INNOPOLIS UNIVERSITY

# Система распределенного реестра «InnoChain»

Общее руководство

Листов 13

2021

## Аннотация

В данном документе приводится общее руководство для взаимодействия с системой распределенного реестра «InnoChain» 1.0.0.

Система распределенного реестра «InnoChain» (далее – система «InnoChain») – облачное решение для хранения и управления общими данными, которые принадлежат различным контрагентам, предоставляя при этом гарантии целостности, доступности и неизменяемости данных.

Общее руководство системы «InnoChain» состоит из 5 частей: назначение и условие применения программы, характеристика программы, обращение к программе, входные выходные данные, сообщения.

«Назначение и условия применения программы» содержит информацию по назначению программы, ее функциям, техническим характеристикам и программным средствам. «Характеристика программы» включает в себя описание основных характеристик и особенностей программы. «Обращение к программе» содержит описание процедуры вызова API системы. «Входные выходные данные» содержит описание входных и выходных данных. «Сообщения» содержит возможные проблемы при взаимодействии с API системы.

**СОДЕРЖАНИЕ**

# 1. Назначение и условия применения программы

1.1. Назначение системы.

Система «InnoChain» предназначена для использования в качестве облачного сервиса для распределенного хранения данных с гарантией целостности, доступности и неизменяемости. Пользователи системы получают доступ к данным через разработанные программистами внешние приложения, взаимодействующие с системой через API. Внешние программы могут быть узконаправленными, например, взаимодействовать только с определенным смарт-контрактом в системе, либо предоставлять пользователю возможность хранить произвольные данные в системе. Стоит отметить, что пользователь имеет доступ только к данным, относящимся к его аккаунту, используя сертификат, который создается или импортируется во внешнее приложение. Таким образом, используя API, внешние приложения позволяют пользователям просматривать и изменять данные аккаунтов, к которым они имеют доступ в системе «InnoChain».

1.2. Условия применения системы.

Для использования API необходимо иметь во внешней программе реализацию следующих компонент:

- Компонент, отвечающий за отправку/получение данных по протоколу https;
- Компонент, реализующий протокол JSON RPC 2.0;
- Компонент, отвечающий за кодирование/декодирование данных формата SCALE;
- Компонент, реализующий хэш функцию ГОСТ 34.11;
- Компонент, реализующий электронно-цифровую подпись ГОСТ 34.10;

В качестве примера реализации перечисленных компонент рекомендуется использовать исходный код мобильного приложения для iOS, которое предназначено для ознакомительных целей и не относится напрямую к исходному коду ПО из данного документа.

## 2. Характеристики программы

Система «InnoChain» опирается на понятие аккаунта с точки зрения хранения данных. Аккаунты подразделяются на 3 типа: аккаунт администратора, аккаунт пользователя и аккаунт смарт-контракта.

Каждый аккаунт пользователя или администратора содержит набор открытых ключей, с помощью которых владелец аккаунта (внешняя программа, имеющая доступ к соответствующему набору закрытых ключей) может изменять данные аккаунта. Аккаунт администратора отличается от аккаунта пользователя тем, что может блокировать/разблокировать доступ к системе распределенного реестра аккаунтам пользователя.

Аккаунт смарт-контракта предназначен для хранения данных и логики работы некоторого смарт-контракта, таким образом, позволяя не просто хранить данные, но и исполнять Тьюринг-полные алгоритмы при вызове его функций внешними программами. На данный момент все типы смарт-контрактов должны компилироваться вместе системой «InnoChain» в связи с требованием их формальной верификации, что минимизирует риск возникновения ошибок в логике их работы.

Для изменения данных аккаунта или вызова функций смарт-контракта (которые приводят к изменениям данных) используется механизм транзакций, требующий включения в запрос электронно-цифровой подписи, которая подтверждает наличие у аккаунта прав на совершение операции.

Получение данных и отправка транзакций осуществляется внешними программами через API системы «InnoChain», которое обладает следующими характеристиками:

- Предполагается, что в качестве алгоритма проверки и формирования электронно-цифровой подписи (ЭЦП) используется ГОСТ 34.10 512 бит, в качестве хэш функции - Стрибог 256 бит.
- В качестве формата бинарного представления данных (сериализация) при вызовах API используется JSON. Для формирования ЭЦП структур данных предполагается использование алгоритма сериализации SCALE (раздел 6 «Проверка программы с помощью мобильного приложения»).

## 3. Обращение к программе

Для обращения к системе «InnoChain» через API необходимо запросить у системного администратора IP и порт, на котором развернута система. Помимо этого, можно воспользоваться разделом 3 документа «Инструкция по установке» и развернуть систему самостоятельно. В качестве примера использования API можно воспользоваться исходным кодом iOS приложения (раздел 6 «Проверка программы с помощью мобильного приложения»). Стоит отметить, что iOS приложения имеет исключительно ознакомительные цели, а его исходный не относится к исходному коду ПО из данного документа.

Внешняя программа может обратиться к API с одним из следующих запросов:

- отправить транзакцию
- получить транзакцию
- получить данные аккаунта

Запрос к API должен быть сформирован в формате JSON RPC 2.0 и отправлен на адрес, ассоциированный с API.

Для формирования запроса или обработки ответа понадобиться структура аккаунта, которая имеет следующий вид:

```
Account
    accountId: GUID;
    publicKeys: List<PublicKey>;
    type: UInt8;
    nonce: UInt32;
    isConfirmed: Bool;
    code: Optional<Bytes>;
    storage: [Key: Value];
```

accountId - уникальный идентификатор аккаунта в формате GUID.

publicKeys - список открытых ключей аккаунта. Каждая транзакция от имени аккаунта должна быть подписана соответствующим множеством закрытых ключей.

type - тип аккаунта:

- смарт-контракт (0)
- пользовательский (1)
- мастер (2)

Создание пользовательского аккаунта должно подтверждаться соответствующей транзакцией от имени мастер-аккаунта при условии, что множество мастер-аккаунтов не пусто. Для создания пользовательского аккаунта достаточно отправить транзакцию, содержащую новый идентификатор аккаунта. Тем не менее аккаунт смарт-контракта может быть создан только с помощью транзакции от имени существующего аккаунта. Мастер-аккаунт имеет возможность блокировать любой пользовательский аккаунт или аккаунт смарт-контракта, отправив соответствующую транзакцию.

nonce - количество транзакций, отправленных от имени данного аккаунта и успешно исполненных системой.

code - машинный код смарт-контракта. Поле сериализуется как Option<Vector<Byte>> согласно спецификации SCALE.

storage - хранилище ключ-значение (key-value). Где ключ — это непустая строка не более 32-х символов латинского алфавита, начинающаяся с буквы и содержащая буквы и цифры как показано ниже. Значением могут быть произвольные бинарные данные.

Key - строка формата [a-zA-Z][a-zA-Z0-9_-][0-31]

Value - произвольные бинарные данные

Данное поле сериализуется как Option<Vector<(Key, Value)>> согласно спецификации SCALE. При это необходимо предварительно отсортировать массив по Key.

Помимо структуры аккаунта необходимо уметь формировать и обрабатывать структуру транзакции:

```
SignedTransaction
    payload: TransactionPayload;
    signatures: Set<Signature>;
TransactionPayload
    timestamp: UInt64;
    from: GUID;
    to: Optional<GUID>;
    type: UInt8;
    nonce: UInt32;
    data: Optional<Bytes>;
```

signatures - множество подписей. Каждая подписи представляет собой массив байт длиной 64 согласно ГОСТ 34.10. В качестве данных для формирования подписи используется SCALE представление структуры TransactionPayload.

timestamp - время создания транзакции в микросекундах в формате unix-timestamp.

from - идентификатор аккаунта от имени которого отправлена транзакция.

to - идентификатор аккаунта, который изменяется под действием транзакции. Данный параметр имеет различную семантику в зависимости от типа транзакции.

nonce - количество транзакций, отправленных от имени аккаунта с идентификатором from. Данный параметр должен совпадать с nonce аккаунта в момент исполнения и необходим для защиты от атаки повторной отправки транзакции.

data - Необязательное поле, которое содержит дополнительные данные транзакции. Например, код смарт-контракта или вызов функции смарт-контракта. Данное поле сериализуется как Option<Vector<Byte>> согласно спецификации SCALE.

type - тип транзакции. Ниже приведены поддерживаемые типы:

- развертывание смарт-контракта (0)
- вызов функции смарт-контракта (1)
- создание пользовательского аккаунта (2)
- подтверждение пользовательского аккаунта мастер-аккаунтом (3)
- блокировка пользовательского аккаунта или смарт-контракта мастер-аккаунтом (4)
- установка хранилища данных аккаунта по определенному ключу (5)
- установка множества публичных ключей аккаунта (6)
- добавление новых публичных ключей аккаунта (7)
- удаление определенных публичных ключей аккаунта (8)

## 4. Входные и выходные данные

4.1. Отправка транзакции.

Для отправки транзакции необходимо поместить в тело запроса JSON следующего формата, записав в поле tx данные транзакции в шестнадцатеричном виде, предварительно закодированные SCALE кодеком:

```
{
    "jsonrpc":"2.0",
    "method":"submitRawTransaction",
    "params":{
        "tx":<транзакция в SCALE формате>,
        "id": <идентификатор запрос>
    }
    "id": <идентификатор запрос>
}
```

В случае успешного выполнения API вернет хэш опубликованной транзакции, например:

```
{
  "jsonrpc": "2.0",
  "result": {
        "msg": {
              "NewTxResponse": {
                    "hash":
"0xde6cb6ce0484b5f07cb7b3da7b8eaba21693f6d5ba11a13dbe0046845f198af4",
              }
        }
  },
  "id": 1
}
```

4.2. Получение данных транзакции.

Для получения данных о транзакции необходимо сформировать запрос содержащий хэш транзакции:

```
{
    "jsonrpc":"2.0",
    "method":"getTransaction",
    "params":{
```

```
        "hash":<хэш транзакции в шестнадцатеричном представлении>,
        "id": <идентификатор запрос>
    }
    "id": <идентификатор запрос>
}
```

В случае если транзакции с переданным хэшем принята системой, то помимо данных о транзакции возвращаются дополнительные данные о принятии транзакции:

```
{
  "jsonrpc": "2.0",
  "result": {
        "msg": {
              "SearchTxResponse": {
                    "tx": <данные SignedTransaction в SCALE формате>,
                    "commitInfo": {
                          "version": 2,
                          "ledgerIndex": 1
                    }
              }
        }
  },
  "id": 1
}
```

В случае если транзакция еще не принята сетью поле commitInfo будет отсутствовать. А если передан хэш несуществующей транзакции, то в качестве значения поля SearchTxResponse API вернет null.

4.3. Получение данных аккаунта.

Для получения данных об аккаунте необходимо сформировать запрос содержащий идентификатор аккаунта:

```
{
    "jsonrpc":"2.0",
    "method":"getAccount",
    "params":{
        "accountId":<идентификатор в шестнадцатеричном представлении>,
        "id": <идентификатор запрос>
    }
    "id": <идентификатор запрос>
}
```

В случае если аккаунт существует API вернет в качестве ответа следующую структуру:

```
{
  "jsonrpc": "2.0",
  "result": {
        "msg": {
              "AccountResponse": {
                    "account": <данные Account в SCALE формате>,
              }
        }
  },
  "id": 1
}
```

В случае если аккаунт не существует API вернет в качестве значения поля AccountResponse null.

## 5. Сообщения

Возможные проблемы при взаимодействии с системой «InnoChain» через JSON RPC 2.0 API описаны в таблице 1.

Таблица 1 – Сообщения при эксплуатации API системы «InnoChain»

| Текст сообщения | Описание | Действия |
|---|---|---|
| При запросе к API в ответ приходит ошибка вида: {"jsonrpc": "2.0", "error": {"code": -32601, "message": "Method not found"}, "id": 1} | В запросе указано неправильное название функции в поле method | Необходимо сверить поле method запроса со списком поддерживаемых функций |
| При запросе к API в ответ приходит ошибка вида: {"jsonrpc": "2.0", "error": {"code": -32000, "message": "Server error"}, "id": 1} | В запросе указано неправильное значение в поле params | Необходимо удостовериться в правильности передаваемых параметров |
| При запросе к API в ответ приходит ошибка вида: {"jsonrpc": "2.0", "result": {"id": 1, "old_len": 33, "msg": {'Error': "Incorrect message id returned from the node"}}, "id": 1} | Используются несовместимые компоненты системы | Необходимо обратиться с заявкой в техническую поддержку для получения актуальной версии системы согласно разделу 3 документа «Описание процессов жизненного цикла программного обеспечения». |

## 6. Проверка программы с помощью мобильного приложения.

Для проверки работоспособности программы также можно воспользоваться мобильным приложением. Для его сборки и запуска необходимо следующее программное обеспечение:

- git - утилита для контроля версий исходного кода;
- pod - утилита для загрузки зависимостей приложения;
- XCode - среда разработки, необходимая для сборки и установки приложения на устройство.

Для подготовки к установке мобильного приложения необходимо:

- скачать и установить последнюю доступную версию XCode в магазине приложений Apple;
- установить менеджер зависимостей: sudo gem install cocoapods;

- установить зависимости, выполнив в директории проекта команду: pod install;
- открыть файл InnochainDemo.xcworkspace исходного кода узла с помощью XCode.

Для установки приложения на симулятор необходимо:
- выбрать один из доступных симуляторов на верхней панели XCode;
- выбрать команду Product -> Run на верхней панели XCode.

Для установки приложения на устройство необходимо:
- подключить устройство к компьютеру через Wifi или специальный кабель;
- выбрать устройство на верхней панели XCode;
- выбрать команду Product -> Run на верхней панели XCode.

Для проверки работоспособности системы необходимо указать IP адрес узла системы в файле InnochainDemo/Common/Configs/PrivateUrl.swift. Далее необходимо запустить приложение и успешно создать аккаунт пользователя, нажав на + в правом верхнем углу как показано на Рис. 1:

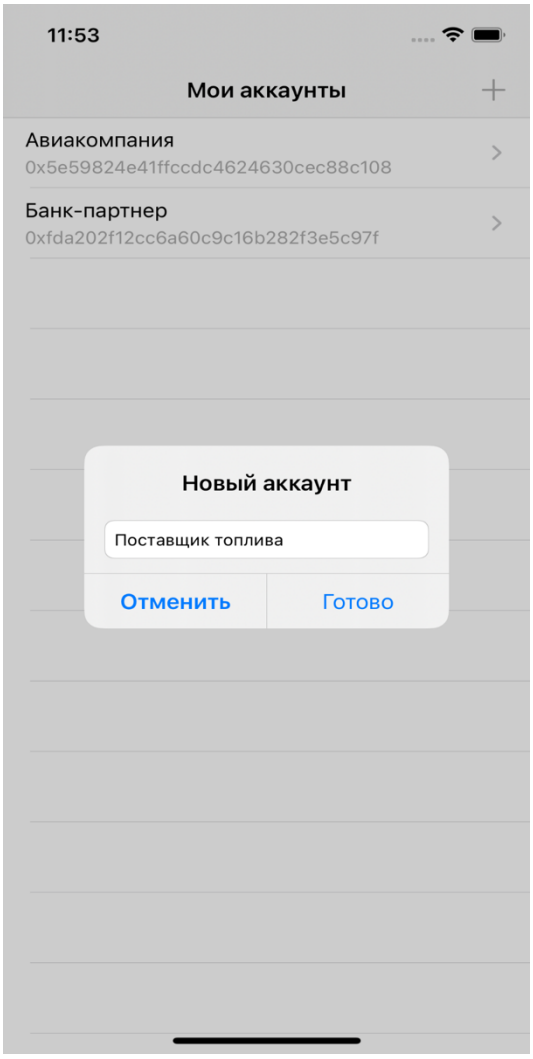


Рис. 1. Создание аккаунта пользователя