 0
  }
}

```

(continues on next page)

(continued from previous page)

```

# Reuse once created container on subsequent executions
reuse-containers = yes

# Remove container with contract after specified duration passed
remove-container-after = 10m

# Allows net access for all contracts
allow-net-access = yes

# Remote registries auth information
remote-registries = []

# Check registry auth on node startup
check-registry-auth-on-startup = yes

# Contract execution messages cache settings
contract-execution-messages-cache {
    # Time to expire for messages in cache
    expire-after = 60m
    # Max number of messages in buffer. When the limit is reached, the node processes all messages
    ↪ in batch
    max-buffer-size = 10
    # Max time for buffer. When time is out, the node processes all messages in batch
    max-buffer-time = 100ms
}
}
}
    
```

24.2 Changes in the node configuration file

This section provides information to help you identify changes in the configuration file depending on the node version.

Warning: If you are updating a node version, you must also update the node configuration file. The node will not run without updating the configuration file!

24.2.1 Differences between the node configuration file version 1.0.0 and version 1.1.0

ntp section 1.0.0 version

```
ntp-server = "0.ru.pool.ntp.org"
```

ntp section 1.1.0 version

```

ntp {
servers = [ "0.pool.ntp.org", "1.pool.ntp.org", "2.pool.ntp.org", "3.pool.ntp.org" ]
# Socket timeout for synchronization request.
request-timeout = 10 seconds
# Time between synchronization requests.
expiration-timeout = 1 minute
    
```

(continues on next page)

(continued from previous page)

```
# Maximum time without synchronization. Required for PoA consensus.
fatal-timeout = 1 minute
}
```

docker-engine section 1.0.0 version

```
docker-engine {
enable = yes
docker-host = "http://proxy-to-dockerhost"
```

docker-engine section 1.1.0 version

```
docker-engine {
enable = yes
docker-host = "tcp://proxy-to-dockerhost"
```

anchoring section 1.0.0 version

```
anchoring {
...

targetnet-node-address = "http://node.weservices"
targetnet-node-port = 6862

...
}
```

anchoring section 1.1.0 version

```
anchoring {
...

targetnet-node-address = "http://node.weservices.com:6862/NodeAddress"

...
}
```

24.3 Description of the node configuration file parameters and sections

Several types of values are used for parameters in the configuration file.

- Integer data which used to specify the exact number of elements. It can be the number of transactions, blocks or connections.
- Integer data including measuring units to specify the time periods or memory volume. You typically specify the time periods in days, hours, or seconds, or the cache memory volume, for example, `leveldb-cache-size = 256M` or `connection-timeout = 30s`.
- String which used to specify the addresses, directory paths, passwords and so on. The directory path is specifying in the acceptable format of your current OS and the value is quoted.

- Array for the list of values like addresses or public keys. The value is specified in square brackets separated by commas.
- Boolean `no` or `yes` which used for option activation.

An example of the node configuration file is represented on the *configuration files prepare* page. It includes the following sections:

- *node* - general section, which includes all sections of blockchain settings.
- *ntp* - NTP server parameters settings.
- *blockchain* - common blockchain settings.
- *features* - network settings.
- *network* - network settings.
- *wallet* - settings of the private keys access.
- *miner* - mining settings.
- *rest-api* - REST API settings.
- *privacy* - confidential information access groups settings.
- *docker-engine* - Docker smart contracts settings.

24.3.1 node section

Additional section parameters:

- *waves-crypto* - *cryptography* type in the blockchain. Possible values: `yes` - Waves cryptography, `no` - GOST cryptography.
- *directory* - the main directory for the storage of the node software.
- *data-directory* - the main directory for the storage of the node software.
- *logging-level* - logging level. Possible values: `DEBUG`, `INFO`, `WARN`, `ERROR`, default value is `INFO`.
- *owner-address* - the node address, the future owner of the configuration file.

24.3.2 ntp section

- *server* - an NTP server addresses list. The recommended value is `pool.ntp.org`.
- *request-timeout* - the timeout of the one request to an NTP server. The recommended value is 10 seconds.
- *expiration-timeout* - the timeout of the NTP server requests synchronization. The recommended value is 1 minute.
- *fatal-timeout* - the timeout of the connection to an NTP server. The recommended value is 1 minute.

24.3.3 blockchain section

- **type** - the blockchain type. Possible values are `MAINNET` or `CUSTOM`. The `MAINNET` value allows you to use the genesis block, consensus and Mainnet settings. When you select `MAINNET` in the configuration file of the node which connects to the Mainnet network, you do not need to specify the parameters of `custom`, `genesis` and `consensus` blocks.
- **consensus.type** - *consensus* type. Possible values are `pos` or `poa`. You can read more [here](#) about consensus settings.

fees unit

- **enabled** - the option of using fees for the *transaction* release. Possible values are `false` or `true`.

custom unit

- **address-scheme-character** - the address feature character which is used to prevent mixing up addresses from different networks. For the “Waves Enterprise Mainnet” - `V` and for the “Waves Enterprise Partnernet” - `P`. You can use any letter you like for the sidechain or test versions of the Waves Enterprise blockchain platform. Nodes must have the same network byte on the same blockchain network.
- **functionality** - main blockchain settings.
- **genesis** - genesis block settings.

functionality unit

- **feature-check-blocks-period** - the blocks period for feature checking and activation.
- **blocks-for-feature-activation** - the number of blocks required to accept feature.
- **pre-activated-features** - a set of blockchain options.

genesis unit

- **average-block-delay** - an average delay between the blocks creation. This parameter is used only for the *PoS* consensus.
- **initial-base-target** - an initial base number for the managing the mining process. This parameter is used for the *PoS* consensus . The frequency of the block creation depends on the parameter value therefore the higher the value, the more often blocks are created. Also, the value of the miner’s balance affects the use of this parameter in mining - the larger the miner’s balance, the less the value of **initial-base-target** is used. When setting a value for this parameter, it is recommended to take into account the combination of miners balances and the expected interval between blocks.
- **block-timestamp** - a time and data code. The time is specified in milliseconds and the value must consist of 13 digits. If you specify the standard value **timestamp** consisting of 10 digits, then you need to add any three digits at the end.
- **initial-balance** - an initial balance in smallest units. The parameter value affects on the mining process with the *PoS* consensus. The larger the miner’s balance, the smaller the **initial-base-target** value is used for the mining node determination for the current round.
- **genesis-public-key-base-58** - the public key hash of the genesis block, encrypted in Base58.
- **signature** - the genesis block signature, encrypted in Base58.
- **transactions** - a list of network participants with an initial balance, the creation of which will be included in the genesis block.
- **network-participants** - a list of network participants with specified roles, the creation of which will be included in the genesis block.

24.3.4 network section

- `bind-address` - the node network address.
- `port` - the port number.
- `known-peers` - a list of IP addresses of well known nodes.
- `declared-address` - a string with IP address and port to send as external address during the handshake.

24.3.5 wallet section

- `file` - a path to the private keys storage.
- `password` - a password for the private keys file access.

24.3.6 miner section

- `enable` - a miner option activation.
- `quorum` - required number of connections (both incoming and outgoing) to attempt block generation. Setting this value to 0 enables offline generation. When you are specifying the value, it is necessary to consider that the own mining node is not summed with the parameter value, i.e., if it is `quorum = 2`, then you need at least 3 mining nodes in the network.
- `interval-after-last-block-then-generation-is-allowed` - enable block generation only if the last block is not older the given period of time.
- `micro-block-interval` - an interval between microblocks.
- `min-micro-block-age` - a minimal age of the microblock.
- `max-transactions-in-micro-block` - a maximum number of transaction in the microblock.
- `minimal-block-generation-offset` - a minimal time interval between blocks.

24.3.7 features section

- `supported` - a list of supported options.

24.4 Accounts creation

The user account includes an address and a key pair which consists of public and private keys. The address and public key are shown to the user during account creation on the command line. The private key is written to the `keystore.dat`.

24.4.1 Key pairs generating

Public and private keys for initial participants are creating by the generator. You can get the last version of the generator on our [GitHub](#) page. Before running the utility you need to specify the `accounts.conf` configuration file which contains parameters for keys creating. During the creation think up and enter a password, then save it for later configuration. The given password will be used at creation of a global variable `WE_NODE_OWNER_PASSWORD` further. Press `enter` key if you do not want to use this password. Use the following command to run the generator:

```
java -jar generators-x.x.x.jar AccountsGeneratorApp accounts.conf
```

24.4.2 Global variables

We recommend to use a password for the keys pair to increase security. The Waves Enterprise platform supports two ways of the password usage:

1. Enter the password manually at the each start of the node.
2. Create global variables in your OS.

If you are using the manual enter the password there is no need to create global variables. But when you are planning to use containers or any similar services to run the node then create the following global variables in the OS for your convenience:

1. `WE_NODE_OWNER_PASSWORD` - the keys pair password specified during the key pair creation.
2. `WE_NODE_OWNER_PASSWORD_EMPTY` - `true` or `false`, specify the `true` value if you do not want to use the keys pair password, in this case it is not necessary to create the `WE_NODE_OWNER_PASSWORD` variable. When you are using the password than specify the `false` value and write into the `WE_NODE_OWNER_PASSWORD` variable the keys pair password.

24.5 Signing the genesis block

Sign the genesis block using utility `generators-x.x.x.jar`. Command for signing: `java -jar generators-x.x.x.jar GenesisBlockGenerator node.conf`, where `Name.conf` is the edited in *this section* node configuration file. After signing `genesis-public-key-base-58` and `signature` fields of the configuration file will be filled with values of the public key and the proof of the genesis block.

Example:

```
genesis-public-key-base-58: "4ozcAj...penxrm"  
signature: "5QNVGF...7Bj4Pc"
```

24.6 Consensus settings

Waves Enterprise blockchain platform supports two types of consensus - *PoS* and *PoA*. The consensus settings are located in the *blockchain* section.

24.6.1 PoS configuration

The PoS consensus will be used by default if you have not specified the consensus type in the `consensus.type` field of the `blockchain` section. Here are the mining responsible parameters which are located in the `genesis` unit of the `blockchain` section:

- `average-block-delay` - an average delay between the blocks creation. The default value is 60 seconds. The value of this parameter is ignored if PoA consensus is selected.
- `initial-base-target` - an initial base number for the managing the mining process. The frequency of the block creation depends on the parameter value therefore the higher the value, the more often blocks are created. Also, the value of the miner's balance affects the use of this parameter in mining - the larger the miner's balance, the less the value of `initial-base-target` is used.
- `initial-balance` - an initial balance in smallest units. The greater the share of the miner's balance from the network initial balance, the smaller becomes the value of `initial-base-target` to determine the node miner of the current round.

We recommend to use the default parameter values specified in the configuration files examples which are represented on the [GitHub](#) page.

24.6.2 PoA settings

Please, uncomment or add the `consensus` unit of the `blockchain` section for the *PoA* consensus usage:

```
consensus {
  type = "poa"
  round-duration = "17s"
  sync-duration = "3s"
  ban-duration-blocks = 100
  warnings-for-ban = 3
  max-bans-percentage = 40
}
```

Represented in the `consensus` unit parameters are used only for the *PoA* consensus.

- `type` - the consensus type. Possible values are `pos` or `poa`. If you will specify the `pos` value, than other parameters will not be considered.
- `round-duration` - a round length of the block mining in seconds.
- `sync-duration` - a block mining synchronization period in seconds. The total time of the round is the sum of `round-duration` and `sync-duration`.
- `ban-duration-blocks` - a blocks quantity of the ban period for the mining node.
- `warnings-for-ban` - a number of rounds which is used for ban warnings for miner nodes.
- `max-bans-percentage` - a percentage of mining nodes from the total number of nodes in the network that can be placed in the ban.

Using the *PoA* consensus allows to adjust the order of blocks creation by limiting the mining function for certain nodes. The reason is to distribute evenly the network load, if any mining nodes left the network or became inactive. Mining node can get banned for the following reasons:

- if a node will miss its queue for mining;
- if a node provides an invalid block;
- if a node went offline.

Before getting into the `blacklist` the mining node receives warnings about the ban possibility during the number of rounds that is specified in the `warnings-for-ban` parameter. The mining node will be back to the mining after the `ban-duration-blocks` parameter value will end.

24.6.3 Consensus settings in the miner section

When you are configuring consensus settings, please, consider the following settings of the `miner` section:

- `micro-block-interval` - an interval between microblocks. The value is specified in seconds.
- `min-micro-block-age` - a minimal age of the microblock. The value is specified in seconds and should not be more than the `micro-block-interval` parameter value.
- `minimal-block-generation-offset` - a minimal time interval between blocks. The value is specified in milliseconds.

The values of the microblock creation parameters should not conflict with the parameters values of the `average-block-delay` for PoS and `round-duration` for PoA. The number of microblocks in a block is not limited, but depends on the transactions size in the microblock.

24.7 Docker configuration

Installation and execution of docker smart contracts configures in the `docker-engine` of the `node configuration file`.

```
# Docker smart contracts settings
docker-engine {
# Docker smart contracts enabled flag
enable = no
# Basic auth credentials for docker host
docker-auth {
  username = "some user"
  password = "some password"
}
# Optional connection string to docker host
# docker-host = "unix:///var/run/docker.sock"
# Optional string to node REST API if we use remote docker host
# node-rest-api = "https://clinton.wavesenterprise.com/node-0"
# Run for integration tests
integration-tests-mode-enable = no
# Execution settings
execution-limits {
  # Contract execution timeout
  timeout = 60s
  # Memory limit in Megabytes
  memory = 512
  # Memory swap value in Megabytes (see https://docs.docker.com/config/containers/resource_
↪constraints/)
  memory-swap = 0
}
# Reuse once created container on subsequent executions
reuse-containers = yes
# Remove container with contract after specified duration passed
remove-container-after = 10m
# Allows net access for all contracts
```

(continues on next page)

(continued from previous page)

```

allow-net-access = no
# Remote registries auth information
remote-registries = []
# Check registry auth on node startup
check-registry-auth-on-startup = yes
# Authorization timeout for the contract
contract-auth-expires-in = 1m
# Contract execution messages cache settings
contract-execution-messages-cache {
    # Time to expire for messages in cache
    expire-after = 60m
    # Max number of messages in buffer. When the limit is reached, the node processes all messages
    ↪ in batch
    max-buffer-size = 10
    # Max time for buffer. When time is out, the node processes all messages in batch
    max-buffer-time = 100ms
}
}
    
```

Parameters:

- **enable** - the Docker smart contracts option activation (yes/no).
- **docker-auth** - the authorization parameters with login/password section.
- **docker-host** - a Docker host URL address.
- **node-rest-api** - the REST API address if you are using the remote Docker host.
- **integration-tests-mode-enable** - the integration tests run option (yes/no).
- **execution-limits** - the Docker contracts run limits section:
 - **timeout** - a timeout for the smart contract execution;
 - **memory** - a memory limit for a smart contract in megabytes;
 - **memory-swap** - a memory swap value in megabytes.
- **reuse-containers** - reuse option for the existing Docker contract.
- **remove-container-after** - container remove option after contract execution (yes/no).
- **allow-net-access** - the option which allows network access for all smart contracts (yes/no).
- **remote-registries** - a list of remote registry repositories for the Docker contracts run.
- **check-registry-auth-on-startup** - the option which checks the registry repositories authorization during the node start (yes/no).
- **contract-auth-expires-in** - a timeout for the Docker contract authorization token.
- **contract-execution-messages-cache** - the contract execution messages cache settings section. When the limit is reached, the node processes all messages in batch:
 - **expire-after** - a time period to expire for messages in cache;
 - **max-buffer-size** - a maximum number of messages in buffer;
 - **max-buffer-time** - a maximum time period in milliseconds of messages in buffer.

24.8 Authorization type configuration for the REST API access

The Waves Enterprise blockchain platform supports the following two types of authorization for the node's REST API access:

- `api-key` string hash authorization;
- authorization via the authorization service.

The authorization type is specified in the REST API configuration section of the node configuration file. `api-key` string hash authorization type is a simple method of the access management to a node with a low level security. If the `api-key` hash is leaking out to the attacker, he is getting the full access to the node. When you utilize the separate authorization service with access tokens, you increase the security level of your blockchain network to the high level. You can read more information about the authorization service in the *Authorization service* section.

24.8.1 `rest-api` section of the node configuration file

The `rest-api` section allows to bound the node network address to the REST API interface, to choose and configure the authorization type, also to specify the limits for some REST API methods.

```
# Node's REST API settings
rest-api {
# Enable/disable REST API
enable = yes

# Network address to bind to
bind-address = "127.0.0.1"

# Port to listen to REST API requests
port = 6862

# Authorization strategy should be either 'oauth2' or 'api-key', default is 'api-key'
auth {
    type = "api-key"

    # Hash of API key string
    api-key-hash = "H6nsiifwYKYEx6YzYD7woP1XCn72RVvx6tC1zjjLXqsu"

    # Hash of API key string for PrivacyApi routes
    privacy-api-key-hash = "H6nsiifwYKYEx6YzYD7woP1XCn72RVvx6tC1zjjLXqsu"
}
# For OAuth2:
# auth {
#     type: "oauth2"

#     # OAuth2 service public key to verify auth tokens
#     public-key: "AuthorizationServicePublicKeyInBase64"

#     # OAuth2 settings for initial sync
#     service-url: "auth.service.url"
#     service-token: "auth-token"
# }

# Enable/disable CORS support
cors = yes
```

(continues on next page)

(continued from previous page)

```
# Enable/disable X-API-Key from different host
api-key-different-host = no

# Max number of transactions
# returned by /transactions/address/{address}/limit/{limit}
transactions-by-address-limit = 10000
distribution-address-limit = 1000
}
```

Parameters description

- `enable` - REST API option activation.
- `bind-address` - a network address to bind the REST API interface.
- `port` - a port to listen to REST API requests.
- `cors` - enable/disable CORS support.
- `transactions-by-address-limit` - a maximum number of transactions returned by `/transactions/address/{address}/limit/{limit}` method.
- `distribution-address-limit` - GET `/assets/{assetId}/distribution/{height}/limit/{limit}`.

auth unit

- `auth-type` - the authorization type. `oauth2` - the token authorization, `api-key` - the string hash authorization.
- `api-key-hash` - a hash of API key string.
- `privacy-api-key-hash` - a hash of API key string for `privacy` methods.
- `oauth-public-key` - a public key of the authorization service.
- `oauth-private-key` - a private key of the authorization service.
- `service-token` - a node service token for getting the access token for the authorization between participants of the blockchain network.

24.8.2 When you use the key string hash for the authorization

Specify the `api-key` value for the `auth-type` parameter. Create the `api-key-hash` for the REST API access by using the `generators-x.x.x.jar` utility. To run the utility, you need to specify the `api-key-hash.conf` file as one of the parameters, which defines the parameters of creating the `api-key-hash`. Use the following command to run the generator:

```
java -jar generators-x.x.x.jar ApiKeyHash api-key-hash.conf
```

Specify the value obtained as a result of the utility execution in the parameter `api-key-hash` in the node configuration file.

Create the `privacy-api-key-hash` by the same way as the `api-key-hash` to get the `privacy` methods access. Specify the value obtained as a result of the utility execution in the parameter `privacy-api-key-hash` in the node configuration file.

24.8.3 When you use the token authorization

Specify the `oauth2` value for the `auth-type` parameter, write the public key of the authorization service into the `oauth-public-key` parameter.

24.9 Anchoring settings

If you are using the *anchoring* option, please, configure the `anchoring` unit. `targetnet` is the blockchain network which will be used by the sidechain node to send anchoring transactions.

```

anchoring {
  enable = yes
  height-range = 5
  height-above = 6
  threshold = 1

  targetnet-authorization {
    type = "oauth2" # "api-key" or "oauth2"
    authorization-token = "PawC6b86r2pNRTR5e88wvcL3gfkG87w2Lqkvk4Jph2PUG3zPLedCTjnjh2ZTw3Rf
    ↪"
    authorization-service-url = "https://washington.testnet.com/authServiceAddress/v1/auth/
    ↪token"
    token-update-interval = "60s"
    # api-key-hash = "5M7C14rf3TAaWHvU6Kqo97iscd8fJFpuFwyQ3Q6vfztS"
    # privacy-api-key-hash = "5M7C14rf3TAaWHvU6Kqo97iscd8fJFpuFwyQ3Q6vfztS"
  }

  targetnet-scheme-byte = "K"
  targetnet-node-address = "http://node.weservices.com:6862/NodeAddress"
  targetnet-node-recipient-address = "3JWveBpXS1EcDpxcoAwVNAjFfUMrxaALgZt"
  targetnet-private-key-password = ""

  wallet {
    file = "node-1_mainnet-wallet.dat"
    password = "small"
  }

  targetnet-fee = 500000
  sidechain-fee = 500000
}
    
```

Anchoring parameters

- `height-range` - the number of blocks which is used as an interval between anchoring transactions to the Targetnet.
- `height-above` - the number of blocks in the Targetnet after which the private blockchain node creates the confirming data-transaction containing data from the first data-transaction. We recommend specifying this value close to the Targetnet maximum rollback depth `max-rollback`.
- `threshold` - the number of blocks subtracted from the current height of the private blockchain. The anchoring transaction sent to the Targetnet includes the data from the block at height `current-height - threshold`. When the value is 0, the current block is anchored. We recommend specifying this value close to the private blockchain maximum rollback depth `max-rollback`.

Anchoring authorization parameters

- **type** - authorization type for anchoring. **api-key** - **api-key-hash** authorization , **auth-service** - authorization by a special security token.

For authorization by **api-key-hash** necessary a current key-value as **api-key**. For authorization by a special security token you must use a **type = "auth-service"** and comment config-file structure values:

- **authorization-token** - a constant authorization token.
- **authorization-service-url** - URL address authorization service.
- **token-update-interval** - data interval for a token refresh.

Targetnet access parameters

A separate **keystore.dat** file with a key pair for the Targetnet access is generated for the node that will send the anchoring transaction to the Targetnet.

- **targetnet-scheme-byte** - the Targetnet network byte.
- **targetnet-node-address** - the full node network address including the port number in the Targetnet for the sending of anchoring transactions. The address should be specified along with the connection type (**http/https**), the port number and the **NodeAddress** parameter as in the example **http://node.weservices.com:6862/NodeAddress**.
- **targetnet-node-recipient-address** - the node address in the Targetnet for the recording of anchoring transactions signed with a key pair of this address.
- **targetnet-private-key-password** - the node private key password for the anchoring transactions signing.

The network address and the port for the Targetnet/Partnet networks anchoring can be obtained from Waves Enterprise technical support staff. If multiple private blockchains with mutual anchoring are used, you should use the appropriate private network settings.

Parameters of key pair file for the Targetnet anchoring transactions signing, wallet unit

- **file** - a file name and a path to the key pair file for the Targetnet anchoring transactions signing. The file is located on the private network node.
- **password** - a password of the key pair file.

Fee parameters

- **targetnet-fee** - the fee for the anchoring transaction issue in the Targetnet.
- **sidechain-fee** - the fee for the anchoring transaction issue in the private blockchain.

24.10 Privacy data access groups configuration

When using the *privacy* methods activate the option and fill in the **storage** block with database settings for storing the private data:

```

privacy {
  storage {
    enabled = true
    url = "jdbc:postgresql://" + "${POSTGRES_ADDRESS}" + ":" + "${POSTGRES_PORT}" + "/" + "${POSTGRES_DB}"
    driver = "org.postgresql.Driver"
    profile = "slick.jdbc.PostgresProfile$"
    user = "${POSTGRES_USER}"
    password = "${POSTGRES_PASSWORD}"
    connectionPool = HikariCP
  }
}

```

(continues on next page)

(continued from previous page)

```

connectionTimeout = 5000
connectionTestQuery = "SELECT 1"
queueSize = 10000
numThreads = 20
schema = "public"
migration-dir = "db/migration"
}
}

```

Parameters description

- `enabled` - the option activation.
- `url` - the PostgreSQL DB address.
- `driver` - the JDBC driver name.
- `profile` - a profile name for the JDBC access.
- `user` - a user name for the DB access.
- `password` - a password for the DB access.
- `connectionPool` - a connection pool name. Default is HikariCP.
- `connectionTimeout` - a connection timeout.
- `connectionTestQuery` - a query name for the connection test.
- `queueSize` - a requests queue size.
- `numThreads` - a number of parallel connections.
- `schema` - an interaction scheme.
- `migration-dir` - a path to the data migration directory.

DB PostgreSQL is using as a database for the confidential data storage. The database should be installed on the same machine with the node and should have an DB access account. You can use the PostgreSQL tutorial for download and install the database according with your operation system type.

During the installation the system will offer to create an access account. These credentials must be entered into the appropriate `user/password` parameters.

Specify the URL for the PostgreSQL connection into the `url` parameter. URL consists of:

- `POSTGRES_ADDRESS` - a PostgreSQL host address.
- `POSTGRES_PORT` - a PostgreSQL host port number.
- `POSTGRES_DB` - a PostgreSQL name.

You can specify the PostgreSQL credentials with the URL in the same string. The example is represented bellow, where `user=user_privacy_node_0@we-dev` is a login, `password=7nZL7Jr41q0WUHz5qKdypA&sslmode=require` - a password with require option during the authorization.

Example

```

privacy.storage.url = "jdbc:postgresql://vostk-dev.postgres.database.azure.com:5432/
↳privacy_node_0?user=user_privacy_node_0@we-dev&password=7nZL7Jr41q0WUHz5qKdypA&
↳sslmode=require"

```


You can download the latest distributives and configuration files examples from the [GitHub Waves Enterprise release page](#).

ADDITIONAL SERVICES DEPLOY

The additional services set needs the apps [Docker CE](#) and [Docker-compose](#) for the fully running. You can check the full list of environment requirements for the Waves Enterprise platform in the *System requirements* page.

The additional services set is offered as a Docker container with the following list of services:

- The corporate web client.
- The data service.
- The data crawler.
- The authorization service.
- PostgreSQL DB.
- Nginx-proxy.

Follow these steps to deploy the additional services set:

1. Download and unzip the file `frontend-deployment.zip`.
2. Specify the path to REST API and node hostname in the configuration file of the application `frontend-deployment/config/nginx-proxy.env`:

```
// nginx-proxy.env listing  
  
WE_NODE_ADDRESS=http://yournet.wavesenterprise.com:6862  
WE_NODE_HOST=http://node-1:6862
```

3. Specify the path to REST API and authorization service hostname in the configuration file of the application `frontend-deployment/config/crawler.env`:

```
// crawler.env listing  
  
VOSTOK_AUTH_SERVICE_ADDRESS=http://auth-service:3000  
VOSTOK_NODE_ADDRESS=http://yournet.wavesenterprise.com:6862
```

4. The following PostgreSQL DB access parameters are used in the configuration files of `frontend-deployment/config/postgres.env` and `frontend-deployment/config/auth-service.env` applications:

```
// postgres.env listing  
  
POSTGRES_HOST=crawler-db  
POSTGRES_DB=blockchain  
POSTGRES_USER=postgres
```

(continues on next page)

(continued from previous page)

```
POSTGRES_PASSWORD=wenterprise
// auth-service.env listing
POSTGRES_HOST=db
POSTGRES_DB=auth_db
POSTGRES_USER=postgres
POSTGRES_PASSWORD=wenterprise
POSTGRES_PORT=5432
```

5. You must generate an RSA key pair for the `RSA_PUBLIC_FILE_PATH` and `RSA_PRIVATE_FILE_PATH` parameters. Run these commands sequentially:

```
ssh-keygen -t rsa -b 4096 -m PEM -f jwtRS256.key
openssl rsa -in jwtRS256.key -pubout -outform PEM -out jwtRS256.key.pub
cat jwtRS256.key
cat jwtRS256.key.pub
```

6. Specify the following parameters in the configuration file of the application `frontend-deployment/config/auth-service.env`:

- The REST API path and the node URL:

```
API_URL=http://yournet.wavesenterprise.com:3000
SITE_URL=http://yournet.wavesenterprise.com:8080
```

- The data string, which is passed to the hash function along with the password.

```
PASSWORD_HASH_SALT='X7ZAhlIVpqaJpXVVAZusGBC0cWagZ1DY'
```

- Parameters of the mail account which will be used by the authorization service for letters.

```
IS_MAIL_TRANSPORT_ENABLED=true
MAIL_HOST=mail.example.com
MAIL_USER=noreply@example.com
MAIL_PASSWORD=3hSsgt3!8wr5
MAIL_PORT=587
IS_MAIL_SECURE=false
```

Important: After registration each user must confirm his account by the following the link from the letter with account activation.

7. Specify the path to the authorization service REST API and URL-address in the configuration file of the application `frontend-deployment/config/we-data-service.env`.

```
VOSTOK_AUTH_SERVICE_ADDRESS=http://yournet.wavesenterprise.com:3000
```

8. Run `docker-compose` with the command `docker-compose -f docker-compose-frontend.yml up -d`. Before starting the frontend service and all additional services a running node must be deployed.
9. Open the browser and go to `localhost:8080` to check if the system client is successfully deployed.

GLOSSARY

Account

A client data set which is stored in database and used for client identification

Alias

A user's login associated with his address as a result of the transaction, the result of which is used to record the alias address matching in the database, and it is possible to specify this alias in the subsequent transactions

Anonymous network

Unpermissioned public blockchain which can be accessed by any participant as an anonymous person

Blockchain

A decentralized, distributed and public digital ledger that is used to record in such way that any involved record cannot be altered retroactively, without the alteration of all subsequent blocks

Genesis block

The first block in the blockchain which contains special genesis transactions distributing the initial balance and permissions

Access group

A table inside the node state containing the net participants list which can exchange the privacy data according to this policy

Cryptocurrency

A form of digital currency based on encryption algorithms and ran inside decentralized platforms built on the blockchain

Consensus

The way to agree on a single point of the data value in a network between participants

Mining

The process by which transactions are verified and added to a blockchain

Mainnet

A real network where transactions are executing, tokens are issuing and storing

Node

A computer which is ran the node software and connected to the blockchain network

Peer

A net address of the node

Private key

A privately held string of data that allows you to sign transactions and to get access to tokens. The private key is inextricably bound to the public key

Public network

Permissioned public blockchain where each participant is known and registered in the network

Public key

A string of data bound with the private key and used for interactions with net participants. The public key is applied to transactions to confirm the correctness of the user's signature made on the private key

Public address

A public address is the cryptographic hash of a public key and a net byte. They act as email addresses that can be published anywhere, unlike private keys

Swagger

API tool

Seed phrase

A set from 24 accidentally chosen words for restoring the access to the tokens

Smart account

An account with specified features for creating and running smart-contracts

Smart asset

A token with an attached script, during each new transaction with such a token the transaction will be confirmed first by the script, then by the blockchain

Smart contract

A computer program code that is capable of facilitating, executing, and enforcing the negotiation or performance of an agreement between participant

State

The full history of transactions which is stored in the node DB

Token

An account unit, a blockchain asset, which is not a cryptocurrency and is intended to represent the digital balance, it is an equivalent of the company's shares

Transaction

An operation that participants on the blockchain network use to interact with each other

Participant

A blockchain participant who send transactions to the net for getting approve

Hash

A unique configuration of the symbols (letters and digits), it is a result of the hash function performing over the data according with the specified algorithm. Hash uniquely identifies the object

Private network

Permissioned private blockchain where all transactions are controlled by a central authority

Gateway

The app for tokens transfer from one blockchain net to another one

Airdrop

A distribution of cryptocurrency to users, entirely for free

PoS (Proof-of-Stake)

A consensus algorithm based on the stake which is used for choosing the node for checking transactions and generating a new block

PoA (Proof-of-Authority)

A consensus algorithm in a private blockchain that grants to the most authority nodes the right to check transactions and generate a new block

WHAT IS NEW IN THE WAVES ENTERPRISE

27.1 1.1.0

The following pages have been added:

- *API methods available to smart contract*
- *Sandbox*
- *Changes in the node configuration file*

The following sections have been rebuilt:

- *Docker Smart Contracts*
- *Example of starting a contract*
- *Node installation*
- *Additional services deploy*

27.2 1.0.0

The following pages have been added:

- *Authorization service*

The following sections have been rebuilt:

- *Node configuration*
- *Mainnet and Partnernet connection*
- *REST API*
- *Node installation*

Changes in the node configuration file node.conf

- The *NTP server* section is added
- The `auth` section is added into the authorization type selection of the *REST API* section