Техническое описание платформы Waves Enterprise

Выпуск master

https://wavesenterprise.com

апр. 09, 2020

ГЛАВА 1

Обзор возможностей

Блокчейн-платформа Waves Enterprise — это универсальное решение для масштабируемой цифровой инфраструктуры, сочетающее в себе свойства публичных и приватных блокчейнов для корпоративного и государственного использования. Корпоративная блокчейн-платформа решает проблему доверия между участниками отношений на уровне протокола работы платформы. Поддерживаемые типы консенсусов *PoS* и *PoA* гарантируют корректность добавляемых в блокчейн данных, а децентрализация обеспечивает независимый от контрагентов доступ к данным.

1.1 Блокчейн Waves Enterprise

- Построен на языке программирования Scala.
- Реализует лучшие технологии и практики использования, унаследованные от публичной блокчейн-платформы Waves.
- Адаптирован для использования в корпоративном и государственном секторах.
- Поддерживает два алгоритма консенсуса *PoS* и *PoA*: при развёртывании сети можно выбрать наиболее подходящий под вашу задачу.
- Обладает высокой пропускной способностью.
- Поддерживает два вида *смарт-контрактов*, Тьюринг-неполные на RIDE и Тьюринг-полные на Docker.
- Поставляется в виде набора микросервисов.
- Использует алгоритмы сертифицированной государственной криптографии.
- Позволяет отправлять конфиденциальные данные через блокчейн адресно при помощи групп доступа, без публикации данных в открытом доступе.
- Реализует на уровне консенсуса систему управления полномочиями.

- Веб-клиент Waves Enterprise реализует следующие функции: просмотр *транзакций*, кошелек, отправка всех типов транзакций, разработка смарт-контрактов, мониторинг состояния блокчейна, управление полномочиями участников сети.

1.1.1 Варианты развертывания сети Waves Enterprise

1. Работа в основной публичной сети.
2. Работа в частной сети с анкорингом в основную сеть.
3. Работа в независимой частной сети.

1.2 Основная сеть

Основная сеть поддерживается консорциумом крупных компаний из различных областей: банки, промышленность, девелопмент, логистика и т.д. Компании, поддерживающие основную сеть, реализуют свои проекты в публичном блокчейне, а также обеспечивают сопутствующие процессы, например, банки предоставляют фиатные *шлюзы*, регистраторы — доступ к облачной ГОСТ-криптографии и т.п.

1.3 Независимая частная сеть

Коробочное решение Waves Enterprise позволяет развернуть собственную частную сеть для тех компаний, которые не готовы публично вести свои корпоративные процессы. Сеть конфигурируется под потребности каждой отдельной компании.

Основные конфигурируемые свойства решения:

- Тип консенсуса.
- Криптография.
- Число нод.
- Параметры работы блокчейна.

1.4 Частная сеть с публикацией хешей в основную сеть

Данное решение объединяет преимущества двух предыдущих вариантов и может быть актуально для небольших компаний и/или партнёров компаний, поддерживающих основную сеть. Использование частной сети позволяет избежать публичной демонстрации транзакций. При этом регулярная публикация хешей приватных блоков в основной сети позволяет повысить надежность информации внутри частной сети за счёт эффекта масштаба основной сети.

ГЛАВА 2

Официальные ресурсы

- Официальный сайт блокчейн-платформы Waves Enterprise
- Github проекта
- Официальный сайт блокчейн-платформы Waves

ГЛАВА 3

Архитектура

Платформа Waves Enterprise построена на базе технологии распределенного реестра и представляет собой фрактальную сеть, состоящую из:

- мастер-блокчейна (Waves Enterprise Mainnet), который обеспечивает работу сети в целом, выступая в качестве глобального арбитра и опорной цепи, и ряда пользовательских;
- отдельных сайдчейнов, легко настраиваемых в соответствии с конкретной бизнес-задачей.

Применение такого принципа построения позволяет добиться оптимизации для более высоких скоростей или больших объемов вычислений, согласованности и доступности данных, а также устойчивости к злонамеренному изменению информации.

Также в Waves Enterprise реализован *механизм анкоринга сетей*, позволяющий создать конфигурацию сети, использующую сильные стороны обоих алгоритмов консенсуса. Например, основной блокчейн Waves Enterprise базируется на алгоритме консенсуса Proof-of-Stake, так как поддерживается независимыми участниками. В то же время корпоративные сайдчейны, в которых нет необходимости стимуляции майнеров за счёт комиссий за транзакции, могут использовать алгоритм Proof-of-Authority. Сайдчейны встраиваются в основной блокчейн с помощью механизма анкоринга (помещая криптографические доказательства транзакций в основную блокчейн-сеть).

3.1 Архитектура ноды и дополнительных сервисов

ПО ноды может быть установлено без дополнительных сервисов, поскольку оно обеспечивает функционирование и взаимодействие внутри блокчейн-сети. Другие компоненты облегчают и значительно упрощают взаимодействие пользователя с блокчейн-платформой. Платформа Waves Enterprise состоит из пяти основных модулей и нескольких дополнительных микросервисов. В состав основных модулей входят:

- Нода - основное приложение, устанавливаемое на компьютер и настраиваемое для работы в блокчейне по любому сценарию.

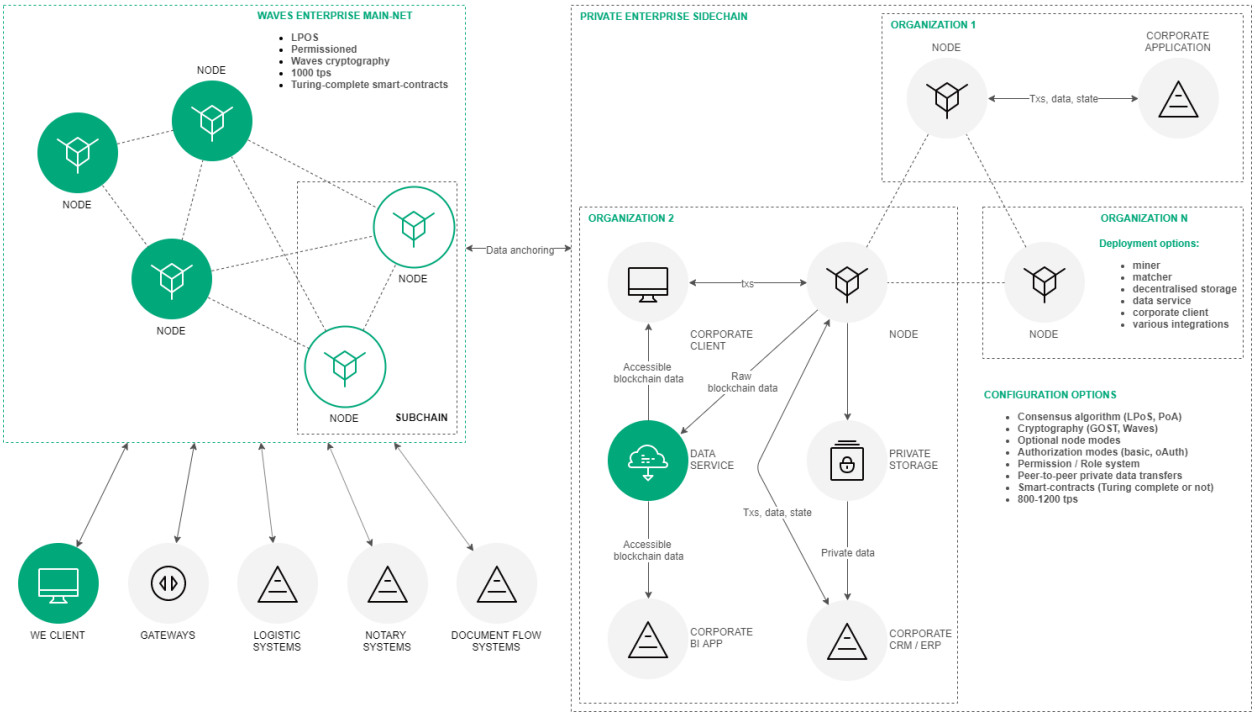


Рис. 1: Топология сети, включающая Waves Enterprise Mainnet и сайдчейны

- Корпоративный клиент - *веб-приложение*, предоставляющее современный и многофункциональный интерфейс для взаимодействия с блокчейном.
- Среда разработки смарт-контрактов - среда для развёртывания и выполнения Тьюринг-полных *Docker смарт-контрактов*. Разворачивание Docker-контейнеров со смарт-контрактами происходит на удалённой виртуальной машине для обеспечения дополнительной безопасности.
- Дата-сервис - *сервис* агрегирует данные из блокчейна в хранилище RDBMS (PostrgreSQL) и обеспечивает полнотекстовый поиск по любой информации, содержащейся в блокчейне, через веб-сервис RESTfull.
- Приватное хранилище - БД PostgreSQL обеспечивает обработку и хранение приватных данных, а также коммуникации через зашифрованное соединение peer-to-peer.

Дополнительные микросервисы включают в себя:

- Сервис авторизации - сервис обеспечения авторизации для всех компонентов.
- Дата-краулер - сервис извлечения данных с ноды и загрузки извлечённых данных в Дата-сервис.
- Генератор - сервис генерации ключевых пар для новых аккаунтов и создания `api-key-hash`.
- Плагины кастомизации данных - набор плагинов для обработки и кастомизации данных, передаваемых и принимаемых от внешних систем.
- Сервис мониторинга - внешний сервис мониторинга, использующий базу данных (InfluxDB) для хранения временных рядов с данными и метриками приложения. БД InfluxDB является ПО с открытым исходным кодом и устанавливается клиентом отдельно.

Компоненты ноды

Нода имеет следующие внутренние компоненты:

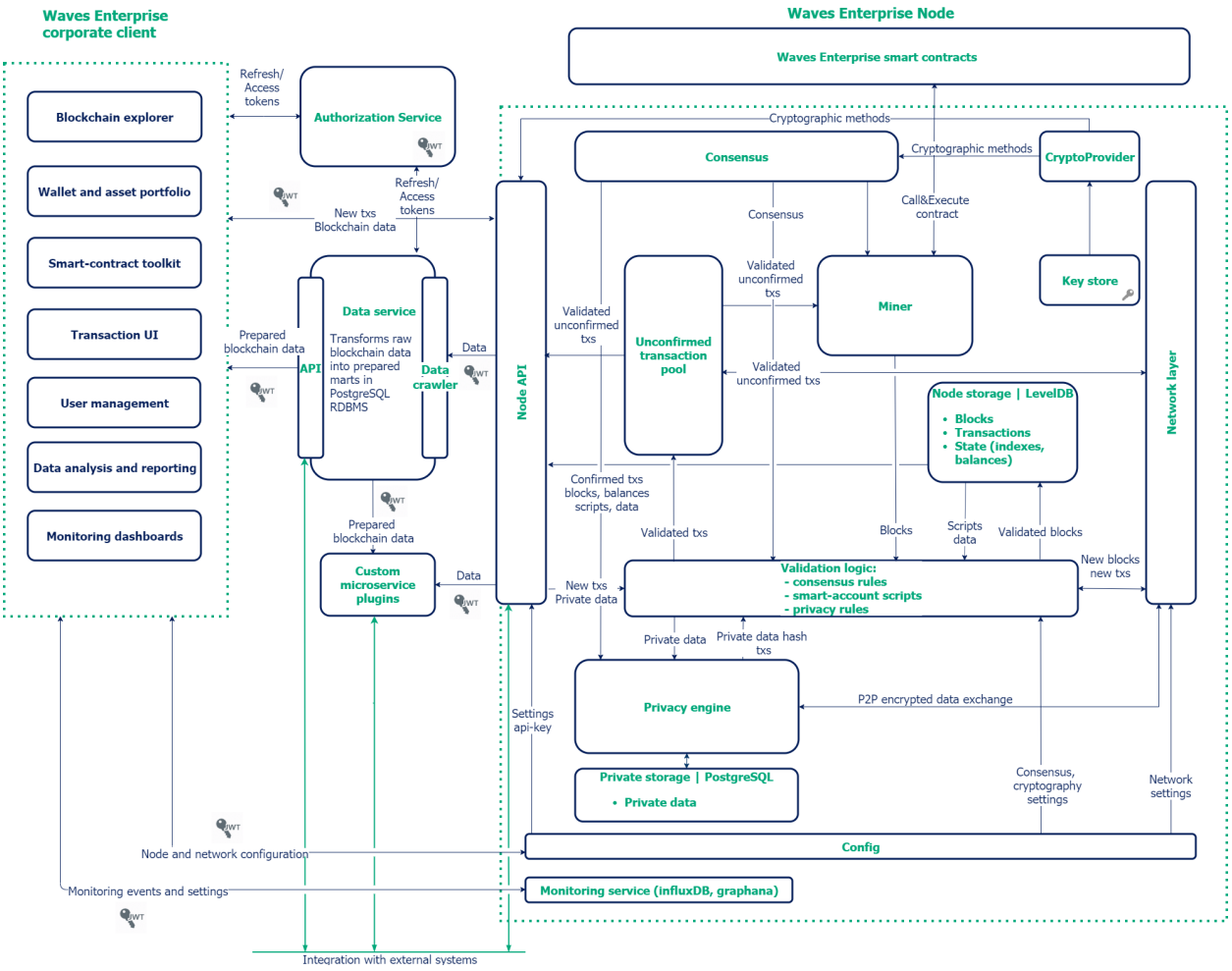


Рис. 2: Подробная схема архитектуры ноды и дополнительных микросервисов

- Node API – интерфейс REST API ноды, позволяющий получать данные из блокчейна, подписывать и отправлять транзакции, отправлять конфиденциальные данные, создавать и выполнять смарт-контракты и др.
- Node storage – компонент системы на базе LevelDB, обеспечивающий хранение пар ключ-значение для полного набора проверенных и подтверждённых транзакций и блоков, а также текущего состояния блокчейна.
- Unconfirmed transaction pool – компонент, обеспечивающий хранение неподтвержденных транзакций до момента их проверки и отправки в блокчейн.
- Consensus and cryptolibraries – компоненты, отвечающие за механизм достижения согласия между узлами, а также за криптографические алгоритмы.
- Key store - хранилище ключевых пар ноды и пользователей, все ключи защищены паролем.
- Miner – компонент, отвечающий за формирование блоков транзакций для записи в блокчейн, а также за взаимодействие с Docker смарт-контрактами.
- Network layer – слой логики, обеспечивающий взаимодействие нод на прикладном уровне по сетевому протоколу поверх TCP.
- Validation logic – слой логики, содержащий такие правила проверки транзакций, как базовая проверка подписи и расширенная проверка по сценарию.
- Config – конфигурационные параметры ноды, задаваемые в файле `node-name.conf`.
- Monitoring Service – внешний сервис мониторинга, использующий базу данных (InfluxDB) для хранения временных рядов с данными и метриками приложения. БД InfluxDB является ПО с открытым исходным кодом и устанавливается клиентом отдельно.

ГЛАВА 4

Протокол Waves-NG

Описание протокола работы блокчейна Waves Enterprise, обеспечивающего преимущество производительности относительно других блокчейнов.

4.1 Термины

- Блок — зафиксированный в блокчейне набор транзакций, подписанный майнером и содержащий ссылку на подпись предыдущего блока. Ограничен 1 Мб или 6000 транзакций.
- Раунд — период времени между выпуском ключевых блоков. Плавающее значение, регулируется алгоритмом консесуса в зависимости от нагрузки на сеть, в среднем 40 секунд.
- Доказательство доли владения — получение права майнинга в консенсусе PoS.
- Нода — узел сети с запущенным приложением блокчейна Waves Enterprise.
- Майнер — нода, адрес которой обладает достаточным для майнинга балансом и разрешением «Майнинг».
- Ключевой блок — не содержит транзакций, только служебную информацию, такую как:
  - публичный ключ майнера — для проверки подписи микроблоков;
  - сумму комиссии майнера за предыдущий блок;
  - подпись майнера;
  - ссылку на предыдущий ключевой блок.
- Liquid Block — служебный термин, для описания состояния блока до выпуска следующего ключевого блока, т.е. завершения его майнинга.
- Микроблоки — служебный термин, наборы транзакций, применяемых к состоянию блокчейна раз в 5 секунд. Ограничен 500 транзакциями. Каждый микроблок подписан приватным ключом майнера.

4.2 Описание протокола

Waves-NG — разработан Waves Platform на основе Bitcoin-NG для повышения пропускной способности блокчейна Waves, на архитектуре которого реализован Waves Enterprise. Идея протокола — в каждом раунде майнинга создавать не 1 большой блок, а непрерывно создавать микроблоки. Маленькие блоки быстрее пересылать и проверять.

Раунды майнинга начинаются с выпуска ключевого блока. Момент появления каждого ключевого блока и адрес указанного в нём майнера определяются консенсусом, подробнее *Консенсус*. Ключевой блок, не содержащий транзакций, только доказательство — формируется быстро. Далее, до появления следующего блока, раз в 5 секунд формируются микроблоки с транзакциями без доказательства доли, что также повышает скорость обработки. Каждый микроблок ссылается на предыдущий. Ключевой блок добавляется в блокчейн, как только следующий майнер выпустит свой ключевой блок со ссылкой на него.

Такой подход снижает время подтверждения транзакции по сравнению с другими блокчейнами.

4.2.1 1. Процесс создания Liquid Block

1. Консенсусом определяется майнящий адрес.
2. Майнер создает и рассылает по сети ключевой блок.
3. Каждые 5 секунд майнер создает и рассылает по сети микроблок, который содержит транзакции. Он должен ссылаться на предыдущий микро- или ключевой блок.
4. Процесс продолжается до тех пор, пока в сети не появится новый валидный ключевой блок.

4.2.2 2. Механизм вознаграждения майнеров в Waves-NG

В протоколе предусмотрена финансовая мотивация соблюдения правил для участников. Комиссия от транзакций в блоке распределяется в пропорции 40 % майнеру создавшему блок и 60 % майнеру следующего блока. Транзакция по начислению комиссии происходит через 100 блоков для обеспечения доверительного интервала проверок.

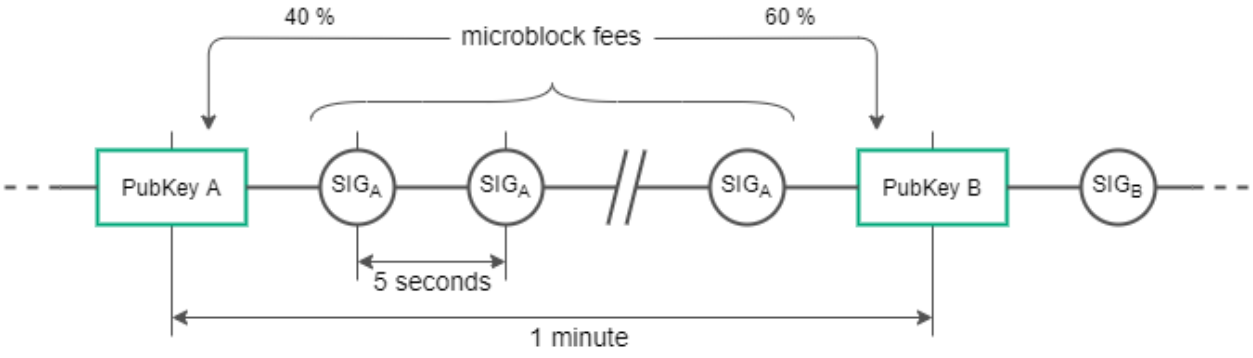


Рис. 1: Схема разделения комиссий

4.2.3 3. Разрешение конфликтов

Если майнер продолжает цепочку, создавая два микроблока с одним и тем же родителем, он наказывается отменой дохода от комиссий; тот, кто обнаруживает мошенничество, получает награду майнера за блок. Блокчейн — распределенная система и каждая нода хранит копию состояния всей сети. При появлении очередного микроблока, нода применяет полученные изменения к своей копии состояния сети и сверяет с остальными узлами сети. В этот момент происходит обнаружение несогласованности транзакций.

ГЛАВА 5

Алгоритмы консенсуса

Блокчейн — это децентрализованная система, в которой нет центрального органа власти. Это делает систему некоррумпированной, но создает сложности с итоговым принятием решений и организацией работы. Эти задачи решает механизм консенсуса, который является способом достижения согласия в группе участников. Голосование происходит в пользу большинства, не учитывая интересы меньшинства, но с другой стороны, это гарантирует достижение соглашения, которое несет пользу всей сети.

В Waves Enterprise вы можете выбрать механизм консенсуса при первичной конфигурации сети. Описание доступных механизмов, а также их плюсы и минусы, разобраны в соответствующих разделах.

5.1 Алгоритм консенсуса LPoS

Доказательство доли владения с правом аренды. В PoS (Proof of Stake) системах создание блока не требует энергозатратных вычислений, задача майнера — создание цифровой подписи блока.

5.1.1 Proof of Stake

Механизм распределения прав создания блоков основан на количестве токенов на счету пользователя. Чем больше у пользователя токенов, тем выше вероятность, что он сможет создать блок.

В консенсусе Proof of Stake (доказательство доли) право выпуска блока определяется псевдослучайным образом, поскольку зная предыдущего майнера и балансы всех пользователей в системе можно вычислить следующего майнера. Это возможно, благодаря детерминированному вычислению генерирующей подписи блока, которая получается путем SHA256 хеширования генерирующей подписи текущего блока и публичного ключа аккаунта. Первые 8 байт полученного хеша преобразуются в число, называемое hit аккаунта - X_n , и являются указателем на следующего майнера. Время генерации блока для аккаунта i, рассчитывается как:

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

где:

- b_i - это стейк (доля баланса участника от общего баланса системы);
- A_n - baseTarget, адаптивный коэффициент, регулирующий среднее время выпуска блока;
- X_n - hit аккаунта;
- T_{min} - 5 секунд, константа, определяющая минимальный временной интервал между блоками;
- C_1 - константа, равная 70 и корректирующая форму распределения интервала между блоками;
- C_2 - константа, равная 5E17 и предназначенная для регулировки значения baseTarget (сложности).

Из приведенной формулы легко убедиться, что вероятность выбора участника зависит от доли активов участника в системе: больше доля — выше шанс. Минимальное количество токенов на балансе для майнинга — **50 000 WEST**. BaseTarget — сложность вычислений, параметр, удерживающий время генерации блоков в заданном диапазоне. BaseTarget в свою очередь вычисляется как:

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

где

- $R_{max} = 90$ - максимальное уменьшение сложности, когда время генерации блока в сети превышает 40 секунд;
- $R_{min} = 30$ - минимальное увеличение сложности, когда время генерации блока в сети меньше 40 секунд;
- S - среднее время генерации, как минимум для трех последних блоков;
- T_p - предыдущее значение baseTarget;
- T_b - вычисленное значение baseTarget.

Глубокое описание технических особенностей и доработок классического PoS алгоритма вы можете найти в этой статье.

Преимущества перед PoW

Отсутствие сложных вычислений позволяет PoS сетям снизить требования к аппаратному обеспечению участников системы, что снижает стоимость разворачивания приватных сетей. Также не нужна дополнительная эмиссия, которая в PoW (Proof of Work) системах используется для вознаграждения майнеров за нахождение нового блока. В PoS-системах майнер получает вознаграждение в виде комиссий за транзакции, которые попали в его блок.

5.1.2 Leased Proof of Stake

Для пользователя, который обладает стейком, недостаточным для эффективного майнинга, есть возможность передать свой баланс в аренду другим участникам, и получать долю дохода от майнинга. Так вы можете увеличить вероятность выбора майнера, за что вы можете получать часть от комиссий за транзакций, которые этот майнер поместил в свои блоки. Лизинг является полностью безопасной операцией. Токены не покидают ваш кошелек, вы передаете право учитывать свой баланс при розыгрыше права майнинга другому участнику сети.

5.2 Алгоритм консенсуса PoA

В приватном блокчейне не всегда нужны токены - например, блокчейн может быть использован для хранения хешей документов, которыми обмениваются организации. В таком случае, при отсутствии токенов и комиссий с транзакций, решение на базе алгоритма консенсуса PoS является избыточным. В Waves Enterprise можно выбрать альтернативный алгоритм консенсуса — PoA (Proof of Authority). Разрешение на майнинг в алгоритме PoA выдаётся централизовано. Это упрощает принятие решений по сравнению с алгоритмом PoS. Модель Proof of Authority основана на ограниченном количестве валидаторов блока, что делает её масштабируемой системой. Блоки и транзакции проверяются заранее утвержденными участниками, которые выступают в качестве модераторов системы.

5.2.1 Описание алгоритма

На базе приведенных ниже параметров формируется алгоритм определения майнера текущего блока. Параметры консенсуса указываются в блоке `consensus` конфигурационного файла ноды.

- t - длительность раунда в секундах (параметр конфигурационного файла ноды: round-duration).
- t_s - длительность периода синхронизации, вычисляется как t*0,1, но не более 30 секунд (параметр конфигурационного файла ноды: sync-duration).
- N_{ban} - количество пропущенных подряд раундов для выдачи бана майнеру (параметр конфигурационного файла ноды: warnings-for-ban);
- P_{ban} - доля максимального количества забаненных майнеров, в процентах от 0 до 100 (параметр конфигурационного файла ноды: max-bans-percentage);
- t_{ban} - продолжительность бана майнера в блоках (параметр конфигурационного файла ноды: ban-duration-blocks).
- T_0 - unix time создания genesis блока.
- T_H - unix time создания блока H — ключевой блок для NG.
- r - номер раунда, вычисляется как ($T_{Current}$ -T_0) div (t+ t_s).
- A_r - лидер раунда r, имеющий право на создание ключевых блоков и микроблоков для NG в раунде r.
- H – высота цепочки, на которой создается ключевой блок и микроблоки для NG. Право на выпуск блока на высоте H имеет лидер раунда A_r .
- M_H - майнер, выпустивший блок на высоте H.
- Q_H - очередь активных на высоте H майнеров.

Очередь Q_H формируется из адресов, которым permission транзакцией выдано разрешение на майнинг, у которых оно не было отозвано до высоты H, и не истекло до момента времени T_H .

Очередь сортируется по временной метке транзакции предоставления прав на майнинг – узел, которому права были предоставлены раньше, будет выше в очереди. Для согласованной сети эта очередь будет одинакова на каждой ноде.

Новый блок создается в течение каждого раунда r. Раунд длится t секунд. После каждого раунда отводится t_s секунд на синхронизацию данных в сети. В период синхронизации микроблоки и ключевые блоки не формируются. Для каждого раунда существует единственный лидер A_r , который имеет право создать блок в этом раунде. Определение лидера может производиться на каждом узле сети с одинаковым результатом. Определение лидера раунда осуществляется следующим образом:

1. Определяется майнер M_{H-1} , который создал предыдущий ключевой блок на высоте H-1.

2. Вычисляется очередь Q_H активных майнеров.
3. Из очереди исключаются неактивные майнеры (подробнее в пункте *Исключение неактивных майнеров*).
4. Если майнер блока H-1 (M_{H-1}) есть в очереди Q_H , лидером A_r становится следующий по очереди майнер.
5. Если майнера блока H-1 (M_{H-1}), нет в очереди Q_H , лидером A_r становится майнер, идущий в очереди за майнером блока H-2(M_{H-2}), и так далее.
6. Если ни одного из майнеров блоков (H-1..1) нет в очереди, лидером становится первый майнер очереди.

Данный алгоритм позволяет детерминировано вычислить и проверить майнера, который должен был создать каждый блок цепочки, за счет возможности вычислить список авторизованных майнеров на каждый момент времени. Если блок не был создан назначенным лидером в отведенное время, в текущий раунд не производится блоков, раунд «пропускается». Лидеры, пропускающие создание блоков, временно исключаются из очереди по алгоритму, описанному в пункте *Исключение неактивных майнеров*.

Валидным считается блок, выпущенный лидером A_r с временем блока T_H из полуинтервала (T_0 +(r-1)*(t+t_s); T_0 +(r-1)*(t+t_s)+t]. Блок, созданный майнером не в свою очередь или не вовремя, не считается валидным. После раунда длительностью t сеть синхронизирует данные в течение t_s . У лидера A_r есть время t_s для того, чтобы распространить валидный блок по сети. Если каким-либо узлом сети за время t_s не был получен блок от лидера A_r , этот узел признает раунд «пропущенным» и ожидает новый блок H в следующем раунде r+1, от следующего лидера A_{r+1} .

Параметры консенсуса: тип (PoS или PoA), t, t_s задаются в конфигурационном файле узла сети. *Параметр t при этом должен совпадать у всех участников сети*, иначе произойдет форк сети.

5.2.2 Синхронизация времени между узлами сети

Каждый узел сети должен синхронизировать время приложения с доверенным NTP-сервером в начале каждого раунда. Адрес и порт сервера указывается в конфигурационном файле ноды. Сервер должен быть доступен каждой ноде сети.

5.2.3 Исключение неактивных майнеров

Если каким-либо майнером N_{ban} раз подряд было пропущено создание блока, этот майнер исключается из очереди на t_{ban} последующих блоков (параметр `ban-duration-blocks` в конфигурационном файле). Исключение выполняется каждым узлом самостоятельно на основании вычисляемой очереди Q_H и информации о блоке H и майнере M_H . С помощью параметра P_{ban} задается максимально допустимая доля исключенных майнеров в сети относительно всех активных майнеров в любой момент времени. Если при достижении N_{ban} пропусков раунда известно, что максимальная доля исключенных майнеров P_{ban} достигнута, то исключение очередного майнера не производится.

5.2.4 Мониторинг

Мониторинг консенсуса PoA помогает выявлять факты создания и распространения невалидных блоков, а также пропуски очереди майнерами. Дальнейшие действия по выявлению и устранению неисправностей, а также блокировке вредоносных узлов выполняются администраторами сети.

В целях мониторинга процесса формирования блоков для алгоритма PoA в InfluxDB размещаются данные:

- Активный список майнеров, отсортированный в порядке предоставления прав на майнинг.
- Плановая временная метка раунда.
- Фактическая временная метка раунда.
- Текущий майнер.

5.2.5 Изменение параметров консенсуса

Изменение параметров консенсуса (время раунда и периода синхронизации) выполняется на основании данных конфигурационного файла ноды (см. врезку) на высоте «from-height». Если одна из нод не укажет новые параметры, то произойдет форк.

Пример конфигурации:

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

ГЛАВА 6

Криптография

Платформа Waves Enterprise предоставляет возможность выбора используемой криптографии в зависимости от особенностей реализуемого проекта и юрисдикции заказчика.

6.1 Хеширование

Операции хеширования в платформе выполняются функциями Blake2b256 и Keccak256 последовательно, либо функцией «Стрибог» в соответствии с ГОСТ Р 34.11-2012 «Информационная технология. Криптографическая защита информации. Функция хеширования». Размер блока выходных данных 256 бит.

6.2 Электронная подпись

Алгоритмы генерации ключей, формирования и проверки электронной подписи реализованы на базе эллиптической кривой Curve25519 (ED25519 с ключами X25519), либо в соответствии с ГОСТ Р 34.10-2012 «Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи».

6.3 Шифрование данных

В платформе реализована возможность шифрования данных при помощи сессионных ключей на базе протокола Диффи-Хелмана. Операция применяется для шифрования любого вида текстовой информации, например, данных смарт-контрактов, которые не должны быть доступны для других участников блокчейна. Шифрование может выполняться как индивидуально для каждого получателя, с формированием уникального экземпляра шифротекста, так и с формированием единого шифротекста для группы получателей.

Используемые алгоритмы для симметричного шифрования соответствуют стандарту AES, либо ГОСТ Р 34.12-2015 «Информационная технология. Криптографическая защита информации. Блочные шифры».

Для операций шифрования/расшифрования данных применяются симметричные ключи CEK и KEK. CEK (Content Encryption Key) используется для шифрования текстовых данных, KEK (Key Encryption Key) используется для шифрования CEK. Ключ CEK формируется блокчейн-узлом случайным образом с применением соответствующих алгоритмов хеширования. Ключ KEK формируется нодой на базе алгоритма Диффи-Хелмана, используя публичные и приватные ключи отправителя и получателей, и применяется для шифрования ключа CEK.

Описание методов шифрования и их использования приведено в разделе *Операции шифрования данных*.

ГЛАВА 7

Ролевая модель

В блокчейн-платформе реализован механизм ограничения действий участников на основании ролевой модели, которая позволяет участникам платформы защититься от угроз следующих типов:

- атаки недобросовестных майнеров на блокчейн-сеть;
- несанкционированный выпуск токенов;
- несанкционированный доступ к конфиденциальной информации;
- иные противоправные действия злоумышленников.

Порядок выдачи и отзыва разрешений приведен в разделе *Управление ролями участников*.

7.1 Список ролей

В таблице ниже приведен список возможных ролей платформы:

Название роли	Полномочия
permissioner	Формирование транзакций для изменения списка разрешений
blacklister	Формирование транзакций для изменения black list
miner	Создание новых блоков
issuer	Формирование транзакций по выпуску, перевыпуску и сжиганию токенов
dex	Формирование exchange транзакции (deprecated)
contract_developer	Формирование транзакций на создание docker-контракта
connection-manager	Формирование транзакций на регистрацию/удаление ноды в блокчейн сети
banned	Запрещено отправлять какие-либо транзакции в блокчейн. Группа всех участников с данной ролью образует blacklist

7.2 Модель разрешений

Модель разрешений описывает механизм применения различных типов разрешений при валидации операций в блокчейн-сети.

Подсказка: Нода с ролью **permissioner** может себе присвоить любую роль из существующих в системе.

Действие	Условие разрешения действия
Назначение или удаление роли	Наличие роли permissioner
Добавление или удаление из blacklist	Наличие роли blacklister
Регистрация нового узла сети	Наличие роли connection-manager
Формирование и выпуск блоков	Наличие роли miner
Операции с токенами (issue, reissue, burn)	Наличие роли issuer
Перевод токенов (transfer, masstransfer)	Отсуствие пользователя в blacklist
Лизинг токенов (lease, lease cancel)	Отсуствие пользователя в blacklist
Создание псевдонима (alias)	Отсуствие пользователя в blacklist
Создание docker-контракта	Наличие роли contract_developer
Исполнение docker-контракта	Отсуствие пользователя в blacklist

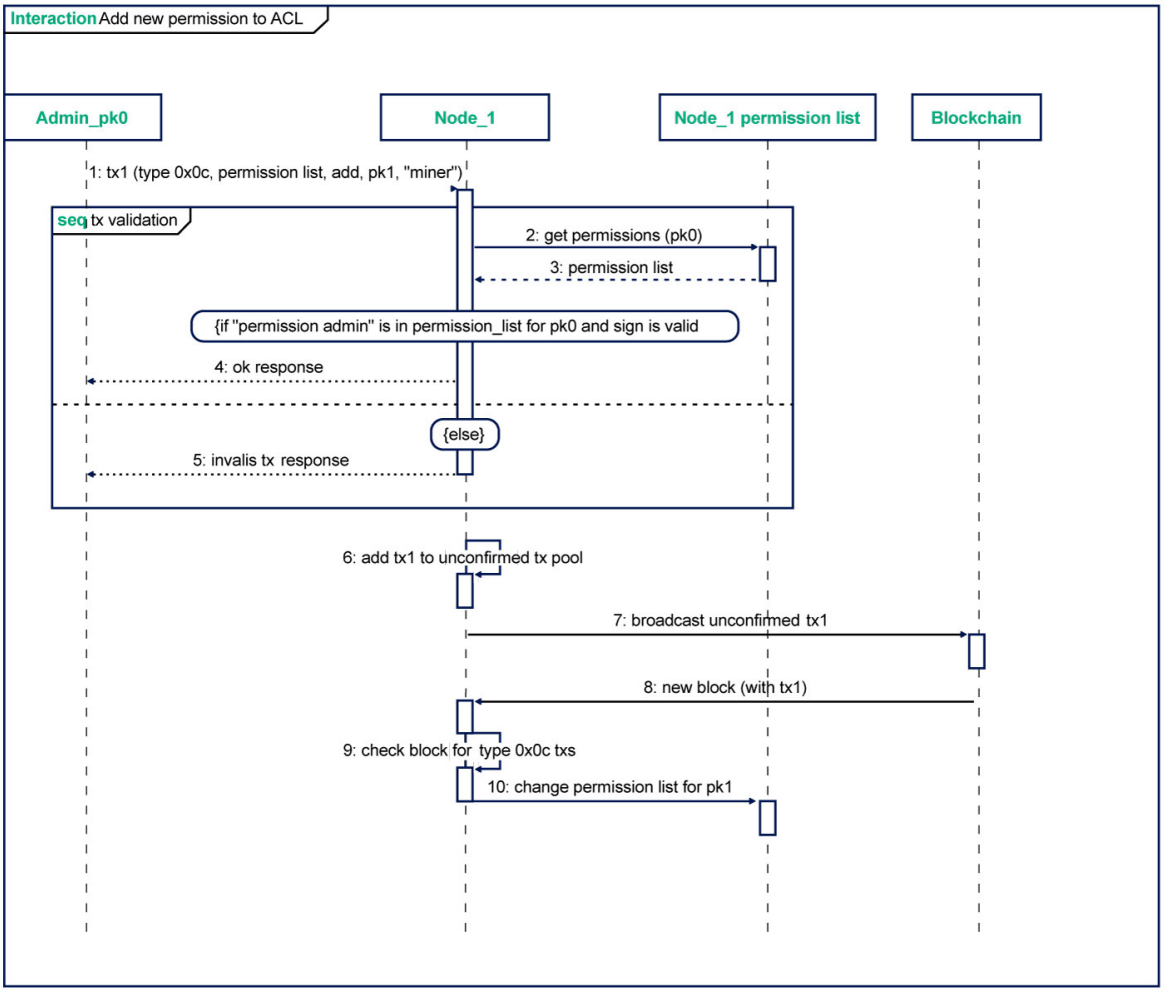
7.3 Обновление списка разрешений

Для изменения списка разрешений используется permission-транзакция.

JSON описание:

- 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

Последовательность действий при обновлении списка разрешений приведена на схеме ниже.



При изменении списка разрешений платформа выполняет следующие проверки:

1. Отправитель не находится в blacklist.
2. У отправителя есть роль permissioner.
3. DueTimestamp (время действия роли) > Timestamp (текущее время).
4. Такая роль не активна (в случае добавления), либо активна (в случае удаления).

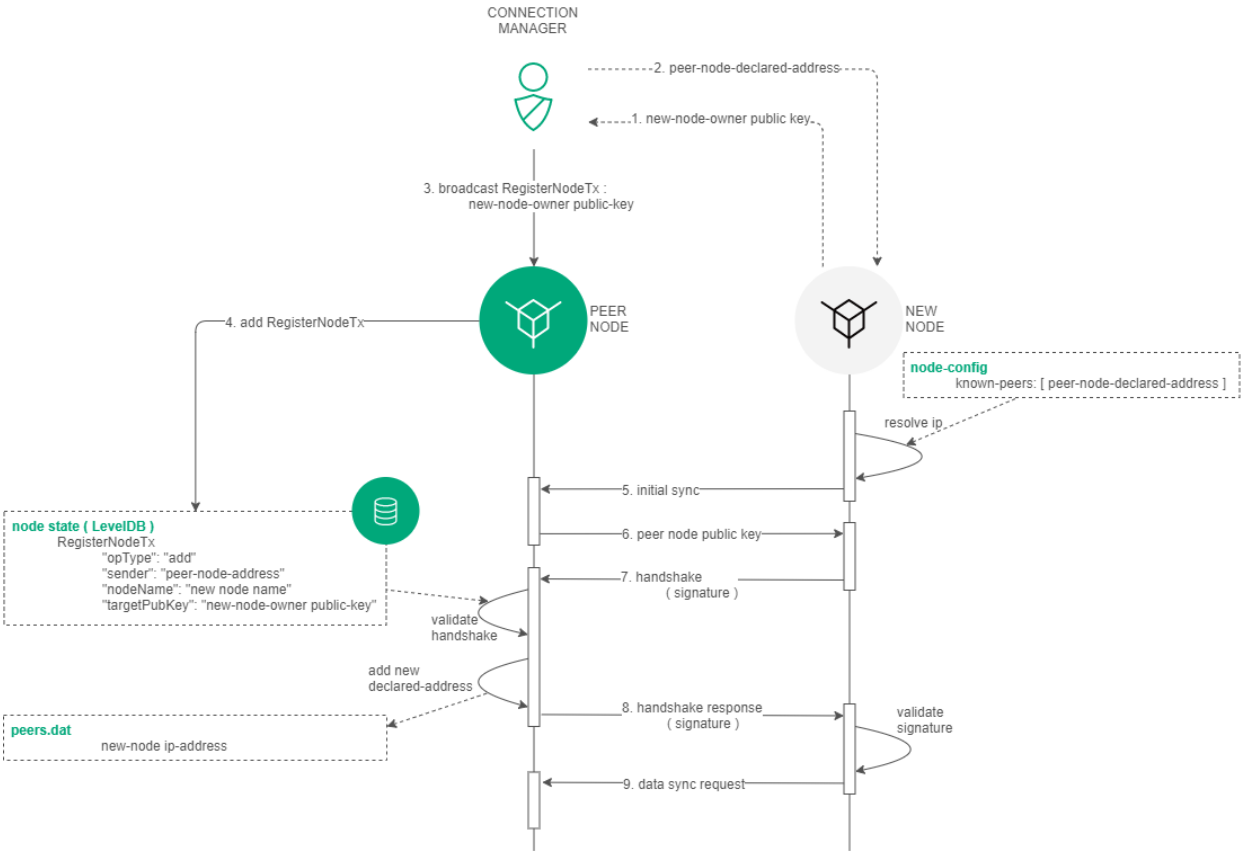
ГЛАВА 8

Управление доступом

Блокчейн-платформа Waves Enterprise реализует закрытую модель блокчейна, в которой подключение новых участников модерируется полномочиями отдельного пользователя. Этот вид блокчейна похож на публичный открытый блокчейн за исключением того, что данные в нем открыты не для всех. Преимуществом данной модели является его повышенная, по сравнению с открытыми блокчейнами, безопасность, а также возможность гибкой настройки уровней доступа и распределения прав.

В блокчейне Waves Enterprise правом на подключение участников к сети обладает пользователь с ролью «Connection Manager». Доказательством возможности подключения к блокчейн сети является отдельная транзакция *111 RegisterNode*. В данной транзакции указываются учетные данные подключаемого узла. По результатам добавления подобных транзакций у каждого узла формируется таблица разрешенных участников.

Каждая попытка подключения участника сопровождается *handshake-сообщением*, в котором помимо служебной информации указывается область данных с доказательством принадлежности к подключаемой сети - в упрощенном виде это совокупность публичного ключа с электронной подписью участника. Поскольку публичный ключ подключаемого участника уже сохранен в хранилище остальных пиров, то участник получивший handshake-запрос сверяет подпись и предоставленный ранее в блокчейне публичный ключ. Если проверка завершилась успехом, то участник формирует ответный handshake-запрос, при успехе которого устанавливается соединения между сторонами. После успешного подключения участники выполняет синхронизацию с сетью, а так же синхронизацию таблицы соотстветствия блокчейн и сетевых адресов узлов, что необхомо в дальнейшем в процессе пересылки *конфиденциальных данных*.

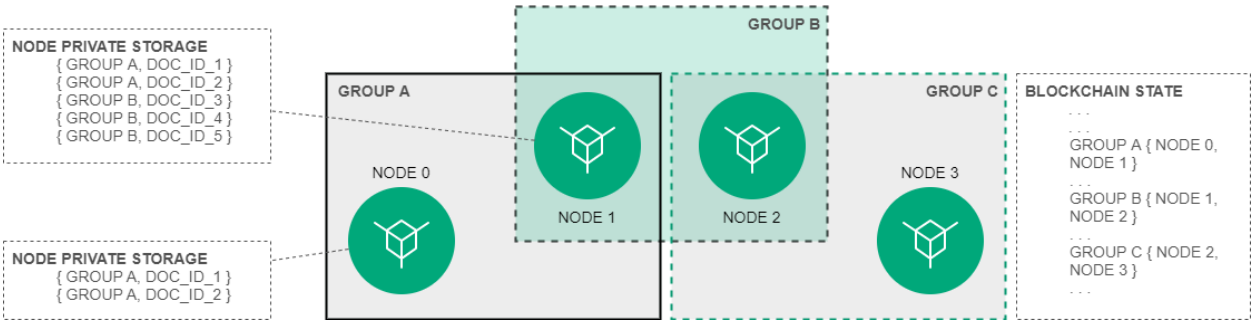


Процесс отключения какого-либо участника от сети аналогичен ранее описанному процессу за тем исключением, того что пользователь с ролью «Connection Manager» выпускает транзакцию *111 RegisterNode* с параметром `"opType": "remove"`. Поскольку, handshake-запрос выполняется с частотой 1 раз в 30 секунд, то следующий, после удаления участника из сети запрос, будет запрещен, ввиду отсутствия учетных данных подключаемого участника в таблице блокчейн-узла.

ГЛАВА 9

Конфиденциальность данных

Блокчейн-платформа Waves Enterprise позволяет организовать передачу и хранение конфиденциальных данных между участниками сетевого взаимодействия. Защита конфиденциальных данных при их передаче и хранении обеспечивается набором групп, которые содержат список участников, которые могут обмениваться приватными данными между собой.



9.1 Группы доступа

Группа доступа создаётся участниками, которым необходимо обмениваться приватными данными. Группу доступа может создать любой участник сети и включить в неё любой состав других нод сети. Обмениваться информацией внутри группы могут только ноды.

Группа доступа имеет следующие параметры:

- имя (policyName);
- описание (policyDescription);
- срок действия (policyDueDate);
- список получателей конфиденциальных данных (policyRecipients);
- список участников с правами на редактирование состава участников группы (policyOwners).

Создание группы доступа происходит при помощи отправки в блокчейн транзакции *CreatePolicy* (type = 112, создание группы).

Владельцы группы имеют право изменять состав участников группы доступа. Для изменения состава участников группы необходимо отправить в блокчейн транзакцию *UpdatePolicy* (type = 113, редактирование группы доступа).

Для внешних приложений в *API ноды* реализованы запросы, возвращающие сведения по группе и данным, которые передаются в рамках данной группы: `GET /privacy/{policy}/recipients`, `GET /privacy/{policy}/getHashes`, `GET /privacy/getInfo/{hash}`.

9.2 Отправка и получение данных

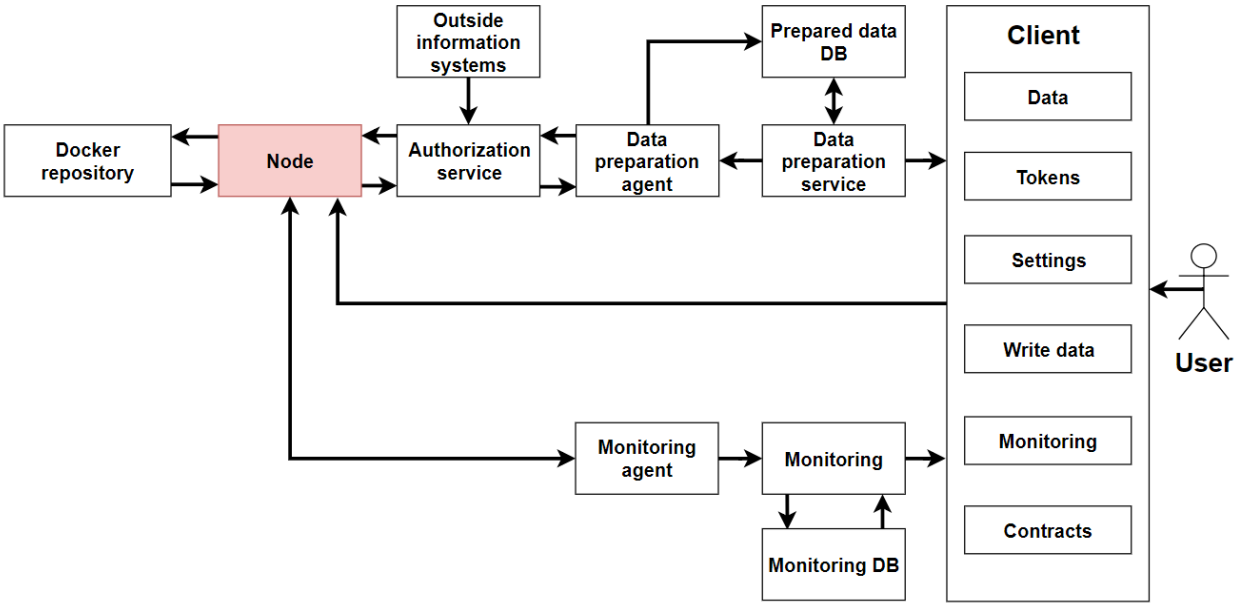
Отправляемые данные пересылаются посредством `POST /privacy/sendData` запросом через собственную ноду организации, в которой проверяется принадлежность отправителя к указанной им группе. Если проверка выполнена успешно, то данные записываются в хранилище ноды, и инициируется транзакция *PolicyDataHash* (type = 114, отправка хеш-суммы данных в сеть) с посчитанной хеш-суммой от передаваемых данных. Передать в сеть данные можно размером не более 20 МБ.

При получении транзакции с хеш-суммой от передаваемых данных принимающая сторона проверяет причастность блокчейн-узла организации к указанной в транзакции группе. Если участник состоит в группе, то выполняется запрос `getPrivateData` на получение конфиденциальных данных. Запрос выполняется по сетевому адресу участника группы по установленному P2P соединению. Для обеспечения безопасности при передачи данных по незащищенному каналу связи используется набор алгоритмов шифрования на симметричном ключе и создания сессионных ключей, а также протокол Диффи - Хеллмана.

ГЛАВА 10

Клиент

Клиент Waves Enterprise — это удобный способ управления вашим блокчейном Waves Enterprise. Клиент предназначен для работы в *публичной сети* Waves Enterprise.



Клиент включает в себя разделы для использования всех возможностей блокчейна:

- «Данные» — позволяет найти информацию о транзакциях или пользователях, благодаря гибкому поиску и развитой системе фильтров.
- «Токены» — позволяет переводить, выпускать, передавать в аренду токены.
- «Контракты» — предоставляет инструменты для публикации и вызова docker-контрактов. Для публикации доступны контракты из репозитория, адрес которого был указан при сборке клиента.
- «Запись данных» — позволяет отправлять из интерфейса транзакции с данными и файлы.

- «Настройки» — позволяет управлять разрешениями на действия пользователей в блокчейне.

Клиент поддерживает следующие типы браузеров:

- Google Chrome.
- Mozilla Firefox.
- Opera.
- Apple Safari.
- Microsoft Edge.

Если веб-интерфейс клиента некорректно отображается, или возникают какие-либо ошибки в процессе загрузки страниц, обновите ваш браузер до последней версии.

Данные

Раздел содержит информацию о транзакциях в блокчейне. Для получения информации используйте фильтры и поиск с указанием полей транзакций для поиска.

Доступны фильтры транзакций:

- Все транзакции - отображение всех транзакций.
- Транзакции с данными - отображение транзакций только с данными.
- Токены - выборка транзакций с токенами. При выборе доступны контекстные фильтры по типам операций с токенами (например, перевод, аренда или эмиссия токенов).
- Разрешения - выборка транзакций по операциям с псевдонимами и по пользовательским разрешениям. При выборе доступны контекстные фильтры по типам разрешений (например, майнинг, публикация контрактов или управление доступом).
- Группы - выборка транзакций по операциям с группами доступа к конфиденциальным данным. При выборе доступны контекстные фильтры по типам операций (например, создание или редактирование группы доступа).
- Контракты - выборка транзакций по операциям с контрактами. При выборе доступны контекстные фильтры по типам контрактов (например, Docker или RIDE).
- Неподтвержденные транзакции - выборка по неподтверждённым транзакциям.
- Пользователи - информация о пользователях. При выборе доступны контекстные фильтры по типам разрешений (например, майнинг, публикация контрактов или управление доступом).

Токены

Раздел отображает баланс авторизованной учетной записи, а также позволяет переводить токены другим участникам сети, передавать токены в аренду и выпускать токены. Выпуск токенов требует разрешения «Управление токенами».

Контракты

Раздел отображает информацию по существующим контрактам в сети, позволяет публиковать и запускать выбранные контракты. Доступна фильтрация в поисковой строке по параметрам транзакций. Для публикации контрактов нужно разрешение «Публикация контрактов».

Запись данных

Раздел позволяет создавать транзакции с данными и просматривать информацию о существующих транзакциях с данными.

Настройки

Раздел отображает информацию об аккаунте пользователя (публичный и приватный ключи, секретная фраза), о текущей версии клиента и позволяет сменить язык интерфейса. Также в настройках выдаются разрешения другим пользователям. Для выдачи необходимо разрешение «Управление разрешениями».

ГЛАВА 11

Блоки, транзакции, сообщения

11.1 Блоки

В этом разделе приведена структура хранения блоков в блокчейн-платформе Waves Enterprise.

Порядковый номер поля	Поле	Тип	Размер поля в байтах
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

Генерирующая подпись (Generation signature) вычисляется на основе хеша (Blake2b256) от следующих полей:

Порядковый номер поля	Поле	Тип	Размер поля в байтах
1	Previous block's generation signature	Bytes	32
2	Generator's public key	Bytes	32

Подпись блока вычисляется на основе следующих данных:

Порядковый номер поля	Поле	Тип	Размер поля в байтах
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 Транзакции

В этом разделе приведена структура хранения транзакций в блокчейн-платформе Waves Enterprise. Для некоторых типов транзакций введено версионирование.

Важно: Все транзакции используют поле `timestamp`, содержащее временную метку в формате **Unix Timestamp** в миллисекундах.

Таблица 1: Типы транзакций

№	Тип транзакции	Описание
1	*Genesis transaction*	Первоначальная привязка баланса к адресам, создаваемым при старте блокчейна нод
3	*Issue Transaction*	Выпуск токенов
4	*Transfer Transaction*	Перевод токенов
5	*Reissue Transaction*	Перевыпуск токенов
6	*Burn Transaction*	Сжигание токенов
8	*Lease Transaction*	Передача токенов в аренду
9	*Lease Cancel Transaction*	Отмена аренды токенов
10	*Create Alias Transaction*	Создание псевдонима
11	*MassTransfer Transaction*	Массовый перевод токенов. Указана минимальная комиссия
12	*Data Transaction*	Транзакция с данными в виде полей с парой ключ-значение. Указана минимальная комиссия
13	*SetScript Transaction*	Транзакция, привязывающая скрипт с RIDE-контрактом к аккаунту
14	*Sponsorship Transaction*	Транзакция, подписывающая спонсорский ассет
15	*SetAssetScript*	Транзакция, привязывающая скрипт с RIDE-контрактом к ассету
101	*Genesis Permission Transaction*	Назначение первого администратора сети для дальнейшей раздачи прав
102	*Permission Transaction*	Выдача/отзыв прав у аккаунта
103	*CreateContract Transaction*	Создание Docker-контракта
104	*CallContract Transaction*	Вызов Docker-контракта
105	*ExecutedContract Transaction*	Выполнение Docker-контракта
106	*DisableContract Transaction*	Отключение Docker-контракта
107	*UpdateContract Transaction*	Обновление Docker-контракта
110	*GenesisRegisterNode Transaction*	Регистрация ноды в генезис-блоке при старте блокчейна
111	*RegisterNode Transaction*	Регистрация новой ноды в сети
112	*CreatePolicy Transaction*	Создание группы доступа к конфиденциальным данным
113	*UpdatePolicy Transaction*	Изменение группы доступа
114	*PolicyDataHash Transaction*	Отправка в сеть хеша данных

Дополнительная информация приведена в разделе *Комиссии в сети «Waves Enterprise Mainnet»*

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. issueTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
senderPublicKey		+	+	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 для вызова метода sign

```
{
   "type": 3,
   "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. TransferTransaction

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

JSON для вызова метода sign


```
{
  "type": 4,
  "version": 2,
  "sender": "3M6dRZXaJY9oMA3fJKhMALyYKt13D1aimZX",
  "password": "",
  "recipient": "3M6dRZXaJY9oMA3fJKhMALyYKt13D1aimZX",
  "amount": 40000000000,
  "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. ReissueTransaction

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

JSON для вызова метода sign


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


Broadcasted JSON


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


(continues on next page)

(продолжение с предыдущей страницы)


```
    "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. BurnTransaction

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

JSON для вызова метода sign

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

Broadcasted JSON


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


(continues on next page)

(продолжение с предыдущей страницы)

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

11.2.6 8. LeaseTransaction

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

JSON для вызова метода sign

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

Broadcasted JSON

```
{
    "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
    "amount": 1000,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "proofs": [
↪"5jvmWKmU89HnxXFXNAd9X41zmiB5fSGoXMirsaJ9tNeyiCAJmjm7MR48g789VucckQw2UExaVXfhsdEBuUrchvrq" ],
    "fee": 100000,
    "recipient": "3N1ksBqc6uSksdiYjCzMtvEpiHhS1JjkbPh",
    "id": "6Tn7ir9MycHW6Gq2F2dGok2stokSwXJadPh4hW8eZ8Sp",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    "type": 8,
    "version": 2,
    "timestamp": 1551449299545,
    "height": 1190
}
```

11.2.7 9. LeaseCancelTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
senderPublicKey		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
chainId		+	+	Byte
leaseId	+ (txId)	+	+	Byte
lease		+		
password	+ (opt)			String
height		+		

JSON для вызова метода sign

```
{
    "type": 9,
    "version": 2,
    "fee": 100000,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "password": "",
    "txId": "6Tn7ir9MycHW6Gq2F2dGok2stokSwXJadPh4hW8eZ8Sp"
}
```

Broadcasted JSON

```
{
   "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
    "leaseId": "6Tn7ir9MycHW6Gq2F2dGok2stokSwXJadPh4hW8eZ8Sp",
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "chainId": 84,
    "proofs": [
↪"2Gns72hraH5yay3eiWeyHQEA1wTqiiAztaLjHinEYX91FEv62HFW38Hq89GnsEJFHUvo9KHYtBBrb8hgTA9wN7DM" ],
    "fee": 100000,
    "id": "9vhxB2ZDQcqiumhQbCPnAoPBLuir727qgJhFeBNmPwmu",
    "type": 9,
    "version": 2,
    "timestamp": 1551449835205,
    "height": 1190
}
```

11.2.8 10. CreateAliasTransaction

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

JSON для вызова метода sign

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

Broadcasted JSON

```
{
    "type": 10,
    "id": "DJTaiMpb7eLuPW5GcE4ndeE8jWsWPjx8gPYmbZPJjpag",
    "sender": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",
    "senderPublicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
    "fee": 0,
    "timestamp": 1549290335781,
    "signature":
↪"2qYepod9DhpxVad1yQDbv1QzU4KLKcbjjdtGY7De2272K76nbQfaXsRnyd31hUE8bhvLjjpHRdtoLVzbBDzRZYEY",
    "proofs": [
↪"2qYepod9DhpxVad1yQDbv1QzU4KLKcbjjdtGY7De2272K76nbQfaXsRnyd31hUE8bhvLjjpHRdtoLVzbBDzRZYEY" ],
    "version": 1,
    "alias": "testperson4",
    "height": 59245
}
```

11.2.9 11. MassTransferTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
senderPublicKey		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
assetId	+ (opt)	+	+	ByteStr
attachment	+ (opt)	+	+	
transfers	+	+	+	List[Transfer]
transferCount		+	+	
totalAmount		+		
password	+ (opt)			String
height		+		

JSON для вызова метода sign

```
{
    "type": 11,
    "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
        },
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        {
            "recipient": "3N7BA6J9VUBfBRutuMyjF4yKTUEtrRFfHMc",
            "amount": 100000
        }
    ],
    "id": "D9jUSHHcJqVAvkFMiRfDBhQbUzoSfQqd9cjaunMmtjdu",
    "timestamp": 1551450279637
}
```

11.2.10 12. DataTransaction

Предупреждение: Транзакция имеет ограничения:

1. Количество данных в секции «data» передаваемого JSON должно быть не более 100 пар `"key":"value"`,

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

2. Байтовое представление транзакции после подписания не должно превышать размер в 150 КБ.

Подсказка: Параметр `senderPublicKey` не требуется указывать, если подписывается транзакция, в которой автор и отправитель совпадают.

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type	Size (Bytes)
type	+	+	+	Byte	1
id		+		Byte	1
sender	+	+		PublicKeyAccount	3264
senderPublicKey	+ (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 для вызова метода sign

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

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
senderPublicKey		+	+	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 для вызова метода 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": [
↪"2QiGYS2dqh8QyN7Vu2tAYaioX5WM6rTSDPGbt4zrWS7QKTzojmR2kjppvGNj4tDPsYPbcDunqBaqhaudLyMeGFgG" ],
    "chainId": 84,
    "version": 1,
    "script": "base64:AQQAAAAHJG1hdGNoMAUAAAACdHgG+RXSzQ==",
    "name": "faucet",
    "description": "",
    "height": 3805
}
```

11.2.12 14. SponsorshipTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
senderPublicKey		+	+	PublicKeyAccount
assetId	+ (opt)	+	+	ByteStr
fee	+	+	+	Long
isEnabled	+	+	+	Boolean
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 для вызова метода sign

```
{
    "sender": "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwD9t",
    "assetId": "G16FvJk9vabwxjQswh9CQAhbZzn3QrwqWjwnZB3qNVox",
    "fee": 100000000,
    "isEnabled": false,
    "type": 14,
    "password": "1234",
    "version": 1
}
```

Broadcasted JSON

```
{
    "type": 14,
    "id": "Ht6kpnQJHJskN8kwszF8rck3E5tQiuiM1fEN42wGfdk7",
    "sender": "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwD9t",
    "senderPublicKey": "Gt55fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUophy89",
    "fee": 100000000,
    "assetId": "G16FvJk9vabwxjQswh9CQAhbZzn3QrwqWjwnZB3qNVox",
    "timestamp": 1545986757233,
    "proofs": [
↪"5TfgYS2dqh8QyN7Vu2tAYaioX5WM6rTSDPGbt4zrWS7QKTzojmR2kjppvGNj4tDPsYPbcDunqBaqhaudLyMeGFh7" ],
    "chainId": 84,
    "version": 1,
    "isEnabled": false,
    "height": 3865
}
```

11.2.13 15. SetAssetScriptTransaction

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

JSON для вызова метода sign

```
{
    "type": 15,
    "version": 1,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "password": "",
    "fee": 100000000,
    "script": "base64:AQQAAAAHJG1hdGNoMAUAAAACdHgG+RXSzQ==",
    "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+RXSzQ==",
    "height": 61895
}
```

11.2.14 101. 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				

11.2.15 102. PermitTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
senderPublicKey		+	+	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 для вызова метода sign

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

Broadcasted JSON

```
{
  "senderPublicKey": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
  "role": "contract_developer",
```

(continues on next page)

(продолжение с предыдущей страницы)


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


11.2.16 103. CreateContractTransaction

Предупреждение: Байтовое представление транзакции после подписания не должно превышать размер в 150 КБ.

В поле `contractVersion` указывается версия контракта, значение `1` - для нового контракта, значение `2` - для обновленного контракта. Контракт обновляется при помощи *107* транзакции. При создании контракта автоматически создается транзакция *104*, вызывающая контракт для его проверки. Если контракт не выполнился или выполнился с ошибкой, то транзакции 103 и 104 отбрасываются и не попадают в блок.

Поле `feeAssetId` опционально и используется только для *gRPC контрактов* (значение поля `version = 2`).

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type	Size(Bytes)
type	+	+	+	Byte	1
id		+		Byte	1
sender	+	+		PublicKeyAccount	3264
senderPublicKey		+	+	PublicKeyAccount	3264
password	+ (opt)			String	32767
fee	+	+	+	Long	8
timestamp	+ (opt)	+	+	Long	8
proofs		+	+	List[ByteStr]	32767
version		+	+	Byte	1
feeAssetId	+ (opt)			Byte	1
image	+	+	+	Array[Bytes]	32767
imageHash	+	+	+	Array[Bytes]	32767
contractName	+	+	+	Array[Bytes]	32767
params	+	+	+	List[DataEntry[_]]	32767
height		+			8

JSON для вызова метода sign


```
{
  "fee": 100000000,
```


(continues on next page)

(продолжение с предыдущей страницы)


```
  "image": "stateful-increment-contract:latest",
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
  "contractName": "stateful-increment-contract",
  "sender": "3PudkbvjV1nPj1TkuuRahh4sGdgfr4YAUV2",
  "password": "",
  "params": [],
  "type": 103,
  "version": 1,
 }
```


Broadcasted JSON


```
{
    "type": 103,
    "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
    "fee": 500000,
    "timestamp": 1550591678479,
    "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
    "version": 1,
    "image": "stateful-increment-contract:latest",
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
    "contractName": "stateful-increment-contract",
    "params": [],
    "height": 1619
}
```


11.2.17 104. CallContractTransaction

Предупреждение: Байтовое представление транзакции после подписания не должно превышать размер в 150 КБ.

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type	Size(Bytes)
type	+	+	+	Byte	1
id		+		Byte	1
sender	+	+		PublicKeyAccount	3264
senderPublicKey		+	+	PublicKeyAccount	3264
fee	+	+	+	Long	8
timestamp	+ (opt)	+	+	Long	8
proofs		+	+	List[ByteStr]	32767
version		+	+	Byte	1
contractVersion	+	+	+	Byte	1
contractId	+	+	+	ByteStr	32767
params	+	+	+	List[DataEntry[_]]	32767
height		+			8
password	+ (opt)			String	32767

JSON для вызова метода sign


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

        }
    ],
    "version": 1,
    "contractVersion": 1
}
```


Broadcasted JSON


```
{
    "type": 104,
    "id": "9fBrL2n5TN473g1gNfoZqaAqAsAJCuHRHYxZpLexL3VP",
    "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
    "senderPublicKey": "2YvzcVLrqLCqouVrFZynjfotEuPNV9GrdauNpgdWXLsq",
    "fee": 10,
    "timestamp": 1549365736923,
    "proofs": [
↪"2q4cTBhDkEDkFxr7iYaHPAv1dzaKo5rDaTxPF5VHryyYTXxTPvN9Wb3YrsDYixKiUPXBnAyXzEcnKPFRCW9xVp4v" ],
    "version": 1,
    "contractVersion": 1,
    "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
    "params":
    [
        {
        "key": "a",
        "type": "integer",
        "value": 1
        },
        {
        "key": "b",
        "type": "integer",
        "value": 100
        }
    ]
}
```

11.2.18 105. ExecutedContractTransaction

Предупреждение: Байтовое представление транзакции после подписания не должно превышать размер в 150 КБ.

Field	Broadcasted JSON	Blockchain state	Type
type	+	+	Byte
id	+		Byte
sender	+		PublicKeyAccount
senderPublicKey	+	+	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": [
↪"5whBipAWQgFvm3myNZe6GDd9Ky8199C9qNxLBHqDNmVAUJW9gLf7t9LBQDi68CKT57dzmnPJpJkrwKh2HBSwUer6" ],
    "version": 1,
    "tx":
         {
            "type": 103,
            "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
            "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
            "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
            "fee": 500000,
            "timestamp": 1550591678479,
            "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
            "version": 1,
            "image": "stateful-increment-contract:latest",
            "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
            "contractName": "stateful-increment-contract",
            "params": [],
            "height": 1619
          },
    "results": [],
    "height": 1619
}
```

11.2.19 106. DisableContractTransaction

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

JSON для вызова метода sign

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

Broadcasted JSON

```
{
"type": 106,
"id": "8Nw34YbosEVhCx18pd81HqYac4C2pGjyLKck8NhSoGYH",
"sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
"senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
"fee": 500000,
"proofs": [
↪"5GqPQkuRvG6LPXgPoCr9FogAdmhAaMbyFb5UfjQPUKdSc6BLuQSz75LAWix1ok2Z6PC5ezPpjqzqnr15i3RQmaEc" ],
"version": 1,
"contractId": "Fz3wqAWWcPMT4M1q6H7crLKtToFJvbeLSvqjaU4ZwMpg",
"height": 1632
}
```

11.2.20 107. UpdateContractTransaction

Предупреждение: Байтовое представление транзакции после подписания не должно превышать размер в 150 КБ.

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

JSON для вызова метода sign

```
{
  "image" : "registry.wvservices.com/we-sc/tdm-increment3:1028.1",
  "sender" : "3Mxxz9pBYS5fJMARJNQmzYUHxiWAtvMzSRT",
  "password": "",
  "fee" : 100000000,
  "contractId" : "EnsihTUHSNAB9RcWXJbiWT98X3hYtCw3SBzK8nHQRCWA",
  "imageHash" : "0e5d280b9acf6efd8000184ad008757bb967b5266e9ebf476031fad1488c86a3",
  "type" : 107,
  "version" : 1
 }
```

Broadcasted JSON

```
{
 "senderPublicKey":
↪"5qBRDm74WKR5xK7LPs8vCy9QjzzqK4KCb8PL36fm55S3kEi2XZETHFgMgp3D13AwgE8bBkYrzvEvQZuabMfEyJwW",
 "tx":
 {
 "senderPublicKey":
↪"5qBRDm74WKR5xK7LPs8vCy9QjzzqK4KCb8PL36fm55S3kEi2XZETHFgMgp3D13AwgE8bBkYrzvEvQZuabMfEyJwW",
 "image":"registry.wvservices.com/we-sc/tdm-increment3:1028.1",
 "sender":"3Mxxz9pBYS5fJMARJNQmzYUHxiWAtvMzSRT",
 "proofs":[
↪"3tNsTyteeZrxEbVSv5zPT6dr247nXsVWR5v7Khx8spypgZQUdorCQZV2guTomutUTcyxhJUjNkQW4VmSgbCtgm1Z"],
 "fee":0,
 "contractId":"EnsihTUHSNAB9RcWXJbiWT98X3hYtCw3SBzK8nHQRCWA",
 "id":"HdZdhXVveMT1vYzGTviCoGQU3aH6ZS3YtFpYujWeGCH6",
 "imageHash":"17d72ca20bf9393eb4f4496fa2b8aa002e851908b77af1d5db6abc9b8eae0217",
 "type":107,"version":1,"timestamp":1572355661572},
```

(continues on next page)

(продолжение с предыдущей страницы)


```
 "sender":"3HfRBedCpWi3vEzFSKEZDFXkyNWbWLWQmmG",
 "proofs":[
↪"28ADV8miUVN5EFjhqeFj6MADSXYjbxA3TsxSwFVs18jXAsHVaBczvnyoUSaYJsjRNmaWgXbpbduccRxpKGTs6tro"],
 "fee":0,"id":"7niVY8mjzeKqLBePvhTxFRfLu7BmcwVfqaqtbWAN8AA2",
 "type":105,
 "version":1,
 "results":[],
 "timestamp":1572355666866
 }
}
```


11.2.21 110. GenesisRegisterNodeTransaction

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

11.2.22 111. RegisterNodeTransaction

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

JSON для вызова метода sign


```
{
"type": 111,
"opType": "add",
"sender":"3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
"password": "",
"targetPubKey": "apgJP9atQccdBPAgJPwH3NBVqYXrapgJP9atQccdBPAgJPwHapgJP9atQccdBPAgJPwHDKkh6A8",
```


(continues on next page)

(продолжение с предыдущей страницы)

```
"nodeName": "Node #1",
"fee": 500000,
}
```

11.2.23 112. CreatePolicyTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+	+	Byte
sender	+	+	+	PublicKeyAccount
senderPublicKey		+	+	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 для вызова метода sign

```
 {
  "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "policyName": "Policy# 7777",
  "password":"sfgKYBFCF@#$fsdf()*%",
  "recipients": [
    "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
    "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
        "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
        "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx"
  ],
  "fee": 15000000,
  "description": "Buy bitcoin by 1c",
  "owners": [
    "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
    "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
  ],
  "type": 112
}
```

11.2.24 113. UpdatePolicyTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+	+	Byte
sender	+	+	+	PublicKeyAccount
senderPublicKey		+	+	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 для вызова метода sign

```
{
  "policyId": "7wphGbhqbmUgzuN5wzgqwqtViTiMdFezSa11fxRV58Lm",
  "password":"sfgKYBFCF@#$fsdf()*%",
  "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "proofs": [],
  "recipients": [
  "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
  "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
      "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
      "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
      "3NwJfjG5RpaDfxEhkwXgwD7oX21NMFCxJHL"
   ],
   "fee": 15000000,
   "opType": "add",
  "owners": [
    "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
    "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
  ],
  "type": 113,
 }
```

11.2.25 114. PolicyDataHashTransaction

Когда пользователь отправляет конфиденциальные данные в сеть при помощи *POST /privacy/sendData*, нода автоматически формирует транзакцию 114.

Field	Broadcasted JSON	Blockchain state	Type
type	+	+	Byte
id	+	+	Byte
sender	+	+	PublicKeyAccount
senderPublicKey	+	+	PublicKeyAccount
policyId	+	+	String
dataHash	+	+	String
fee	+	+	Long
timestamp	+	+	Long
proofs	+	+	List[ByteStr]
height		+	Long

11.3 Сетевые сообщения

В этом разделе приведена структура сетевых сообщений в блокчейн-платформе Waves Enterprise.

11.3.1 Network message

Все сетевые сообщения, за исключением Handshake, базируются на следующей структуре:

Порядковый номер поля	Поле	Тип	Размер поля в байтах
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 следующие: 0x12, 0x34, 0x56, 0x78. Контрольная сумма полезной нагрузки это первые 4 байта от _FastHash_ от байтов _Payload_. FastHash это хеш-функция Blake2b256(data).

11.3.2 Handshake message

Handshake сообщение предназначена для первичного обмена данными между двумя нодами. Авторизованный Handshake содержит блокчейн-адрес владельца ноды и подпись. Неподписанные Handshake сообщения не принимаются.

Авторизованный Handshake

Порядковый номер поля	Поле	Тип	Размер поля в байтах
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 lenght (P)	Byte	1
8	Consensus name lenght (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 сообщение отправляется для запроса сетевых адресов участников сети.

Порядковый номер поля	Поле	Тип	Размер поля в байтах
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 сообщение является ответом на запрос GetPeers.

Порядковый номер поля	Поле	Тип	Размер поля в байтах
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

Порядковый номер поля	Поле	Тип	Размер поля в байтах
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

Порядковый номер поля	Поле	Тип	Размер поля в байтах
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

Порядковый номер поля	Поле	Тип	Размер поля в байтах
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

Порядковый номер поля	Поле	Тип	Размер поля в байтах
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

Порядковый номер поля	Поле	Тип	Размер поля в байтах
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

Порядковый номер поля	Поле	Тип	Размер поля в байтах
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

Порядковый номер поля	Поле	Тип	Размер поля в байтах
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

ГЛАВА 12

Смарт-контракты

12.1 Смарт-контракты RIDE

Смарт-контракт это скрипт, проверяющий транзакции на соблюдение условий. Скрипты расширяют логику блокчейна под ваши бизнес-задачи. Комиссия за смарт-контракт фиксирована. Скрипт может быть опубликован как на аккаунт, так и на набор выпущенных вами токенов.

Для аккаунта — проверяться будут все транзакции, исходящие с данного адреса. Аккаунт с опубликованным скриптом называется Смарт-аккаунт. Для набора токенов — проверяться будут все транзакции с данным набором токенов. Набор токенов с опубликованным скриптом называется Смарт-ассет. На одном аккаунте может быть только 1 скрипт. Соответственно, любой установленный скрипт заменяет предыдущий, в т.ч. «скрипт по умолчанию».

12.1.1 RIDE

Для создания скрипта в блокчейне Waves Enterprise используется язык RIDE (о языке RIDE можно почитать на портале WAVES). Скрипты, написанные на RIDE, при проверке условий используют следующие данные:

- Данные исходящей транзакции.
- Данные аккаунта, от имени которого осуществляется транзакции.
- Данные о балансе третьих аккаунтов.
- Данные о высоте блокчейна.

Принцип работы скрипта это pattern matching, т.е. сопоставление с образцом. В скрипте указываются типы транзакций и проверки для них с условиями, при которых возможно исполнение соответствующих транзакций. Также доступны возможности:

- запретить транзакцию независимо от условий,
- разрешить независимо от условий.

Работа с разрешениями и запретами по типам транзакций возможна как указанием конкретных типов транзакций, так и механикой «всё, кроме». Скрипт устанавливается транзакцией типа SetScript, соответственно, её разрешение, запрещение или проверку на выполнение условий надо явно указывать.

Важно: Скрипт не изменяет транзакцию, только проверяет соответствие условиям.

12.1.2 Сложность скриптов

RIDE не является Тьюринг полным языком, что накладывает ограничения на доступную сложность логики. Вычислительная сложность принудительно ограничена сверху для гарантии производительности сети. Для сложных бизнес-процессов, механика которых не укладывается в один скрипт, возможна комбинация из нескольких скриптов (на нескольких адресах, соответственно), либо комбинации скриптов на наборе токенов и на адресе. Мы активно развиваем возможности RIDE, в ближайшее время в языке появятся вложенные функции, что расширит его возможности по сложности реализуемых задач.

12.1.3 Подписи и скрипт «по умолчанию»

Каждая транзакция в блокчейне обладает криптографическим доказательством целостности, основанном на подписи транзакции закрытым ключом отправителя. Это также гарантирует неотчуждаемость авторства транзакций. Для лучшего понимания механизма работы представьте, что «по умолчанию» на каждом адресе установлен скрипт, который проверяет единственное условие для каждой исходящей транзакции — принадлежность подписи адресу отправителя.

Пример кода скрипта, установленного «по умолчанию»:

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

Механика скриптов расширяет возможности по проверке подписи. Транзакция может быть подписана несколькими пользователями или не от имени того адреса, от которого отправлена. Это необходимо, т.к. контракт проверяет только транзакции, исходящие со своего адреса. Соответственно, пользователь формирует транзакцию от имени контракта, подписывает её своей подписью и она успешно проходит проверку скриптом.

Важно: Если в вашем скрипте явно не указана проверка подписи, то она не осуществляется. Соответственно, при ручном формировании тела транзакции, возможно отправлять транзакции от имени адреса со скриптом, с подписью другого адреса.

12.1.4 Данные на аккаунте

На адресах в блокчейне Waves Enterprise можно хранить данные в формате ключ-значение. Данные, хранящиеся на адресе, доступны для просмотра по запросу вида *вернуть данные с адреса по ключу*. Данные размещаются на адресе при отправке транзакции с данными. Т.к. скрипты на RIDE stateless, транзакции с даннными формируют обновляемое хранилище данные, к которому обращается скрипт. Настройка проверки подписи на смарт-аккаунте позволяет нескольким пользователям совместно работать с данными на смарт-аккаунте. Например, со статусами движения документа.

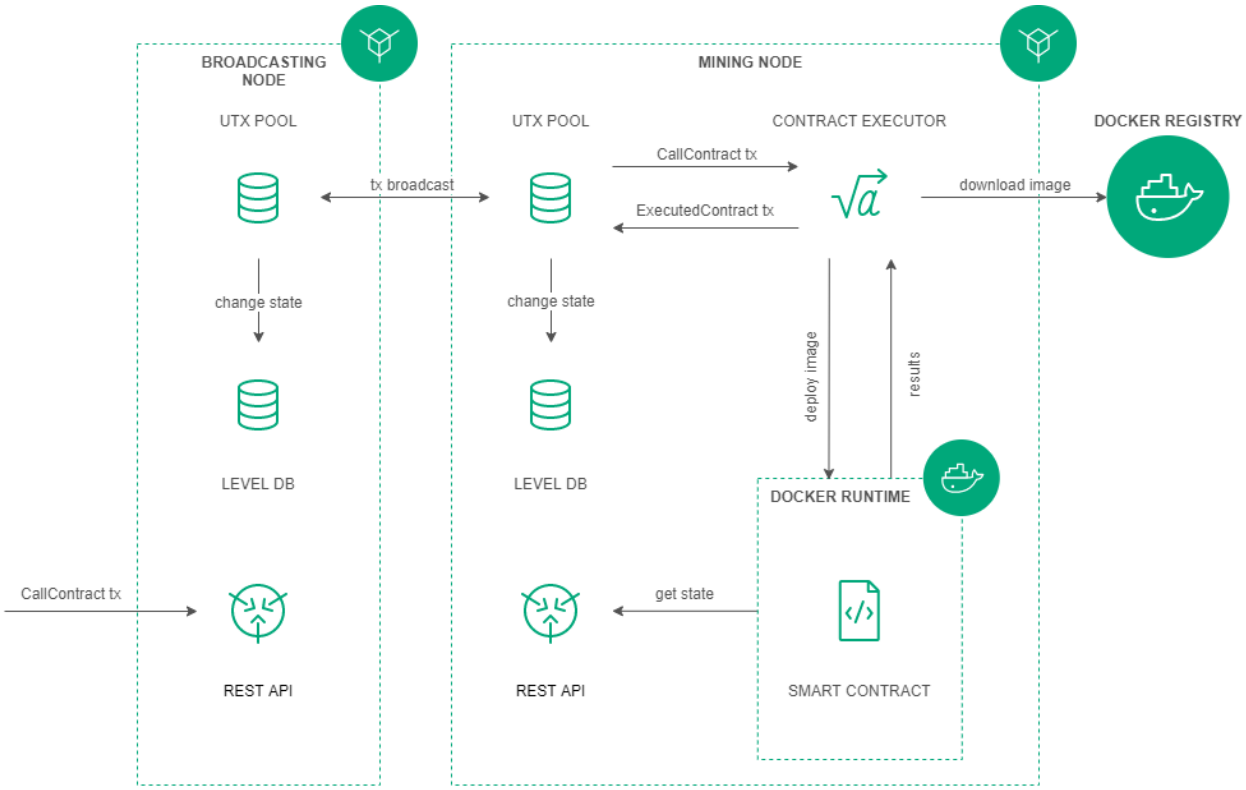
Важно: Ключи уникальны для адреса. Одному ключу на адресе соответствует только одно значение. При публикации нового значения для существующего ключа, оно будет перезаписано. Историю и

автора изменений можно отследить в блокчейне.

12.2 Смарт-контракты Docker

В дополнение к контрактам, реализованным на базе *скриптов RIDE*, для смарт-аккаунтов и смарт-ассетов платформа Waves Enterprise предоставляет возможность разработки и использования Тьюринг-полных смарт-контрактов. Для реализации Тьюринг-полных контрактов выбран подход, в котором программы запускаются в изолированной среде Docker-контейнера. При этом разработка приложений может выполняться без ограничений на используемый язык программирования. Каждое приложение запускается в Docker-контейнере для возможности изоляции и управления ресурсами, доступными конкретному приложению. Для хранения смарт-контрактов используется Docker Registry с доступом на чтение образов (Docker images) контрактов для машин с нодами. Доступ к состоянию ноды может выполняться через REST API ноды или через gRPC.

Важно: Если нода выполняет майнинг смарт-контрактов Docker, и на ней запущен движок Docker, то на этой ноде также должен работать и демон Docker.



12.2.1 Создание контракта

Создание смарт-контракта начинается с подготовки Docker-образа, который состоит из программного кода контракта, необходимого окружения и из специального сценарного файла Dockerfile. Подготовленный Docker-образ собирается (build) и отправляется в Docker Registry.

Пример Dockerfile при использовании REST API:

```
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"
```

Пример Dockerfile при использовании gRPC:

```
FROM python:3.9-rc-buster
RUN pip3 install grpcio-tools
ADD src/contract.py /
ADD src/protobuf/common_pb2.py /protobuf/
ADD src/protobuf/contract_pb2.py /protobuf/
ADD src/protobuf/contract_pb2_grpc.py /protobuf/
ADD run.sh /
RUN chmod +x run.sh
ENTRYPOINT ["/run.sh"]
```

Установка контракта реализуется через публикацию специальной (CreateContractTransaction) транзакции, содержащей ссылку на образ в Docker Registry. Для использования REST API или gRPC необходимо указать версию транзакции *103*. После получения транзакции нода скачивает образ по указанной в поле «image» ссылке, образ проверяется и запускается в виде Docker-контейнера.

12.2.2 Исполнение контракта

Исполнение смарт-контрактов инициируется специальной (CallContractTransaction) транзакцией, в которой содержится идентификатор контракта и параметры для его вызова. По идентификатору транзакции определяется Docker-контейнер. Контейнер запускается, если не был запущен ранее. В контейнер передаются параметры запуска контракта. Смарт-контракты изменяют своё состояние через обновление пар ключ - значение.

12.2.3 Изменение контракта

Изменять Docker смарт-контракт может только его разработчик, который создал транзакцию *103* и сохранил свою *роль* `contract_developer` в момент изменения смарт-контракта. Изменение смарт-контракта выполняется при помощи транзакции *107*. Необходимо, чтобы смарт-контракт был активным.

После включения *107* транзакции в блок ноды-майнеры скачивают образ контракта и запускают его для проверки корректности исполнения. Далее выпускается *105* транзакция с включением в неё *107* транзакции.

12.2.4 Запрет вызова контракта

При необходимости разработчик контракта может запретить его вызов. Для этого публикуется специальная (DisableContractTransaction) транзакция с указанием идентификатора контракта. Контракт становится недоступным после его отключения, но по нему можно получить информацию из блокчейна впоследствии.

12.2.5 Описание транзакций

Для реализации взаимодействия между блокчейном и Docker контрактом реализованы следующие транзакции:

Код	Тип транзакции	Назначение
103	*CreateContractTransaction*	Публикация контракта. Подписание транзакции производится пользователем с ролью *«contract_developer»*
104	*CallContractTransaction*	Вызов контракта. Подписание транзакции производится инициатором исполнения контракта
105	*ExecutedContractTransaction*	Запись результата исполнения контракта на стейт контракта. Подписание транзакции производится нодой, формирующей блок
106	*DisableContractTransaction*	Запрет вызова контракта. Подписание транзакции производится пользователем с ролью *«contract_developer»*
107	*UpdateContractTransaction*	Обновление кода контракта. Подписание транзакции производится пользователем с ролью *«contract_developer»* Изменять контракт может только его разработчик и инициатор *103* транзакции

12.2.6 Конфигурация ноды

Скачивание и исполнение Docker-контрактов, инициированных транзакциями с кодами `103 - 107` выполняется на нодах с включенной опцией `docker-engine.enable = yes` (подробнее в разделе *«Установка и настройка» > «Запуск Docker-контрактов»*).

12.2.7 REST API

Описание методов REST API, которые может использовать Docker-контракт, приведено в разделе *Методы API, доступные смарт-контракту*.

12.2.8 gRPC

Описание методов gRPC, которые может использовать Docker-контракт, приведено в разделе *Сервисы gRPC, используемые смарт-контрактом*.

12.2.9 Примеры реализации

- *Создание простого контракта*

ГЛАВА 13

Анкоринг

В приватном блокчейне транзакции обрабатываются определенным списком участников, каждый из которых заранее известен. Малое, по сравнению с публичной сетью, количество участников блоков и транзакций в приватном блокчейне несёт угрозу подмены информации. Перезапись цепочки блоков и транзакций, особенно в случае использования PoS консенсуса, становится реальной.

Для повышения доверия участников приватного блокчейна к размещенным в нём данным разработан механизм анкоринга. Анкоринг позволяет проверить данные на неизменность. Гарантия неизменности достигается публикацией данных из приватного блокчейна в более крупную сеть, где подмена данных маловероятна из-за большего количества участников и блоков. Публикуемые данные — подпись и высота блоков приватной сети. Взаимная связность двух и более сетей повышает их стойкость, т.к. для подлога или изменнения данных в результате long-range атаки необходимо атаковать все связанные сети.

13.1 Как работает анкоринг в блокчейне Waves Enterprise

Анкоринг работает следующим образом:

1. Выполняются *настройки анкоринга* в конфигурационном файле ноды приватного блокчейна. При указании параметров, ответственных за работу анкоринга, устанавливайте рекомендованные значения, чтобы избежать сложностей в работе приватного блокчейна.
2. Через каждый заданный диапазон блоков `height-range` нода фиксирует информацию о блоке на высоте `current-height - threshold` в виде транзакции в Targetnet. В качестве такой транзакции используется *Data Transaction* со списком пар полей `key-value`, описание которых приведено *ниже*. После отправки транзакции нода получает её высоту в Targetnet.
3. Нода выполняет проверку высоты блокчейна в Targetnet каждые 30 секунд, пока высота не достигнет значения **высота созданной транзакции** + `height-above`.
4. При достижении высоты блокчейна Targetnet, определённой в пункте 3, и положительной проверке наличия первой транзакции в блокчейне Targetnet нода создаёт вторую транзакцию с данными для анкоринга уже в приватном блокчейне.

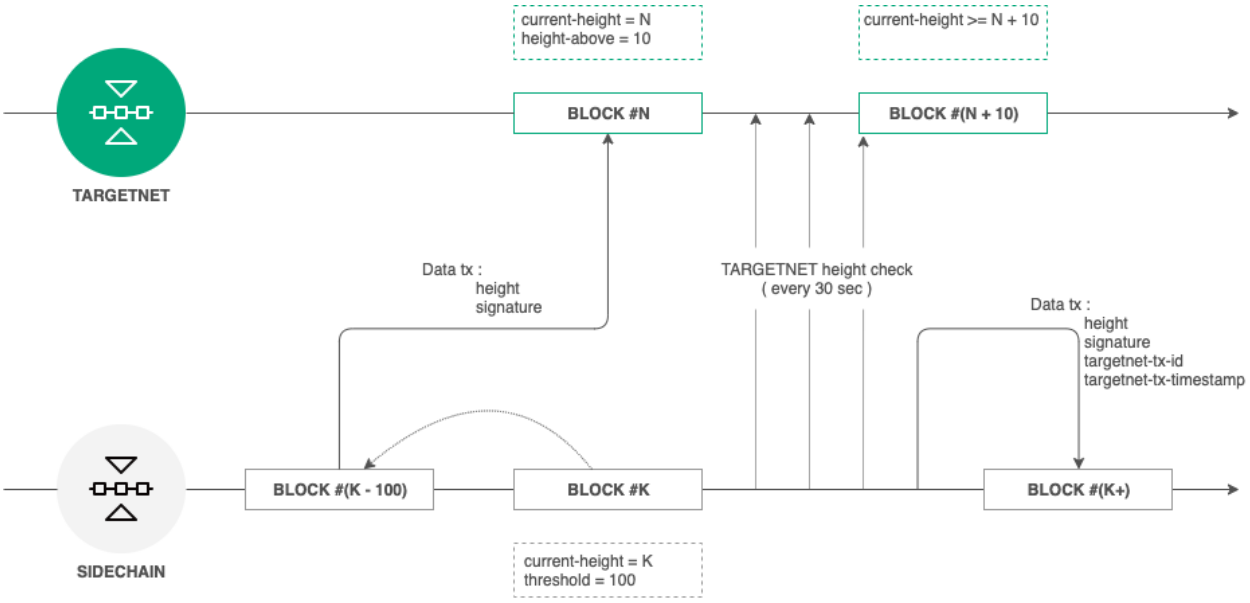


Рис. 1: Схема анкоринга в Targetnet

13.2 Структура транзакции для анкоринга

Транзакция для отправки в Targetnet содержит следующие поля:

- `height` - высота сохраняемого блока из приватного блокчейна.
- `signature` - подпись сохраняемого блока из приватного блокчейна.

Транзакция, создаваемая в приватном блокчейне, содержит следующие поля:

- `height` - высота сохраняемого блока из приватного блокчейна.
- `signature` - подпись сохраняемого блока из приватного блокчейна.
- `targetnet-tx-id` - идентификатор транзакции для анкоринга в Targetnet.
- `argetnet-tx-timestamp` - дата и время создания транзакции для анкоринга в Targetnet.

13.3 Ошибки, возникающие в процессе анкоринга

Ошибки в анкоринге могут возникать на любом этапе. В случае возникновения ошибок в приватном блокчейне всегда публикуется *Data Transaction* с кодом и описанием ошибки. Транзакция об ошибке содержит следующие данные:

- `height` - высота сохраняемого блока из приватного блокчейна.
- `signature` - подпись сохраняемого блока из приватного блокчейна.
- `error-code` - код ошибки.
- `error-message` - описание ошибки.

Таблица 1: Типы ошибок при анкоринге

Код	Сообщение об ошибке	Возможная причина
0	`Unknown error`	При отправке транзакции в Targetnet произошла неизвестная ошибка
1	`Fail to create data transaction for Targetnet`	Создание транзакции для отправки в Targetnet завершилась ошибкой
2	`Fail send transaction to Targetnet`	Публикация транзакции в Targetnet завершилась ошибкой (это может быть ошибка JSON-запроса)
3	`Invalid http status of response from Targetnet transaction broadcast`	В результате публикации транзакции в Targetnet вернулся отличный от 200 код
4	`Fail to parse http body of response from Targetnet transaction broadcast`	В результате отправки транзакции в Targetnet вернулся нераспознаваемый JSON-запрос
5	`Targetnet return transaction with id='$TargetnetTxId' but it differ from transaction that we sent id='$sentTxId`	В результате отправки транзакции в Targetnet вернулся отличный от первой транзакции идентификатор
6	`Targetnet didn't respond on transaction info request`	Targetnet не ответил на запрос об информации о транзакции
7	`Fail to get current height in Targetnet`	Не удалось получить текущую высоту в Targetnet
8	`Anchoring transaction in Targetnet disappeared after height rise enough`	Анкоринг транзакция пропала из Targetnet после увеличения высоты на значение `height-above`
9	`Fail to create sidechain anchoring transaction`	Не удалось опубликовать анкоринг транзакцию в приватном блокчейне
10	`Anchored transaction in sidechain was changed during Targetnet height arise await, looks like a rollback has happened`	Ожидалось подтверждение транзакции в Targetnet произошел откат приватного блокчейна, идентификатор анкоринг транзакции был изменен

ГЛАВА 14

Интеграционные сервисы

14.1 Сервис авторизации

Сервис авторизации является внешним по отношению к ноде и обеспечивает авторизацию всех компонентов блокчейн-сети. Сервис авторизации построен на базе протокола OAuth 2.0. OAuth 2.0 является открытым фреймворком для реализации механизма авторизации, позволяющим предоставлять третьей стороне ограниченный доступ к защищенным ресурсам пользователя без передачи третьей стороне логина и пароля. В общем виде поток данных между участниками информационного взаимодействия на базе протокола OAuth 2.0 приведен ниже.

Средством авторизации является JSON Web Token. Токены используются для авторизации каждого запроса от клиента к серверу и имеют ограниченное время жизни. Клиент получает два вида токена - access и refresh. Access токен используется для авторизации запросов на доступ к защищенным ресурсам и для хранения дополнительной информации о пользователе. Refresh токен используется для получения нового access токена и обновления refresh токена.

В общем виде схема авторизацию включает в себя следующие операции:

1. Клиент (компонент блокчейн-сети, такой как корпоративный клиент, дата-сервис или сторонее приложение) единоразово предоставляет свои аутентификационные данные сервису авторизации.
2. В случае успешного прохождения процедуры первичной аутентификации сервис авторизации сохраняет аутентификационные данные клиента в хранилище данных, генерирует и отправляет клиенту подписанные access и refresh токены. В токенах указываются время жизни токена и основные данные клиента, такие как идентификатор и роль. Аутентификационные данные клиентов хранятся в конфигурационном файле сервиса авторизации. Каждый раз перед отправкой запроса стороннему сервису клиент проверяет время жизни access токена и, в случае истечения срока жизни токена, обращается к сервису авторизации для получения нового access токена. Для запросов к сервису авторизации используется refresh токен.
3. Используя актуальный access токен, клиент отправляет запрос на получение данных стороннего сервиса.

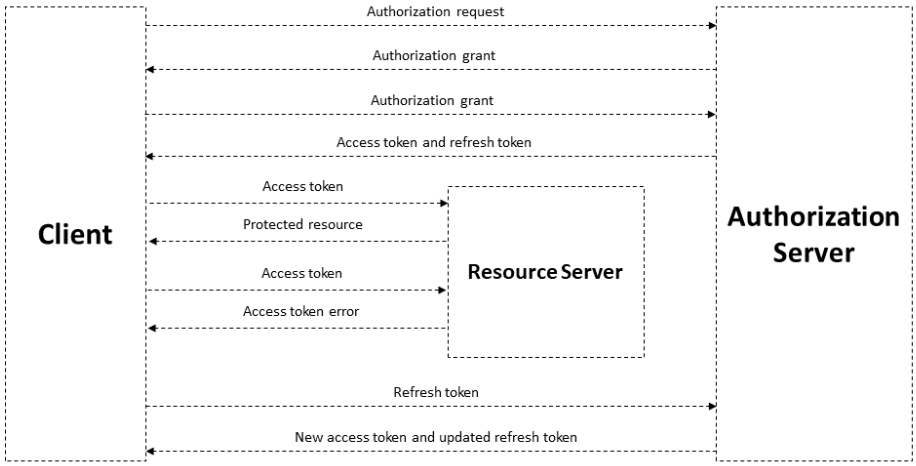


Рис. 1: Общая схема авторизации на базе протокола OAuth 2.0

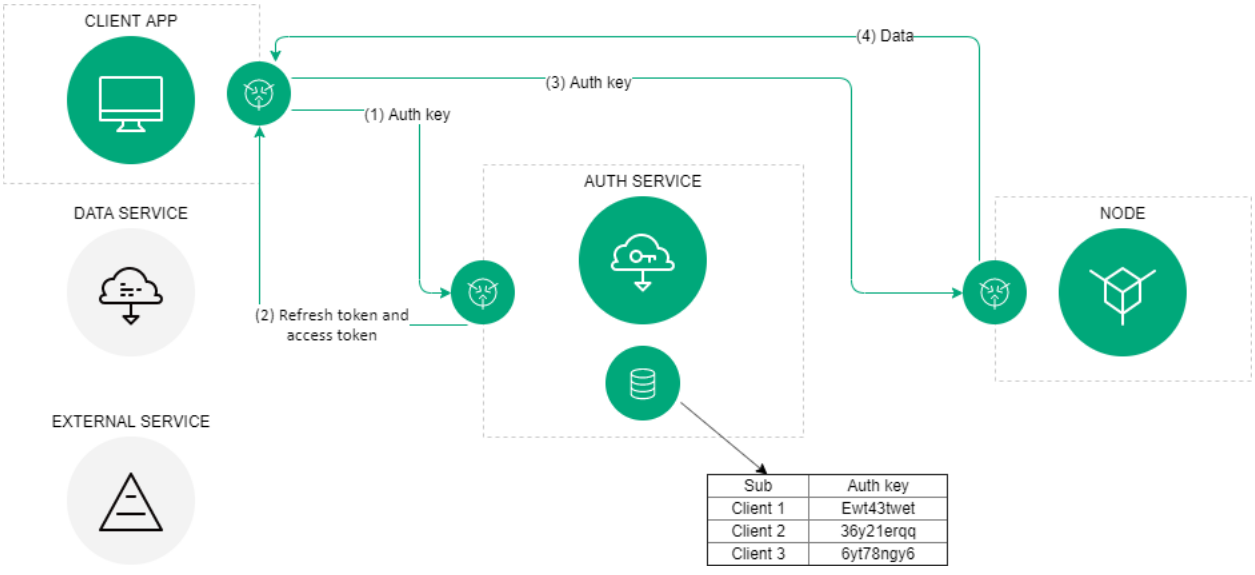


Рис. 2: Схема авторизации на блокчейн-платформе Waves Enterprise

4. Сторонний сервис проверяет время жизни access токена, его целостность, а также сравнивает полученный ранее публичный ключ сервиса авторизации с ключом, содержащимся в подписи access токена. В случае успешной проверки сторонний сервис предоставляет клиенту запрашиваемые данные.

14.2 Сервис подготовки данных

Сервис агрегирует данные из блокчейна в реляционную БД и предоставляет API для доступа к этим данным. Функциональные возможности сервиса спроектированы под потребности клиента Waves Enterprise. Для запросов доступны уточняющие параметры.

Для использования сервиса необходимо развернуть свой экземпляр клиента и ноды из комплекта поставки. На данный момент доступ к API сервиса подготовки данных в публичной сети ограничен. Описание REST API сервиса подготовки данных приведено в разделе *REST API сервиса подготовки данных*.

ГЛАВА 15

Системные требования

Ниже приведены аппартные и системные требования.

Вариант	vCPU	RAM	SSD	Режим работы JVM
Минимальные требования	2+	2Gb	50Gb	java -Xmx2048M -jar
Рекомендуемые требования	2+	4+ Gb	50+ Gb	java -Xmx4096M -jar

Подсказка: «Xmx» - флаг, определяющий максимальный размер доступной для JVM памяти.

Требования к окружению для платформы Waves Enterprise

- Oracle JRE 1.8 (64-bit) или OpenJDK 12.0.1
- Docker CE
- Docker-compose

ГЛАВА 16

Установка и запуск платформы

На данный момент мы поддерживаем операционные системы на базе Unix (например, популярные дистрибутивы Linux или MacOS). Однако платформа Waves Enterprise может быть запущена и под Windows в экспериментальном режиме. Вы также можете использовать такие решения, как виртуальные машины с Unix подобной системой и среду Docker для установки и запуска платформы Waves Enterprise на операционной системе Windows.

Для установки платформы вам потребуется наличие установленных Docker Engine и Docker Compose в среде развертки.

В зависимости от целей установки вам могут потребоваться следующие файлы:

1. `docker-compose.yml` – настроечный файл, используемый Docker Compose для запуска приложений в контейнерах.
2. `generators-X.X.X.jar` - вспомогательная утилита, используемая в платформе Waves Enterprise для создания ключевых пар, API keys, подписания генезис блоков и других операций.

`docker-compose.yml` и `generators-X.X.X.jar` вы можете скачать пройдя по ссылке.

3. Конфигурационные файлы для утилиты `generators-X.X.X.jar`:
    - `accounts.conf` - конфигурационный файл для генерации аккаунтов;
    - `api-key-hash.conf` - конфигурационный файл для генерации значений полей `api-key-hash` и `privacy-api-key-hash` при выборе авторизации по хешу ключевой строки `api-key`.
4. `node.conf` - основной конфигурационный файл ноды, определяющий ее принципы работы и набор опций. Ниже приведены примеры конфигурационных файлов ноды:
    - `mainnet.conf` - для подключения к сети Mainnet;
    - `partnernet.conf` - для подключения к сети Partnernet.

Примеры конфигурационных файлов для утилиты `generators-X.X.X.jar` и `node.conf` вы можете скачать на официальной странице платформы Waves Enterprise на GitHub.

Подробная информация о конфигурационных файлах приведена в разделе *Подготовка конфигурационных файлов*.

5. `node.license` - лицензия на использование ноды, не является обязательной до высоты 30 000 блоков.

О способе получения `node.license` можно узнать в разделе *Получение лицензии*.

Цель установки	docker-compose.yml	generators-X.X.X.jar	node.conf	node.license
Проверка возможностей платформы	Требуется	Не требуется	Не требуется	Не требуется
Подключение ноды к сети Mainnet	Требуется	Требуется	Требуется	Требуется
Подключение ноды к сети Partnernet	Требуется	Требуется	Требуется	Требуется

16.1 Развёртывание платформы в режиме проверки возможностей (Sandbox)

Команда Waves Enterprise предлагает полностью автоматический режим развертывания для целей ознакомления с возможностями платформы. В этом режиме будет установлена блокчейн-сеть из трёх нод, а также дополнительные компоненты - *сервис авторизации*, *сервис подготовки данных* и *корпоративный клиент*. Все ключевые пары, используемые для подписания транзакций и блоков будут сгенерированы случайным образом.

В ознакомительном режиме вы можете взаимодействовать с блокчейном через клиентское приложение, либо REST/gRPC-интерфейсы ноды: отправлять транзакции, получать данные из блокчейна, устанавливать и вызывать смарт-контракты, а также передавать конфиденциальные данные между нодами.

1. Для установки платформы в режиме Sandbox откройте терминал и перейдите в директорию, содержащую файл `docker-compose.yml`, и выполните следующую команду:

```
docker run --rm -ti -v $(pwd):/config-manager/output wavesenterprise/config-manager:v1.2.1
```

В качестве последних трёх цифр укажите актуальную версию платформы.

2. Дождитесь результатов выполнения предыдущей команды и выполните следующую команду:

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

Внимание: На ОС Linux для выполнения команд могут понадобиться права администратора (права root).

После запуска контейнеров клиентское приложение будет доступно по адресу `http://localhost`, REST API ноды - `http://localhost/node-0`.

Для остановки запущенных нод и сервисов выполните следующую команду:

```
docker-compose down
```

16.2 Подключение одной ноды к сети Mainnet

По приведенной ниже инструкции можно подключить ноду к любой существующей сети.

Для подключения ноды к сети Mainnet вам потребуются следующие файлы: `docker-compose.yml`, `node.license`, `node.conf` и ключевое хранилище в виде файла `keystores.dat`.

Подсказка: Файл `keystores.dat` создается при генерации адреса нового участника.

---

1. Скачайте файл docker-compose.yml.
2. Скачайте файл mainnet.conf, переименуйте его в `node.conf` и отредактируйте следующие параметры:

- `owner-address`, `wallet.password` - адрес нового участника, от имени которого нода будет выполнять операции в блокчейне. Процесс генерации новой ключевой пары и файла `keystores.dat` с помощью утилиты `generators-X.X.X.jar` описан в разделе *Создание аккаунтов*;
- `node-name` - произвольное имя ноды;
- `auth.api-key-hash`, `auth.privacy-api-key-hash` - хеш от секретной фразы для получения доступа к *REST API ноды*. Процесс создания хешированной секретной фразы с помощью утилиты `generators-X.X.X.jar` описан в разделе *Создание аккаунтов*.

3. Создайте пустые файлы: `postgres.env`, `node-0.env`, `nginx-proxy.env`, `frontend.env`, `data-service.env`, `crawler.env`, `auth-service.env`.
4. Откройте файл `my-node/env/node-0.env`, скопируйте текст приведенный ниже:

```
LOG_LEVEL=DEBUG
WE_NODE_OWNER_PASSWORD_EMPTY=false
WE_NODE_OWNER_PASSWORD= /FILL
JAVA_OPTS=-Dwe.check-resources=false
```

5. В поле `WE_NODE_OWNER_PASSWORD` вместо `/FILL` введите пароль от ключевой пары, созданный при генерации адреса нового участника утилитой `generators-X.X.X.jar`.
6. Разместите скаченные и созданные ранее файлы в соответствии со приведенной ниже структурой:

```
my-node                              (directory with any name)
|- configs                           (directory)
|   |- nodes                         (directory)
|       |- node-0                    (directory)
|           |- node.conf             (file)
|           |- keystores.dat         (file)
|           |- node.license          (file, optional)
|- env                               (directory)
|   |- postgres.env                  (file)
|   |- node-0.env                    (file)
|   |- nginx-proxy.env               (file)
|   |- frontend.env                  (file)
|   |- data-service.env              (file)
|   |- crawler.env                   (file)
|   |- auth-service.env              (file)
|- docker-compose.yml                (file)
```

7. Выполните команду для запуска ноды:

```
docker-compose up -d node-0
```

После запуска контейнера REST API ноды будет доступен по адресу `http://localhost/node-0`.

Внимание: При наличии ошибок убедитесь, что не запущены другие конкурирующие контейнеры или программы. Для вывода списка запущенных контейнеров и их состояния введите командой `docker ps -a`. Для остановки выбранного контейнера - командой `docker stop [myContainer]`. Для остановки всех контейнеров вы можете ввести `docker stop $(docker ps -a -q)`. Команда `docker rm [myContainer]` удалит выбранный, `docker rm $(docker ps -a -q)` удалит все контейнеры.

Для остановки запущенных нод и сервисов выполните следующую команду:

```
docker-compose down
```

ГЛАВА 17

Ручная конфигурация ноды

Конфигурация ноды включает в себя следующие шаги:

17.1 Подготовка конфигурационных файлов

В конфигурации ноды используются следующие файлы:

- `accounts.conf` - конфигурационный файл для генерации аккаунтов.
- `api-key-hash.conf` - конфигурационный файл для генерации значений полей `api-key-hash` и `privacy-api-key-hash` при выборе авторизации по хешу ключевой строки `api-key`.
- `node.conf` - основной конфигурационный файл ноды, определяющий ее принципы работы и набор опций.

17.1.1 Конфигурационный файл для создания аккаунтов `accounts.conf`

При указании пути в параметрах файла `accounts.conf` необходимо использовать символ «прямого слэша» - / как разделитель уровней иерархии директорий. При работе в ОС Linux значение `wallet` должно соответствовать структуре каталогов операционной системы, например, `/home/contract/we/keystore.dat`. При настройке ноды не допускается использование кириллических символов при указании путей до рабочей директории, хранилища ключей и т.д.

```
// 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 {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    enabled = false
    url = "http://localhost:6862/utils/reload-wallet"
  }
}
```

Описание параметров конфигурационного файла представлено ниже.

- `waves-crypto` – выбор криптографического алгоритма («yes» - использовать *криптографию Waves*, «no» - использовать *ГОСТ-криптографию*);
- `chain-id` – идентифицирующий байт сети, значение потребуется дальше для внесения в параметр `address-scheme-character` в конфигурационный файл ноды;
- `amount` – количество генерируемых ключевых пар;
- `wallet` – путь до каталога хранения ключей на ноде, значение потребуется дальше для внесения в параметр `wallet > file` в конфигурационный файл ноды. Для криптографии Waves указывается путь до файла `keystore.dat` (пример, `${user.home}/we/keystore.dat`), для ГОСТ-криптографии - путь до директории (`${user.home}/we/keystore/`);
- `wallet-password` – пароль для доступа к закрытым ключам ноды, значение потребуется дальше для внесения в параметр `wallet > password` в конфигурационный файл ноды;
- `reload-node-wallet` - опция для обновления keyStore ноды без перезапуска приложения, по умолчанию установлено в значение «Выключено» (`false`). В параметре `url` указывается путь до метода `/utils/reload-wallet` REST API ноды.

17.1.2 Конфигурационный файл `api-key-hash.conf`

Конфигурационный файл `api-key-hash.conf` нужен только для генерации значений полей `api-key-hash` и `privacy-api-key-hash` при выборе авторизации по хешу ключевой строки `api-key`.

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

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

Описание параметров:

- `waves-crypto` – выбор криптографического алгоритма («yes» - использовать *криптографию Waves*, «no» - использовать *ГОСТ-криптографию*);
- `api-key` – ключ, который необходимо придумать. Значение данного ключа потребуется указать в запросах к REST API ноды (подробнее на странице *REST API ноды*).

17.1.3 Конфигурационный файл ноды `node.conf`

Если планируется подключение к существующей сети, то для упрощения подключения запросите готовый конфигурационный файл ноды у одного из участников сетевого взаимодействия или у администратора вашей сети. При создании сети с нуля или подключении к сети «Waves Enterprise Mainnet» пример конфигурационного файла ноды можно взять на странице проекта на GitHub. Об изменениях в конфигурационном файле ноды можно почитать в разделе *Изменения в конфигурационном файле ноды*.

Предупреждение: Для нод версии 1.0 и выше в конфигурационном файле ноды в корневой секции `node` необходимо наличие следующего параметра:

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

Данная опция становится активной после достижения суммарного количества блоков из параметров `feature-check-blocks-period = 15000` и `blocks-for-feature-activation = 10000` (25000 блоков), которые находятся в секции `blockchain`. При подключении к Mainnet или Partnernet данные параметры не могут быть изменены. Ноды без активации данной опции не смогут подключиться к сети.

Пример конфигурационного файла ноды представлен ниже. В данном примере отключены опции *анкоринга*, *Docker* смарт-контрактов и *групп* доступа к приватным данным. Также установлена *авторизация* по хешу ключевой строки `api-key` и криптография Waves. Описание параметров конфигурационного файла ноды вы можете найти *тут*.

Примечание: Если вы планируете использовать дополнительные опции, установите поле `enable` выбранной опции в значение `yes` или `true` и настройте секцию опции в соответствии с описанием её настройки.

Предупреждение: Заполните **ТОЛЬКО** те поля, где в качестве значений указано слово /**FILL**/.

```
  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"
```

(continues on next page)

(продолжение с предыдущей страницы)

```
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
    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: "3HQSr3VFCiE6JcWwV1yX8xttYbAGKTLV3Gz", amount:␣
↪3000000000000000 }
          #
          # Note:
          #   Sum of amounts must be equal to initial-balance above.
          #
        { recipient: " /FILL/ ", amount: 100000000000000 },
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        { recipient: " /FILL/ ", amount: 150000000000000 },
        { recipient: " /FILL/ ", amount: 50000000000000 },
      ]
      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: "EPxkVA9iQejsjQikovyxkkY8iHnbXsR3wjgkgE7ZW1Tt", 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

# 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 {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  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
    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"
  }
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# 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"

  # gRPC server settings for docker contracts with the gRPC API
  grpc-server {
    # gRPC server port
    port = 6865
    # Optional node host
    # host = "192.168.65.2"
  }

  # 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
  }

  # 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
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    max-buffer-size = 10
    # Max time for buffer. When time is out, the node processes all messages in batch
    max-buffer-time = 100ms
  }
 }
}
```

17.2 Изменения в конфигурационном файле ноды

В этом разделе приведена информация, помогающая выделить изменения в конфигурационном файле в зависимости от версии ноды.

Предупреждение: Если вы обновляете версию ноды, необходимо также обновить конфигурационный файл ноды. Без обновления конфигурационного файла нода работать не будет!

17.2.1 Изменения в конфигурационном файле ноды версии 1.2.0

Секция docker-engine

В секцию `docker-engine` добавлен параметр `grpc-server`, отвечающий за настройку gRPC сервера для работы docker контрактов с gRPC API:

```
grpc-server {
  # gRPC server port
  port = 6865
  # Optional node host
  # host = "192.168.65.2"
}
```

17.2.2 Изменения в конфигурационном файле ноды более ранних версий

Версия ноды 1.1.2

Версия ноды 1.1.0

17.3 Описание основных параметров и секций конфигурационного файла ноды

Для параметров в конфигурационном файле применяется несколько типов значений:

- Числовое значение, используемое для указания точного количества элементов. Это может быть количество транзакций, блоков, соединений.
- Числовое значение с указанием единиц измерения, используемое для определения временного периода или объёма памяти. В таком виде, как правило, указываются временные периоды в днях, часах или секундах, или объём кэш-памяти, например, `leveldb-cache-size = 256M` или `connection-timeout = 30s`.

- Строковое значение, используемое для указания адресов, путей к директориям, паролям и т.д. Путь к директории указывается в формате, приемлемом для текущей ОС. Значение указывается в кавычках.
- Массив, используемый для указания списков значений, например, адресов или публичных ключей. Значение указывается в квадратных скобках через запятую.
- Логический тип `no` или `yes`, используемый для активации различных опций.

Пример конфигурационного файла приведен на странице *подготовки конфигурационных файлов*. В состав файла входят следующие секции:

- *node* - общая секция, куда входят все секции и дополнительные параметры для настройки ноды.
- *synchronization.transaction-broadcaster* - настройка параметров синхронизации для отправки неподтверждённых транзакций в блокчейн.
- *ntp* - настройка параметров NTP-сервера.
- *blockchain* - настройка основных параметров блокчейна.
- *features* - настройка дополнительных параметров ноды.
- *network* - сетевые настройки.
- *wallet* - настройка доступа к закрытым ключам ноды.
- *miner* - настройка майнинга.
- *rest-api* - настройка REST API и типа авторизации для доступа к REST API.
- *privacy* - настройка групп доступа к конфиденциальной информации.
- *docker-engine* - настройка Docker смарт-контрактов.

17.3.1 Секция `node`

Дополнительные параметры секции:

- `waves-crypto` - тип *шифрования* в блокчейне. Возможные значения: `yes` - выбор криптографии Waves, `no` - выбор ГОСТ-криптографии.
- `directory` - основная директория для хранения ПО ноды.
- `data-directory` - директория для хранения данных блокчейна в LevelDB: блоки, транзакции, стейт ноды.
- `logging-level` - уровень логирования работы ноды. Возможные значения: `DEBUG`, `INFO`, `WARN`, `ERROR`, по умолчанию установлено значение `INFO`.
- `owner-address` - адрес ноды, которая будет владельцем конфигурационного файла. Если владелец ноды будет начальным участником сети, то его адрес и публичный ключ должны быть в genesis блоке.

17.3.2 Секция synchronization.transaction-broadcaster

- max-batch-size и max-batch-time – технические параметры, позволяющие регулировать скорость уменьшения очереди транзакций.
- min-broadcast-count – минимальное число соединений, которые можно использовать для отправки каждой транзакции в блокчейн. Значение не должно превышать число нод в сети минус один (нода-отправитель в расчёт не принимается).
- retry-delay – интервал повторной отправки транзакции, если количества текущих соединений не хватило, или произошли ошибки во время отправки.

17.3.3 Секция ntp

- servers - список адресов NTP-серверов. Рекомендуемое значение - ["0.pool.ntp.org", "1.pool.ntp.org", ... "10.pool.ntp.org"].
- request-timeout - таймаут для одного запроса на NTP-сервер. Рекомендуемое значение - 10 секунд.
- expiration-timeout - таймаут синхронизации запросов к NTP-серверу. Рекомендуемое значение - 1 минута.
- fatal-timeout - таймаут подключения к NTP-серверу. Рекомендуемое значение - 1 минута.

17.3.4 Секция blockchain

- type - тип блокчейна. Возможные значения MAINNET или CUSTOM. Значение MAINNET позволяет использовать genesis-блок, консенсус и настройки сети Mainnet. При выборе значения MAINNET в конфигурационном файле ноды, которая подключается к сети Mainnet, не нужно указывать параметры блоков custom, genesis и consensus.
- consensus.type - тип *консенсуса* в блокчейне. Возможные значения pos или poa. Вы можете подробно почитать о настройке консенсуса *здесь*.

Блок fees

- enabled - опция использования комиссий за выпуск *транзакций*. Возможные значения false или true.

Блок custom

- address-scheme-character - байт сети, для «Waves Enterprise Mainnet» - V, для «Waves Enterprise Partnernet» - P. Данный параметр используется для предотвращения конфликта адресов из разных сетей. Для сайдчейна или для тестовых версий блокчейн-платформы Waves Enterprise можно использовать любые буквы. Ноды в одной блокчейн-сети должны иметь одинаковый байт сети.
- functionality - блок настройки основных параметров блокчейна.
- genesis - блок настройки параметров генезис-блока.

Блок functionality

- feature-check-blocks-period - количество блоков, через которые выполняется проверка и активация опций блокчейна.
- blocks-for-feature-activation - количество блоков, через которые применяется активированная опция.
- pre-activated-features - набор опций блокчейна.

Блок genesis

- `average-block-delay` - средняя задержка создания блоков. Данный параметр используется для консенсуса *PoS*.
- `initial-base-target` - начальное базовое число для регулирования процесса майнинга. Данный параметр используется для консенсуса *PoS*. От значения параметра зависит частота формирования блоков - чем больше значение, тем чаще создаются блоки. Также величина баланса майнера влияет на использование данного параметра в майнинге - чем больше баланс майнера, тем меньше становится значение `initial-base-target` для выбора ноды-майнера. При выставлении значения данному параметру рекомендуется учитывать комбинацию балансов майнеров и ожидаемый интервал между блоками.
- `block-timestamp` - числовой код даты и времени. Время указывается в миллисекундах, значение должно состоять из 13 цифр. Если вы берёте стандартное значение `timpstamp`, состоящее из 10 цифр, то в конце необходимо добавить три любые цифры.
- `initial-balance` - начальный баланс сети. Значение этого параметра влияет на процесс майнинга при *PoS* консенсусе. Чем больше баланс майнера, тем меньше становится значение `initial-base-target` для определения ноды-майнера текущего раунда.
- `genesis-public-key-base-58` - хеш публичного ключа генезис-блока, зашифрованный в Base58.
- `signature` - подпись генезис-блока, зашифрованная в Base58.
- `transactions` - список участников сети с первоначальным балансом, создание которых войдёт в генезис-блок в виде генезис-транзакций.
- `network-participants` - список сетевых участников с ролями, создание которых войдёт в генезис-блок в виде генезис-транзакций.

17.3.5 Секция `network`

- `bind-address` - сетевой адрес ноды.
- `port` - номер порта.
- `known-peers` - список известных сетевых адресов нод. Этот параметр должен быть обязательно заполнен. Список адресов передаётся пользователю администратором сети до подключения новой ноды.
- `declared-address` - сетевой адрес ноды вместе с номером порта для процедуры handshake.
- `max-simultaneous-connections` - максимальное количество одновременно поддерживаемых соединений. Параметр ограничен количеством нод в блокчейне, т.е. максимальное количество одновременных соединений будет не больше числа нод в сети.
- `peers-request-interval` - интервал запроса списка пиров. Значение указывается в секундах или минутах. Рекомендуемое значение - 1-2 минуты.

17.3.6 Секция `wallet`

- `file` - директория для хранения приватных ключей.
- `password` - пароль для доступа к файлу с приватными ключами.

17.3.7 Секция `miner`

- `enable` - активация опции майнинга.
- `quorum` - необходимое количество нод-майнеров для создания блока. Значение `0` позволит генерировать блоки оффлайн. При указании значения необходимо учитывать, что собственная нода-майнер не суммируется со значением этого параметра, т.е., если вы указываете `quorum = 2`, то для майнинга нужно минимум 3 ноды-майнера.
- `interval-after-last-block-then-generation-is-allowed` - создание блока только в том случае, если последний блок не старше указанного периода времени.
- `micro-block-interval` - интервал между микроблоками.
- `min-micro-block-age` - минимальный возраст микроблока.
- `max-transactions-in-micro-block` - максимальное количество транзакций в микроблоке.
- `minimal-block-generation-offset` - минимальный временной интервал между блоками.

17.3.8 Секция `features`

- `supported` - список поддерживаемых опций.

17.4 Создание аккаунтов

Аккаунт ноды включает в себя адрес и ключевую пару - публичный и приватный ключи. Адрес и публичный ключ демонстрируются пользователю во время создания аккаунта в командной строке. Приватный ключ ноды записывается в хранилище ключей `keystore.dat`.

17.4.1 Генерирование ключевых пар

Адрес ноды и ключевая пара будущих участников сети создаются при помощи утилиты генератора. Получить последнюю версию генератора можно на странице проекта на GitHub. Для запуска утилиты необходимо в качестве одного из параметров указать файл `accounts.conf`, в котором определяются параметры генерации ключей. При создании ключевой пары придумайте и введите пароль, сохраните его для последующей конфигурации. Данный пароль будет использоваться при создании глобальной переменной `WE_NODE_OWNER_PASSWORD` далее. Если не хотите устанавливать пароль от ключевой пары ноды, нажмите `enter`. Команда для запуска генератора:

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

17.4.2 Глобальные переменные

Для обеспечения дополнительной безопасности рекомендуется использовать пароль для ключевой пары ноды. Платформа Waves Enterprise поддерживает два способа использования пароля:

1. Ввод пароля в ручном режиме при каждом старте ноды.
2. Создание глобальных переменных в операционной системе.

При использовании ручного ввода пароля создавать глобальные переменные необязательно. Если вы планируете разворачивать ноду в контейнерах или подобных сервисах, то удобнее будет задать в ОС следующие глобальные переменные:

1. `WE_NODE_OWNER_PASSWORD` - пароль от ключевой пары ноды, который вводится на этапе создания этой ключевой пары.
2. `WE_NODE_OWNER_PASSWORD_EMPTY` - `true` или `false`, установите значение `true`, если не хотите устанавливать пароль на ключевую пару ноды, в таком случае создавать переменную `WE_NODE_OWNER_PASSWORD` необходимости нет. Если вы используете пароль, установите значение `false` и пропишите в переменную `WE_NODE_OWNER_PASSWORD` пароль от ключевой пары ноды.

17.5 Подпись genesis блока

Подпишите genesis-блок утилитой generators-x.x.x.jar. Команда для подписания: `java -jar generators-x.x.x.jar GenesisBlockGenerator node.conf`, где `node.conf` это отредактированный в этом *пункте* конфигурационный файл ноды. После подписания поля `genesis-public-key-base-58` и `signature` конфигурационного файла будут заполнены значениями открытого ключа и подписи genesis-блока.

Пример:

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

17.6 Настройка консенсуса

Блокчейн-платформа Waves Enterprise поддерживает два типа консенсуса - *PoS* и *PoA*. Настройки консенсуса располагаются в секции *blockchain*.

17.6.1 Настройка PoS

Если вы не указали тип консенсуса в поле `consensus.type` секции *blockchain*, то консенсус PoS будет использоваться по умолчанию. За работу майнинга с консенсусом PoS отвечают следующие параметры, расположенные в блоке `genesis` секции *blockchain*:

- `average-block-delay` - средняя задержка создания блоков. Значение по умолчанию - 60 секунд. Значение этого параметра игнорируется, если выбран консенсус PoA.
- `initial-base-target` - начальное базовое число для регулирования процесса майнинга. От значения параметра зависит частота формирования блоков - чем выше значение, тем чаще создаются блоки. Также величина баланса майнера влияет на использование данного параметра в майнинге - чем больше баланс майнера, тем меньше становится значение `initial-base-target` при расчёте очереди ноды-майнера в текущем раунде.

- `initial-balance` - начальный баланс сети. Чем больше доля баланса майнера от изначального баланса сети, тем меньше становится значение `initial-base-target` для определения ноды-майнера текущего раунда.

Для демонстрационной работы с нодой мы рекомендуем использовать значения, которые установлены по умолчанию в примерах конфигурационных файлов, приведенных на странице проекта на GitHub.

17.6.2 Настройка PoA

Для использования консенсуса *PoA* раскомментируйте или добавьте блок `consensus` в секции *blockchain*:

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

Приведенные в блоке `consensus` параметры используются только для консенсуса *PoA*.

- `type` - тип консенсуса. Возможные значения `pos` или `poa`. При указании значения `pos` другие параметры блока учитываться не будут.
- `round-duration` - длина раунда майнинга блока в секундах.
- `sync-duration` - период синхронизации майнинга блока в секундах. Полное время раунда складывается из суммы `round-duration` и `sync-duration`.
- `ban-duration-blocks` - количество блоков, на которые нода-майнер попадает в бан.
- `warnings-for-ban` - количество раундов, которое нода-майнер предупреждается о попадании в бан.
- `max-bans-percentage` - процент нод-майнеров от общего числа нод в сети, который может быть помещён в бан.

Использование консенсуса *PoA* позволяет регулировать очередность создания блоков путем ограничения функции майнинга для определенных нод. Это делается для того, чтобы распределить равномерно нагрузку в сети, если какие-либо ноды-майнеры вышли из сети или стали неактивными. Нода-майнер может попасть в бан по следующим причинам:

- если нода пропустит свою очередь майнинга;
- если нода предоставит невалидный блок;
- если нода вышла из сети.

Перед попаданием в `blacklist` нода-майнер получает предупреждения о попадании в бан на такое количество раундов, какое указано в параметре `warnings-for-ban`. После окончания количества блоков, определенного в параметре `ban-duration-blocks`, нода-майнер получает возможность участвовать в создании блоков.

17.6.3 Настройка консенсуса в секции `miner`

При настройке параметров консенсуса необходимо учитывать следующие параметры секции *miner*:

- `micro-block-interval` - интервал между микроблоками. Значение указывается в секундах.
- `min-micro-block-age` - минимальный возраст микроблока. Значение указывается в секундах и не должно превышать значения параметра `micro-block-interval`.
- `minimal-block-generation-offset` - минимальный временной интервал между блоками. Значение указывается в миллисекундах.

Значения параметров создания микроблоков не должны конфликтовать со значениями параметров `average-block-delay` для PoS и `round-duration` для PoA. Количество микроблоков в блоке не ограничено, но зависит от размера транзакций, попавших в микроблок.

17.7 Настройка Docker

Установка и исполнение docker-контрактов задается секцией `docker-engine` в *конфигурационном файле ноды*.

```
  # 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 {
    # gRPC contract startup timeout
    startup-timeout = 10s
    # 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
  allow-net-access = no
  # Remote registries auth information
  remote-registries = [
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    {
      domain = "myregistry.com:5000"
      username = "user"
      password = "password"
    }
    ]
  # 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
  }
  remove-container-on-fail = yes
  grpc-server {
    # host = "192.168.65.2"
    port = 6865
    akka-http-settings {
      akka {
        http.server.idle-timeout = infinite
        http.client.idle-timeout = infinite
        http.host-connection-pool.idle-timeout = infinite
        http.host-connection-pool.client.idle-timeout = infinite
      }
    }
  }
}
```

Параметры:

- `enable` - включение обработки транзакций для docker-контрактов (yes/no).
- `docker-auth` - секция базовой авторизации по логину и паролю.
- `docker-host` - URL-адрес Docker-хоста.
- `node-rest-api` - URL-адрес к REST API ноды при использовании Docker-хоста.
- `integration-tests-mode-enable` - режим тестирования Docker-контрактов (yes/no).
- `execution-limits` - секция ограничений на выполнение Docker-контрактов:
  - `startup-timeout` - таймаут на создание контейнера gRPC контракта и его регистрацию в ноде (в секундах);
  - `timeout` - таймаут на выполнение контракта (в секундах);
  - `memory` - ограничение по памяти для контейнера контракта (в Мб);
  - `memory-swap` - файл подкачки памяти для контракта (в Мб).
- `reuse-containers` - повторное использование существующего Docker-контракта.
- `remove-container-after` - обязательность удаления контейнера после исполнения (yes/no).

- `allow-net-access` - доступ к сети (yes/no).
- `remote-registries` - адреса удаленных репозиториев и настройки авторизации к ним.
- `check-registry-auth-on-startup` - обязательность проверки авторизации к репозиториям при запуске ноды (yes/no).
- `contract-auth-expires-in` - таймаут токена авторизации для Docker-контракта.
- `contract-execution-messages-cache` - секция настройки кеш-памяти Docker-контракта. При достижении лимита буфера нода обработает все сообщения в пакетном режиме:
  - `expire-after` - время жизни сообщений в кеш-памяти в минутах;
  - `max-buffer-size` - максимальное количество сообщений в буфере;
  - `max-buffer-time` - максимальное время для буфера в миллисекундах.
- `remove-container-on-fail` – удалять ли контейнер, если при его старте произошла ошибка. Этот параметр может быть полезен при поиске ошибок при работе с контрактами (yes/no).

gRPC сервер

Секция настроек gRPC сервера для работы смарт контрактов с gRPC API.

- `host` – сетевой адрес ноды (опциональный параметр).
- `port` – порт gRPC сервера.
- `akka-http-settings` – секция настроек Akka HTTP фреймворка, используемого для gRPC сервера.

17.8 Настройка типа авторизации для доступа к REST API ноды

Блокчейн-платформа Waves Enterprise поддерживает следующие два типа авторизации для доступа к REST API ноды:

- авторизация по хешу ключевой строки `api-key`;
- авторизация через сервис авторизации.

Тип авторизации указывается в секции настройки REST API `rest-api` конфигурационного файла ноды. Авторизация по хешу ключевой строки `api-key` является простым средством управления доступом к ноде с низким уровнем безопасности. В случае попадания хеша ключевой строки `api-key` к злоумышленнику, тот получает полный доступ к ноде. Применение авторизации с использованием отдельного сервиса авторизации, где доступ к ноде предоставляется по специальному токену, повышает безопасность блокчейн сети до высокого уровня. Подробнее о сервисе авторизации можно почитать в разделе *Сервис авторизации*.

17.8.1 Секция `rest-api` конфигурационного файла ноды

Секция `rest-api` позволяет привязать сетевой адрес ноды к REST API, выбрать и настроить тип авторизации, а также указать ограничения для некоторых методов REST API.

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

(continues on next page)

(продолжение с предыдущей страницы)

```
# 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"
# }

# Enable/disable CORS support
cors = yes

# 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
}
```

Описание параметров

- `enable` - активация опции REST API на ноде.
- `bind-address` - сетевой адрес ноды для привязки REST API.
- `port` - порт прослушивания REST API запросов.
- `cors` - поддержка кросс-доменных запросов к REST API.
- `transactions-by-address-limit` - максимальное количество транзакций, возвращаемых методом `/transactions/address/{address}/limit/{limit}`.
- `distribution-address-limit` - максимальное количество адресов, указываемых в поле `limit` и возвращаемых методом `GET /assets/{assetId}/distribution/{height}/limit/{limit}`.

Секция `auth` для типа `api-key`

- `auth-type` - тип авторизации, установите значение `api-key` - авторизация по хешу ключевой строки.
- `api-key-hash`- хеш от ключевой строки доступа к REST API.
- `privacy-api-key-hash` - хеш от ключевой строки доступа к методам `privacy`.

Секция `auth` для типа `oauth2`

- `auth-type` - тип авторизации, установите значение `oauth2` - авторизация по токену.
- `public-key` - публичный ключ сервиса авторизации.

17.8.2 Использование авторизации по хешу ключевой строки

В параметре `auth-type` установите значение `api-key`. Используя утилиту generators-x.x.x.jar, создайте `api-key-hash` для доступа к REST API ноды. Для запуска утилиты требуется в качестве одного из параметров указать *файл api-key-hash.conf*, в котором определяются параметры создания `api-key-hash`. Команда для запуска утилиты:

```
java -jar generators-x.x.x.jar ApiKeyHash api-key-hash.conf
```

Полученное в результате исполнения утилиты значение укажите в параметре `api-key-hash` конфигурационного файла ноды.

Для доступа к методам *privacy* создайте `privacy-api-key-hash` аналогичным методом, как и `api-key-hash`, описанным выше. Полученное значение укажите в параметре `privacy-api-key-hash` конфигурационного файла ноды.

17.8.3 Использование авторизации по токену

В параметре `auth-type` установите значение `oauth2`, в параметре `public-key` укажите публичный ключ сервиса авторизации.

17.9 Настройка анкоринга

Если используете опцию *анкоринга*, необходимо настроить блок `anchoring`. `targetnet` в данном случае - блокчейн-сеть, в которую нода из сайдчейна будет выполнять анкоринг-транзакции.

```
anchoring {
enable = yes
height-range = 50
height-above = 10
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 = "5M7C14rf3TAaWHvU6Kqo97iscd8fJFpvFwyQ3Q6vfztS"
# privacy-api-key-hash = "5M7C14rf3TAaWHvU6Kqo97iscd8fJFpvFwyQ3Q6vfztS"
}

targetnet-scheme-byte = "K"
targetnet-node-address = "http://node.weservices.com:6862/NodeAddress"
targetnet-node-recipient-address = "3JWveBpXS1EcDpxcoAwVNAjFfUMrxaALgZt"
targetnet-private-key-password = ""
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  wallet {
  file = "node-1_mainnet-wallet.dat"
  password = "small"
   }

 targetnet-fee = 500000
 sidechain-fee = 500000
}
```

Параметры анкоринга

- `height-range` - число блоков, через которое нода приватного блокчейна отправляет в Targetnet транзакции для анкоринга.
- `height-above` - число блоков в Targetnet, через которое нода приватного блокчейна создаёт подтверждающую анкоринг транзакцию с данными первой транзакции. Рекомендуется устанавливать значение, не превышающее максимальную величину отката в Targetnet `max-rollback`.
- `threshold` - число блоков, которое отнимается от текущей высоты приватного блокчейна. В транзакцию для анкоринга, отправляемую в Targetnet, попадёт информация из блока на высоте `current-height - threshold`. Если устанавливается значение 0, то берётся информация из текущего блока. Рекомендуется устанавливать значение, близкое к максимальной величине отката в приватном блокчейне `max-rollback`.

В зависимости от настроек майнинга в сети `Targetnet` расстояние между транзакциями анкоринга может меняться. Установленное значение `height-range` задаёт приблизительный интервал между транзакциями анкоринга. Реальное время попадания транзакций анкоринга в смайненный блок сети `Targetnet` может превышать время, потраченное на майнинг количества блоков `height-range` в сети `Targetnet`.

Параметры авторизации при использовании анкоринга

- `type` - тип авторизации при использовании анкоринга. `api-key` - авторизация по `api-key-hash`, `auth-service` - авторизация по специальному токену.

В случае выбора авторизации по `api-key-hash` достаточно указать значение ключа в параметре `api-key` ниже. Если вы выбираете авторизацию по токену, необходимо указать `type = "auth-service"`, раскомментировать параметры ниже и установить для них значения:

- `authorization-token` - постоянный авторизационный токен.
- `authorization-service-url` - URL-адрес сервиса авторизации.
- `token-update-interval` - интервал обновления авторизационного токена.

Параметры для доступа Targetnet

Для ноды, которая будет отправлять транзакции анкоринга в Targetnet, генерируется отдельный файл `keystore.dat` с ключевой парой для доступа в Targetnet.

- `targetnet-scheme-byte` - байт сети Targetnet.
- `targetnet-node-address` - полный сетевой адрес ноды вместе с номером порта в сети Targetnet, на который будут отправляться транзакции для анкоринга. Адрес необходимо указывать вместе с типом соединения (http/https), номером порта и параметром `NodeAddress` как в примере `http://node.weservices.com:6862/NodeAddress`.
- `targetnet-node-recipient-address` - адрес ноды в сети Targetnet, на который будут записываться транзакции для анкоринга, подписанные ключевой парой данного адреса.

- `targetnet-private-key-password` - пароль от приватного ключа ноды для подписи анкоринг-транзакций.

Сетевой адрес и порт для анкоринга в сеть Targetnet/Partnernet можно получить у сотрудников технической поддержки Waves Enterprise. Если используется несколько приватных блокчейнов с взаимным анкорингом, то необходимо использовать соответствующие сетевые настройки частных сетей.

Параметры файла с ключевой парой для подписания транзакций анкоринга в Targetnet, секция wallet

- `file` - имя файла и путь до каталога хранения файла с ключевой парой для подписания транзакций анкоринга в сети Targetnet. Файл находится на ноде приватной сети.
- `password` - пароль от файла с ключевой парой.

Параметры комиссий

- `targetnet-fee` - плата за выпуск транзакции для анкоринга в сети Targetnet.
- `sidechain-fee` - плата за выпуск транзакции в приватном блокчейне.

17.10 Настройка групп доступа к конфиденциальным данным

При использовании методов *privacy* активируйте функциональность и заполните блок `storage` параметрами настройки БД для хранения конфиденциальных данных:

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

Описание параметров

- `enabled` - активация опции;
- `url` - адрес БД PostgreSQL;
- `driver` - имя драйвера JDBC;
- `profile` - имя профиля для доступа к JDBC;
- `user` - имя пользователя для доступа к БД;
- `password` - пароль для доступа к БД;
- `connectionPool` - имя пула соединений, по умолчанию `HikariCP`;
- `connectionTimeout` - таймаут для соединения;

- `connectionTestQuery` - имя тестового запроса;
- `queueSize` - размер очереди запросов;
- `numThreads` - количество одновременных подключений;
- `schema` - схема взаимодействия;
- `migration-dir` - директория для миграции данных.

Для хранения конфиденциальных данных используется БД PostgreSQL. База данных устанавливается на машину с нодой, также создается аккаунт доступа к БД. Вы можете воспользоваться справочной документацией на PostgreSQL для скачивания и установки версии, которая соответствует вашему типу операционной системы.

Во время установки БД система предложит создать аккаунт для доступа к БД. Эти логин и пароль для доступа необходимо внести в соответствующие параметры `user/password`.

Внесите URL подключения к PostgreSQL в параметр `url`. В URL входят следующие параметры:

- POSTGRES_ADDRESS - адрес хоста PostgreSQL;
- POSTGRES_PORT - номер порта хоста PostgreSQL;
- POSTGRES_DB - наименование БД PostgreSQL.

Можно указывать URL до БД PostgreSQL в одной строке с данными аккаунта. Пример приведен ниже, где `user=user_privacy_node_0@we-dev` это логин пользователя, `password=7nZL7Jr41qOWUHz5qKdypA&sslmode=require` - пароль с опцией требования его ввода при авторизации.

Пример

```
privacy.storage.url = "jdbc:postgresql://vostk-dev.postgres.database.azure.com:5432/
↪privacy_node_0?user=user_privacy_node_0@we-dev&password=7nZL7Jr41qOWUHz5qKdypA&
↪sslmode=require"
```

На странице GitHub Waves Enterprise вы можете получить примеры конфигурационных файлов и дистрибутивы последнего релиза платформы.

ГЛАВА 18

Получение лицензии

Блокчейн-платформа Waves Enterprise - коммерческая и рассчитана в первую очередь на применение в крупных компаниях и государственном секторе. Для использования технологии необходимо получить лицензию на платформу. Быстрый и удобный доступ к списку лицензий обеспечивается сервисом управления лицензиями.

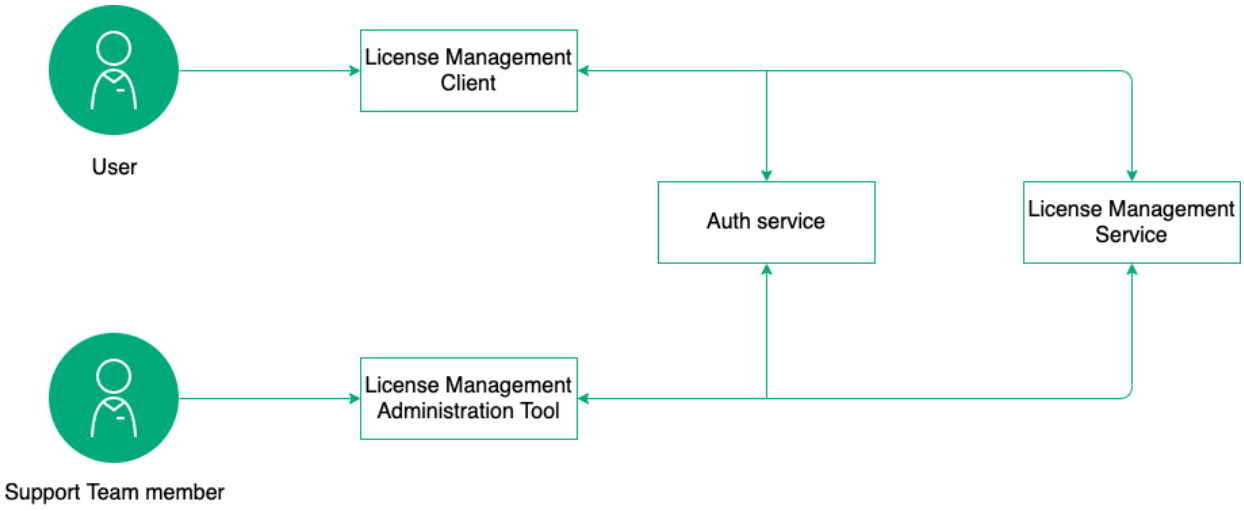


Рис. 1: Схема получения лицензии блокчейн-платформы Waves Enterprise

Для ознакомления с возможностями платформы лицензия на продукт не требуется. Платформа сохраняет полную функциональность до достижения высоты блокчейна равной **30 000** блоков, что при времени раунда блока равным 30 секундам составляет 10 дней работы без ограничений.

Пользователям блокчейн-платформы Waves Enterprise предлагаются к использованию следующие типы лицензии:

- **Коммерческая лицензия** - позволяет использовать платформу для реализации коммерческих проектов. Выдается на срок, определяемый договорными отношениями с партнёром.

- **Некоммерческая лицензия** - позволяет использовать платформу для реализации некоммерческих проектов. Выдается на срок, определяемый договорными отношениями с партнёром.
- **Пробная лицензия** - позволяет ознакомиться с платформой и технологией. Выдается на срок пилотного проекта по договору, либо на время разработки и отладки продукта.
- **Лицензия для работы в сети Mainnet** - специальная лицензия, позволяющая запустить ноду в сети *Mainnet*. Для работы в сети необходимо иметь на балансе или в лизинге не менее **50 000 WEST**. При снижении указанного остатка вводятся ограничения на формирование блоков и доступ к API ноды. Отправка заявки на регистрацию новых участников осуществляется в системе Service Desk.

Внимание: Одна лицензия распространяется на одну ноду!

Чтобы оформить запрос на лицензию, выполните следующие действия:

1. Перейдите в сервис управления лицензиями и создайте новую учётную запись, если она не была создана ранее.
2. Отправьте заявку на получение лицензии в службу поддержки Waves Enterprise. Представитель службы поддержки свяжется с вами, согласует детали, создаст профиль компании и привяжет к нему созданную учётную запись.
3. После активируйте лицензию указав адрес вашей ноды (*node_owner_address*).
4. Указанный файл лицензии в виде JSON отправьте в запросе *POST /licenses/upload* к ноде.
5. Для просмотра состояния лицензии воспользуйтесь запросом *GET /licenses/status*.

ГЛАВА 19

Подключение к Mainnet и Partnernet

19.1 Работа в сети «Waves Enterprise Mainnet»

19.1.1 Подключение ноды в сеть «Waves Enterprise Mainnet»

Предупреждение: При подключении своей ноды к сети «Waves Enterprise Mainnet» и майнинге на балансе аккаунта должно быть не менее **50 000 WEST**!

Для подключения своей ноды к сети «Waves Enterprise Mainnet» выполните следующие действия:

1. Зайдите на сайт Waves Enterprise и создайте учётную запись, следуя подсказкам веб-интерфейса.
2. Переведите токены в сеть «Waves Enterprise Mainnet».
3. Передайте токены в аренду в любом количестве на адрес `3NrKDuHjUG7vSCiMMD259msBKcPRm4MvaJu` и сохраните идентификатор этой транзакции. В дальнейшем вы можете отозвать токены из аренды, поскольку эта операция нужна для верификации владения вами данным адресом и балансом.
4. *Разверните* одну ноду.
5. Выполните *ручную конфигурацию* ноды. Пример конфигурационного файла ноды можно взять на странице проекта на GitHub. Для добавления ноды в сеть «Mainnet» название конфигурационного файла - `mainnet.conf`. В конфигурационном файле ноды `mainnet.conf` заполните только те поля, где в качестве значений указано слово /**FILL**/.
6. Зайдите на сайт службы поддержки Waves Enterprise и зарегистрируйтесь.
7. Выберите тип заявки «Подключение участника» для юридического или физического лица.
8. Заполните все необходимые поля формы. Если вы хотите майнить, отметьте поле **Прошу предоставить права майнинга**.
9. В поле **Подтверждение владения токенами WEST** введите идентификатор транзакции передачи токенов в аренду.

10. Дождитесь рассмотрения заявки на подключение. После успешной регистрации можете начинать работу в сети «Waves Enterprise Mainnet».
11. После получения разрешения на подключение к сети «Waves Enterprise Mainnet» *запустите* ноду, публичной ключ которой вы указали в заявке.
12. Для выполнения майнинга и работы в сети переведите или передайте в аренду токены на адрес подключённой ноды.

19.1.2 Комиссии в сети «Waves Enterprise Mainnet»

№	Тип транзакции	Комиссия	Описание
1	*Genesis transaction*	отсутствует	Первоначальная привязка баланса к адресам, создаваемым при старте блокчейна нод
3	*Issue Transaction*	1 WEST	Выпуск токенов
4	*Transfer Transaction*	0.1 WEST	Перевод токенов
5	*Reissue Transaction*	1 WEST	Перевыпуск токенов
6	*Burn Transaction*	1 WEST	Сжигание токенов
8	*Lease Transaction*	0.1 WEST	Передача токенов в аренду
9	*Lease Cancel Transaction*	0.1 WEST	Отмена аренды токенов
10	*Create Alias Transaction*	1 WEST	Создание псевдонима
11	*MassTransfer Transaction*	0.1 WEST	Массовый перевод токенов. Указана минимальная комиссия
12	*Data Transaction*	0.1 WEST	Транзакция с данными в виде полей с парой ключ-значение. Указана минимальная комиссия
13	*SetScript Transaction*	0.5 WEST	Транзакция, привязывающая скрипт с RIDE-контрактом к аккаунту
14	*Sponsorship*	1 WEST	Транзакция, подписывающая спонсорский ассет
15	*SetAssetScript*	1 WEST	Транзакция, привязывающая скрипт с RIDE-контрактом к ассету
101	*Genesis Permission Transaction*	отсутствует	Назначение первого администратора сети для дальнейшей раздачи прав
102	*Permission Transaction*	0.05 WEST	Выдача/отзыв прав у аккаунта
103	*CreateContract Transaction*	1 WEST	Создание Docker-контракта
104	*CallContract Transaction*	0.1 WEST	Вызов Docker-контракта
105	*ExecutedContract Transaction*	отсутствует	Выполнение Docker-контракта
106	*DisableContract Transaction*	0.05 WEST	Отключение Docker-контракта
107	*UpdateContract Transaction*	1 WEST	Обновление Docker-контракта
110	*GenesisRegisterNode Transaction*	отсутствует	Регистрация ноды в генезис-блоке при старте блокчейна
111	*RegisterNode Transaction*	0.05 WEST	Регистрация новой ноды в сети
112	*CreatePolicy Transaction*	1 WEST	Создание группы доступа к конфиденциальным данным
113	*UpdatePolicy Transaction*	0.5 WEST	Изменение группы доступа
114	*PolicyDataHash Transaction*	0.1 WEST	Отправка в сеть хеша данных

19.1.3 Примеры конфигурационных файлов для сети «Waves Enterprise Mainnet»

Описание полной конфигурации ноды вы можете почитать *здесь*.

Пример конфигурационного файла accounts.conf

```
// 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/utils/reload-wallet"
  }
}
```

Параметр `chain-id` содержит идентификационный байт сети, для сети «Waves Enterprise Mainnet» это значение `V`.

Пример конфигурационного файла api-key-hash

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

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

Пример конфигурационного файла ноды

```
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 {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
#    storage {
#      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
#      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: 100000000000000000
    genesis-public-key-base-58: "D7tDsKd7DQ7H9m6fPRyk1GsNQxjAQXsETtuVgqSaaXDs"
    signature:
↪"P7kwe3dWSWgUYL8FZu5kccPfPzoxGgLuKjTCkeapTxoDbdpo6EtcqndXoSjqKUUVS67xXfogGmaNroLgNocWcBg
↪"
    transactions = [
      {recipient: "3Nnq14SGqeYETSd1SJ6z8LsgBRYB2ya1yRC", amount: 99990000000000000}
      {recipient: "3Nrysx7J1TN6vB1eYdHgug2nfxA7um918zy", amount: 1000000000000},
      {recipient: "3NuiCzDhmeSKL5QFa5sqZzzm9zTL4max4fZ", amount: 1500000000000},
      {recipient: "3NqaDWdEgGsqJj1HjznDQMtk6v5KVxmRceg", amount: 2000000000000},
      {recipient: "3Nckru7f8Y8vS3PXGyy5iwoheRrKvqW5u8x", amount: 2500000000000},
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      {recipient: "3NmHrYoC8S2SUosy6UJp47bBwq2Cr2X6Yq1", amount: 3000000000000}

    ]
    network-participants = [
      {public-key: "GasRtAUXMhifrUUmgU66rRZPii68tE4QxdQmtCcrV3xL", roles: [permissioner,
↪ connection_manager]},
      {public-key: "Er29kgV3yeumEAtPxBAk5fXPERYYa1wmAcPgzWw4mxHi", roles: [miner]},
      {public-key: "9eoVBycnr2m8bgu1WvYySoFJ1QqFLPAMzhnmErp291f6", roles: [miner]},
      {public-key: "9ngXJ3d1XSQgXcYbgZm2wH4QHS8CTc5mtf9M4XDoz5db", 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
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# 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-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
```

(continues on next page)

(продолжение с предыдущей страницы)

```
docker-engine {
enable = no
execution-limits {
  timeout = 10s
  memory = 512
  memory-swap = 512
}
grpc-server {
  # gRPC server port
  port = 6865
  # Optional node host
  # host = "192.168.65.2"
}
}
}
```

19.2 Работа в сети «Waves Enterprise Partnernet»

19.2.1 Подключение ноды в сеть «Waves Enterprise Partnernet»

Выполните следующие действия для подключения ноды в сеть «Waves Enterprise Partnernet»:

1. *Разверните* одну ноду.
2. Перед запуском генератора *создайте* для него отдельный конфигурационный файл `accounts.conf`.
3. Скачайте актуальный релиз ноды и генератора в jar-формате.
4. *Сгенерируйте* ключевую пару для подключаемой ноды при помощи генератора. Для удобства рекомендуется создавать пару для одной ноды, укажите в конфигурационном файле `accounts.conf` количество 1 в поле `amount`. При создании ключевой пары введите пароль от адреса ноды и сохраните его для последующей конфигурации. Если не хотите устанавливать пароль от адреса ноды, нажмите `enter`.
5. Создайте конфигурационный файл ноды, взяв шаблон с сайта проекта на GitHub. Заполните все поля, помеченные строкой **#FILL**. Если нода будет майнить, укажите значение `yes` параметра `enable` блока `miner` и запросите права майнера в заявке на подключение. В противном случае укажите значение `no`. Также укажите в качестве параметра `url` блока `privacy {storage {}}` адрес к БД PostgreSQL.
6. Если не хотите вводить пароль от адреса ноды при её старте в ручном режиме, создайте *глобальные переменные* `WE_NODE_OWNER_PASSWORD` и `WE_NODE_OWNER_PASSWORD_EMPTY` в вашей ОС.
7. Зайдите на сайт службы поддержки Waves Enterprise и зарегистрируйтесь.
8. Выберите тип заявки «Подключение участника» для юридического или физического лица.
9. Заполните все необходимые поля формы. Если вы хотите майнить, отметьте поле **Прошу предоставить права майнинга**.
10. Дождитесь рассмотрения заявки на подключение. После успешной регистрации можете начинать работу в сети «Waves Enterprise Partnernet».
11. После получения разрешения на подключение к сети «Waves Enterprise Partnernet» *запустите* ноду.

19.2.2 Примеры конфигурационных файлов для сети «Waves Enterprise Partnernet»

Описание полной конфигурации ноды вы можете почитать *здесь*.

Пример конфигурационного файла accounts.conf

```
// 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"
  }
}
```

Параметр `chain-id` содержит идентификационный байт сети, для сети «Waves Enterprise Partnernet» это значение `P`. Если хотите использовать ГОСТ-криптографию, установите значение `no` параметра `waves-crypto` во всех конфигурационных файлах. Также перед конфигурацией ноды установите CryptoPro JCP 2.0.40035. Полная инструкция по установке приведена *здесь*.

Пример конфигурационного файла api-key-hash

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

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

Пример конфигурационного файла ноды

```
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
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
genesis {
  average-block-delay: 60s
  initial-base-target: 153722867
  block-timestamp: 1559260800000
  initial-balance: 1625000000000000
  genesis-public-key-base-58: "8RbU8qKWWxLuVk49LgeE39y83LUTVp1zHEJwMM7zKaMC"
  signature:
↪"2dKzduxL9bdWz1B9wBPnGALfowrPDSidEoGAQEoRogGuBB4sQanCr4JySXvWoAmpu1EmcU8MsCQTL3TaSMnFxG2U
↪"
  transactions = [
    { recipient: "3LWg4n6VmN6DKBSwGF1hwnaCzXdjMkQCFrn", amount: 1250000000000000 },
    { recipient: "3LPPZNhakdm9ZPiGShNvWGCshFqsQXFjUQ1", amount: 300000000000000 },
    { recipient: "3LEpXfh7XmCRias92swo6LUJqyo9MA7SaFc", amount: 75000000000000 }
  ]
  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
```

(continues on next page)

(продолжение с предыдущей страницы)

```
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
}
# 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
grpc-server {
# gRPC server port
  port = 6865
  # Optional node host
  # host = "192.168.65.2"
}
}

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"
}
}
}
```

ГЛАВА 20

REST API

Блокчейн-платформа Waves Enterprise предоставляет возможность взаимодействия с блокчейном как в части получения данных (транзакции, блоки, балансы и др.), так и в части записи информации в блокчейн (подписание и отправка транзакций) при помощи RESTful API ноды. REST API позволяет пользователям удалённо взаимодействовать с нодой через запросы и ответы в формате JSON. Работа с API происходит по протоколу HTTPS. Удобным интерфейсом к API служит известный фреймворк Swagger.

20.1 Методы REST API ноды

Полное описание REST API вы можете найти на странице Документация API. Почти все методы REST API закрыты *авторизацией*. Если метод открыт, то вы увидите значок .

20.1.1 Activation

Подсказка: Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

GET /activation/status

Возвращает статус активации нового функционала в ноде/нодах.

Ответ метода:


```
{"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 },
    {"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",
```


(continues on next page)

(продолжение с предыдущей страницы)


```
    "blockchainStatus": "ACTIVATED",
    "nodeStatus": "IMPLEMENTED",
    "activationHeight": 0 } ]
}
```


20.1.2 Addresses

Подсказка: Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

GET /addresses/info/{address}

Получение публичного ключа по адресу. Метод возвращает только те публичные ключи, которые хранятся в файле ноды `keystore.dat`.

Ответ метода:


```
{
  "address": "3JFR1pmL6biTzr9oa63gJcjZ8ih429KD3aF",
  "publicKey": "EPxkVA9iQejsjQikovyxkkY8iHnbXsR3wjgkgE7ZW1Tt"
}
```


GET /addresses

Получение всех адресов участников, ключевые пары которых хранятся в keystore ноды.

Ответ метода:


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


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

Получение всех адресов участников, ключевые пары которых хранятся в keystore ноды в заданном диапазоне.

Ответ метода:


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

GET /addresses/balance/{address}

Получение баланса для адреса {address}.

Ответ метода:

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

POST /addresses/balance/details

Получение балансов для списка адресов.

Запрос метода:

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

GET /addresses/effectiveBalance/{address}/{confirmations}

Получение баланса для адреса {address} после количества подтверждений >= значению {confirmations}. Возвращается общий баланс участника, включая средства переданные участнику за лизинг.

Ответ метода:

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

GET /addresses/effectiveBalance/{address}

Возвращает общий баланс аккаунта.

Ответ метода

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

GET /addresses/balance/details/{address}

Возвращает подробные сведения о балансе адресата {address}.

Запрос метода:

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

Ответ метода:

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

Параметры ответа

- Regular — общий баланс участника, включая средства переданные в лизинг
- Available — общий баланс участника, за исключением средств переданных в лизинг
- Effective — общий баланс участника, включая средства переданные участнику за лизинг (Available + средства переданные вам в лизинг)
- Generating — минимальный баланс участинка, включая средства переденные участнику за лизинг, за последние 1000 блоков (используется для майнинга)

GET /addresses/scriptInfo/{address}

Получение данных об установленном скрипте на адресе {address}.

Ответ метода:

```
{
  "address": "3N3keodUiS8WLEw9W4BKDNxgNdUpwSnpb3K",
  "script":
↪"3rbFDtbPwAvSp2vBvqGfGR9nRS1nBVnfuSCN3HxSZ7fVRpt3tuFG5JSmyTmvHPxYf34SocMRkRKFgzTtXXnnv7upRHXJzZrLSQo8tUW6yMtEiZ
↪",
  "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
}
```

Параметры ответа

- «address» - адрес в формате Base58
- «script» - Base64 представление скрипта

- «scriptText» - исходный код скрипта
- «complexity» - сложность скрипта
- «extraFee» - комиссия за исходящие транзакции, установленные скриптом

POST /addresses/sign/{address}

Возвращает закодированное в формарт Base58 сообщение, подписанное приватным ключом адресата {address}, сохраненным в keystore ноды. Сообщение сначала подписывается, после этого выполняется преобразование.

Запрос метода:

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

Ответ метода:

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

POST /addresses/verify/{address}

Проверяет подпись сообщения, выполненную адресатом {address}, в т.ч. созданную через метод POST /addresses/sign/{address}.

Запрос метода:

```
{
  "message": "wWshKhJj",
  "publickey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
↪"5kwwE9sDZzssoNaoBSJnb8RLqfYGt1NDGbTWWXUeX8b9amRRJN3hr5fhs9vHBq6VES5ng4hqbCUoDEsoQNauRRts"
}
```

Ответ метода:

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

POST /addresses/signText/{address}

Возвращает сообщение, подписанное приватным ключом адресата {address}, сохраненным в keystore ноды.

Запрос метода:

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

Ответ метода:

```
{
  "message": "message",
  "publicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
↪"5kVZfWfFmoYn38cJfNhkdct5WCyksMgQ7kjwHK7Zjnrzs9QYRWo6HuJoGc8WRMozdYcAVJvojJnPpArqPvu2uc3u"
}
```

POST /addresses/verifyText/{address}

Проверяет подпись сообщения, выполненную адресатом {address}, в т.ч. созданную через метод POST /addresses/signText/{address}.

Запрос метода:

```
{
  "message": "message",
  "publicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
↪"5kVZfWfFmoYn38cJfNhkdct5WCyksMgQ7kjwHK7Zjnrzs9QYRWo6HuJoGc8WRMozdYcAVJvojJnPpArqPvu2uc3u"
}
```

Ответ метода:

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

GET /addresses/validate/{addressOrAlias}

Проверяет корректность заданого адресата или его псевдонима {addressOrAlias} в блокчейн-сети работающей ноды.

Ответ метода:

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

POST /addresses/validateMany

Проверяет валидность адресов или алиасов.

Запрос метода:

```
{
  addressesOrAliases: [
    "3HSVTtjim3FmV21HWQ1LurMhFzjut7Aa1Ac",
    "alias:T:asdfghjk",
    "alias:T:1nvAliDAl1ass99911%^&$$$ "
  ]
}
```

Ответ метода:

```
{
  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: -.01234567890_
↪abcdefghijklmnopqrstuvwxyz)"
    }
  ]
}
```

GET /addresses/publicKey/{publicKey}

Возвращает адрес участника на основании его публичного ключа.

Ответ метода:

```
{
  "address": "3N4WaaaNAVLMQgVKTRSePgwBuAKvZTjAQbq"
}
```

GET /addresses/data/{address}

Возвращает все данные, записанные на аккаунт адресата {address}.

Ответ метода:

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

(continues on next page)

(продолжение с предыдущей страницы)

```
  }
]
```

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

Возвращает данные, записанные на аккаунт адресата {address} по ключу {key}.

Ответ метода:

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

20.1.3 Alias

Подсказка: Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

GET /alias/by-alias/{alias}

Получает адрес участника по его псевдониму {alias}.

Ответ метода

```
{

     "address": "address:3Mx2afTZ2KbRrLNbytyzTtXukZvqEB8SkW7"

}
```

GET /alias/by-address/{address}

Получает {alias} псевдоним участника по его адресу {address}.

Ответ метода

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

20.1.4 Anchoring

GET /anchoring/config

Подсказка: Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

Получение секции *анкоринга* конфигурационного файла ноды.

Ответ метода


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


20.1.5 Assets

Подсказка: Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

GET /assets/balance/{address}

Возвращает баланс всех ассетов адресата {address}.

Ответ метода:


```
{
  "address": "3Mv61qe6egMSjRDZiiuvJDnf3Q1qW9tTZDB",
  "balances": [
    {
      "assetId": "Ax9T4grFxx5m3KPUEKjMdnQkCKtBktf694wU2wJYvQUD",
      "balance": 4879179221,
      "quantity": 48791792210,
      "reissuable": true,
      "minSponsoredAssetFee" : 100,
      "sponsorBalance" : 1233221,
      "issueTransaction" : {
         "type" : 3,
```


(continues on next page)

(продолжение с предыдущей страницы)


```
        ...
      }
    },
    {
      "assetId": "49KfHPJcKvSAvNKwM7CTofjKHzL87SaSx8eyADBjv5Wi",
      "balance": 10,
      "quantity": 10000000000,
      "reissuable": false,
      "issueTransaction" : {
         "type" : 3,
         ...
      }
    }
  ]
}
```


Параметры метода:

- «address» - адрес участника
- «balances» - объект с балансами участника
- «assetId» - ассет ID
- «balance» - баланс ассета
- «quantity» - кол-во выпущенных ассетов
- «reissuable» - признак может быть ассет перевыпущен или нет
- «issueTransaction» - транзакция создания ассета
- «minSponsoredAssetFee» - минимальное значение коммиссии для спонсорских транзакций
- «sponsorBalance» - средства, выделенные для оплаты транзакций по спонсируемым ассетам

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

Возвращает баланс адресата {address} по ассету {assetId}.

Ответ метода:


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

GET /assets/details/{assetId}

Возвращает описание ассета {assetId}.

Ответ метода:

```
{
  "assetId" : "8tdULCMr598Kn2dUaKwHkvsNyFbDB1Uj5NxvVRTQRnMQ",
  "issueHeight" : 140194,
  "issueTimestamp" : 1504015013373,
  "issuer" : "3NCBMxgdghg4tUhEEffSXy11L6hUi6fcBpd",
  "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

Возвращает распределение ассета {assetId}.

Ответ метода:

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

POST /assets/balance

Возвращает баланс ассетов для одного или нескольких адресов.

Ответ метода

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

20.1.6 Blocks

Подсказка: Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

Последний блок в течение периода его создания может содержать разное количество транзакций. Это связано с тем, что пока блок не принят нодами-майнерами, количество транзакций в нем может постоянно меняться. Поэтому при использовании методов, предоставляющих информацию о последнем блоке, следует иметь ввиду, что количество транзакций в последнем блоке может измениться.

GET /blocks/height

Возвращает номер блока текущего состояния блокчейна.

Ответ метода:

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

GET /blocks/height/{signature}

Возвращает высоту (номер) блока по его подписи.

GET /blocks/first

Возвращает содержимое первого блока (genesis block).

GET /blocks/last

Возвращает содержимое последнего блока.

Ответ метода:

```
{
  "version": 2,
  "timestamp": 1479313809528,
  "reference":
↪"4MLXQDbARiJDEAoy5vZ8QYh1yNnDhdGhGWkDKna8J6QXb7agVpFEi16hHBGUxxnq8x4myG4w66DR4Ze8FM5dh8Gi",
  "nxtconsensus": {
    "basetarget": 464,
    "generationsignature": "7WUV2TufaRAyjiCPFdnAWbn2Q7Jk7nBmWbnnDXKDEeJv"
  },
  "transactions": [
    {
      "type": 2,
      "id":
↪"64hxaxZvB9iD1cfRf1j8KPTXs4qE7SHaDWTZKoUvgfVZotaJUtSGa5Bxi86ufAfp5ifoNAGknBqS9CpxBKG9RNVR",
      "fee": 100000,
      "timestamp": 1479313757194,
```

(continues on next page)

(продолжение с предыдущей страницы)


```
      "signature":
↪"64hxaxZvB9iD1cfRf1j8KPTXs4qE7SHaDWTZKoUvgfVZotaJUtSGa5Bxi86ufAfp5ifoNAGknBqS9CpxBKG9RNVR",
      "sender": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
      "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMhM3Uki7pLw",
      "recipient": "3N8UPtqiy322NVr1fLP7SaK1AaCU7oPaVuy",
      "amount": 1000000000
    }
  ],
  "generator": "3N5GRqzDBhjVXnCn44baHcz2GoZy5qLxtTh",
  "signature":
↪"4ZhZdLAvaGneLU4K4b2eTgRQvbBjEZrtwo1qAhM9ar3A3weGEutbfNKM4WJ9JZnV8BXenx8JRGVNwpfxf3prGaxd",
  "fee": 100000,
  "blocksize": 369
}
```


GET /blocks/at/{height}

Возвращает содержимое блока на высоте {height}.

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

Возвращает содержимое блоков в диапазоне от {from} до {to}.

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

Возвращает содержимое блоков с расширенной информацией о транзакциях в диапазоне от {from} до {to}.

GET /blocks/signature/{signature}

Возвращает содержимое блока по его подписи {signature}.

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

Возвращает все блоки сформированные (смайненные) адресатом {address}.

GET /blocks/child/{signature}

Возвращает унаследованный блок от блока с подписью {signature}.

GET /blocks/headers/at/{height}

Возвращает заголовок блока на высоте {height}.

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

Возвращает заголовки блоков диапазоне от {from} до {to}.

GET /blocks/headers/last

Возвращает заголовок последнего блока в блокчейне.

20.1.7 Consensus

Подсказка: Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

GET /consensus/algo

Возвращает тип алгоритма консенсуса, используемый в сети.

Ответ метода:

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

GET /consensus/settings

Возвращает параметры консенсуса, заданные в конфигурационном файле ноды.

Ответ метода:

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

GET /consensus/minersAtHeight/{height}

Возвращает очередь майнеров на высоте {height}.

Ответ метода:


```
{
"miners": [
    "3Mx5sDq4NXef1BRzJRAofa3orYFxLanxmd7",
    "3N2EsS6hJPYgRn7WFJHLJNnrsm92sUKcXkd",
    "3N2cQFfUDzG2iujBrFTnD2TAsCNohDxYu8w",
    "3N6pfQJyqjLCmMbU7G5sNABLmSF5aFT4KTF",
    "3NBbipRYQmZFudFCoVJXg9JMkkyZ4DEdZNS"
],
"height": 1
}
```


GET /consensus/miners/{timestamp}

Возвращает очередь майнеров на время {timestamp}.

Ответ метода:


```
{
    "miners": [
        "3Mx5sDq4NXef1BRzJRAofa3orYFxLanxmd7",
        "3N2EsS6hJPYgRn7WFJHLJNnrsm92sUKcXkd",
        "3N2cQFfUDzG2iujBrFTnD2TAsCNohDxYu8w",
        "3N6pfQJyqjLCmMbU7G5sNABLmSF5aFT4KTF",
        "3NBbipRYQmZFudFCoVJXg9JMkkyZ4DEdZNS"
    ],
    "timestamp": 1547804621000
}
```


GET /consensus/bannedMiners/{height}

Возвращает список заблокированных майнеров на высоте {height}.

Ответ метода:


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

GET /consensus/basetarget/{blockId}

Возвращает значение базовой сложности (basetarget) создания блока {blockId}.

GET /consensus/basetarget

Возвращает значение базовой сложности (basetarget) создания последнего блока.

GET /consensus/generatingbalance/{address}

Возвращает генерирующий баланс доступный для майниновой ноды {address} - минимальный баланс участинка, включая средства переденные участнику за лизинг, за последние 1000 блоков.

GET /consensus/generationsignature/{blockId}

Возвращает значение генерирующей подписи (generation signature) создания блока {blockId}.

GET /consensus/generationsignature

Возвращает значение генерирующей подписи (generation signature) последнего блока.

20.1.8 Contracts

Подсказка: Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

GET /contracts

Возвращает информацию по контрактам.

Ответ метода


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

POST /contracts

Возвращает некоторые параметры по одному или нескольким заданным в запросе идентификаторам контрактов.

Ответ метода


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


GET /contracts/info/{contractId}

Возвращает актуальную информацию по версии указанного контракта, его локации и хеш-суммы образа.

Ответ метода


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


GET /contracts/status/{id}

Возвращает статус исполняемой транзакции по контракту.

Ответ метода


```
[
  {
    "sender": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
    "senderPublicKey": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
    "txId": "4q5Q8vLeGBpcdQofZikyrrjHUS4pB1AB4qNEn2yHRKWU",
    "status": "Success",
    "code": null,
    "message": "Smart contract transaction successfully mined",
    "timestamp": 1558961372834,
    "signature":
↪"4gXy7qtzkaHHH6NkksnZ5pnv8juF65MvjQ9JgVztpgNwLNwuyyr27Db3gCh5YyADqZeBH72EyAkBouUoKvwJ3RQJ"
```


(continues on next page)

(продолжение с предыдущей страницы)

```
  },
  }
    "sender": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
    "senderPublicKey": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
    "txId": "4q5Q8vLeGBpcdQofZikyrrjHUS4pB1AB4qNEn2yHRKWU",
    "status": "Success",
    "code": null,
    "message": "Smart contract transaction successfully mined",
    "timestamp": 1558961376012,
    "signature":
↪"3Vhqc9DvNhMvFFtWnBuV4XwQ62ZcTAvLNZYmeGc7mGzMcnGZ3RLshDs393fnQu1WTh8CmL58YnvnjyULEEi5yorV"
  }
]
```

GET /contracts/{contractId}

Возвращает результат исполнения смарт-контракта по его идентификатору (id транзакции создания контракта).

Ответ метода:

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

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

Возвращает результат исполнения смарт-контракта по идентификатору транзакции исполнения контракта.

Ответ метода:

```
{
  "type": 105,
  "id": "2UAHvs4KsfBbRVPm2dCigWtqUHuaNQou83CXy6DGDiRa",
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
  "senderPublicKey": "2YvzcVLrqLCqouVrFZynjfotEuPNV9GrdauNpgdWXLsq",
  "fee": 500000,
  "timestamp": 1549365523980,
  "proofs": [
    "4BoG6wQnYyZWyUKzAwh5n1184tsEWUqUTWmXMExvvCU95xgk4UFB8iCnHJ4GhvJm86REB69hKM7s2WLAwTSXquAs"
  ],
  "version": 1,
  "tx": {
      "type": 103,
```

(continues on next page)

(продолжение с предыдущей страницы)


```
      "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
      "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
      "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
      "fee": 500000,
      "timestamp": 1550591678479,
      "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
      "version": 1,
      "image": "stateful-increment-contract:latest",
      "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
      "contractName": "stateful-increment-contract",
      "params": [],
      "height": 1619
  },
  "results": []
}
```


GET /contracts/{contractId}/{key}

Возвращает значение исполнения смарт-контракта по его идентификатору (id транзакции создания контракта) и ключу {key}.

Ответ метода:


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


20.1.9 Crypto

Подсказка: Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

POST /crypto/encryptSeparate

Шифрует текст отдельно для каждого получателя уникальным ключом.

Запрос метода


```
{
 "sender": "3MCUfX4P4U56hoQwSqXnLJenB6cDkxBjisL",
 "password": "some string as a password",
 "encryptionText": "some text to encrypt",
 "recipientsPublicKeys": [
↪"5R65oLxp3iwPekwirA4VwwUXaySz6W6YKXBKBRL352pwwcpsFcjRHJ1VVHLp63LkrkxsNod64V1pffeiZz5i2qXc",
 "9LopMj2GqWxBYgnZ2gxaNxwXqxXHuWd6ZAdVqkprR1fFMNvDUHYUCwFxsB79B9sefgxNdqwNtqzuDS8Zmn48w3S"]
}
```

Пример ответа

```
{
 "encryptedText": "IZ5Kk5YNspMWl/jmlTizVxD6Nik=",
 "publicKey":
↪"5R65oLxp3iwPekwirA4VwwUXaySz6W6YKXBKBRL352pwwcpsFcjRHJ1VVHLp63LkrkxsNod64V1pffeiZz5i2qXc",
 "wrappedKey":
↪"uWVoxJAzruwTDDSbphDS31TjSQX6CSWXivp3x34uE3XtnMqqK9swoaZ3LyAgFDR7o6CfkgzFkWmTen4qAZewPfBbwR"
},
{
 "encryptedText": "F9uO1ORGvSEDe6dWm1pzJQ+3xqE=",
 "publicKey":
↪"9LopMj2GqWxBYgnZ2gxaNxwXqxXHuWd6ZAdVqkprR1fFMNvDUHYUCwFxsB79B9sefgxNdqwNtqzuDS8Zmn48w3S",
 "wrappedKey":
↪"LdzdoKadUzBTMwczGYgu1AM4YrbbLr9Uh1MvQ3MPcLZUhCD9herz4dv1m6ssaVHPiBNUGgqKnLZ6Si4Cc64UvhXBbG"
}
```

POST /crypto/encryptCommon

Шифрует данные единым ключом CEK для всех получателей, CEK оборачивается уникальными KEK для каждого получателя.

Запрос метода

```
{
 "sender": "3MCUfX4P4U56hoQwSqXnLJenB6cDkxBjisL",
 "password": "some string as a password",
 "encryptionText": "some text to encrypt",
 "recipientsPublicKeys": [
↪"5R65oLxp3iwPekwirA4VwwUXaySz6W6YKXBKBRL352pwwcpsFcjRHJ1VVHLp63LkrkxsNod64V1pffeiZz5i2qXc",
 "9LopMj2GqWxBYgnZ2gxaNxwXqxXHuWd6ZAdVqkprR1fFMNvDUHYUCwFxsB79B9sefgxNdqwNtqzuDS8Zmn48w3S"]
}
```

Пример ответа

```
{
"encryptedText": "NpCCig2i3jzo0xBnfqjfedbti8Y=",
"recipientToWrappedStructure": {
  "5R65oLxp3iwPekwirA4VwwUXaySz6W6YKXBKBRL352pwwcpsFcjRHJ1VVHLp63LkrkxsNod64V1pffeiZz5i2qXc":
"M8pAe8HnKiWLE1HsC1ML5t8b7giWxiHfvagh7Y3F7rZL8q1tqMCJMYJo4qz4b3xjcuuUiV57tY3k7oSig53Aw1Dkkw",
  "9LopMj2GqWxBYgnZ2gxaNxwXqxXHuWd6ZAdVqkprR1fFMNvDUHYUCwFxsB79B9sefgxNdqwNtqzuDS8Zmn48w3S":
"Doqn6gPvBBeSu2vdwgFYMbDHM4knEGMbqPn8Np76mNRRoZXLDioofyVbSSaTTEr4cvXwzEwVMugiy2wuzFWk3zCiT3"
 }
}
```

POST /crypto/decrypt

Расшифровывает данные. Расшифровка доступна в случае, если ключ получателя сообщения находится в keystore ноды.

Запрос метода

```
{
 "recipient": "3M5F8B1qxSY1W6kA2ZnQiDB4JTGz9W1jvQy",
 "password": "some string as a password",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
 "encryptedText": "oiKFJijfid8HkjsjdhKHhud987d",
 "wrappedKey": "M5F8B1qxSY1W6kA2ZnQiDB4JTGzA2ZnQiDB4JTGz9W1jvQy"
 "senderPublicKey": "M5F8B1qxSY1W6kA2ZnQiDB4JTGzA2ZnQiDB4JTGz9W1jvQy",
}
```

Пример ответа

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

20.1.10 Debug

Подсказка: Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

GET /debug/blocks/{howMany}

Отображает размер и полный хеш последних блоков. Количество блоков указывается при запросе.

Ответ метода

```
[
    {
        "226": "7CkZxrAjU8bnat8CjVAPagobNYazyv1HASubmp7YYqGe"
    },
    {
        "226": "GS3y9fUHAKCamq52TPsjizDVir8J7iGoe8P2XZLasxsC"
    },
    {
        "226": "B9LmhGGDdvcfUA9JEWvyVrT9sazZE6gibpAN13xUN7KV"
    },
    {
        "226": "Byb9MHtwYf3MFyi2tbhQ3GTdCct5phKq9REkbjQTzdne"
    },
    {
        "226": "HSxSHbiV4tZc8RaN6jxdhgtkAhjxuLn76uHxerMRUefA"
    }
]
```

GET /debug/info

Отображает необходимую информацию для отладки и тестирования.

Ответ метода


```
{
    "stateHeight": 74015,
    "extensionLoaderState": "State(Idle)",
    "historyReplierCacheSizes": {
        "blocks": 13,
        "microBlocks": 2
    },
    "microBlockSynchronizerCacheSizes": {
        "microBlockOwners": 0,
        "nextInventories": 0,
        "awaiting": 0,
        "successfullyReceived": 0
    },
    "scoreObserverStats": {
        "localScore": 42142328633037120000,
        "scoresCacheSize": 4
    },
    "minerState": "mining microblocks"
}
```


POST /debug/rollback

Убирает из блокчейна все блоки после указанной высоты.

Запрос метода


```
{
    "rollbackTo": 100,
    "returnTransactionsToUtx": true
}
```


Ответ метода


```
{
    "BlockId":
↪"4U4Hmg4mDYrvxaZ3JVzL1Z1piPDZ1PJ61vd1PeS7ESZFkHsUCUqeeAZoszTVr43Z4NV44dqbLv9WdrLytDL6gHuv"
}
```


POST /debug/validate

Валидирует транзакции и измеряет затраченное время в миллисекундах.

Параметры запроса

```
"id" - Transaction ID
```

Ответ метода

```
{
    "valid": false,
    "validationTime": 14444
}
```

GET /debug/minerInfo

Отображает информацию о майнере, необходимую для отладки.

Ответ метода

```
[
    {
        "address": "3JFR1pmL6biTzr9oa63gJcjZ8ih429KD3aF",
        "miningBalance": 1248959867200000,
        "timestamp": 1585923248329
    }
]
```

GET /debug/historyInfo

Отображает историю последнего блока, необходимую для отладки.

Ответ метода

```
{
"lastBlockIds": [
"37P4fvexYHPUzNPRRqYbRYxGz7x3r5jFznck7amaS6aWnHL5oQqrqCzsSh1HvYKnd2ZhU6n6sWYPb3hxsY8FBfmZ",
"5RRu1qtesz4KvrVp4fxzQHebq2fRanNsg3HJKwD4uChqySm7vFHCdHKU6iZYXJDVmfSxiE9Maeb6sM2JireaWLbx",
"3Lo27JfjekcZnJsYEe7st7evDZ6TgmCUBtiZrSxUCobKL48DZQ4dXMfp89WYjEykH15HEHSXzqMSTQigE8vEcN2r",
"r4RuxEXAqgfDMKVXRWmZcGMaWKDsAvVxfXDtw8d6bamLR61J1gaoesargYSoZQqRbDrBcefLprk7D78fA728719",
"3F4Up46crZbpKVWUeieL6GeSrVMYm7JJ7aX6aHD6B8wedFggSKv8d3H39Qy9MLEauFBU9m3qZV1U8emhmnqwmLbg",
"QSuBkEtVe9nik5T5S33ogeCbgKy7ihBkS2pwYayK23m4ANier83ThpajEzvpbyPy9pPWZc5St8mYUKxXDscKuRC",
"4udpNnz3e1M1GbVZxtwfg8gpF6EbiKxRCRBwi6iRMyLsvh5J2Ec9Wqyu2sq2KYL75o12yiP8TszworeUfuxNmJ5g",
"5BZYZ4RZAJjM5KKCaHpyUsXnb4uunnM5kcfTojc5QzQo3vyP2w3YD4qrALizkkQQR4ziS77BoAGb56QCecUtHFFN",
"5JwfLaF1oGxRXVCdDbFuKpxrvxgLCGU3kCFwxUhLL8G3xV211MrKBuAuQ4MaC5uN574uV9U8M6HfHTMERnfr5jGJ",
"4bysMhz14E1rC7dLYScfVVqPmHqzi8jdhcnkruJmCNL86TwV2cbF7G9YVchvTrv9qbQZ7JQownV59gRRcD26zm16"
],
"microBlockIds": []
}
```

GET /debug/configInfo

Отображает конфигурационный файл ноды.

Ответ метода

```
  {
"node": {
  "anchoring": {
    "enable": "no"
  },
  "blockchain": {
    "consensus": {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      "type": "pos"
    },
    "custom": {
      "address-scheme-character": "K",
      "functionality": {
        "blocks-for-feature-activation": 10,
        "feature-check-blocks-period": 30,
        "pre-activated-features": {  ...
  ....................................
  "wallet": {
    "file": "wallet.dat",
    "password": ""
  },
  "waves-crypto": "yes"
  }
  }
```

DELETE /debug/rollback-to/{signature}

Откатывает блокчейн до блока с указанной подписью.

Параметры запроса

```
"signature" - Block signature
```

Ответ метода

```
{
    "BlockId":
↪"4U4Hmg4mDYrvxaZ3JVzL1Z1piPDZ1PJ61vd1PeS7ESZFkHsUCUqeeAZoszTVr43Z4NV44dqbLv9WdrLytDL6gHuv"
}
```

GET /debug/portfolios/{address}

Отображает текущий портфель неучтённых транзакций в UTX пуле.

Параметры запроса

```
"address" - Node address
```

Ответ метода

```
{
    "balance": 104665861710336,
    "lease": {
        "in": 0,
        "out": 0
    },
    "assets": {}
}
```

POST /debug/print

Распечатывает строку при уровне логирования DEBUG в лог-файл.

Запрос метода

```
{
    "message": "string"
}
```

GET /debug/state

Отображает текущий стейт ноды.

Ответ метода

```
{
    "3JD3qDmgL1icDaxa3n24YSjxr9Jze5MBVVs": 4899000000,
    "3JPWx147Xf3f9fE89YtfvRhtKWBHy9rWnMK": 17528100000,
    "3JU5tCoswHH7FKPBUowySWBnQwpbZiYyNhB": 300021381800000,
    "3JCJChsQ2CGyHc9Ymu8cnsES6YzjjJELu3a": 75000362600000,
    "3JEW9XnPC8w3qQ4AJyVTDBmsVUp32QKoCGD": 5000000000,
    "3JSaKNX94deXJkywQwTFgbigTxJa36TDVg3": 6847000000,
    "3JFR1pmL6biTzr9oa63gJcjZ8ih429KD3aF": 1248938560600000,
    "3JV6V4JEVc3a9uSqRmdUMvMKMfZa16HbGmq": 4770000000,
    "3JZtYeGEZHjb2zQ6EcSEo524PdafPn6vWkc": 900000000,
    "3JMMFLX9d1rmXaBK9AF7Wuwzu4vRkkoVQBC": 4670000000,
    "3JJDpPDqSPokKp5jEmzwMzmaPUyopLZjW1C": 800000000,
    "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwD9t": 994280900000
}
```

GET /debug/stateWE/{height}

Отображает стейт ноды на указанной высоте.

Параметры запроса

```
"height" - Block height
```

Ответ метода

```
{
    "3JPWx147Xf3f9fE89YtfvRhtKWBHy9rWnMK": 17528100000,
    "3JU5tCoswHH7FKPBUowySWBnQwpbZiYyNhB": 300020907600000,
    "3JCJChsQ2CGyHc9Ymu8cnsES6YzjjJELu3a": 75000350600000,
    "3JSaKNX94deXJkywQwTFgbigTxJa36TDVg3": 6847000000,
    "3JFR1pmL6biTzr9oa63gJcjZ8ih429KD3aF": 1248960085800000,
    "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwD9t": 994280900000
}
```

20.1.11 Leasing

Подсказка: Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

GET /leasing/active/{address}

Возвращает список транзакций создания лизинга, в которых принимал участие {address}, как отправитель или получатель.

Ответ метода:


```
[
  {
    "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
  }
]
```

20.1.12 Licenses

Подсказка: Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

GET /licenses

Возвращает список всех загруженных лицензий.

Ответ метода


```
[
  {
    "license": {
      "version": 1,
      "id": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
      "license_type": null,
      "issued_at": "2020-02-27T16:11:14.784Z",
      "node_owner_address": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
      "valid_from": "2020-02-20",
      "valid_to": "2020-02-27",
      "features": [
        "all_inclusive"
      ]
    },
    "signer_public_key": "dmLT1ippM7tmfSC8u9P4wU6sBgHXGYy6JYxCq1CCh8i",
    "signature":
↪"ff9b8af966b4c84e66d3847a514e65f55b2c1f63afcd8b708b9948a814cb8957mLT1ippM7tmfSC8u",
    "signer_id": "ff9b8af966b4c84e66d3847a514e65f55b2c1f63afcd8b708b9948a814cb8957"
  },
  {
    "license": {
      "version": 1,
      "id": "49KfHPJcKvSAvNKwM7CTofjKHzL87SaSx8eyADBjv5Wi",
      "license_type": null,
      "issued_at": "2020-02-27T16:12:34.327Z",
      "node_owner_address": "3N4WaaaNAVLMQgVKTRSePgwBuAKvZTjAQbq",
      "valid_from": "2020-02-29",
      "valid_to": null,
      "features": [
        "all_inclusive"
      ]
    },
    "signer_public_key": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
    "signature":
↪"5kwwE9sDZzssoNaoBSJnb8RLqfYGt1NDGbTWWXUeX8b9amRRJN3hr5fhs9vHBq6VES5ng4hqbCUoDEsoQNauRRts",
    "signer_id": "8tdULCMr598Kn2dUaKwHkvsNyFbDB1Uj5NxvVRTQRnMQ"
  }
]
```

GET /licenses/status

Возвращает статус активации лицензии ноды.

Ответ метода

```
{
  "status" : "TRIAL",
  "description" : "Trial period is active. Blocks before expiration: '{num}'"
}
```

POST /licenses/upload

Добавляет новую лицензию в JSON формате в ноду.

Запрос метода

```
{
  "license": {
    "version": 1,
    "id": "49KfHPJcKvSAvNKwM7CTofjKHzL87SaSx8eyADBjv5Wi",
    "license_type": null,
    "issued_at": "2020-02-27T16:12:34.327Z",
    "node_owner_address": "3N4WaaaNAVLMQgVKTRSePgwBuAKvZTjAQbq",
    "valid_from": "2020-02-29",
    "valid_to": null,
    "features": [
      "all_inclusive"
    ]
  },
  "signer_public_key": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
↪"5kwwE9sDZzssoNaoBSJnb8RLqfYGt1NDGbTWWXUeX8b9amRRJN3hr5fhs9vHBq6VES5ng4hqbCUoDEsoQNauRRts",
  "signer_id": "8tdULCMr598Kn2dUaKwHkvsNyFbDB1Uj5NxvVRTQRnMQ"
}
```

Ответ метода

```
{
    "message": "License upload successfully"
}
```

20.1.13 Node

Подсказка: Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

GET /node/config

Возвращает основные конфигурационные параметры ноды.

Ответ метода:


```
{
    "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": 1000000,
        "14": 100000000,
        "15": 100000000,
        "102": 0
    }
}
```


POST /node/stop

Запрос останавливает ноду.

GET /node/status

Возвращает основные конфигурационные параметры ноды.

Ответ метода:


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

GET /node/version

Возвращает версию приложения.

Ответ метода:

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

GET /node/owner

Возвращает адрес и публичный ключ владельца ноды.

Ответ метода:

```
{
  "address": "3JFR1pmL6biTzr9oa63gJcjZ8ih429KD3aF",
  "publicKey": "EPxkVA9iQejsjQikovyxkkY8iHnbXsR3wjgkgE7ZW1Tt"
}
```

20.1.14 Peers

Подсказка: Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

POST /peers/connect

Запрос на подключение нового узла к ноде.

Запрос метода:

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

Ответ метода:

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

GET /peers/connected

Возвращает список подключенных нод.

Ответ метода:


```
{
  "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
    }
  ]
}
```


GET /peers/all

Возвращает список всех известных нод.

Ответ метода:


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


GET /peers/suspended

Возвращает список suspended нод.

Ответ метода:


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

POST /peers/identity

Получение публичного ключа пира, к которому подключается нода для передачи конфиденциальных данных.

Запрос метода:

```
{
  "address": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
  "signature":
↪"6RwMUQcwrxtKDgM4ANes9Amu5EJgyfF9Bo6nTpXyD89ZKMAcpCM97igbWf2MmLXLdqNxdsUc68fd5TyRBEB6nqf"
}
```

Параметры:

- address - блокчейн-адрес, который соответствует параметру «privacy.owner-address» в конфигурационном файле ноды;
- signature - электронная подпись от значения поля «address».

Ответ метода:

```
{
  "publicKey": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8"
}
```

Параметры:

- publicKey - публичный ключ пира, связанный с параметром «privacy.owner-address» в его конфигурационном файле. Если выключен режим проверки handshakes, то параметр publicKey не отображается.

GET /peers/hostname/{address}

Получение hostname и IP-адреса ноды по ее адресу в сети Waves Enterprise.

Ответ метода:

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

GET /peers/allowedNodes

Получение актуального списка разрешенных участников сети на момент запроса.

Ответ метода:

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

(continues on next page)

(продолжение с предыдущей страницы)


```
      "publicKey": "J3ffCciVu3sustgb5vxmEHczACMR89Vty5ZBLbPn9xyg"
    },
    {
      "address": "3JRY1cp7atRMBd8QQoswRpH7DLawM5Pnk3L",
      "publicKey": "5vn4UcB9En1XgY6w2N6e9W7bqFshG4SL2RLFqEWEbWxG"
    }
  ],
  "timestamp": 1558697649489
}
```


20.1.15 Permissions

Подсказка: Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

GET /permissions/{address}

Возвращает роли (permissions), назначенные на указанный адрес {address}, действительные на текущий момент времени.

Ответ метода:


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


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

Возвращает роли (permissions), назначенные на указанный адрес {address}, действительные на момент времени {timestamp}.

Ответ метода:


```
{
    "roles": [
        {
        "role": "miner"
        },
        {
        "role": "permissioner"
        }
```


(continues on next page)

(продолжение с предыдущей страницы)

```
    ],
    "timestamp": 1544703449430
}
```

POST /permissions/addresses

Возвращает роли (permissions), назначенные на список адресов, действительные на текущий момент времени.

Запрос метода:

```
{
    "addresses": [
        "3N2cQFfUDzG2iujBrFTnD2TAsCNohDxYu8w", "3Mx5sDq4NXef1BRzJRAofa3orYFxLanxmd7"
    ],
    "timestamp": 1544703449430
}
```

Ответ метода:

```
{
    "addressToRoles": [
        {
        "address": "3N2cQFfUDzG2iujBrFTnD2TAsCNohDxYu8w",
        "roles": [
            {
            "role": "miner"
            },
            {
            "role": "permissioner"
            }
        ]
        },
        {
        "address": "3Mx5sDq4NXef1BRzJRAofa3orYFxLanxmd7",
        "roles": [
            {
            "role": "miner"
            }
        ]
        }
    ],
    "timestamp": 1544703449430
}
```

20.1.16 PKI

Предупреждение: Методы PKI работают только с ГОСТ-криптографией.

В PKI используются форматы ЭП, приведенные в таблице ниже. Номер формата ЭП из таблицы соответствует значению поля `sigtype`.

Таблица 1: Форматы ЭП

№	Формат ЭП
1	CAdES-BES
2	CAdES-X Long Type 1
3	CAdES-T

POST /pki/sign

Подсказка: Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

Метод формирует отсоединённую ЭП для данных, передаваемых в запросе. В данном запросе `inputData` - это данные для формирования ЭП в виде массива байт в кодировке **Base64**, `keystoreAlias` - это наименование ключевого контейнера закрытого ключа ЭП. Также необходимо указать пароль от ключевого контейнера в параметре `password`.

Пример запроса

```
{
"inputData" : "SGVsbG8gd29ybGQh",
"keystoreAlias" : "key1",
"password" : "password",
"sigType" : "CAdES_X_Long_Type_1",
}
```

Пример ответа

```
{
  "signature" :
↪"c2RmZ3NkZmZoZ2ZkZ2hmZGpkZ2ZoamhnZmtqaGdmamtkZmdoZmdkc2doZmQjsndjfvnksdnjfn="
 }
```

GET /pki/keystoreAliases

Метод возвращает список всех keystore-алиасов на ГОСТ-криптографии.

Пример ответа

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

POST /pki/verify

Метод выполняет проверку УКЭП для данных, переданных в запросе. Поле `extendedKeyUsageList` является опциональным и может содержать массив значений OID (Объектный идентификатор) для определения области действия сертификата. Проверку сертификата может осуществлять любая нода, имеющая параметры запроса.

Пример запроса

```
{
 "inputData" : "SGVsbG8gd29ybGQh",
 "signature" : "c2RmZ3NkZmZoZ2ZkZ2hmZGpkZ2ZoamhnZmtqaGdmamtkZmdoZmdkc2doZmQ=",
 "sigType" : "CAdES_X_Long_Type_1",
 "extendedKeyUsageList": [
 "1.2.643.7.1.1.1.1",
 "1.2.643.2.2.35.2"
 ]
 }
```

Пример ответа

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

Работа с методом POST /pki/verify

Нода имеет возможность проверять УКЭП (Усиленная квалифицированная электронная подпись), используя метод API *Post /pki/verify*. Для корректности работы метода API *Post /pki/verify* необходимо установить корневой сертификат на ноду. Корневой сертификат УЦ однозначно идентифицирует центр сертификации и является основанием в цепочке доверия.

Как установить корневой сертификат на ноду

Корневой сертификат устанавливается в следующую папку со средой Java:

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

Пароль по умолчанию на хранилище сертификатов Java cacerts - `changeit`. При желании вы можете изменить пароль. Установка сертификатов выполняется следующей командой:

```
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
```

20.1.17 Privacy

Подсказка: Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

POST /privacy/sendData

Запись конфиденциальных данных в хранилище ноды.

Запрос метода:

```
{
  "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"
   },
   "data":
↪"TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyByZWFzb24sIGJ1dCBieSBOaGlzIHNpbmd1bGFyIHBhc3Npb24gZnJvbSBvdGhl
↪",
   "hash": "FRog42mnzTA292ukng6PHoEK9Mpx9GZNrEHecfvpwmta"
}
```

Параметры:

- sender - блокчейн-адрес, от которого должны рассылаться данные (соответствуют значению параметра «privacy.owner-address» в конфигурационном файле ноды);
- password - пароль для доступа к закрытому ключу keystore ноды;
- policyId - идентификатор группы, в рамках которой пересыалются данные;
- type - тип данных;
- info - информация о данных;
- data - данные в бинарном представлении;
- hash - хеш от данных.

Ответ метода:


```
{
  "senderPublicKey": "Gt3o1ghh2M2TS65UrHZCTJ82LLcMcBrxuaJyrgsLk5VY",
  "policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "dataHash": "FRog42mnzTA292ukng6PHoEK9Mpx9GZNrEHecfvpwmta",
  "proofs": [
  "2jM4tw4uDmspuXUBt6492T7opuZskYhFGW9gkbq532BvLYRF6RJn3hVGNLuMLK8JSM61GkVgYvYJg9UscAayEYfc"
  ],
  "fee": 110000000,
  "id": "H3bdFTatppjnMmUe38YWh35Lmf4XDYrgsDK1P3KgQ5aa",
  "type": 114,
  "timestamp": 1571043910570
 }
```


GET /privacy/{policy-id}/recipients

Получение адресов всех участников, записанных в группу {policy-id}.

Ответ метода:


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


GET /privacy/{policy-id}/owners

Получение адресов всех владельцев, записанных в группу {policy-id}.

Ответ метода:


```
[
  "3GCFaCWtvLDnC9yX29YftMbn75gwfdwGsBn",
  "3GGxcmNyq8ZAHzK7or14Ma84khwW8peBohJ",
  "3GRLFi4rz3SniCuC7rbd9UuD2KUZyNh84pn",
  "3GKpShRQRTddF1yYhQ58ZnKMTnp2xdEzKqW"
]
```

GET /privacy/{policy-id}/hashes

Получение массива идентификационный хешей, которые записаны в привязке к {policy-id}.

Ответ метода:

```
[
"FdfdNBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
"eedfdNBVqYXrapgJP9atQccdBPAgJPwHDKkh6A"
]
```

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

Получение пакета конфиденциальных данных по идентификационному хешу.

Ответ метода:

```
c29tZV9iYXNlNjRfZW5jb2RlZF9zdHJpbmc=
```

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

Получение метаданных для пакета конфиденциальных данных по идентификационному хешу.

Ответ метода:

```
{
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "policy": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "type": "file",
  "info": {
    "filename": "Contract №100/5.doc",
    "size": 2048,
    "timestamp": 1000000000,
    "author": "AIvanov@org.com",
    "comment": "Comment"
  },
"hash": "e67ad392ab4d933f39d5723aeed96c18c491140e119d590103e7fd6de15623f1"
}
```

POST /privacy/forceSync

Запрос на принудительное получение пакета конфиденциальных данных.

Ответ метода:

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

POST /privacy/getInfos

Запрос на возвращение массива мета-информации о приватных данных по предоставленным идентификатору группы и хешу данных.

Пример запроса:

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

Ответ метода:

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

20.1.18 Transactions

Подсказка: Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

GET /transactions/info/{id}

Запрос сведений по транзакции по ее ID.

Параметры запроса:

```
"id" - Transaction ID
```

Ответ метода:

```
{
  "type": 4,
  "id": "52GG9U2e6foYRKp5vAzsTQ86aDAABfRJ7synz7ohBp19",
  "sender": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
  "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMhM3Uki7pLw",
  "recipient": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
  "assetId": "E9yZC4cVhCDfbjFJCc9CqkAtkoFy5KaCe64iaxHM2adG",
  "amount": 100000,
  "fee": 100000,
  "timestamp": 1549365736923,
  "attachment": "string",
  "signature":
↪"GknccUA79dBcwWgKjqB7vYHcnsj7caYETfncJhRkkaetbQon7DxbpMmvK9LYqUkirJp17geBJCRTNkHEoAjtsUm",
  "height": 7782
}
```

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

Возвращает последние {limit} транзакций с адреса {address}.

Ответ метода:

```
[
  [
    {
      "type": 2,
      "id":
↪"4XE4M9eSoVWVdHwDYXqZsXhEc4q8PH9mDMUBegCSBBVHJyP2Yb1ZoGi59c1Qzq2TowLmymLNkFQjWp95CdddnyBW",
      "fee": 100000,
      "timestamp": 1549365736923,
      "signature":
↪"4XE4M9eSoVWVdHwDYXqZsXhEc4q8PH9mDMUBegCSBBVHJyP2Yb1ZoGi59c1Qzq2TowLmymLNkFQjWp95CdddnyBW",
      "sender": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
      "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMhM3Uki7pLw",
      "recipient": "3N9iRMou3pgmyPbFZn5QZQvBTQBkL2fR6R1",
      "amount": 1000000000
    }
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  ]
]
```

GET /transactions/unconfirmed

Возвращает все неподтвержденные транзакции из utx-pool ноды.

Ответ метода:

```
[
  {
    "type": 4,
    "id": "52GG9U2e6foYRKp5vAzsTQ86aDAABfRJ7synz7ohBp19",
    "sender": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
    "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMhM3Uki7pLw",
    "recipient": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
    "assetId": "E9yZC4cVhCDfbjFJCc9CqkAtkoFy5KaCe64iaxHM2adG",
    "amount": 100000,
    "fee": 100000,
    "timestamp": 1549365736923,
    "attachment": "string",
    "signature":
↪"GknccUA79dBcwWgKjqB7vYHcnsj7caYETfncJhRkkaetbQon7DxbpMmvK9LYqUkirJp17geBJCRTNkHEoAjtsUm"
  }
]
```

GET /transactions/unconfirmed/size

Возвращает количество транзакций, находящихся в UTX-пуле.

GET /unconfirmed/info/{id}

Запрос сведений по транзакции из UTX-пула по ее ID.

POST /transactions/calculateFee

Расчитывет размер комиссии по переданной транзакции.

Параметры запроса

```
"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.
```

Запрос метода

```
{
 "type": 10,
 "timestamp": 1549365736923,
 "sender": "3MtrNP7AkTRuBhX4CBti6iT21pQpEnmHtyw",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
 "alias": "ALIAS",
}
```

или

```
{
  "type": 4,
  "sender": "3MtrNP7AkTRuBhX4CBti6iT21pQpEnmHtyw",
  "recipient": "3P8JYPHrnXSfsWP1LVXySdzU1P83FE1ssDa",
  "amount": 1317209272,
  "feeAssetId": "8LQW8f7P5d5PZM7GtZEBgaqRPGSzS3DfPuiXrURJ4AJS",
  "attachment": "string"
}
```

Ответ метода

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

или

```
{
  "feeAssetId": "8LQW8f7P5d5PZM7GtZEBgaqRPGSzS3DfPuiXrURJ4AJS",
  "feeAmount": 10000
}
```

POST /transactions/sign

Подписывает транзакцию закрытым ключом отправителя, сохраненным в keystore ноды. После подписания ответ метода должен быть подан на вход метода *Broadcast*.

Для подписания запросов ключом из keystore ноды требуется обязательное указание пароля в поле `password`.

Примеры запросов

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

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": 1000000,
   "name": "faucet",
   "script": "base64:AQQAAAAHJG1hdGNoMAUAAAACdHgG+RXSzQ=="
}

.. _tx-sponsorship:
```

14. Sponsorship

```
{
   "sender": "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwD9t",
   "assetId": "G16FvJk9vabwxjQswh9CQAhbZzn3QrwqWjwnZB3qNVox",
   "fee": 100000000,
   "isEnabled": false,
   "type": 14,
   "password": "1234",
   "version": 1
}
```

102. PermissionTransaction

Пример запроса

```
{
   "type":102,
```

(continues on next page)

(продолжение с предыдущей страницы)


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


103. CreateContractTransaction

Пример запроса


```
{
  "fee": 100000000,
  "image": "stateful-increment-contract:latest",
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
  "contractName": "stateful-increment-contract",
  "sender": "3PudkbvjV1nPj1TkuuRahh4sGdgfr4YAUV2",
  "password": "",
  "params": [],
  "type": 103,
  "version": 1,
 }
```


Пример ответа


```
{
    "type": 103,
    "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
    "fee": 500000,
    "timestamp": 1550591678479,
    "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
    "version": 1,
    "image": "stateful-increment-contract:latest",
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
    "contractName": "stateful-increment-contract",
    "params": [],
    "height": 1619
}
```


104. CallContractTransaction

Пример запроса


```
{
   "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
   "fee": 10,
   "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
   "type": 104,
   "version": 1,
   "contractVersion": 1
   "password": "",
```


(continues on next page)

(продолжение с предыдущей страницы)


```
   "params": [
      {
            "type": "integer",
            "key": "a",
            "value": 1
      },
      {
            "type": "integer",
            "key": "b",
            "value": 100

      }
   ]
}
```


Пример ответа


```
{
   "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

Пример ответа


```
{
   "type": 105,
   "id": "2UAHvs4KsfBbRVPm2dCigWtqUHuaNQou83CXy6DGDiRa",
   "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
   "senderPublicKey": "2YvzcVLrqLCqouVrFZynjfotEuPNV9GrdauNpgdWXLsq",
   "fee": 500000,
   "timestamp": 1549365523980,
   "proofs": [
      "4BoG6wQnYyZWyUKzAwh5n1184tsEWUqUTWmXMExvvCU95xgk4UFB8iCnHJ4GhvJm86REB69hKM7s2WLAwTSXquAs"
   ],
```


(continues on next page)

(продолжение с предыдущей страницы)


```
   "version": 1,
   "tx": {
      "type": 103,
      "id": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
      "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
      "senderPublicKey": "2YvzcVLrqLCqouVrFZynjfotEuPNV9GrdauNpgdWXLsq",
      "fee": 500000,
      "timestamp": 1549365501462,
      "proofs": [
         "2ZK1Y1ecfQXeWsS5sfcTLM5W1KA3kwi9Up2H7z3Q6yVzMeGxT9xWJT6jREQsmuDBcvk3DCCiWBdFHaxazU8pbo41"
      ],
      "version": 1,
      "image": "localhost:5000/contract256",
      "imageHash": "930d18dacb4f49e07e2637a62115510f045da55ca16b9c7c503486828641d662",
      "params": []
   },
   "results": []
}
```


106. DisableContractTransaction

Пример запроса

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

Пример ответа


```
{
 "type": 106,
 "id": "8Nw34YbosEVhCx18pd81HqYac4C2pGjyLKck8NhSoGYH",
 "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
 "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
 "fee": 500000,
 "proofs": [
↪"5GqPQkuRvG6LPXgPoCr9FogAdmhAaMbyFb5UfjQPUKdSc6BLuQSz75LAWix1ok2Z6PC5ezPpjqzqnr15i3RQmaEc" ],
 "version": 1,
 "contractId": "Fz3wqAWWcPMT4M1q6H7crLKtToFJvbeLSvqjaU4ZwMpg",
 "height": 1632
}
```


107. UpdateContractTransaction

Пример запроса


```
{
  "image" : "registry.wvservices.com/we-sc/tdm-increment3:1028.1",
  "sender" : "3Mxxz9pBYS5fJMARJNQmzYUHxiWAtvMzSRT",
  "password": "",
  "fee" : 100000000,
  "contractId" : "EnsihTUHSNAB9RcWXJbiWT98X3hYtCw3SBzK8nHQRCWA",
  "imageHash" : "0e5d280b9acf6efd8000184ad008757bb967b5266e9ebf476031fad1488c86a3",
```


(continues on next page)

(продолжение с предыдущей страницы)

```
  "type" : 107,
  "version" : 1
 }
```

Пример ответа

```
{
  "senderPublicKey":
↪"5qBRDm74WKR5xK7LPs8vCy9QjzzqK4KCb8PL36fm55S3kEi2XZETHFgMgp3D13AwgE8bBkYrzvEvQZuabMfEyJwW",
  "tx":
  {
  "senderPublicKey":
↪"5qBRDm74WKR5xK7LPs8vCy9QjzzqK4KCb8PL36fm55S3kEi2XZETHFgMgp3D13AwgE8bBkYrzvEvQZuabMfEyJwW",
  "image":"registry.wvservices.com/we-sc/tdm-increment3:1028.1",
  "sender":"3Mxxz9pBYS5fJMARJNQmzYUHxiWAtvMzSRT",
  "proofs":[
↪"3tNsTyteeZrxEbVSv5zPT6dr247nXsVWR5v7Khx8spypgZQUdorCQZV2guTomutUTcyxhJUjNkQW4VmSgbCtgm1Z"],
  "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
  }
 }
```

110. GenesisRegisterNode

Пример запроса

```
{
   "type": 110,
   "id": "2Xgbsqgfbp5fiq4nsaAoTkQsXc399tXdnKom8prEZqPW2Q7xZKNKCCqpkyMtmJMgYLpvwynbxHPTFpFEfFdyLpJ",
   "fee": 0,
   "timestamp": 1489352400000,
   "signature":
↪"2Xgbsqgfbp5fiq4nsaAoTkQsXc399tXdnKom8prEZqPW2Q7xZKNKCCqpkyMtmJMgYLpvwynbxHPTFpFEfFdyLpJ",
   "targetPublicKey": "3JNLQYuHYSHZiHr5KjJ89wwFJpDMdrAEJpj",
   "target": "3JNLQYuHYSHZiHr5KjJ89wwFJpDMdrAEJpj"
}
```

Пример ответа

```
{
   "signature":
↪"2Xgbsqgfbp5fiq4nsaAoTkQsXc399tXdnKom8prEZqPW2Q7xZKNKCCqpkyMtmJMgYLpvwynbxHPTFpFEfFdyLpJ",
   "fee": 0,
   "id": "2Xgbsqgfbp5fiq4nsaAoTkQsXc399tXdnKom8prEZqPW2Q7xZKNKCCqpkyMtmJMgYLpvwynbxHPTFpFEfFdyLpJ",
   "type": 110,
   "targetPublicKey": "3JNLQYuHYSHZiHr5KjJ89wwFJpDMdrAEJpj",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
   "timestamp": 1489352400000,
   "target": "3JNLQYuHYSHZiHr5KjJ89wwFJpDMdrAEJpj",
   "height": 1
}
```

111. RegisterNode

Пример запроса

```
{
 "type": 111,
 "opType": "add",
 "sender":"3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
 "password": "",
 "targetPubKey": "apgJP9atQccdBPAgJPwH3NBVqYXrapgJP9atQccdBPAgJPwHapgJP9atQccdBPAgJPwHDKkh6A8",
 "nodeName": "Node #1",
 "fee": 500000,
 }
```

112. CreatePolicy

Пример запроса

```
{
   "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
   "policyName": "Policy# 7777",
   "password":"sfgKYBFCF@#$fsdf()*%",
   "recipients": [
     "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
     "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
     "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
         "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
         "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx"
   ],
   "fee": 15000000,
   "description": "Buy bitcoin by 1c",
   "owners": [
     "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
     "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
     "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
   ],
   "type": 112
 }
```

113. UpdatePolicy

Пример запроса

```
{
  "policyId": "7wphGbhqbmUgzuN5wzgqwqtViTiMdFezSa11fxRV58Lm",
  "password":"sfgKYBFCF@#$fsdf()*%",
  "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "proofs": [],
  "recipients": [
  "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
  "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
```

(continues on next page)

(продолжение с предыдущей страницы)


```
      "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
      "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
      "3NwJfjG5RpaDfxEhkwXgwD7oX21NMFCxJHL"
   ],
   "fee": 15000000,
   "opType": "add",
  "owners": [
    "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
    "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
  ],
  "type": 113,
 }
```


114. PolicyDataHash

Когда пользователь отправляет конфиденциальные данные в сеть при помощи *POST /privacy/sendData*, нода автоматически формирует транзакцию 114.

POST /transactions/broadcast

Отправляет подписанную транзакцию в блокчейн.

Запрос метода


```
{
 "type":10,
 "senderPublicKey":"G6h72icCSjdW2A89QWDb37hyXJoYKq3XuCUJY2joS3EU",
 "fee":100000000,
 "timestamp":1550591678479,
 "signature":
↪"4gQyPXzJFEzMbsCd9u5n3B2WauEc4172ssyrXCL882oNa8NfNihnpKianHXrHWnZs1RzDLbQ9rcRYnSqxKWfEPJG",
 "alias":"dajzmj6gfuzmbfnhamsbuxivc"
}
```


Ответ метода


```
{
 "type":10,
 "id":"9q7X84wFuVvKqRdDQeWbtBmpsHt9SXFbvPPtUuKBVxxr",
 "sender":"3MtrNP7AkTRuBhX4CBti6iT21pQpEnmHtyw",
 "senderPublicKey":"G6h72icCSjdW2A89QWDb37hyXJoYKq3XuCUJY2joS3EU",
 "fee":100000000,
 "timestamp":1550591678479,
 "signature":
↪"4gQyPXzJFEzMbsCd9u5n3B2WauEc4172ssyrXCL882oNa8NfNihnpKianHXrHWnZs1RzDLbQ9rcRYnSqxKWfEPJG",
 "alias":"dajzmj6gfuzmbfnhamsbuxivc"
}
```

POST /transactions/signAndBroadcast

Подписывает и отправляет подписанную транзакцию в блокчейн.

Запрос метода

```
{
"sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
"policyName": "Policy# 7777",
"password":"sfgKYBFCF@#$fsdf()*%",
"recipients": [
  "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
  "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
      "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
      "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx"
],
"fee": 15000000,
"description": "Buy bitcoin by 1c",
"owners": [
  "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
  "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
],
"type": 112
}
```

Ответ метода

```
{
  "senderPublicKey": "3X6Qb6p96dY4drVt3x4XyHKCRvree4QDqNZyDWHzjJ79",
  "policyName": "Policy for sponsored v1",
  "fee": 100000000,
  "description": "Privacy for sponsored",
  "owners": [
     "3JSaKNX94deXJkywQwTFgbigTxJa36TDVg3",
     "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwD9t"
  ],
  "type": 112,
  "version": 2,
  "sender": "3JSaKNX94deXJkywQwTFgbigTxJa36TDVg3",
  "feeAssetId": "G16FvJk9vabwxjQswh9CQAhbZzn3QrwqWjwnZB3qNVox",
  "proofs": [
     "3vDVjp6UJeN9ahtNcQWt5WDVqC9KqdEsrr9HTToHfoXFd1HtVwnUPPtJKM8tAsCtby81XYQReLj33hLEZ8qbGA3V"
  ],
  "recipients": [
     "3JSaKNX94deXJkywQwTFgbigTxJa36TDVg3",
     "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwD9t"
  ],
  "id": "EyymzQcM2LrsgGDFFxeGn8DhahJbFYmorcBrEh8phv5S",
  "timestamp": 1585307711344
}
```

20.1.19 Utils

Подсказка: Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

POST /utils/hash/secure

Возвращает secure (двойной) hash от заданного сообщения.

Запрос метода:

```
ridethewaves!
```

Ответ метода:

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

POST /utils/hash/fast

Возвращает hash от заданного сообщения.

Запрос метода:

```
ridethewaves!
```

Ответ метода:

```
{
  "message": "ridethewaves!",
  "hash": "DJ35ymschUFDmqCnDJewjcnVExVkWgX7mJDXhFy9X8oQ"
}
```

POST /utils/script/compile

Параметры ответа:

```
"script" - Base64 script
"complexity" - script complexity
"extraFee" - the fee for outgoing transactions set by the script
```

Запрос метода:

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

Ответ метода:

```
{
  "script":
↪"3rbFDtbPwAvSp2vBvqGfGR9nRS1nBVnfuSCN3HxSZ7fVRpt3tuFG5JSmyTmvHPxYf34SocMRkRKFgzTtXXnnv7upRHXJzZrLSQo8tUW6yMtEiZ
↪",
  "complexity": 11,
  "extraFee": 10001
}
```


или

Запрос метода:

```
x == 1
```

Ответ метода:


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


POST /utils/script/estimate

Декодирование base64 скрипта.

Запрос метода:

```
AQQAAAABeAAAAAAAAAAAAQkAAAAAAAACCQAAZAAAAAIFAAAAAXgAAAAAAAAAAAEAAAAAAAAAAAJdecYi
```

Ответ метода:


```
{
  "script":
↪"3rbFDtbPwAvSp2vBvqGfGR9nRS1nBVnfuSCN3HxSZ7fVRpt3tuFG5JSmyTmvHPxYf34SocMRkRKFgzTtXXnnv7upRHXJzZrLSQo8tUW6yMtEiZ
↪",
  "scriptText": "FUNCTION_CALL(FunctionHeader(==,List(LONG, LONG)),List(CONST_LONG(1), CONST_
↪LONG(2)),BOOLEAN)",
  "complexity": 11,
  "extraFee": 10001
}
```


GET /utils/time

Возвращает текущее время на ноде.

Ответ метода:


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

POST /utils/reload-wallet

Перезагружает keystore ноды. Выполняется, если новая ключевая пара была создана в keystore без перезапуска ноды.

Ответ метода:

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

20.2 Методы REST API сервиса авторизации

Подробно о работе с REST API можно почитать в *этом* разделе. Доступ к REST API сервиса авторизации осуществляется по протоколу HTTPS. Методы, закрытые авторизацией, отмечены значком  .

20.2.1 GET /status

Получение статуса сервиса авторизации.

Ответ метода

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

20.2.2 POST /v1/user

Регистрация нового пользователя.

Запрос метода

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

Ответ метода

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

20.2.3 GET /v1/user/profile

Получение данных пользователя.

Ответ метода


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


20.2.4 POST /v1/user/address

Получение адреса пользователя.

Запрос метода


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


Ответ метода


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


20.2.5 GET /v1/user/doesEmailExist

Проверка адреса электронной почты пользователя.

Ответ метода


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

20.2.6 POST /v1/user/password/restore

Восстановление пароля доступа к аккаунту пользователя.

Запрос метода

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

Ответ метода

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

20.2.7 POST /v1/user/password/reset

Сброс пароля пользователя.

Запрос метода

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

Ответ метода

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

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

Ввод кода подтверждения для восстановления пароля для доступа к аккаунту пользователя.

20.2.9 POST /v1/user/resendEmail

Повторная отправка кода восстановления пароля на указанный электронный адрес.

Запрос метода

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

Ответ метода

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

20.2.10 POST /v1/auth/login

Регистрация нового пользователя в сервисе авторизации.

Запрос метода

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

Ответ метода

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

20.2.11 POST /v1/auth/token

Регистрация внешних сервисов и приложений в сервисе авторизации.

Запрос метода

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

Ответ метода

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

20.2.12 POST /v1/auth/refresh

Получение нового refresh токена.

Запрос метода

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

Ответ метода

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


20.2.13 GET /v1/auth/publicKey

Получение публичного ключа сервиса авторизации.

Ответ метода


```
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEA7d9Oj/ZQTkkjf4UuMfUu
QIFDTYxYf6QBKMVJnq/wXyPYYkV8HVFYFizCaEciv3CXmBH77sXnuTlrEtvK7zHB
KvV87OHmZuazjIgZVSkOnOY7F8UUVNXnlzVD1dPsOGJ6orM41DnC1W65mCrP3bjn
fV4RbmykN/lk7McA6EsMcLEGbKkFhmeq2Nk4hn2CQvoTkupJUnOCP1dhO4bq1lQ7
Ffj9K/FJq73wSXDoH+qqdRG9sfrtgrhtJHerruhv3456e0zyAcDO8+sJUQFKY8OB
SZMEndVzFS2ub9Q8e7BfcNxTmQPM4PhHO5wuTqL32qt3uJBx2OI4lu3OND44ZrDJ
BbVog73oPjRYXj+kTbwUZI66SP4aLcQ8sypQyLwqKk5DtLRozSNOOIrupJJ/pwZs
9zPEggL91T0rirbEhGlf5U8/6XN8GVXX4iMk2fD8FHLFJuXCD7Oj4JC2iWfFDC6a
uUkwUfqfjJB8BzIHkncoqOZbpidEE2lTWl+svuEu/wyP5rNlyMiE/e/fZQqM2+oO
cH5Qow6HH35BrloCSZciutUcd1U7YPqESJ5tryy1xn9bsMb+On1ocZTtvec/ow4M
RmnJwmOj1nd+cc19OKLG5/boeA+2zqWuOjCbWR9c0oCmgbhuqZCHaHTBEAKDWcsC
VRz5qD6FPpePpTQDb6ss3bkCAwEAAQ==
-----END PUBLIC KEY-----
```


20.3 Методы REST API сервиса подготовки данных

20.3.1 Транзакции

GET /transactions

Возвращает список транзакций, соответствующий условиям поискового запроса и применённым фильтрам.

Важно: За один запрос через метод API **GET /transactions** возвращается не более 500 транзакций.

Ответ метода:


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


(continues on next page)

(продолжение с предыдущей страницы)

```
 }
]
```

GET /transactions/count

Возвращает количество транзакций, соответствующих условиям поискового запроса и применённым фильтрам.

Ответ метода:

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

GET /transactions/id/{id}

Возвращает транзакцию по идентификатору {id}.

Ответ метода:

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

20.3.2 Наборы токенов

GET /assets

Возвращает список доступных в блокчейне наборов токенов (в виде транзакций выпуска токенов).

Ответ метода:

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

(continues on next page)

(продолжение с предыдущей страницы)

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

20.3.3 Пользователи

GET /users

Возвращает список пользователей, соответствующий условиям поискового запроса и применённым фильтрам.

Ответ метода:

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

GET /users/{userAddress}

Возвращает информацию о пользователе по его адресу.

Ответ метода:

```
{
"address": "string",
"aliases": [
    "string"
],
"registration_date": "string",
"permissions": [
    "string"
],
"balances": [
    {
```

(continues on next page)

(продолжение с предыдущей страницы)


```
      "assetId": "string",
    "amount": 0
    }
]
}
```


20.3.4 Блоки

GET /blocks/{height}

Возвращает блок на указанной высоте.

Ответ метода:


```
{
 "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
     }
 ]
 }
```

20.3.5 Транзакции с данными

GET /api/v1/txIds/{key}

Возвращает список идентификаторов транзакций с данными, содержащих указанный ключ.

Ответ метода:

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

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

Возвращает список идентификаторов транзакций с данными, содержащих указанный ключ и значение.

Ответ метода:

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

GET /api/v1/txData/{key}

Возвращает тела транзакций с данными, содержащие указанный ключ.

Ответ метода:

```
[
{
 "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}

Возвращает тела транзакций с данными, содержащие указанный ключ и значение.

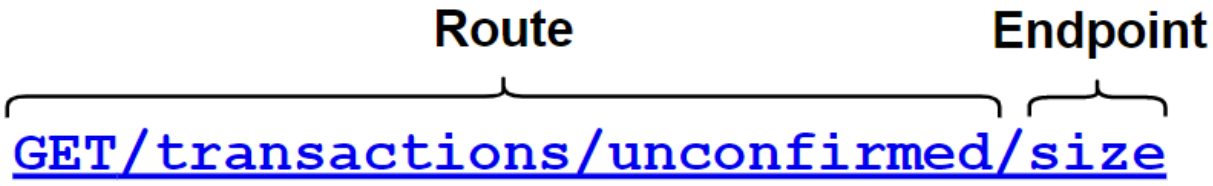
Ответ метода:

```
[
{
 "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
}
]
```

20.4 Как использовать REST API

Все вызовы методов API — это GET, POST или DELETE HTTPS-запросы к URL `https://yournetwork.com/node-N/api-docs/swagger.json` с набором параметров. В интерфейсе Swagger выбираются нужные группы запросов и далее маршруты с точками доступа. Маршрут в Swagger это URL к HTTP-методу, а точка доступа (endpoint) - конечная часть маршрута, само обращение к методу. Пример:

URL к HTTP-методу



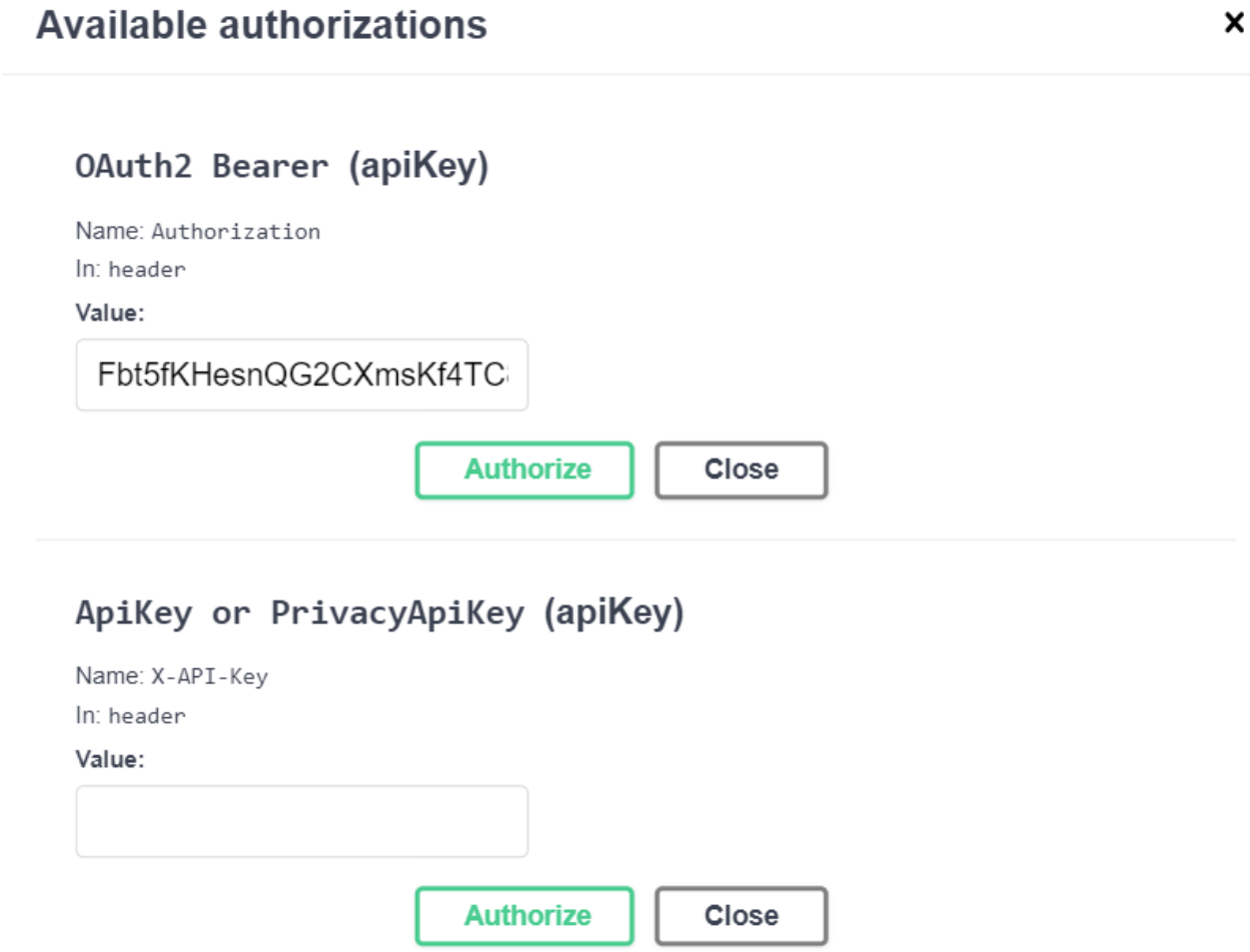
Для запросов, требующих нижеперечисленных действий, необходима обязательная авторизация по `api-key-hash`. Тип авторизации устанавливается в конфигурационном файле ноды. Если выбран тип авторизации по `api-key-hash`, то при авторизации необходимо указывать значение секретной фразы, hash которой указан в конфигурационном файле ноды (поле `rest-api.api-key-hash`).

- доступ к keystore ноды (например, метод sign);
- доступ к операциям с группами доступа к приватным данным;
- доступ к конфигурации ноды.

При авторизации по токену в соответствующем поле указывается значение **access** токена. Если выбрана авторизация по токену, в таком случае закрыты все методы REST API для доступа к ноде.

20.5 Методы авторизации

В зависимости от метода авторизации указываются разные значения для получения доступа к REST API ноды.



- `OAuth2 Bearer (apiKey)` - значение **access** токена.
- `ApiKey or PrivacyApiKey (apiKey)` - значение `api-key-hash` как для общего доступа к REST API ноды, так и для доступа к методам *privacy*.

20.5.1 Авторизация по `api-key-hash`

Генерация значения `api-key-hash` выполняется при *конфигурации ноды*. Также получить значение поля `rest-api.api-key-hash` можно при помощи метода */utils/hash/secure* REST API ноды. Для подписания запросов ключем из keystore ноды в поле `password` запроса `POST /transaction/sign` требуется указания пароля доступа к keystore.

Пример запроса:

```
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'
```

20.5.2 Авторизация по токену

Если используется *сервис авторизации*, для доступа к ноде и другим сервисам клиент получает пару токенов - **refresh** и **access**. Токены можно получить через REST API сервиса авторизации.

ГЛАВА 21

Смарт-контракты Docker

21.1 Смарт-контракты Docker с использованием REST API ноды

Подсказка: Техническое описание особенностей реализации контрактов приведено в разделе *Смарт-контракты Docker*.

21.1.1 Описание логики программы

В разделе рассматривается пример создания и запуска простого смарт-контракта. Контракт выполняет инкремент переданного на вход числа при каждой *call-транзакции*.

Листинг программы `contract.py` на 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)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    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}
    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))
```

Описание работы

- Программа ожидает получить структуру данных в формате json с полем «params».
- Считывает значение поля «a».
- Возвращает результат в виде значения поля «{a}+1» в формате json.

Пример входящих параметров

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

21.1.2 Установка смарт-контракта

1. Скачать и установить Docker for Developers для вашей операционной системы.
2. Подготовить образ контракта. В папке `stateful-increment-contract` создать следующие файлы:
   - `contract.py`
   - `Dockerfile`
   - `run.sh`

Листинг файла run.sh

```
#!/bin/sh

python contract.py
```

Листинг файла Dockerfile

```
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"
```

Важно: В контейнер со смарт-контрактом необходимо установить iptables.

3. Установить образ в Docker registry. Выполнить в терминале следующие команды:

```
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. Для получения информации о контейнере выполнить в терминале следующую команду:

```
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,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        "AttachStdout": false,
        "AttachStderr": false,
```

Идентификатор смарт-контракта `Id` является значением поля `imageHash` для использования в транзакциях с созданным смарт-контрактом.

5. Подписать транзакцию на создание смарт-контракта. В рассматриваемом примере транзакция подписывается ключом, сохраненным в keystore ноды.

Подсказка: Для создания ключевой пары и адреса участника используется утилита generators.jar. Порядок действий создания ключевой пары приведен в подразделе *Генерирование ключевых пар*. Правила формирования запросов к ноде приведены в разделе *REST API ноды*.

Тело запроса

```
{
  "fee": 100000000,
  "image": "stateful-increment-contract:latest",
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
  "contractName": "stateful-increment-contract",
  "sender": "3PudkbvjV1nPj1TkuuRahh4sGdgfr4YAUV2",
  "password": "",
  "params": [],
  "type": 103,
  "version": 1
 }
```

Пример запроса

```
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": "3PudkbvjV1nPj1TkuuRahh4sGdgfr4YAUV2", \
        "password": "", \
        "params": [], \
        "type": 103, \
        "version": 1 \
    }' 'http://localhost:6862/transactions/sign'
```

Пример ответа

```
{
    "type": 103,
    "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
    "fee": 500000,
    "timestamp": 1550591678479,
    "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
    "version": 1,
    "image": "stateful-increment-contract:latest",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
    "contractName": "stateful-increment-contract",
    "params": [],
    "height": 1619
}
```

6. Отправить подписанную транзакцию в блокчейн. Ответ от метода sign необходимо передать на вход для метода broadcast.

Пример запроса

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --
↪header 'X-Contract-Api-Token' -d '{ \
{
    "type": 103, \
    "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky", \
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew", \
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M", \
    "fee": 500000, \
    "timestamp": 1550591678479, \
    "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ], \
    "version": 1, \
    "image": "stateful-increment-contract:latest", \
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65", \
    "contractName": "stateful-increment-contract", \
    "params": [], \
    "height": 1619 \
}
}' 'http://localhost:6862/transactions/broadcast'
```

7. По id транзакции убедиться, что транзакция с инициализацией контракта размещена в блокчейне.

Пример ответа

```
{
    "type": 103,
    "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
    "fee": 500000,
    "timestamp": 1550591678479,
    "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
    "version": 1,
    "image": "stateful-increment-contract:latest",
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
    "contractName": "stateful-increment-contract",
    "params": [],
    "height": 1619
}
```

21.1.3 Исполнение смарт-контракта

1. Подписать call-транзакцию на вызов (исполнение) смарт-контракта.

В поле `contractId` указать идентификатор транзакции инициализации контракта.

Тело запроса

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

Пример запроса

```
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'
```

Пример ответа

```
{
    "type": 104,
    "id": "9fBrL2n5TN473g1gNfoZqaAqAsAJCuHRHYxZpLexL3VP",
    "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
    "senderPublicKey": "2YvzcVLrqLCqouVrFZynjfotEuPNV9GrdauNpgdWXLsq",
    "fee": 10,
    "timestamp": 1549365736923,
    "proofs": [
        "2q4cTBhDkEDkFxr7iYaHPAv1dzaKo5rDaTxPF5VHryyYTXxTPvN9Wb3YrsDYixKiUPXBnAyXzEcnKPFRCW9xVp4v"
    ],
    "version": 1,
    "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
    "params": [
```

(continues on next page)

(продолжение с предыдущей страницы)

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

2. Отправить подписанную транзакцию в блокчейн. Ответ от метода sign необходимо передать на вход для метода broadcast.

Пример запроса

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

3. Получить результат выполнения смарт-контракта по его идентификатору.

Пример ответа

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

21.2 Методы API, доступные смарт-контракту

Смарт-контракты на базе контейнеров Docker могут использовать *REST API ноды*. Разработчикам Docker смарт-контрактов доступны не все методы REST API. Ниже приведен список методов REST API ноды, которые смарт-контракт может использовать прямо из Docker контейнера.

Методы Addresses

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

Методы Crypto

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

Методы Privacy

- *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*

Методы Transactions

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

Методы Contracts

Для улучшения производительности смарт-контракт может использовать методы *Contracts* по выделенному маршруту `/internal/contracts/`, которые полностью идентичны обычным методам *Contracts*.

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

Методы PKI

- *PKI /verify*

21.2.1 Авторизация Docker смарт-контракта

Для работы с *REST API ноды* смарт-контракту необходима авторизация. Чтобы Docker-контракт корректно работал с методами API, выполняются следующие действия:

1. В переменных окружения Docker-контракта должны быть определены следующие переменные:

- `NODE_API` - URL-адрес к *REST API ноды*.
- `API_TOKEN` - токен авторизации для Docker-контракта.
- `COMMAND` - команды для создания и вызова Docker-контракта.
- `TX` - транзакция, необходимая Docker-контракту для работы (коды *103* - *107*).

2. Разработчик Docker-контракта присваивает значение переменной `API_TOKEN` заголовку запроса `X-Contract-Api-Token`. В переменную `API_TOKEN` нода прописывает JWT токен авторизации при создании и выполнении контракта.
3. Код контракта должен передавать полученный токен в заголовке запроса (`X-Contract-Api-Token`) при каждом обращении к API ноды.

21.3 Смарт-контракты Docker с использованием gRPC

Помимо использования REST API смарт-контракт может работать с нодой через фреймворк gRPC. gRPC — это высокопроизводительный фреймворк для вызов удаленных процедур (RPC), который работает поверх HTTP/2. В качестве инструмента описания типов данных и сериализации используется протокол Protobuf.

Подсказка: Техническое описание особенностей реализации контрактов приведено в разделе *Смарт-контракты Docker*.

Официально фреймворк gRPC поддерживает 10 языков программирования. Список языков вы можете найти в официальной документации gRPC. Рассмотрим пример создания смарт-контракта на Python, который выполняет операцию инкремента (увеличение заданного числа на единицу).

21.3.1 Описание работы смарт-контракта

В нашем примере транзакция *103* для создания контракта инициализирует начальное состояние контракта, сохраняя в нем числовой ключ `sum` со значением `0`:

```
{
    "key": "sum",
    "type": "integer",
    "value": 0
}
```

Каждая следующая транзакция вызова *104* увеличивает значение ключа `sum` на единицу (т.е. `sum = sum + 1`).

Как работает смарт-контракт после вызова:

1. После старта программы выполняется проверка на наличие переменных окружения. Переменные окружения, используемые контрактом:

- `CONNECTION_ID` – идентификатор соединения, передаваемый контрактом при соединении с нодой.
- `CONNECTION_TOKEN` – токен авторизации, передаваемый контрактом при соединении с нодой.
- `NODE` – ip-адрес или доменное имя ноды.
- `NODE_PORT` – порт gRPC сервиса, развёрнутого на ноде.

Значения переменных `NODE` и `NODE_PORT` берутся из конфигурационного файла ноды секции *docker-engine.grpc-server*. Остальные переменные генерируются нодой и передаются в контейнер при создании смарт контракта.

2. Используя значения переменных окружения `NODE` и `NODE_PORT`, контракт создает gRPC-подключение с нодой.
3. Далее вызывается потоковый метод `Connect` gRPC сервиса `ContractService` (см. *contract.proto* файл). Метод принимает параметр `ConnectionRequest`, в котором указывается идентификатор соединения (полученный из переменной окружения `CONNECTION_ID`). В метаданных метода указывается заголовок `authorization` со значением токена авторизации (полученного из переменной окружения `CONNECTION_TOKEN`).
4. В случае успешного вызова метода возвращается gRPC поток (`stream`) с объектами типа `ContractTransactionResponse` для исполнения. Объект `ContractTransactionResponse` содержит два поля:
   - `transaction` – транзакция создания или вызова контракта.
   - `auth_token` – токен авторизации, указываемый в заголовке `authorization` метаданных вызываемого метода gRPC сервисов.

   Если `transaction` содержит транзакцию создания (тип транзакции – *103*), то для контракта инициализируется начальное состояние. Если `transaction` содержит транзакцию вызова (тип транзакции – *104*), то выполняются следующие действия:
   - с ноды запрашивается значение ключа `sum` (метод `GetContractKey` сервиса `ContractService`);
   - значение ключа увеличивается на единицу, т.е. `sum = sum + 1`);
   - новое значение ключа сохраняется на ноде (метод `CommitExecutionSuccess` сервиса `ContractService`), т.е. происходит обновление состояния контракта.

21.3.2 Создание смарт-контракта

1. Скачайте и установите Docker for Developers (https://www.docker.com/get-started) для вашей операционной системы.
2. Подготовьте образ контракта. В папке с контрактом должны быть следующие файлы:
   - `src/contract.py`
   - `Dockerfile`
   - `run.sh`
   - `src/protobuf/contract.proto`
   - `src/protobuf/common.proto`
   - `src/protobuf/common_pb2.py`
   - `src/protobuf/contract_pb2.py`

- `src/protobuf/contract_pb2_grpc.py`

Файлы `src/protobuf/common_pb2.py`, `src/protobuf/contract_pb2.py`, `src/protobuf/contract_pb2_grpc.py` должны быть сгенерированы gRPC-компилятором из protobuf файлов `contract.proto` и `common.proto`. Описание процедуры генерации программных файлов из protobuf файлов вы моете найти на официальной странице gRPC.

Важно: После компиляции файлов необходимо поменять `import` директиву в сгенерированных файлах:

- в файле `contract_pb2.py` должно быть `import protobuf.common_pb2 as common__pb2`;
- в файле `contract_pb2_grpc.py` должно быть `import protobuf.contract_pb2 as contract__pb2`.

---

3. Установите образ в Docker репозиторий образов. Если используете локальный репозиторий, выполните в терминале следующие команды:

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

4. Для получения информации о смарт-контракте используйте команду `docker inspect`:

```
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,
```

Важно: Идентификатор Docker образа контракта `Id` является значением поля `imageHash` для использования в `CreateContractTransaction` транзакциях с созданным смарт-контрактом.

---

5. Подпишите транзакцию *103* на создание смарт-контракта. В нашем примере транзакция подписывается ключом, сохраненным в keystore ноды. Описание REST API ноды и правила формирования транзакций приведены в разделе *REST API*.

Пример запроса для транзакции создания смарт-контракта:

```
{
    "fee": 100000000,
    "image": "localhost:5000/grpc-increment-contract",
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
    "contractName": "grpc-increment-contract",
    "sender": "3PudkbvjV1nPj1TkuuRahh4sGdgfr4YAUV2",
    "password": "",
    "params": [],
    "type": 103,
    "version": 2,
}
```

Пример curl-запроса:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --
↪header 'X-Contract-Api-Token' -d '{ \
    "fee": 100000000, \
    "image": "localhost:5000/grpc-increment-contract", \
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65", \
    "contractName": "grpc-increment-contract", \
    "sender": "3PudkbvjV1nPj1TkuuRahh4sGdgfr4YAUV2", \
    "password": "", \
    "params": [], \
    "type": 103, \
    "version": 2 \
}' 'http://localhost:6862/transactions/sign'
```

Пример ответа:

```
{
    "type": 103,
    "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
    "fee": 100000000,
    "timestamp": 1550591678479,
    "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
    "version": 2,
    "image": "localhost:5000/grpc-increment-contract",
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
    "contractName": "grpc-increment-contract",
    "params": [],
    "height": 1619
}
```

6. Отправьте подписанную транзакцию в блокчейн. Ответ от метода *sign* необходимо передать на вход для метода *broadcast*.

Пример запроса на отправку транзакции создания смарт-контракта в блокчейн:

```
{
    "type": 103,
    "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
    "fee": 500000,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    "timestamp": 1550591678479,
    "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
    "version": 1,
    "image": "stateful-increment-contract:latest",
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
    "contractName": "stateful-increment-contract",
    "params": [],
    "height": 1619
}
```

Пример curl-запроса:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --
↪header 'X-Contract-Api-Token' -d '{ \
    "type": 103, \
    "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky", \
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew", \
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M", \
    "fee": 100000000, \
    "timestamp": 1550591678479, \
    "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ], \
    "version": 2, \
    "image": "localhost:5000/grpc-increment-contract", \
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65", \
    "contractName": "grpc-increment-contract", \
    "params": [], \
    "height": 1619 \
}' 'http://localhost:6862/transactions/broadcast'
```

Пример ответа:

```
{
    "type": 103,
    "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
    "fee": 100000000,
    "timestamp": 1550591678479,
    "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
    "version": 2,
    "image": "localhost:5000/grpc-increment-contract",
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
    "contractName": "grpc-increment-contract",
    "params": [],
    "height": 1619
}
```

Сравните идентификатор транзакции в обеих операциях (поле `id`) и убедитесь, что транзакция с инициализацией контракта размещена в блокчейне.

21.3.3 Вызов смарт-контракта

1. Подпишите транзакцию *104* на вызов смарт-контракта.

Пример запроса для транзакции вызова смарт-контракта:

```
{
    "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
    "fee": 15000000,
    "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
    "password": "",
    "type": 104,
    "version": 2,
    "contractVersion": 1,
    "params": []
}
```

2. Отправьте подписанную транзакцию в блокчейн. Ответ от метода *sign* необходимо передать на вход для метода *broadcast*.

Пример запроса на отправку транзакции вызова смарт-контракта в блокчейн:

```
{
    "type": 104,
    "id": "9fBrL2n5TN473g1gNfoZqaAqAsAJCuHRHYxZpLexL3VP",
    "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
    "senderPublicKey": "2YvzcVLrqLCqouVrFZynjfotEuPNV9GrdauNpgdWXLsq",
    "fee": 15000000,
    "timestamp": 1549365736923,
    "proofs": [
        "2q4cTBhDkEDkFxr7iYaHPAv1dzaKo5rDaTxPF5VHryyYTXxTPvN9Wb3YrsDYixKiUPXBnAyXzEcnKPFRCW9xVp4v"
    ],
    "version": 1,
    "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
    "params": []
}
```

Пример curl-запроса:

```
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": 15000000, \
    "timestamp": 1549365736923, \
    "proofs": [ \
        "2q4cTBhDkEDkFxr7iYaHPAv1dzaKo5rDaTxPF5VHryyYTXxTPvN9Wb3YrsDYixKiUPXBnAyXzEcnKPFRCW9xVp4v"␣
↪\
    ], \
    "version": 1, \
    "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2", \
    "params": [] \
}' 'http://localhost:6862/transactions/broadcast'
```

Пример ответа:

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

Получите результат выполнения смарт-контракта по его идентификатору.

21.3.4 Примеры файлов

Листинг run.sh:

```
#!/bin/sh

eval $SET_ENV_CMD
python contract.py
```

Листинг Dockerfile:

```
FROM python:3.8-slim-buster
RUN apt install iptables
RUN apt update && apt install -yq dnsutils
RUN pip3 install grpcio-tools
ADD src/contract.py /
ADD src/protobuf/common_pb2.py /protobuf/
ADD src/protobuf/contract_pb2.py /protobuf/
ADD src/protobuf/contract_pb2_grpc.py /protobuf/
ADD run.sh /
RUN chmod +x run.sh
ENTRYPOINT ["/run.sh"]
```

Листинг смарт-контракта на Python:

```
import grpc
import os
import sys

from protobuf import common_pb2, contract_pb2, contract_pb2_grpc

CreateContractTransactionType = 103
CallContractTransactionType = 104

AUTH_METADATA_KEY = "authorization"

class ContractHandler:
    def __init__(self, stub, connection_id):
        self.client = stub
        self.connection_id = connection_id
        return

    def start(self, connection_token):
        self.__connect(connection_token)

    def __connect(self, connection_token):
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        request = contract_pb2.ConnectionRequest(
            connection_id=self.connection_id
        )
        metadata = [(AUTH_METADATA_KEY, connection_token)]
        for contract_transaction_response in self.client.Connect(request=request,␣
↪metadata=metadata):
            self.__process_connect_response(contract_transaction_response)

    def __process_connect_response(self, contract_transaction_response):
        print("receive: {}".format(contract_transaction_response))
        contract_transaction = contract_transaction_response.transaction
        if contract_transaction.type == CreateContractTransactionType:
            self.__handle_create_transaction(contract_transaction_response)
        elif contract_transaction.type == CallContractTransactionType:
            self.__handle_call_transaction(contract_transaction_response)
        else:
            print("Error: unknown transaction type '{}'".format(contract_transaction.type),␣
↪file=sys.stderr)

    def __handle_create_transaction(self, contract_transaction_response):
        create_transaction = contract_transaction_response.transaction
        request = contract_pb2.ExecutionSuccessRequest(
            tx_id=create_transaction.id,
        r   esults=[common_pb2.DataEntry(
                    key="sum",
                    int_value=0)]
        )
        metadata = [(AUTH_METADATA_KEY, contract_transaction_response.auth_token)]
        response = self.client.CommitExecutionSuccess(request=request, metadata=metadata)
        print("in create tx response '{}'".format(response))

    def __handle_call_transaction(self, contract_transaction_response):
        call_transaction = contract_transaction_response.transaction
        metadata = [(AUTH_METADATA_KEY, contract_transaction_response.auth_token)]

        contract_key_request = contract_pb2.ContractKeyRequest(
            contract_id=call_transaction.contract_id,
            key="sum"
        )
        contract_key = self.client.GetContractKey(request=contract_key_request, metadata=metadata)
        old_value = contract_key.entry.int_value

        request = contract_pb2.ExecutionSuccessRequest(
            tx_id=call_transaction.id,
            results=[common_pb2.DataEntry(
                key="sum",
                int_value=old_value + 1)]
        )
        response = self.client.CommitExecutionSuccess(request=request, metadata=metadata)
        print("in call tx response '{}'".format(response))

def run(connection_id, node_host, node_port, connection_token):
    # NOTE(gRPC Python Team): .close() is possible on a channel and should be
    # used in circumstances in which the with statement does not fit the needs
    # of the code.
    with grpc.insecure_channel('{}:{}'.format(node_host, node_port)) as channel:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        stub = contract_pb2_grpc.ContractServiceStub(channel)
        handler = ContractHandler(stub, connection_id)
        handler.start(connection_token)

CONNECTION_ID_KEY = 'CONNECTION_ID'
CONNECTION_TOKEN_KEY = 'CONNECTION_TOKEN'
NODE_KEY = 'NODE'
NODE_PORT_KEY = 'NODE_PORT'

if __name__ == '__main__':
    if CONNECTION_ID_KEY not in os.environ:
        sys.exit("Connection id is not set")
    if CONNECTION_TOKEN_KEY not in os.environ:
        sys.exit("Connection token is not set")
    if NODE_KEY not in os.environ:
        sys.exit("Node host is not set")
    if NODE_PORT_KEY not in os.environ:
        sys.exit("Node port is not set")

    connection_id = os.environ['CONNECTION_ID']
    connection_token = os.environ['CONNECTION_TOKEN']
    node_host = os.environ['NODE']
    node_port = os.environ['NODE_PORT']

    run(connection_id, node_host, node_port, connection_token)
```

Листинг contract.proto:

```
syntax = "proto3";
package wavesenterprise;

option java_multiple_files = true;
option java_package = "com.wavesplatform.protobuf.service";
option csharp_namespace = "WavesEnterprise";

import "google/protobuf/wrappers.proto";
import "common.proto";

service ContractService {

  rpc Connect (ConnectionRequest) returns (stream ContractTransactionResponse);

  rpc CommitExecutionSuccess (ExecutionSuccessRequest) returns (CommitExecutionResponse);

  rpc CommitExecutionError (ExecutionErrorRequest) returns (CommitExecutionResponse);

  rpc GetContractKeys (ContractKeysRequest) returns (ContractKeysResponse);

  rpc GetContractKey (ContractKeyRequest) returns (ContractKeyResponse);
}

message ConnectionRequest {
  string connection_id = 1;
}

message ContractTransactionResponse {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  ContractTransaction transaction = 1;
  string auth_token = 2;
}

message ContractTransaction {
  string id = 1;
  int32 type = 2;
  string sender = 3;
  string sender_public_key = 4;
  string contract_id = 5;
  repeated DataEntry params = 6;
  int64 fee = 7;
  int32 version = 8;
  bytes proofs = 9;
  int64 timestamp = 10;
  AssetId fee_asset_id = 11;

  oneof data {
    CreateContractTransactionData create_data = 20;
    CallContractTransactionData call_data = 21;
  }
}

message CreateContractTransactionData {
  string image = 1;
  string image_hash = 2;
  string contract_name = 3;
}

message CallContractTransactionData {
  int32 contract_version = 1;
}

message ExecutionSuccessRequest {
  string tx_id = 1;
  repeated DataEntry results = 2;
}

message ExecutionErrorRequest {
  string tx_id = 1;
  string message = 2;
}

message CommitExecutionResponse {
}

message ContractKeysRequest {
  string contract_id = 1;
  google.protobuf.Int32Value limit = 2;
  google.protobuf.Int32Value offset = 3;
  google.protobuf.StringValue matches = 4;
  KeysFilter keys_filter = 5;
}

message KeysFilter {
  repeated string keys = 1;
```

(continues on next page)

(продолжение с предыдущей страницы)

```
}

message ContractKeysResponse {
  repeated DataEntry entries = 1;
}

message ContractKeyRequest {
  string contract_id = 1;
  string key = 2;
}

message ContractKeyResponse {
  DataEntry entry = 1;
}

message AssetId {
  string value = 1;
}
```

Листинг common.proto:

```
syntax = "proto3";
package wavesenterprise;

option java_multiple_files = true;
option java_package = "com.wavesplatform.protobuf.common";
option csharp_namespace = "WavesEnterprise";

message DataEntry {
  string key = 1;
  oneof value {
    int64 int_value = 10;
    bool bool_value = 11;
    bytes binary_value = 12;
    string string_value = 13;
  }
}
```

21.4 Сервисы gRPC, используемые смарт-контрактом

Protobuf файлы, необходимые для работы с нодой через gRPC, вы можете скачать со страницы проекта в GitHub. Список protobuf файлов следующий:

- `address.proto` - методы для работы с адресами.
- `common.proto` - общий файл для работы других protobuf файлов.
- `crypto.proto` - методы для работы с шифрованием данных.
- `permission.proto` - методы для работы с выдачей разрешений.
- `pki.proto` - методы работы с PKI.
- `privacy.proto` - методы работы с приватными данными.
- `util.proto` - методы для служебных утилит.

Каждый protobuf файл (кроме `common.proto`) содержит набор небольших блоков, включающих набор полей «ключ-значение». Список таких блоков для каждого файла приведен ниже.

address.proto

- `GetAddresses` - получение всех адресов участников, ключевые пары которых хранятся в keystore ноды.
- `GetAddressData` - получение всех данных, записанных на аккаунт адресата {address}.

contract.proto

- `Connect` - подключение контракта к ноде.
- `CommitExecutionSuccess` - получение результата успешного исполнения контракта и отправка результатов на ноду.
- `CommitExecutionError` - получение ошибки исполнения контракта и отправка результатов на ноду.
- `GetContractKeys` - получение результата исполнения контракта по его идентификатору (id транзакции создания контракта).
- `GetContractKey` - получение результата исполнения контракта по его идентификатору (id транзакции создания контракта) и ключу {key}.

crypto.proto

- `EncryptSeparate` - шифрование данных отдельно для каждого получателя уникальным ключом.
- `EncryptCommon` - шифрование данных единым ключом CEK для всех получателей, CEK оборачивается уникальными KEK для каждого получателя.
- `Decrypt` - расшифровка данных. Расшифровка доступна в случае, если ключ получателя сообщения находится в keystore ноды.

permission.proto

- `GetPermissions` - получение списка всех ролей на указанный адрес, действительных на текущий момент.
- `GetPermissionsForAddresses` - получение списка всех ролей, действительных на текущий момент, на указанный диапазон адресов.

pki.proto

- `Sign` - формирование отсоединённой ЭП для данных, передаваемых в запросе.
- `Verify` - проверка отсоединённой ЭП для данных, передаваемых в запросе.

privacy.proto

- `GetPolicyRecipients` - получение адресов всех участников, записанных в группу {policy-id}.
- `GetPolicyOwners` - получение адресов всех владельцев, записанных в группу {policy-id}.
- `GetPolicyHashes` - получение массива идентификационный хешей, которые записаны в привязке к {policy-id}.
- `GetPolicyItemData` - получение пакета конфиденциальных данных по идентификационному хешу.
- `GetPolicyItemInfo` - получение метаданных для пакета конфиденциальных данных по идентификационному хешу.

util.proto

- `GetNodeTime` - получение текущего времени на ноде.

ГЛАВА 22

Управление ролями участников

Список возможных ролей в блокчейн-платформе приведен в разделе *«Авторизация участников»*.

Важно: Обязательным условием для изменения полномочий участников (добавления или удаления ролей) является наличие приватного ключа участника с ролью «permissioner» в keystore ноды, с которой осуществляется запрос.

22.1 Вариант №1 (через REST API)

Управление полномочиями участника выполняется путем подписания (метод sign) и рассылки (метод broadcast) permission-транзакций через *REST API ноды*.

Объект запроса для метода sign:

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

Поля запроса:

- type - тип транзакции для управления полномочиями участников (type = 102);
- sender - адрес участника с полномочиями на выпуск permission-транзакций;
- proofs - подпись транзакции;

- target - адрес участника, для которого требуется установить или удалить полномочия;
- role - полономочия участника, которые требуется установить или удалить. Возможные значения: «miner», «issuer», «dex», «permissioner», «blacklister», «banned», «contract_developer», «connection_manager»;
- opType - тип операции «add» (добавить полномочия) или «remove» (удалить полномочия);
- dueTimestamp - дата действия permission в формате timestamp. Поле является опциональным.

Полученный ответ от ноды передать методу broadcast.

22.2 Вариант №2 (через утилиту)

Используя утилиту Generators возможно автоматизировать процесс.

Пример запуска из консоли:

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

Пример конфига:

```
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"
        }
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    ]
    }
]
txs-per-bucket = 10
}
```

Поле «due-timestamp» ограничивает время действия роли; Поля nodes, roles - обязательные.

Если у ноды уже задана какая-либо из ролей, которая задана в конфиге, то ситуация обрабатывается в соответствии с правилами:

Текущее состояние ноды	Состояние полученное из транзакции	Результат обработки
Роль не назначена	Новая роль	Success - назначается роль
Назначена роль без dueDate	Роль с dueDate	Проверка dueDate, если меньше текущей, то IncorrectDatetime, иначе Success - назначается роль с dueDate
Назначена роль с dueDate	Роль с dueDate	Проверка dueDate, если меньше текущей, то IncorrectDatetime, иначе Success - обновление dueDate
Назначена роль с dueDate	Роль без dueDate	Success - назначается роль без dueDate
Назначена роль с/без dueDate	Удаление роли	Проверка адреса ноды, если <> адресу генезиса, то Success - удаляется роль

ГЛАВА 23

Подключение участников к сети

Момент *запуска* первой ноды можно считать началом создания новой блокчейн-сети. Разворачивать блокчейн-сеть вы можете со старта всего одной ноды, добавляя новые ноды по мере необходимости.

- *Подключите* новую ноду к уже существующей сети.
- *Удалите* лишние ноды из сети.

23.1 Подключение новой ноды к существующей сети

Новые ноды можно добавлять в сеть в любой момент времени. Настройка конфигурационных файлов новой ноды описана в подразделе *Установка и запуск платформы Waves Enterprise*. Выполните все указанные в приведённом подразделе действия и *запустите* ноду. Далее выполняются следующие действия:

1. Пользователь новой ноды передаёт публичный ключ и описание ноды администратору сети.
2. Администратор сети (нода с ролью «Connection-manager») использует полученные публичный ключ и описание ноды при создании транзакции *111 RegisterNode* с параметром `"opType": "add"`.
3. Транзакция попадает в блок и далее в стейты нод участников сети. Вследствие транзакции среди сохраняемых данных каждый участник сети хранит обязательно публичный ключ и адрес новой ноды.
4. При необходимости администратор сети может добавить новой ноде дополнительные роли при помощи транзакции *102 Permit*.
5. Пользователь *запускает* ноду.
6. После запуска нода отправляет *handshake-сообщение* со своим публичным ключом участникам из списка «peers» своего конфигурационного файла.
7. Участники сети сравнивают публичный ключ из *handshake-сообщения* и ключ из транзакции *111 RegisterNode*, отправленной администратором сети. Если проверка успешна, участник сети обновляет свою БД и рассылает в сеть сообщение *Peers Message*.

8. Успешно подключившись, новая нода выполняет синхронизацию с сетью и получает таблицу адресов участников сети.

23.2 Удаление ноды

1. Администратор сети создает транзакцию *111 RegisterNode* с параметром `"opType": "remove"` для удаления ноды из сети, в которую помещается её публичный ключ.
2. Транзакция с удалением ноды вместе с остальными попадает в блок, и её принимают другие ноды.
3. После принятия транзакции ноды находят в своем стейте публичный ключ, указанный в транзакции *111 RegisterNode*, и удаляют его из стейта.
4. Далее ноды удаляют сетевой адрес ноды с ключом, указанным в транзакции *111 RegisterNode*, из списка `network.known-peers` конфигурационного файла ноды.

ГЛАВА 24

Обмен конфиденциальными данными

Перед тем, как обмениваться конфиденциальными данными, вам нужно создать группы доступа. Используя транзакции, вы можете *добавлять* или *изменять* группы доступа к конфиденциальным данным.

24.1 Создание группы доступа к конфиденциальным данным

Группу доступа к конфиденциальным данным может создать любой участник сети. Перед созданием группы необходимо определиться с кругом участников сети, которые будут получать конфиденциальные данные. Далее любой из участников выполняет следующие действия:

1. Участник сети, который будет владельцем группы доступа, создаёт транзакцию *112 CreatePolicy* со следующими основными параметрами:

- sender - публичный ключ создателя группы доступа.
- description - описание группы доступа.
- policyName - имя группы доступа.
- recipients - публичные ключи участников группы доступа, которые будут иметь право получать конфиденциальные данные.
- owners - публичные ключи владельцев группы доступа, которые, помимо доступа к данным, смогут изменять состав участников группы.

2. Транзакция с созданием группы доступа вместе с остальными попадает в блок, и её принимают другие ноды.
3. После принятия транзакции доступ к отправляемым в сеть конфиденциальным данным получают все участники, зарегистрированные в созданной группе доступа.

24.2 Изменение группы доступа

Изменять группы доступа могут только их владельцы. Выполняются следующие действия для изменения списка участников в группе доступа:

1. Владелец группы доступа создаёт транзакцию *113 UpdatePolicy* со следующими основными параметрами:

- policyId - идентификатор группы доступа;
- sender - публичный ключ владельца группы доступа;
- opType - опция добавления (`add`) или удаления (`remove`) участников группы;
- recipients - публичные ключи участников группы доступа, которые добавляются или удаляются из группы доступа;
- owners - публичные ключи владельцев группы доступа, которые добавляются или удаляются из группы доступа.

2. Транзакция с изменением группы доступа вместе с остальными попадает в блок, и её принимают другие ноды.
3. После принятия транзакции в сети обновляется информация об участниках изменённой группы.

24.3 Процесс обмена конфиденциальными данными

Важно: Через метод API *POST /privacy/sendData* можно отправить данные размером не более 20МБ.

1. Пользователь отправляет данные в сеть, используя инструмент API *POST /privacy/sendData* (параметры API: отправитель, пароль, ID группы, тип данных, информация о данных, данные и хеш).
2. Участники группы доступа используют инструмент *GET /privacy/{policyId}/getData/{policyItemHash}* для получения информации о данных и их последующей загрузки.

Выполните следующие действия для создания значений в поля `data` и `hash`:

1. Переведите байтовую последовательность данных в кодировку **Base64**.
2. Результат преобразования данных поместите в поле `"data": "29sCt...RgdC60LL"` запроса API *POST /privacy/sendData*.
3. В поле `"hash": "9wetTB...SU2zr1Uh"` укажите хеш-сумму от данных по алгоритму **SHA-256**. Результат хеширования нужно указывать в кодировке **Base58**.
4. Отправьте данные в сеть, нажав кнопку **Try it out!**.
5. В результате отправки данных в сеть нода автоматически сформирует транзакцию *114 PolicyDataHash*.

ГЛАВА 25

Операции шифрования данных

Для операций шифрования/расшифрования данных применяются симметричные ключи CEK и KEK. CEK (Content Encryption Key) используется для шифрования текстовых данных, KEK (Key Encryption Key) используется для шифрования CEK. Ключ CEK формируется блокчейн-узлом случайным образом с применением соответствующих алгоритмов хеширования. Ключ KEK формируется нодой на базе алгоритма Диффи-Хелмана, используя публичные и приватные ключи отправителя и получателей, и применяется для шифрования ключа CEK.

Симметричный ключ CEK недоступен для прочтения и не отображается в процессе шифрования. От отправителя к получателю он передается в зашифрованном виде (wrappedKey) по открытым каналам связи вместе с зашифрованным сообщением. Одним из таких каналов может являться запись в блокчейн - DataTransaction или стейт смарт-контракта. Ключ KEK от отправителя к получателю не передается, он восстанавливается получателем на основе своего закрытого ключа и известного публичного ключа отправителя (алгоритм Diffie-Hellman key exchange).

Процесс шифрования/расшифрования данных включает в себя следующие действия:

1. Для шифрования данных для каждого получателя отдельно используется метод *POST /crypto/encryptSeparate*. Параметры в объекте запроса:

- `sender` - адрес отправителя;
- `password` - пароль от ключевой пары отправителя, создаваемый вместе с аккаунтом;
- `encryptionText` - текст для шифрования;
- `recipientsPublicKeys` - массив публичных ключей получателей.

2. Для шифрования данных для всех получателей единым ключом CEK используется метод *POST /crypto/encryptCommon*.
3. Для расшифровывания данных используется метод *POST /crypto/decrypt*. Параметры в объекте запроса:

- `recipient` - адрес получателя;
- `password` - пароль от ключевой пары получателя, создаваемый вместе с аккаунтом;

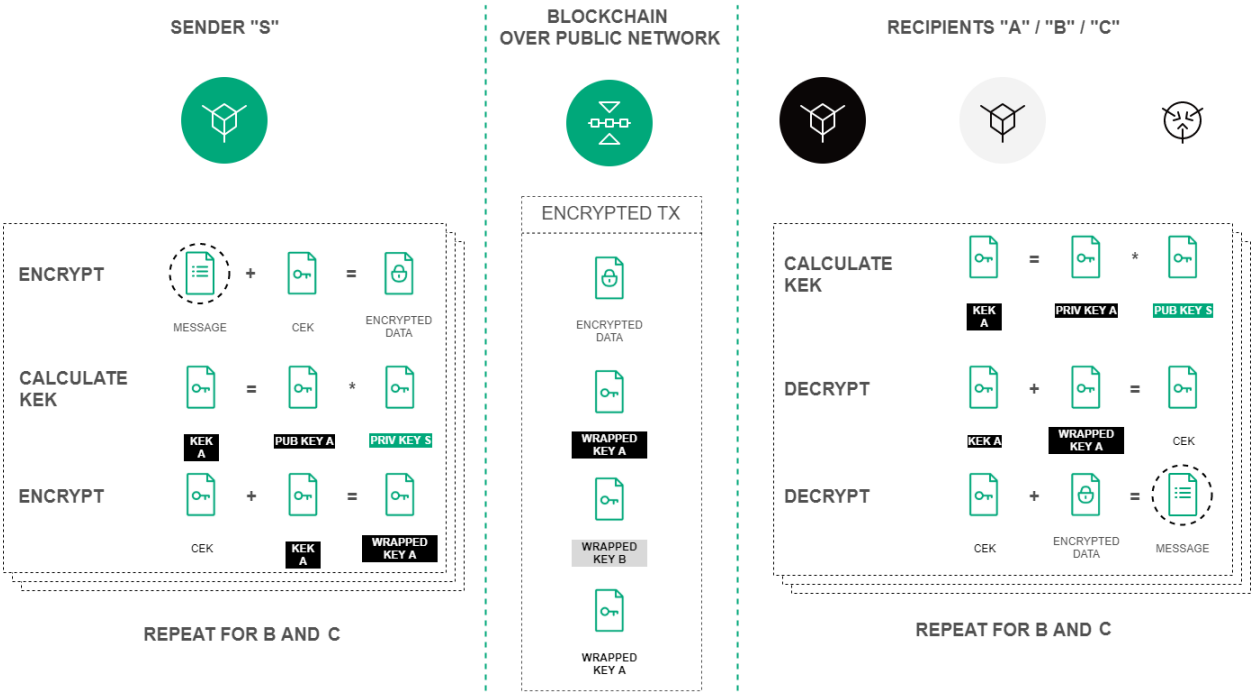


Рис. 1: Процедура шифрования текстовых данных на базе алгоритма Диффи-Хелмана

- `encryptedText` - зашифрованный текст;
- `wrappedKey` - обёрнутый ключ, полученный при шифровании данных;
- `senderPublicKey` - публичный ключ отправителя.

глава 26

Словарь терминов

Аккаунт

Набор данных о пользователе, содержащийся в БД и использующийся для его идентификации

Алиас

Логин пользователя, связанный с его адресом в результате транзакции, по итогам которой в БД записывается сопоставление алиас-адрес, и можно в последующих транзакциях указывать алиас

Анонимная сеть

Блокчейн открытого типа, к которой может подключиться любой участник на правах анонимности

Блокчейн

Децентрализованный, распределённый и общедоступный цифровой реестр, записывающий информацию таким образом, что любая соответствующая запись не может быть изменена задним числом без внесения изменений во все последующие блоки

Генезис блок

Первый блок в блокчейне, содержащий специальные генезис транзакции, распределяющие первоначальный баланс и разрешения

Группа доступа

Таблица в стейте ноды, содержащая список участников сети, которые могут обмениваться конфиденциальными данными в рамках этой группы

Криптовалюта

Цифровая монета, основанная на криптографических алгоритмах и обращающаяся на децентрализованных платформах, построенных по технологии блокчейн

Консенсус

Способ согласования об информационном событии о транзакциях и блоках в сети между участниками

Майнинг

Процесс верификации и добавления в блокчейн транзакций

Мейннет

Рабочая сеть, в которой происходят транзакции, выпуск и хранение токенов

Нода

Компьютер с запущенным ПО ноды и включённый в блокчейн

Пир

Сетевой адрес ноды

Приватный ключ

Строковая комбинация символов для подписания транзакий и доступа к токенам, хранимая приватно. Приватный ключ неразрывно связан с публичным ключом

Публичная сеть

Блокчейн закрытого типа, где каждый участник известен и зарегистрирован в сети

Публичный ключ

Строковая комбинация символов, неразрывно связанная с приватным ключом. Публичный ключ прикладывается к транзакциями для подтверждения корректности подписи пользователя сделанной на закрытом ключе

Публичный адрес

Публичный адрес представляет собой хеш публичного ключа и байт сети. В отличии от приватных ключей публичный адрес может использоваться где угодно в сети, как электронный адрес

Сваггер

Инструмент для работы с API

Секретная фраза

Набор из 24 произвольно заданных слов для восстановления доступа к токенам

Смарт-аккаунт

Аккаунт, к которому добавлены определённые свойства для создания и исполнения смарт-контрактов

Смарт-ассет

Токен с привязанным к нему скриптом, при осуществлении каждой новой транзакции с таким токеном она будет подтверждаться сначала скриптом, потом уже блокчейном

Смарт-контракт

Программный код, предназначенный для облегчения, прямого выполнения или обеспечения выполнения соглашения между участниками

Стейт

Полная история транзакций, хранящаяся во внутренней БД ноды

Токен

Расчётная единица, актив блокчейна, не являющийся криптовалютой и предназначенный для представления цифрового баланса, аналог акций компании

Транзакция

Операция, производимая участниками в сети, для инициации любых действий

Участник

Участник блокчейна, который направляет транзакции на подтверждение в сеть

Хеш

Уникальная конфигурация символов (буквы, цифры), результат исполнения хеш-функции по заданному алгоритму над данными. Хеш однозначно идентифицирует объект

Частная сеть

Блокчейн закрытого типа, где все транзакции контролируются центральным органом

Шлюз

Приложение для перевода токенов из одного блокчейна в другой

Эйрдроп

Процесс раздачи токенов пользователям blockchain-кошельков на безвозмездной основе

PoS (Proof-of-Stake)

Алгоритм консенсуса на основе «доли», которая применяется для выбора ноды для следующего процесса проверки транзакций и создания блока

PoA (Proof-of-Authority)

Алгоритм консенсуса в приватном блокчейне, при котором возможность проверки транзакций и создание новых блоков отводится более авторитетным узлам

ГЛАВА 27

Что нового в Waves Enterprise

27.1 1.2.1

Добавились следующие разделы:

- Методы REST API *Debug*
- Полное описание REST API на странице Документация API

Изменены следующие разделы:

- *Установка и запуск платформы Waves Enterprise*

27.2 1.2.0

Добавлены следующие разделы:

- Новый раздел справки *Интеграционные сервисы*, в который вошли *Сервис авторизации* и *Сервис подготовки данных*
- Добавлена инструкция по *получению лицензии*
- Добавлен новый метод REST API ноды *Licenses*
- Добавлен новый раздел *Смарт-контракты Docker с использованием gRPC*
- Добавлен новый раздел *Сервисы gRPC, используемые смарт-контрактом*

Изменены следующие разделы:

- *Установка и запуск платформы Waves Enterprise*
- Обновлен раздел *Криптография*. Часть информации была перенесена в *Операции шифрования данных*
- *Изменения в конфигурационном файле ноды*
- *Транзакции*

27.3 1.1.2

Изменены следующие разделы:

- Демо-версия
- *Изменения в конфигурационном файле ноды*
- Раздел *Установка ноды* преобразован в раздел «Установка и запуск платформы Waves Enterprise»
- *Подключение участников к сети*
- *Настройка анкоринга*
- *Настройка типа авторизации для доступа к REST API ноды*
- *Подключение ноды в сеть «Waves Enterprise Partnernet»*
- *Подключение ноды в сеть «Waves Enterprise Mainnet»*
- *Системные требования*

27.4 1.1.0

Добавились следующие разделы:

- *Методы API, доступные смарт-контракту*
- Демо-версия
- *Изменения в конфигурационном файле ноды*

Изменены следующие разделы:

- *Смарт-контракты Docker*
- *Пример запуска контракта Docker*
- *Установка ноды*
- Конфигурация и запуск дополнительных сервисов

27.5 1.0.0

Добавились следующие разделы:

- *Сервис авторизации*

Переработаны следующие разделы:

- *Конфигурация ноды*
- *Подключение к Mainnet и Partnernet*
- *REST API*
- *Установка ноды*

Изменения в конфигурационном файле ноды `node.conf`

- Добавлена секция *NTP-сервер*
- Добавлена секция `auth` выбора типа авторизации в *REST API* секции