Реализация Lykke API abstraction layer для RiaBlocks

# Введение

Решение выполнено в рамках участия в конкурсе <https://streams.lykke.com/Project/ProjectDetails/raiblocks-blockchain-integration-api> в соответствии с требованиями <https://docs.google.com/document/d/1KVd-2tg-Ze5-b3kFYh1GUdGn9jvoo7HFO3wH_knpd3U>

# Состав решения

|  |  |
| --- | --- |
| **Компонент** | **Адрес исходников** |
| Blockchain.API | <https://github.com/artem-kruglov/Raiblocks.Api/tree/dev> |
| Blockchain.SignService | <https://github.com/artem-kruglov/Raiblocks.Sign/tree/dev> |
| Тестовая приватная нода RiaBlocks | <https://github.com/artem-kruglov/raiblocks/tree/testnet> |
| Дoкументация | Настоящий [документ](https://github.com/artem-kruglov/Raiblocks.Api/blob/dev/%D0%A0%D0%B5%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F%20Lykke%20API%20abstraction%20layer%20%D0%B4%D0%BB%D1%8F%20RiaBlocks.docx) и описания в исходном коде |

# Особенности сети RiaBlocks

|  |  |
| --- | --- |
| **Особенность сети** | **Влияние на реализацию API** |
| Перевод средств с одного кошелька на другой требует создания двух транзакций: send-транзакция, подписанная ключом отправителя, и receive-транзакция, подписанная ключом получателя. Send-транзакция списывает средства с аккаунта отправителя, на счету получателя эти средства находятся в статусе pending. После формирования Send-Транзакции получатель может узнать о необходимости подтверждения получения средств, и получить их, сформировав receive-транзакцию на основе соответствующей send-транзакции. После формирования send-транзакции отправитель может формировать другие транзакции независимости от подтверждений получения средств получателями. | Метод transaction/single реализует только send-транзакцию.  В текущей логике API не предусмотрен метод для формирования receive-Транзакций. Его можно формировать самому API в виде некоторого джоба, но API-сервис не имеет доступа к ключам, а значит не может сам сформировать receive-транзакцию.  Нода может сама формировать receive-транзакции, если на ней создан кошелёк (wallet) и в него помещены приватные ключи соответствующих аккаунтов. |
| Одна транзакция – это один блок. Понятие блока и транзакции в сети riablocks совпадают. | Операции с мультивходами и мультвыходами не поддерживаются сетью |
| Отсутствие комиссии | Метод PUT для изменения комиссии не поддерживается. unsign -транзакция и sign-транзакции с одним operationId не могут иметь разный amount. |
| Мельчайшая неделимая единица сети называется raw. Актив, торгуемый на биржах, обозначается XRB и равен 10^30 raw (Источник - <https://raiblocks.net/page/faq>). Публичный эксплорер (<https://raiblocks.net/page/representatives>) отображает баланс в XBR с 6 знаками после запятой. | Параметр accurance для актива равен 6, минимальная единица для перевода с помощью API – 10^24 raw, т.е. 1 xbr (1 XBR = 1 Mxbr = 10^6 xbr). |
| Блок сети не содержим метку времени в целях уменьшения размера блока | В истории транзакции отображается время, когда API стало известно об этом транзакции: либо время получения информации из сети, либо время самостоятельного проведения транзакций (для транзакций с operationId). |
| В сети используется PoW для защиты от спама: нода после формирования транзакции должна выполнить работу и включить информацию о ней (“work”) в следующую транзакцию. Work считается на основе хэша предыдущей транзакции. При офлайн-подписании транзакции work должен быть уже вычислен. | При вызове метода broadcast необходимо соблюдать порядок, в котором транзакции генерировались. |
| Отсутствие публичной тестовой сети | Мы вручную изменили исходники ноды, скомпилировали, и построили тестовую сеть с кастомным генезисом. |

# Raiblocks SignService

Для создания сервиса генерации ключей и подпоясывания транзакций был использован оригинальный код raiblocks. Для доступа к необходимым функциям была создана прослойка, которая позволяет вызвать соответствующие функции из кода C#.

Код: <https://github.com/artem-kruglov/raiblocks/tree/sign_service>

## Детали реализации

Прослойка реализована в виде двух функций block\_create\_c и key\_create, описанных в файле sign\_service.cpp. Эти функции являются облегченными вариантами соответствующих RPC методов block\_create и key\_create. Ввиду ограничения на использование кода ноды raiblocks метод block\_create\_c не производит самостоятельное вычисление work в случае его отсутствия, а предполагает, что он уже присутствует в транзакции.

## Обновление кодовой базы raiblocks

Для обновления кодовой базы необходимо слить изменения в ветку sign\_service и выполнить построение цели sign\_service. При сохранении совместимости в интерфейсе работы с блоками дополнительных изменений в коде не требуется. Полученную динамическую библиотеку libsign\_service необходимо заменить в экземпляре SignService.

**Примечание: библиотека,** построенная для платформ на основе Linux имеет расширение **.so,** однако для универсальности кода импортируется под расширением **dll** для всех платформ**:** [DllImport("libsign\_service.dll")]

# Тестовая сеть

Для создания частной тестовой сети был сгенерирован генезис блок. Модифицированный код клиента тестовой сети расположен в ветке testnet:

<https://github.com/artem-kruglov/raiblocks/tree/testnet>

в приложении находятся два файла rai\_node1 и rai\_node2 – скомпилированные клиенты, которые считывают настройки из директорий RaiBlocks1 и RaiBlocks2, соответственно. Это необходимо для возможности простого запуска сети из двух нод на одной машине.

Обсуждение на reddit: <https://www.reddit.com/r/RaiBlocks/comments/7m47nh/how_to_run_a_private_raiblocks_testnet/>

# Контакты

Email: [artem.kruglov.ru@gmail.com](mailto:artem.kruglov.ru@gmail.com)

telegram: @artem\_kruglov