109052, г. Москва, ул. Смирновская,
д. 2 , стр. 1 , пом. 108А
https://web3tech.ru/confident
E-mail: support@web3tech.ru

Блокчейн-платформа Конфидент

Версия 1.9

Руководство по эксплуатации

Листов 129

2026 г.

Содержание

Аннотация

Настоящий документ описывает состав функций Блокчейн-платформы Конфидент и предназначен для разработки прикладного программного обеспечения (ППО) с непосредственным вызовом функций Блокчейн-платформы Конфидент (далее – Платформа «Конфидент»), а также определяет требования к операционным системам при работе с Платформой «Конфидент». Помимо этого, документ описывает следующие инструменты платформы:

- API:
  - gRPC интерфейс;
  - REST API методы.
- смарт-контракты:
  в руководстве приведены примеры разработки смарт-контракта с использованием gRPC API;
- транзакции;
- механизмы обмена конфиденциальными данными;
- роли участников;
- подход к работе с узлами (нодами).

Развёртывание и настройка Платформы «Конфидент» описаны в документе «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

Примечание: Доступ к документации по следующим криптографическим операциям предоставляется после оформления договора на приобретение Платформы «Конфидент» одновременно с дистрибутивом платформы:

- двусторонняя аутентификация абонентов (между узлами сети, между клиентом и узлом сети) и обеспечение конфиденциальности и целостности информации, передаваемой по каналу связи по протоколу TLS;
- вычисление хэш-значения в соответствии с ГОСТ Р 34.11-2012 (256 бит);
- создание/проверка ЭП блока, проверка ЭП транзакции в соответствии с ГОСТ Р 34.10-2012 (256 бит);
- генерация ключевой пары ГОСТ Р 34.10-2012 (256 бит).

1 Системные требования

На данный момент Платформа «Конфидент» поддерживает операционные системы на базе Unix. Эффективная работа платформы обеспечивается для следующих операционных систем:

• Astra Linux Special Edition 1.6 (x64).

Ниже приведены аппаратные и системные требования к компьютеру, на котором разворачивается узел Платформы «Конфидент».

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

Примечание: «Xmx» – флаг, определяющий максимальный размер доступной для JVM памяти.

1.1 Требования к окружению для Платформы «Конфидент»

Платформа «Конфидент» распространяется в формате jar-файлов. Ниже приведены требования к окружению для Платформы «Конфидент»:

- Eclipse Temurin Java SE 11 (64-bit) или OpenJDK 11.0.12-jre (64-bit) (необходимы для запуска генератора);

Для работы со смарт-контрактами необходима удалённая машина Docker-хост, на которой установлено ПО Docker CE версии 19.03.14. Настройка, необходимая для работы со смарт-контрактами, описана в разделе «Общая настройка платформы: настройка исполнения смарт-контрактов» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

1.2 Требования к окружению для узла (ноды) при использовании криптографии Waves

- OpenJDK 11.0.12-jre (64-bit)

1.3 Требования к окружению для узла (ноды)

- OpenJDK 11.0.12-jre (64-bit)

Ядро, реализующее криптографические алгоритмы и протоколы, описано в документации, которая поставляется вместе с дистрибутивом платформы

1.4 Тестовый режим функционирования Платформы «Конфидент»

Ряд gRPC и REST API методов Платформы «Конфидент», которые подразумевают работу с закрытым ключом на узле, доступны только в тестовом режиме функционирования Платформы «Конфидент». Для активации тестового режима необходимо присвоить значение `TEST` параметру `node.crypto.pki.mode` в конфигурационном файле узла. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

1.5 Отключение поддержки алгоритмов ГОСТ криптографии в тестовых целях

Тип и параметры используемого в блокчейне криптографического алгоритма задаются в разделе `crypto` конфигурационного файла узла.

Настройки, необходимые для тестового отключения поддержки алгоритмов ГОСТ криптографии с PKI описаны в документации, которая поставляется вместе с дистрибутивом платформы.

2 Инструментарий gRPC

Платформа «Конфидент» предоставляет возможность взаимодействия с блокчейном при помощи gRPC-интерфейса – высокопроизводительного фреймворка для удаленного вызова процедур (Remote Procedure Call, RPC). Фреймворк работает поверх HTTP/2. Для передачи данных между клиентом и сервером используется формат сериализации protobuf, описывающий применяемые типы данных. Взаимодействие осуществляется по каналу связи по протоколу TLS с использованием алгоритмов ГОСТ.

Официально gRPC поддерживает 10 языков программирования, список которых доступен в официальной документации gRPC (https://grpc.io/docs/languages/).

Некоторые gRPC сервисы представлены в двух вариантах: для внешней интеграции (паблик сервисы) и для смарт-контрактов (контрактные сервисы описаны ниже в разделе «API-инструменты, доступные смарт-контракту»). Используйте паблик сервисы для интеграции с Платформой «Конфидент». Контрактные сервисы не предназначены для вызова внешним пользователем, они имеют другую авторизацию и поведение. Контрактные сервисы расположены в директории contract и описаны в разделе grpc-contract.

2.1 Для чего предназначен gRPC-интерфейс платформы

Вы можете использовать gRPC-интерфейс каждого узла для решения следующих задач:

- отслеживание событий в блокчейне;
- получение информации об узле (ноде);
- получение информации о транзакции по ее ID;
- получение информации о результатах исполнения вызова смарт-контракта;
- получение информации о размере UTX-пула;
- получение сертификатов;
- работа с транзакциями;
- работа с конфиденциальными данными;
- получение вспомогательной информации;
- получение информации об участниках сети.

Для каждой из этих задач предусмотрен собственный набор методов, упакованный в соответствующие protobuf-файлы.

2.2 Предварительная настройка gRPC-интерфейса

Перед использованием gRPC-интерфейса:

1. определитесь с языком программирования, который вы будете применять для взаимодействия с узлом;
2. установите фреймворк gRPC для вашего языка программирования в соответствии с официальной документацией gRPC (https://grpc.io/docs/languages/);
3. распакуйте пакет protobuf-файлов confident-proto-1.5-1.9.0.zip, который поставляется вместе с jar-файлами Платформы «Конфидент»;
4. скачайте плагин protoc (доступен для скачивания по адресу https://developers.google.com/protocol-buffers/docs/downloads) для компиляции protobuf-файлов;
5. убедитесь, что gRPC-интерфейс запущен и настроен в конфигурационном файле узла, с которым будет производиться обмен данными. Настройка gRPC API узла описана в разделе «Тонкая настройка платформы: настройка инструментов gRPC и REST API узла» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

Для взаимодействия с узлом через gRPC-интерфейс по умолчанию предусмотрен порт 6865.

Для доступа к gRPC API инструментам узла необходима авторизация. Типы авторизации, которые поддерживает Платформа «Конфидент», описаны в разделе «Тонкая настройка платформы: настройка авторизации для gRPC и REST API» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

2.2.1 gRPC: отслеживание событий в блокчейне

gRPC-интерфейс имеет возможность отслеживания определенных групп событий, происходящих в блокчейне. Информация о выбранных группах событий собирается в потоки, которые поступают в gRPC-интерфейс узла.

Набор полей, предназначенный для сериализации и передачи данных о событиях в блокчейне, приведен в файлах, которые находятся в каталоге **messagebroker** пакета `confident-proto-1.5-1.9.0.zip`:

- `messagebroker_blockchain_events_service.proto` – основной protobuf-файл;
- `messagebroker_subscribe_on_request.proto` – файл, содержащий поля с параметрами запроса;
- `messagebroker_blockchain_event.proto` – файл, содержащий поля ответов с данными групп событий и сообщениями об ошибках.

Для отслеживания определенной группы событий в блокчейне отправьте запрос `SubscribeOn(startFrom, transactionTypeFilter)`, который инициализирует подписку на выбранную группу событий.

Примечание: Типы данных полей для запросов и ответов указаны в protobuf-файлах.

Параметры запроса:

- `startFrom` – момент начала отслеживания событий:
  - `CurrentEvent` – начало отслеживания от текущего события;
  - `GenesisBlock` – получение всех событий выбранной группы, начиная от генезис-блока;
  - `BlockSignature` – начало отслеживания от указанного блока.
- `transactionTypeFilter` – фильтрация выводимых событий по транзакциям, которые производятся в ходе этих событий:
  - `Any` – выводить события со всеми типами транзакций;
  - `Filter` – выводить события с типами транзакций, указанными в виде списка;
  - `FilterNot` – выводить события со всеми транзакциями кроме тех, которые указаны в этом параметре в виде списка.
- `connectionId` – опциональный параметр, отправляемый для удобства идентификации запроса в логах узла.

Вместе с запросом `SubscribeOnRequest` отправляются данные авторизации.

2.2.1.1 Информация о событиях

После успешной отправки запроса на gRPC-интерфейс будут приходить данные следующих групп событий:

1. **MicroBlockAppended** – успешный майнинг микроблока:
   - `transactions` – полные тела транзакций из полученного микроблока.
2. **BlockAppended** – успешное завершение раунда майнинга с формированием блока:
   - `block_signature` – подпись полученного блока;
   - `reference` – подпись предыдущего блока;
   - `tx_ids` – список ID транзакций из полученного блока;
   - `miner_address` – адрес майнера;
   - `height` – высота, на которой расположен полученный блок;
   - `version` – версия блока;
   - `timestamp` – время формирования блока;
   - `fee` – сумма комиссий за транзакции внутри блока;
   - `block_size` – размер блока (в байтах);
   - `features` – список изменений блокчейна, за которые голосовал майнер в ходе раунда.
3. **RollbackCompleted** – откат блока:
   - `return_to_block_signature` – подпись блока, до которого произошел откат;
   - `rollback_tx_ids` – список ID транзакций, которые будут удалены из блокчейна.

4. **AppendedBlockHistory** – информация о транзакциях сформированного блока. Данный тип событий поступает на gRPC-интерфейс до достижения текущей высоты блокчейна, если в запросе в качестве отправной точки для получения событий указаны `GenesisBlock` или `BlockSignature`. После достижения текущей высоты начинают выводиться текущие события по заданным фильтрам.

Данные ответа:

- `signature` – подпись блока;
- `reference` – подпись предыдущего блока;
- `transactions` – полные тела транзакций из блока;
- `miner address` – адрес майнера;
- `height` – высота, на которой расположен блок;
- `version` – версия блока;
- `timestamp` – время формирования блока;
- `fee` – сумма комиссий за транзакции внутри блока;
- `block_size` – размер блока (в байтах);
- `features` – список изменений блокчейна, за которые голосовал майнер в ходе раунда.

2.2.1.2 Информация об ошибках

Для вывода информации об ошибках в ходе отслеживания событий в блокчейне предусмотрено сообщение ErrorEvent со следующими вариантами ошибок:

- `GenericError` – общая или неизвестная ошибка с текстом сообщения;
- `MissingRequiredRequestField` – не заполнено обязательное поле при формировании запроса SubscribeOnRequest;
- `BlockSignatureNotFoundError` – в блокчейне отсутствует подпись запрошенного блока;
- `MissingAuthorizationMetadata` – при формировании запроса `SubscribeOn` не введены данные авторизации;
- `InvalidApiKey` – при авторизации по api-key, неверная ключевая фраза.

2.2.2 gRPC: получение информации об узле (ноде)

Для получения параметров конфигурации узла и данных о её владельце предусмотрен gRPC сервис **NodeInfoService**.

У сервиса **NodeInfoService** есть следующие методы, описанные в protobuf-файле **util_node_info_service.proto**:

- **NodeConfig;**
- **NodeOwner.**

Примечание: Типы данных полей для запросов и ответов указаны в protobuf-файле.

2.2.2.1 gRPC: получение параметров конфигурации узла (ноды)

Используйте метод `NodeConfig` для получения параметров конфигурации узла. Метод `NodeConfig` не требует ввода дополнительных параметров запроса. В ответе выводятся следующие параметры конфигурации узла, к которому был осуществлен запрос:

- `version` – используемая версия блокчейн-платформы;
- `crypto_type` – используемый криптографический алгоритм;

  используемая на платформе криптография подробнее описана в документации, которая поставляется вместе с дистрибутивом платформы.

- `chain_id` – идентифицирующий байт сети;
- `consensus` – используемый алгоритм консенсуса;
- `minimum_fee` – минимальная комиссия за транзакции;
- `additional_fee` – дополнительная комиссия за транзакции;
- `max_transactions_in_micro_block` – максимальное установленное количество транзакций в микроблоке;

- `min_micro_block_age` – минимальное время существования микроблока (в секундах);
- `micro_block_interval` – интервал формирования микроблоков (в секундах);
- `pki_mode` – при использовании алгоритмов ГОСТ криптографии с PKI выводится используемый режим PKI:
  - `ON – PKI используется,`
  - `OFF – PKI не используется,`
  - `TEST – тестовый режим.`
- `required_oids` – при использовании алгоритмов ГОСТ криптографии с PKI выводится список OID-строк пользователей, которым УЦ выдал OID специально для работы с блокчейн платформой; подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» в документе «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.
- `pos_round_info` – при использовании алгоритма консенсуса PoS, выводится значение параметра `average_block_delay` (время средней задержки создания блоков, в секундах), которое задано в конфигурационном файле узла;
- `poa_round_info` – при использовании алгоритма консенсуса PoA, выводятся параметры:
  - `round_duration` – длина раунда майнинга блока, в секундах и
  - `sync_duration` – период синхронизации майнинга блока, в секундах.
- `crlChecksEnabled` – режим проверки списка отозванных сертификатов (CRL) при валидации сертификатов.

2.2.2.2 gRPC: получение данных о владельце узла (ноды)

Используйте метод `NodeOwner` для получения данных о владельце узла. Метод `NodeOwner` не требует ввода дополнительных параметров запроса. В ответе выводятся следующие данные узла, к которому был осуществлен запрос:

- `address` – адрес узла;
- `public_key` – публичный ключ.

2.2.3 gRPC: получение информации о транзакции по ее ID

Для получения информации о транзакции по ее ID на gRPC-интерфейс узла предназначены два запроса, описанные в файле **transaction_public_service.proto**:

- `TransactionExists` – запрос о существовании транзакции с указанным ID;
- `TransactionInfo` – запрос информации о транзакции с указанным ID.

Оба запроса требуют ввода параметра `tx_id` – ID транзакции, о которой запрашивается информация.

Примечание: Типы данных полей для запросов и ответов указаны в protobuf-файле.

Ответ на запрос `TransactionExists` содержит одно поле:

- `exists` – булев тип данных: `true` – транзакция существует, `false` – транзакция не существует.

Ответ на запрос `TransactionInfo` содержит, в частности, следующую информацию о транзакции:

- `height` – высота блокчейна, на которой была произведена транзакция;
- `transaction` – название транзакции.

2.2.4 gRPC: получение информации о результатах исполнения вызова смарт-контракта

Для получения информации о результатах вызовов смарт-контрактов служит gRPC сервис **ContractStatusService**.

У сервиса есть два метода, описанных в protobuf-файле **util_contract_status_service.proto**:

- **ContractExecutionStatuses,**
- **ContractsExecutionEvents.**

Примечание: Типы данных полей для запросов и ответов указаны в protobuf-файле.

Используйте метод **ContractExecutionStatuses** для получения информации о результатах исполнения вызова отдельного смарт-контракта. Метод принимает запрос **ContractExecutionRequest**, который требует

ввода параметра `tx_id` – идентификатора вызывающей транзакции смарт-контракта, информацию о состоянии которого необходимо получить.

Используйте метод **ContractsExecutionEvents** для подписки на поток (стрим) с результатами исполнения вызова всех смарт-контрактов. Метод не требует ввода входных параметров.

2.2.4.1 Информация о результатах исполнения вызова смарт-контракта

В ответе на запрос оба метода возвращают следующие данные смарт-контракта:

- `sender` – участник, отправивший смарт-контракт в блокчейн;
- `tx_id` – идентификатор транзакции вызова смарт-контракта;
- `Status` – информация об исполнении смарт-контракта:
  - `0` – успешно исполнен (SUCCESS);
  - `1` – бизнес-ошибка, контракт не исполнен, вызов отклонён (ERROR);
  - `2` – системная ошибка в ходе исполнения смарт-контракта (FAILURE).
- `code` – код ошибки в ходе выполнения смарт-контракта;
- `message` – сообщение об ошибке;
- `timestamp` – время вызова смарт-контракта;
- `signature` – подпись смарт-контракта.

2.2.5 gRPC: получение информации о размере UTX-пула

Запрос о размере UTX-пула `UtxInfo` отправляется в виде подписки: после его отправки ответ от узла приходит раз в 1 секунду. Это запрос не требует ввода дополнительных параметров и описан в файле **transaction_public_service.proto**.

В ответ на запрос выводится сообщение `UtxSize`, которое содержит два параметра:

- `size` – размер UTX-пула в килобайтах;
- `size_in_bytes` – размер UTX-пула в байтах.

Примечание: Типы данных полей для запросов и ответов указаны в protobuf-файлах.

2.2.6 gRPC: получение сертификатов

Для запроса у узла сертификата из хранилища сертификатов предусмотрена группа методов gRPC сервиса **PkiPublicService**. Методы этой группы позволяют получить сертификат по разным полям:

- **GetCertificateByDn(CertByDNRequest)** – по полю `DN` (distinguished name),
- **GetCertificateByDnHash(CertByDNHashRequest)** – по полю `DN Hash`,
- **GetCertificateByPublicKey(CertByPublicKeyRequest)** – по полю `publicKey`,
- **GetCertificateByFingerprint(CertByFingerprintRequest)** – по полю `fingerprint`.

В запросе эти методы принимают значение соответствующего поля сертификата и, опционально, параметр `plainText`, который задаёт формат ответа.

Если сертификат существует, то в ответе каждого из этих методов узел возвращает сертификат в формате DER (как он и записан в хранилище сертификатов узла). Если в запросе метода параметру `plainText` задано значение `true`, то сертификат возвращается в формате plainText.

Если сертификата не существует, то в ответе каждого из этих методов возвращается ошибка.

Методы не требуют авторизации.

2.2.6.1 Получение сертификата по DN

Метод **GetCertificateByDn(CertByDNRequest)** возвращает сертификат по его отличительному имени (distinguished name), записанному в поле `DN`.

2.2.6.2 Получение сертификата по хэшу DN

Метод **GetCertificateByDnHash(CertByDNHashRequest)** возвращает сертификат по хэшу SHA-1 (Keccak) от поля `DN` сертификата.

2.2.6.3 Получение сертификата по публичному ключу

Метод **GetCertificateByPublicKey(CertByPublicKeyRequest)** возвращает сертификат по байтам публичного ключа (поле `publicKey`).

2.2.6.4 Получение сертификата по его отпечатку

Метод **GetCertificateByFingerprint(CertByFingerprintRequest)** возвращает сертификат по его SHA-1 отпечатку (поле `fingerprint`).

2.2.7 gRPC: работа с транзакциями

Для работы с транзакциями предусмотрен gRPC сервис **TransactionPublicService**.

У сервиса **TransactionPublicService** есть следующие методы, описанные в protobuf-файле **transaction_public_service.proto**:

- **Broadcast;**
- **BroadcastWithCerts**;
- **UtxInfo;**
- **TransactionInfo;**
- **UnconfirmedTransactionInfo.**

Примечание: Типы данных полей для запросов и ответов указаны в protobuf-файлах.

2.2.7.1 Отправка транзакций в блокчейн

Выберите подходящий для вашей задачи метод отправки транзакций в блокчейн:

- `BroadcastWithCerts` – для отправки транзакции RegisterNode, описанной ниже в разделе «Описание транзакций» – «111. RegisterNode Transaction»;
- `Broadcast` – для отправки всех остальных транзакций.

2.2.7.1.1 Broadcast

Метод требует ввода следующих параметров запроса:

- `version` – версия транзакции;
- `transaction` – название транзакции вместе с предназначенным для нее набором параметров.
- `certificates` – цепочка сертификатов байтами в формате DER; параметр является обязательным при одновременном соблюдении следующих условий:
  - используется PKI или тестовый режим PKI (то есть в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `ON` или `TEST`),
  - новый пользователь, который не является владельцем узла (node-owner), делает свою первую транзакцию.

  В этом случае необходимо в запросе в поле `certificates` передать цепочку сертификатов пользователя; в других случаях поле `certificates` является необязательным.

Для каждой транзакции предусмотрен отдельный protobuf-файл, описывающий поля запросов и ответов. Эти поля универсальны для запросов по gRPC и REST API и приведены ниже в разделе «Транзакции блокчейн-платформы Конфидент».

2.2.7.1.2 BroadcastWithCerts

Метод используется для отправки транзакции RegisterNode (описанной ниже в разделе «Описание транзакций» – «111. RegisterNode Transaction») и требует тех же входных параметров, что и описанный выше метод Broadcast.

Поле `certificates` является обязательным и должно содержать цепочку сертификатов, которая соответствует публичному ключу в поле `target` транзакции.

2.2.7.2 Получение данных транзакции, находящейся в UTX-пуле

Используйте метод `UnconfirmedTransactionInfo`, чтобы получить данные транзакции, находящейся в UTX-пуле. В ответе метода содержатся данные транзакции, аналогичные ответу метода `Broadcast`.

2.2.8 gRPC: работа с конфиденциальными данными

Для работы с конфиденциальными данными (privacy) предусмотрены gRPC сервисы **PrivacyEventsService** и **PrivacyPublicService**.

Важно: Все методы работы с конфиденциальными данными, описанные ниже в подразделах данного раздела, доступны только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

2.2.8.1 Авторизация методов PrivacyEventsService и PrivacyPublicService в тестовом режиме

Для использования методов gRPC API сервисов **PrivacyEventsService** и **PrivacyPublicService** требуется авторизация по api-key. Авторизация методов реализована следующим образом:

- в случае api-key авторизации требуется PrivacyApiKey.

Для каждого из методов необходимо передавать следующие данные:

- Recipients — userAuth;
- Owners — userAuth;
- Hashes — userAuth;
- GetPolicyItemData — privacyAuth;
- GetPolicyItemInfo — privacyAuth;
- SendData — privacyAuth; SendLargeData — privacyAuth, forceSync — privacyAuth.

где

- userAuth — api-key пользователя, передаваемый в заголовке „X-Api-Key“ к запросу;
- privacyAuth — api-key privacy пользователя в заголовке „X-Api-Key“ к запросу.

2.2.8.2 PrivacyEventsService

У сервиса **PrivacyEventsService** есть один метод **SubscribeOn**, описанный в protobuf-файле **privacy_events_service.proto**. Используйте этот метод для получения потока (стрима) событий по получению или удалению конфиденциальных данных, которые поступают в gRPC-интерфейс узла. Для этого отправьте запрос `SubscribeOn (SubscribeOnRequest)`, который инициализирует подписку на стрим.

2.2.8.2.1 Информация о получении или удалении конфиденциальных данных

После успешной отправки запроса на gRPC-интерфейс будут приходить следующие данные:

- `policy_id` – идентификатор группы доступа к конфиденциальным данным;
- `data_hash` – идентификационный хэш конфиденциальных данных;
- `event_type` – тип события; доступны следующие типы:
  - DATA_ACQUIRED – данные сохранены в БД;

- DATA_INVALIDATED – данные помечены на удаление в связи с отсутствием активности по ним или при откате (роллбэке).

2.2.8.3 PrivacyPublicService

У сервиса **PrivacyPublicService** есть следующие методы, описанные в protobuf-файле **privacy_public_service.proto**:

- GetPolicyItemData;
- GetDataLarge;
- GetPolicyItemInfo;
- PolicyItemDataExists;
- SendData;
- SendLargeData;
- Recipients;
- Owners;
- Hashes;
- forceSync.

Примечание: Типы данных полей для запросов и ответов указаны в protobuf-файле.

Важно: Все методы для работы с конфиденциальными данными, описанные ниже в подразделах данного раздела, доступны только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

2.2.8.3.1 Получение хэш-суммы конфиденциальных данных

Используйте метод **GetPolicyItemData** для получения пакета конфиденциальных данных группы доступа по идентификационному хэшу. Метод требует ввода параметров запроса `policy_id` (идентификатор группы доступа) и `data_hash` (идентификационный хэш). После успешной отправки запроса на gRPC-интерфейс возвращается хэш-сумма конфиденциальных данных.

2.2.8.3.2 Скачивание из узла (ноды) больших данных

Используйте метод **GetDataLarge** для скачивания из узла больших данных, которые были отправлены с помощью метода SendLargeData, описанного ниже. Метод требует ввода параметров запроса `policy_id` (идентификатор группы доступа) и `data_hash` (идентификационный хэш). После успешной отправки запроса на gRPC-интерфейс возвращается поток PolicyItemDataResponse с данными.

2.2.8.3.3 Получение метаданных для пакета конфиденциальных данных

Используйте метод **GetPolicyItemInfo** для получения метаданных для пакета конфиденциальных данных группы по идентификационному хэшу. Метод требует ввода параметров запроса `policy_id` (идентификатор группы доступа) и `data_hash` (идентификационный хэш). После успешной отправки запроса на gRPC-интерфейс возвращаются следующие данные:

- `sender` – адрес отправителя конфиденциальных данных;
- `policy_id` – идентификатор группы доступа;
- `type` – тип конфиденциальных данных (`file`);
- `info` – массив данных о файле:
  - `filename` – имя файла;
  - `size` – размер файла;
  - `timestamp` – временная метка размещения файла в формате Unix Timestamp (в миллисекундах);

- `author` – автор файла;
- `comment` – опциональный комментарий к файлу;

- `hash` – идентификационный хэш конфиденциальных данных.

2.2.8.3.4 Проверка существования пакета конфиденциальных данных

Используйте метод **PolicyItemDataExists** для получения информации о наличии пакета конфиденциальных данных группы доступа по идентификационному хэшу. Метод требует ввода параметров запроса `policy_id` (идентификатор группы доступа) и `data_hash` (идентификационный хэш). После успешной отправки запроса на gRPC-интерфейс возвращается значение `true`, если данные в наличии, или `false`, если данные отсутствуют.

2.2.8.3.5 Отправка в блокчейн конфиденциальных данных

Используйте метод **SendData** для отправки в блокчейн конфиденциальных данных, доступных только для участников группы доступа, определенной для этих данных.

Важно: Метод SendData доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

Примечание: Для отправки данных размером более 20 МБ используйте метод SendLargeData, описанный ниже.

Примечание: Для отправки в блокчейн потока конфиденциальных данных используйте метод SendLargeData, описанный ниже.

Метод требует ввода следующих параметров запроса:

- `sender_address` – блокчейн-адрес, от которого должны рассылаться данные (соответствуют значению параметра `privacy.owner-address` в конфигурационном файле узла);
- `policy_id` – идентификатор группы доступа к конфиденциальным данным, которая будет иметь доступ к отправляемым данным;
- `data_hash` – идентификационный sha256-хэш конфиденциальных данных в формате base58;
- `info` – информация об отправляемых данных;
- `fee` – комиссия за транзакции;
- `fee_asset_id` – поле опционально и используется только для смарт-контрактов с поддержкой gRPC;
- `atomic_badge` – поле-метка, указывающая, что транзакция поддерживается атомарной транзакцией;
- `password` – пароль для доступа к закрытому ключу keystore узла;
- `broadcast_tx` – если передается значение `true`, то созданная PolicyDataHash транзакция отправляется в блокчейн, если `false`, то транзакция и сообщение о наличии данных (Privacy Inventory) не отправляется; подробнее см. ниже;
- `data` – строка, содержащая данные в формате base64.
- `certificates` – цепочка сертификатов байтами в формате DER; параметр является обязательным при одновременном соблюдении следующих условий:
  - используется тестовый режим PKI (то есть в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`); подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы,
  - новый пользователь, который не является владельцем узла (node-owner), делает свою первую транзакцию.

  В этом случае необходимо в запросе в поле `certificates` передать цепочку сертификатов пользователя; в других случаях поле `certificates` является необязательным.

После успешной отправки запроса на gRPC-интерфейс будут приходить следующие данные:

- `tx_version` – версия транзакции;
- `tx` – созданная PolicyDataHash транзакция.

Параметр broadcast_tx

Для снижения вероятности ошибок доставки данных рекомендуется установить для параметра `broadcast_tx` значение `false`, если после отправки данных с помощью API метода **SendData** отправляется атомарная транзакция, которая содержит транзакцию CreatePolicy и транзакцию PolicyDataHash.

2.2.8.3.6 Отправка в блокчейн потока конфиденциальных данных

Используйте метод **SendLargeData** для отправки в блокчейн потока конфиденциальных данных. Данные будут доступны только для участников группы доступа, определенной для этих данных.

Важно: Метод **SendLargeData** доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

Примечание: Для отправки данных размером менее 20 МБ используйте метод SendData, описанный выше.

Метод принимает в запросе поток данных в следующем формате:

- `metadata` – метаданные для пакета конфиденциальных данных, аналогичные входным данным метода SendData, описанного выше;
- `content` – массив байт, представляющих собой пакет конфиденциальных данных.

После успешной отправки запроса на gRPC-интерфейс будут приходить те же данные, что и для метода SendData, описанного выше.

2.2.8.3.7 Получение адресов всех участников группы доступа к конфиденциальным данным

Используйте метод **Recipients** для получения адресов всех участников группы доступа к конфиденциальным данным. Метод требует ввода параметра запроса `policy_id` – идентификатор группы доступа. В ответе метод возвращает массив строк с адресами участников группы доступа.

2.2.8.3.8 Получение адресов владельцев группы доступа к конфиденциальным данным

Используйте метод **Owners** для получения адресов владельцев группы доступа к конфиденциальным данным. Метод требует ввода параметра запроса `policy_id` (идентификатор группы доступа). В ответе метод возвращает массив строк с адресами владельцев группы доступа.

2.2.8.3.9 Получение массива идентификационных хэшей данных

Используйте метод **Hashes** для получения массива идентификационных хэшей данных, которые привязаны к группе доступа к конфиденциальным данным. Метод требует ввода параметра запроса `policy_id` (идентификатор группы доступа). В ответе метод возвращает массив строк с идентификационными хэшами данных группы доступа.

2.2.8.3.10 Синхронизация данных по указанной группе доступа к конфиденциальным данным

Используйте метод **forceSync** для синхронизации данных по указанной группе доступа к конфиденциальным данным. Метод требует ввода параметра запроса `policy_id` (идентификатор группы доступа).

В результате выполнения метода узел запускает процесс синхронизации и возвращает размер конфиденциальных данных в Мб. Если синхронизацию не удалось запустить, узел возвращает описание ошибки.

2.2.9 gRPC: получение вспомогательной информации

Для получения вспомогательной информации предусмотрен gRPC сервис **UtilPublicService**.

2.2.9.1 Получение текущего времени узла (ноды)

У сервиса **UtilPublicService** есть один метод, описанный в protobuf-файле **util_public_service.proto**:

- **GetNodeTime.**

Используйте этот метод для получения текущего времени узла. Метод не требует ввода дополнительных параметров запроса.

Примечание: Типы данных полей для ответов указаны в protobuf-файлах.

Метод возвращает текущее время узла в двух форматах:

- `system` – системное время на машине узла; **GetNodeTime**
- `ntp` – сетевое время.

2.2.10 gRPC: получение информации об участниках сети

Для получения информации об участниках сети предусмотрены gRPC сервисы **AddressPublicService** и **AliasPublicService**.

2.2.10.1 gRPC: получение информации об адресах участников сети

Для получения информации об адресах участников сети предусмотрен gRPC сервис **AddressPublicService**. У сервиса **AddressPublicService** есть следующие методы, описанные в protobuf-файле **address_public_service.proto**:

- **GetAddresses**;
- **GetAddressData**;
- **GetAddressDataByKey**.

Примечание. Типы данных полей для запросов и ответов указаны в protobuf-файле.

Получение всех адресов участников

Используйте метод **GetAddresses** для получения всех адресов участников, ключевые пары которых хранятся в keystore узла. Метод не требует ввода дополнительных параметров запроса.

Метод возвращает массив адресов участников.

Получение данных с указанного адреса

Используйте метод **GetAddressData** для получения данных, записанных на указанном адресе при помощи транзакций 12. Метод требует ввода следующих параметров запроса:

- `address` – адрес узла;
- `limit` – максимальное количество записей, которые вернет метод;
- `offset` – количество первых записей по данному адресу, которые метод пропустит.

Метод возвращает данные, записанные на указанном адресе.

Получение данных с указанного адреса по ключу

Используйте метод **GetAddressDataByKey** для получения данных, записанных на указанном адресе с ключом при помощи транзакций 12. Этот ключ указывается в транзакции 12 в поле `data.key`. Метод требует ввода следующих параметров запроса:

- `address` – адрес узла;
- `key` – ключ.

Метод возвращает данные, записанные на указанном адресе с ключом key.

2.2.10.2 gRPC: получение информации об участниках сети по псевдониму

Для получения информации об участниках сети по псевдониму предусмотрен gRPC сервис **AliasPublicService**.

У сервиса **AliasPublicService** есть следующие методы, описанные в protobuf-файле **alias_public_service.proto**:

- **AddressByAlias**;
- **AliasesByAddress**.

Получение адреса по псевдониму

Используйте метод **AddressByAlias** для получения адреса по псевдониму. Метод требует ввода одного параметра запроса:

- `alias` – псевдоним участника сети.

Метод возвращает адрес участника сети.

Получение псевдонима по адресу

Используйте метод **AliasesByAddress** для получения псевдонима по адресу. Метод требует ввода в запросе адреса участника сети.

Метод возвращает все псевдонимы участника сети.

3 Методы REST API

REST API позволяет пользователям удалённо взаимодействовать с узлом блокчейн-платформы Конфидент через запросы и ответы в формате JSON. Работа с API происходит по протоколу https. Взаимодействие осуществляется по каналу связи по протоколу TLS с использованием алгоритмов ГОСТ.

3.1 Использование REST API

Все вызовы методов REST API — это http-запросы GET, POST или DELETE к URL `https://yournetwork.com/node-N`, содержащие соответствующие наборы параметров. Платформа также предоставляет доступ к интерфейсу Swagger `https://yournetwork.com/node-N/api-docs/index.html`, который позволяет составлять и отправлять HTTP-запросы в узел через веб-интерфейс. Нужные группы запросов выбираются в интерфейсе Swagger посредством выбора маршрутов (routes) — URL к отдельным методам REST API. В конце каждого маршрута предусмотрена точка доступа (endpoint) — обращение к методу.

Ниже представлен пример запроса о размере UTX-пула (см. Рисунок 1):

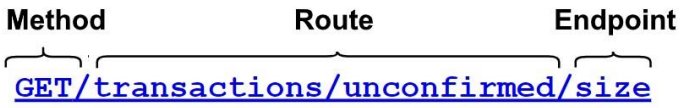

Рисунок 1. Пример запроса о размере UTX-пула

Для доступа к REST API инструментам узла используется авторизация по списку `tls-whitelist`: необходимо, чтобы публичный ключ в клиентском TLS сертификате был равен одному из публичных ключей администраторов, перечисленных в разделе `node.api.auth` конфигурационного файла узла.

3.2 Для чего предназначен REST API платформы

Вы можете использовать интерфейс REST API для выполнения следующих задач:

- работа с транзакциями;
- формирование и проверка электронной подписи данных (PKI);
- получение сертификатов;
- реализация методов шифрования;
- обмен конфиденциальными данными и получение информации о группах доступа;
- работа с лицензиями;
- валидация адресов и псевдонимов участников сети;
- подписание и валидация сообщений в блокчейне;
- получение информации о конфигурации и состоянии узла (ноды), остановка узла (ноды);
- получение информации об участниках сети;
- получение информации об активации новых функциональных возможностей платформы;
- получение информации об используемом алгоритме консенсуса;
- получение информации о смарт-контрактах;
- получение информации о блоках сети;
- получение информации о ролях участников;
- получение информации об ассетах и балансах адресов;
- работа с узлами блокчейна;
- хэширование, работа со скриптами и отправка вспомогательных запросов;
- отладка блокчейна.

В следующих разделах приведены адреса методов, описаны поля запросов и ответов каждого метода, указано, требуется ли для описываемых методов REST API авторизация.

3.2.1 REST API: работа с транзакциями

Для работы с транзакциями предусмотрены методы группы `transactions`. Эти методы требуют авторизации.

3.2.1.1 Подписание и отправка транзакций

REST API узла использует JSON-представление транзакции для отправки запросов.

Основные принципы работы с транзакциями приведены ниже в разделе «Транзакции блокчейн-платформы Конфидент». Описание полей для каждой транзакции приведено ниже в разделе «Описание транзакций».

POST /transactions/broadcast

Для публикации транзакции предназначен метод **POST /transactions/broadcast**. На вход этого метода подаются поля ответа метода **sign**.

Когда новый пользователь, который не является владельцем узла (node-owner), делает свою первую транзакцию, ему необходимо в запросе в поле `certificates` приложить цепочку своих сертификатов. В других случаях поле `certificates` является необязательным.

Примечание. Поле `certificates` в запросе на публикацию транзакции RegisterNode (описана ниже в разделе «Описание транзакций» – «111. RegisterNode Transaction») является обязательным при использовании PKI или тестового режима PKI (то есть когда в конфигурационном файле узла параметру node.crypto.pki.mode присвоено значение ON или TEST; подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы). В этом случае поле `certificates` должно содержать цепочку сертификатов, которая соответствует публичному ключу в поле `target` транзакции.

Пример запроса на публикацию транзакции:


```
{
   "type": 3,
   "id": "DnK5Xfi2wXUJx9BjK9X6ZpFdTLdq2GtWH9pWrcxcmrhB",
   "sender": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",
   "senderPublicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
   "fee": 100000000,
   "timestamp": 1549378509516,
   "proofs": [, "NqZGcbcQ82FZrPh6aCEjuo9nNnkPTvyhrNq329YWydaYcZTywXUwDxFAknTMEGuFrEndCjXBtrueLWaqbJhpeiG" ],
   "version": 2,
   "assetId": "DnK5Xfi2wXUJx9BjK9X6ZpFdTLdq2GtWH9pWrcxcmrhB",
   "name": "Test Asset 1",
   "quantity": 10000,
   "reissuable": true,
   "decimals": 8,
   "description": "Some description",
   "chainId": 84,
   "script": "base64:AQa3b8tH",
   "height": 60719
   "certificates": ["a", "b", ...]
}
```


В случае успешной публикации транзакции метод возвращает json с транзакцией и сообщение `200OK`.

Важно: Описанные ниже методы **transactions/sign** и **transactions/signAndBroadcast** доступны только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

POST /transactions/sign

Для подписания транзакций предназначен метод **POST/transactions/sign**. Этот метод подписывает транзакцию закрытым ключом отправителя, сохраненным в keystore узла. Для подписания запросов ключом из keystore узла обязательно укажите пароль к ключевой паре в поле `password`.

Метод POST /transactions/sign в ответе возвращает поля, необходимые для публикации транзакции.

Примеры запроса на подписание транзакции 3. Issue Transaction и ответа с транзакцией 3. Issue Transaction:

Запрос:


```
{
   "type": 3,
   "version": 2,
   "name": "Test Asset 1",
   "quantity": 100000000000,
   "description": "Some description",
   "sender": "3FSCKyfFo3566zwiJjSFLBwKvd826KXUaqR",
   "decimals": 8,
   "reissuable": true,
   "password": "1234",
   "fee": 100000000
}
```


Ответ:


```
{
    "type": 3,
    "id": "DnK5Xfi2wXUJx9BjK9X6ZpFdTLdq2GtWH9pWrcxcmrhB",
    "sender": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",
    "senderPublicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
    "fee": 100000000,
    "timestamp": 1549378509516,
    "proofs": [ "NqZGcbcQ82FZrPh6aCEjuo9nNnkPTvyhrNq329YWydaYcZTywXUwDxFAknTMEGuFrEndCjXBtrueLWaqbJhpeiG" ],
    "version": 2,
    "assetId": "DnK5Xfi2wXUJx9BjK9X6ZpFdTLdq2GtWH9pWrcxcmrhB",
    "name": "Test Asset 1",
    "quantity": 10000,
    "reissuable": true,
    "decimals": 8,
    "description": "Some description",
    "chainId": 84,
    "script": "base64:AQa3b8tH",
    "height": 60719
}
```


POST /transactions/signAndBroadcast

Помимо отдельных методов подписания и отправки транзакций, предусмотрен комбинированный метод **POST /transactions/signAndBroadcast**. Этот метод подписывает и отправляет транзакцию в блокчейн без промежуточной передачи информации между методами.

Когда новый пользователь, который не является владельцем узла (node-owner), делает свою первую транзакцию, ему необходимо в запросе в поле `certificates` приложить цепочку своих сертификатов. В других случаях поле `certificates` является необязательным.

Пример запроса и ответа метода транзакцией 112. CreatePolicy Transaction:

Запрос:

```
{
   "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
   "policyName": "Policy for sponsored v1",
   "password":"sfgKYBFCF@#$fsdf()*%",
   "recipients": [
     "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
     "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
     "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
     "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
     "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx"
   ],
   "fee": 100000000,
   "description": "Privacy for sponsored",
   "owners": [
     "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
     "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
     "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
],
   "type": 112
   "certificates": ["a", "b", ...]
}
```

Примечание. Поле `certificates` в запросе на публикацию транзакции RegisterNode (описана ниже в разделе «Описание транзакций» – «111. RegisterNode Transaction») является обязательным при использовании тестового режима PKI (то есть когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение TEST. В этом случае поле `certificates` должно содержать цепочку сертификатов, которая соответствует публичному ключу в поле `target` транзакции. Подробнее о параметре `node.crypto.pki.mode` см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы).

Ответ:

```
{
   "senderPublicKey": "3X6Qb6p96dY4drVt3x4XyHKCRvree4QDqNZyDWHzjJ79",
   "policyName": "Policy for sponsored v1",
   "fee": 100000000,
   "description": "Privacy for sponsored",
   "owners": [
     "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
     "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
     "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
   ],
   "type": 112,
   "version": 2,
   "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
   "feeAssetId": "G16FvJk9vabwxjQswh9CQAhbZzn3QrwqWjwnZB3qNVox",
   "proofs": [
       "3vDVjp6UJeN9ahtNcQWt5WDVqC9KqdEsrr9HTToHfoXFd1HtVwnUPPtJKM8tAsCtby81XYQReLj33hLEZ8qbGA3V" ],
   "recipients": [
      "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
      "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
      "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
      "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
      "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx"
   ],
   "id": "EyymzQcM2LrsgGDFFxeGn8DhahJbFYmorcBrEh8phv5S",
   "timestamp": 1585307711344 }
```

3.2.1.2 Информация о транзакциях

Группа `transactions` также включает следующие методы получения информации о транзакциях в блокчейне:

GET /transactions/info/{id}

Получение информации о транзакции по ее идентификатору `{id}`. Идентификатор транзакции указывается в ответе метода **POST /transactions/broadcast**.

Метод возвращает данные транзакции, аналогичные ответу метода **POST /transactions/broadcast**.

Пример ответа метода:


```
{
   "type": 4,
   "id": "52GG9U2e6foYRKp5vAzsTQ86aDAABfRJ7synz7ohBp19",
   "sender": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
   "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMhM3Uki7pLw",
   "recipient": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
   "assetId": "E9yZC4cVhCDfbjFJCc9CqkAtkoFy5KaCe64iaxHM2adG",
   "amount": 100000,
   "fee": 100000,
   "timestamp": 1549365736923,
   "attachment": "string",
   "signature": "GkniccUA79dBcwWgKjqB7vYHcnsj7caYETfncJhRkkaetbQon7DxbpMmvK9LYqUkirJp17geBJCRTNkHEoAjtsUm",
   "height": 7782
}
```


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

Метод возвращает данные последних `{limit}` транзакций адреса `{address}`.

Для каждой транзакции возвращаются данные, аналогичные ответу метода **POST /transactions/broadcast**.

Пример ответа метода для одной транзакции:


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


GET /transactions/unconfirmed

Метод возвращает данные всех транзакций из UTX-пула узла.

Для каждой транзакции возвращаются данные, аналогичные ответу метода **POST /transactions/broadcast**.

Пример ответа метода для одной транзакции:


```
[
  {
    "type": 4,
    "id": "52GG9U2e6foYRKp5vAzsTQ86aDAABfRJ7synz7ohBp19",
    "sender": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
    "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMhM3Uki7pLw",
    "recipient": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
    "assetId": "E9yZC4cVhCDfbjFJCc9CqkAtkoFy5KaCe64iaxHM2adG",
    "amount": 100000,
    "fee": 100000,
    "timestamp": 1549365736923,
    "attachment": "string",
    "signature": "GkнccUA79dBcwWgKjqB7vYHcnsj7caYETfncJhRkkaetbQon7DxbpMmvK9LYqUkirJp17geBJCRTNkHEoAjtsUm"
  }
]
```


GET /transactions/unconfirmed/size

Метод возвращает количество транзакций, находящихся в UTX-пуле в виде числа.

Пример ответа метода:


```
{
   "size": 4
}
```


GET /unconfirmed/info/{id}

Метод возвращает данные транзакции, находящейся в UTX-пуле, по ее `{id}`.

В ответе метода содержатся данные транзакции, аналогичные ответу метода **POST /transactions/broadcast**.

Пример ответа метода:


```
{
  "type": 4,
  "id": "52GG9U2e6foYRKp5vAzsTQ86aDAABfRJ7synz7ohBp19",
  "sender": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
  "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMhM3Uki7pLw",
  "recipient": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
  "assetId": "E9yZC4cVhCDfbjFJCc9CqkAtkoFy5KaCe64iaxHM2adG",
  "amount": 100000,
  "fee": 100000,
  "timestamp": 1549365736923,
  "attachment": "string",
  "signature": "GkнccUA79dBcwWgKjqB7vYHcnsj7caYETfncJhRkkaetbQon7DxbpMmvK9LYqUkirJp17geBJCRTNkHEoAjtsUm",
  "height": 7782
}
```


POST /transactions/calculateFee

Метод возвращает сумму комиссии за отправленную транзакцию.

В запросе указываются параметры, аналогичные запросу **POST /transactions/broadcast**. В ответе метода возвращается идентификатор ассета, в котором взимается комиссия (`null` для WAVES).

Пример ответа метода:


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

3.2.2 REST API: формирование и проверка электронной подписи данных (PKI)

Для сетей, работающих с использованием алгоритмов ГОСТ криптографии, REST API-интерфейс предоставляет возможность формирования отсоединенной электронной подписи для передаваемых данных, а также ее проверки. Для формирования и проверки электронных подписей предусмотрена группа методов `pki`:

- **POST /pki/sign**,
- **POST /pki/verify**,
- **GET /pki/keystoreAliases**.

Эти методы требуют авторизации.

Важно: Методы **pki/sign** и **pki/verify**, описанные ниже в данном разделе, доступны только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

GET /pki/keystoreAliases

Метод возвращает список с именами всех доступных хранилищ закрытых ключей ЭП.
Пример ответа метода:

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

POST /pki/sign

Метод формирует отсоединённую ЭП для данных, передаваемых в запросе.

Важно: Метод **POST /pki/sign** доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

Запрос состоит из следующих полей:

- `inputData` – данные, для которых требуется ЭП (в виде массива байт в кодировке base64);
- `keystoreAlias` – имя хранилища для закрытого ключа ЭП;
- `password` – пароль хранилища для закрытого ключа;
- `sigType` – формат ЭП. Поддерживаемые форматы:
  - 1 – CAdES-BES;
  - 2 – CAdES-X Long Type 1;
  - 3 – CAdES-T.

Метод возвращает поле `signature`, содержащее сгенерированную отсоединенную ЭП.

Примеры запроса и ответа метода:

Запрос:

```
{
  "inputData" : "SGVsbG8gd29ybGQh",
  "keystoreAlias" : "key1",
  "password" : "password",
  "sigType" : 1,
}
```

Ответ:

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

POST /pki/verify

Метод предназначен для проверки отсоединённой ЭП для данных, передаваемых в запросе.

Важно: Метод **POST /pki/verify** доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

Запрос к методу POST /pki/verify состоит из следующих полей:

- `inputData` – данные, закрытые ЭП (в виде массива байт в кодировке base64);
- `signature` – электронная подпись в виде массива байт в кодировке base64;
- `sigType` – формат ЭП. Поддерживаются значения:
  - 1 – CAdES-BES;
  - 2 – CAdES-X Long Type 1;
  - 3 – CAdES-T;
- `extended_key_usage_list` – список объектных идентификаторов (OID) криптографических алгоритмов, которые используются при формировании ЭП (опциональное поле).

Ответ метода содержит поле `sigStatus` с булевым типом данных: `true` – подпись действительна, `false` – подпись скомпрометирована.

Примеры запроса и ответа метода:

Запрос:

```
{
  "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"
  ]
}
```

Ответ:

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

3.2.2.1 Проверка УКЭП

Метод **POST /pki/verify** имеет возможность проверки усиленной квалифицированной электронной подписи (УКЭП).

Важно: Метод **POST /pki/verify** доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

Для корректной проверки УКЭП установите на ваш узел корневой сертификат ЭЦП удостоверяющего центра (УЦ), при помощи которого будет осуществляться валидация подписи.

Корневой сертификат устанавливается в хранилище сертификатов **cacerts** используемой вами виртуальной машины Java (JVM) при помощи утилиты **keytool**:

```
sudo keytool -import -alias certificate_alias -keystore path_to_your_JVM/lib/security/cacerts file path_to_the_certificate/cert.cer
```

После флага `-alias` укажите предпочтительное вам имя сертификата в хранилище.

Хранилище сертификатов cacerts расположено в поддиректории `/lib/security/` вашей виртуальной машины Java. Чтобы узнать путь к виртуальной машине на Linux, воспользуйтесь следующей командой:

```
readlink -f /usr/bin/java | sed "s:bin/java::"
```

Затем добавьте к полученному пути `/lib/security/cacerts` и вставьте полученный абсолютный путь к **cacerts** после флага `-keystore`.

После флага `-file` укажите абсолютный или относительный путь к полученному сертификату ЭЦП удостоверяющего центра.

Пароль по умолчанию для **cacerts** – `changeit`. При необходимости вы можете изменить его при помощи утилиты **keytool**:

```
sudo keytool -keystore cacerts -storepasswd
```

3.2.3 REST API: получение сертификатов

Для запроса у узла сертификата из хранилища сертификатов предусмотрена группа методов `/pki/certificate`. Методы этой группы позволяют получить сертификат по разным полям:

- `/pki/certificate/by-dn/{percent-encoded-DN}` – по полю `DN` (distinguished name),
- `/pki/certificate/by-dn-hash/{DN-hash-string%` – по хэш-значению поля `DN Hash`,
- `/pki/certificate/by-public-key/{public-key-base58}` – по полю `publicKey`,
- `/pki/certificate/by-fingerprint/{fingerprint-base64}` – по полю `fingerprint`.

В запросе эти методы принимают значение соответствующего поля сертификата и, опционально, параметр `plainText`, который задаёт формат ответа.

Если сертификат существует, то в ответе каждого из этих методов узел возвращает сертификат в формате DER (как он и записан в хранилище сертификатов узла), байты кодируются в формат Base64. Если в запросе метода параметру `plainText` задано значение `true`, то сертификат возвращается в формате plainText.

Если сертификата не существует, то в ответе каждого из этих методов возвращается ошибка с кодом `404 Not Found`.

3.2.3.1 GET /pki/certificate/by-dn/{percent-encoded-DN}

Метод возвращает сертификат по полю `DN`.

Пример запроса метода:

```
{
  "DN": "CN=Steve Kille,O=Isode Limited,C=GB",
  "plainText": false
}
```

3.2.3.2 GET /pki/certificate/by-dn-hash/{DN-hash-string}

Метод возвращает сертификат по хэш-значению (SHA-1) поля поля `DN` сертификата.

3.2.3.3 GET /pki/certificate/by-public-key/{public-key-base58}

Метод возвращает сертификат по ключу проверки ЭП (поле `publicKey`).

3.2.3.4 GET /pki/certificate/by-fingerprint/{fingerprint-base64}

Метод возвращает сертификат по его отпечатку (поле `fingerprint`).

3.2.4 REST API: реализация методов шифрования

Для реализации методов шифрования произвольных данных при помощи алгоритмов шифрования, применяемых на Платформе «Конфидент», а также расшифрования этих данных предусмотрены методы REST API группы `crypto`. Эти методы требуют авторизации.

Важно: Методы **crypto/encryptSeparate**, **crypto/encryptCommon** и **crypto/decrypt**, описанные ниже в подразделах данного раздела, доступны только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

POST /crypto/encryptSeparate

Шифрование данных, переданных в запросе, уникальными ключами CEK отдельно для каждого получателя, каждый CEK шифруется (*оборачивается*) отдельным ключом KEK.

В запросе подаются следующие данные:

- `sender` – адрес отправителя данных;
- `password` – пароль к зашифрованным данным;
- `encryptionText` – шифруемые данные (в виде строки);
- `recipientsPublicKeys` – публичные ключи получателей-участников сети;
- `crypto_algo` – используемый алгоритм шифрования. Доступные значения:
  - `gost-3412-2015-k` – поддержка только алгоритма ГОСТ 34.12-2015;
  - `aes` – поддержка только алгоритма AES.

  Подробнее алгоритмы шифрования, используемые на платформе, описаны в документации, которая поставляется вместе с дистрибутивом платформы.

В ответе метода поступают следующие данные для каждого получателя:

- `encrypted_data` – зашифрованные данные;
- `public_key` – публичный ключ получателя;
- `wrapped_key` – результат шифрования ключа для получателя.

Примеры запроса и ответа метода:

Запрос:

```
{
  "sender": "3MsHHc8LvyjPCKeSst9vsYcsHeQVzH6YJkL",
  "password": "",
  "encryptionText": "some string to encrypt",
  "recipientsPublicKeys": [
      "3MuNFC1Z8Tuy73pMzVUT6yowk4anWA8MNNE"
  ],
  "cryptoAlgo": "aes" }
}
```

Ответ:

```
{
  "encryptedText": "IZ5Kk5YNspMWl/jmlTizVxD6Nik=",
  "publicKey": "5R65oLxp3iwPekwirA4VwwUXaySz6W6YKXBKBRL352pwwcpsFcjRHJ1VVHLp63LkrkxsNod64V1pffeiZz5i2qXc",
  "wrappedKey": "uWVoxJAzruwTDDSbphDS31TjSQX6CSWXivp3x34uE3XtnMqqK9swoaZ3LyAgFDR7o6CfkgzFkWmTen4qAZewPfBbwR"
}
```

Важно: Метод **crypto/encryptSeparate** доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

POST /crypto/encryptCommon

Шифрование данных, переданных в запросе, единым ключом CEK для всех получателей, каждый ключ CEK шифруется (*оборачивается*) отдельным ключом KEK для каждого получателя.

Важно: Метод **crypto/encryptCommon** доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

В запросе **POST/crypto/encryptCommon** подаются данные, аналогичные запросу POST/crypto/encryptSeparate.

В ответе метода поступают следующие данные:

- `encrypted_data` – зашифрованные данные;

- `recipient_to_wrapped_structure` – структура в формате «ключ : значение», содержащая публичные ключи получателей с соответствующими результатами шифрования ключей для каждого из них.

Пример ответа метода:

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

POST /crypto/decrypt

Расшифрование данных, зашифрованных при помощи криптографического алгоритма, используемого сетью. Расшифрование осуществляетсяпри наличии ключа получателя сообщения в keystore узла.

Важно: Метод **crypto/decrypt** доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

В запросе подаются следующие данные:

- `recipient` – публичный ключ получателя из keystore узла;
- `password` – пароль к зашифрованным данным;
- `encryptedText` – зашифрованная строка;
- `wrappedKey` – результат шифрования ключа для указанного получателя;
- `senderPublicKey` – публичный ключ отправителя данных;
- `cryptAlgo` – используемый алгоритм шифрования; доступные значения:
  - `gost-3412-2015-k` – поддержка только алгоритма ГОСТ 34.12-2015;
  - `aes` – поддержка только алгоритма AES.

В ответ на запрос поступает поле `decryptedText`, содержащее расшифрованную строку.

Примеры запроса и ответа метода:

Запрос:

```
{
  "recipient": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "password": "12345qwert",
  "encryptedText": "t859AE7idnjPpn3lUiorfzSGwcGPMVdOhQe1HAhoI0MOXOQPBc8TUhn+8pKRCL8evH2Ra9Vc",
  "wrappedKey": "2nfob2yW76xj2rQBWZkzFD2UjYymWqQUCpFqbSWQiSYnuaw6DZoAde8KsTCMxPFVHA",
  "senderPublicKey": "CgqRPcPnexY533gCh2SSvBXh5bca1qMs7KFGntawHGww",
  "cryptoAlgo": "aes"
}
```

Ответ:

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

3.2.5 REST API: обмен конфиденциальными данными и получение информации о группах доступа

Подробнее об обмене конфиденциальными данными и группах доступа см. раздел «Обмен конфиденциальными данными» ниже.

Для реализации этих функций при помощи REST API предусмотрен набор методов группы `Privacy`. Эти методы требуют авторизации.

Важно: Методы **/privacy/sendData**, **/privacy/sendDataV2** и **/privacy/sendLargeData**, а также все остальные методы работы с конфиденциальными данными, описанные ниже в подразделах данного раздела, доступны только в тестовом режиме функционирования Платформы «Конфидент», то есть когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии»

документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

3.2.5.1 Авторизация методов группы Privacy в тестовом режиме

Для использования методов REST API группы `Privacy` в тестовом режиме функционирования Платформы «Конфидент» требуется авторизация по api-key. Авторизация методов реализована следующим образом:

- в случае api-key авторизации требуется PrivacyApiKey;

Для каждого из методов необходимо передавать следующие данные:

- policyRecipients — contractAuth | userAuth;
- policyOwners — userAuth;
- policyHashes — contractAuth | userAuth;
- policyItemData — contractAuth | privacyAuth;
- policyItemLargeData — contractAuth | privacyAuth;
- policyItemInfo — contractAuth | privacyAuth;
- policyItemsInfo — contractAuth | privacyAuth;
- sendData — privacyAuth;
- sendDataV2 — privacyAuth;
- forceSync — privacyAuth;
- forceSyncByPolicyId — privacyAuth;
- sendLargeData — privacyAuth,

где

- contractAuth — токен авторизации смарт контракта, передаваемый в заголовке „X-ContractApi-Token“ к запросу;
- userAuth — api-key пользователя, передаваемый в заголовке „X-Api-Key“ к запросу;
- privacyAuth — api-key privacy пользователя в заголовке „X-Api-Key“ к запросу.

POST /privacy/sendData

Метод предназначен для отправки в блокчейн конфиденциальных данных, доступных только для участников группы доступа, определенной для этих данных.

Примечание: Для отправки данных размером более20 МБ используйте метод POST/privacy/sendLargeData.

Запрос метода POST /privacy/sendData должен содержать следующую информацию:

- `sender` – блокчейн-адрес, от которого должны рассылаться данные (соответствуют значению параметра `privacy.owner-address` в конфигурационном файле узла);
- `password` – пароль для доступа к закрытому ключу keystore узла;
- `policyId` – идентификатор группы, которая будет иметь доступ к отправляемым данным;
- `info` – информация об отправляемых данных;
- `data` – строка, содержащая данные в формате base64;
- `hash` – sha256-хэш данных в формате base58;
- `broadcast` – если передается значение `true`, то созданная PolicyDataHash транзакция отправляется в блокчейн, если `false`, то транзакция и сообщение о наличии данных (Privacy Inventory) не отправляется; подробнее см. ниже.
- `certificates` – цепочка сертификатов байтами в формате DER; параметр является обязательным при одновременном соблюдении следующих условий:
  - используется тестовый режим PKI (то есть в конфигурационном файле узла параметру node.crypto.pki.mode присвоено значение TEST. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. Блокчейн-платформа «Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы),
  - новый пользователь, который не является владельцем узла (node-owner), делает свою первую транзакцию.

В этом случае необходимо в запросе в поле `certificates` передать цепочку сертификатов пользователя; в других случаях поле `certificates` является необязательным.

В результате отправки запроса будет сформирована транзакция 114 PolicyDataHash, которая отправит хэш конфиденциальных данных в блокчейн.

Параметр `broadcast`

Для снижения вероятности ошибок доставки данных рекомендуется установить для параметра `broadcast` значение `false`, если после отправки данных с помощью API метода **sendData** отправляется атомарная транзакция, которая содержит транзакцию CreatePolicy и транзакцию PolicyDataHash.

Примеры запроса и ответа метода:

Запрос:


```
{
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "password": "apgJP9atQccdBPA",
  "policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "info": {
    "filename":"Service contract #100/5.doc",
    "size": 2048,
    "timestamp": 1000000000,
    "author": "AIvanov@org.com",
    "comment": "some comments"
  },
  "data": "TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyByZWFzb24sIGJ1dCBieSB0aGlzIHNpbmd1bGFyIHBhc3Npb
           24gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaCBpcyBhIGx1c3Qgb2YgdGhlIG1pbmQsIHRoYXQgYnkgYSBwZXJzZXZlcmFuY2
           Ugb2YgZGVsaWdodCBpbiB0aGUgY29udGludWVkIGFuZCBpbmRlZmF0aWdhYmxlIGdlbmVyYXRpb24gb2Yga25vd2xlZGdlLCB
           leGNlZWRzIHRoZSBzaG9ydCB2ZWhlbWVuY2Ugb2YgY55IGNhcm5hbCBwbGVhc3VyZS4=",
  "hash": "FRog42mnzTA292ukng6PHoEK9Mpx9GZNrEHecfvpwmta"
  "broadcast": false
}
```


Ответ:


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


POST /privacy/sendDataV2

Метод **POST /privacy/sendDataV2** аналогичен методу **POST /privacy/sendData**, однако позволяет приложить файл в окне Swagger, не прибегая к его конверсии в формат base64. Метод предоставляет возможность потоковой передачи данных. Поле `Data` в этой версии метода отсутствует.

Когда новый пользователь, который не является владельцем узла (node-owner), в тестовом режиме PKI (параметру `node.crypto.pki.mode` присвоено значение `TEST`) делает свою первую транзакцию, ему необходимо в запросе в поле `certificates` приложить цепочку своих сертификатов. В других случаях поле `certificates` является необязательным. Подробнее о параметре `node.crypto.pki.mode` см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. Блокчейн-платформа «Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы).

Примечание: Для отправки данных размером более 20 МБ используйте метод POST/privacy/sendLargeData.

Примеры запроса и ответа метода:

Запрос:


```
{
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "password": "apgJP9atQccdBPA",
  "policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
```

```
  "info": {
    "filename":"Service contract #100/5.doc",
    "size": 2048,
    "timestamp": 1000000000,
    "author": "AIvanov@org.com",
    "comment": "some comments"
  },
  "hash": "FRog42mnzTA292ukng6PHoEK9Mpx9GZNrEHecfvpwmta"
  "broadcast": false
}
```

Ответ:

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

POST /privacy/sendLargeData

Метод **POST /privacy/sendLargeData** аналогичен методу POST /privacy/sendDataV2, но используется для отправки данных размером не менее 20 МБ.

Примечание: Для отправки данных размером менее 20 МБ используйте методы **POST /privacy/sendData** и **POST /privacy/sendDataV2**.

В конфигурационном файле узла в секции `node.privacy.service` можно настроить обратное давление на входящие фрагменты данных: задать максимальный размер для буфера в памяти (по умолчанию – 100 МБ).

Когда новый пользователь, который не является владельцем узла (node-owner), в тестовом режиме PKI (параметру `node.crypto.pki.mode` присвоено значение `TEST`) делает свою первую транзакцию, ему необходимо в запросе в поле `certificates` приложить цепочку своих сертификатов. В других случаях поле `certificates` является необязательным. Подробнее о параметре `node.crypto.pki.mode` см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. Блокчейн-платформа «Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы).

Примеры запроса и ответа метода:

Запрос:

```
{
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "password": "apgJP9atQccdBPA",
  "policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "info": {
    "filename":"Service contract #100/5.doc",
    "size": 2048,
    "timestamp": 1000000000,
    "author": "AIvanov@org.com",
    "comment": "some comments"
  },
  "hash": "FRog42mnzTA292ukng6PHoEK9Mpx9GZNrEHecfvpwmta"
  "broadcast": false
}
```

Ответ:


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


GET /privacy/{policy-id}/recipients

Метод предназначен для получения адресов всех участников, записанных в группу `{policy-id}`.
В ответе метода возвращается массив строк с адресами участников группы доступа.
Пример ответа метода:


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


GET /privacy/{policy-id}/owners

Метод предназначен для получения адресов владельцев группы доступа `{policy-id}`.
В ответе метода возвращается массив строк с адресами владельцев группы доступа.
Пример ответа метода:


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


GET /privacy/{policy-id}/hashes

Метод предназначен для получения массива идентификационный хэшей данных, которые привязаны к группе доступа `{policy-id}`.
В ответе метода возвращается массив строк с идентификационными хэшами данных группы доступа.
Пример ответа метода:


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


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

Метод предназначен для получения пакета конфиденциальных данных группы доступа `{policyId}` по идентификационному хэшу `{policyItemHash}`.
В ответе метода возвращается хэш-сумма конфиденциальных данных.
Пример ответа метода:


```
  c29tZV9iYXNlNjRfZW5jb2RlZF9zdHJpbmc=
```


GET /privacy/{policyId}/getLargeData/{policyItemHash}

Метод предназначен для получения пакета конфиденциальных данных группы доступа `{policyId}` по хэш-значению `{policyItemHash}`.
Метод возвращает стрим, что позволяет пользователю скачать файл с данными неограниченного объёма.

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

Метод предназначен для получения метаданных для пакета конфиденциальных данных группы `{policyId}` по идентификационному хэшу `{policyItemHash}`.
В ответе метода возвращаются следующие данные:

- `sender` – адрес отправителя конфиденциальных данных;
- `policy_id` – идентификатор группы доступа;
- `type` – тип конфиденциальных данных `(file)`;
- `info` – массив данных о файле:
  - `filename` – имя файла;
  - `size` – размер файла;
  - `timestamp` – временная метка размещения файла в формате Unix Timestamp (в миллисекундах);
  - `author` – автор файла;
  - `comment` – опциональный комментарий к файлу;
- `hash` – идентификационный хэш конфиденциальных данных.

Пример ответа 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

Метод предназначен для принудительного получения пакета конфиденциальных данных. Применяется в случае, если транзакция с конфиденциальными данными для группы доступа присутствует в блокчейне, но по какой-либо причине эти данные не были записаны в хранилище конфиденциальных данных узла. В этом случае метод позволяет принудительно скачать отсутствующие данные. Метод синхронизирует данные по всем группам доступа к конфиденциальным данным.

Запрос метода содержит следующие данные:

- `sender` – адрес узла-участника группы доступа, отправляющего запрос;
- `policy` – идентификатор группы доступа;
- `source` – адрес узла, с которого должны скачиваться отсутствующие данные. В случае, если узел неизвестен, установите параметр на `null` или оставьте поле пустым: в этом случае скачивание файла будет произведено из хранилища первого узла из списка группы доступа.

Ответ метода содержит поле `result` с результатом получения данных и поле `message` с текстом возможной ошибки. В случае успешного получения возвращается значение `success`, конфиденциальные данные записываются в хранилище узла.

В случае возникновения ошибки возвращается значение `error`, в поле `message` приводится описание ошибки.

Примеры запроса и ответа метода:

Запрос:

```
{
  "sender": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
  "policy": "my_policy",
  "source": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz"
}
```

Ответ:

```
{
  "result": "error"
  "message": "Address '3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8' not in policy 'my_policy'"
}
```

GET /privacy/forceSync/{policyId}

Метод аналогичен методу **POST /privacy/forceSync** с той разницей, что синхронизирует данные по указанной группе доступа к конфиденциальным данным (`policyId`).

POST /privacy/getInfos

Метод предназначен для получения массива метаданных конфиденциальных данных по идентификатору группы доступа и хэш-значению.

Запрос метода содержит следующие данные:

- `policiesDataHashes` – массив данных с двумя элементами для каждой отдельной группы доступа:
  - `policyId` – идентификатор группы доступа,
  - `datahashes` – массив хэшей конфиденциальных данных для получения метаданных по каждому из них.

В ответе метода для каждого отдельного хэша конфиденциальных данных возвращается массив данных, аналогичный ответу метода **GET /privacy/{policyId}/getInfo/{policyItemHash}**.

Примеры запроса и ответа метода:

Запрос:

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

Ответ:

```
{
   "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"
               }
            }
            }
          ]
      ]
}
```

3.2.6 REST API: работа с лицензиями узла (ноды)

Для работы с лицензиями Платформы «Конфидент» предусмотрена группа методов `licenses`. Эти методы требуют авторизации.

Важно: Все методы работы с лицензиями используют аутентификацию по списку tls-whitelist и доступны только администратору сети. Настройки авторизации описаны в разделе «Тонкая настройка платформы: настройка авторизации для gRPC и REST API» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

GET /licenses

Метод возвращает информацию о всех загруженных лицензиях.

В ответе для каждой лицензии поступает набор данных `license`, в котором содержатся параметры, указанные в файле лицензии, полученном от Платформы «Конфидент».

Пример ответа метода для одной лицензии:


```
[
  {
    "license": {
      "version": 1,
      "id": "a3d0d17e-eb05-45ac-906c-da847a9d726d",
      "issued_at": "2021-01-28T15:39:59.456Z",
      "node_owner_address": "3JNFkQ2cVu7ndEHLCs9A5HT63jSi1TV3mWK",
      "valid_after": "2021-01-29",
      "valid_before": "2022-11-20",
      "features": [
        "all_inclusive"
      ]
    },
    "signer_public_key": "p9HrAcGytSBxixJnQXQ87SNXPoXTdnwRzo4FMFvvbNSPzCToqdpJrcgFP6wxmsG23wBfYzcth",
    "signature": "jNjwCXdMPxmdaibXtjYSd8WocFinXKNsrTdPkbWrPTkQstswBp9SHFe",
    "signer_id": "2WDmdaibXtjYSd8WocFinX"
  }
]
```


GET /licenses/status

Метод возвращает статус активации лицензии узла. В ответе метода поступают следующие данные:

- `status` – статус активации лицензии:
  - `TRIAL` – активна пробная лицензия (максимальная высота блокчейна - 30000 блоков), по завершении пробного периода валидных лицензий нет;
  - `TRIAL_EXPIRED` – пробная лицензия истекла, валидных лицензий нет;
  - `ACTIVE` – валидная лицензия активна на момент запроса;
  - `PENDING` – на момент запроса активной лицензии нет, есть валидная лицензия, начинающаяся с более поздней даты: этот статус поступает по окончании пробного периода при наличии валидной лицензии с более поздней датой начала;
  - `EXPIRED` – валидная лицензия на момент запроса истекла, валидных лицензий с более поздней датой начала нет.
- `description` – краткое описание статуса, оставшееся количество блоков или дата истечения активной лицензии.

Пример ответа метода:


```
{
  "status" : "TRIAL",
  "description" : "Trial period is active. Blocks before expiration: 23412"
}
```


POST /licenses/upload

Метод добавляет новую лицензию для узла. Параметры, которые передаются в JSON-формате в запросе, указаны в файле, предоставляемом специалистами ООО «Веб3 Интегратора» при оформлении лицензии.

Примеры запроса и ответа метода:

Запрос:


```
{
  "license": {
    "version": 1,
    "id": "a3d0d17e-eb05-45ac-906c-da847a9d726d",
    "issued_at": "2021-01-28T15:39:59.456Z",
    "node_owner_address": "3JNFkQ2cVu7ndEHLCs9A5HT63jSi1TV3mWK",
    "valid_after": "2021-01-29",
    "valid_before": "2022-11-20",
    "features": [
      "all_inclusive"
    ]
  },
  "signer_public_key": "p9HrAcGytSBxixJnQXQ87SNXPoXTdnwRzo4FMFvvbNSPzCToqdpJrcgFP6wxmsG23wBfYzcth",
  "signature": "jNjwCXdMPxmdaibXtjYSd8WocFinXKNsrTdPkbWrPTkQstswBp9SHFe",
  "signer_id": "2WDmdaibXtjYSd8WocFinX"
}
```


Ответ:


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


DELETE /licenses/{license_id}

Метод удаляет загруженную лицензию по ее идентификатору `{license_id}`. Идентификатор лицензии указан в файле лицензии, который вы получите от специалистов ООО «Веб3 Интегратора», а также в ответе метода GET /licenses.

Пример ответа метода:


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


3.2.7 REST API: валидация адресов и псевдонимов участников сети

Для валидации адресов и псевдонимов в сети предусмотрены следующие методы группы `addresses`. Эти методы требуют авторизации.

GET /addresses/validate/{addressOrAlias}

Валидация заданного адресата или его псевдонима `{addressOrAlias}` в блокчейн-сети работающего узла.

Пример ответа метода:


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


POST /addresses/validateMany

Валидация нескольких адресов или псевдонимов, передаваемых в поле `addressesOrAliases` в виде массива. Информация в ответе для каждого адреса идентична ответу метода **GET/addresses/validate/{addressOrAlias}**.

Примеры запроса и ответа для одного адреса, одного существующего и одного несуществующего псевдонимов:

Запрос:


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

Ответ:

```
{
  validations: [
    {
      addressOrAlias: "3HSVTtjim3FmV21HWQ1LurMhFzjut7Aa1Ac",
      valid: true
    },
    {
      addressOrAlias: "alias:T:asdfghjk",
      valid: true
    },
    {
      addressOrAlias: "alias:T:1nvAliDAl1ass99911%^&$$$ ",
      valid: false,
      reason: "GenericError(Alias should contain only following characters:
 -.0123456789@_abcdefghijklmnopqrstuvwxyz)"
    }
  ]
}
```

3.2.8 REST API: подписание и валидация сообщений в блокчейне

Для подписания и валидации сообщений предусмотрены следующие методы группы `addresses`. Эти методы требуют авторизации.

Важно: Методы **POST /addresses/sign**, **POST /addresses/signText**, **POST /addresses/verify** и **POST /addresses/verifyText**, описанные ниже в подразделах данного раздела, доступны только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

POST /addresses/sign/{address}

Метод подписывает строку, переданную в поле `message`, ключом ЭП адресата `{address}`, а затем сериализует ее в формат base58.

Важно: REST API метод POST /addresses/sign/{address} доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

В ответе метода возвращается сериализованная строка, ключ проверки ЭП и подпись адресата.

Примеры запроса и ответа метода:

Запрос:

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

Ответ:

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

POST /addresses/verify/{address}

Проверка подписи сообщения, выполненной адресатом `{address}`.

Важно: REST API метод POST /addresses/verify/{address} доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая

настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

Примеры запроса и ответа метода:

Запрос:

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

Ответ:

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

POST /addresses/signText/{address}

Метод подписывает строку, переданную в поле `message`, ключом ЭП адресата `{address}`. В отличие от метода **POST /addresses/sign/{address}**, строка передается в исходном формате.

Важно: REST API метод POST /addresses/signText/{address} доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

Примеры запроса и ответа метода:

Запрос:

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

Ответ:

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

POST /addresses/verifyText/{address}

Проверка подписи сообщения, выполненной адресатом `{address}` посредством метода POST /addresses/signText/{address}.

Важно: REST API метод POST /addresses/verifyText/{address} доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

Примеры запроса и ответа метода:

Запрос:

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

Ответ:

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

3.2.9 REST API: информация о конфигурации и состоянии узла (ноды), остановка узла (ноды)

Для получения информации о конфигурации узла предусмотрены две группы методов:

- `node` – получение основных конфигурационных параметров узла, информации о состоянии узла, остановка узла, изменение уровня логирования;
- `anchoring` – запрос `GET /anchoring/config`, возвращающий секцию `anchoring` конфигурационного файла узла.

Для получения основных конфигурационных параметров узла предусмотрены как методы, требующие авторизации, так и открытые методы.

3.2.9.1 Группа `node`:

GET /node/config

Метод возвращает основные конфигурационные параметры узла. Метод требует авторизации.

Пример ответа метода :


```
{
    {
      "version": "1.3.0-RC7",
      "gostCrypto": true,
      "chainId": "V",
      "consensus": "CFT",
      "minimumFee": {
          "3": 0,
          "4": 0,
          "5": 0,
          "6": 0,
          "7": 0,
          "8": 0,
          "9": 0,
          "10": 0,
          "11": 0,
          "12": 0,
          "13": 0,
          "14": 0,
          "15": 0,
          "102": 0,
          "103": 0,
          "104": 0,
          "106": 0,
          "107": 0,
          "111": 0,
          "112": 0,
          "113": 0,
          "114": 0
      },
      "additionalFee": {
          "11": 0,
          "12": 0
      },
      "maxTransactionsInMicroBlock": 500,
      "minMicroBlockAge": 0,
      "microBlockInterval": 1000,
      "blockTiming": {
          "roundDuration": 7000,
          "syncDuration": 700
    }
}
```


GET /node/owner

Метод возвращает адрес и ключ проверки ЭП владельца узла. Метод требует авторизации.
Пример ответа метода:


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


GET /node/status

Метод возвращает информацию о текущем состоянии узла. Метод не требует авторизации.

Пример ответа метода:

```
{
    "blockchainHeight": 47041,
    "stateHeight": 47041,
    "updatedTimestamp": 1544709501138,
    "updatedDate": "2018-12-13T13:58:21.138Z"
    "lastCheckTimestamp": 1543719501123,
}
```

При использовании алгоритмов ГОСТ криптографии на узле при возникновении ошибок метод вернет описание ошибки:

```
{
    "error": 199,
    "message": "Environment check failed: Supported JCSP version is 5.0.40424, actual is 2.0.40424"
}
```

GET /node/version

Метод возвращает версию узла. Метод не требует авторизации.
Пример ответа метода:

```
{
    "version": "Confident v1.9.0"
}
```

GET /node/logging

Метод отображает список логгеров, указанных при конфигурировании узла, и уровень логирования для каждого из них. Метод требует авторизации.

Важно: Метод GET /node/logging использует аутентификацию по списку tls-whitelist и доступен только администратору сети. Настройки авторизации описаны в разделе «Тонкая настройка платформы: настройка авторизации для gRPC и REST API» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

Уровни логирования узла:

- `ERROR` – логирование ошибок;
- `WARN` – логирование предупреждений;
- `INFO` – логирование событий узла;
- `DEBUG` – расширенная информация о событиях по каждому работающему модулю узла: запись произошедших событий и выполняемых действий;
- `TRACE` – подробная информация о событиях уровня DEBUG;
- `ALL` – отображение информации на всех уровнях логирования.

Пример ответа метода:

```
ROOT-DEBUG
akka-DEBUG
akka.actor-DEBUG
akka.actor.ActorSystemImpl-DEBUG
akka.event-DEBUG
akka.event.slf4j-DEBUG
akka.event.slf4j.Slf4jLogger-DEBUG
com-DEBUG
com.github-DEBUG
com.github.dockerjava-DEBUG
com.github.dockerjava.core-DEBUG
com.github.dockerjava.core.command-DEBUG
com.github.dockerjava.core.command.AbstrDockerCmd-DEBUG
com.github.dockerjava.core.exec-DEBUG
```

GET /node/healthcheck

Метод производит проверку доступности внешнего сервиса, указанного в запросе. Метод не требует авторизации.

Важно: Метод GET /node/healthcheck использует аутентификацию по списку tls-whitelist и доступен только администратору сети. Настройки авторизации описаны в разделе «Тонкая настройка платформы: настройка авторизации для gRPC и REST API» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

В запросе должен быть указан параметр `service`, который может принимать одно из следующих значений:

- `docker;`
- `privacy-storage;`
- `anchoring-auth.`

По умолчанию используется значение `docker`.

Метод возвращает значение `200 OK` и пустой ответ, если проверка прошла успешно, иначе – `503 Service Unavailable` и описание ошибки. Если один из внешних сервисов не настроен (на узле отключена функциональность докер смарт контрактов, отключена настройка групп доступа к конфиденциальным данным, отключен анкоринг), метод возвращает ошибку `404 Not Found` с сообщением о том, что определенная настройка отключена.

Пример ответа метода:

```
{
  "error": 199,
  "message": "Docker host is not available"
}
```

POST /node/logging

Метод предназначен для смены уровня логирования для выбранных логгеров. Метод требует авторизации.

Важно: Метод POST /node/logging использует аутентификацию по списку tls-whitelist и доступен только администратору сети. Настройки авторизации описаны в разделе «Тонкая настройка платформы: настройка авторизации для gRPC и REST API» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

Пример запроса метода:

```
{
    "logger": "com.wavesplatform.Application",
    "level": "ALL"
}
```

POST /node/stop

Метод останавливает узел; ответа не предусмотрено. Метод требует авторизации.

Важно: Метод POST `/node/stop` доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

3.2.9.2 Группа `anchoring`:

GET /anchoring/config

Метод выводит секцию `anchoring` конфигурационного файла узла. Метод требует авторизации.
Пример ответа метода:

```
{
     "enabled": true,
     "currentChainOwnerAddress": "3FWwx4o1177A4oeHAEW5EQ6Bkn4Lv48quYz",
     "targetnetNodeAddress": " node.web3techservices.com:443",
     "targetnetSchemeByte": "L",
     "targetnetRecipientAddress": "3JzVWCSV6v4ucSxtGSjZsvdiCT1FAzwpqrP",
     "targetnetFee": 8000000,
     "currentChainFee": 666666,
     "heightRange": 5,
     "heightAbove": 3,
     "threshold": 10
}
```

3.2.10 REST API: информация об участниках сети

Для получения информации об участниках сети предусмотрено три группы методов:

- `addresses` – методы, предназначенные для получения информации об адресах участников сети;
- `alias` – получение адреса участника по установленному для него псевдониму или псевдонима по адресу участника;
- `leasing` – запрос `GET /leasing/active/{address}`, выводящий список транзакций лизинга, в которых адрес принимал участие как отправитель или получатель.

Методы требует авторизации.

3.2.10.1 Группа `addresses`:

GET /addresses

Получение всех адресов участников, ключевые пары которых хранятся в keystore узла.
Пример ответа метода:

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

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

Получение адресов участников, которые хранятся в keystore узла в заданном диапазоне: от адреса `{from}` до адреса `{to}, где нумерация начинается с 0 ({from})`.

Формат ответа метода идентичен формату `GET /addresses`.

GET /addresses/balance/{address}

Получение баланса для адреса {address}.
Пример ответа метода:

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

POST /addresses/balance/details

Получение подробной информации о балансе для списка адресов, который указывается в виде массива в поле `addresses` при запросе.

Параметры, возвращаемые в ответе метода:

- `regular` — сумма токенов, принадлежащих непосредственно участнику (**R**);
- `available` — общий баланс участника, за исключением средств, переданных участником в лизинг (**A = R – L**);
- `effective` — общий баланс участника, включая средства, переданные участнику в лизинг, и за вычетом средств, которые участник сам передал в лизинг (E = R + F – L);
- `generating` — генерирующий баланс участника, включая средства, переданные в лизинг, за последние 1000 блоков.

Переменные в скобках: **L** – средства, переданные участником в лизинг другим участникам, **F** – средства, полученные участником в лизинг.

Пример ответа метода для одного адреса:

```
[
  {
    "address": "3M4Bxh2VfzKFXqiQB8bDgRfVnPWrZUQ2MEF",
    "regular": 59899999999400000,
    "generating": 59899999999400000,
    "available": 59899999999400000,
    "effective": 59899999999400000
  }
]
```

GET /addresses/balance/details/{address}

Получение подробной информации о балансе для отдельного адреса. Информация в ответе идентична методу `POST /addresses/balance/details`.

Пример ответа метода:

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

GET /addresses/effectiveBalance/{address}

Получение общего баланса адреса, включая средства, переданные в лизинг.

Пример ответа метода:

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

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

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

Пример ответа для количества подтверждений >= 1:

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

GET /addresses/generatingBalance/{address}/at/{heigh}

Получение генерирующего баланса адреса на указанной высоте блокчейна `{height}`.
Пример ответа метода:

```
{
  "address": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "generatingBalance": 1011543800600
}
```

GET /addresses/scriptInfo/{address}

Получение данных о скрипте, установленном на адресе.
Параметры, возвращаемые в ответе метода:

- `address` – адрес в формате base58;
- `script` – тело скрипта в формате base64;
- `scriptText` – исходный код скрипта;
- `complexity` – сложность скрипта;
- `extraFee` – комиссия за исходящие транзакции, установленные скриптом.

Сложность скрипта – число от 1 до 100, отражающее количество вычислительных ресурсов, требуемое для исполнения скрипта.
Пример ответа 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
}
```

GET /addresses/publicKey/{publicKey}

Метод возвращает адрес участника на основании его публичного ключа.
Пример ответа метода:

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

GET /addresses/data/{address}

Метод возвращает данные, записанные на указанном адресе при помощи транзакций 12 (Data Transaction).
Пример ответа метода:

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

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

Метод возвращает данные, записанные на указанном адресе с ключом `{key}`. Этот ключ указывается в транзакции 12 (Data Transaction) в поле `data.key`.
Пример ответа метода:

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

3.2.10.2 Группа `alias`:

GET /alias/by-alias/{alias}

Получение адреса участника по его псевдониму `{alias}`.
Пример ответа метода:

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

GET /alias/by-address/{address}

Получение псевдонима участника по его адресу `{address}`.
Пример ответа метода:

```
[
    "alias:participant1",
]
```

3.2.10.3 Группа `leasing`:

GET /leasing/active/{address}

Метод возвращает список транзакций создания лизинга, в которых адрес принимал участие как отправитель или получатель.
Пример ответа метода с одной транзакцией:

```
[
  {
    "type": 8,
    "id": "2jWhz6uGYsgvfoMzNR5EEGdi9eafyCA2zLFfkM4NP6T7",
    "sender": "3PP6vdkEWoif7AZDtSeSDtZcwiqSfhmwttE",
    "senderPublicKey": "DW9NKLYeyoEWDqJKhWv87EdFfTqpFtJBWoCqfCVwRhsY",
    "fee": 100000,
    "timestamp": 1544390280347,
    "signature": "25kpwh7nYjRUtfbAbWYRyMDPCUCoyMoUuWTJ6vZQrXsZYXbdiWHa9iGscTTGnPFyegP82sNSfM2bXNX3K7p6D3HD",
    "version": 1,
    "amount": 31377465877,
    "recipient": "3P3RD3yJW2gQ9dSVwVVDVCQiFWqaLtZcyzH",
    "height": 1298747
  }
]
```

3.2.11 REST API: информация об активации новых функциональных возможностей платформы

GET /activation/status

Метод возвращает статус активации новых функциональных возможностей. Метод требует авторизации. Метод использует следующий список идентификаторов функциональных возможностей:

Идентификатор	Название
100	Алгоритм консенсуса LPoS
101	Поддержка gRPC смарт-контрактами Docker
119	Оптимизация производительности для алгоритма консенсуса PoA
120	Поддержка спонсорских транзакций
130	Оптимизация скорости работы с историей банов майнера
140	Поддержка атомарных транзакций
160	Поддержка параллельного создания liquid-block и микроблока
162	Валидация смарт-контрактов в блокчейне
173	Поддержка сбора инвентаризационной информации о микроблоках (версия 2)
180	Поддержка передачи больших файлов в подсистеме конфиденциальных данных

Ответ метода содержит следующие общие поля:

- `height` – текущая высота блокчейна;
- `votingInterval` – интервал проведения голосования за активацию;
- `votingThreshold`
- `nextCheck`

Далее выводится массив `features`, содержащий информацию по каждой отдельной функциональной возможности:

- `id` – идентификатор функциональной возможности;
- `description` - описание функциональной возможности;
- `blockchainStatus` – статус функциональной возможности в блокчейне:
  - `UNDEFINED` – функциональная возможность не активирована, голосование за нее не проводилось;
  - `APPROVED` – голосование за функциональную возможность проведено, активация будет произведена на установленной высоте блокчейна;
  - `ACTIVATED` – функциональная возможность активирована;
- `nodeStatus` – статус функциональной возможности на узле участника:
  - `VOTED` – узел проголосовал за активацию функциональной возможности;
  - `NOT IMPLEMENTED` – функциональная возможность не запущена на узле;
  - `IMPLEMENTED` – функциональная возможность запущена;
- `activationHeight` – высота блокчейна, на которой активируется функциональная возможность.

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

3.2.12 REST API: информация об используемом алгоритме консенсуса

Для получения информации, относящейся к используемому алгоритму консенсуса, предусмотрены методы группы `consensus`. Методы требуют авторизации.

Важно: Консенсусы PoA и PoS доступны только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Соответственно методы, которые описаны ниже и доступны при использовании алгоритма консенсусов PoA или POS, доступны только в тестовом режиме. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

GET /consensus/algo

Метод возвращает название используемого алгоритма консенсуса.
Пример ответа метода:

```
{
    "consensusAlgo": "Leased Proof-of-Stake (LPoS)"
}
```

GET /consensus/settings

Метод возвращает параметры используемого алгоритма консенсуса, заданные в конфигурационном файле узла.
Пример ответа метода:

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

GET /consensus/minersAtHeight/{height}

Метод возвращает очередь майнеров на высоте `{height}`. Доступен при использовании алгоритма консенсуса PoA.
Пример ответа метода:

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

GET /consensus/miners/{timestamp}

Метод возвращает очередь майнеров на время `{timestamp}` (указывается в формате Unix Timestamp, в миллисекундах). Доступен при использовании алгоритма консенсуса PoA.

Пример ответа метода:

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

GET /consensus/bannedMiners/{height}

Метод возвращает список заблокированных майнеров на высоте `{height}`. Доступен при использовании алгоритма консенсуса PoA.

Пример ответа метода:

```
{
  "miners": [
      "3N6pfQJyqjLCmMbU7G5sNABLmSF5aFT4KTF",
      "3NBbipRYQmZFudFCoVJXg9JMkkyZ4DEdZNS"
  ],
  "height": 440
}
```

GET /consensus/basetarget/{signature}

Метод возвращает значение базовой сложности (basetarget) создания блока по его подписи `{signature}`. Доступен при использовании алгоритма консенсуса PoS.

GET /consensus/basetarget

Метод возвращает значение базовой сложности (basetarget) создания текущего блока. Доступен при использовании алгоритма консенсуса PoS.

GET /consensus/generatingbalance/{address}

Метод возвращает генерирующий баланс, доступный для узла `{address}`, включая средства, переведенные участнику в лизинг. Доступен при использовании алгоритма консенсуса PoS.

GET /consensus/generationsignature/{signature}

Метод возвращает значение генерирующей подписи (generation signature) создания блока по его подписи `{signature}`. Доступен при использовании алгоритма консенсуса PoS.

GET /consensus/generationsignature

Возвращает значение генерирующей подписи (generation signature) текущего блока. Доступен при использовании алгоритма консенсуса PoS.

3.2.13 REST API: информация о смарт-контрактах

Для получения информации о смарт-контрактах, загруженных в сеть, предусмотрен набор методов группы `contracts`. Методы требуют авторизации.

GET /contracts

Метод возвращает информацию по всем смарт-контрактам, загруженным в сеть. Для каждого смарт контракта в ответе возвращаются следующие параметры:

- `contractId` – идентификатор смарт-контракта;
- `image` – имя Docker-образа смарт-контракта, либо его абсолютный путь в репозитории;
- `imageHash` – хэш-сумма смарт-контракта;
- `version` – версия смарт-контракта;
- `active` – статус смарт-контракта на момент отправки запроса:
  - `true` – запущен;
  - `false` – не запущен.

Пример ответа метода для одного смарт-контракта:


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


POST /contracts

Метод возвращает набор полей «ключ : значение», записанных в стейт одного или нескольких смарт контрактов. ID запрашиваемых смарт-контрактов указываются в поле contracts запроса.

Пример ответа метода для одного смарт-контракта:


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


GET /contracts/status/{id}

Метод возвращает статус исполняемой транзакции 103 на создание смарт-контракта по идентификатору транзакции `{id}`. Однако если после отправки транзакции в блокчейн узел перезапускается, метод не вернет корректное состояние этой транзакции.

В ответе метода возвращаются следующие параметры:

- `sender` – адрес отправителя транзакции;
- `senderPublicKey` – публичный ключ отправителя транзакции;
- `txId` – ID транзакции;
- `status` – статус транзакции: успешно попала в блок, подтверждена, отклонена;
- `code` – код ошибки (при наличии);
- `message` – сообщение о статусе транзакции;
- `timestamp` – временная метка в формате Unix Timestamp, в миллисекундах;
- `signature` – подпись транзакции.

Пример ответа метода:


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


GET /contracts/{contractId}

Метод возвращает результат исполнения смарт-контракта по его идентификатору `{contractId}`.
Пример ответа метода:


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


POST /contracts/{contractId}

Метод возвращает значения ключей из стейта смарт-контракта `{contractId}`. В запросе указываются следующие данные:

- `contractId` – идентификатор смарт-контракта;
- `limit` – ограничение количества выводимых блоков данных;
- `offset` – количество блоков данных для пропуска в выводе;
- `matches` – опциональный параметр для составления регулярного выражения, по которому фильтруются ключи.

Пример ответа метода:


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


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

Метод возвращает результат исполнения смарт-контракта по идентификатору транзакции 105.
В ответе метода возвращаются данные транзакции 105, а также результаты исполнения в поле `results`.
Пример ответа, когда смарт-контракт не исполнялся:


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

```
    "4BoG6wQnYyZWyUKzAwh5n1184tsEWUqUTWmXMExvvCU95xgk4UFB8iCnHJ4GhvJm86REB69hKM7s2WLAwTSXquAs"
  ],
  "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": []
}
```


GET /contracts/{contractId}/{key}

Возвращает значение ключа {key} исполненного смарт-контракта по его идентификатору.
Пример ответа метода:


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


3.2.14 REST API: информация о блоках сети

Для получения информации о различных блоках сети предусмотрена группа методов `blocks`. Эти методы требуют авторизации.

GET /blocks/height

Метод возвращает номер текущего блока в блокчейне (высоту блокчейна).
Пример ответа метода:


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


GET /blocks/height/{signature}

Метод возвращает высоту блока по его подписи `{signature}`.
Ответ метода содержит поле `height`, как и метод `GET /blocks/height`.

GET /blocks/first

Метод возвращает информацию о генезис-блоке сети. В ответе содержатся следующие параметры:

- `reference` – хэш-сумма генезис-блока;
- `blocksize` – размер генезис-блока;
- `signature` – подпись генезис-блока;
- `fee` – комиссия за транзакции, включенные в генезис-блок;
- `generator` – адрес создателя генезис-блока;
- `transactionCount` – количество транзакций 1 и 101, включенных в генезис-блок;
- `transactions` – массив с телами транзакций 1 и 101, включенных в генезис-блок;
- `version` – версия генезис-блока;
- `timestamp` – временная метка создания генезис-блока в формате Unix Timestamp (в миллисекундах);
- `height` – высота создания генезис-блока (1).

Пример ответа метода:


```
{
    "reference": "67rpwLCuS5DGA8KGZXKsVQ7dnPb9goRLoKfgGbLfQg9WoLUgNY77E2jT11fem3coV9nAkguBACzrU1iyZM4B8roQ",
    "blocksize": 1435,
    "signature": "4HENriUyMthzMSqWa5sYPFMATbzpQugTBMk6mXUh5HmnvHfUhmQk6EqmdhGvNFcUvTDrsyiVqkxtm8iiV2xNTSNK",
    "fee": 0,
    "generator": "3MvQKx98a713B28rdUAtbWJ8DFJEXhnTjKs",
    "transactionCount": 26,
    "transactions": [
      {
        "type": 1,
        "id": "2AdCY254MFSrgxpr6otBisV5Zz7neH8YoM6VGW5egoVJnwD8cJpYZVR42aVKTZnwGT9ee7LCpAGMNSUV86FEAGXu",
        "fee": 0,
        "timestamp": 1606211535610,
        "signature": "2AdCY254MFSrgxpr6otBisV5Zz7neH8YoM6VGW5egoVJnwD8cJpYZVR42aVKTZnwGT9ee7LCpAGMNSUV86FEAGXu",
        "recipient": "3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP",
        "amount": 1250000000000000
      },
      {
        "type": 1,
        "id": "5VC2LoFTbrfLkd48bjQkp8CmTyqXJSkJh723qxo9v5pz38tBUjRW9tHLuvwajSvkzQNFxrCc6Yjkgx5R2YR3x5VC",
        "fee": 0,
        "timestamp": 1606211535610,
        "signature": "5VC2LoFTbrfLkd48bjQkp8CmTyqXJSkJh723qxo9v5pz38tBUjRW9tHLuvwajSvkzQNFxrCc6Yjkgx5R2YR3x5VC",
        "recipient": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c",
        "amount": 300000000000000
      },
      {
        "type": 1,
        "id": "4cmwEkSnBLc3TBTPUiT7HwmdER25X7GzCj2mgiEJ8K149vnNa1orBZUNstwNXtXFyKcQbkRPym39d9wJXTE4wgbU",
        "fee": 0,
        "timestamp": 1606211535610,
        "signature": "4cmwEkSnBLc3TBTPUiT7HwmdER25X7GzCj2mgiEJ8K149vnNa1orBZUNstwNXtXFyKcQbkRPym39d9wJXTE4wgbU",
        "recipient": "3N9nNFySk1zVSVf9DUWR9DiBA1jEmmDDpaJ",
        "amount": 100000000000000
      },
      {
        "type": 1,
        "id": "5Etq3o1eWoN3bqR9cYV6149qxAE3ru4CoSCf1Mm5sSJEedcbmLhsbfg8rh4S6ESrAPq7ZEbghEgHjyb3xzUbDDRh",
        "fee": 0,
        "timestamp": 1606211535610,
        "signature": "5Etq3o1eWoN3bqR9cYV6149qxAE3ru4CoSCf1Mm5sSJEedcbmLhsbfg8rh4S6ESrAPq7ZEbghEgHjyb3xzUbDDRh",
        "recipient": "3N3jgxvmSsBBV4oz9BcKhT8War1em2sKoJn",
        "amount": 1000000000000000
      },
      {
        "type": 110,
        "id": "3HewQJtzuaumzX4TvmN7fxVCgnsWTTaLeQjYBVDDuYoEW2ijWd7JME8h1gtsqepv5SDhHPvoMesVNm96br8WRgF8",
        "fee": 0,
        "timestamp": 1606211535610,
        "signature": "3HewQJtzuaumzX4TvmN7fxVCgnsWTTaLeQjYBVDDuYoEW2ijWd7JME8h1gtsqepv5SDhHPvoMesVNm96br8WRgF8",
        "targetPublicKey": "56rV5kcR9SBsxQ9LtNmp6V72S4BDkZUJaA6ujZswDneDmCImeSG6UE2FQP1rPXdfpWQNunRw4aijGXxoK3o4puj",
        "target": "3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP"
      },
      {
        "type": 101,
        "id": "5r4uLWn3rwmqbBygNj29iR4YsiV82dYWFeCbepAHhKGXqnn27vE6i811U9H2UZgX8zNQYZciyw3PR6nAdwjSPSp5",
        "fee": 0,
        "timestamp": 1606211535609,
        "signature": "5r4uLWn3rwmqbBygNj29iR4YsiV82dYWFeCbepAHhKGXqnn27vE6i811U9H2UZgX8zNQYZciyw3PR6nAdwjSPSp5",
        "target": "3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP",
        "role": "permissioner"
      },
      {
        "type": 101,
        "id": "4pBwjviNLtSPEBY5YB7ZdUXVSFnEk4rgscW8r9QQKxdxQZzjwjdq1ZnruMxQo7tomQVJf1Ni6SyVxSHrQZhBJaFN",
        "fee": 0,
        "timestamp": 1606211535608,
        "signature": "4pBwjviNLtSPEBY5YB7ZdUXVSFnEk4rgscW8r9QQKxdxQZzjwjdq1ZnruMxQo7tomQVJf1Ni6SyVxSHrQZhBJaFN",
        "target": "3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP",
```

```
      "role": "miner"
    },
    {
      "type": 101,
      "id": "5kwQwLH8oTy1ztF6xxsBxE3MDGio1NJm8F7Mtpynf3QTw9CWCsp5Fio5SxLmPxnB1bUVQHMCHbQCD4wXJLJgjSrp",
      "fee": 0,
      "timestamp": 1606211535607,
      "signature": "5kwQwLH8oTy1ztF6xxsBxE3MDGio1NJm8F7Mtpynf3QTw9CWCsp5Fio5SxLmPxnB1bUVQHMCHbQCD4wXJLJgjSrp",
      "target": "3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP",
      "role": "connection_manager"
    },
    {
      "type": 101,
      "id": "62xS2qkR7chFMSdryTjwB15BKd4CH5Hwn9PbzasZo1Qx6Bwg82nixMPKRQobDy3JW7cLmzMHi97hJk1JSDqhwUgM",
      "fee": 0,
      "timestamp": 1606211535606,
      "signature": "62xS2qkR7chFMSdryTjwB15BKd4CH5Hwn9PbzasZo1Qx6Bwg82nixMPKRQobDy3JW7cLmzMHi97hJk1JSDqhwUgM",
      "target": "3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP",
      "role": "contract_developer"
    },
    {
      "type": 101,
      "id": "2sNwzGbwDL2Es53P8XY5wA9T9wwu3eXJbJUrtXJ9wg49urPjuBejWbidat2z3yZ8JrTpkWWFEsrerCtnC38XuRTJ",
      "fee": 0,
      "timestamp": 1606211535605,
      "signature": "2sNwzGbwDL2Es53P8XY5wA9T9wwu3eXJbJUrtXJ9wg49urPjuBejWbidat2z3yZ8JrTpkWWFEsrerCtnC38XuRTJ",
      "target": "3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP",
      "role": "issuer"
    },
    {
      "type": 110,
      "id": "4hLep3GngPEBH2xEmuUZ323muT8BstFdT552e42z6ZXCKGnF1PABGGjEiCkHfr6hMuyvRJ7axD9qoGeWQCU5yaCk",
      "fee": 0,
      "timestamp": 1606211535610,
      "signature": "4hLep3GngPEBH2xEmuUZ323muT8BstFdT552e42z6ZXCKGnF1PABGGjEiCkHfr6hMuyvRJ7axD9qoGeWQCU5yaCk",
      "targetPublicKey": "5nGi8XoiGjjyjbPmjLNy1k2bus4yXLaeuA3Hb7BikwD9tboFwFXJYUmto5Joox76c3pp2Mr1LjgodUJuxryCJofQ",
      "target": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c"
    },
    {
      "type": 101,
      "id": "nj9Xfqm3pPLmuLsWfDZx4htKaNKAyvhen7tF95T9YwdmK1pqkiCjtaV9AxCwzEceViyo5rHPapigxPyCZdBWvRn",
      "fee": 0,
      "timestamp": 1606211535604,
      "signature": "nj9Xfqm3pPLmuLsWfDZx4htKaNKAyvhen7tF95T9YwdmK1pqkiCjtaV9AxCwzEceViyo5rHPapigxPyCZdBWvRn",
      "target": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c",
      "role": "permissioner"
    },
    {
      "type": 101,
      "id": "24AmxdGyH3afYRxPXn5zqvU1Fro1MwVQPDqwkdjCKLddSEiKVhyeMHTAVrRpHu83ZDPMyQkf3ty161PrujmGYtef",
      "fee": 0,
      "timestamp": 1606211535603,
      "signature": "24AmxdGyH3afYRxPXn5zqvU1Fro1MwVQPDqwkdjCKLddSEiKVhyeMHTAVrRpHu83ZDPMyQkf3ty161PrujmGYtef",
      "target": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c",
      "role": "miner"
    },
    {
      "type": 101,
      "id": "4xsEQoh6Z4wDW6jT9UP3SqA1Yv5trbaGfF4uHajWxayBU8hrw2ZAYmtAWwDFytTdc6yqDepj6GwzxZuFYTq6638v",
      "fee": 0,
      "timestamp": 1606211535602,
      "signature": "4xsEQoh6Z4wDW6jT9UP3SqA1Yv5trbaGfF4uHajWxayBU8hrw2ZAYmtAWwDFytTdc6yqDepj6GwzxZuFYTq6638v",
      "target": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c",
      "role": "connection_manager"
    },
    {
      "type": 101,
      "id": "FSNaHMC11W3VskpGYfgxt3fqAMvt6gUmgy61CX8mm93QykuRp2E9Z8BtQc8w22Awc6W8CpXGJn6VcpkcBdAx4Tj",
      "fee": 0,
      "timestamp": 1606211535601,
      "signature": "FSNaHMC11W3VskpGYfgxt3fqAMvt6gUmgy61CX8mm93QykuRp2E9Z8BtQc8w22Awc6W8CpXGJn6VcpkcBdAx4Tj",
```

```
    "target": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c",
    "role": "contract_developer"
  },
  {
    "type": 101,
    "id": "4rfDMTGjbHENy3uiACLmfAHFJWyouhridZHGpynfV8S6aX3XmZHjUSfCvadn3KSzb8eHRq1kmzEaLMxvbqWkUKBY",
    "fee": 0,
    "timestamp": 1606211535600,
    "signature": "4rfDMTGjbHENy3uiACLmfAHFJWyouhridZHGpynfV8S6aX3XmZHjUSfCvadn3KSzb8eHRq1kmzEaLMxvbqWkUKBY",
    "target": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c",
    "role": "issuer"
  },
  {
    "type": 110,
    "id": "4q5iXHv8jZ1qw5FptfBCz1cic14u1M4zCzE1i5qqEA4z6TQmeVFaqhZRpepFpdyGiSyKH4s6XqKPTgxuEJ8Sp4QQ",
    "fee": 0,
    "timestamp": 1606211535610,
    "signature": "4q5iXHv8jZ1qw5FptfBCz1cic14u1M4zCzE1i5qqEA4z6TQmeVFaqhZRpepFpdyGiSyKH4s6XqKPTgxuEJ8Sp4QQ",
    "targetPublicKey": "25GXtqKBAHTCrHuDoXvwQGXnHKBdeVcjdLvSmQ7SVFq4FDcMWzV78oRkgoS32AFDQ23DvfGFX6QpRkQRShQ4zMJy",
    "target": "3N9nNFySk1zVSVf9DUWR9DiBA1jEmmDDpaJ"
  },
  {
    "type": 101,
    "id": "2gjzK3qSp89ywXCjEpvCHKSeyqoBYR2XCKegZ1ngGrQF8cDGXjA19HN8eYTgw8DRoXy62MM138EXXiZyV7oCaZrt",
    "fee": 0,
    "timestamp": 1606211535599,
    "signature": "2gjzK3qSp89ywXCjEpvCHKSeyqoBYR2XCKegZ1ngGrQF8cDGXjA19HN8eYTgw8DRoXy62MM138EXXiZyV7oCaZrt",
    "target": "3N9nNFySk1zVSVf9DUWR9DiBA1jEmmDDpaJ",
    "role": "permissioner"
  },
  {
    "type": 101,
    "id": "3zq1bCbeiNt4Z35rVtKwPo2MnW8peEcx2fQtgMseiJSb3TN7TKfU9auLEWKAgRXoNjpbpi9XA4aJw8Ly4gcpEaTv",
    "fee": 0,
    "timestamp": 1606211535598,
    "signature": "3zq1bCbeiNt4Z35rVtKwPo2MnW8peEcx2fQtgMseiJSb3TN7TKfU9auLEWKAgRXoNjpbpi9XA4aJw8Ly4gcpEaTv",
    "target": "3N9nNFySk1zVSVf9DUWR9DiBA1jEmmDDpaJ",
    "role": "miner"
  },
  {
    "type": 101,
    "id": "Aikgzt9ChSDfK4foF9oQJ8qRjV5cRyqF9okU9gr9JdpXh2LpyVB7GW4XSjmyc4MK9btPh3xd2whFDoCr8J5F4Hs",
    "fee": 0,
    "timestamp": 1606211535597,
    "signature": "Aikgzt9ChSDfK4foF9oQJ8qRjV5cRyqF9okU9gr9JdpXh2LpyVB7GW4XSjmyc4MK9btPh3xd2whFDoCr8J5F4Hs",
    "target": "3N9nNFySk1zVSVf9DUWR9DiBA1jEmmDDpaJ",
    "role": "connection_manager"
  },
  {
    "type": 101,
    "id": "48EGdWC133vQeydqMSXjmXJKB6L2brnu8Sh5W8r4anKCaUQZp5iKGrpVUAwsiUHfHrMXGA52roeoqo7abUHQbbVw",
    "fee": 0,
    "timestamp": 1606211535596,
    "signature": "48EGdWC133vQeydqMSXjmXJKB6L2brnu8Sh5W8r4anKCaUQZp5iKGrpVUAwsiUHfHrMXGA52roeoqo7abUHQbbVw",
    "target": "3N9nNFySk1zVSVf9DUWR9DiBA1jEmmDDpaJ",
    "role": "contract_developer"
  },
  {
    "type": 101,
    "id": "FwNbJyr2Est9DFi5uch1ZfkQjDg13asqSsAdm37381aMWMrdaxcjqXMpKus1rxDcxZd5YnD4MNkz1ZpPgZ8nupn",
    "fee": 0,
    "timestamp": 1606211535595,
    "signature": "FwNbJyr2Est9DFi5uch1ZfkQjDg13asqSsAdm37381aMWMrdaxcjqXMpKus1rxDcxZd5YnD4MNkz1ZpPgZ8nupn",
    "target": "3N9nNFySk1zVSVf9DUWR9DiBA1jEmmDDpaJ",
    "role": "issuer"
  },
  {
    "type": 110,
    "id": "ps5vGHxv4DfTFnTXsqeS22hXQQm8uBf1mwnc7gtDvGxGGfEhDq8DvnCjtKukYmuEW6adz5NQGLbaqbMJK7ChYdA",
    "fee": 0,
    "timestamp": 1606211535610,
```

```
      "signature": "ps5vGHxv4DfTFnTXsqeS22hXQQm8uBf1mwnc7gtDvGxGGfEhDq8DvnCjtKukYmuEW6adz5NQGLbaqbMJK7ChYdA",
      "targetPublicKey": "5fbBNmkW9LJBUFNJW6vsjrmBzGf2AMwdqgHNvne2iYPMNW2wtDJGmF4PGnqyzTYJyYN3kWNWd4cFf9xBZ8Qi9Hki",
      "target": "3N3jgxvmSsBBV4oz9BcKhT8War1em2sKoJn"
    },
    {
      "type": 101,
      "id": "5BG3AhFnGbDcSDJ88KmXViU2tCxs4VNhXGjgocn2ZCcvCjTbxGjso4DKPkcajUNJBhPZHqgMmEKugVxqBMjNf2YY",
      "fee": 0,
      "timestamp": 1606211535594,
      "signature": "5BG3AhFnGbDcSDJ88KmXViU2tCxs4VNhXGjgocn2ZCcvCjTbxGjso4DKPkcajUNJBhPZHqgMmEKugVxqBMjNf2YY",
      "target": "3N3jgxvmSsBBV4oz9BcKhT8War1em2sKoJn",
      "role": "permissioner"
    },
    {
      "type": 101,
      "id": "HYoFXRgsyHGTa9JTnCDpJtBu6hr61LTYTA2zGPkUAVaTn6mhHfSKoVJbn91DN2gtqZxNreQnrV4GGnMR4cFikAE",
      "fee": 0,
      "timestamp": 1606211535593,
      "signature": "HYoFXRgsyHGTa9JTnCDpJtBu6hr61LTYTA2zGPkUAVaTn6mhHfSKoVJbn91DN2gtqZxNreQnrV4GGnMR4cFikAE",
      "target": "3N3jgxvmSsBBV4oz9BcKhT8War1em2sKoJn",
      "role": "contract_developer"
    },
    {
      "type": 101,
      "id": "4snBMYD3dDw9pivJM2YFSJBPPtK4K43YGL8Qjw4APadgZCtqsR4yoo3CZC4bgf5ZffwVWQQzVmfSjxpzsiwCjNju",
      "fee": 0,
      "timestamp": 1606211535592,
      "signature": "4snBMYD3dDw9pivJM2YFSJBPPtK4K43YGL8Qjw4APadgZCtqsR4yoo3CZC4bgf5ZffwVWQQzVmfSjxpzsiwCjNju",
      "target": "3N3jgxvmSsBBV4oz9BcKhT8War1em2sKoJn",
      "role": "issuer"
    }
  ],
  "version": 1,
  "poa-consensus": {
    "overall-skipped-rounds": 0
  },
  "timestamp": 1606211535610,
  "height": 1
}
```


GET /blocks/last

Метод возвращает содержимое текущего блока блокчейна.

Текущий блок находится в процессе создания, пока он не будет принят узлами-майнерами, количество транзакций в нем может меняться.

В ответе метода возвращаются следующие параметры:

- `reference` – подпись предыдущего блока;
- `blocksize` – размер блока;
- `features` – функциональные возможности, запущенные на момент создания блока;
- `signature` – подпись блока;
- `fee` – комиссия за транзакции, включенные в блок;
- `generator` – адрес создателя блока;
- `transactionCount` – количество транзакций 1 и 101, включенных в блок;
- `transactions` – массив с телами транзакций, включенных в блок;
- `version` – версия блока;
- `poa-consensus.overall-skipped-rounds` – количество пропущенных раундов майнинга, при использовании алгоритма консенсуса PoA;
- `timestamp` – временная метка создания блока в формате Unix Timestamp (в миллисекундах);
- `height` – высота создания блока.

Пример ответа метода для пустого текущего блока:


```
{
  "reference": "hT5RcPT4jDVoNspfZkNhKqfGuMbrizjpG4vmPecVfWgWaGMoAn5hgPBJpC9696TL8wGDKJzkwewiqe8m26C4aPd",
  "blocksize": 226,
  "features": [],
  "signature": "5GAM7jfQScw4g3g7PCNNtz5xG3JzjJnW4Ap2soThirSx1AmUQHQMjz8VMtkFEzK7L447ouKHfj2gMvZyP5u94Rps",
  "fee": 0,
  "generator": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c",
  "transactionCount": 0,
  "transactions": [],
  "version": 3,
  "poa-consensus": {
    "overall-skipped-rounds": 1065423
  },
  "timestamp": 1615816767694,
  "height": 1826
}
```


GET /blocks/at/{height}

Метод возвращает содержимое блока на высоте height.
В ответе метода возвращаются следующие параметры:

- `reference` – подпись предыдущего блока;
- `blocksize` – размер блока;
- `features` – функциональные возможности, запущенные на момент создания блока;
- `signature` – подпись блока;
- `fee` – комиссия за транзакции, включенные в блок;
- `generator` – адрес создателя блока;
- `transactionCount` – количество транзакций, включенных в блок;
- `transactions` – массив с телами транзакций, включенных в блок;
- `version` – версия блока
- `poa-consensus.overall-skipped-rounds` – количество пропущенных раундов майнинга, при использовании алгоритма консенсуса PoA;
- `timestamp` – временная метка создания блока в формате Unix Timestamp (в миллисекундах);
- `height` – высота создания блока.

Пример ответа метода для пустого текущего блока:


```
{
  "reference": "hT5RcPT4jDVoNspfZkNhKqfGuMbrizjpG4vmPecVfWgWaGMoAn5hgPBJpC9696TL8wGDKJzkwewiqe8m26C4aPd",
  "blocksize": 226,
  "features": [],
  "signature": "5GAM7jfQScw4g3g7PCNNtz5xG3JzjJnW4Ap2soThirSx1AmUQHQMjz8VMtkFEzK7L447ouKHfj2gMvZyP5u94Rps",
  "fee": 0,
  "generator": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c",
  "transactionCount": 0,
  "transactions": [],
  "version": 3,
  "poa-consensus": {
    "overall-skipped-rounds": 1065423
  },
  "timestamp": 1615816767694,
  "height": 1826 }
```


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

Метод возвращает содержимое блоков от высоты `{from}` до высоты `{to}`.
Для каждого блока возвращаются параметры, идентичные методу `GET /blocks/at/{height}`.

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

Метод возвращает содержимое блоков с расширенной информацией о транзакциях от высоты `{from}` до высоты `{to}`.

В остальном, для каждого блока возвращаются параметры, идентичные методу `GET /blocks/at/{height}`.

GET /blocks/signature/{signature}

Метод возвращает содержимое блока по его подписи `{signature}`.
В ответе метода возвращаются параметры, идентичные методу `GET /blocks/at/{height}`.

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

Метод возвращает содержимое всех блоков, сформированных адресатом `{address}` от высоты `{from}` до высоты `{to}`.

В ответе метода для каждого блока возвращаются параметры, идентичные методу `GET /blocks/at/{height}`.

GET /blocks/child/{signature}

Метод возвращает унаследованный блок от блока с подписью `{signature}`.
В ответе метода возвращаются параметры, идентичные методу `GET /blocks/at/{height}`.

GET /blocks/headers/at/{height}

Метод возвращает заголовок блока на высоте `{height}`. В ответе метода возвращаются следующие параметры:

- `reference` – подпись предыдущего блока;
- `blocksize` – размер блока;
- `features` – функциональные возможности, запущенные на момент создания блока; signature – подпись блока;
- `fee` – комиссия за транзакции, включенные в блок;
- `generator` – адрес создателя блока;
- `pos-consensus.base-target` – коэффициент, регулирующий время выпуска блока, при использовании алгоритма консенсуса PoS;
- `pos-consensus.generation-signature` – подпись, необходимая для валидации майнера блока;
- `poa-consensus.overall-skipped-rounds` – количество пропущенных раундов майнинга, при использовании алгоритма консенсуса PoA;
- `version` – версия блока;
- `timestamp` – временная метка создания блока в формате Unix Timestamp (в миллисекундах);
- `height` – высота создания блока.

Пример ответа метода:


```
{
  "reference": "5qWJh9aQ2hkwnBWygGYmrBhzMe5inRZ2r6WhEXz3VJsiMtASWkvbsVeZGychZKzcPDbWmpzdhQwNQJ19PfK2dst9",
  "blocksize": 589,
  "features": [
    0
  ],
  "signature": "4U4Hmg4mDYrvxaZ3JVzL1Z1piPDZ1PJ61vd1PeS7ESZFkHsUCUqeeAZoszTVr43Z4NV44dqbLv9WdrLytDL6gHuv",
  "fee": 5000000,
  "generator": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "pos-consensus": {
    "base-target": 249912231,
    "generation-signature": "LM83w6eWQHnLJF2D9RQNdNcHAdnZLCLWrn5bfcoqcZy"
  },
  "poa-consensus": {
    "overall-skipped-rounds": 2
  },
  "transactionCount": 2,
  "version": 12,
  "timestamp": 1568287320962,
  "height": 48260
}
```


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

Метод возвращает заголовки блоков с высоты `{from}` до высоты `{to}`.

В ответе метода для каждого блока возвращаются параметры, идентичные методу `GET /blocks/headers/at/{height}`.

GET /blocks/headers/last

Метод возвращает заголовок текущего блока.

В ответе метода для каждого блока возвращаются параметры, идентичные методу `GET /blocks/headers/at/{height}`.

3.2.15 REST API: информация о ролях участников

Для получения информации о ролях участников в сети предназначены методы группы `permissions`. Методы требуют авторизации. Подробнее об управлении ролями участников сети см. раздел «Управление ролями участников» ниже. Подробнее о ролях см. раздел «6.1 Описание ролей участников сети» в документе «643.19172778.2019662198-01 95. «Блокчейн-платформа Конфидент» Версия 1.9. Правила пользования»; этот документ поставляется вместе с дистрибутивом платформы.

GET /permissions/{address}

Метод возвращает информацию об активных ролях участника `{address}`, а также время формирования запроса в формате Unix Timestamp (в миллисекундах).

Пример ответа метода:

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

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

Метод возвращает информацию о ролях участника `{address}`, активных на момент времени `{timestamp}`. Время указывается в формате Unix Timestamp (в миллисекундах).

Пример ответа метода:

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

POST /permissions/addresses

Метод возвращает роли для нескольких адресов, активные на указанный момент времени.

В запросе передаются следующие данные:

- `addresses` – список адресов в виде массива строк;
- `timestamp` – время в формате Unix Timestamp (в миллисекундах).

Пример запроса с двумя адресами:

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

В ответе метода возвращается массив данных `addressToRoles`, в котором указаны роли для каждого адреса, а также время `timestamp`.

Пример ответа метода для двух адресов:


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


3.2.16 REST API: информация об ассетах и балансах адресов

Для получения информации об ассетах и балансах адресов предусмотрены методы группы `assets`. Методы требуют авторизации.

GET /assets/balance/{address}

Метод возвращает баланс всех ассетов адреса. В ответе возвращаются следующие параметры:

- `address` – адрес участника;
- `balances` – объект с балансами участника:
  - `assetId` – ID ассета;
  - `balance` – баланс ассета;
  - `quantity` – общее количество выпущенных токенов ассета;
  - `reissuable` – перевыпускаемость ассета;
  - `issueTransaction` – тело транзакции 3 создания ассета;
  - `minSponsoredAssetFee` – минимальное значение комиссии для спонсорских транзакций;
  - `sponsorBalance` – средства, выделенные для оплаты транзакций по спонсируемым ассетам.

Пример ответа метода:


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

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

POST /assets/balance

Метод возвращает баланс всех ассетов для нескольких адресов, переданных в поле `addresses`. Параметры, передаваемые в ответе для каждого адреса, идентичны методу `GET /assets/balance/{address}`.

Пример ответа метода для одного адреса:

```
{
  "address": "3Mv61qe6egMSjRDZiiuvJDnf3Q1qW9tTZDB",
  "balances": [
    {
      "assetId": "Ax9T4grFxx5m3KPUEKjMdnQkCKtBktf694wU2wJYvQUD",
      "balance": 4879179221,
      "quantity": 48791792210,
      "reissuable": true,
      "minSponsoredAssetFee" : 100,
      "sponsorBalance" : 1233221,
      "issueTransaction" : {
         "type" : 3,
         ...
      }
    },
    {
      "assetId": "49KfHPJcKvSAvNKwM7CTofjKHzL87SaSx8eyADBjv5Wi",
      "balance": 10,
      "quantity": 10000000000,
      "reissuable": false,
      "issueTransaction" : {
         "type" : 3,
         ...
      }
    }
  ]
}
```

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

Метод возвращает баланс адреса в указанном `{assetId}`.

Пример ответа метода:

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

GET /assets/details/{assetId}

Метод возвращает описание ассета `{assetId}`.
Пример ответа 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
}
```

3.2.17 REST API: работа с узлами блокчейна

Для работы с узлами блокчейна предусмотрена группа методов `peers`. Методы требуют авторизации.

POST /peers/connect

Метод предназначен для подключения нового узла участника к вашему узлу.
Примеры запроса и ответа метода:
Запрос:

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

Ответ:

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

GET /peers/connected

Метод возвращает список подключенных узлов.
Пример ответа 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

Метод возвращает список всех известных узлов.
Пример ответа метода:

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

GET /peers/suspended

Метод возвращает список приостановленных узлов.
Пример ответа метода:

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

POST /peers/identity

Метод возвращает публичный ключ узла, к которому подключается ваш узел для передачи конфиденциальных данных.

В запросе передаются следующие параметры:

- `address` – блокчейн-адрес, который соответствует параметру `privacy.owner-address` в конфигурационном файле узла;
- `signature` – электронная подпись от значения поля `address`.

Ответ метода содержит параметр `publicKey` – публичный ключ узла, связанный с параметром `privacy.owner-address` в его конфигурационном файле. Если выключен режим проверки *handshakes*, то параметр `publicKey` не отображается.

Примеры запроса и ответа метода:

Запрос:

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

Ответ:

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

GET /peers/hostname/{address}

Метод получает блокчейн адрес (owner-address) и, если среди пиров такой адрес есть, то возвращает соответствующее ему имя хоста (hostname) и IP-адрес узла.

Важно: Метод GET /peers/hostname/{address} использует аутентификацию по списку tls-whitelist и доступен только администратору сети. Настройки авторизации описаны в разделе «Тонкая настройка платформы: настройка авторизации для gRPC и REST API» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

Пример ответа метода:

```
{
  "hostname": "node1.web3tech.io",
  "ip": "10.0.0.1"
}
```

GET /peers/allowedNodes

Получение актуального списка разрешенных участников сети на момент запроса.

Важно: Метод GET /peers/allowedNodes использует аутентификацию по списку tls-whitelist и доступен только администратору сети. Настройки авторизации описаны в разделе «Тонкая настройка платформы: настройка авторизации для gRPC и REST API» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

Пример ответа метода:


```
{
  "allowedNodes": [
    {
      "address": "3JNLQYuHYSHZiHr5KjJ89wwFJpDMdrAEJpj",
      "publicKey": "Gt3o1ghh2M2TS65UrHZCTJ82LLcMcBrxuaJyrgsLk5VY"
    },
    {
      "address": "3JLp8wt7rEUdn4Cca5Hp9jZ7w8T5XDAKicd",
      "publicKey": "J3ffCciVu3sustgb5vxmEHczACMR89Vty5ZBLbPn9xyg"
    },
    {
      "address": "3JRY1cp7atRMBd8QQoswRpH7DLawM5Pnk3L",
      "publicKey": "5vn4UcB9En1XgY6w2N6e9W7bqFshG4SL2RLFqEWEbWxG"
    }
  ],
  "timestamp": 1558697649489
}
```


3.2.18 REST API: хэширование, работа со скриптами и отправка вспомогательных запросов

Для хэширования, работы со скриптами и отправки вспомогательных запросов к узлу предусмотрена группа методов `utils`. Эти методы требуют авторизации.

3.2.18.1 Работа со скриптами: `utils/script`

Данная группа методов предназначена для конвертации кода скриптов в формат base64 и их декодирования. Скрипты привязываются к аккаунтам при помощи транзакций 13 (привязка скрипта к адресу) и 15 (привязка скрипта к ассету для адреса).

POST /utils/script/compile

Метод конвертирует код скрипта в формат base64.
В ответе метода возвращаются следующие параметры:

- `script` – тело скрипта в формате base64;
- `complexity` – сложность скрипта: число от 1 до 100, отражающее количество вычислительных ресурсов, требуемое для его исполнения;
- `extraFee` – комиссия за исходящие транзакции, установленные скриптом.

Примеры запроса и ответа метода:
Запрос:

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

Ответ:


```
{
  "script": "base64:AQQAAAABeAAAAAAAAAAAAQkAAAAAAAACCQAAZAAAAAIFAAAAAXgAAAAAAAAAAAEAAAAAAAAAAAKH6D8P",
  "complexity": 12
}
```

POST /utils/script/estimate

Метод предназначен для декодирования и оценки сложности скрипта, переданного в запросе в формате base64.

В ответе метода возвращаются следующие параметры:

- `script` – тело скрипта в формате base64;
- `scriptText` – код скрипта;
- `complexity` – сложность скрипта: число от 1 до 100, отражающее количество вычислительных ресурсов, требуемое для его исполнения;
- `extraFee` – комиссия за исходящие транзакции, установленные скриптом.

Пример ответа метода:


```
{
  "script": "AQQAAAABeAAAAAAAAAAAAQkAAAAAAAACCQAAZAAAAAIFAAAAAXgAAAAAAAAAAAEAAAAAAAAAAAKH6D8P",
  "scriptText": "BLOCK(LET(x,CONST_LONG(1)),FUNCTION_CALL(Native(0),List(FUNCTION_CALL(Native(100),List(REF(x),
   CONST_LONG(1))), CONST_LONG(2))))",
  "complexity": 12
}
```


3.2.18.2 Вспомогательные запросы

GET /utils/time

Метод возвращает текущее время узла в двух форматах:

- `system` – системное время на машине узла;
- `ntp` – сетевое время.

Пример ответа метода:


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


POST /utils/reload-wallet

Метод перезагружает keystore узла. Применяется в случае, если новая ключевая пара была добавлена в keystore без перезапуска узла.

Важно: Метод POST /utils/reload-wallet доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы. Метод POST /utils/reload-wallet использует аутентификацию по списку tls-whitelist и доступен только администратору сети. Настройки авторизации описаны в разделе «Тонкая настройка платформы: настройка авторизации для gRPC и REST API» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора».

Пример ответа метода:


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

3.2.18.3 Хэширование: utils/hash

POST /utils/hash/fast

Метод возвращает хэш-сумму строки, переданной в запросе.

Важно: Метод POST /utils/hash/fast доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру node.crypto.pki.mode присвоено значение TEST. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

Пример ответа метода:

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

POST /utils/hash/secure

Метод возвращает двойную хэш-сумму строки, переданной в запросе.

Важно: Метод POST /utils/hash/secure доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру node.crypto.pki.mode присвоено значение TEST. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

Пример ответа метода:

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

3.2.19 REST API: Отладка блокчейна

Для отладки блокчейн-сети предусмотрены методы группы debug. Эти методы требуют авторизации.

Важно: Методы группы debug доступны только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру node.crypto.pki.mode присвоено значение TEST. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

GET /debug/blocks/{howMany}

Метод отображает размер и полный хэш последних блоков. Количество блоков указывается при запросе.
Пример ответа метода:

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

```
    },
    {
        "226": "Byb9MHtwYf3MFyi2tbhQ3GTdCct5phKq9REkbjQTzdne"
    },
    {
        "226": "HSxSHbiV4tZc8RaN6jxdhgtkAhjxuLn76uHxerMRUefA"
    }
]
```


GET /debug/info

Метод отображает общую информацию о блокчейне, необходимую для отладки и тестирования. Пример ответа 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

Метод откатывает блокчейн до заданной высоты, удаляя все блоки после нее. В запросе передаются следующие параметры:

- `rollbackTo` – высота, до которой необходимо откатить блокчейн;
- `returnTransactionsToUtx` – возвращение транзакций, которые содержатся в откатываемых блоках, в UTX-пул:
  - `true` – вернуть,
  - `false` – удалить.

Примеры запроса и ответа метода:

Запрос:


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


Ответ:


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

POST /debug/validate

Метод валидирует транзакции по их идентификатору и измеряет затраченное время в миллисекундах. В запросе передается `id` транзакции.

Пример ответа метода:

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

GET /debug/minerInfo

Метод отображает информацию о майнере.
Пример ответа метода:

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

GET /debug/historyInfo

Метод отображает историю последнего блока.
Пример ответа метода:

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

GET /debug/configInfo

Метод полностью выводит используемый конфигурационный файл узла.
Пример ответа метода:

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

Метод откатывает блокчейн до блока с указанной подписью `{signature}`.
Пример ответа метода:

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

GET /debug/portfolios/{address}

Метод отображает текущий баланс по транзакциям, находящимся в UTX-пуле узла `{address}`.
Пример ответа метода:

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

POST /debug/print

Метод выводит текущие сообщения логгера, имеющего уровень логирования DEBUG.
Ответ выводится в формате `"message" : "string"`.

GET /debug/state

Метод отображает текущий стейт узла.
Пример ответа 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}

Метод отображает стейт узла на указанной высоте `{height}`.
Пример ответа метода:

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

4 Разработка и применение смарт-контрактов

Смарт-контракт – это отдельное приложение, которое записывает в блокчейн свои входные данные и результаты исполнения заложенного алгоритма. Платформа «Конфидент» поддерживает разработку и применение Тьюринг-полных смарт-контрактов для создания высокоуровневых бизнес-приложений.

Смарт-контракт может быть разработан на любом языке программирования и не имеет ограничений на реализацию заложенной логики. Для того, чтобы отделить запуск и исполнение смарт-контракта от самой блокчейн-платформы, смарт-контракт исполняется в контейнере Docker.

Когда смарт-контракт запускается в блокчейн сети, его код нельзя произвольно изменить, заменить или запретить его выполнение без вмешательства в работу всей сети. Это свойство позволяет обеспечить безопасность работы бизнес-приложений.

4.1.1 Подготовка к работе

Перед началом разработки смарт-контракта убедитесь, что на вашей машине установлен пакет ПО для контейнеризации приложений Docker, доступный для скачивания по адресу https://www.docker.com/get-started/. Принципы работы с Docker изложены в официальной документации Docker, доступной по адресу https://docs.docker.com.

Также убедитесь, что на используемом вами узле настроено исполнение смарт-контрактов. Подробнее о настройке смарт-контрактов см. в разделе «Общая настройка платформы: настройка исполнения смарт-контрактов» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

Если вы разрабатываете смарт-контракт для работы в частной сети, разверните собственный репозиторий для Docker-образов (доступен для скачивания по адресу https://docs.docker.com/registry/) и укажите его адрес и учетные данные на вашем сервере в блоке `remote-registries` конфигурационного файла узла. В этом блоке вы можете указать несколько репозиториев, если вам необходимо определить несколько мест хранения различных смарт-контрактов. Также вы можете загрузить Docker-образ контракта из репозитория, не указанного в конфигурационном файле узла, при помощи транзакции 103, инициирующей создание смарт-контракта. Подробнее см. описание транзакции 103 в разделе «Транзакции блокчейн-платформы Конфидент» – «Описание транзакций» – «103. CreateContract Transaction» ниже.

4.1.2 Разработка смарт-контракта

Смарт-контракты Платформы «Конфидент» могут разрабатываться на любом языке программирования и реализовывать любые алгоритмы. Готовый код смарт-контракта упаковывается в Docker-образ с protobuf-файлами, в которые упакованы используемые gRPC методы.

Пример кода смарт-контракта на Python с применением gRPC API-методов для обмена данными с узлом, а также пошаговое руководство по созданию соответствующих Docker-образов приведены ниже в разделе «Пример смарт-контракта с использованием gRPC».

4.1.2.1 API-инструменты, доступные смарт-контракту

Для обмена данными между смарт-контрактом и узлом предназначены контрактные сервисы gRPC API. Они описаны ниже. Методы этих сервисов позволяют осуществлять широкий спектр операций с блокчейном.

Важно: Перечисленные ниже контрактные методы gRPC интерфейса Платформы «Конфидент» доступны только смарт-контрактам. Внешний пользователь не сможет вызвать контрактные сервисы и использовать их методы.

Версии API смарт-контрактов

gRPC-методы, используемые смарт-контрактами, формируют API, заданное protobuf-файлами. Для четкого определения новых методов и внесения изменений в существующие предусмотрено версионирование API. Благодаря присвоенному номеру версии, узел при исполнении смарт-контракта определяет соответствующий набор методов для использования.

Актуальная версия gRPC API для версии блокчейн-платформы содержится в файле `api_version.proto`. Смарт-контракты, которые требуют версию API выше, чем у майнящего узла, игнорируются при майнинге.

Для создания и обновления смарт-контрактов предусмотрены поля `apiVersion` в транзакциях 103 CreateContract Transaction и 107 UpdateContract Transaction версии 4 (подробное описание транзакций см. в

разделах «103. CreateContract Transaction» и «107. UpdateContract Transaction» ниже). Эти поля указывают майнящему узлу на версию API, используемую смарт-контрактом.

В таблице ниже приведены версии API, соответствующие версиям блокчейн-платформы:

Версия API	Версия платформы
1.0	1.6.0 и старше
1.1	1.6.2
1.4	1.7.0

4.1.2.1.1 Получение информации об адресах участников сети

Группа контрактных методов сервиса **AddressService**, которые позволяют получить информацию об адресах участников сети, упакована в protobuf-файл `contract_address_service.proto`.

GetAddresses

Метод возвращает все адреса участников, ключевые пары которых хранятся в keystore узла. Ответ метода представляет собой массив строк `addresses`.

GetAddressData

Метод возвращает все данные, записанные на аккаунт адресата при помощи транзакций 12 (подробное описание транзакции см. в разделе «12 Data Transaction» ниже). В запросе метода вводятся следующие параметры:

- `address` – адрес, данные которого необходимо вывести;
- `limit` – ограничение количества выводимых блоков данных;
- `offset` – количество блоков данных для пропуска в выводе.

Ответ метода представляет собой массив DataEntry, содержащий записанные данные адреса.

GetAssetBalance

Метод возвращает текущий баланса определенного ассета для определенного пользователя. В запросе метода вводятся следующие параметры:

- `address` – адрес, баланс которого необходимо вывести;
- `assetId` – идентификатор ассета. Для WEST параметр остается пустым.

4.1.2.1.2 Исполнение смарт-контракта и чтение информации о состоянии смарт-контрактов

Группа контрактных методов сервиса **ContractService** упакована в protobuf-файл `contract_contract_service.proto`.

Группа содержит служебные методы для исполнения контракта, а также методы для чтения информации о состоянии смарт-контрактов.

Connect

Метод подключает смарт-контракт к узлу. В запросе метода указывается идентификатор соединения смарт-контракта `connection_id`.

CommitExecutionSuccess

Метод возвращает результат успешного исполнения смарт-контракта по его идентификатору и отправляет результат успешного исполнения смарт-контракта на узел.

CommitExecutionError

Метод возвращает ошибку, возникшую при исполнении смарт-контракта, по его идентификатору и отправляет ошибку исполнения смарт-контракта на узел.

GetContractKeys

Метод запрашивает значения из состояния смарт-контракта по переданному фильтру ключей. В запросе метода указываются следующие данные:

- `contract_id` – идентификатор смарт-контракта;
- `limit` – ограничение количества выводимых блоков данных;
- `offset` – количество блоков данных для пропуска в выводе;
- `matches` – опциональный параметр для регулярного выражения, по которому фильтруются ключи.
- `keys_filter` – фильтр ключей.

В ответе метод возвращает массив `Entries`, содержащий результаты исполнения смарт-контракта.

GetContractKey

Метод возвращает значение определённого ключа из состояния смарт-контракта. В запросе метода указывается идентификатор смарт-контракта `contract_id` и запрашиваемый ключ `key`.

В ответе метод возвращает `DataEntry` из текущего состояния смарт-контракта, который соответствует переданному ключу.

4.1.2.1.3 Получение информации о ролях участников

Группа контрактных методов сервиса **PermissionService** для получения информации о ролях участников упакована в protobuf-файл `contract_permission_service.proto`.

GetPermissions

Метод возвращает список ролей участника `address`, активных на момент времени `timestamp`, а также время формирования запроса.

В запросе передаются следующие данные:

- `address` – запрашиваемый адрес;
- `timestamp` – временная метка в формате Unix Timestamp (в миллисекундах), на момент которой запрашиваются действующие роли.

В ответе метод выводит массив `roles`, содержащий роли запрашиваемого адреса, и указанную временную метку `timestamp`.

GetPermissionsForAddresses

Метод возвращает список ролей, активных на указанный момент времени, для нескольких указанных адресов.

В запросе передаются следующие данные:

- `addresses` – массив строк с запрашиваемыми адресами;
- `timestamp` – временная метка в формате Unix Timestamp (в миллисекундах), на момент которой запрашиваются действующие роли.

В ответе метода выводится массив `address_to_roles`, содержащий роли для каждого запрашиваемого адреса, и указанная временная метка timestamp.

4.1.2.1.4 Проверка электронных подписей данных

Контрактный метод сервиса **PkiService** для проверки ЭП для данных упакован в protobuf-файл `contract_pki_service.proto`.

Важно: Контрактный gRPC API метод Verify доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

Verify

Метод выполняет проверку электронной подписи. Метод требует ввода следующих параметров:

- `input_data` – данные, закрытые ЭП, в виде массива байт в кодировке base64;
- `signature` – электронная подпись в виде массива байт в кодировке base64;

- sig_type – формат ЭП. Поддерживаются значения:
  - 1 – CAdES-BES;
  - 2 – CAdES-X Long Type 1;
  - 3 – CAdES-T.
- extended_key_usage_list – список объектных идентификаторов (OID) криптографических алгоритмов, которые используются при формировании ЭП (опциональное поле).

Ответ метода содержит поле status с булевым типом данных:

- true – подпись действительна,
- false – подпись скомпрометирована.

4.1.2.1.5 Получение информации о группах обмена конфиденциальными данными

Контрактные методы сервиса **PrivacyService** для получения информации о группах для обмена конфиденциальными данными и работы с конфиденциальными данными упакованы в protobuf-файл contract_privacy_service.proto.

Важно: Контрактные gRPC API методы, упакованные в protobuf-файл contract_privacy_service.proto, доступны только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру node.crypto.pki.mode присвоено значение TEST. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

GetPolicyRecipients

Метод возвращает адреса участников группы доступа к конфиденциальным данным, идентификатор которой передается в запросе как policy_id. В ответе метод выводит массив строк recipients, содержащий адреса участников группы доступа.

GetPolicyOwners

Метод возвращает адреса владельцев группы для обмена конфиденциальными данными, идентификатор которой передается в запросе как policy_id. В ответе метод выводит массив строк owners, содержащий адреса владельцев группы доступа.

GetPolicyHashes

Метод возвращает идентификационные хэши пакетов конфиденциальных данных по идентификатору группы доступа к конфиденциальным данным (policy_id).

GetPolicyItemData

Метод возвращает пакет конфиденциальных данных по его идентификационному хэшу. Метод доступен, если вызывающий адрес принадлежит к группе доступа к конфиденциальным данным.

В запросе метода указываются идентификатор группы доступа policy_id и идентификационный хэш конфиденциальных данных item_hash. В ответе метод возвращает строку data, содержащую хэш пакета конфиденциальных данных.

GetPolicyItemInfo

Метод возвращает метаданные пакета конфиденциальных данных по его идентификационному хэшу. Метод доступен, если вызывающий адрес принадлежит к группе доступа к конфиденциальным данным.

В запросе метода указываются идентификатор группы доступа policy_id и идентификационный хэш конфиденциальных данных item_hash. В ответе метод возвращает следующие данные:

- sender – адрес отправителя конфиденциальных данных;
- policy_id – идентификатор группы доступа;
- type – тип конфиденциальных данных (file);
- info – массив данных о файле:
  - filename – имя файла;
  - size – размер файла;
  - timestamp – временная метка размещения файла в формате Unix Timestamp (в миллисекундах);
  - author – автор файла;

- `comment` – опциональный комментарий к файлу;
- `hash` – идентификационный хэш конфиденциальных данных.

4.1.2.1.6 Работа с транзакциями

Контрактные методы сервиса **TransactionService** для работы с транзакциями упакованы в protobuf-файл `contract_transaction_service.proto`.

TransactionExists

Метод проводит проверку существования транзакции с указанным идентификатором.

Метод возвращает значение `true`, если транзакция с указанным идентификатором существует, `false` – если не существует.

В отличии от метода `TransactionExists`, доступного для интеграции извне (подробнее см. раздел «gRPC: получение информации о транзакции по ее ID» выше), контрактный метод возвращает информацию не только о транзакциях, которые уже записаны в блок, но и о транзакциях, которые только готовятся к упаковке в блок.

TransactionInfo

Метод возвращает данные о транзакции с указанным идентификатором: название транзакции, версия транзакции, высота блокчейна, на которой была произведена данная транзакция, другие данные о транзакции в зависимости от типа этой транзакции.

В отличии от метода `TransactionInfo`, доступного для интеграции извне (подробнее см. раздел «gRPC: получение информации о транзакции по ее ID» выше), контрактный метод возвращает информацию не только о транзакциях, которые уже записаны в блок, но и о транзакциях, которые только готовятся к упаковке в блок.

4.1.2.1.7 Получение вспомогательной информации

Контрактные методы сервиса **UtilService** для получения вспомогательной информации об узле упакованы в protobuf-файл `contract_transaction_service.proto`.

GetNodeTime

Метод возвращает текущее время узла в двух форматах:

- `system` – системное время на машине узла;
- `ntp` – сетевое время.

Метод не требует ввода дополнительных параметров запроса.

4.1.2.2 Пример смарт-контракта с использованием gRPC

В этом разделе рассмотрен пример создания простого смарт-контракта на Python. Для обмена данными с узлом смарт-контракт применяет gRPC-интерфейс.

Перед началом работы убедитесь, что на вашей машине установлены утилиты из пакета grpcio для Python:

```
pip3 install grpcio
```

Порядок установки и использования gRPC-утилит для других доступных языков программирования приведен на официальном сайте gRPC по адресу https://grpc.io/docs/languages/.

Описание и листинг программы

При инициализации смарт контракта при помощи транзакции 103 (описанной ниже в разделе «Транзакции блокчейн-платформы Конфидент» – «Описание транзакций» – «103. CreateContract Transaction») для него устанавливается целочисленный параметр `sum` со значением `0`.

При каждом вызове смарт-контракта при помощи транзакции 104 (описанной ниже в разделе «Транзакции блокчейн-платформы Конфидент» – «Описание транзакций» – «104. CallContract Transaction») он возвращает инкремент параметра `sum` (`sum + 1`).

Листинг программы:

```
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):
        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,
            results=[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:
        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)
```

Если вы хотите, чтобы транзакции с вызовом вашего контракта могли обрабатываться одновременно, то необходимо в самом коде контракта передать параметр `async-factor`. Контракт передаёт значение параметра `async-factor` в составе gRPC-сообщения `ConnectionRequest`, определенном в файле `contract_contract_service.proto`:

```
message ConnectionRequest {
string connection_id = 1;
int32 async_factor = 2;
}
```

Авторизация смарт-контракта с gRPC

Для работы с gRPC смарт-контракту необходима авторизация. Подробнее о gRPC-интерфейсе см. раздел «Инструментарий gRPC» выше. Чтобы смарт-контракт корректно работал с методами API, выполняются следующие действия:

1. В переменных окружения смарт-контракта должны быть определены следующие параметры:
   - `CONNECTION_ID` – идентификатор соединения, передаваемый контрактом при соединении с узлом;
   - `CONNECTION_TOKEN` – токен авторизации, передаваемый контрактом при соединении с узлом;
   - `NODE` – ip-адрес или доменное имя узла;
   - `NODE_PORT` – порт gRPC сервиса, развёрнутого на узле.

Значения переменных `NODE` и `NODE_PORT` берутся из секции dockerengine.grpc-server конфигурационного файла узла, которая описана в разделе «Общая настройка платформы: настройка исполнения смарт-контрактов» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы. Остальные переменные генерируются узлом и передаются в контейнер при создании смарт контракта.

Создание смарт-контракта

1. В директории, которая будет содержать файлы вашего смарт-контракта, создайте поддиректорию `src` и поместите в нее файл **contract.py** с кодом смарт-контракта.

2. В директории `src` создайте директорию `protobuf` и поместите в нее следующие protobuf-файлы:
   - `contract_contract_service.proto`

- `data_entry.proto`

Эти файлы помещены в архив `confident-proto-1.5-1.9.0.zip`, которые поставляются вместе с jar-файлами.

3. Сгенерируйте код gRPC-методов на Python на основе файла `contract_contract_service.proto`:

```
python3 -m grpc.tools.protoc -I. --python_out=. --grpc_python_out=. contract_contract_service.proto
```

В результате будет создано два файла:

- `contract_contract_service_pb2.py`
- `contract_contract_service_pb2_grpc.py`

В файле `contract_contract_service_pb2.py` измените строку `import data_entry_pb2 as data__entry__pb2` следующим образом:

```
import protobuf.data_entry_pb2 as data__entry__pb2
```

Таким же образом измените строку `import contract_contract_service_pb2 as contract__contract__service__pb2` в файле `contract_contract_service_pb2_grpc.py`:

```
import protobuf.contract_contract_service_pb2 as contract__contract__service__pb2
```

Затем сгенерируйте вспомогательный файл `data_entry_pb2.py` на основе `data_entry.proto`:

```
python3 -m grpc.tools.protoc -I. --python_out=. data_entry.proto
```

Все три полученных файла должны находиться в директории **protobuf** вместе с исходными файлами.

4. Создайте shell-скрипт **run.sh**, который будет запускать код смарт-контракта в контейнере:

```
#!/bin/sh

eval $SET_ENV_CMD
python contract.py
```

Поместите файл **run.sh** в корневую директорию вашего смарт-контракта.

5. Создайте сценарный файл **Dockerfile** для сборки и управления запуском смарт-контракта. При разработке на Python основой образа вашего смарт-контракта может служить официальный образ Python `python:3.8-slim-buster`. Обратите внимание, что для обеспечения работы смарт-контракта в контейнере Docker должны быть установлены пакеты `dnsutils` и `grpcio-tools`.

Пример Dockerfile:

```
FROM python:3.8-slim-buster
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"]
```

Поместите **Dockerfile** в корневую директорию вашего смарт-контракта.

6. Если вы работаете в частной сети, соберите смарт-контракт самостоятельно и разместите его в собственном репозитории, как описано ниже в разделе «Загрузка смарт-контракта в репозиторий».

Примечание: При добавлении смарт-контракта необходимо проводить оценку влияния, подробнее см. п.1.5 документа «643.19172778.2019662198-01 30. «Блокчейн-платформа Конфидент» Версия 1.9. Формуляр»; этот документ поставляется вместе с дистрибутивом платформы.

Как работает смарт-контракт с использованием gRPC

После вызова смарт-контракт с gRPC работает следующим образом:

1. После старта программы выполняется проверка на наличие переменных окружения.

2. Используя значения переменных окружения `NODE` и `NODE_PORT`, контракт создает gRPC-подключение с узлом по каналу связи по протоколу TLS с использованием алгоритмов ГОСТ.
3. Далее вызывается потоковый метод `Connect` gRPC-сервиса `ContractService`. Метод принимает gRPC-сообщение `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 103 (описанную ниже в разделе «Транзакции блокчейн-платформы Конфидент» – «Описание транзакций» – «103. CreateContract Transaction»), то для контракта инициализируется начальное состояние. Если `transaction` содержит транзакцию вызова (тип транзакции – 104, см. раздел «Транзакции блокчейн-платформы Конфидент» – «Описание транзакций» – «104. CallContract Transaction» ниже), то выполняются следующие действия:

- с узла запрашивается значение ключа `sum` (метод `GetContractKey` сервиса `ContractService`);
- значение ключа увеличивается на единицу, т. е. `sum = sum + 1;`
- новое значение ключа сохраняется на узле (метод `CommitExecutionSuccess` сервиса `ContractService`), т. е. происходит обновление состояния контракта.

4.1.3 Загрузка смарт-контракта в репозиторий

При работе в частной сети, загрузите Docker-образ смарт-контракта в собственный репозиторий. Для этого выполните следующие шаги:

1. Запустите ваш репозиторий в контейнере:

```
docker run -d -p 5000:5000 --name my-registry-container my-registry:2
```

2. Перейдите в директорию, содержащую файлы смарт-контракта и сценарный файл Dockerfile с командами для сборки образа.
3. Соберите образ вашего смарт-контракта:

```
docker build -t my-contract .
```

4. Укажите имя образа и адрес его размещения в репозитории:

```
docker image tag my-contract my-registry:5000/my-contract
```

5. Запустите созданный вами контейнер репозитория:

```
docker start my-registry-container
```

6. Загрузите ваш смарт-контракт в репозиторий:

```
docker push my-registry:5000/my-contract
```

7. Получите информацию о смарт-контракте. Для этого выведите информацию о контейнере:

```
docker image ls|grep 'my-node:5000/my-contract'
```

Таким образом вы получите идентификатор контейнера. Выведите информацию о нем при помощи команды `docker inspect`:

```
docker inspect my-contract-id
```

Пример ответа:

```
{
"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,
```


Поле `Id` – это идентификатор Docker-образа смарт-контракта, который вводится в поле `ImageHash` транзакции 103 при создании смарт-контракта. Транзакция 103 описана ниже в разделе «Транзакции блокчейн-платформы Конфидент» – «Описание транзакций» – «103. CreateContract Transaction».

4.1.4 Размещение смарт-контракта в блокчейне

После загрузки смарт-контракта в репозиторий установите его в сети при помощи транзакции 103, описанной ниже в разделе «Транзакции блокчейн-платформы Конфидент» – «Описание транзакций» – «103. CreateContract Transaction». Для этого необходимо подписать транзакцию, например, с помощью метода `sign` REST API, описанного в разделе «Подписание и отправка транзакций» выше.

Важно: REST API метод transactions/sign доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

Данные, возвращенные в ответе метода, подаются на вход при публикации транзакции 103.

Ниже приведены примеры подписания и отправки транзакции при помощи методов `sign` и `broadcast`. В примерах транзакции подписываются ключом, сохраненным в keystore узла.

Curl-запрос на подписание транзакции 103:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --header 'X-Contract-Api-Token' -d ' { \
    "fee": 100000000, \
    "image": "my-contract:latest", \
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65", \
    "contractName": "my-contract", \
    "sender": "3PudkbvjV1nPj1TkuuRahh4sGdgfr4YAUV2", \
    "password": "", \
    "params": [], \
    "type": 103, \
    "version": 1 \
}' 'http://my-node:6862/transactions/sign'
```

Ответ метода `sign`, который передается методу `broadcast`:


```
{
  "type": 103,
  "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
  "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
  "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
  "fee": 100000000,
  "timestamp": 1550591678479,
  "proofs": [ "yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
  "version": 1,
  "image": "my-contract:latest",
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
  "contractName": "my-contract",
  "params": [],
  "height": 1619
}
```

Curl-запрос на отправку транзакции 103:

```
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": "my-contract:latest", \
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65", \
    "contractName": "my-contract", \
    "params": [], \
    "height": 1619 \
}
}' 'http://my-node:6862/transactions/broadcast'
```

4.1.5 Исполнение смарт-контракта

После размещения смарт-контракта в блокчейне он может быть вызван при помощи транзакции 104 CallContract Transaction, которая описана ниже в разделе «Транзакции блокчейн-платформы Конфидент» – «Описание транзакций» – «104. CallContract Transaction»

Эта транзакция также может быть подписана и отправлена в блокчейн посредством метода `sign` REST API, описанного в разделе «Подписание и отправка транзакций» выше.

Важно: REST API метод transactions/sign доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

При подписании транзакции 104 в поле `contractId` укажите идентификатор транзакции 103 для вызываемого смарт-контракта (поле `id` ответа метода `sign`).

Примеры подписания и отправки транзакции при помощи методов `sign` и `broadcast` с использованием ключа, сохраненного в keystore узла:

Curl-запрос на подписание транзакции 104:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --header 'X-Contract-Api-Token' -d '{ \
"contractId": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky", \
"fee": 10, \
"sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew", \
"password": "", \
"type": 104, \
"version": 1, \
"params": [ \
    { \
        "type": "integer", \
        "key": "a", \
        "value": 1 \
    } \
] \
}' 'http://my-node:6862/transactions/sign'
```

Ответ метода `sign`, который передается методу `broadcast`:

```
{
"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
      }
      ]
  }
```

Curl-запрос на отправку транзакции 104:

```
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://my-node:6862/transactions/broadcast'
```

5 Транзакции блокчейн-платформы Конфидент

Транзакция – это отдельная операция в блокчейне от имени участника, изменяющая стейт сети. Отправляя ту или иную транзакцию, участник отправляет в сеть запрос с набором данных, необходимых для соответствующего изменения стейта.

Подписание транзакций доступно только из внешнего ПО, на узел передаются уже подписанные транзакции.

Подписание транзакций на узле доступно только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

5.1 Подписание и отправка транзакций

Перед отправкой транзакции участник генерирует для нее цифровую подпись, используя закрытый ключ своего аккаунта, в соответствии с описанной ниже в разделе «5.3 Описание транзакций» структурой данных соответствующей транзакции.

Подпись транзакции записывается в поле `proofs` при отправке транзакции в блокчейн. Как правило, в это поле записывается одна подпись участника, отправившего транзакцию. Однако поле поддерживает до 8 подписей: в случае подписания транзакции смарт-аккаунтом, при заполнении атомарной транзакции или при публикации смарт-контракта.

После подписания транзакция отправляется в блокчейн – это можно сделать, в частности, при помощи gRPC-интерфейса (см. раздел «gRPC: отправка транзакций в блокчейн» выше).

5.2 Обработка транзакций в блокчейне

Получив транзакцию, узел проверяет ее на валидность:

1. Соответствие временной метки (*timestamp*): временная метка транзакции должна отклоняться от временной метки текущего блока на более, чем на 2 часа назад или 1,5 часа вперед;
2. Тип и версия транзакции: активирована ли в блокчейне поддержка транзакций указанного типа и версии;
3. Соответствие полей транзакции заданному типу данных;
4. Проверка баланса отправителя: достаточно ли средств для оплаты комиссии;
5. Проверка подписи транзакции.

Если транзакция не проходит валидацию, узел отклоняет ее. В случае успешного прохождения проверок транзакция добавляется в пул неподтвержденных транзакций (UTX-пул), где ожидает следующего раунда майнинга для передачи в блокчейн. Вместе с передачей транзакции в UTX-пул узел рассылает ее другим узлам в сети.

Поскольку у каждого микроблока есть ограничение на количество поступающих транзакций, отдельная транзакция может попасть из UTX-пула в блокчейн далеко не сразу. Во время нахождения транзакции в UTX-пуле транзакция может стать невалидной. Например, ее временная метка перестала соответствовать параметрам временной метки текущего блока, либо транзакция, попавшая в блокчейн, уменьшила баланс отправителя, сделав его недостаточным для оплаты транзакции. В таком случае транзакция отклоняется и удаляется из UTX-пула.

После добавления в блок транзакция меняет стейт блокчейна. После этого транзакция считается выполненной.

Платформа «Конфидент» использует следующие транзакции:

- 1. Genesis Transaction
- 3. Issue Transaction
- 4. Transfer Transaction
- 5. Reissue Transaction

5.3 Описание транзакций

Платформа «Конфидент» поддерживает 28 типов транзакций. Для каждой из них предусмотрен свой набор данных, отправляемых в блокчейн.

Запросы и ответы, передаваемые в рамках каждой транзакции по REST API-интерфейсу узла, имеют формат JSON. Формат запросов и ответов, передающихся по gRPC-интерфейсу узла, определяется соответствующими proto-схемами. JSON и protobuf-представления запросов и ответов каждой транзакции приведены ниже.

Примечание: В случае если вы защитили ключевую пару вашего узла паролем при генерации аккаунта, укажите пароль от вашей ключевой пары в поле password транзакции.

5.3.1 1. Genesis Transaction

Первая транзакция нового блокчейна, которая осуществляет первоначальную привязку баланса к адресам созданных узлов.

Подписание этой транзакции не требуется, поэтому выполняется только ее публикация. Транзакция не версионируется.

Структура данных транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (1)
id	Byte	ID транзакции
fee	Long	Комиссия за транзакцию в системном токене
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах)
signature	ByteStr	Подпись генезис-блока. Генерируется при старте блокчейна
recipient	ByteStr	Адрес получателя распределенных токенов
amount	Long	Сумма токенов
height	Int	Высота выполнения транзакции. Для первой транзакции – 1

5.3.2 3. Issue Transaction

Транзакция, инициирующая выпуск токенов в обращение.
Структуры данных транзакции:

Подписание:

Поле	**Тип данных**	**Описание**
type	Byte	Номер транзакции (3)
version	Byte	Версия транзакции
name	Array[byte]	Произвольное имя транзакции
quantity	Long	Количество выпускаемых токенов
description	Array[byte]	Произвольное описание транзакции (в формате base58)
sender	ByteStr	Адрес отправителя транзакции распределенных токенов
password	String	Пароль от ключевой пары в keystore узла – опциональное поле
decimals	Byte	Количество разрядов после запятой у используемого токена (WEST - 8)
reissuable	Boolean	Возможность довыпуска токенов
fee	Long	Комиссия за транзакцию в системном токене

Публикация:

Поле	**Тип данных**	**Описание**
type	Byte	Номер транзакции
id	Byte	ID транзакции
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
fee	Long	Комиссия за транзакцию в системном токене
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) – опциональное поле
proofs	List(ByteStr)	Массив подтверждений транзакции
version	Byte	Версия транзакции
assetId	Byte	ID выпускаемого токена
name	Array[byte]	Произвольное имя транзакции
quantity	Long	Количество выпускаемых токенов
reissuable	Boolean	Возможность довыпуска токенов
decimals	Byte	Количество разрядов после запятой у используемого токена (WAVES – 8)
description	Array[byte]	Произвольное описание транзакции
chainId	Byte	Идентификационный байт сети
script	Array[Byte]	Скрипт для валидации транзакции – опциональное поле
height	Int	Высота выполнения транзакции

JSON-представления запроса и ответа транзакции 3. Issue Transaction приведены ниже.

Version 2

Подписание:

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

Публикация:


```
{
    "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
}
```


5.3.3 4. Transfer Transaction

Транзакция для перевода токенов с одного адреса на другой.
Структуры данных транзакции:

Подписание:

Поле	**Тип данных**	**Описание**
type	Byte	Номер транзакции (4)
version	Byte	Версия транзакции
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore узла – опциональное поле
recipient	ByteStr	Адрес получателя токенов
amount	Long	Сумма токенов
fee	Long	Комиссия за транзакцию в системном токене

Публикация:

Поле	**Тип данных**	**Описание**
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
amount	Long	Сумма токенов
fee	Long	Комиссия за транзакцию в системном токене
type	Byte	Номер транзакции (4)
version	Byte	Версия транзакции
attachment	Byte	Комментарий к транзакции (в формате base58) – опциональное поле
sender	ByteStr	Адрес отправителя транзакции
feeAssetId	Byte	ID токена комиссии – опциональное поле
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
assetId	Byte	ID токена для перевода – опциональное поле
recipient	ByteStr	Адрес получателя токенов
id	Byte	ID транзакции
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) – опциональное поле

JSON-представления запроса и ответа транзакции 4. Transfer Transaction приведены ниже.

Version 2

Подписание:


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


Публикация:


```
{
    "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
}
```


Version 3

Подписание:


```
{
  "type": 4,
  "version": 3,
  "sender": "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
  "password": "",
  "recipient": "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
  "amount": 40000000000,
  "fee": 10000000
}
```


Публикация:


```
{
  "senderPublicKey" : "7GiFGcGaEN87ycK8v71Un6b7RUoeKBU4UvUHPYbeHaki",
  "amount" : 10,
  "fee" : 10000000,
  "type" : 4,
  "version" : 3,
  "atomicBadge" : {
    "trustedSender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx"
  },
  "attachment" : "",
  "sender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
  "feeAssetId" : null,
  "proofs" : [ "2vbAJmwzQw2FCtozcewxJVfxoHxf97BTNdGuaeSATV4vEHZ3XYA4Z7nXGsSnf18aesnAWTKWCfzwM5yGpWEyGM7f" ],
  "assetId" : null,
  "recipient" : "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
  "id" : "2wCEMREFbgk318hFFaNGsgFzyjZHuCrtwSnpK35qhiw4",
  "timestamp" : 1619186861204,
  "height" : 861644
}
```

5.3.4 5. Reissue Transaction

Транзакция для довыпуска токенов.

Структуры данных транзакции:

Подписание:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (5)
version	Byte	Версия транзакции
quantity	Long	Количество токенов для довыпуска
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore узла – опциональное поле
assetId	Byte	ID довыпускаемого токена – опциональное поле
reissuable	Boolean	Возможность довыпуска токенов
fee	Long	Комиссия за транзакцию в системном токене

Публикация:

Поле	Тип данных	Описание
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
quantity	Long	Количество токенов для довыпуска
sender	ByteStr	Адрес отправителя транзакции
chainId	Byte	Идентификационный байт сети
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
assetId	Byte	ID довыпускаемого токена – опциональное поле
fee	Long	Комиссия за транзакцию в системном токене
id	Byte	ID транзакции
type	Byte	Номер транзакции (5)
version	Byte	Версия транзакции
reissuable	Boolean	Возможность довыпуска токенов
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) – опциональное поле
height	Int	Высота выполнения транзакции

JSON-представления запроса и ответа транзакции 5. Reissue Transaction приведены ниже.

Version 2

Подписание:

```
{
    "type": 5,
    "version":2,
    "quantity": 556105,
    "sender": "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
    "password": "",
    "assetId": "6UAMZA6RshxyPvt9W7aoWiUiB6N73yLQMMfiRQYXdWZh",
    "reissuable": true,
    "fee": 100000000
}
```

Публикация:

```
{
  "senderPublicKey" : "7GiFGcGaEN87ycK8v71Un6b7RUoeKBU4UvUHPYbeHaki",
  "quantity" : 556105,
  "fee" : 100000000,
  "type" : 5,
  "version" : 2,
  "reissuable" : true,
  "sender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
  "chainId" : 86,
  "proofs" : [ "5ahD78wciu8YTsLoxo1XRghJWAGG7At7ePiBWTNzdkvX7cViRCKRLjjjPTGCoAH2mdGQK9i1JiY1wh18eh4h7pGy" ],
  "assetId" : "6UAMZA6RshxyPvt9W7aoWiUiB6N73yLQMMfiRQYXdWZh",
  "id" : "8T9jJUusN5KBexxDUX1XBjoDydXGP34zWH7Qvp5mnmES",
  "timestamp" : 1619187184206,
  "height" : 861645
}
```

5.3.5 6. Burn Transaction

Транзакция для сжигания токенов: уменьшает количество токенов на счету отправителя, тем самым снижая общее количество токенов в обращении. Сожженные токены невозможно восстановить.

Структуры данных транзакции:

Подписание:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (6)
version	Byte	Версия транзакции
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore узла – опциональное поле
assetId	Byte	ID сжигаемого токена – опциональное поле
quantity	Long	Количество токенов для сжигания
fee	Long	Комиссия за транзакцию в системном токене
attachment	Byte	Комментарий к транзакции (в формате base58) – опциональное поле

Публикация:

Поле	Тип данных	Описание
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
amount	Long	Количество токенов для сжигания
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore узла – опциональное поле
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
assetId	Byte	ID сжигаемого токена – опциональное поле
fee	Long	Комиссия за транзакцию в системном токене
id	Byte	ID транзакции
type	Byte	Номер транзакции (6)
version	Byte	Версия транзакции
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) – опциональное поле
height	Int	Высота выполнения транзакции

JSON-представления запроса и ответа транзакции 6. Burn Transaction приведены ниже.

Version 2

Подписание:

```
{
    "type": 6,
    "version": 2,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "password": "",
    "assetId": "7bE3JPwZC3QcN9edctFrLAKYysjfMEk1SDjZx5gitSGg",
    "quantity": 1000,
    "fee": 100000,
    "attachment": "string"
}
```

Публикация:

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

5.3.6 8. Lease Transaction

Передача токенов в аренду другому адресу. Средства, переданные в аренду, начинают учитываться в генерирующем балансе получателя через 1000 блоков.

Передача токенов в лизинг может проводиться для повышения вероятности выбора узла в качестве майнера следующего раунда. Как правило, в обмен на аренду токенов получатель делится вознаграждением, полученным за генерацию блока, с адресом, предоставившим токены в лизинг.

Токены, переданные в лизинг, остаются заблокированными на адресе отправителя. Отмена лизинга производится с помощью транзакции отмены лизинга.

Структуры данных транзакции:

Подписание:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (8)
version	Byte	Версия транзакции
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore узла – опциональное поле
recipient	ByteStr	Адрес получателя токенов
amount	Long	Количество токенов для передачи в аренду
fee	Long	Комиссия за транзакцию в системном токене

Публикация:

Поле	Тип данных	Описание
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
amount	Long	Количество токенов для передачи в аренду
sender	ByteStr	Адрес отправителя транзакции
proofs	List(ByteStr)	Массив подтверждений транзакции (в base58) формате
fee	Long	Комиссия за транзакцию в системном токене
recipient	ByteStr	Адрес получателя токенов
id	Byte	ID транзакции
type	Byte	Номер транзакции (8)
version	Byte	Версия транзакции
height	Int	Высота выполнения транзакции
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) – опциональное поле

JSON-представления запроса и ответа транзакции 8. Lease Transaction приведены ниже.

Version 2

Подписание:

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

Публикация:

```
{
    "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
    "amount": 1000,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "proofs": [ "5jvmWKmU89HnxXFXNAd9X41zmiB5fSGoXMirsaJ9tNeyiCAJmjm7MR48g789VucckQw2UExaVXfhsdEBuUrchvrq" ],
    "fee": 100000,
    "recipient": "3N1ksBqc6uSksdiYjCzMtvEpiHhS1JjkbPh",
    "id": "6Tn7ir9MycHW6Gq2F2dGok2stokSwXJadPh4hW8eZ8Sp",
    "type": 8,
    "version": 2,
    "timestamp": 1551449299545,
    "height": 1190
}
```

5.3.7 9. LeaseCancel Transaction

Отмена аренды токенов, переданных в транзакции с определенным ID. Структура lease данной транзакции не заполняется: узел автоматически заполняет ее при предоставлении данных о транзакции.

Структуры данных транзакции:

Подписание:

Поле	**Тип данных**	**Описание**
type	Byte	Номер транзакции (9)
version	Byte	Версия транзакции
fee	Long	Комиссия за транзакцию в системном токене
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore узла – опциональное поле
txId	Byte	ID транзакции аренды токенов

Публикация:

Поле	**Тип данных**	**Описание**
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
leaseId	Byte	ID транзакции аренды токенов
sender	ByteStr	Адрес отправителя транзакции
chainId	Byte	Идентификационный байт сети
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
fee	Long	Комиссия за транзакцию в системном токене
id	Byte	ID транзакции отмены аренды токенов
type	Byte	Номер транзакции (9)
version	Byte	Версия транзакции
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) – опциональное поле
height	Int	Высота выполнения транзакции

JSON-представления запроса и ответа транзакции 9. LeaseCancel Transaction приведены ниже.

Version 2

Подписание:


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


Публикация:


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

5.3.8 10. CreateAlias Transaction

Создание псевдонима для адреса отправителя. Псевдоним может использоваться для проведения транзакций в качестве идентификатора получателя.

Структуры данных транзакции:

Подписание:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (10)
version	Byte	Версия транзакции
fee	Long	Комиссия за транзакцию в системном токене
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore узла – опциональное поле
alias	Byte	Псевдоним

Публикация:

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (10)
id	Byte	ID транзакции создания псевдонима
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
fee	Long	Комиссия за транзакцию в системном токене
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) – опциональное поле
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
version	Byte	Версия транзакции
alias	Byte	Псевдоним
height	Byte	Высота выполнения транзакции

JSON-представления запроса и ответа транзакции 10. CreateAlias Transaction приведены ниже.

Version 2

Подписание:


```
{
    "type": 10,
    "version": 2,
    "fee": 100000000,
    "sender": "3NwTvbW7TMckBc785XjtGTUfHmcesaWBe1A",
    "password": "",
    "alias": "1@k1_kv29"
}
```


Публикация:


```
{
  "senderPublicKey" : "C4eRfdUFaZMRkfUp91bYr7uMgdBRnUfAxuAjetxmK7KY",
  "sender" : "3NwTvbW7TMckBc785XjtGTUfHmcesaWBe1A",
  "proofs" : [ "3fhJztBNnTDjppmqgi4GugAYo1aS1mzZhVhPdnNsqYqCEyLLHfzgb75psRPntHD4uBZgk8jByFP9mwwx2Ezsdg59" ],
  "fee" : 100000000,
  "alias" : "1@k1_kv29",
  "id" : "AavgVzV7avPMpERro6YqikwFESAgG2wViprtPJUtXP6F",
  "type" : 10,
  "version" : 2,
  "timestamp" : 1608737444468,
  "height" : 595942
}
```

5.3.9 11. MassTransfer Transaction

Перевод токенов нескольким получателям (от 1 до 100 адресов). Комиссия за транзакцию зависит от количества задействованных адресов.

Структуры данных транзакции:

Подписание:

Поле	**Тип данных**	**Описание**
type	Byte	Номер транзакции (11)
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore узла – опциональное поле
fee	Long	Комиссия за транзакцию в системном токене
version	Byte	Версия транзакции
transfers	List	Список получателей с полями `recipient` и `amount` через запятую
recipient	ByteStr	Адрес получателя токенов
amount	Long	Количество токенов для передачи адресу

Публикация:

Поле	**Тип данных**	**Описание**
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
fee	Long	Комиссия за транзакцию в системном токене
type	Byte	Номер транзакции (11)
transferCount	Byte	Количество адресов-получателей
version	Byte	Версия транзакции
totalAmount	Byte	Общая сумма токенов для перевода
attachment	Byte	Комментарий к транзакции (в формате base58) – опциональное поле
sender	ByteStr	Адрес отправителя транзакции
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
assetId	Byte	ID токена для перевода – опциональное поле
id	Byte	ID транзакции перевода токенов
transfers	List	Список получателей с полями `recipient` и `amount` через запятую
transfers.recipient	ByteStr	Адрес получателя токенов
transfers.amount	Long	Количество токенов для передачи адресу
height	Byte	Высота выполнения транзакции

Пример заполнения поля transfers и JSON-представления запроса и ответа транзакции 11. MassTransfer Transaction приведены ниже.

Version 2

Подписание:

```
{
    "type": 11,
    "sender": "3NydXoTq3UgUW5rxsNwEMs1iwbbvVEwxoHU",
    "password": "",
    "fee": 30000000,
    "version": 2,
    "transfers":
    [
       { "recipient": "3MtHszoTn399NfsH3v5foeEXRRrchEVtTRB", "amount": 100000 },
       { "recipient": "3N7BA6J9VUBfBRutuMyjF4yKTUEtrRFfHMc", "amount": 100000 }
    ]
}
```

Публикация:


```
{
  "senderPublicKey" : "AMhAY8RMy5QsPqj58xeMY3fJxTZKx71QztsjDzqWprHo",
  "fee" : 30000000,
  "type" : 11,
  "transferCount" : 4,
  "version" : 2,
  "totalAmount" : 400000000,
  "attachment" : "",
  "sender" : "3NydXoTq3UgUW5rxsNwEMs1iwbbvVEwxoHU",
  "feeAssetId" : "8bec1mhqTiveMeRTHgYr6az12XdqBBtpeV3ZpXMRHfSB",
  "proofs" : [ "21hhAMmwze6nLLQ9K6AoU6scek9Sk5KabR4VggGfdTVFHonfMGwVTse6qL2f8zR8DRm7RckMaikiYRt5XxWEKWcA" ],
  "assetId" : "8bec1mhqTiveMeRTHgYr6az12XdqBBtpeV3ZpXMRHfSB",
  "transfers" : [ {
    "recipient" : "3NqEjAkFVzem9CGa3bEPhakQc1Sm2G8gAFU",
    "amount" : 100000000
  }, {
    "recipient" : "3NzkzibVRkKUzaRzjUxndpTPvoBzQ3iLng3",
    "amount" : 100000000
  }, {
    "recipient" : "3Nnx8cX3UiyfQeC3YQKVRqVr2ewSxrvaDyB",
    "amount" : 100000000
  }, {
    "recipient" : "3NzC4Ex91VBQKfJHPiGhuPEomLg48NMi2ZF",
    "amount" : 100000000
  } ],
  "id" : "EvnxFxdYhYxHgQSMhkyLaqgyUDZdnBknfAWEXyqEHt97",
  "timestamp" : 1627643861044,
  "height" : 1076874
}
```


5.3.10 12 Data Transaction

Транзакция для добавления, изменения или удаления записей в хранилище данных адреса. В хранилище данных адреса представлены записи в формате «ключ:значение».

Размер хранилища данных адреса неограничен, однако при помощи одной транзакции данных можно внести до 100 новых пар «ключ:значение». Также байтовое представление транзакции после подписания не должно превышать 150 килобайт.

Если автор данных (адрес в поле `author`) совпадает с отправителем транзакции (адрес в поле `sender`), при подписании транзакции не требуется указывать параметр `senderPublicKey`.

Структура данных запроса на подписание транзакции:

Подписание:

Поле	**Тип данных**	**Описание**
type	Byte	Номер транзакции (12)
version	Byte	Версия транзакции
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore узла – опциональное поле
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
author	Byte	Адрес автора вносимых данных
data	List	Список данных, в который вносятся поля `key`:, `type`: и `value`: через запятую
data.key	Byte	Ключ записи
data.type	Byte	Тип данных записи. Возможные значения: `binary`, `bool`, `integer`, `string`, и `null` (удаление записи по ее ключу)
data.value	Byte	Значение записи
fee	Long	Комиссия за транзакцию в системном токене

Публикация:

Поле	**Тип данных**	**Описание**
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
senderPublicKey	PublicKeyAccount	Открытый ключ автора данных
data	List	Список данных с полями “`key`:“, “`type`:“ и “`value`:“ через запятую

data.key	Byte	Ключ записи
data.type	Byte	Тип данных записи. Возможные значения: `binary`, `bool`, `integer`, `string` и `null` (удаление записи по ее ключу)
data.value	Byte	Значение записи
sender	ByteStr	Адрес отправителя транзакции
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
author	Byte	Адрес автора вносимых данных
fee	Long	Комиссия за транзакцию в системном токене
id	Byte	ID транзакции с данными
type	Byte	Номер транзакции (12)
version	Byte	Версия транзакции
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) – опциональное поле

Пример заполнения поля data и JSON-представления запроса и ответа транзакции 12 Data Transaction приведены ниже.

Version 2

Подписание:

```
{
    "type": 12,
    "version": 2,
    "sender": "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
    "password": "",
    "senderPublicKey": "7GiFGcGaEN87ycK8v71Un6b7RUoeKBU4UvUHPYbeHaki",
    "author": "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
    "data": [
      ...
    ],
    "fee": 150000000
}
```

Публикация:

```
{
  "senderPublicKey" : "7GiFGcGaEN87ycK8v71Un6b7RUoeKBU4UvUHPYbeHaki",
  "data" : [
    ...
  ],
  "author" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
  "fee" : 150000000,
  "type" : 12,
  "version" : 2,
  "authorPublicKey" : "7GiFGcGaEN87ycK8v71Un6b7RUoeKBU4UvUHPYbeHaki",
  "sender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
  "feeAssetId" : null,
  "proofs" : [ "4wFNmn32NZqGwP4D4aAxCMyigGEVZLWftqi919pHAK7mCj3sFw7Ekf76g2rr51PZuk5sLwzjkKiZArQvWY8uEGqk" ],
  "id" : "GcDy84oTFf5NQzDtixkfUqiFNZwMaN2vfXqxsbGxumfo",
  "timestamp" : 1619187166499,
  "height" : 861644
}
```

5.3.11 13. SetScript Transaction

Транзакция для привязки скрипта к аккаунту или удаления скрипта. Аккаунт с привязанным к нему скриптом называется *смарт-аккаунтом*.

Скрипт позволяет верифицировать транзакции, передаваемые от имени аккаунта, без использования механизма верификации транзакций блокчейна.

Структуры данных транзакции:

Подписание:

Поле	**Тип данных**	**Описание**
type	Byte	Номер транзакции (13)
version	Byte	Версия транзакции
sender	ByteStr	Адрес отправителя транзакции

password	String	Пароль от ключевой пары в keystore узла – опциональное поле
fee	Long	Комиссия за транзакцию в системном токене
name	Array[Byte]	Имя скрипта
script	Array[Byte]	Скомпилированный скрипт в формате base64. Если вы оставите это поле пустым (`null`) – скрипт будет отвязан от аккаунта

Публикация:

Поле	**Тип данных**	**Описание**
type	Byte	Номер транзакции (13)
id	Byte	ID транзакции установки скрипта
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
fee	Long	Комиссия за транзакцию в системном токене
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) – опциональное поле
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
chainId	Byte	Идентификационный байт сети
version	Byte	Версия транзакции
script	Array[Byte]	Скомпилированный скрипт в формате base64 – опциональное поле
name	Array[Byte]	Имя скрипта
description	Byte	Комментарий к транзакции (в формате base58) – опциональное поле
height	Byte	Высота выполнения транзакции

JSON-представления запроса и ответа транзакции 13. SetScript Transaction приведены ниже.

Version 1

Подписание:


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


Публикация:


```
{
    "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
}
```

5.3.12 14. Sponsorship Transaction

Транзакция, устанавливающая или отменяющая спонсирование.

Механизм спонсирования позволяет адресам выплачивать комиссии за транзакции вызова скрипта и транзакции перевода в спонсорском ассете, заменяющем WEST.

Структуры данных транзакции:

Подписание:

Поле	Тип данных	Описание
sender	ByteStr	Адрес отправителя транзакции
assetId	Byte	ID спонсорского ассета (токена) – опциональное поле
fee	Long	Комиссия за транзакцию в системном токене
isEnabled	Bool	Установка спонсирования (`true`) или его отмена (`false`)
type	Byte	Номер транзакции (**14**)
password	String	Пароль от ключевой пары в keystore узла – опциональное поле
version	Byte	Версия транзакции

Публикация:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**14**)
id	Byte	ID транзакции спонсирования
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
fee	Long	Комиссия за транзакцию в системном токене
assetId	Byte	ID спонсорского ассета (токена) – опциональное поле
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) – опциональное поле
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
chainId	Byte	Идентификационный байт сети
version	Byte	Версия транзакции
isEnabled	Bool	Установка спонсирования (`true`) или его отмена (`false`)
height	Byte	Высота выполнения транзакции

JSON-представления запроса и ответа транзакции 14. Sponsorship Transaction приведены ниже.

Version 1

Подписание:

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

Публикация:

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

5.3.13 15. SetAssetScript Transaction

Транзакция для установки или удаления скрипта ассета для адреса. Скрипт ассета позволяет верифицировать транзакции с участием того или иного ассета (токена) без использования механизма верификации транзакций блокчейна.

Структуры данных транзакции:

Подписание:

Поле	**Тип данных**	**Описание**
type	Byte	Номер транзакции (**15**)
version	Byte	Версия транзакции скрипта ассета
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore узла – опциональное поле
fee	Long	Комиссия за транзакцию в системном токене
script	Array[Byte]	Скомпилированный скрипт в формате base64 – опциональное поле
assetId	Byte	ID спонсорского ассета (токена) – опциональное поле

Публикация:

Поле	**Тип данных**	**Описание**
type	Byte	Номер транзакции (15)
id	Byte	ID транзакции скрипта ассета
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
fee	Long	Комиссия за транзакцию в системном токене
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) – опциональное поле
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
version	Byte	Версия транзакции
chainId	Byte	Идентификационный байт сети
assetId	Byte	ID спонсорского ассета (токена) – опциональное поле
script	Array[Byte]	Скомпилированный скрипт в формате base64. Если вы оставите это поле пустым (`null`) – скрипт будет отвязан от аккаунта
height	Byte	Высота выполнения транзакции

JSON-представления запроса и ответа транзакции 15. SetAssetScript Transaction приведены ниже.

Version 1

Подписание:

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

Публикация:

```
{
    "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
}
```

5.3.14 102. Permission Transaction

Выдача или отзыв роли участника. Отправлять транзакцию 102 в блокчейн может только участник с ролью `permissioner`.

Возможные роли для указания в поле `role`:

- permissioner;
- sender;
- blacklister;
- miner;
- issuer;
- contract_developer;
- connection_manager;
- contract_validator;
- banned.

Описание ролей см. в разделе «6.1 Описание ролей участников сети» документа «643.19172778.2019662198-01 95. «Блокчейн-платформа Конфидент» Версия 1.9. Правила пользования»; этот документ поставляется вместе с дистрибутивом платформы.

Структуры данных транзакции:

Подписание:

Поле	**Тип данных**	**Описание**
type	Byte	Номер транзакции (102)
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore узла – опциональное поле
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
fee	Long	Комиссия за транзакцию в системном токене
target	ByteStr	Адрес участника для назначения роли
opType	String	Тип операции: `add` – добавить роль; `remove` – отозвать роль
dueTimestamp	Long	Временная метка срока действия роли в формате Unix Timestamp (в миллисекундах) – опциональное поле
version	Byte	Версия транзакции

Публикация:

Поле	**Тип данных**	**Описание**
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
role	String	Назначаемая роль (для администратора – `permissioner`)
sender	ByteStr	Адрес отправителя транзакции
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
fee	Long	Комиссия за транзакцию в системном токене
opType	String	Тип операции: `add` – добавить роль; `remove` – отозвать роль
id	Byte	ID транзакции назначения или отмены роли
type	Byte	Номер транзакции (102)
dueTimestamp	Long	Временная метка срока действия роли в формате Unix Timestamp (в миллисекундах) – опциональное поле
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) – опциональное поле
target	ByteStr	Адрес назначаемого первого администратора

JSON-представления запроса и ответа транзакции 102. Permission Transaction приведены ниже.

Version 1

Подписание:


```
{
   "type": 102,
   "sender": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
   "password": "",
   "senderPublicKey": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
   "fee": 0,
   "target": "3GPtj5osoYqHpyfmsFv7BMiyKsVzbG1ykfL",
   "opType": "add",
   "role": "contract_developer",
   "dueTimestamp": null,
   "version": 1
 }
```


Публикация:


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


5.3.15 101. GenesisPermission Transaction

Транзакция для назначения первого администратора сети, который раздает роли другим участникам.

Структуры данных транзакции:

Подписание:

Поле	**Тип данных**	**Описание**
type	Byte	Номер транзакции (101)
id	Byte	ID транзакции
fee	Long	Комиссия за транзакцию в системном токене
timestam	p Long	Временная метка в формате Unix Timestamp (в миллисекундах) – опциональное поле
signature	ByteStr	Подпись транзакции (в формате base58)
target	ByteStr	Адрес назначаемого первого администратора
role	String	Назначаемая роль (для администратора – `permissioner`)

Публикация:

Поле	**Тип данных**	**Описание**
type	Byte	Номер транзакции (101)
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) – опциональное поле
target	ByteStr	Адрес назначаемого первого администратора
role	String	Назначаемая роль (для администратора – `permissioner`)

5.3.16 103. CreateContract Transaction

Создание смарт-контракта. Байтовое представление этой транзакции после ее подписания не должно превышать 150 килобайт.

Поле `feeAssetId` этой транзакции опционально и используется только для смарт-контрактов с поддержкой gRPC. Значение поля `version` для этого типа смарт-контрактов – 2.

Подписание транзакции 103 может производиться только пользователем с ролью `contract_developer`.

Структуры данных транзакции:

Подписание:

Поле	Тип данных	Описание
fee	Long	Комиссия за транзакцию в системном токене
image	Array[Bytes]	Имя Docker-образа смарт-контракта
imageHash	Array[Bytes]	Хэш Docker-образа смарт-контракта
contractName	Array[Bytes]	Имя смарт-контракта (при загрузке из предустановленного репозитория) или его полный адрес (если репозиторий смарт-контракта не указан в конфигурационном файле узла)
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore узла – опциональное поле
params	List[DataEntry[_]]	Входные и выходные данные смарт-контракта. Вносятся при помощи полей `type`, `value` и `key` через запятую – опциональное поле
params.key	Byte	Ключ параметра
params.type	Byte	Тип данных параметра. Возможные значения: `binary`, `bool`, `integer`, `string`
params.value	Byte	Значение параметра
type	Byte	Номер транзакции (103)
version	Byte	Версия транзакции
apiVersion	Byte	Версия API для gRPC-методов смарт-контракта (см. раздел «API-инструменты, доступные смарт-контракту» выше).
validationPolicy.type	String	Тип политики валидации смарт-контрактов.

Публикация:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (103)
id	Byte	ID транзакции создания контракта
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
fee	Long	Комиссия за транзакцию в системном токене
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - опциональное поле
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
version	Byte	Версия транзакции
image	Array[Bytes]	Имя смарт-контракта (при загрузке из предустановленного репозитория) или его полный адрес (если репозиторий смарт-контракта не указан в конфигурационном файле узла)
imageHash	Array[Bytes]	Хэш Docker-образа смарт-контракта
contractName	Array[Bytes]	Имя смарт-контракта
params	List[DataEntry[_]]	Входные и выходные данные смарт-контракта. Вносятся при помощи полей `type`, `value` и `key` через запятую – опциональное поле
params.key	Byte	Ключ параметра
params.type	Byte	Тип данных параметра. Возможные значения: `binary`, `bool`, `integer`, `string`
params.value	Byte	Значение параметра

height	Byte	Высота выполнения транзакции
apiVersion	Byte	Версия API для gRPC-методов смарт-контракта (см. раздел «API-инструменты, доступные смарт-контракту» выше).
validationPolicy.type	String	Тип политики валидации смарт-контрактов.

JSON-представления запроса и ответа транзакции 103. CreateContract Transaction приведены ниже.

Version 2

Подписание:


```
{
  "type": 103,
  "version": 2,
  "sender": "3NpN3HyHzGj7Ny1k5F9zMMQ2n54TZg86G9D",
  "password": "signing-key-password",
  "image": "registry.yourdomain.com/test-docker-repo/contract:v1.0.0",
  "contractName": "Your contract name",
  "imageHash": "573387bbf50cfdeda462054b8d85d6c24007f91044501250877392e43ff5ed50",
  "params": [
    {
      "type": "string",
      "key": "test_key",
      "value": "test_value"
    }
  ],
  "fee": 100000000,
  "timestamp": 1651487626477,
  "feeAssetId": null
}
```


Публикация:


```
{
  "id": "4WVhw3QdiinpE5QXDG7QfqLiLanM7ewBw4ChX4qyGjs2",
  "type": 103,
  "version": 2,
  "sender": "3NpN3HyHzGj7Ny1k5F9zMMQ2n54TZg86G9D",
  "senderPublicKey": "YNpp7chAaudMqEtSZZPyN4GYLJ5ZTXdjCXrQdszzuRp",
  "contractName": "Your contract name",
  "image": "registry.yourdomain.com/test-docker-repo/contract:v1.0.0",
  "imageHash": "573387bbf50cfdeda462054b8d85d6c24007f91044501250877392e43ff5ed50",
  "params": [
    {
      "type": "string",
      "key": "test_key",
      "value": "test_value"
    }
  ],
  "fee": 100000000,
  "timestamp": 1651487626477,
  "feeAssetId": null,
  "proofs": [
    "4vqLnpJRFpcDgM5vgi78DpZnVfqztsARHNb7Hbmq3mQBjS3SRnzFAiYjRvPazEVMhBM9cE4Rcp6H5K29kk75Uxyh"
  ]
}
```

Version 3

Подписание:


```
{
  "type": 103,
  "version": 3,
  "sender": "3NpN3HyHzGj7Ny1k5F9zMMQ2n54TZg86G9D",
  "password": "signing-key-password",
  "image": "registry.yourdomain.com/test-docker-repo/contract:v1.0.0",
  "contractName": "Your contract name",
  "imageHash": "573387bbf50cfdeda462054b8d85d6c24007f9104450125087739 2e43ff5ed50",
  "params": [
    {
      "type": "string",
      "key": "test_key",
      "value": "test_value"
    }
  ],
  "fee": 100000000,
  "timestamp": 1651487626477,
  "feeAssetId": null,
  "atomicBadge": null
}
```


Публикация:


```
{
  "id": "4WVhw3QdiinpE5QXDG7QfqLiLanM7ewBw4ChX4qyGjs2",
  "type": 103,
  "version": 3,
  "sender": "3NpN3HyHzGj7Ny1k5F9zMMQ2n54TZg86G9D",
  "senderPublicKey": "YNpp7chAaudMqEtSZZPyN4GYLJ5ZTXdjCXrQdszzuRp",
  "contractName": "Your contract name",
  "image": "registry.yourdomain.com/test-docker-repo/contract:v1.0.0",
  "imageHash": "573387bbf50cfdeda462054b8d85d6c24007f9104450125087739 2e43ff5ed50",
  "params": [
    {
      "type": "string",
      "key": "test_key",
      "value": "test_value"
    }
  ],
  "fee": 100000000,
  "timestamp": 1651487626477,
  "feeAssetId": null,
  "atomicBadge": null,
  "proofs": [
    "4vqLnpJRFpcDgM5vgi78DpZnVfqztsARHNb7Hbmq3mQBjS3SRnzFAiYjRvPazEVMhBM9cE4Rcp6H5K29kk75Uxyh"
  ]
}
```

Version 4

Подписание:

```
{
  "type": 103,
  "version": 4,
  "sender": "3NpN3HyHzGj7Ny1k5F9zMMQ2n54TZg86G9D",
  "password": "signing-key-password",
  "image": "registry.yourdomain.com/test-docker-repo/contract:v1.0.0",
  "contractName": "Your contract name",
  "imageHash": "573387bbf50cfdeda462054b8d85d6c24007f91044501250877392e43ff5ed50",
  "params": [
    {
      "type": "string",
      "key": "test_key",
      "value": "test_value"
    }
  ],
  "fee": 100000000,
  "timestamp": 1651487626477,
  "feeAssetId": null,
  "atomicBadge": null,
  "validationPolicy": {
    "type": "majority"
  },
  "apiVersion": "1.0"
}
```

Публикация:

```
{
  "id": "4WVhw3QdiinpE5QXDG7QfqLiLanM7ewBw4ChX4qyGjs2",
  "type": 103,
  "version": 4,
  "sender": "3NpN3HyHzGj7Ny1k5F9zMMQ2n54TZg86G9D",
  "senderPublicKey": "YNpp7chAaudMqEtSZZPyN4GYLJ5ZTXdjCXrQdszzuRp",
  "contractName": "Your contract name",
  "image": "registry.yourdomain.com/test-docker-repo/contract:v1.0.0",
  "imageHash": "573387bbf50cfdeda462054b8d85d6c24007f91044501250877392e43ff5ed50",
  "params": [
    {
      "type": "string",
      "key": "test_key",
      "value": "test_value"
    }
  ],
  "fee": 100000000,
  "timestamp": 1651487626477,
  "feeAssetId": null,
  "atomicBadge": null,
  "proofs": [
    "4vqLnpJRFpcDgM5vgi78DpZnVfqztsARHNb7Hbmq3mQBjS3SRnzFAiYjRvPazEVMhBM9cE4Rcp6H5K29kk75Uxyh"
  ]
}
```

В версии 4 данной транзакции настраивается валидация результатов исполнения обновляемого смарт-контракта при помощи поля `validationPolicy.type`. Варианты политик валидации:

- `any` – сохраняется действующая в сети общая политика валидации: для майнинга загружаемого смарт-контракта, майнер подписывает соответствующую транзакцию 105, описанную ниже в разделе «105. ExecutedContract Transaction». Также этот параметр устанавливается, если в вашей сети нет ни одного зарегистрированного валидатора.
- `majority` – транзакция считается валидной, если её подтвердило большинство валидаторов: 2/3 от общего числа зарегистрированных адресов с ролью `contract_validator`.
- `majorityWithOneOf(List[Address])` – транзакция считается валидной, если собрано большинство валидаторов, среди которых присутствует хотя бы один из адресов, включенных в список параметра. Адреса, включаемые в список, должны иметь действующую роль `contract_validator`.

Предупреждение: При выборе политики валидации `majorityWithOneOf(List[Address])`, заполните список адресов, передача пустого списка запрещена.

При работе в частной сети транзакция 103 предусматривает загрузку Docker-образа контракта не только из репозиториев, указанных в секции `docker-engine` конфигурационного файла узла. Если вам необходимо загрузить смарт-контракт из репозитория, не внесенного в конфигурационный файл, укажите в поле `name` транзакции полный адрес смарт-контракта в созданном вами репозитории.

Ниже приведён пример запроса на публикацию смарт-контракта из непредустановленного репозитория.

```
{
  "senderPublicKey" : "CgqRPcPnexY533gCh2SSvBXh5bca1qMs7KFGntawHGww",
  "image": "customregistry.com:5000/stateful-increment-contract:latest",
  "fee" : 100000000,
  "imageHash" : "ad6d0f8a61222794da15571749bc9db08e76b6a120fc1db90e393fc0ee9540d8",
  "type" : 103,
  "params" : [ {
    "type" : "string",
    "value" : "Value_here",
    "key" : "data"
  }, {
    "type" : "integer",
    "value" : 500,
    "key" : "length"
  } ],
  "version" : 4,
  "atomicBadge" : null,
  "apiVersion" : "1.0",
  "sender" : "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "feeAssetId" : null,
  "proofs" : [ "L521YncSMJDPqwBjQyS7m7Q6tseAw51nYE8iiPChEALx7S2WvpSosCVtWkXxh2ZqJ6LHkCvjVjRVuVs793kzjw8" ],
  "contractName" : "grpc_validatable_statefull here_often",
  "id" : "HSLdKYqLq4LcZpq9LPki8Yv4ZRkFapVyHEYw1vZW2MoG",
  "validationPolicy" : {
    "type" : "any"
  },
  "timestamp" : 1625732696641,
  "height" : 1028130
}
```

5.3.17 104. CallContract Transaction

Вызов смарт-контракта на исполнение. Байтовое представление этой транзакции после ее подписания не должно превышать 150 килобайт.

Подписание транзакции производится инициатором исполнения контракта.

В поле `contractVersion` транзакции указывается версия контракта:

- `1` – для нового контракта;
- `2` – для обновленного контракта.

Данное поле доступно только для транзакций второй версии и старше: если в поле `version` транзакции создания смарт-контракта указано значение >= `2`. Контракт обновляется при помощи транзакции 107, описанной ниже в разделе «107. UpdateContract Transaction».

Если контракт не выполнился или выполнился с ошибкой, то транзакции 103 и 104 удаляются и не попадают в блок.

Структуры данных транзакции:

Подписание:

Поле	Тип данных	Описание
contractId	ByteStr	ID смарт-контракта
fee	Long	Комиссия за транзакцию в системном токене
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore узла – опциональное поле
type	Byte	Номер транзакции (104)
params	List[DataEntry[_]]	Входные и выходные данные смарт-контракта. Вносятся при помощи полей `type, value` и `key` через запятую – опциональное поле
params.key	Byte	Ключ параметра
params.type	Byte	Тип данных параметра. Возможные значения: `binary, bool, integer, string`
params.value	Byte	Значение параметра

version	Byte	Версия транзакции

Публикация:

Поле	**Тип данных**	**Описание**
type	Byte	Номер транзакции (104)
id	Byte	ID транзакции вызова контракта
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
fee	Long	Комиссия за транзакцию в системном токене
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) опциональное поле
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
version	Byte	Версия транзакции
contractId	ByteStr	ID смарт-контракта
params	List[DataEntry[_]]	Входные и выходные данные смарт-контракта. Вносятся при помощи полей `type, value и key` через запятую - опциональное поле
params.key	Byte	Ключ параметра
params.type	Byte	Тип данных параметра. Возможные значения: `binary, bool, integer, string`
params.value	Byte	Значение параметра

JSON-представления запроса и ответа транзакции 104. CallContract Transaction приведены ниже.

Version 2

Подписание:

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

Публикация:


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


Version 3

Подписание:


```
{
    "contractId": "Dgk1hR7xRnDT1KJreaXCVtZLrnd5LJ8uUYtoZyQrV1LJ",
    "fee": 10000000,
    "sender": "3NpkC1FSW9xNfmAMuhRSRArLgnfyGyEry7w",
    "password": "",
    "type": 104,
    "params":
    [ {
      "type" : "string",
      "value" : "value",
      "key" : "data"
    }, {
      "type" : "integer",
      "value" : 500,
      "key" : "length"
    } ],
    "version": 3,
    "contractVersion": 1,
}
```

Публикация:


```
{
  "senderPublicKey" : "9Kgnqqxr5MU3PNrLgf1dkZL2HH6LBktB5Pv9L1cVELi1",
  "fee" : 10000000,
  "type" : 104,
  "params" : [ {
    "type" : "string",
    "value" : "data_response",
    "key" : "action"
  }, {
    "type" : "string",
    "value" : "000008_regular_data_request_2m3SgcnQz9LXVi9ETy3CFHVGM1EyiqJi3vvRRQUM3oPp",
    "key" : "request_id"
  }, {
    "type" : "string",
    "value" : "76.33",
    "key" : "value"
  }, {
    "type" : "string",
    "value" : "1627678789267",
    "key" : "timestamp"
  } ],
  "version" : 3,
  "contractVersion" : 1,
  "sender" : "3NpkC1FSW9xNfmAMuhRSRArLgnfyGyEry7w",
  "feeAssetId" : null,
  "proofs" : [ "4aanqYjaTVNot8Fbz5ixjwKSdqS5x3DdvzxQ4WsTaPcftYdoFx99xwLC3UPN91VAtez4RTMzaYb1TECaVxHHT9AH" ],
  "contractId" : "Dgk1hR7xRnDT1KJreaXCVtZLrnd5LJ8uUYtoZyQrV1LJ",
  "id" : "55imLuEXyVpBXb1S64R5PRx9acQQHaEATPwYwUVpqjAT",
  "timestamp" : 1627678789267,
  "height" : 1076064
}
```


Version 4

Подписание:


```
{
    "contractId": "HSLdKYqLq4LcZpq9LPki8Yv4ZRkFapVyHEYw1vZW2MoG",
    "fee": 10000000,
    "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
    "password": "",
    "type": 104,
    "params":
    [ {
      "type" : "string",
      "value" : "value",
      "key" : "data"
    }, {
      "type" : "integer",
      "value" : 500,
      "key" : "length"
    } ],
    "version": 4,
    "contractVersion": 3,
    "atomicBadge" : null
}
```

Публикация:


```
{
  "senderPublicKey" : "CgqRPcPnexY533gCh2SSvBXh5bca1qMs7KFGntawHGww",
  "fee" : 10000000,
  "type" : 104,
  "params" : [ {
    "type" : "string",
    "value" : "value",
    "key" : "data"
  }, {
    "type" : "integer",
    "value" : 500,
    "key" : "length"
  } ],
  "version" : 4,
  "contractVersion" : 3,
  "atomicBadge" : null,
  "sender" : "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "feeAssetId" : null,
  "proofs" : [ "2bpALen4diR7DTFhNQCrZKPueCPds2gFFPxe1KVzQwfRuGaK6QfvtpN8oqaZMsStoEHAa5DrTkKM8AuzHPYyMPVP" ],
  "contractId" : "HSLdKYqLq4LcZpq9LPki8Yv4ZRkFapVyHEYw1vZW2MoG",
  "id" : "GBfibn8VjGmDS9ex4Nd4JNRLvDyvJjj8jLUUcbYwFTCf",
  "timestamp" : 1625732766458,
  "height" : 1028132
}
```


5.3.18 105. ExecutedContract Transaction

Запись результата исполнения смарт-контракта в его стейт. Байтовое представление этой транзакции после ее подписания не должно превышать 150 килобайт.

Транзакция 105 содержит все поля (тело) транзакции 103, 104 или 107 смарт-контракта, результат исполнения которого необходимо записать в его стейт (поле `tx`). Результат исполнения смарт-контракта вносится в его стейт из соответствующих параметров поля `params` транзакции 103 или 104.

Подписание транзакции производится узлом, формирующим блок после отправки запроса на публикацию транзакции.

Структура данных на публикацию транзакции:

Поле	**Тип данных**	**Описание**
type	Byte	Номер транзакции (105)
id	Byte	ID транзакции исполнения контракта
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
password	String	Пароль от ключевой пары в keystore узла – опциональное поле
fee	Long	Комиссия за транзакцию в системном токене
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) – опциональное поле
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
version	Byte	Версия транзакции
tx	Array	Тело транзакции 103 или 104 исполняемого смарт-контракта
results	List[DataEntry[_]]	Список возможных результатов исполнения смарт-контракта
height	Byte	Высота выполнения транзакции

JSON-представления запроса и ответа транзакции 105. ExecutedContract Transaction приведены ниже.

Version 2

Публикация:


```
{
    "type": 105,
    "id": "38GmSVC5s8Sjeybzfe9RQ6p1Mb6ajb8LYJDcep8G8Umj",
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
    "password": "",
    "fee": 500000,
    "timestamp": 1550591780234,
    "proofs": [ "5whBipAWQgFvm3myNZe6GDd9Ky8199C9qNxLBHqDNmVAUJW9gLf7t9LBQDi68CKT57dzmnPJpJkrwKh2HBSwUer6" ],
    "version": 2,
    "tx":
         {
            "type": 103,
            "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
            "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
            "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
            "fee": 500000,
            "timestamp": 1550591678479,
            "proofs": [ "yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
            "version": 2,
            "image": "stateful-increment-contract:latest",
            "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
            "contractName": "stateful-increment-contract",
            "params": [],
            "height": 1619
          },
    "results": [],
    "height": 1619,
    "atomicBadge" : null
}
```


5.3.19 106. DisableContract Transaction

Отключение смарт-контракта. Байтовое представление этой транзакции после ее подписания не должно превышать 150 килобайт.

Подписание транзакции 106 может производиться только пользователем с ролью `contract_developer`.

Структуры данных транзакции:

Подписание:

Поле	Тип данных	Описание
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore узла - опциональное поле
contractId	ByteStr	ID смарт-контракта
fee	Long	Комиссия за транзакцию в системном токене
type	Byte	Номер транзакции (106)
version	Byte	Версия транзакции

Публикация:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (106)
id	Byte	ID транзакции отключения контракта
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
fee	Long	Комиссия за транзакцию в системном токене
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
version	Byte	Версия транзакции
contractId	ByteStr	ID смарт-контракта
height	Byte	Высота выполнения транзакции

JSON-представления запроса и ответа транзакции 106. DisableContract Transaction приведены ниже.

Version 2

Подписание:


```
{
    "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "password": "",
    "contractId": "HKftkVDTcQp6kxdqVYNdzB9d4rhND4YRKxwJV1thMXcr",
    "fee": 1000000,
    "type": 106,
    "version": 2,
}
```


Публикация:


```
{
  "senderPublicKey" : "CgqRPcPnexY533gCh2SSvBXh5bca1qMs7KFGntawHGww",
  "sender" : "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "feeAssetId" : "7QpXWLGuaspzrMsESRaHTgksndq5mcvfbVrqBTuLbxuy",
  "proofs" : [ "3FKPGT8YbLVun5cffZi1sHkgr9JZVxkeN7z2kUqDVLfhB5CwMtCAfyStRz1tpZuriKsR3MaBqNfReGx5sM2qey8i" ],
  "fee" : 1000000,
  "contractId" : "HKftkVDTcQp6kxdqVYNdzB9d4rhND4YRKxwJV1thMXcr",
  "id" : "5hXuHs5HVhZSfek153t76HfW6egmCLdZmi5AeFzYBFN",
  "type" : 106,
  "version" : 2,
  "timestamp" : 1625648619321,
  "height" : 1025992
}
```


Version 3

Подписание:


```
{
    "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "password": "",
    "contractId": "75PumcfCVxzV3v7RAPYQUwCtSpU21hxfaWFhureCRTLM",
    "fee": 1000000,
    "type": 106,
    "version": 3,
  "atomicBadge" : {
    "trustedSender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx"
  }
}
```


Публикация:


```
{
  "senderPublicKey" : "7GiFGcGaEN87ycK8v71Un6b7RUoeKBU4UvUHPYbeHaki",
  "atomicBadge" : {
    "trustedSender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx"
  },
  "sender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
  "feeAssetId" : null,
  "proofs" : [ "22tK24qHhgbTDjtRmR86z3WeLLqLnqPvhUhQrz8ohfbCwQ9nrwmHESuT9aFuwABeBRJ7MfVob1FiJnqg3y2PHLSj" ],
  "fee" : 1000000,
  "contractId" : "75PumcfCVxzV3v7RAPYQUwCtSpU21hxfaWFhureCRTLM",
  "id" : "7opPrLd6x1hATRr9R5oXnEbYjYQzo5cn4Qpkiz12Mw9b",
  "type" : 106,
  "version" : 3,
  "timestamp" : 1619186857911,
  "height" : 861644
}
```

5.3.20 107. UpdateContract Transaction

Обновление кода смарт-контракта. Байтовое представление этой транзакции после ее подписания не должно превышать 150 килобайт.

Подписание транзакции 107, как и обновление смарт-контракта, может производиться только пользователем с ролью `contract_developer`.

Структуры данных транзакции:

Подписание:

Поле	Тип данных	Описание
image	Array[Bytes]	Имя Docker-образа смарт-контракта
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore узла – опциональное поле
fee	Long	Комиссия за транзакцию в системном токене
contractId	ByteStr	ID смарт-контракта
imageHash	Array[Bytes]	Хэш Docker-образа смарт-контракта
type	Byte	Номер транзакции (107)
version	Byte	Версия транзакции
apiVersion	Byte	Версия API для gRPC-методов смарт-контракта (см. раздел «API-инструменты, доступные смарт-контракту» выше).
validationPolicy.type	String	Тип политики валидации смарт-контрактов.

JSON-представления запроса и ответа транзакции 107. UpdateContract Transaction приведены ниже.

Version 2

Подписание:

```
{
 "image" : "vostok-sc/grpc-contract-example:2.2-test-update",
 "sender" : "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
 "password": "",
 "fee" : 100000000,
 "contractId" : "BWzX4mRBEnHKgn3HB78My5DZzDAqnCLWCCNpCuRkZrJA",
 "imageHash" : "075ad1607f193cc6fdb5e85c201f9ca3907c622718d75706bbc2a94a330de5b5",
 "type" : 107,
 "version" : 2
}
```

Публикация:

```
{
  "senderPublicKey" : "CgqRPcPnexY533gCh2SSvBXh5bca1qMs7KFGntawHGww",
  "image" : "vostok-sc/grpc-contract-example:2.2-test-update",
  "fee" : 100000000,
  "imageHash" : "075ad1607f193cc6fdb5e85c201f9ca3907c622718d75706bbc2a94a330de5b5",
  "type" : 107,
  "version" : 2,
  "sender" : "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "feeAssetId" : null,
  "proofs" : [ "RetQwzuWZWxpSNMqwB7k7o6hSm6nhFCc49zKUpwZEedzBYcohj9NVEPwAbKLW9RzRKX168xApV7Nu2qV2jaHAMg" ],
  "contractId" : "BWzX4mRBEnHKgn3HB78My5DZzDAqnCLWCCNpCuRkZrJA",
  "id" : "6oopqcEf4AF943SCAqkBPrghyeQhmwn64TrhtCZbAn3v",
  "timestamp" : 1625649822957,
  "height" : 1026022
}
```

Version 3

Подписание:


```
{
 "image" : "registry.vostokservices.com/vostok-sc/grpc-contract-example:2.2-
   test-update",
 "sender" : "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
 "password": "",
 "fee" : 100000000,
 "contractId" : "HTqdjXUPTHZqGen2KKUkEenTELAqQ8irN58LA8EcP17q",
 "imageHash" : "075ad1607f193cc6fdb5e85c201f9ca3907c622718d75706bbc2a94a330de5b5",
 "type" : 107,
 "version" : 3,
 "atomicBadge" : null
}
```


Публикация:


```
{
  "senderPublicKey" : "7GiFGcGaEN87ycK8v71Un6b7RUoeKBU4UvUHPYbeHaki",
  "image" : "registry.vostokservices.com/vostok-sc/grpc-contract-example:2.2-
   test-update",
  "fee" : 100000000,
  "imageHash" : "075ad1607f193cc6fdb5e85c201f9ca3907c622718d75706bbc2a94a330de5b5",
  "type" : 107,
  "version" : 3,
  "atomicBadge" : null,
  "sender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
  "feeAssetId" : null,
  "proofs" : [ "3ncWfFPqBAdgh65YceCCvF2RhUWWokQc9MsnHk27YLrYmPj9gWgrbRcousymJVA7ARFSz5UJcdW4Sa62FFhR5en3" ],
  "contractId" : "HTqdjXUPTHZqGen2KKUkEenTELAqQ8irN58LA8EcP17q",
  "id" : "B7qjgCa9N6M6FwV63PbLwvtVpFo4bzB5gRZzGjwJpKJV",
  "timestamp" : 1619187337697,
  "height" : 861650
}
```


Version 4

Подписание:


```
{
 "image" : "vostok-sc/grpc_validatable_stateless:0.1",
 "sender" : "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
 "password": "",
 "fee" : 100000000,
 "contractId" : "HSLdKYqLq4LcZpq9LPki8Yv4ZRkFapVyHEYw1vZW2MoG",
 "imageHash" :
   "bd98a7d3e55506ff936d8ea15e170a24d27662edd1b47e4fd20801d10655af8d",
 "type" : 107,
 "version" : 4,
 "atomicBadge" : null
}
```


Публикация:


```
{
  "senderPublicKey" : "CgqRPcPnexY533gCh2SSvBXh5bca1qMs7KFGntawHGww",
  "image" : "vostok-sc/grpc_validatable_stateless:0.1",
  "fee" : 100000000,
  "imageHash" : "bd98a7d3e55506ff936d8ea15e170a24d27662edd1b47e4fd20801d10655af8d",
  "type" : 107,
  "version" : 4,
  "atomicBadge" : null,
  "apiVersion" : "1.0",
  "sender" : "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "feeAssetId" : null,
  "proofs" : [ "fZr9LpqSWbPcUzArSZxFDEuygN62hR63j2Cz1GyTFxPNRrNvVwkDhTDcC8zwRp235gA1gSM8fvPps9mvPTWDQ4p" ],
  "contractId" : "HSLdKYqLq4LcZpq9LPki8Yv4ZRkFapVyHEYw1vZW2MoG",
  "id" : "HWZy7219Nx5oxj2QnK3ReEuZiqsjoULbmfdQz8YysFSz",
  "validationPolicy" : {
    "type" : "any"
  },
  "timestamp" : 1625732772746,
  "height" : 1028132
}
```


В версии 4 данной транзакции настраивается валидация результатов исполнения обновляемого смарт-контракта при помощи поля `validationPolicy.type`.

Варианты политик валидации:

- `any` – сохраняется действующая в сети общая политика валидации: для майнинга обновляемого смарт-контракта майнер подписывает соответствующую транзакцию 105, описанная выше в разделе «105. ExecutedContract Transaction». Также этот параметр устанавливается, если в вашей сети нет ни одного зарегистрированного валидатора.
- `majority` – транзакция считается валидной, если она подтверждена большинством валидаторов: 2/3 от общего числа зарегистрированных адресов с ролью `contract_validator`.
- `majorityWithOneOf(List[Address])` – транзакция считается валидной, если собрано большинство валидаторов, среди которых присутствует хотя бы один из адресов, включенных в список параметра. Адреса, включаемые в список, должны иметь действующую роль `contract_validator`.

Предупреждение: При выборе политики валидации `majorityWithOneOf(List[Address])` заполните список адресов, передача пустого списка запрещена.

5.3.21 110. GenesisRegisterNode Transaction

Регистрация узла в генезис-блоке при старте блокчейна.
Данная транзакция не требует подписания.

Структура данных на публикацию транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (110)
id	Byte	ID транзакции регистрации узла в генезис-блоке
fee	Long	Комиссия за транзакцию в системном токене
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) – опциональное поле
signature	ByteStr	Подпись транзакции (в формате base58)
version	Byte	Версия транзакции
targetPubKey	Byte	Публичный ключ регистрируемого узла
height	Byte	Высота выполнения транзакции

5.3.22 111. RegisterNode Transaction

Регистрация нового узла в блокчейне или его удаление.

Структуры данных транзакции:

Подписание:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**111**)
opType	String	Тип операции: `add` – добавить узел; `remove` – удалить узел
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore узла – опциональное поле
targetPubKey	Byte	Публичный ключ регистрируемого или удаляемого узла
NodeName	Byte	Имя узла
fee	Long	Комиссия за транзакцию в системном токене

Публикация:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**111**)
id	Byte	ID транзакции регистрации узла
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
fee	Long	Комиссия за транзакцию в системном токене
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) – опциональное поле

proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
version	Byte	Версия транзакции
targetPubKey	Byte	Публичный ключ регистрируемого или удаляемого узла
NodeName	Byte	Имя узла
opType	String	Тип операции: `add` – добавить узел; `remove` – удалить узел
height	Byte	Высота выполнения транзакции
password	String	Пароль от ключевой пары в keystore узла – опциональное поле

JSON-представления запроса и ответа транзакции 111. RegisterNode Transaction приведены ниже.

Version 1

Подписание:

```
{
  "type": 111,
  "opType": "add",
  "sender":"3NgSJRdMYu4ZbNpSbyRNZLJDX926W7e1EKQ",
  "password": "",
  "targetPubKey": "
   3CHxJjoSzJam19Pkk2BdEzZspT9nPS68b8RjyCSZoDe7DokgLToLDUXTShhCV5bv2UbyMPU5eHYj1i7nzD
   NuD2pM",
  "nodeName": "GATEs node",
  "fee": 1100000,
}
```

Публикация:

```
{
    "senderPublicKey" : "FWz5gZ2w2ZxXbKEiwhgEcZKT4we1Wneh9XqmCeGPsA4r",
    "nodeName" : "GATEs node",
    "fee" : 1100000,
    "opType" : "add",
    "type" : 111,
    "version" : 1,
    "target" : "3NtieMGjVAH1nDsvnSEJ37BSW3hpJV2CneY",
    "sender" : "3NgSJRdMYu4ZbNpSbyRNZLJDX926W7e1EKQ",
    "proofs" : [ "FHEexr8MqMCkdqaVRrfxv7dnQFwo1VQxQFb4rW2VKh1NkuAhjhtzftKybBQCVbpKcCD1ZTRhwATpwERF9re2Viz" ],
    "id" : "6WnDGkBDeSjg5y6QqVdy3BFHUy5nnr4QsxZCeNXZtZoq",
    "targetPubKey" : "6caEKC1UBgRvgAe9A7L5PWcrawrnEZGxtsXynGESwSj7",
    "timestamp" : 1619078302988,
    "height" : 858895
}
```

5.3.23 112. CreatePolicy Transaction

Создание группы доступа к конфиденциальным данным из указанных адресов.

Структуры данных транзакции:

Подписание:

Поле	**Тип данных**	**Описание**
sender	ByteStr	Адрес отправителя транзакции
policyName	String	Имя создаваемой группы доступа
password	String	Пароль от ключевой пары в keystore узла - опциональное поле
recipients	Array[Bytes]	Массив адресов участников группы доступа к конфиденциальным данным через запятую
fee	Long	Комиссия за транзакцию в системном токене
description	Array[byte]	Произвольное описание транзакции (в формате base58)
owners	Array[Bytes]	Массив адресов-администраторов группы доступа через запятую: администраторы имеют право изменять группу доступа
type	Byte	Номер транзакции (**112**)
version	Byte	Версия транзакции

Публикация:

Поле	**Тип данных**	**Описание**
type	Byte	Номер транзакции (**112**)
id	Byte	ID транзакции создания группы доступа
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
policyName	String	Имя создаваемой группы доступа
recipients	Array[Bytes]	Массив адресов участников группы доступа к конфиденциальным данным через запятую
owners	Array[Bytes]	Массив адресов-администраторов группы доступа через запятую: администраторы имеют право изменять группу доступа
fee	Long	Комиссия за транзакцию в системном токене
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) опциональное поле
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
height	Byte	Высота выполнения транзакции
description	Array[byte]	Произвольное описание транзакции (в формате base58)
version	Byte	Версия транзакции

JSON-представления запроса и ответа транзакции 112. CreatePolicy Transaction приведены ниже.

Version 2

Подписание:

```
{
    "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,
    "version": 2,
}
```

Публикация:

```
{
    "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,
    "version": 2,
}
```

Version 3

Подписание:

```
{
    "sender": "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
    "policyName": "Policy_v3_for_demo_txs",
    "password":"sfgKYBFCF@#$fsdf()%",
    "recipients" : [ "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
   "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF", "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
   "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d", "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn"
   ],
    "fee": 100000000,
    "description": "",
    "owners" : [ "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
   "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF", "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
   "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d", "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn"
   ],
    "type": 112,
    "version": 3
}
```

Публикация:

```
{
    "senderPublicKey" : "7GiFGcGaEN87ycK8v71Un6b7RUoeKBU4UvUHPYbeHaki",
    "policyName" : "Policy_v3_for_demo_txs",
    "fee" : 100000000,
    "description" : "",
    "owners" : [ "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
   "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF", "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
   "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d", "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn"
   ],
    "type" : 112,
    "version" : 3,
    "atomicBadge" : null,
    "sender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
    "feeAssetId" : null,
    "proofs" : [
   "4NccZyPCgchDjeMdMmFKu7kxyU8AFF4e9cWaPFTQVQyYU1ZCCu3QmtmkfJkrDpDwGs4eJhYUVh5Tn
   wYvjZYKPhLp" ],
    "recipients" : [ "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
   "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF", "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
   "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d", "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn"
   ],
    "id" : "5aYtmTr1AYYG8BrYvTTSqKzfJZxfgorx1BLGVwSAhwrz",
    "timestamp" : 1619186864092,
    "height" : 861637
}
```

5.3.24 113. UpdatePolicy Transaction

Изменение группы доступа к конфиденциальным данным.

Структуры данных транзакции:

Подписание:

Поле	**Тип данных**	**Описание**
policyId	String	Идентификатор создаваемой группы доступа
password	String	Пароль от ключевой пары в keystore узла – опциональное поле
sender	ByteStr	Адрес отправителя транзакции
recipients	Array[Bytes]	Массив адресов участников группы доступа к конфиденциальным данным через запятую
fee	Long	Комиссия за транзакцию в системном токене
opType	String	Тип операции: `add` – добавить участников; `remove` – удалить участников
owners	Array[Bytes]	Массив адресов-администраторов группы доступа через запятую: администраторы имеют право изменять группу доступа
type	Byte	Номер транзакции (**113**)
version	Byte	Версия транзакции

Публикация:

Поле	**Тип данных**	**Описание**
type	Byte	Номер транзакции (**113**)
id	Byte	ID транзакции изменения группы доступа
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
policyId	String	Идентификатор создаваемой группы доступа
recipients	Array[Bytes]	Массив адресов для добавления или удаления участников группы доступа к конфиденциальным данным через запятую
owners	Array[Bytes]	Массив адресов-администраторов группы доступа через запятую: администраторы имеют право изменять группу доступа
fee	Long	Комиссия за транзакцию в системном токене
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) – опциональное поле
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
height	Byte	Высота выполнения транзакции
opType	String	Тип операции: `add` – добавить роль; `remove` – отозвать роль
description	Array[byte]	Произвольное описание транзакции (в формате base58)
version	Byte	Версия транзакции

JSON-представления запроса и ответа транзакции 113. UpdatePolicy Transaction приведеныниже.

Version 2

Подписание:

```
{
  "policyId": "UkvoboGXiwWpASr1GLG9M1MUbhrEMo4NBS7kquxVMw5",
  "password":"sfgKYBFCF@#$fsdf()*%",
  "sender": "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
  "recipients" : [ "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF" ],
   "fee": 50000000,
   "opType": "remove",
  "owners" : [ "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF" ],
  "type": 113,
  "version": 2
 }
```

Публикация:

```
{
  "senderPublicKey" : "7GiFGcGaEN87ycK8v71Un6b7RUoeKBU4UvUHPYbeHaki",
  "fee" : 50000000,
  "opType" : "remove",
  "owners" : [ "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF" ],
  "type" : 113,
  "version" : 2,
  "policyId" : "UkvoboGXiwWpASr1GLG9M1MUbhrEMo4NBS7kquxVMw5",
  "sender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
  "feeAssetId" : null,
  "proofs" : [ "2CKd57kU3wbxdrHxMPNbrWHptnf5ZcydYjqxMPk46miMcUUAxgFGXcy621cjYFXC3vjpKNNrB2QcgtKe1Yx9TcLY" ],
  "recipients" : [ "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF" ],
  "id" : "6o4azRwzmMg9SqWUq6rv6GAe5gzTYJvE5ek1v9VM3Mb",
  "timestamp" : 1619004195630,
  "height" : 856970
}
```

Version 3

Подписание:

```
{
  "policyId": "5aYtmTr1AYYG8BrYvTTSqKzfJZxfgorx1BLGVwSAhwrz",
  "password":"sfgKYBFCF@#$fsdf()*%",
  "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "recipients" : [ "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF" ],
   "fee": 50000000,
   "opType": "remove",
  "owners" : [ "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF" ],
  "type": 113,
  "version": 3
 }
```

Публикация:

```
{
  "senderPublicKey" : "7GiFGcGaEN87ycK8v71Un6b7RUoeKBU4UvUHPYbeHaki",
  "fee" : 50000000,
  "opType" : "remove",
  "owners" : [ "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF" ],
  "type" : 113,
  "version" : 3,
  "atomicBadge" : null,
  "policyId" : "5aYtmTr1AYYG8BrYvTTSqKzfJZxfgorx1BLGVwSAhwrz",
  "sender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
  "feeAssetId" : null,
  "proofs" : [ "2QMGoz6rycNsDLhN3mDce2mqGRQQ8r26vDDw551pnYcAecpFBDA8j38FVqDjLTGuFHs6ScX32fsGcaemmptpCFHk" ],
  "recipients" : [ "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF" ],
  "id" : "Hwqf8LgpQfEcUYX9nMNG8uU2Cw1xSuGFqYxmuACpvU1L",
  "timestamp" : 1619187450552,
  "height" : 861653
}
```

5.3.25 114. PolicyDataHash Transaction

Отправка хэша конфиденциальных данных в сеть. Эта транзакция создается автоматически при отправке в сеть конфиденциальных данных при помощи REST API метода **POST /privacy/sendData**.

Данная транзакция не требует подписания.

Структура данных на публикацию транзакции:

Поле	**Тип данных**	**Описание**
type	Byte	Номер транзакции (**114**)
id	Byte	ID транзакции
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
policyId	String	Имя создаваемой группы доступа
dataHash	String	Хэш конфиденциальных данных для отправки
fee	Long	Комиссия за транзакцию в системном токене
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) – опциональное поле
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
height	Byte	Высота выполнения транзакции
version	Byte	Версия транзакции

5.3.26 120. Atomic Transaction

Атомарная транзакция помещает в контейнер другие транзакции для их атомарного выполнения. Транзакция этого типа исполняется полностью (ни одна из включенных транзакций не отклоняется) или не исполняется в принципе.

Поддерживается включение 2 и более транзакций следующих типов:

- 4. Transfer Transaction, версия 3
- 102. Permission Transaction, версия 2
- 103. CreateContract Transaction, версия 3
- 104. CallContract Transaction, версия 4
- 106. DisableContract Transaction, версия 3
- 107. UpdateContract Transaction, версия 3
- 112. CreatePolicy Transaction, версия 3
- 113. UpdatePolicy Transaction, версия 3
- 114. PolicyDataHash Transaction, версия 3

Атомарная транзакция сама по себе не требует комиссии: общая сумма складывается из комиссий за транзакции, помещенные в атомарную транзакцию.

Структуры данных транзакции:

Подписание:

Поле	**Тип данных**	**Описание**
type	Byte	Номер транзакции (**120**)
sender	ByteStr	Адрес отправителя транзакции
transactions	Array	Полные тела включаемых транзакций
password	String	Пароль от ключевой пары в keystore узла – опциональное поле
fee	Long	Комиссия за транзакцию в системном токене
version	Byte	Версия транзакции

Публикация:

Поле	**Тип данных**	**Описание**
type	Byte	Номер транзакции (**114**)
id	Byte	ID транзакции
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
fee	Long	Комиссия за транзакцию в системном токене
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) – опциональное поле
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
height	Byte	Высота выполнения транзакции
transactions	Array	Полные тела включаемых транзакций
miner	String	Публичный ключ майнера блока; заполняется в ходе раунда майнинга
password	String	Пароль от ключевой пары в keystore узла – опциональное поле
version	Byte	Версия транзакции

JSON-представления запроса и ответа транзакции 120. Atomic Transaction приведены ниже.

Version 1

Подписание:

```
{
    "sender": sender_0,
    "transactions": [
        signed_transfer_tx,
        signed_transfer_tx2
    ],
    "type": 120,
    "version": 1,
    "password":"lskjbJJk$%^#298",
    "fee": 0,
}
```

Публикация:

```
{
  "sender": "3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP",
  "transactions": [
        signed_transfer_tx,
        signed_transfer_tx2
  ],
  "type": 120,
  "version": 1,
}
```

5.4 Актуальные версии транзакций

При отправке транзакций в сеть рекомендуется использовать актуальные версии транзакций. Версия транзакции указывается в поле `version` при подписании и отправке.

Номер транзакции	**Название транзакции**	**Актуальная версия**
1	Genesis Transaction	Без версии
3	Issue Transaction	2
4	Transfer Transaction	3
5	Reissue Transaction	2
6	Burn Transaction	2
8	Lease Transaction	2
9	Lease Cancel Transaction	2
10	Create Alias Transaction	3
11	Mass Transfer Transaction	2
12	Data Transaction	2
13	Set Script Transaction	1
14	Sponsorship Transaction	1
15	Set Asset Script Transaction	1
101	Genesis Permission Transaction	Без версии
102	Permission Transaction	2
103	Create Contract Transaction	4
104	Call Contract Transaction	4
105	Executed Contract Transaction	2
106	Disable Contract Transaction	3
107	Update Contract Transaction	4
110	Genesis Resgister Node Transaction	1
111	Register Node Transaction	1
112	Create Policy Transaction	3
113	Update Policy Transaction	3
114	Policy Data Hash Transaction	3
120	Atomic Transaction	1

5.5 Атомарные транзакции

Платформа «Конфидент» поддерживает выполнение атомарных операций. Атомарные операции состоят из нескольких действий, при невыполнении одного из действий все остальные также не выполняются. Для этого в системе существует транзакция 120 AtomicTransaction (см. также раздел «120. Atomic Transaction» выше), представляющая собой контейнер, в который помещаются две и более подписанные транзакции.

Поддерживается включение 2 и более транзакций следующих типов:

- 4. Transfer Transaction, версия 3;
- 102. Permission Transaction, версия 2;
- 103. CreateContract Transaction, версия 3;
- 104. CallContract Transaction, версия 4;
- 105. ExecutedContract Transaction, версии 1 и 2;
- 106. DisableContract Transaction, версия 3;
- 107. UpdateContract Transaction, версия 3;
- 112. CreatePolicy Transaction, версия 3;
- 113. UpdatePolicy Transaction, версия 3;
- 114. PolicyDataHash Transaction, версия 3.

Ключевым отличием версий транзакций, которые поддерживаются атомарной транзакцией, является наличие поля-метки `atomicBadge`. Это поле содержит доверенный адрес отправителя транзакции `trustedSender` для добавления в контейнер транзакции 120. Если адрес отправителя не указывается, отправителем становится адрес, с которого в блокчейн отправляется транзакция 120.

5.5.1 Обработка атомарной транзакции

Атомарная транзакция имеет две подписи. Первым транзакцию подписывает отправитель для её успешной отправки в сеть. Вторая подпись формируется майнером и необходима для добавления транзакции в блокчейн. При добавлении атомарной транзакции в UTX-пул, проверяется её подпись, а также подписи всех транзакций, входящих в контейнер.

Валидация таких транзакций выполняется по следующим правилам:

- Количество транзакций должно быть больше одной.
- Все транзакции должны иметь разные идентификаторы.
- Список транзакций должен содержать только поддерживаемые типы транзакций.

Вкладывать одну атомарную транзакцию в другую не допускается.

Внутри атомарной транзакции, отправляемой в UTX пул, не должно быть исполненных (executed) транзакций, и поле miner должно быть пустым. Это поле заполняется при передаче атомарной транзакции в блок.

Внутри атомарной транзакции, попавшей в блок, не должно быть исполняемых (executable) транзакций.

После исполнения атомарной транзакции в блок попадает ее «копия», сформированная по следующим правилам:

- Поле `miner` не участвует в формировании подписи транзакции и заполняется публичным ключом майнера блока.
- Майнер блока формирует массив `proofs`, источником которого служат идентификаторы транзакций, входящих в атомарную транзакцию. При включении в блок, атомарная транзакция имеет 2 подписи – подпись исходной транзакции и подпись майнера.
- Если в списке присутствуют executable транзакции, они заменяются на executed транзакции. При валидации атомарной транзакции в составе блока проверяются обе подписи.

5.5.2 Создание атомарной транзакции

Для создания атомарной транзакции необходим доступ к REST API узла. Подробнее о REST API см. раздел «Методы REST API» выше.

1. Пользователь подбирает из списка поддерживаемых транзакций те транзакции, которые должны выполняться как атомарная операция.
2. Затем корректно заполняет поля всех транзакций и подписывает их.
3. Далее пользователь заполняет поле `transactions` атомарной транзакции данными подписанных, но не отправленных в блокчейн транзакций.
4. После внесения всех данных о транзакциях пользователь подписывает и отправляет в блокчейн готовую атомарную транзакцию.

Структуры данных для подписания и отправки атомарной транзакции приведены в списке транзакций.

Внимание. Если вы создаёте атомарную транзакцию с включением 114 транзакции (см. раздел «114. PolicyDataHash Transaction»), то при её подписании установите значение параметра `broadcast = false`.

6 Обмен конфиденциальными данными

Платформа «Конфидент» позволяет ограничить доступ к определенным данным, размещаемым в блокчейне. Для этого пользователи объединяются в группы, получающие доступ к конфиденциальным данным.

Важно: Обмен конфиденциальными данными (privacy) доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

6.1 Создание группы доступа

Создать группу доступа к конфиденциальным данным может любой участник сети. Прежде, чем создавать группу доступа, определитесь со списком участников, которые будут в нее входить. Затем подпишите и отправьте транзакцию 112 CreatePolicy (подробнее о транзакции см. раздел «112. CreatePolicy Transaction»):

1. В поле `recipients` впишите через запятую адреса участников, которые получат доступ к конфиденциальным данным.
2. В поле `owners` добавьте через запятую адреса участников группы, которым будут предоставлены права администраторов. Администраторы группы доступа, помимо доступа к конфиденциальным данным, смогут изменять состав группы доступа.

При отправке транзакции вы получите идентификатор созданной группы доступа (`policyId`). Он потребуется при изменении состава ее участников.

После отправки транзакции в блокчейн доступ к отправляемым в сеть конфиденциальным данным получат все участники, зарегистрированные в созданной группе доступа. Как создатель транзакции, вы сможете изменять ее состав, как и участники, добавленные в поле `owners`.

6.2 Изменение группы доступа

Изменять состав группы доступа могут только ее участники, добавленные в поле `owners` при создании группы, а также сам ее создатель – владельцы группы доступа к конфиденциальным данным.

Для изменения состава группы доступа подпишите и отправьте транзакцию 113 UpdatePolicy (подробнее о транзакции см. раздел «113. UpdatePolicy Transaction»):

1. В поле `policyId` введите идентификатор изменяемой группы доступа.
2. В поле `opType` введите действие, которое необходимо произвести с группой:
   - `add` – добавить участников;
   - `remove` – удалить участников.
3. Если вы хотите добавить или удалить участников группы доступа, впишите их публичные ключи в поле `recipients`.
4. Для добавления или удаления владельцев группы доступа впишите их публичные ключи в поле `owners`.

Информация о группе доступа обновляется после отправки транзакции в блокчейн.

6.3 Отправка конфиденциальных данных в сеть

Для отправки конфиденциальных данных в сеть предусмотрены следующие gRPC методы, подробно описанные выше в разделе «Отправка в блокчейн конфиденциальных данных»:

- SendData,
- SendLargeData,

а также REST API методы, подробно описанные выше в разделе «REST API: обмен конфиденциальными данными и получение информации о группах доступа»:

- POST /privacy/sendData,
- POST /privacy/sendDataV2,
- POST /privacy/sendLargeData.

С помощью методов POST /privacy/sendData и POST /privacy/sendDataV2 вы можете отправить данные размером до 20 мегабайт, с помощью метода POST /privacy/sendLargeData – данные размером не менее 20 мегабайт.

Эти методы требуют авторизации.

7 Управление ролями участников

Описание всех ролей блокчейн-платформы приведено в разделе «6.1 Описание ролей участников сети» документа «643.19172778.2019662198-01 95. «Блокчейн-платформа Конфидент» Версия 1.9. Правила пользования»; этот документ поставляется вместе с дистрибутивом платформы. Роли могут быть произвольно скомбинированы для любого адреса, отдельные роли могут быть отозваны в любой момент.

Для управления ролями участников предусмотрена транзакция 102 (подробнее см. раздел «102 Permission Transaction» выше), которая может быть подписана при помощи метода `sign` REST API узла (подробнее см. раздел «Подписание и отправка транзакций» выше) и отправлена при помощи соответствующего метода gRPC или REST API. Отправлять транзакцию 102 в блокчейн может только участник с ролью `permissioner`.

Важно: Метод sign доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы: настройка криптографии» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы.

Вне зависимости от применяемого метода отправки, транзакция включает следующие поля:

- `type` – тип транзакции для управления полномочиями участников (`type = 102`);
- `sender` – адрес участника с полномочиями на отправку транзакции 102 (ролью permissioner);
- `password` – Пароль от ключевой пары в keystore узла, опциональное поле;
- `proofs` – подпись транзакции;
- `target` – адрес участника, для которого требуется установить или удалить полномочия;
- `role` – полномочия участника, которые требуется установить или удалить;
- `opType` – тип операции: `add` (добавить роль) или `remove` (удалить роль);
- `dueTimestamp` – дата действия permission в формате Unix Timestamp (в миллисекундах), опциональное поле.

Полученный ответ метода `sign` передается методу `broadcast` gRPC или REST API узла.

8 Подключение и удаление узлов (нод)

В частной сети подключение и удаление новых участников выполняется после ручной конфигурации и старта первого узла системным администратором (см. документ «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы).

8.1 Подключение нового узла к частной сети

Для подключения нового узла выполните следующее:

1. Настройте узел в соответствии с указаниями, приведенными в разделе «Развертывание платформы в частной сети» документа «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора».
2. Передайте публичный ключ нового узла и его описание администратору вашей сети.
3. Администратор сети (узел с ролью `connection-manager`) использует полученные публичный ключ и описание узла при создании транзакции 111 RegisterNode (подробнее см. раздел «111 RegisterNode Transaction» выше). Для регистрации узла в параметре `opType`, определяющем тип совершаемого действия, указывается `add` (добавление нового узла).
4. Транзакция 111 попадает в блок, а затем – в стейты узлов участников сети. В дальнейшем каждый участник сети обязательно хранит публичный ключ и адрес нового узла.
5. При необходимости администратор сети может добавить новому узлу дополнительные роли при помощи транзакции 102 (подробнее см. раздел «102. Permission Transaction» выше). Подробнее о назначении ролей участников см. раздел «Управление ролями участников» выше.
6. Запустите новый узел.

8.2 Удаление узла из частной сети

Для удаления узла из сети администратор сети отправляет в блокчейн транзакцию 111 RegisterNode (подробнее см. раздел «111. RegisterNode Transaction» выше). В ней он указывает публичный ключ удаляемого узла и параметр `"opType": "remove"` (удаление узла из сети).

После публикации транзакции в блокчейн данные узла удаляются из стейтов всех участников.

9 Запуск узла (ноды) с созданным снимком данных

Для изменения параметров приватного блокчейна без потери сохраненных в нем данных в Платформе «Конфидент» предусмотрен механизм создания снимка данных (снэпшота).

Важно: Перед созданием снимка данных администратор должен зафиксировать контрольные суммы, а при развертывании сверить их.

Настройка механизма создания снимка данных выполняется в конфигурационном файле узла (см. раздел «Тонкая настройка платформы: настройка механизма создания снимка данных» в документе «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»; этот документ поставляется вместе с дистрибутивом платформы).

После создания снимка данных в приватном блокчейне вы, как администратор сети, можете изменить его параметры и перезапустить его с использованием данных, сохраненных в снимке.

Для этого выполните следующие действия:

1. При помощи метода `GET /snapshot/status` убедитесь, что снимок данных был получен вашим узлом и успешно верифицирован;
2. При помощи метода `GET /snapshot/genesis-config` запросите конфигурацию нового genesis-блока и сохраните ее;
3. Методом `POST /snapshot/swap-state` замените текущий стейт сети на снимок данных и дождитесь успешного ответа;
4. Подготовьте конфигурационные файлы узла для перезапуска:
   - измените параметры генезис-блока на полученные в пункте 2;
   - отключите механизм создания снимка данных: `node.consensual-snapshot.enable = no`;
   - при необходимости, измените параметры секции `blockchain` конфигурационного файла узла;

   конфигурационный файл узла описан в документе «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора».
5. Перезапустите узел.

После перезапуска узла генерируется новый genesis-блок сети. Сеть запускается с обновленными параметрами и данными, записанными в снимке данных.

9.1 Механизм создания снимка данных

Механизм создания снимка данных – это вспомогательный механизм Платформы «Конфидент», который позволяет сохранить данные работающей блокчейн-сети для последующего изменения параметров конфигурации сети и ее запуска с сохраненными данными.

Механизм создания снимка данных позволяет изменять параметры конфигурации блокчейн-сети без потери данных. Процесс изменения параметров конфигурации сети при помощи снимка данных называется миграцией.

Снимок данных включает следующие данные:

- стейты адресов сети: балансы, роли в сети, ключи;
- стейты смарт-контрактов, загруженных в сеть: данные, полученные в результате исполнения смарт-контрактов и прикрепленные к ним при помощи транзакций 105 (подробнее см. раздел «105. ExecutedContract Transaction» выше);
- данные майнеров прошедших раундов;
- данные групп доступа к конфиденциальным данным (подробнее см. раздел «Обмен конфиденциальными данными» выше).

В снимке данных не сохраняется история транзакций, банов и блоков сети.

При выполнении миграции снимок данных становится начальным стейтом блокчейн-сети с новыми параметрами, сама сеть перезапускается с формированием нового генезис-блока.

Механизм создания снимка данных включается и настраивается в секции `node.consensual-snapshot` конфигурационного файла узла (подробнее см. раздел «Тонкая настройка платформы: настройка механизма создания снимка данных» в документе «643.19172778.2019662198-01 92. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство администратора»).