СРЕДСТВО КРИПТОГРАФИЧЕСКОЙ ЗАЩИТЫ ИНФОРМАЦИИ

СКЗИ МАСТЕРЧЕЙН

Версия 1.0

Описание программы

Листов 196

АННОТАЦИЯ

Настоящий документ представляет собой общее описание средства криптографической защиты информации Мастерчейн версия 1.0 (далее по тексту - СКЗИ Мастерчейн).

В разделе 1 приводится обозначение и наименование СКЗИ Мастерчейн, указывается ПО, необходимое для его функционирования, указываются языки программирования, на которых написан СКЗИ Мастерчейн.

В разделе 2 отражены классы решаемых задач, назначение СКЗИ Мастерчейн, сведения о функциональных ограничениях на применение.

В разделе 3 приводится описание логической структуры СКЗИ Мастерчейн, а именно алгоритм, методы, используемые при программировании, структура СКЗИ Мастерчейн, связи с другими программами.

В разделе 4 приведены используемые технические средства.

В разделе 5 приведены способы вызова и загрузки СКЗИ Мастерчейн.

В разделе 6 приведены характер, организация и подготовка входных данных.

В разделе 7 приведены характер, организация и подготовка выходных данных.

СОДЕРЖАНИЕ

1. Общие сведения 5
1.1. Обозначение и наименование программы 5
1.2. Программное обеспечение, необходимое для функционирования программы 5
1.3. Языки программирования 5
2. Функциональное назначение 6
2.1. Компонент ведения распределенного реестра 6
2.2. Компонент обеспечения сетевого взаимодействия между узлами распределенного реестра 7
2.3. Компонент исполнения прикладных задач 7
2.4. Компонент взаимодействия с криптосредствами 7
2.5. Компонент адресации участников сетевого взаимодействия 8
2.6. Компонент взаимодействия с пользователями 8
2.7. Компонент КриптоПро CSP 8
2.8. Компонент Whitelist 8
2.9. Програмный комплекс «Сервис передачи конфеденциальных сообщений» 8
2.9.1. Комопнент передачи конфеденциальных сообщений 8
2.9.2. Комопнент комопнент контроля и упраления доступом 9
2.9.3. Комопнент КриптоПро CSP 9
3. Описание логической структуры 10
3.1. Алгоритм работы СКЗИ Мастерчейн 10
3.2. Используемые методы 18
3.2.1. Эксплуатационные свойства алгоритма PoW: 18
3.2.2. Свойства безопасности алгоритма PoW: 19
3.3. Структура СКЗИ Мастерчейн 19
3.3.1. Компонент ведения распределенного реестра 21
3.3.2. Компонент обеспечения сетевого взаимодействия между узлами распределенного реестра 22
3.3.3. Компонент исполнения прикладных задач 31
3.3.4. Компонент взаимодействия с криптосредствами 34
3.3.5. Компонент адресации участников сетевого взаимодействия 34
3.3.6. Компонент взаимодействия с пользователями 35
3.3.7. Компонент Whitelist 35
3.3.8. Компонент КриптоПро 36
3.3.9. Программный комплекс «Сервис передачи конфеденциальных сообщений». 36
3.4. Связи с другими программами 38
4. Используемые технические средства 39

5. Вызов и загрузка 40
6. Входные данные 41
6.1. Характер, организация и подготовка входных данных 41
6.2. Формат, описание и способ кодирования входных данных 41
7. Выходные данные 43
7.1. Характер, организация и подготовка выходных данных 43
7.2. Формат, описание и способ кодирования выходных данных 43
7.2.1. Заголовок блока 44
7.2.2. Список транзакций 46
7.2.3. Список заголовков альтернативных блоков 48
Приложение А Описание алгоритма работы с использованием интерфейса RPC 49
Приложение Б Описание алгоритма работы с использованием интерфейса JSON RPC API 54
Приложение В Описание алгоритма Выполнить EVM код 60
Приложение Г Описание алгоритма добавление новой транзакции в пул 67
Приложение Д Описание алгоритма Whitelist проверка при создании блока 76
Приложение Е Описание алгоритма создания блока 79
Приложение Ж Описание алгоритма обработка транзакций для включения в блок 84
Приложение И Описание алгоритма Валидация блока 106
Приложение К Описание алгоритма TLS-соединение 117
Приложение Л Описание алгоритма Whitelist проверка при TLS 124
Приложение М Описание алгоритма Консенсус 128
Приложение Н Взаимосвязь компонент СКЗИ Мастерчейн с пакетами и файлами 131
Приложение О Описание функций криптоадаптера 175
Приложение П Описание RLP кодирования 185
Приложение Р Протокол TLS 188
Перечень терминов 191
Перечень сокращений 194

1. ОБЩИЕ СВЕДЕНИЯ

1.1. Обозначение и наименование программы

Средство криптографической защиты информации Мастерчейн версии 1.0 (СКЗИ Мастерчейн).

СКЗИ Мастерчейн предназначен для обмена и хранения финансовой информации с использованием технологии распределенных реестров.

1.2. Программное обеспечение, необходимое для функционирования программы

Требования к программному обеспечению:

- ОС с ядром Linux версии 3.0 и выше:
  - Ubuntu 16.04, Ubuntu 18.04, Astra Linux Community Edition/Enterprise Edition;
  - CentOS 7.5 или Red Hat Enterprise Linux 7.4;
- Интерпритатор языка Python 3.7, входящий в состав операционной системы (делее - ОС).

1.3. Языки программирования

СКЗИ Мастерчейн разработан с использованием следующих языков программирования:

- Go;
- Solidity.

2. ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

СКЗИ Мастерчейн предназначен для автоматизации формализованных процессов электронного взаимодействия участников финансового рынка, путем создания доверенной среды электронного взаимодействия. Автоматизация процессов направлена на снижение доли участия посредников при взаимодействии.

СКЗИ Мастерчейн реализован на основе технологии распределённых реестров (далее - РР). Запущенные экземпляры СКЗИ Мастерчейн формируют сеть Мастерчейн, которая обладает UDP+TCP+IP связностью.

С точки зрения организационной принадлежности, субъектами, которыми предполагается использование СКЗИ Мастерчейн, является Банк России, а также кредитные и не кредитные финансовые организации Российской Федерации.

СКЗИ Мастерчейн не может использоваться для работы со сведениями, касающимися государственной тайны.

СКЗИ Мастерчейн состоит из следующих компонентов и программных комплексов:

- Компонент ведения распределенного реестра;
- Компонент сетевого взаимодействия между узлами распределенного реестра;
- Компонент исполнения прикладных задач;
- Компонент взаимодействия с криптосредствами;
- Компонент адресации участников сетевого взаимодействия;
- Компонент взаимодействия с пользователями;
- Компонент КриптоПро CSP;
- Компонент Whitelist;
- Программный комплекс «Сервис передачи конфиденциальных сообщений» состоящий:
    - Компонента передачи конфиденциальных сообщений;
    - Компонента контроля и управления доступом;
    - Компонента КриптоПро CSP.

Функциональное назначение каждого компонента приведено ниже.

2.1. Компонент ведения распределенного реестра

Компонент обладает следующей функциональностью:

1) Проверка целостности копии РР;
2) Обработка новых транзакций:
    - Проверка корректности транзакций;
    - Проверка аутентичности транзакций.
3) Создание нового блока для РР:
    - Формирование списка транзакций, включаемых в блок;
    - Формирование нового блока;

- Валидация нового блока;
- Формирование структуры блока;
- Формирование структуры заголовка блока;
- Начисление вознаграждения: эмиссия, расчет комиссии, начисление на счет валидатора.

4) Определение консенсуса в условиях одноранговой (P2P) сети для создания единой версии РР:
   - Проверка корректности входящего блока;
   - Валидация блока.

2.2. Компонент обеспечения сетевого взаимодействия между узлами распределенного реестра

Компонент обладает следующей функциональностью:

1) Поиск участников сети;
2) Обход сети;
3) Установка сетевого соединения;
4) Синхронизация копий РР на всех участниках сети;
5) Обмен транзакциями между участниками сети по TLS.

2.3. Компонент исполнения прикладных задач

Компонент реализован на базе EVM. EVM - виртуальная машина Эфириума, для выполнения смарт-контрактов. Смарт-контракты могут быть созданы с использованием языка Solidity и предварительно скопмпилированы в байт-код EVM. Основная задача выполнения смарт-контрактов - изменение состояния хранилища РР.

Компонент обладает функциональностью, необходимой для:

1) Смарт-контрактов:
   - Публикация смарт-контракта;
   - Взаимодействие пользователей сети с существующим смарт-контрактом;
   - Прекращение действия смарт-контракта;
   - Выполнение кода смарт-контрактов, обработка транзакций.
2) Расчета комиссии за выполненные операции.
3) Возможности перенесения и обработки состояний из других систем РР.

2.4. Компонент взаимодействия с криптосредствами

Компонент позволяет подключать сторонние средства криптографической защиты информации без изменения архитектуры и логики работы СКЗИ Мастерчейн за счет использования криптоадаптера.

2.5. Компонент адресации участников сетевого взаимодействия

Компонент обладает следующей функциональностью:

1) Учёт затрат на выполнение транзакций;
2) Однозначная привязка всех действий участника к его адресу;
3) Доказательство собственности адреса: пользователь (владелец адреса) должен доказывать собственность адреса при создании транзакций с этого адреса.

2.6. Компонент взаимодействия с пользователями

Компонент обладает следующей функциональностью:

1) Вывод служебной информации;
2) Получение информации из РР;
3) Работа с сетью Мастерчейн;
4) Работа с транзакциями;
5) Работа со смарт-контрактами;
6) Вызов криптографических операций СКЗИ Мастерчейн.

2.7. Компонент КриптоПро CSP

Компонент КриптоПро используется в СКЗИ Мастерчейн в качестве криптопровайдера и выполняет функции криптографического шифрования и дешифрования данных.

2.8. Компонент Whitelist

Компонент Whitelist приеняется для управления и контроля доступа узлов Мастерчейн в сети.

2.9. Програмный комплекс «Сервис передачи конфеденциальных сообщений»

Программный комплекс «Сервис передачи конфеденциальных сообщений» (далее - ПК СПКС) предназначен для конфеденциального обмена данными между определенными участниками сети. В состав ПК СПКС входят программные комопненты перечисленные ниже.

2.9.1. Комопнент передачи конфеденциальных сообщений

Комопнент передачи конфеденциальных сообщений выполняет следубщие функции:

- Получение файлов конфеденциальных сообщений и их сохранение в хранилище;
- Предосталение файлов конфеденциальных сообщений из хранилища;
- Запрос и предоставление файлов конфеденциальных сообщений между узлами СПКС;
- Авторизация доступа к файлам конфеденциальных сообщений, находящимся в хранилище.

2.9.2. Комопнент комопнент контроля и упраления доступом

Комопнент контроля и управления доступом выполняет следующие функции:

- Ведение организационной структуры (пользователи, их роли, организации, их роли, принадлежность пользователей к организации) сети в РР;
- Хранение сертификатов пользователей;
- Ведение информационных объектов, соотвествующих файлам конфеденциальных сообщений в хранилище в РР;
- Ведение прав доступа к файлам конфеденциальных сообщений.

2.9.3. Комопнент КриптоПро CSP

Компонент КриптоПро используется в СКЗИ Мастерчейн в качестве криптопровайдера и выполняет функции криптографического шифрования и дешифрования данных.

3. ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ

3.1. Алгоритм работы СКЗИ Мастерчейн

СКЗИ Мастерчейн обеспечивает обмен и хранение финансовой информации, поступающей от пользователей посредством создания ими запросов.

СКЗИ Мастерчейн формируется из блоков данных, которые, в свою очередь, содержат в себе сообщения, создаваемые на узлах – транзакции. Каждый блок и транзакция обязательно созданы в соответствии с заданной структурой. Транзакции при создании распространяются по сети узлов широковещательным методом (от всех ко всем). Каждый узел поддерживает пул транзакций – очередь, в которую заводятся транзакции, перед тем как быть включёнными в блок. В пул отправляются транзакции, формально прошедшие проверки на корректность. Далее, транзакции из пула используются узлами для формирования блоков. Поскольку множество авторизованных узлов в сети создаёт и транслирует блоки, то необходим алгоритм консенсуса, который позволит в таких условиях выбирать единственно верную версию данных, чтоб избежать противоречивых версий РР. В сети Мастерчейн для этого используется алгоритм Hashcash Proof Of Work (POW). Таким образом, каждый создаваемый блок должен успешно проходить определённый набор проверок, перед тем как будет записан узлами (которые его получили) в их локальные копии Мастерчейн.

В общем виде алгоритм работы СКЗИ Мастерчейн представлен на рисунке – см. Рисунок 1.

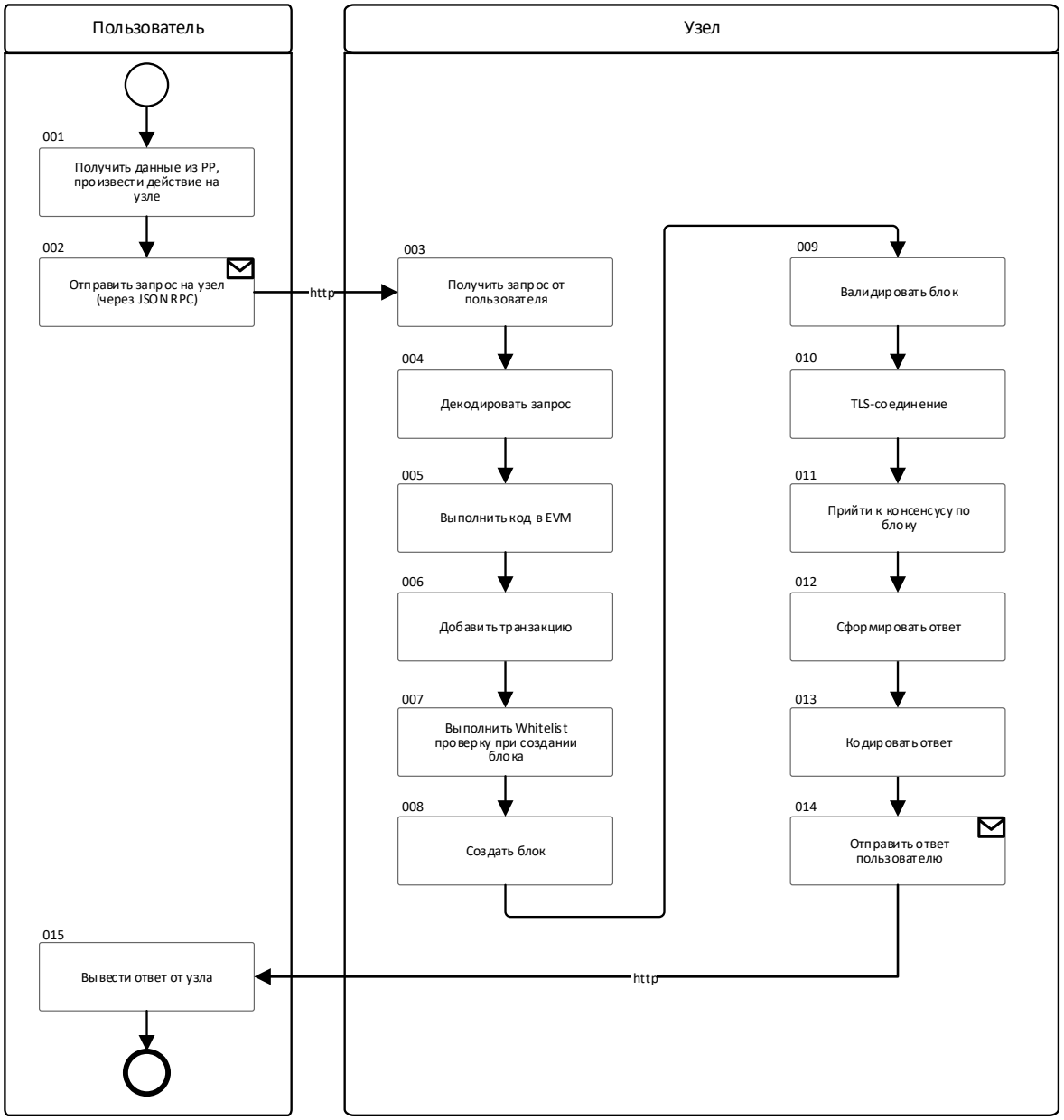


Рисунок 1 – алгоритм работы СКЗИ Мастерчейн

Пояснение к алгоритму приведены в - см. Таблица 1.

Таблица 1 – Описание алгоритма работы СКЗИ

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
1.	001	Получить данные из РР, произвести действие на узле	Пользователь может: – получить данные, хранящиеся на узле в сети Мастерчейн (например, получить блок); – отправить транзакцию; – произвести действия с узлом (например, запустить майнинг).	Входные данные, поступившие из прикладного ПО.	Данные для отправки в РР через интерфейсы доступа к JSON - RPC API, описаны в документе ЖТЯИ.00107-01 92 01 «СКЗИ Мастерчейн. Руководство системного программиста».	Компонент обеспечения сетевого взаимодействия между узлами РР, компонент взаимодействия с пользователями.
2.	002	Отправить запрос на узел (через JSON – RPC API)	Пользователь имеет возможность отправлять запросы через интерфейсы JSON – RPC API или JS API. Более подробно эти алгоритмы описаны - см. **0, 0**.	Данные для отправки в РР через интерфейсы доступа к JSON – RPC API, описаны в Приложении А документа ЖТЯИ.00107-01 92 01 «СКЗИ Мастерчейн. Руководство системного программиста».	Данные, отправленные в РР через интерфейсы доступа к JSON – RPC API, описаны в Приложении А документа ЖТЯИ.00107-01 92 01 «СКЗИ Мастерчейн. Руководство системного программиста».	Компонент адресации участников сетевого взаимодействия, компонент взаимодействия с пользователями.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
3.	003	Получить запрос от пользователя	Передача запроса пользователя на сервер через интерфейс JSON – RPC API, запущенный на узле. JSON – RPC API является частью СКЗИ Мастерчейн.	Данные, отправленные в РР через интерфейсы доступа к JSON – RPC API, описаны в Приложении А документа ЖТЯИ.00107-01 32 01 «СКЗИ Мастерчейн. Руководство системного программиста».	Полученный кодированный запрос. Состав данных описан – см.**0.**	Компонент взаимодействия с пользователями, компонент КриптоПро.
4.	004	Декодировать запрос	Отправка запроса через интерфейс RPC для декодирования. Подробно процедура декодирования описана – см.**0**.	Полученный кодированный запрос. Состав данных описан – см.**0**.	Декодированный запрос. Состав данных описан – см.**0**.	Компонент КриптоПро.
5.	005	Выполнить код в EVM	Отправка декодированного запроса через интерфейс JSON – RPC API на обработку, выполнение кода EVM. Алгоритм работы EVM описан - см. **0.**	Декодированный запрос. Состав данных описан – см.**0**. EVM код. Количество доступного Газа. Адрес смарт контракта, которому принадлежит исполняемый код, формат данных описан в п.3.3.3.1. Адрес отправителя транзакции, инициировавшей это выполнение - объект транзакции string from.	Транзакции, подписанные отправителем. Формат данных описан в п.6.2.	Компонент исполнения прикладных задач.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
6.	006	Добавить транзакцию	Алгоритм добавления новой транзакции в пул описан – см.**0**.	Транзакция, подписанная отправителем. Формат данных описан в п.6.2. Минимальное значение цены на Газ.	Транзакция, подписанная отправителем, добавленная в пул. Формат данных описан в п.6.2.	Компонент ведения РР, компонент адресации участников сетевого взаимодействия, компонент взаимодействия с пользователями.
7.	007	Выполнить Whitelist проверку при создании блока	Алгоритм Whitelist проверки при создании блока описан – см. **0**.	Whitelist узлов, идентификатор ключа узла, формат описан в п.3.3.7. Блок, подписанный идентификатором ключа узла, формат данных описан в п.7.2.	Локальный РР с блоком, формат данных описан в п.7.2.	комопнент Whitelist

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
8.	008	Создать блок	Алгоритм создания блока описан – см. **0**, **0**.	Данные о пройденных проверках.	Отправленный блок, записанный в локальный РР, формат данных описан в п.7.2.	Компонент ведения РР, компонент обеспечения сетевого взаимодействия между узлами РР, компонент исполнения прикладных задач, компонент адресации участников сетевого взаимодействия.
9.	009	Валидировать блок	Алгоритм валидации блока описан – см.**0**.	Отправленный блок, записанный в локальный РР, формат данных описан в п.7.2.	Провалидированный блок, записанный в РР, формат данных описан в п.7.2.	Компонент ведения РР, компонент обеспечения сетевого взаимодействия между узлами РР, компонент исполнения прикладных задач.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
10.	010	TLS-соединение	Алгоритм TLS-соединения описан – см. **0, 0**.	Провалидированный блок, записанный в РР, формат данных описан в п.7.2. Адреса узлов. Данные для Hello-сообщения (отправленное Hello-сообщение), формат данных описан – см. **0**.	Блок, формат данных описан в п.7.2.	Компонент обеспечения сетевого взаимодействия между узлами РР.
11.	011	Прийти к консенсусу по блоку	Алгоритм достижения консенсуса описан – см.**0**.	Блок, формат данных описан в п.7.2.	Блок, включенный в РР, формат данных описан в п.7.2. Данные о пройденных проверках, состав описан в п.3.3.1.4.	Компонент ведения РР.
12.	012	Сформировать ответ	Формирование данных для ответа через интерфейс JSON – RPC API.	Блок, включенный в РР, формат данных описан в п.7.2. Данные о пройденных проверках, состав описан в п.3.3.1.4.	Обработанные данные РР для ответа. Состав данных описан – см.**0**.	Компонент взаимодействия с пользователями, компонент КриптоПро.
13.	013	Кодировать ответ	Отправкаданных через интерфейс RPC для кодирования. Подробно процедура кодирования описана – см.**0**.	Обработанные данные РР для ответа. Состав данных описан – см.**0**.	Кодированный ответ. Состав данных описан – см.**0**.	Компонент КриптоПро.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
14.	014	Отправить ответ пользователю	Отправка закодированного ответа пользователю по http через интерфейс RPC.	Кодированный ответ. Состав данных описан – см.**0**.	Отправленный кодированный ответ пользователю. Состав данных описан – см.**0**.	Компонент взаимодействия с пользователями, компонент КриптоПро.
15.	015	Вывести ответ от узла	Пользователь получает ответ от узла. Более подробно эти алгоритмы описаны - см.**0, 0**.	Отправленный кодированный ответ пользователю. Состав данных описан – см.**0**.	Ответ пользователю, возвращаемый интерфейсами доступа JSON – RPC API, описанными в документе ЖТЯИ.00107-01 92 01 «СКЗИ Мастерчейн. Руководство системного программиста».	Компонент взаимодействия с пользователями.

3.2. Используемые методы

В СКЗИ Мастерчейн применены следующие языки программирования:

- Go;
- Solidity.

Дополнительно в СКЗИ Мастерчейн применены методы программной библиотеки КриптоПро CSP. Компонент КриптоПро используется в СКЗИ Мастерчейн в качестве криптопровайдера.

При криптографических операциях происходит обращение к криптоадаптеру. Криптоадаптер вызывает необходимые функции криптопровайдера.

Компонент КриптоПро используется для выполнения следующих криптографических операций:

- Аутентификации и проверки целостности;
- Обеспечение конфиденциальности.

Методы КриптоПро CSP используются в алгоритмах, описанных в приложениях: **0 - 0**.

В СКЗИ Мастерчейн криптографические методы реализованы в соответствии со следующими российскими стандартами криптографии:

1. Алгоритм вычисления значения хэш-функции реализован в соответствии с требованиями ГОСТ Р 34.11-94 / ГОСТ Р 34.11-2012 «Информационная технология. Криптографическая защита информации. Функция хэширования».
2. Алгоритмы формирования и проверки электронной подписи реализованы в соответствии с требованиями ГОСТ Р 34.10-2001 / ГОСТ Р 34.10-2012 «Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи».
3. Алгоритм шифрования – расшифровывания данных и вычисление имитовставки реализованы в соответствии с требованиями ГОСТ 28147-89 «Системы обработки информации. Защита криптографическая».

Для достижения консенсуса в сети используется алгоритм PoW (Proof-of-Work), так как он обладает следующими свойствами, описанными в п.п. 3.2.1, 3.2.2.

3.2.1. Эксплуатационные свойства алгоритма PoW:

1) Масштабируемость по количеству узлов – рост количества узлов не влияет на сложность алгоритма консенсуса.
2) Масштабируемость по количеству транзакций – для обработки большего количества транзакций достаточно увеличения количества узлов.

3) Поддержка режима распределённой системы, когда нижеперечисленные утверждения не выполнимы:

– Сеть работает безотказно;
– Пропускная способность сети бесконечна;
– Сеть безопасна;
– Топология сети постоянна;
– Возможно централизованное администрирование;
– Передача информации бесплатна;
– Все узлы и все каналы связи идентичны.

4) Отсутствие единой точки доверия или управления.

5) Наличие «истории успеха» в условиях массового использования в агрессивной среде.

Скорость не рассматривается как важное эксплуатационное свойство, так как скорость не зависит от алгоритма консенсуса в условиях распределённой системы без централизованного управления и при неопределённом количестве участников.

3.2.2. Свойства безопасности алгоритма PoW:

1) Доказательство аутентичности цепочки – если цепочек несколько, каждый узел имеет возможность определить, какая из существующих цепочек является единственно верной.

2) Влияние данных внутри цепочки на логику алгоритма консенсуса – наличие каких-либо расчётных единиц, которые определяют роль валидатора и управление которыми требует дополнительных мер безопасности.

3) Постоянное присутствие дополнительных «секретов» в системе для поддержания её работы – наличие каких-либо ключей для подписания блоков. «Секрет» представляет собой случайную последовательность байтов, зашифрованную открытым ключом сервера, которая позволяет пользователю и серверу вычислить секретный ключ, который будет использоваться для шифрования сообщений на уровне соединения.

4) Возможность рассчитать цену взлома системы с высокой точностью.

3.3. Структура СКЗИ Мастерчейн

Ввиду использования технологии распределенных реестров СКЗИ Мастерчейн представляет собой совокупность программных средств, установленных на отдельных узлах и объединенных в сеть Мастерчейн. Структура СКЗИ Мастерчейн в разрезе компонент представлена – см. Рисунок 2.

Рисунок 2 – компонентная структура СКЗИ Мастерчейн.

Взаимосвязь между компонентами и пакетами, файлами СКЗИ Мастерчейн приведена – см. **0**.

3.3.1. Компонент ведения распределенного реестра

Проверка целостности копии РР

Сразу после запуска СКЗИ Мастерчейн начинает процесс синхронизации с узлами Мастерчейн. Новые блоки, полученные от других узлов, проверяются один за другим согласно алгоритму, описанному – см.**0**.

Начальный блок РР имеет номер «0» и это единственный блок, который не имеет родителя. Начальный блок РР уникален для сети. В текущей версии СКЗИ Мастерчейн он объявлен в скомпилированном коде константой.

Начальный блок РР включает в себя смарт-контракт с Whitelist узлов.

Блок будет включён в РР только если он подписан узлом, который есть в Whitelist узлов и обладает ролью майнер.

Если в процессе синхронизации получен некорректный блок или последовательность блоков, то соединение с узлом-источником некорректного фрагмента разрывается, и синхронизация продолжается с других узлов.

Обработка новых транзакций

Транзакция может быть в двух состояниях:

- Транзакция не обработана: транзакция находится в пуле;
- Транзакция обработана: транзакция обработана, включена в блок, блок записан в РР.

Обработка транзакций является частью создания нового блока.

В начале обработки транзакций майнер фиксирует состояние системы, которое соответствует состоянию системы, указанному в последнем блоке, включённом в РР. В конце обработки транзакций для нового блока будет получено новое состояние системы, которое будет записано в заголовок блока. Когда новый блок записывается в РР – это меняет состояние системы для всей сети.

Майнер берёт все транзакции из пула транзакций, очередь pending и сортирует их следующим образом:

- По отправителю и вознаграждению (транзакции с самым высоким вознаграждением в начале списка);
- Для каждого отправителя транзакции сортируются по нонсу.

Формирование отсортированного массива гарантирует, что подпись и другие параметры каждой транзакции верны.

Более подробно алгоритм обработки транзакций описан – см. **0**.

После того как транзакция включена в блок и блок записывается в РР состояние транзакции считается изменённым (транзакция обработана).

Создание нового блока для РР

Алгоритмы создания и валидации блока описаны – см. **0, 0, 0**.

Определение консенсуса в условиях одноранговой (P2P) сети для создания единой версии РР

При проверке корректности входящего блока, производятся проверки соответствия структуры блока стандарту:

- Заголовок: структура заголовка соответствует стандарту, поля заголовка имеют значения соответствующего типа данных;
- Список транзакций может быть пустым, если не пустой: транзакции сформированы согласно стандарту, значение полей транзакций имеют значения соответствующих типов данных;
- Список заголовков альтернативных блоков может быть пустым, если не пустой: заголовки сформированы согласно стандарту, поля заголовков имеют значения соответствующих типов данных.

Более подробно алгоритм работы консенсуса описан - см. **0**.

3.3.2. Компонент обеспечения сетевого взаимодействия между узлами распределенного реестра

Цель каждого узла Мастерчейн – это синхронизация РР. Для этого узел находит другие узлы сети Мастечейн, устанавливает с ними соединение и обменивается блоками и транзакциями.

При запуске узел подсоединяется к стартовым узлам, при дальнейшей работе для «обхода» сети используется разновидность Kademlia DHT протокола: Node Discovery протокол, версия 4. При установлении соединения между узлами используется Whitelist узлов.

Поиск участников сети

Поиск участников сети заключается в поиске стартовых узлов. Стартовые узлы – узлы Мастерчейн, доступные для отправки и получения UDP сообщения по заданному порту. Стартовые узлы необходимы при начале работы узла, когда у узла нет таблицы известных узлов. Таблица маршрутизации составляется при «обходе» сети по Node Discovery протоколу. Есть три способа передачи узлу списка стартовых узлов:

- Несколько стартовых узлов объявлены константами в СКЗИ Мастерчейн;
- Адреса стартовых узлов могут быть указаны при запуске узла;
- Стартовые узлы могут быть добавлены во время работы узла.

Стартовые узлы используются исключительно для получения списка других узлов.

После начала работы узел подсоединяется к одному из стартовых узлов. Стартовый узел возвращает список узлов, которые подсоединялись к этому стартовому узлу за последние 2 часа. Этот протокол используется для постоянного «обхода» сети.

Запрос на обнаружение новых узлов происходит регулярно. Каждый узел старается держать в таблице не менее 25 узлов, с которыми производится обмен данными по TLS. Данные хранятся в RAM и пишется в локальную базу levelDB. При штатном завершении работы, известные узлы сети сохраняются на диск и используются при следующем старте.

Обход сети

Обход сети осуществляется с помощью Node Discovery протокола, версия 4. В качестве идентификаторов ключей узлов используются публичные ключи узлов. Расстояние между двумя узлами – это побитовое исключающее ИЛИ хешей публичных ключей этих узлов:

```
distance (n₁, n₂) = gost256(n₁) XOR gost256(n₂)
```

Узлы при использовании Node Discovery протокола хранят информацию о «соседних» узлах. Соседние узлы хранятся в таблице известных узлов, которая состоит из k-слотов. Для каждого i, 0≤ i <256, у каждого узла есть один k-слот для узлов, находящихся на расстоянии 2i и 2i+1 от данного узла.

Node Discovery протокол использует k = 16, то есть каждый k-слот содержит 16 записей, отсортированных по принципу: самый недавно виденный узел в начале, давно виденный узел в конце.

Для формирования таблицы известных узлов узел запрашивает ближайшие известные узлы о ближайших известных им узлах. Когда новый узел N_1 обнаружен, он помещается в соответствующий слот. Если в слоте менее k записей, новый узел добавляется как первая запись. Если слот уже содержит записи, то самый давний узел необходимо проверить: посылается ping пакет. Если ответа нет, то узел считается недоступным и удаляется, первой записью слота добавляется N_1. Если N_1 ближе известных опрошенных узлов, то этот узел опрашивается о ближайших известных ему узлах.

Для формирования таблицы известных узлов используется рекурсивный поиск. Поиск узлов осуществляется по алгоритму – см. Рисунок 3

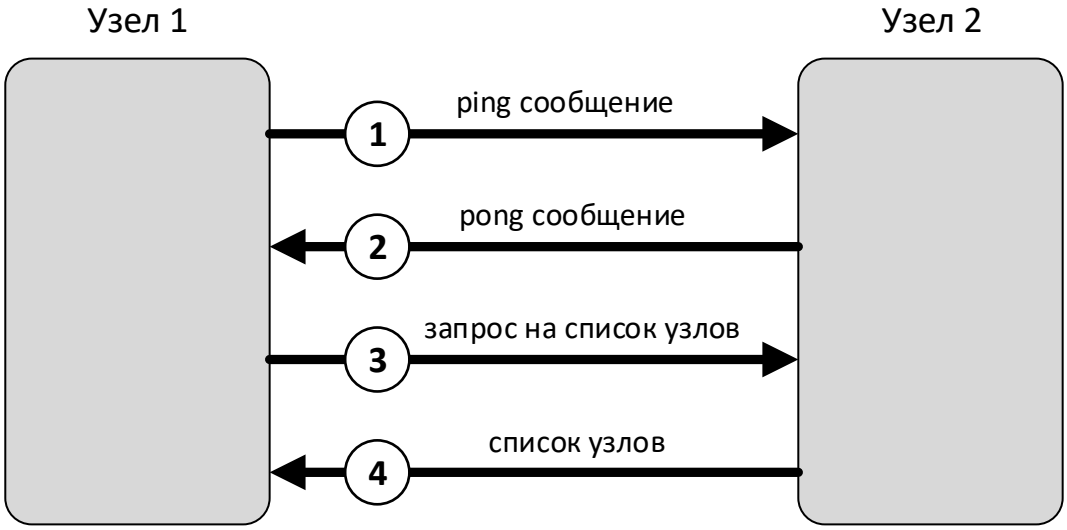


Рисунок 3 – Алгоритм поиска узлов.

Описание шагов алгоритма поиска узлов – см. Таблица 2.

Таблица 2 – Шаги алгоритма поиска узлов.

№ п/п	Шаг	Описание
1.	1	Узел 1 посылает сообщение **ping** узлу из списка.
2.	2	На сообщение **ping** узел 2 отвечает сообщением **pong**. Если узел 1 не получал ответа на сообщение **ping**, узел 2 удаляется из списка узлов на подсоединение.
3.	3	Узел 1 запрашивает список узлов, соседних (по DHT) идентификатору ключа, который указывается в запросе. Сообщение **FindNode**.
4.	4	Узел 2 отправляет список узлов, соседних (по DHT) идентификатору ключа, который указан в запросе (шаг 3). Сообщение **Neighbors**. Пакет **Neighbors** будет послан узлу 1 в том случае, если от этого узла был получен валидный **pong**-ответ (который содержит тот же хеш, что и **ping** пакет) за последние 12 часов.

Node Discovery сообщения посылаются UDP пакетами (максимальный размер 1280 байт):

```
packet = packet-header || packet-data
```

Заголовок пакета:

```
packet-header = hash || signature || packet-type
hash = gost256(signature || packet-type || packet-data)
signature = sign(packet-type || packet-data)
```

Ниже приведено описание параметров заголовка пакета:

hash - используется для того, чтобы отличать пакеты, когда несколько протоколов используют один UDP порт;

signature – подпись пакета узлом, в виде массива байтов, длинной 65: r || s || v (gost 34.10);

packet-type – байт, определяющий тип сообщения.

Данные пакета:

Данные, которые идут сразу после заголовка пакета определяются типом пакета.

Ping пакет:

```
packet-type = 0x01
packet-data = [version, from, to, expiration] version = 4 from = [sender-ip, sender-udp-port,
sender-tcp-port] to = [recipient-ip, recipient-udp-port, 0]
```

expiration – время в UNIX формате. Пакеты, время в которых указано в прошлом, не обрабатываются.

Узел-получатель отвечает на пакет ping пакетом в формате pong. Узел-получатель может добавить узел-отправитель в свою таблицу известных узлов.

Если никакого взаимодействия с узлом отправителем не было последние 12 часов, дополнительно посылается ping пакет.

Pong пакет:

```
packet-type = 0x02
packet-data = [to, ping-hash, expiration]
```

Pong пакет является ответом на ping пакет.

ping-hash – должен быть равен hash соответствующего ping пакета, если это не так, пакет игнорируется.

FindNode пакет:

```
packet-type = 0x03
packet-data = [target, expiration]
```

Данный пакет – запрос информации об узлах, ближайших к target. Узел-получатель должен ответить пакетом формата Neighbors, который содержит 16 ближайших узлов из локальной таблицы известных узлов к запрошенному target.

Пакет Neighbors будет послан узлу в том случае, если от этого узла был получен валидный pong-ответ за последние 12 часов.

Neighbors пакет:

```
packet-type = 0x04
packet-data = [nodes, expiration]
nodes = [[ip, udp-port, tcp-port, node-id], ... ]
```

Neighbors пакет является ответом на FindNode пакет. Данный пакет содержит 16 ближайших узлов из локальной таблицы известных узлов к запрошенному target.

Пакет Neighbors будет послан узлу в том случае, если от этого узла был получен валидный pong-ответ за последние 12 часов.

Установка сетевого соединения

Узел инициирует соединение с узлом из таблицы известных узлов. Если узел не ответил, то этот узел удаляется из таблицы известных узлов. Если узел ответил, устанавливается TLS сессия.

После установления соединения узлы могут начать обмениваться P2P сообщениями. В любой момент сообщение **Disconnect** может быть послано. **Disconnect** [0x01: P, reason: P] – сообщение о неизбежном отключении узла, где

- 0x01 – код сообщения;
- reason – (опционально) причина отключения.

Возможные причины отключения:

- 0x00 – запрошенное отсоединение;
- 0x01 – ошибка TCP подсистемы;
- 0x02 – нарушение протокола;
- 0x03 – бесполезное соединение;
- 0x04 – слишком много соединений;

- 0x05 – соединение уже установлено;
- 0x06 – несовместимые версии P2P протоколов;
- 0x07 – не получен идентификатор ключа узла;
- 0x08 – выход узла из системы;
- 0x09 – неожиданный идентификатор ключа узла (например, отличается от того, что был получен при предыдущем соединении);
- 0x0a – идентификатор ключа узла подключения совпадает с идентификатором ключа данного узла (то есть узел подсоединяется к самому себе);
- 0x0b – таймаут при получении сообщения (то есть ничего не получено с последнего ping сообщения);
- 0x10 – другая причина, специфичная для под-протокола.

Ping [0x02: P] – запрос на немедленный Pong ответ.

Pong [0x03: P] – ответ на Ping пакет.

Синхронизация копий РР на всех участниках сети

Узлу необходимо получить последовательность блоков, которые можно использовать для подтверждения текущего консенсуса. Задача узла – найти лучшую (с максимальной общей сложностью: Total Difficulty) цепочку блоков для локальной копии РР.

Алгоритм синхронизации РР:

- Два узла сразу после установления соединения посылают друг другу сообщение о состоянии (Status message), которое включает в себя общую сложность (Total Difficulty, далее - TD) и хеш заголовка наилучшего блока в копии РР на узле;
- Узел, значение TD которого больше. Отправляет цепочку хешей блоков узлу с меньшим значением TD;
- Пока есть хеши в цепочках хешей, которых нет в локальной копии РР необходимо запросить N блоков используя хеши, пометить, как запрошенные, чтобы не запрашивать у другого узла.

Пример алгоритма синхронизации РР – см. Рисунок 4.

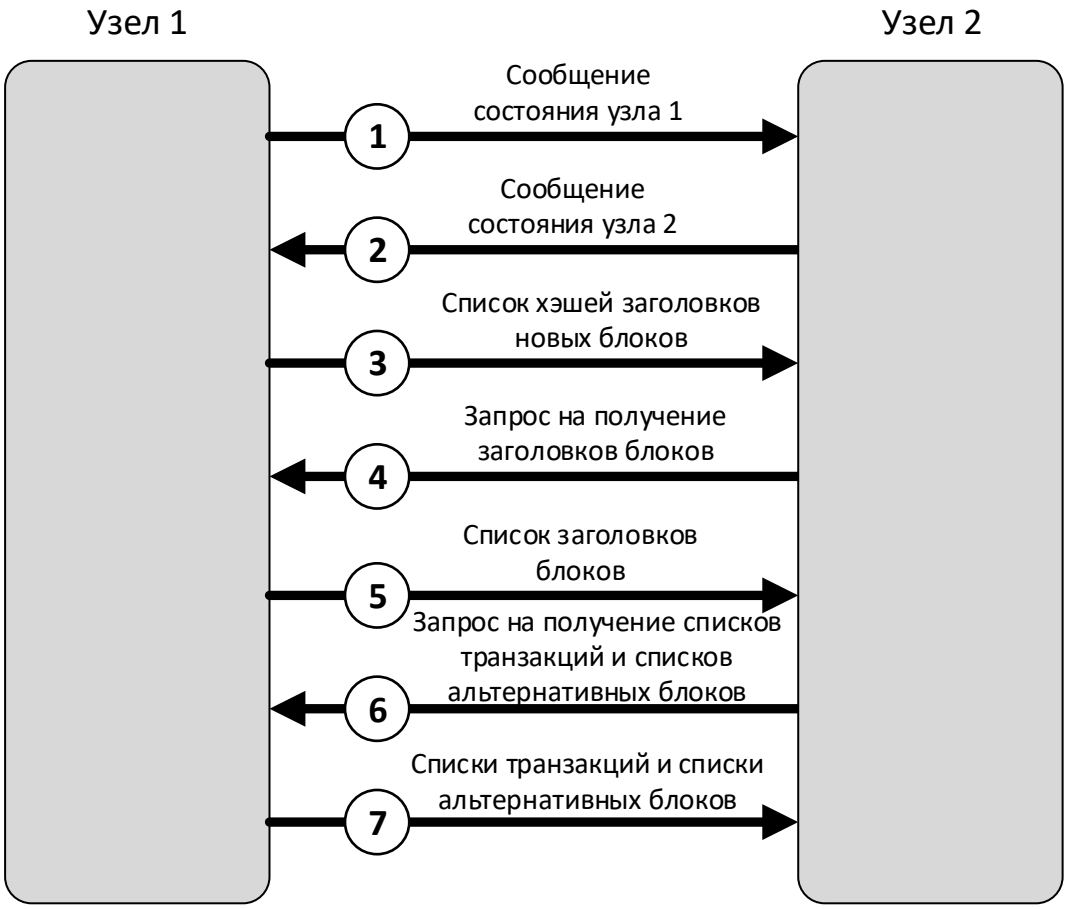


Рисунок 4 - Алгоритм синхронизации РР.

Описание шагов алгоритма синхронизации – см. Таблица 3.

Таблица 3- Шаги алгоритма синхронизации PP.

№ п/п	Шаг	Описание
1.	1	Узел 1 посылает сообщение своего состояния узлу 2. Сообщение **Status**.
2.	2	Узел 2 посылает сообщение своего состояния узлу 1. Сообщение **Status.**
3.	3	Если общая сложность PP узла 1 больше общей сложности узла 2, тогда узел 1 посылает список хешей, новых для узла 2, узлу 2 (см. **Рисунок 3**). Если общая сложность PP узла 2 больше общей сложности узла 1, тогда узел 2 посылает список хешей, новых для узла 1, узлу 1. Сообщение **NewBlockHashes.**
4.	4	Узел 2 запрашивает заголовки блоков для заданных блоков. Сообщение **GetBlockHeaders**.
5.	5	Узел 1 возвращает заголовки блоков для заданных блоков. Сообщение **BlockHeaders**.
6.	6	Узел 2 запрашивает списки транзакций и списки альтернативных блоков для заданных блоков. Сообщение **GetBlockBodies**.
7.	7	Узел 1 возвращает списки транзакций и списки альтернативных блоков для заданных блоков. Сообщение **BlockBodies**.

Сообщение о состоянии

```
Status [+0x00: P, protocolVersion: P, networkId: P, td: P, bestHash: B_32, genesisHash: B_32]
```

Описание полей – см. Таблица 4.

Таблица 4 – Описание полей сообщения о состоянии.

№ п/п	Поле	Описание
1.	0x00	Код, с которого начинается сообщение о состоянии.
2.	protocolVersion	Версия протокола узла, для данного проекта значение 62 (версия протокола PV62).
3.	networkId	Идентификатор сети.
4.	td	Общая сложность лучшей цепочки блоков.
5.	bestHash	Хеш лучшего блока.
6.	genesisHash	Хеш начального блока PP.

Сообщение о новых блоках

Хеш нового блока (или новых блоков) посылается с помощью сообщения NewBlockHashes.

Последовательность заголовков блоков запрашивается и получается с помощью пары сообщений GetBlockHeaders/BlockHeaders. Набор содержимого блоков (списки транзакций и списки альтернативных блоков) получается с помощью пары сообщений GetBlockBodies/BlockBodies.

Запросы на получение блоков распределяются между узлами, с которыми установлены соединения для распределения нагрузки и оптимизации пропускной способности сети.

```
NewBlockHashes [+0x01: P, [hash_0: B_32, number_0: P], [hash_1: B_32, number_1: P], ...]
```

Сообщение содержит список хешей новых блоков. Сообщение содержит информацию минимум об одном новом блоке.

Описание полей – см. Таблица 5.

Таблица 5 – Описание полей сообщения о новых блоках.

№ п/п	Поле	Описание
1.	0x01	Код, с которого начинается сообщение со списком хешей новых блоков.
2.	[hash_0: B_32, number_0: P]	Хеш заголовка блока, порядковый номер блока.
3.	…	…

Сообщение-запрос на BlockHeaders:

```
GetBlockHeaders [+0x03: P, block: { P , B_32 }, maxHeaders: P, skip: P, reverse: P in { 0 , 1 } ]
```

Описание полей – см. Таблица 6.

Таблица 6 – Описание полей сообщения-запроса на получение списка заголовков блоков.

№ п/п	Поле	Описание
1.	0x03	Код с которого начинается сообщение-запрос заголовков блоков.
2.	block	Номер блока или хеш, начиная с которого требуются заголовки блоков.
3.	maxHeaders	Максимальное количество ожидаемых заголовков.
4.	skip	Сколько блоков пропустить, начиная с блока, указанного в block.
5.	reverse	0: заголовки блоков в восходящем порядке, начиная с блока, указанного в поле block; 1: заголовки блоков в восходящем порядке, начиная с блока, указанного в поле block.

Соообщение-ответ на GetBlockHeaders:

```
BlockHeaders [+0x04, blockHeader_0, blockHeader_1, ...] - сообщение-ответ на GetBlockHeaders
```

Сообщение содержит список заголовков блока, в стандартном формате, которые удовлетворяют запросу GetBlockHeaders. Сообщение может не содержать ни одного заголовка в случае, если узел не имеет блоков, удовлетворяющих запросу GetBlockHeaders.

Описание полей – см. Таблица 7.

Таблица 7 - Описание полей сообщения-ответа со списком заголовков блока.

№ п/п	Поле	Описание
1.	0x04	Код, с которого начинается сообщение со списком заголовков блоков.
2.	blockHeader_0	Заголовок блока в стандартном формате.
3.	…	…

Сообщение-запрос на BlockBodies:

```
GetBlockBodies [+0x05, hash_0: B_32, hash_1: B_32, ...] – сообщение-запрос на BlockBodies
```

Описание полей – см. Таблица 8.

Таблица 8 - Описание полей сообщения-запроса на получение списков транзакций и списков альтернативных блоков.

№ п/п	Поле	Описание
1.	0x05	Код, с которого начинается сообщение-запрос на получение блоков.
2.	hash_0	Хеш блока.
3.	…	…

Сообщение-ответ на GetBlockBodies:

```
BlockBodies [+0x06, [transactions_0, uncles_0] , ...] – сообщение-ответ на GetBlockBodies
```

Сообщение содержит списки транзакций и списки альтернативных блоков в стандартном формате, которые удовлетворяют запросу GetBlockBodies. Сообщение может не содержать никакой информации о содержимом блоков в случае, если узел не имеет блоков, удовлетворяющих запросу GetBlockBodies.

Описание полей – см. Таблица 9.

Таблица 9 – Описание полей сообщения-ответа со списками транзакций и альтернативных блоков.

№ п/п	Поле	Описание
1.	0x06	Код, с которого начинается сообщение со списками транзакций и списками альтернативных блоков.
2.	[transactions_0, uncles_0]	Список транзакций и список альтернативных блоков в стандартном формате.
3.	…	…

Обмен транзакциями между участниками сети по TLS

Когда в пуле транзакций появляются новые транзакции, узел посылает сообщение **Transactions** узлам, с которыми установлено соединение. Данное сообщение содержит одну и более новых транзакции в стандартном для транзакций формате. Узел не посылает транзакцию другому узлу, если знает, что на другом узле есть эта транзакция (в случае, если узел получил транзакцию от этого узла или посылал транзакцию этому узлу ранее).

```
Transactions [+0x02: P, [nonce: P, receivingAddress: B_20, value: P, …], …]
```

Сообщение содержит минимум одну транзакцию.

Описание полей – см. Таблица 10.

Таблица 10 –Описание полей при обмене транзакциями.

№ п/п	Поле	Описание
1.	0x02	Код, с которого начинается сообщение с новыми транзакциями.
2.	[nonce: P, receivingAddress: B_20, value: P, …]	Транзакция в стандартном формате.
3.	…	…

3.3.3. Компонент исполнения прикладных задач

Смарт-контракты

Смарт-контракт представляет собой блок кода (фактически Solidity-класс). Язык описания смарт-контрактов - Solidity, является тьюринг-полным и позволяет реализовать любую вычислимую функцию.

Создание смарт-контракта включает в себя следующие этапы:

– Описание смарт-контракта на языке Solidity;

- Компиляция в байт-код EVM. Результатом компиляции являются два файла. Один файл содержит двоичный код, второй файл содержит ABI (Application Binary Interface) смарт-контракта. ABI – это javascript объект, содержащий имена и описание функций смарт-контракта и как их вызывать – это документация смарт-контракта;
- Публикация смарт-контракта в РР. Для того чтобы создать смарт-контракт в РР необходимо послать транзакцию. В поле **data** транзакции указывается код, полученный на предыдущем шаге, в поле **to -** не занятый (не существующий) адрес или ничего не указывается. За обработку этой транзакции (как и за обработку любой транзакции) с адреса отправителя будет снято соответствующее вознаграждение. Через некоторое время транзакция будет включена в блок и записана в РР. Адрес смарт-контракта будет создан со значением, указанном в поле **to**. Если адрес в поле **to** явно не указан, то смарт-контракт будет создан с незанятым адресом.

Когда смарт-контракт записан в РР, к нему можно обращаться по адресу смарт-контракта. С существующим смарт-контрактом может взаимодействовать пользователь или другой смарт-контракт.

К функциям смарт-контракта можно обратиться двумя способами:

- Через вызов (**call**) функции смарт-контракта. В этом случае все вычисления происходят на узле, где вызов производится и состояние РР не меняется. Выполнение происходит немедленно. Вычисления (в терминах Газа) бесплатны;
- Через подписанную транзакцию. Транзакция меняет состояние РР (выполнение происходит с некоторой задержкой), возвращает хэш транзакции. Отправитель транзакции платит (стоимость Газа) за обработку транзакции и выполнение кода смарт-контракта.

Смарт-контракт может существовать пока существует РР. Желательно, чтобы в каждом смарт-контракте была определена функция, вызов которой будет терминировать смарт-контракт. Если такая функция определена, то обращение к этой функции через транзакцию прекращает действие смарт-контракта.

Код смарт-контракта состоит из последовательности байтов, где каждый байт представляет операцию или данные. Выполнение кода – это бесконечный цикл, который состоит из последовательно выполняемых операций до тех пор, пока конец кода не будет достигнут, не произойдёт ошибка или не будет обнаружены инструкции Stop или Return.

Транзакции и код смарт-контракта выполняется внутри EVM. EVM является частью каждого узла, полностью изолирована, что означает, что код, выполняемый внутри EVM, не имеет доступа к сети, файловой системе или к любым другим процессам. Так же исполняемый код смарт-контракта имеет ограниченный доступ к другим смарт-контрактам. Эти меры предосторожности позволяют безопасно выполнять непроверенный код из произвольных источников.

Расчет комиссии за выполненные операции

Для каждой выполненной операции существует определенная стоимость, выраженная в количестве единиц Газа.

Газ – это название платы за выполнение, которую отправитель транзакций должен платить за каждую операцию в РР. Газ покупается за базовые токены, используемые для расчётов в сети. Газ и базовые токены разделены специально, поскольку единицы Газа стоят в одном ряду с вычислительными устройствами, которые имеют стоимость в реальном мире (стоимость электроэнергии, стоимость оборудования), в то время как цена базовых токенов обычно колеблется в результате рыночных сил. Цена Газа фактически определяется майнерами, которые могут не обрабатывать транзакции с более низкой ценой на Газ, чем их минимальный предел. Газ не нужно покупать специально: на адресе должно быть достаточно базовых токенов. При обработке транзакции Газ автоматически покупается за цену и в размере, указанные в транзакции.

Каждая транзакция должна включать в себя лимит Газа и цену Газа, которую посылающий согласен платить. Майнеры могут обработать транзакцию, включить её в блок и собрать комиссию или проигнорировать транзакцию. Если общее количество Газа, используемое вычислительными шагами, вызванными транзакцией (включая исходное сообщение и любые дополнительные сообщения), меньше или равно указанному лимиту Газа, то транзакция обрабатывается. Если общее количество Газа превышает указанный лимит, то все изменения, произведённые при вычислениях, откатываются назад, майнер берёт комиссию за проделанные вычисления. По окончанию вычислений (которые могут закончиться достижением конца кода или возвратом ошибки) стоимость неиспользованного Газа возвращается на адрес отправителя по той же цене, что указана в транзакции (и по которой Газ был куплен для выполнения транзакции).

Комиссия может быть начислена в двух разных случаях:

- Плата за вычисления, необходимые для выполнения отдельной операции;
- Плата за место в РР (плата начисляется за размер записываемых данных).

Для каждой операции EVM определено сколько единиц Газа необходимо для её выполнения. Цена зависит от процессорного времени EVM, которое необходимо для вычисления, размера пересылаемой информации и памяти, используемой при вычислении. Операция за освобождение памяти спонсируется сетью – за её выполнение начисляется вознаграждение.

Сколько единиц Газа потребует выполнение транзакции или кода смарт-контракта можно оценить, используя функцию JS API СКЗИ Мастерчейн eb3.eth.estimateGas. Вызов этой функции симулирует выполнение транзакции или кода смарт-контракта непосредственно на узле и возвращает оценочное количество Газа.

Концепция Газа введена, как защита от DDoS атак. Атакующий, как и любой пользователь, должен платить за каждый ресурс, который он использует, включая вычисления, размер пересылаемых транзакций и за хранение данных. Таким образом, любая транзакция, которая приводит к тому, что сеть потребляет больший объем любого из этих ресурсов, требует оплаты за Газ и увеличение оплаты примерно пропорционально приращению используемых ресурсов сети.

Более подробно алгоритм EVM описан – см. **0**.

Возможность перенесения и обработки состояний из других систем РР

Для перенесения блоков из одного РР(1) в другой РР(2) должен существовать смарт-контракт, такой что функция проверки этого смарт-контракта реализует алгоритм консенсуса РР(1).

На вход функция проверки смарт-контракта принимает блок n и заголовок блока n-1 из РР(1), производит все необходимые проверки. Если функция проверки вызывается через транзакцию, то данные о блоке n из РР(1) будут включены в блок и записаны в РР(2).

3.3.4. Компонент взаимодействия с криптосредствами

При криптографических операциях происходит обращение к криптоадаптеру. Криптоадаптер вызывает необходимые функции криптопровайдера.

Криптоадаптер используется как интерфейс для криптографических функций. Вся логика СКЗИ использует этот интерфейс, а он передает вызовы в КриптоПро CSP.

Функции криптоадаптера в части подписи и проверки подписи блоков и транзакций, TLS описаны в приложениях: **0, 0, 0, 0**.

3.3.5. Компонент адресации участников сетевого взаимодействия

Учёт затрат на выполнение транзакций

Вознаграждение майнера, нашедшего новый блок, включённый в РР состоит из следующих составляющих:

- За нахождение блока начисляется 5 базовых коинов;
- За каждый включенный заголовок альтернативного блока начисляется 1/32*5 базовых коинов (максимально 2 заголовка альтернативных блоков);
- Вознаграждение за выполнение транзакций, включённых в блок. Данное вознаграждение не может превышать **gasLimit** – ограничение потребления Газа на момент создания блока.
- Вознаграждение в базовых коинах начисляется на адрес майнера, нашедшего блок.

В случае, если блок содержит заголовки альтернативных блоков, то на адрес каждого из майнеров, нашедших альтернативные блоки начисляется 7/8*5 базовых коинов.

Доказательство собственности адреса

Каждый участник получает пару ключей: закрытый ключ (используется для подписи сообщений) и открытый ключ (используется для проверки правильности подписи). Последние 160-бит хэша открытого ключа являются адресом участника в сети.

Каждый блок содержит:

- Адрес майнера для начисления вознаграждения;
- Подписанный блок (майнер подписывает блок своим секретным ключом). Из подписи однозначно восстанавливается адрес, которому принадлежит секретный ключ, с помощью которого блок был подписан.

Для того чтобы сообщение было обработано, сообщение должно быть подписано легитимным ключом. Из подписи однозначно восстанавливается адрес, которому принадлежит секретный ключ, с помощью которого сообщение было подписано.

3.3.6. Компонент взаимодействия с пользователями

Интерфейс для работы с Мастерчейн реализуется через API. API реализует функционал для следующих операций:

- Удобного чтения данных из Мастерчейн;
- Разблокировки ключевых контейнеров (через API передаётся PIN);
- Создания, подписи и отправки транзакций;
- Подписи заданных данных заданным ключом (при условии наличия этого ключа в крипто-контейнере);
- Шифрования данных заданным сертификатом;
- Расшифровывание зашифрованных данных (при условии наличия нужного ключа в крипто-контейнере).

Для доступа к API используется встроенный http-сервер, который может открывать как TCP, так и IPC сокеты. Соответственно, для доступа к API нужен доступ к файловой системе ОС или к TCP сокету.

Все функции, использующие криптопреобразования обращаются к криптоадаптеру, который передает их в CSP. Более подробно функции криптоадаптера описаны – см. **0**.

Сам API на низком уровне использует JSON RPC протокол для обмена данными. Поскольку данные в JSON RPC кодируются, то для удобной работы с API используется клиентская JS библиотека web3.js, доступ к которой осуществляется с помощью JS консоли. Более подробно работа с клиентской JS библиотекой приведена – см. **0**.

3.3.7. Компонент Whitelist

Специальный смарт-контракт, который в скомпилированном виде встраивается в начальный блок распределенного реестра. Идентификатор ключа Whitelist смарт-контракта указывается при запуске СКЗИ Мастерчейн.

Whitelist содержит:

- Идентификатор ключа первого узла сети. Узел необходим для запуска сети;

- Список идентификаторов ключей узлов, их роль и время добавления идентификатора ключа узла в смарт-контракт;
- Идентификатор ключа пользователя администратора сети (АС);
- Список идентификаторов ключей пользователей администраторов Whitelist;
- Функционал для управления и получения данных, хранимых в смарт-контракте.

Whitelist используется при:

- Установлении TLS соединения между узлами;
- Валидации блоков.
- Смарт-контракт Whitelist'a допускает возможность перехода на новую версию с помощью развертывания новой сети. Переход на новый Whitelist может быть необходим, например, в случае компрометации подписи администратора сети.

Для получения данных, которые хранятся в справочнике узлов – NodeWhitelist, реализованного в комопненте Whitelist, используются следующие методы:

- возврат состояния по адресу - getMask – маска;
- получение времени добавления адреса – getCreationTime – время добавления адреса в справочник;
- получение узлов - listReaders – список адресов узлов с ролью FLAG_CAN_READ;
- получение узлов – listMiners – список адресов узлов с ролью FLAG_CAN_MINE.

Методы определены в Справочнике узлов - NodeWhitelist, комопнента Whitelist.

3.3.8. Компонент КриптоПро

Компонент КриптоПро используется для выполнения крипто-операций, таких как:

- Аутентификация и проверка целостности;
- Обеспечение конфиденциальности.

Методы КриптоПро CSP используются в алгоритмах, описанных в приложениях: **0 - 0**.

3.3.9. Программный комплекс «Сервис передачи конфеденциальных сообщений».

СКЗИ Мастерчейн включает в свой состав программный комплекс «Сервис передачи конфеденциальных сообщений» (далее – ПК СПКС).

В состав ПК СПКС входят нижеуказанные комопненты.

Компонент «Контроля и управления доступом».

Комопнент Контроля и управления доступом применяется для:

- Ведения организационной структуры участников сети. Организационная структура участников сети включает:
  - Перечень организаций - участников сети, в которых развернуты узлы Мастерчейн;
  - Перечень пользователей организаций участников сети и их прикладных ролей;
- Ведение информационных объектов, соответствующих конфеденциальным документам-сообщениям, находящихся в локальном хранилище организации;
- Ведение прав доступа для информационных объектов;
- Аутентификация и авторизация в части ведения организационной структуры участников сети.

Описание API для взаимодействия с компонентом контроля и управления доступом приведено в документе ЖТЯИ.00107-01 92 01 «СКЗИ Мастерчейн. Руководство системного программиста».

Компонент Передачи конфеденциальных сообщений.

Комопнент Передачи конфеденциальных сообщений применяется для:

- размещение кофеденциальных файлов-сообщений в локальном хранилище организации-участника сети;
- запрос и получене конфеденциальных файлов-сообщений из локуального хранилища организации-участника сети;
- запрос и получение конфеденциальных файлов-сообщений от удаленного узла СПКС;
- авторизация доступа при закпросах получения конфеденциальных файлов-сообщений.

Описание API для взаимодействия с компонентом контроля и управления доступом приведено в документе ЖТЯИ.00107-01 92 01 «СКЗИ Мастерчейн. Руководство системного программиста».

Компонент «Комопнент КриптоПро CSP».

Компонент КриптоПро используется для выполнения крипто-операций, таких как:

- Аутентификация и проверка целостности;

– Обеспечение конфиденциальности.

Методы КриптоПро CSP используются в алгоритмах, описанных в приложениях: **0 - 0**.

3.4. Связи с другими программами

СКЗИ Мастерчейн взаимодействует в части выгрузки и загрузки данных с произвольными прикладными программами.

Описание механизмов для взаимодействия с СКЗИ Мастерчейн других подсистем приведено в документе ЖТЯИ.00107-01 92 01 «СКЗИ Мастерчейн. Руководство системного программиста».

Для выполнения криптографических преобразований, соответствующих ГОСТ, применяется криптопровайдер «КриптоПро».

4. ИСПОЛЬЗУЕМЫЕ ТЕХНИЧЕСКИЕ СРЕДСТВА

Требования к техническому (аппаратному) обеспечению:

- оперативная память: не менее 16 ГБ;
- дисковое пространство: 200 Гб;
- количество ядер процессора: не менее 4 ядер с частотой не менее 700 МГц;

Для СКЗИ Мастерчейн сайзинг по процессору и ОЗУ не рассчитывается. Сайзинг по накопителям приведен ниже.

Минимальный целевой объем накопителей СКЗИ Мастерчейн (V_{min}) рассчитывается по формуле:

$V_{min} = N_1*V_1+V_2$, где

N_1 - Количество транзакций, которое вмещает СКЗИ Мачтерчейн;

V_1 - Средний объем памяти, необходимый для хранения и обработки одного блока (ГБ);

V_2 - Объем памяти для установки СКЗИ Мастерчейн в условиях пустого РР (ГБ).

Расчет сайзинга для промышленной эксплуатации по накопителям на 5 лет:

- Среднее количество блоков в день: 100;
- Средний объем памяти, необходимый для хранения и обработки одного блока: 0,003 ГБ;
- Объем памяти для установки СКЗИ Мастерчейн в условиях пустого РР: 3,033 ГБ.

$V_{min} = 365*5*100*0,003+3,033 = 550,533$ ГБ.

5. ВЫЗОВ И ЗАГРУЗКА

Вызов и загрузка СКЗИ Мастерчейн осуществляется в виде сервиса операционной системы.

Подробное описание команд приведено в документах ЖТЯИ.00107-01 91 01 «СКЗИ Мастерчейн. Руководство администратора информационной безопасности», ЖТЯИ.00107-01 91 02 «СКЗИ Мастерчейн. Руководство администратора информационной безопасности. Linux», ЖТЯИ.00107-01 91 03 «СКЗИ Мастерчейн. Руководство администратора информационной безопасности. Виртуальные среды».

6. ВХОДНЫЕ ДАННЫЕ

6.1. Характер, организация и подготовка входных данных

Входными данными для СКЗИ Мастерчейн являются:

1) Транзакции пользователей.

2) Настройки криптографических механизмов.

6.2. Формат, описание и способ кодирования входных данных

Пользователи СКЗИ Мастерчейн создают запросы на выполнение транзакций через интерфейс пользователя. Транзакция представляет собой подписанный пакет данных, который содержит сообщение от одного идентификатора ключа пользователя другому идентификатору ключа пользователя в РР. Идентификатор ключа пользователя получателя может быть, как идентификатором ключа пользователя, так и идентификатором смарт-контракта.

Транзакции бывают трёх видов:

1) Транзакция-перевод ТРЕ с одного технологического счёта на другой;
2) Транзакция-создание нового смарт-контракта;
3) Транзакция-вызов метода существующего смарт-контракта.

Транзакция содержит следующие поля:

```
"transaction" : {
"nonce" : "int",
"gasPrice" : "int",
"gasLimit" : "int",
"to" : "address"
"value" : "int",
"v" : "byte",
"r" : "256 bit unsigned int",
"s" : "256 bit unsigned int",
"data" : "byte array"
}
```

Описание полей транзакции – см. Таблица 11.

Таблица 11 – Описание полей транзакции.

№ п/п	Поле	Тип	Описание
1.	Nonce или account nonce	int	Количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора ключа смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора ключа. Необходим для предотвращения атаки повторного воспроизведения (replay attack).

№ п/п	Поле	Тип	Описание
2.	gasPrice	int	Цена за Газ (сумма вознаграждения), которую отправитель готов платить. Обычно единица Газа соответствует выполнению одной автоматической инструкции, то есть одного шага вычисления. Некоторые сложные шаги вычисления могут стоить дороже.
3.	gasLimit	int	Максимальное количество Газа, допустимое для выполнения транзакции.
4.	to	address	Идентификатор ключа пользователя получателя или идентификатор смарт-контракта.
5.	v, r, s	"v" : "byte" "r" : "256 bit unsigned int" "s" : "256 bit unsigned int"	Элементы подписи, идентифицирующей отправителя и подтверждающие его намерение отправить сообщение через РР заявленному получателю.
6.	value	int	Цена обработки транзакции в ТРЕ.
7.	data	byte array	– Произвольное сообщение ИЛИ; – Вызов функции смарт-контракта ИЛИ; – Код при создании смарт-контракта. Поле может быть пустым.

Подпись транзакции производится следующим образом:

1) Данные транзакции RLP кодируются. Данные транзакции для RLP кодирования: **nonce, gasPrice, gasLimit, to, value, data, chainId**, в конце добавляется два нуля. Описание RLP кодирования – см. **0.**

   chainId – идентификатор сети Мастерчейн: первые четыре байта Whitelist смарт-контракта.

2) От данных, полученных на шаге 1 вычисляется хеш. Хеш вычисляется по ГОСТ Р 34.11-2012 (Стрибог-256).

3) Хеш подписывается и подпись добавляется в транзакцию. Для подписи используется функция КриптоПро: crypto_csp.Sign().

Взаимодействия пользователя с СКЗИ Мастерчейн происходит с использованием надёжных криптографических функций. Криптографические функции реализуются через подключение сертифицированной криптографической библиотеке КриптоПро CSP.

Криптографическая библиотека подсоединяется через унифицированный интерфейс для использования криптографических операций.

7. ВЫХОДНЫЕ ДАННЫЕ

7.1. Характер, организация и подготовка выходных данных

Выходными данными для СКЗИ Мастерчейн является сформированный РР.

7.2. Формат, описание и способ кодирования выходных данных

Сформированнай РР содержит обработанные транзакции, сгруппированные в блоки. Каждый блок содержит математическое доказательство, которое удостоверяет содержимое блока и то, что блок является последователем предыдущего блока.

Данное математическое доказательство обладает двумя важными свойствами:

- Математическое доказательство дорого найти (время и вычислительные мощности);
- Имея математическое доказательство легко проверить целостность блока и правильность его нахождения относительно предшествующих ему блоков.

РР обладает свойством целостности данных. При наличии копии РР и СКЗИ Мастерчейн целостность РР легко проверяется.

В СКЗИ Мастерчейн предусмотрено два типа контроля целостности данных:

- Распределенный реестр.

  Контроль целостности данных распределенного реестра происходит в процессе синхронизации данных: созданные локально и полученные по сети блоки, и транзакции проходят контроль целостности и записываются узлами в локальную копию РР только в случае успешной проверки. Проверка контроля целостности включает в себя проверку последовательности блоков: проверяемый блок N должен содержать в заголовке хеш от предыдущего блока N-1. Таким образом, проверяющая сторона должна обладать копией реестра с блоком N-1, но полная проверка локальной копии реестра не производится.

- Whitelist узлов.

  Базируется на том, что Whitelist создаётся в начальном блоке РР, который распространяется по узлам в процессе запуска сети. Если он будет изменён в процессе распространения, то узел с изменённым начальным блоком РР (изменение Whitelit узлов изменит хеш блока), не сможет синхронизироваться с сетью, так как хеши N-1 блоков будут различаться.

Каждый блок в РР состоит из трёх частей (см. **Рисунок 5**):

1) Заголовок блока.

2) Список транзакций, включенных в блок, в той последовательности, в которой транзакции обрабатывались. Если нет транзакций в очереди на обработку, тогда список транзакций в блоке будет пустым.

3) Список заголовков альтернативных блоков (список может быть пустым). Если нет альтернативных блоков, список заголовков альтернативных блоков будет пустым.

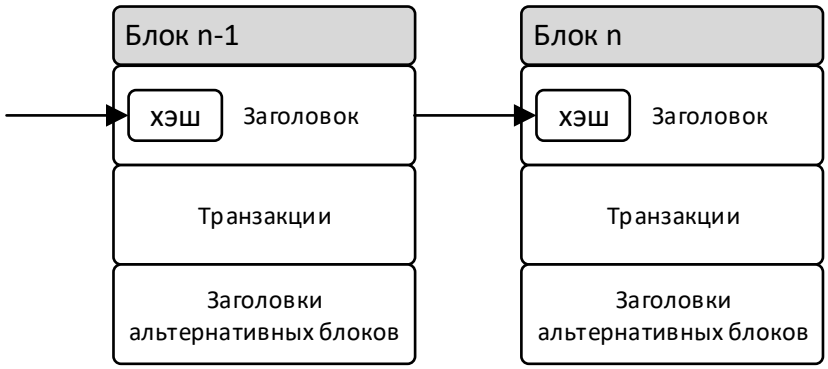


Рисунок 5 – Схема блоков.

7.2.1. Заголовок блока

Описание полей заголовка бока – см. Таблица 12.

Таблица 12 – Описание полей заголовка блока.

№ п/п	Поле	Описание
1.	parentHash	256-битный хеш предшествующего (родительского) блока (хеш считается без учёта подписи блока, то есть в поле extraData учитываются только первые 32 байта).
2.	ommersHash (или sha3Uncles или uncleHash)	256-битный хеш списка заголовков альтернативных блоков, включённых в блок.
3.	beneficiary (или coinbase, или miner)	160-битный идентификатор ключа пользователя, на который переводится вознаграждение за создание блока.
4.	stateRoot	256-битный хеш корневого узла Merkle Patricia дерева состояния системы, после того как все транзакции обработаны и финализированны.
5.	transactionsRoot	256-битный хеш корневого узла Merkle Patricia дерева транзакций, каждый лист (конечный узел) этого дерева – транзакция из списка транзакций данного блока.
6.	receiptsRoot	256-битный хеш корневого узла Merkle Patricia дерева квитанций, каждый лист (конечный узел) этого дерева – квитанция (подтверждение транзакции) для каждой транзакции из списка транзакций данного блока.

№ п/п	Поле	Описание
7.	logsBloom (или bloom)	Фильтр Блума, составленный из индексированной информации (идентификатор ключа пользователя и заголовки логов), которая есть в каждой лог-записи в квитанции для каждой транзакции из списка блока.
8.	difficulty	Скалярное значение, уровень сложности блока. Значение может быть вычислено из уровня сложности предыдущего (родительского) блока и timestamp.
9.	number	Скалярное значение, порядковый номер блока (начальный блок PP имеет значение 0).
10.	gasLimit	Скалярное значение, текущему лимиту потребления Газа за блок.
11.	gasUsed	Скалярное значение, весь Газ, использованный для обработки транзакций, включённых в блок.
12.	timestamp	Скалярное значение, время вычисления блока (Unix функции time() в начале вычисления блока).
13.	extraData (или extra)	Произвольный массив байтов, содержащий информацию, имеющую отношение к этому блоку. В оригинальной версии размер 32 байта, для СКЗИ Мастерчейн значение увеличено до 97 байт. Первые 32 байта содержат информацию, касающуюся блока, последующие 65 байта содержат подпись хеша заголовка данного блока (хеш считается от заголовка, в котором указаны только 32 байта extraData), подписывает майнер, нашедший этот блок.
14.	mixHash	256-битный хеш, доказывающий (совместно с nonce), что было произведено достаточное количество вычислений для этого блока.
15.	nonce	64-битный хеш, доказывающий (совместно с mixHash), что было произведено достаточное количество вычислений для этого блока.

Подпись заголовка блока производится следующим образом:

1) Данные заголовка блока RLP кодируются. Данные заголовка блока для RLP кодирования: **parentHash, uncleHash, coinbase, stateRoot, transactionsRoot, receiptRoot, bloom, difficulty, number, gasLimit, gasUsed, timestamp** и первых 32 байта поля **extraData**. Описание RLP кодирования – см. **0.**
2) От данных, полученных на шаге 1 вычисляется хеш. Хеш вычисляется по ГОСТ Р 34.11-2012 (Стрибог-256).
3) Хеш подписывается и подпись добавляется в последние 65 байт поля **extraData**. Для подписи используется функция КриптоПро: crypto_csp.Sign().

Шаги 1-2 одинаковы при получении:

– хеша заголовка родительского блока (при создании нового блока и при валидации блока);

– хеша для подписи заголовка блока (при создании нового блока и при валидации блока).

В каждый заголовок блока входит три хеша корневых узлов Merkle Patricia деревьев:

– **stateRoot** – хеш корневого узла Merkle Patricia дерева состояний системы;

– **transactionsRoot** – хеш корневого узла Merkle Patricia дерева тарнзакций;

– **receiptsRoot** – хеш корневого узла Merkle Patricia дерева квитанций.

Хеш берется от байтовой конкатинации ключа и значения корневого узла соответствующего дерева.

7.2.2. Список транзакций

Эта часть блока содержит список транзакций, включенных в данный блок, в той последовательности, в которой транзакции обрабатывались (список может быть пустым).

Структура транзакции, включённой в блок, немного отличается от структуры транзакции при создании – см. Таблица 13.

Таблица 13 – Описание полей транзакции, включенной в блок.

№ п/п	Поле	Тип	Описание
1.	nonce или account nonce	int	Количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора ключа смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора ключа. Необходим для предотвращения атаки повторного воспроизведения (replay attack).
2.	gasPrice	int	Цена за Газ (сумма вознаграждения), которую отправитель согласился платить. Обычно единица Газа соответствует выполнению одной автоматической инструкции? то есть одного шага вычисления. Некоторые сложные шаги вычисления могут стоить дороже.
3.	hash	string	Хеш данной транзакции. Вычисляется по ГОСТ Р 34.11-2012 (Стрибог-256).
4.	to	address	Идентификатор ключа пользователя получателя или идентификатор смарт-контракта, функции которого вызываются.
5.	v, r, s	"v" : "byte", "r" : "256 bit unsigned int", "s" : "256 bit unsigned int"	Элементы подписи, идентифицирующей отправителя и подтверждающие его намерение отправить сообщение через распределенный реестр заявленному получателю.
6.	gas	int	Газ, потраченный на выполнение данной транзакции.

№ п/п	Поле	Тип	Описание
7.	value	int	Цена обработки транзакции в TPE.
8.	input	byte array	– Произвольное сообщение ИЛИ; – Вызов функции смарт-контракта ИЛИ; – Код при создании смарт-контракта. Поле может быть пустым

Квитанции

Для каждой транзакции, включаемой в блок, создаётся квитанция. Квитанция – это подтверждение того, что транзакция была обработана и включена в блок. Структура квитанции – см. Таблица 14.

Таблица 14 – Описание полей квитанции.

№ п/п	Поле	Тип	Описание
1.	post-transaction state		Состояние системы после выполнения транзакции.
2.	cumulative gas used	int	Общее количество Газа, использованного в блоке сразу после выполнения этой транзакции.
3.	logs		Журнал логов, полученный при обработке транзакции. Каждая запись в журнале логов состоит из: – идентификатора ключа пользователя или идентификатора смарт-контракта с которого было вызвано событие; – серии заголовков логов (каждый заголовок 32 байта); – данных.
4.	Bloom filter	256-битный хеш	Фильтр Блума составленный из журнала логов. Данная информация включается в заголовок блока.

Квитанции не хранятся в PP. Квитанция получается при последовательной обработке всех транзакций, включенных в PP, каждым узлом Мастерчейн и сохраняется локально на диск. Каждая квитанция помещается в Merkle Patricia дерево, каждый лист этого дерева – квитанция.

Функция JSON RPC API eth.getTransactionReceipt возвращает всю возможную информацию о транзакции, включая информацию о блоке, содержащем эту транзакцию.

7.2.3. Список заголовков альтернативных блоков

Для каждого блока, включённого в РР может быть создано более одного блока-потомка, но только один блок-потомок будет включён в РР. Таким образом, альтернативный блок – это блок, чей родительский блок включен в РР, но сам блок не был включен в РР.

Структура заголовка альтернативного блока идентична структуре заголовка блока, который включает в себя альтернативный блок.

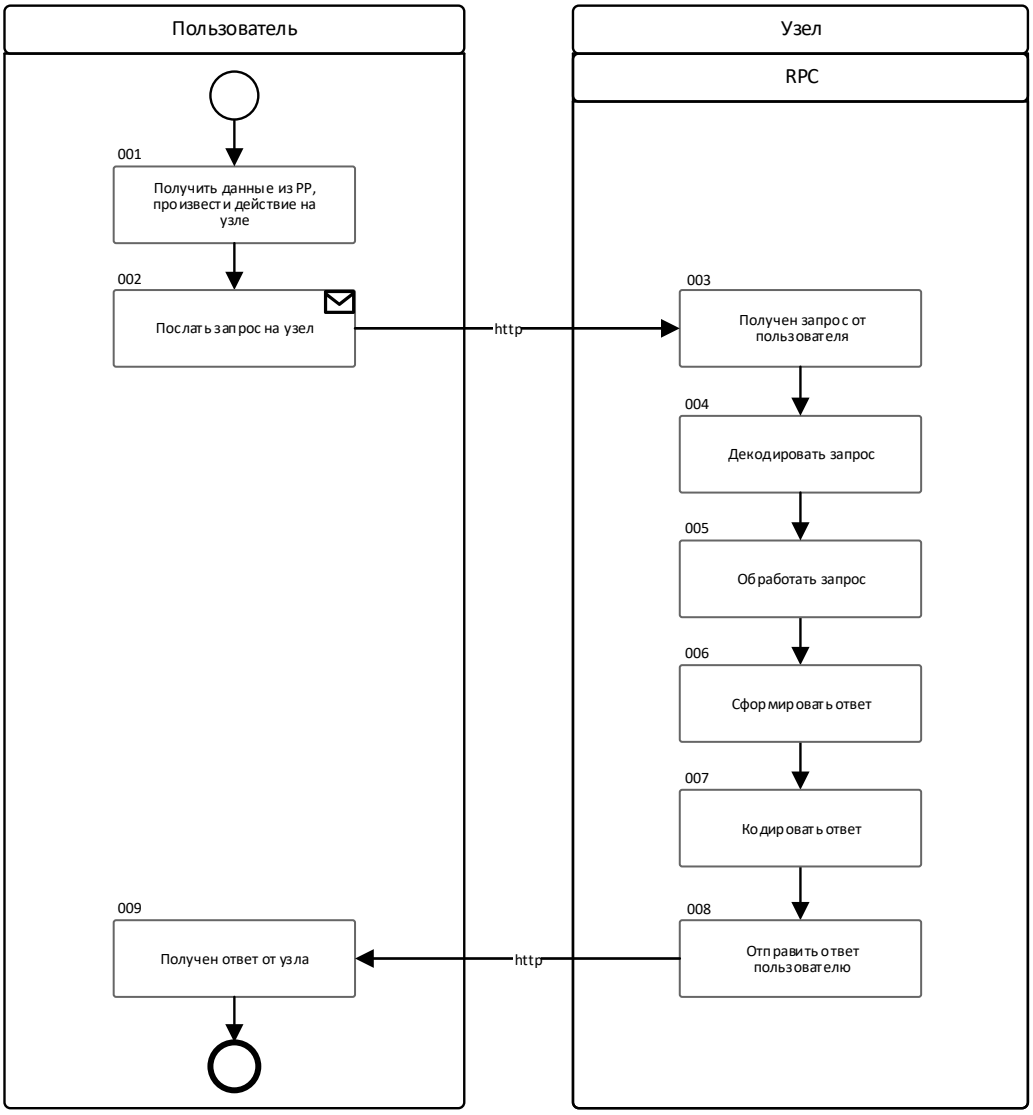
Ограничения на включения альтернативных блоков: блок может включать максимум 2 заголовка альтернативных блока. Включённые альтернативные блоки должны быть созданы не позже 6 блоков.

Причины, для включения альтернативных блоков в РР:

- Включение заголовков альтернативных блоков повышает сложность, а с этим и безопасность РР.

- Чтобы обрабатывать транзакции быстро необходимо быстро создавать блоки, для майнера это означает, что высока вероятность обнаружения блока, который будет отвергнут сетью (такой блок называется альтернативный). Если заголовок альтернативного блока включен в статический блок (блок, который записан в РР), майнер, нашедший этот альтернативный блок, получает 7/8 от вознаграждения за статичный блок (вознаграждение за статический блок 5 TPE, таким образом, вознаграждение за альтернативный блок 4.375), то есть майнеры заинтересованы в быстром нахождении блоков.

ПРИЛОЖЕНИЕ А. ОПИСАНИЕ АЛГОРИТМА РАБОТЫ С ИСПОЛЬЗОВАНИЕМ ИНТЕРФЕЙСА RPC



Пояснения к алгоритму приведены - см. Таблица 15.

Таблица 15 – Описание алгоритма работы с использование итерфейса RPC

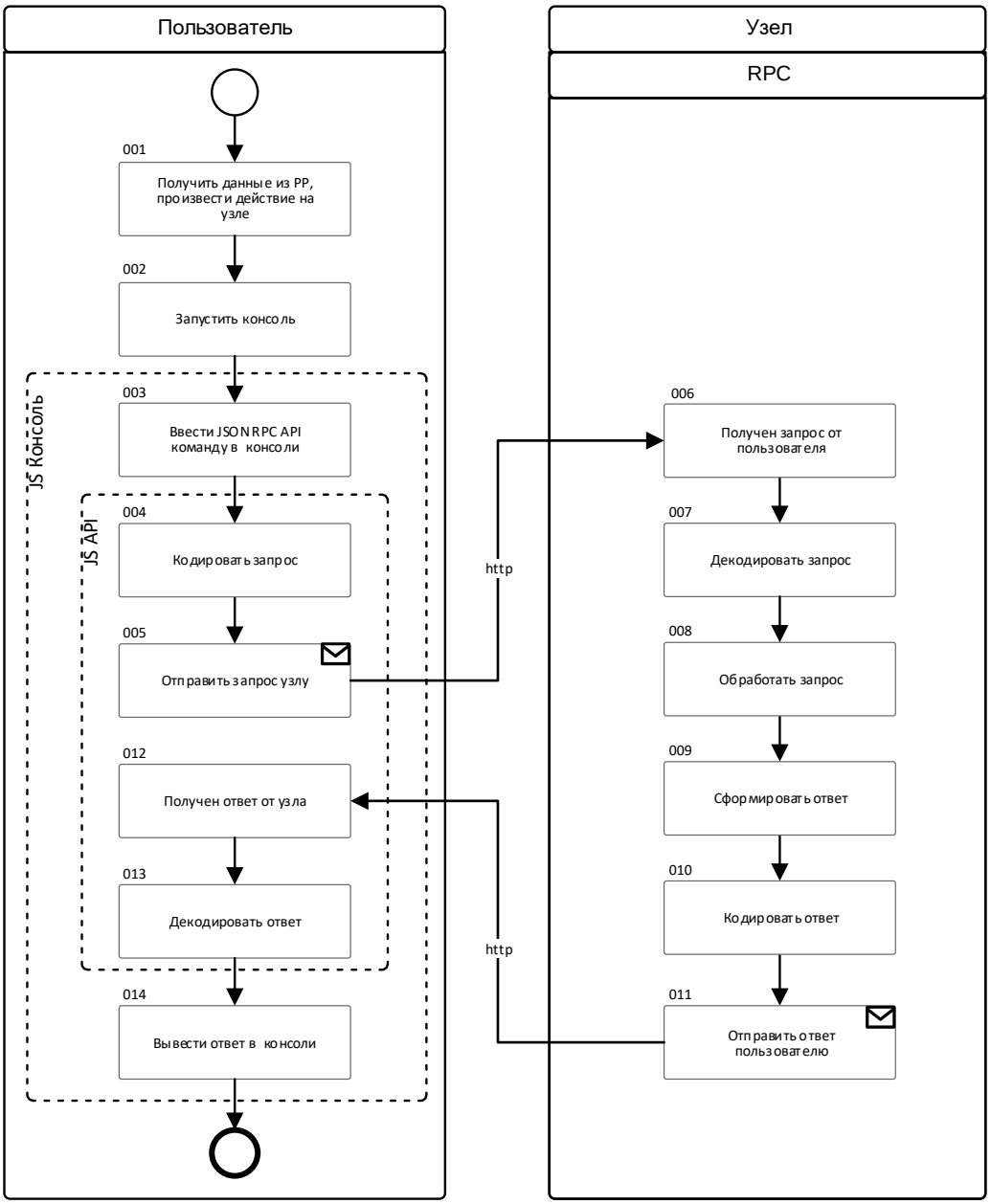
№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
1.	001	Получить данные из РР, произвести действие на узле	Пользователь может: – получить данные, хранящиеся на узле в РР (например, получить блок), – отправить транзакцию, – произвести действия с узлом (например, запустить майнинг).	Входные данные, поступившие из прикладного ПО	Данные для отправки в РР через интерфейс доступа к RPC в документе ЖТЯИ.00107-01 92 01 «СКЗИ Мастерчейн. Руководство системного программиста».	Компонент обеспечения сетевого взаимодействия между узлами РР, компонент взаимодействия с пользователями.
2.	002	Послать запрос на узел	Через JSON RPC API или через другую клиентскую библиотеку запрос посылается узлу по http	Данные для отправки в РР через интерфейс доступа к RPC, описанном в документе ЖТЯИ.00107-01 92 01 «СКЗИ Мастерчейн. Руководство системного программиста».	Данные, для отправки в РР, отправленные на узел через интерфейс доступа RPC, описанном в документе ЖТЯИ.00107-01 92 01 «СКЗИ Мастерчейн. Руководство системного программиста».	Компонент адресации участников сетевого взаимодействия, компонент взаимодействия с пользователями.
3.	003	Получен запрос от пользователя	Передача запроса пользователя на сервер через интерфейс RPC, запущенный на узле. RPC является частью СКЗИ Мастерчейн.	Данные, для отправки в РР, отправленные на узел через интерфейс доступа RPC, описанном в документе ЖТЯИ.00107-01 92 01 «СКЗИ Мастерчейн. Руководство системного программиста».	Данные для отправки в РР принятые на узле через интерфейс доступа RPC, описанном в документе ЖТЯИ.00107-01 92 01 «СКЗИ Мастерчейн. Руководство системного программиста».	Компонент взаимодействия с пользователями.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
4.	004	Декодировать запрос	Отправка запроса через интерфейс RPC для декодирования. Подробно процедура декодирования описана – см.**0**.	Данные для отправки в РР принятые на узле через интерфейс доступа RPC, описанном в документе ЖТЯИ.00107-01 92 01 «СКЗИ Мастерчейн. Руководство системного программиста».	Декодированный запрос. Состав данных описан – см.**0**.	Компонент КриптоПро.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
5.	005	Обработать запрос	Узел обрабатывает запрос через интерфейс RPC (обработка запроса описана в шагах 005-011 алгоритма работы СКЗИ Мастерчейн в части взаимодействия с пользователями – см. Рисунок 1.	Декодированный запрос. Состав данных описан – см.**0**. EVM код. Количество доступного Газа. Адрес смарт контракта, которому принадлежит исполняемый код, формат данных описан в п.3.3.3.1. Адрес отправителя транзакции, инициировавшей это выполнение - объект транзакции string from.	Блок, включенный в РР, формат данных описан в п.7.2. Данные о пройденных проверках, состав описан в п.3.3.1.4.	Компонент ведения РР, компонент обеспечения сетевого взаимодействия между узлами РР, компонент исполнения прикладных задач, компонент адресации участников сетевого взаимодействия, компонент взаимодействия с пользователями.
6.	006	Сформировать ответ	Формирование данных для ответа через интерфейс RPC.	Блок, включенный в РР, формат данных описан в п.7.2. Данные о пройденных проверках, состав описан в п.3.3.1.4.	Обработанные данные РР для ответа. Состав данных описан – см.**0**.	Компонент взаимодействия с пользователями, компонент КриптоПро.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
7.	007	Кодировать ответ	Отправка данных через интерфейс RPC для кодирования. Подробно процедура кодирования описана – см.**0**.	Обработанные данные РР для ответа. Состав данных описан – см.**0**.	Кодированный ответ. Состав данных описан – см.**0**.	Компонент КриптоПро.
8.	008	Отправить ответ пользователю	Отправка закодированного ответа пользователю по http через интерфейс RPC.	Кодированный ответ. Состав данных описан – см.**0**.	Отправленный кодированный ответ пользователю. Состав данных описан – см.**0**.	Компонент взаимодействия с пользователями, компонент КриптоПро.
9.	009	Получен ответ от узла	Пользователь получает ответ от узла.	Отправленный кодированный ответ пользователю. Состав данных описан – см.**0**.	Ответ пользователю через интерфейс RPC. Состав данных описан – см.**0**.	Компонент взаимодействия с пользователями.

ПРИЛОЖЕНИЕ Б. ОПИСАНИЕ АЛГОРИТМА РАБОТЫ С ИСПОЛЬЗОВАНИЕМ ИНТЕРФЕЙСА JSON RPC API



Пояснения к алгоритму приведены - см. Таблица 16.

Таблица 16 – Описание алгоритма работы с ипользованием интерфейса JSON RPC API

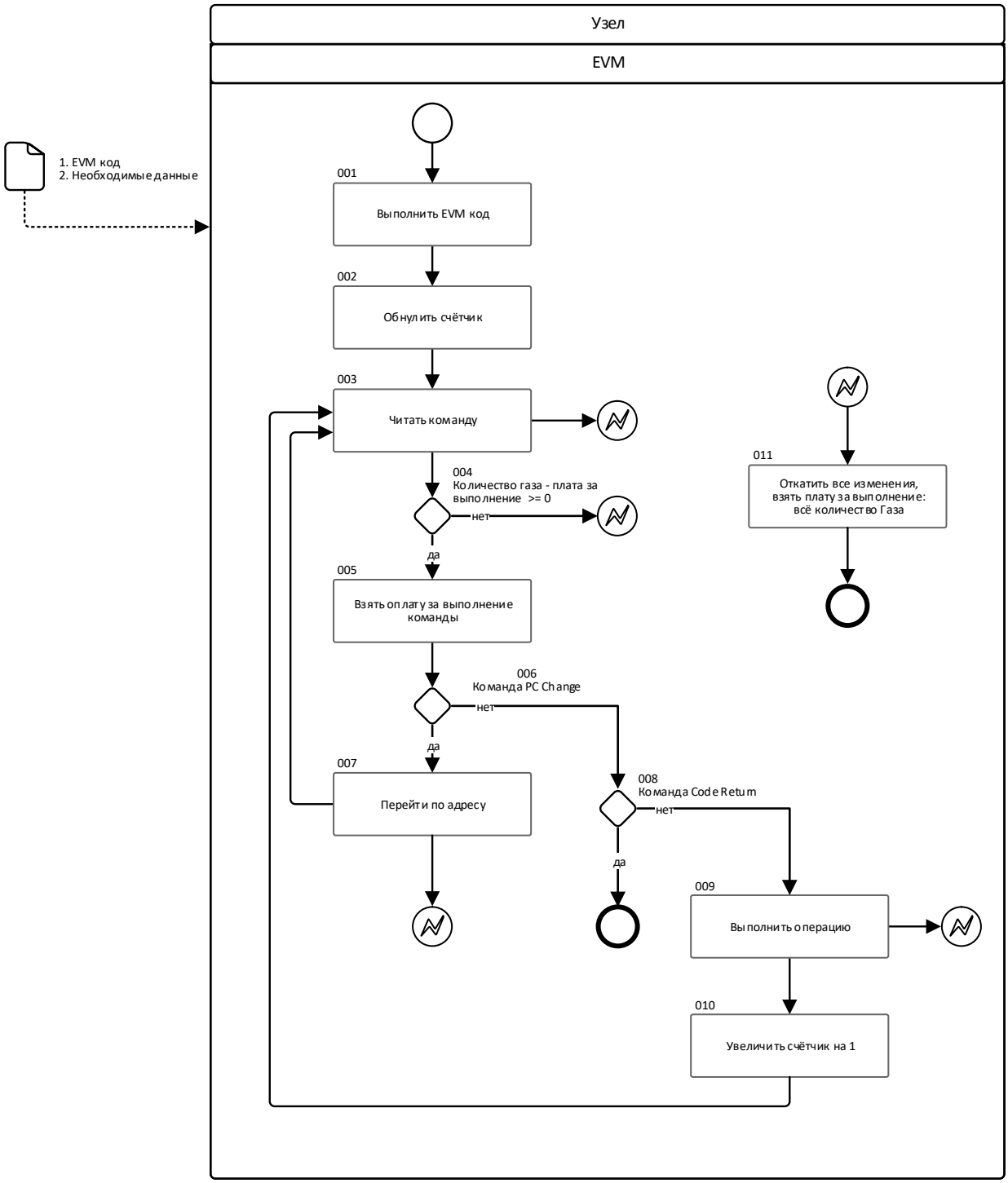
№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
1.	001	Получить данные из РР, произвести действие на узле	Пользователь может: – получить данные, хранящиеся на узле в РР (например, получить блок), – отправить транзакцию, – произвести действия с узлом (например, запустить майнинг).	Входные данные, поступившие из прикладного ПО	Данные для отправки в РР через интерфейсы доступа к JSON API, описаны в документе ЖТЯИ.00107-01 92 01 «СКЗИ Мастерчейн. Руководство системного программиста».	Компонент обеспечения сетевого взаимодействия между узлами РР, компонент взаимодействия с пользователями.
2.	002	Запустить консоль	консоль – это часть СКЗИ Мастерчейн Консоль подсоединяется к определенному узлу	Данные для отправки в РР через интерфейсы доступа к JSON API, описаны в документе ЖТЯИ.00107-01 92 01 «СКЗИ Мастерчейн. Руководство системного программиста».	Данные для отправки в РР через интерфейсы доступа к JSON API на запущенной консоли, описаны в документе ЖТЯИ.00107-01 92 01 «СКЗИ Мастерчейн. Руководство системного программиста».	Компонент взаимодействия с пользователями.
3.	003	Ввести JSON API команду в консоли	Пользователь вводит команды JSON RPC API в консоли.	Данные для отправки в РР через интерфейсы доступа к JSON API на запущенной консоли, описаны в документе ЖТЯИ.00107-01 92 01 «СКЗИ Мастерчейн. Руководство системного программиста».	Данные для отправки в РР, записанные в команду подключения к консоли, в зависимости от интерфейса, описаны в документе ЖТЯИ.00107-01 92 01 «СКЗИ Мастерчейн. Руководство системного программиста».	Компонент адресации участников сетевого взаимодействия, компонент взаимодействия с пользователями.
4.	004	Кодировать запрос	Кодирование полученной команды через консольный интерфейс библиотеки JSON API	Данные для отправки в РР, записанные в команду подключения к консоли, в зависимости от интерфейса,	Кодированный запрос. Состав данных описан – см.**0**.	Компонент КриптоПро.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
			Подробно процедура кодирования описана – см.**0**.	описаны в документе ЖТЯИ.00107-01 92 01 «СКЗИ Мастерчейн. Руководство системного программиста».		
5.	005	Отправить запрос узлу	Отправка закодированной команды через консольный интерфейс библиотеки JSON API узлу по http.	Кодированный запрос. Состав данных описан – см.**0**.	Кодированный запрос, отправленный на узел. Состав данных описан – см.**0**.	Компонент взаимодействия с пользователями, компонент КриптоПро.
6.	006	Получен запрос от пользователя	Передача запроса пользователя на сервер через интерфейс JSON RPC, запущенный на узле. JSON RPC является частью СКЗИ Мастерчейн.	Кодированный запрос, отправленный на узел. Состав данных описан – см.**0**.	Кодированный запрос, принятый на узле. Состав данных описан – см.**0**.	Компонент взаимодействия с пользователями, компонент КриптоПро.
7.	007	Декодировать запрос	Отправка запроса через интерфейс JSON RPC для декодирования. Подробно процедура декодирования описана – см.**0**.	Кодированный запрос, принятый на узле. Состав данных описан – см.**0**.	Декодированный запрос. Состав данных описан – см.**0**.	Компонент КриптоПро.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
8.	008	Обработать запрос	Узел обрабатывает запрос через интерфейс JSON RPC (обработка запроса описана в шагах 005-011 алгоритма работы СКЗИ Мастерчейн в части взаимодействия с пользователями – см. Рисунок 1.	Декодированный запрос. Состав данных описан – см.**0**. EVM код. Количество доступного Газа. Адрес смарт контракта, которому принадлежит исполняемый код, формат данных описан в п.3.3.3.1. Адрес отправителя транзакции, инициировавшей это выполнение - объект транзакции string from.	Блок, включенный в РР, формат данных описан в п.7.2. Данные о пройденных проверках, состав описан в п.3.3.1.4.	Компонент ведения РР, компонент обеспечения сетевого взаимодействия между узлами РР, компонент исполнения прикладных задач, компонент адресации участников сетевого взаимодействия, компонент взаимодействия с пользователями.
9.	009	Сформировать ответ	Формирование данных для ответа через интерфейс RPC.	Блок, включенный в РР, формат данных описан в п.7.2. Данные о пройденных проверках, состав описан в п.3.3.1.4.	Обработанные данные РР для ответа. Состав данных описан – см.**0**.	Компонент взаимодействия с пользователями, компонент КриптоПро.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
10.	010	Кодировать ответ	Отправка данных через интерфейс RPC для кодирования. Подробно процедура кодирования описана – см.**0**.	Обработанные данные РР для ответа. Состав данных описан – см.**0**.	Кодированный ответ. Состав данных описан – см.**0**.	Компонент КриптоПро.
11.	011	Отправить ответ пользователю	Отправка закодированного ответа пользователю по http через интерфейс RPC.	Кодированный ответ. Состав данных описан – см.**0**.	Отправленный кодированный ответ пользователю. Состав данных описан – см.**0**.	Компонент взаимодействия с пользователями, компонент КриптоПро.
12.	012	Получен ответ от узла	Получение ответа от узла через консольный интерфейс библиотеки JSON RPC по http.	Отправленный кодированный ответ пользователю. Состав данных описан – см.**0**.	Кодированный ответ пользователю. Состав данных описан – см.**0**.	Компонент взаимодействия с пользователями, компонент КриптоПро.
13.	013	Декодировать ответ	Декодирование полученного от узла ответа через консольный интерфейс библиотеки JSON API. Подробно процедура декодирования описана – см.**0**.	Кодированный ответ пользователю. Состав данных описан – см.**0**.	Декодированный ответ пользователю. Состав данных описан – см.**0**.	Компонент КриптоПро.
14.	014	Вывести ответ в консоли	Декодированный ответ выводится в консоли.	Декодированный ответ пользователю. Состав данных описан – см.**0**.	Декодированный ответ пользователю, выведенный в консоли. Формат данных описан – см.**0**.	Компонент взаимодействия с пользователями.

ПРИЛОЖЕНИЕ В. ОПИСАНИЕ АЛГОРИТМА ВЫПОЛНЕНИЯ КОДА В EVM



Пояснения к алгоритму приведены - см. Таблица 17.

Таблица 17 – Описание алгоритма выполнения EVM кода

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
1.	001	Выполнить EVM код	На вход подаётся EVM код и данные необходимые для его выполнения: – Состояние системы (номер и заголовок текущего блока, **timestamp**). Данное поле описано в п.12 – см. Таблица 12; – Газ доступный для выполнения кода; – Адрес смарт-контракта, которому принадлежит исполняемый код; – Адрес отправителя транзакции, инициировавшей это выполнение; – Цена Газа, указанная в транзакции, инициировавшей это выполнение; – Данные, необходимые для выполнения кода (в случае кода смарт-контракта – это входные значения для вызываемых функций	EVM код. Количество доступного Газа. Адрес смарт контракта, которому принадлежит исполняемый код, формат данных описан в п.3.3.3.1. Адрес отправителя транзакции, инициировавшей это выполнение - объект транзакции string from.	Транзакция, формат данных описан в п.6.2. EVM код.	Компонент исполнения прикладных задач.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
			– Адрес, с которого вызвано это выполнение (в случае транзакции этот адрес совпадает с адресом отправителя транзакции, инициировавшей это выполнение).			
2.	002	Обнулить счетчик	В начале обработки EVM кода счётчик устанавливается на начало EVM кода.	EVM код	Счетчик смещения от начала EVM кода.	Компонент исполнения прикладных задач.
3.	003	Читать команду	Читать команду из EVM кода, на которую указывает счётчик.	EVM код. Счетчик смещения от начала EVM кода	Прочитанные комады PC Change, Code Return	Компонент исполнения прикладных задач.
4.	004	Количество Газа – плата за выполнение команды ≥0	Проверка, может ли быть оплачено выполнение команды.	Количество доступного Газа. Стоимость выполняемой команды.	Результаты проверки того, что количество доступного Газа ≥0.	Компонент исполнения прикладных задач.
5.	005	Взять оплату выполнения команды	Уменьшить количество Газа, доступного для выполнения, на стоимость выполнения команды.	Положительные результаты проверки того, что количество доступного Газа ≥0.	Идентификатор адреса пользователя. Количество оставшегося Газа после вычета Газа за выполнение команды.	Компонент исполнения прикладных задач.

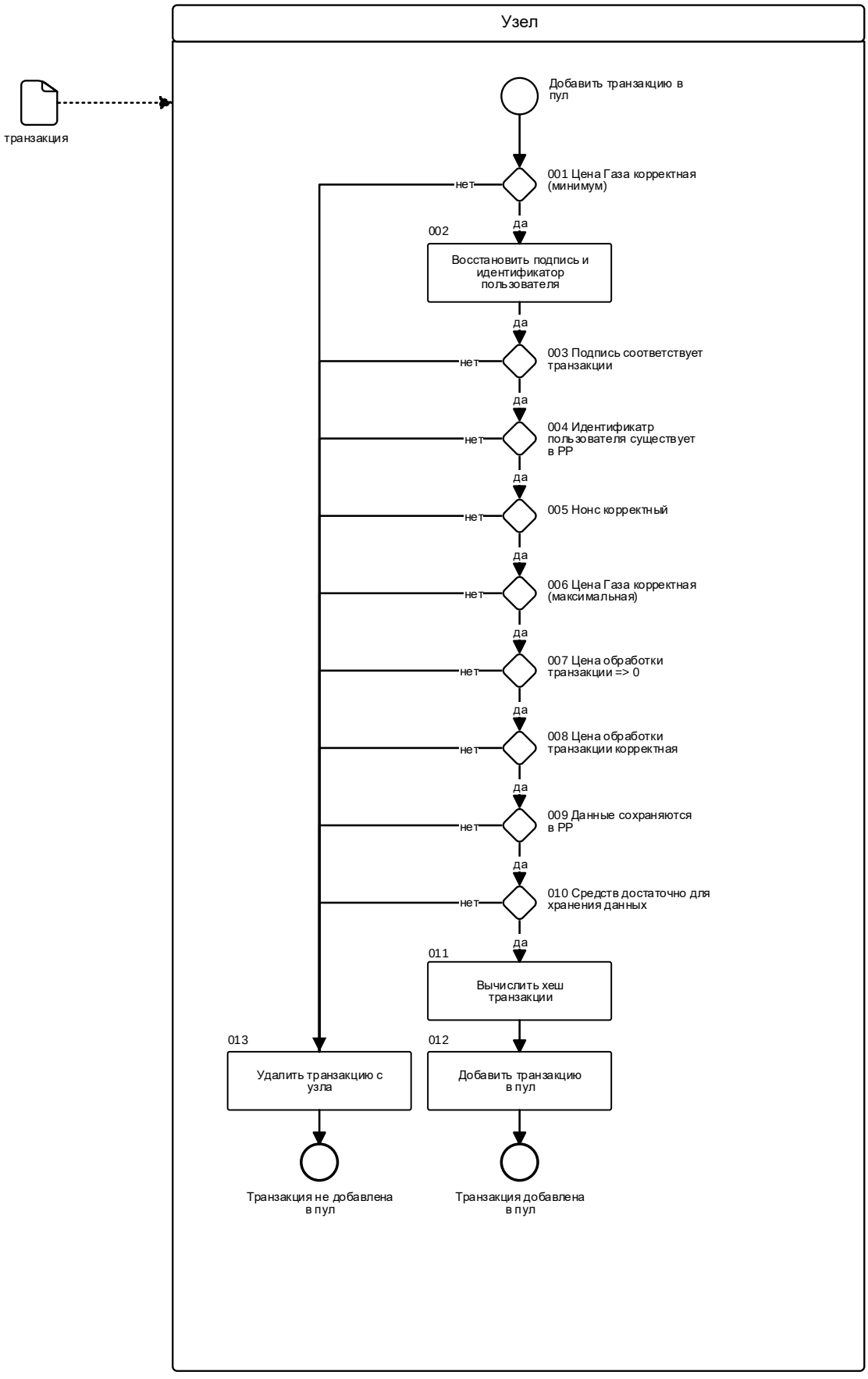
№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
				Адрес пользователя, инициировавшего транзакцию, Количество доступного Газа. Количество газа за выполнение команды		
6.	006	Команда PC Change	Прочитанная на шаге 003 команда PC Change (Program Counter Change).	Состояние счетчика смещения.	Состояние счетчика смещения после исполнения команды PC Change.	Компонент исполнения прикладных задач.
7.	007	Перейти по адресу	Производится переход по указанному в команде адресу.	Состояние счетчика смещения после исполнения команды PC Change.	Команда по адресу.	Компонент исполнения прикладных задач.
8.	008	Команда Code Return	Прочитанная на шаге 003 команда Code Return. Только в этом случае EVM код полностью выполнен без возникновения исключений.	Команда по адресу.	Результат исполнения команды Code Return. Транзакции, формат данных описан в п.6.2.	Компонент исполнения прикладных задач.
9.	009	Выполнить операцию	Выполнить операцию, прочитанную на шаге 003.	Комады PC Change, Code Return.	Результаты исполнения команд PC Change, Code Return.	Компонент исполнения прикладных задач.
10.	010	Увеличить счетчик на 1	Счётчик увеличивается на 1.	Состояние счетчика до изменения.	Состояние счетчика после изменения.	Компонент исполнения прикладных задач.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
11.	011	Откатить все изменения, взять плату за выполнение: все количество Газа	При возникновении исключения все изменения, произведённые с начала выполнения EVM кода, откатываются и с отправителя транзакции взимается плата за выполнение EVM кода в размере количества Газа, определённого на начало выполнения EVM кода.	Количество Газа, на момент начала выполнения EVM кода. Технологический счёт отправителя транзакции (поля **balance, nonce, storageRoot, codeHash**), где: – **balance** - количество TPE на данном технологическом счёте;	Состояние системы, до начала выполнения EVM кода. Блок, PP, формат данных описан в п.7.2. Технологический счёт отправителя транзакции после вычета платы за выполнение EVM кода (поля **balance, nonce, storageRoot, codeHash**), где: – **balance** - количество TPE на данном технологическом счёте;	Компонент исполнения прикладных задач.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
				– **nonce** или account nonce - количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора смарт-контракта. Данное поле необходимо для предотвращения атаки повторного воспроизведения (replay attack);	– **nonce** или account nonce - количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора смарт-контракта. Данное поле необходимо для предотвращения атаки повторного воспроизведения (replay attack);	

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
				– **storageRoot** - хеш корневого элемента Merkle Patricia дерева, которое кодирует содержимое хранилища данного технологического счёта. Хеш берется от байтовой конкатинации ключа и значения корневого узла; – **codeHash** – хеш EVM кода данного технологического счёта (в случае смарт-контракта).	– **storageRoot** - хеш корневого элемента Merkle Patricia дерева, которое кодирует содержимое хранилища данного технологического счёта. Хеш берется от байтовой конкатинации ключа и значения корневого узла; – **codeHash** – хеш EVM кода данного технологического счёта (в случае смарт-контракта).	

ПРИЛОЖЕНИЕ Г. ОПИСАНИЕ АЛГОРИТМА ДОБАВЛЕНИЕ НОВОЙ ТРАНЗАКЦИИ В ПУЛ



Пояснения к алгоритму приведены – см. Таблица 18.

Таблица 18 – Описание алгоритма добавления новой транзакции в пул.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
1.	001	Цена Газа корректная (минимум)	Значение в поле **gasPrice** больше или равно минимальному значению цены на Газ, указанному для пула транзакций (пул транзакций и его значения создаётся при установке СКЗИ Мастерчейн). Данное поле описано в п.2 – см. Таблица 11.	Транзакция, подписанная отправителем, формат данных описан в п.6.2. Минимальное значение цены на Газ.	Транзакция, подписанная отправителем, формат данных описан в п.6.2. Результаты проверки минимума цены на Газ.	Компонент ведения РР.
2.	002	Восстановить подпись и идентификатор пользователя	Восстановить подпись и идентификатор ключа пользователя из полей **v, r, s.** Данные поля описаны в п.5 – см. Таблица 11.	Транзакция, подписанная отправителем, прошедшая проверку минимума цены на Газ. Формат данных описан в п.6.2.	Транзакция, подписанная отправителем, формат данных описан в п.6.2. Подпись и идентификатор ключа пользователя.	Компонент адресации участников сетевого взаимодействия, компонент взаимодействия с пользователями.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
3.	003	Подпись соответствует транзакции	Подпись, восстановленная на шаге 002, соответствует транзакции.	Транзакция, подписанная отправителем, формат данных описан в п.6.2. Подпись.	Транзакция, подписанная отправителем, формат данных описан в п.6.2. Результаты проверки подписи на соответствие транзакции.	Компонент адресации участников сетевого взаимодействия, компонент взаимодействия с пользователями.
4.	004	Идентификатор пользователя существует в РР	Идентификатор ключа пользователя отправителя, восстановленный на шаге 002, существует в РР.	Транзакция, подписанная отправителем, прошедшая проверку подписи, формат данных описан в п.6.2. Идентификатор ключа пользователя. Данные РР.	Транзакция, подписанная отправителем, формат данных описан в п.6.2. Результаты проверки существования идетификатора пользователя в РР.	Компонент адресации участников сетевого взаимодействия, компонент взаимодействия с пользователями.
5.	005	Нонс корректный	Значение в поле **nonce** больше нонса технологического счёта списания. Данное поле описано в п.1 – см. Таблица 11.	Транзакция, подписанная отправителем, прошедшая проверку существования идентификатора пользователя в РР. Формат данных описан в п.6.2. Технологический счёт (поля **balance, nonce, storageRoot, codeHash**), где:	Транзакция, подписанная отправителем, формат данных описан в п.6.2. Результаты проверки корректности нонса.	Компонент ведения РР.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
				– **balance** - количество TPE на данном технологическом счёте; – **nonce** или account nonce - количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора смарт-контракта. Данное поле необходимо для предотвращения атаки повторного воспроизведения (replay attack); – **storageRoot** - хеш корневого элемента Merkle Patricia дерева, которое кодирует содержимое хранилища данного технологического счёта. Хеш берется от байтовой конкатинации ключа и значения корневого узла; – **codeHash** – хеш EVM кода данного технологического счёта (в случае смарт-контракта).		

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
6.	006	Цена Газа корректная (максимальная)	Значение в поле **gasLimit** меньше или равно максимальному значению, указанному для пула транзакций (пул транзакций и его значения создаётся при установке СКЗИ Мастерчейн). Данное поле описано в п.3 – см. Таблица 11.	Транзакция, подписанная отправителем, прошедшая проверку корректности нонса. Формат данных описан в п.6.2. Максимальное значение цены на газ.	Транзакция, подписанная отправителем, формат данных описан в п.6.2. Результаты проверки максимума цены на Газ.	Компонент ведения РР.
7.	007	Цена обработки транзакции ≥0	Цена обработки транзакции (поле **value**) больше или равна 0 (значение не отрицательное). Данное поле описано в п.6 – см. Таблица 11.	Транзакция, прошедшая проверку максимума цены на Газ, формат данных описан в п.6.2.	Транзакция, подписанная отправителем, формат данных описан в п.6.2. Результаты проверки цены обработки транзакции.	Компонент ведения РР.

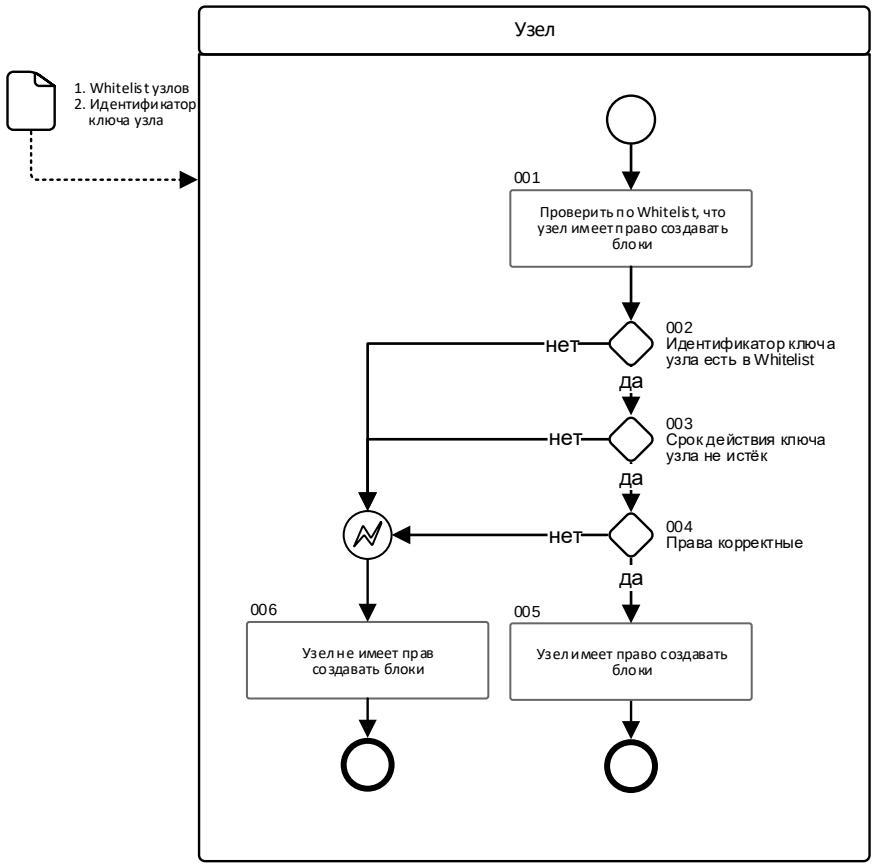
№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
8.	008	Цена обработки транзакции корректная	Баланс технологического счёта списания больше или равен цене обработки транзакции + **gasLimit**. Данное поле описано в п.3 – см. Таблица 11.	Транзакция, подписанная отправителем, прошедшая процерку цены обработки. Формат данных описан в п.6.2. Технологический счёт (поля **balance, nonce, storageRoot, codeHash**), где: – **balance** - количество ТРЕ на данном технологическом счёте; – **nonce** или account nonce - количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора смарт-контракта. Данное поле необходимо для предотвращения атаки повторного воспроизведения (replay attack); – **storageRoot** - хеш корневого элемента Merkle Patricia дерева, которое кодирует содержимое хранилища данного технологического счёта. Хеш берется от байтовой конкатинации ключа и значения корневого узла;	Транзакция, подписанная отправителем, формат данных описан в п.6.2. Результаты проверки корректности цены обработки транзакции.	Компонент ведения РР, компонент адресации участников сетевого взаимодействия, компонент взаимодействия с пользователями

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
				– **codeHash** – хеш EVM кода данного технологического счёта (в случае смарт-контракта).		
9.	009	Данные сохраняются в РР	Поле **data** содержит данные. Данное поле описано в п.7 – см. Таблица 11.	Транзакция, подписанная отправителем, формат данных описан в п.6.2. Положительные результаты проверки корректности цены обработки транзакции.	Транзакция, подписанная отправителем, формат данных описан в п.6.2. Результаты проверки сохранения данных в РР.	Компонент ведения РР.
10.	010	Средств достаточно для хранения данных	Проверяется достаточно ли средств на балансе технологического счёта списания для хранения этих данных в РР.	Положительные результаты проверки сохранения данных в РР. Технологический счёт (поля **balance, nonce, storageRoot, codeHash**), где: – **balance** - количество ТРЕ на данном технологическом счёте;	Результаты проверки достаточности средств для хранения данных.	Компонент адресации участников сетевого взаимодействия, компонент взаимодействия с пользователями

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
				– **nonce** или account nonce - количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора смарт-контракта. Данное поле необходимо для предотвращения атаки повторного воспроизведения (replay attack); – **storageRoot** - хеш корневого элемента Merkle Patricia дерева, которое кодирует содержимое хранилища данного технологического счёта. Хеш берется от байтовой конкатинации ключа и значения корневого узла; – **codeHash** – хеш EVM кода данного технологического счёта (в случае смарт-контракта).		
11.	011	Вычислить хеш транзакции	Хеш транзакции используется для проверки состояния транзакции.	Транзакция, подписанная отправителем, формат данных описан в п.6.2. Положительные результаты проверок шагов 001, 003-010.	Транзакция, подписанная отправителем, с вычесленным хешем. Формат данных описан в п.6.2.	Компонент ведения РР.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
12.	012	Добавить транзакцию в пул	Транзакция добавляется в пул транзакций на узле.	Транзакция, подписанная отправителем, с вычесленным хешем. Формат данных описан в п.6.2.	Транзакция, подписанная отправителем, добавленная в пул. Формат данных описан в п.6.2.	Компонент ведения РР.
13.	013	Удалить транзакцию с узла	Транзакция не корректна и не может быль добавлена в пул. Транзакция удаляется с узла.	Отрицательные результаты проверок хотя бы одного из шагов 001, 003-010.	Удаленная транзакция формат данных описан в п.6.2.	Компонент ведения РР.

ПРИЛОЖЕНИЕ Д. ОПИСАНИЕ АЛГОРИТМА WHITELIST ПРОВЕРКА ПРИ СОЗДАНИИ БЛОКА



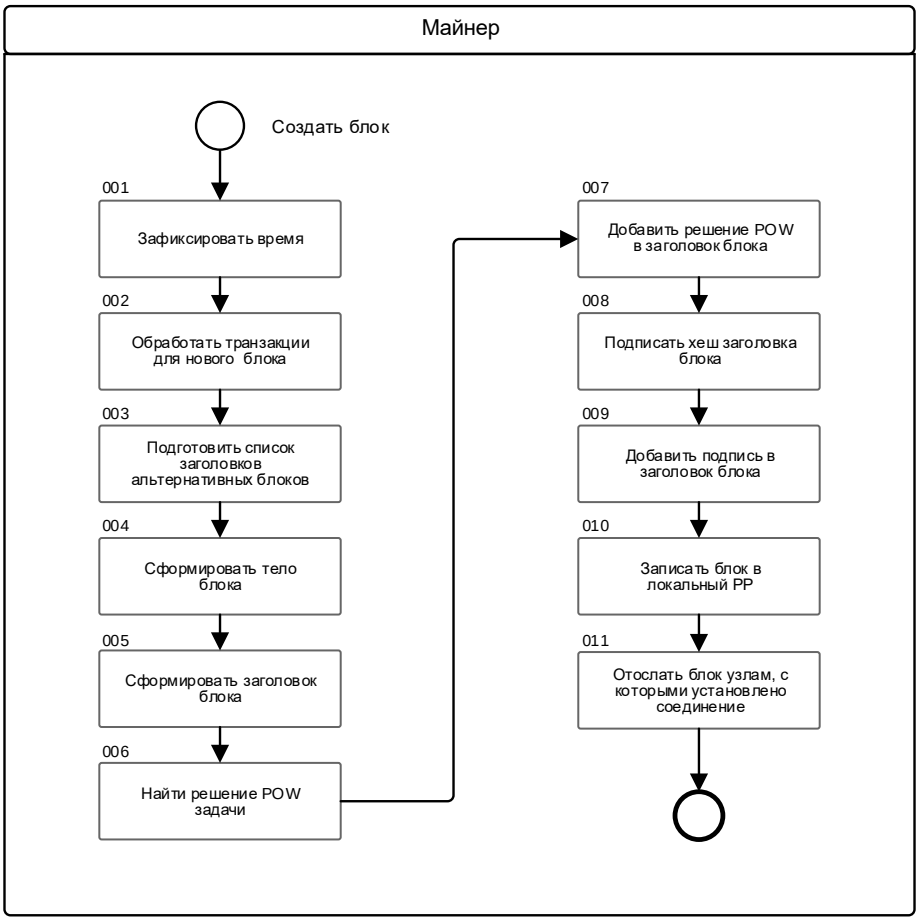
Пояснения к алгоритму приведены – см. Таблица 19.

Таблица 19 – Описание алгоритма Whitelist проверки при создании блока

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
1.	001	Проверить по Whitelist, что узел имеет право создавать блоки	Проверить по Whitelist, что узел с данным идентификатором узла имеет право создавать блоки.	Whitelist узлов, идентификатор ключа узла, формат описан в п.3.3.7. Блок, подписанный идентификатором ключа узла, формат данных описан в п.7.2.	Результаты проверки того, что узел, подписавший блок, входит в Whitelist.	Компонент Whitelist.
2.	002	Идентификатор ключа узла есть в Whitelist	Проверка методами Whitelist смарт-контракта, что идентификатор ключа узла есть в вписке узлов. Функция *isNode.*	Whitelist узлов, идентификатор ключа узла, формат описан в п.3.3.7. Блок, подписанный идентификатором ключа узла, формат данных описан в п.7.2.	Результаты проверки наличия идентификатора ключа узла в Whitelist.	Компонент Whitelist.
3.	003	Срок действия ключа узла не истёк	Проверка методами Whitelist смарт-контракта, что срок действия данного ключа не истёк. Функция *isExpired.*	Положительные результаты проверки наличия идентификатора ключа узла в Whitelist. Whitelist узлов, идентификатор ключа узла, формат описан в п.3.3.7.	Результаты проверки того, что срок действия ключа узла не истек.	Компонент Whitelist.
4.	004	Права корректные	Права корректные, если роль узла валидатор и майнер (выставлены флаги FLAG_CAN_READ и FLAG_CAN_MINE). Более подробно данные флаги описаны в документе ЖТЯИ.00107-01 91 01 «СКЗИ Мастерчейн. Руководство администратора информационной безопасности». Проверка методами Whitelist смарт-контракта, что данные флаги	Положительные результаты проверки того, что срок действия ключа узла не истек. Whitelist узлов, идентификатор ключа узла, формат описан в п.3.3.7.	Результаты проверки корректности прав.	Компонент Whitelist.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
			выставлены. Функции *isFlagSet* и getMask.			
5.	005	Узел имеет право создавать блоки	Если все проверки успешны, узел с данным идентификатором ключа имеет право создавать блоки.	Положительные результаты проверки шагов 002-004.	Локальный РР с блоком, формат данных описан в п.7.2.	Компонент Whitelist.
6.	006	Узел не имеет прав создавать блоки	Если хоть одна проверка не успешна, узел с данным идентификатором ключа не имеет прав создавать блоки.	Отрицательные результаты проверки хотя бы одного из шагов 002-004.	Блок, отвергнутый по результатам проверки, формат данных описан в п.7.2.	Компонент Whitelist.

ПРИЛОЖЕНИЕ Е. ОПИСАНИЕ АЛГОРИТМА СОЗДАНИЯ БЛОКА



Пояснения к алгоритму приведены – см. Таблица 20.

Таблица 20 – Описание алгоритма создания блока

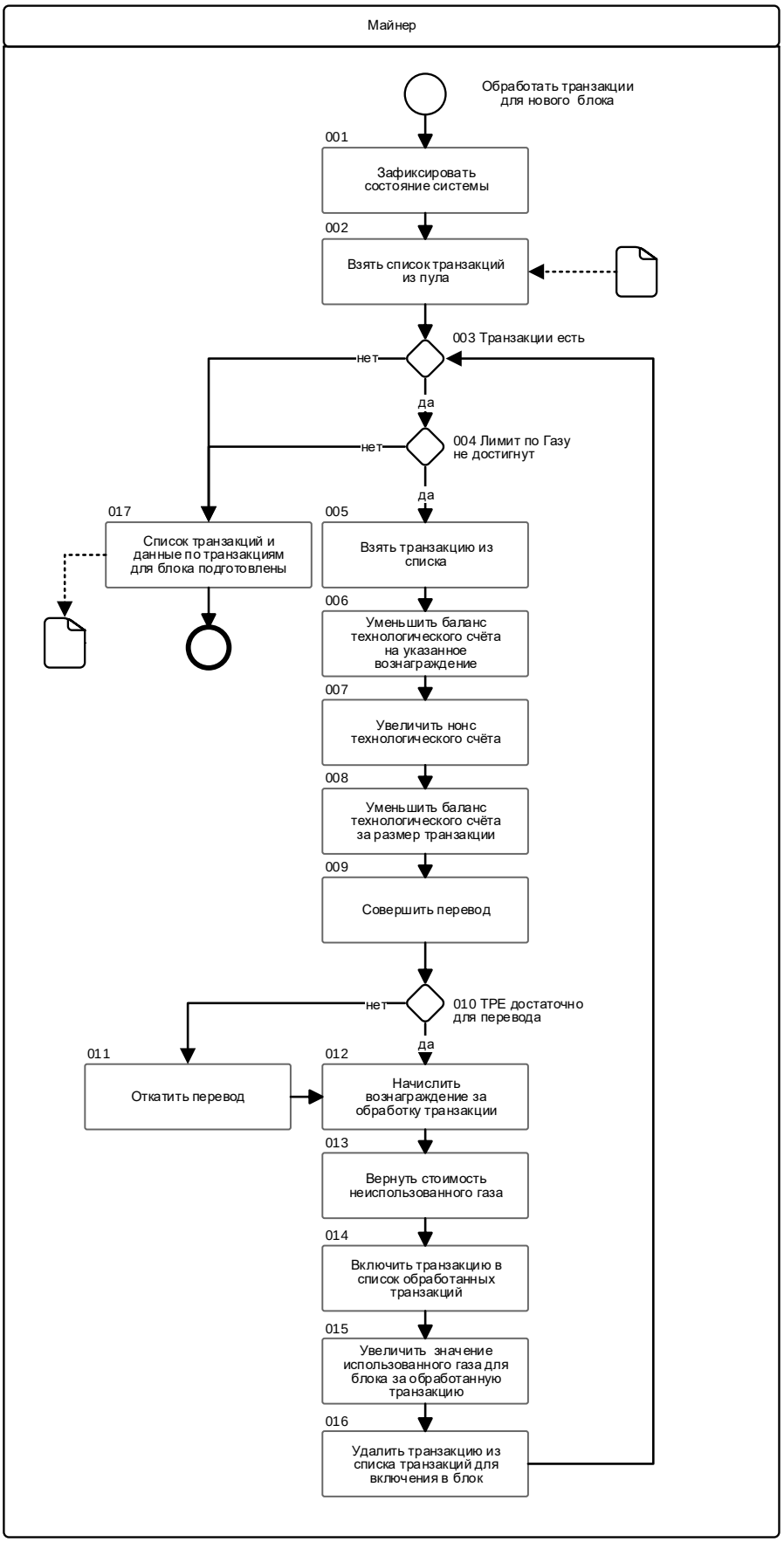
№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
1.	001	Зафиксировать время	В начале всех действий майнер фиксирует время (это время будет записано в блок и будет считаться временем создания нового блока). Берётся системное время на узле.	Данные о пройденных проверках.	Данные о системном времени на узле.	Компонент ведения РР.
2.	002	Обработать транзакции для нового блока	Майнер обрабатывает транзакции для блока по алгоритму, описанному в – см. **0** пока не достигнут лимит блока по Газу или не обработаны все транзакции из очереди pending. Результатом обработки транзакций является: – Список транзакций; – Список квитанций обработанных транзакций; – Фильтр Блума; – Новое состояние системы; – Использованный Газ.	Данные о состоянии системы, зафиксированные в поле **stateRoot** заголовка блока. Данное поле описано в п.4 – см. Таблица 12. Список транзакций из пула, формат данных описан в п.6.2.	Данные о состоянии системы, зафиксированные в поле **stateRoot** заголовка блока. Данное поле описано в п.4 – см. Таблица 12. Дерево транзакций, дерево квитанций. Формат данных описан в п.7.2.2. Количество использованного Газа.	Компонент ведения РР, компонент исполнения прикладных задач, компонент адресации участников сетевого взаимодействия.
3.	003	Подготовить список заголовков альтернативных блоков	Майнер формирует спискок заголовков альтернативных блоков для включения в блок.	Дерево транзакций, дерево квитанций. Формат данных описан в п.7.2.2.	Список заголовков альтернативных блоков, формат данных описан в п.7.2.3.	Компонент ведения РР.
4.	004	Сформировать тело блока	Майнер формирует тело блока. Тело блока состоит из списка транзакций, подготовленных на шаге 002, и списка	Список транзакций, формат данных описан в п.6.2.	Тело блока, содержащее данные в соответствии с шагами 002 и 003.	Компонент ведения РР.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
			заголовков альтернативных блоков, подготовленных на шаге 003.	Список заголовков альтернативных блоков, формат данных описан в п.7.2.3.		
5.	005	Сформировать заголовок блока	Майнер формирует заголовок блока, за исключением: – Полей **mixHash** и **nonce** (значения могут быть получены после нахождения решения POW, шаг 006); – последних 65-ти байтов поля **extraData**, в которых находится подпись блока, шаг 008, 009. Данные поля описаны в п.п.13, 14, 15 – см. Таблица 12.	Тело блока, содержащее данные в соответствии с шагами 002 и 003.	Заголовок блока, формат данных описан в п.7.2.1.	Компонент ведения РР.
6.	006	Найти решение POW задачи	Майнер ищет решение PoW-задачи с параметрами: – хеш от заголовка блока (без параметров **mixHash**, **nonce** и без 65-ти последних байтов поля **extraData**); – **mixHash**; – **nonce.** Данные поля описаны в п.п.13, 14, 15 – см. Таблица 12.	Заголовок блока до решения POW задачи, включающий значения полей **mixHash**, **nonce**, формат данных описан в п.7.2.1.	Заголовок блока в результате решения POW задачи, включающий значения полей **mixHash**, **nonce**, формат данных описан в п.7.2.1.	Компонент ведения РР.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
7.	007	Добавить решение POW в заголовок блока	Значения **mixHash** и **nonce** записываются в заголовок блока. Эти параметры являются доказательством решения POW задачи для этого конкретного блока. Данные поля описаны в п.п.14, 15 – см. Таблица 12.	Заголовок блока в результате решения POW задачи, включающий значения полей **mixHash**, **nonce**, формат данных описан в п.7.2.1.	Записанные значения полей **mixHash**, **nonce** в заголовке блока, формат данных описан в п.7.2.1.	Компонент ведения РР.
8.	008	Подписать хеш заголовка блока	Майнер подписывает хеш заголовка блока (поле заголовка **extraData** при этом не содержит последние 65 байт) ключом электронной подписи. Данное поле описано в п.13 – см. Таблица 12.	Хеш заголовка блока.	Подпись хеша заголовка блока.	Компонент ведения РР.
9.	009	Добавить подпись в заголовок блока	Подпись, полученная на шаге 008, вставляется в последние 65 байт поля заголовка **extraData.** Данное поле описано в п.13 – см. Таблица 12.	Подпись хеша заголовка блока	Записанное значение поля **extrfData** в заголовке блока, формат данных описан в п.7.2.1.	Компонент ведения РР.
10.	010	Записать блок в локальный РР	Блок записывается в локальный РР.	Блок, формат данных описан в п.7.2.	Локальный РР, с записанным в него блоком, формат данных описан в п.7.2.	Компонент ведения РР.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
11.	011	Отослать блок узлам, с которыми установлено соединение	Майнер посылает созданный блок всем узлам, с которыми установлено соединение.	Блок, записанный в локальный РР, формат данных описан в п.7.2.	Отправленный блок, записанный в локальный РР, формат данных описан в п.7.2.	Компонент ведения РР, компонент обеспечения сетевого взаимодействия между узлами РР.

ПРИЛОЖЕНИЕ Ж. ОПИСАНИЕ АЛГОРИТМА ОБРАБОТКА ТРАНЗАКЦИЙ ДЛЯ ВКЛЮЧЕНИЯ В БЛОК



Пояснения к алгоритму приведены – см. Таблица 21.

Таблица 21 – Описание алгоритма обработки транзакции для включения в блок

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
1.	001	Зафиксировать состояние системы	В начале обработки транзакций майнер фиксирует состояние системы, которое соответствует состоянию системы, указанному в последнем блоке, включённом в РР.	Данные о состоянии системы, указанные в поле **stateRoot** заголовка блока. Данное поле описано в п.4 – см. Таблица 12.	Зафиксированные данные о состоянии системы, соответствуют данным о состоянии системы, указанным в поле **stateRoot** заголовка блока. Данное поле описано в п.4 – см. Таблица 12.	Компонент исполнения прикладных задач.
2.	002	Взять список транзакций из пула	Берётся отсортированный массив из очереди pending, из пула транзакций на узле. Транзакции сортируются по максимальному вознаграждению (**gasPrice** в транзакции) и минимальный нонс для каждого идентификатора отправителя (поле nonce в транзакции). Поле **gasPrice** описано в п.2 – см. Таблица 11. Если в очереди pending, пула транзакций на узле нет транзакций, блок будет создаваться без транзакций.	Пул транзакций, формат данных описан в п.6.2.	Отсортированный список транзакций из пула, формат данных описан в п.6.2.	Компонент ведения РР.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
3.	003	Транзакции есть	В списке транзакций на включение в блок есть транзакции.	Отсортированный список транзакций из пула, формат данных описан в п.6.2.	Результаты проверки наличия транзакций в списке транзакций на включение в блок	Компонент ведения РР.
4.	004	Лимит по Газу не достигнут	Каждый блок имеет лимит потребления Газа за обработку транзакций, включаемых в данный блок. Майнер будет обрабатывать транзакции пока этот лимит не достигнут (**gasLimit** в заголовке блока). Данное поле описано в п.3 – см. Таблица 11.	Отсортированный список транзакций из пула с наличием в нем транзакций. Формат данных описан в п.6.2.	Результаты проверки того, что лимит по Газу не достигнут.	Компонент исполнения прикладных задач.
5.	005	Взять транзакцию из списка	Майнер берёт транзакцию из начала списка транзакций на включение в блок (стандартные настройки пула транзакций: максимальное вознаграждение и минимальный нонс).	Отсортированный список транзакций из пула с наличием в нем транзакций. Формат данных описан в п.6.2.	Первая транзакция из списка, формат данных описан в п.6.2.	Компонент ведения РР, компонент исполнения прикладных задач.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
6.	006	Уменьшить баланс технологического счета на указанное вознаграждение	Уменьшить баланс технологического счёта списания на сумму вознаграждения (**gasLimit** в транзакции). Данное поле описано в п.3 – см. Таблица 11.	Первая транзакция из списка, формат данных описан в п.6.2. Технологический счёт пользователя (поля **balance, nonce, storageRoot, codeHash**), где: – **balance** - количество TPE на данном технологическом счёте; – **nonce** или account nonce - количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора смарт-контракта. Данное поле необходимо для предотвращения атаки повторного воспроизведения (replay attack);	Технологический счёт пользователя с уменьшенным на указанное вознаграждение балансом (поля **balance, nonce, storageRoot, codeHash**), где: – **balance** - количество TPE на данном технологическом счёте;	Компонент исполнения прикладных задач, компонент адресации участников сетевого взаимодействия.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
				– **storageRoot** - хеш корневого элемента Merkle Patricia дерева, которое кодирует содержимое хранилища данного технологического счёта. Хеш берется от байтовой конкатинации ключа и значения корневого узла; – **codeHash** – хеш EVM кода данного технологического счёта (в случае смарт-контракта).	– **nonce** или account nonce - количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора смарт-контракта. Данное поле необходимо для предотвращения атаки повторного воспроизведения (replay attack);	

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
					– **storageRoot** - хеш корневого элемента Merkle Patricia дерева, которое кодирует содержимое хранилища данного технологического счёта. Хеш берется от байтовой конкатинации ключа и значения корневого узла; – **codeHash** – хеш EVM кода данного технологического счёта (в случае смарт-контракта).	
7.	007	Увеличить нонс технологического счета	Увеличить нонс технологического счёта списания.	Технологический счёт пользователя с уменьшенным на указанное вознаграждение балансом (поля **balance, nonce, storageRoot, codeHash**), где: – **balance** - количество TPE на данном технологическом счёте;	Технологический счёт пользователя с увеличенным нонсом (поля **balance, nonce, storageRoot, codeHash**), где: – **balance** - количество TPE на данном технологическом счёте;	Компонент исполнения прикладных задач, компонент адресации участников сетевого взаимодействия.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
				– **nonce** или account nonce - количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора смарт-контракта. Данное поле необходимо для предотвращения атаки повторного воспроизведения (replay attack); – **storageRoot** - хеш корневого элемента Merkle Patricia дерева, которое кодирует содержимое хранилища данного технологического счёта. Хеш берется от байтовой конкатинации ключа и значения корневого узла;	– **nonce** или account nonce - количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора смарт-контракта. Данное поле необходимо для предотвращения атаки повторного воспроизведения (replay attack);	

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
				– **codeHash** – хеш EVM кода данного технологического счёта (в случае смарт-контракта).	– **storageRoot** - хеш корневого элемента Merkle Patricia дерева, которое кодирует содержимое хранилища данного технологического счёта. Хеш берется от байтовой конкатинации ключа и значения корневого узла; – **codeHash** – хеш EVM кода данного технологического счёта (в случае смарт-контракта).	
8.	008	Уменьшить баланс технологического счета за размер транзакции	Уменьшить баланс технологического счёта списания на стоимость Газа за размер транзакции в байтах (Газ/байт).	Значение стоимости Газа за размер транзакции. Технологический счёт пользователя с увеличенным нонсом (поля **balance, nonce, storageRoot, codeHash**), где:	Технологический счёт пользователя с уменьшенным балансом за размер транзакции (поля **balance, nonce, storageRoot, codeHash**), где:	Компонент исполнения прикладных задач, компонент адресации участников сетевого взаимодействия.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
				– **balance** - количество TPE на данном технологическом счёте; – **nonce** или account nonce - количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора смарт-контракта. Данное поле необходимо для предотвращения атаки повторного воспроизведения (replay attack);	– **balance** - количество TPE на данном технологическом счёте; – **nonce** или account nonce - количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора смарт-контракта. Данное поле необходимо для предотвращения атаки повторного воспроизведения (replay attack);	

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
				– **storageRoot** - хеш корневого элемента Merkle Patricia дерева, которое кодирует содержимое хранилища данного технологического счёта. Хеш берется от байтовой конкатинации ключа и значения корневого узла; – **codeHash** – хеш EVM кода данного технологического счёта (в случае смарт-контракта).	– **storageRoot** - хеш корневого элемента Merkle Patricia дерева, которое кодирует содержимое хранилища данного технологического счёта. Хеш берется от байтовой конкатинации ключа и значения корневого узла; – **codeHash** – хеш EVM кода данного технологического счёта (в случае смарт-контракта).	
9.	009	Совершить перевод	Совершение перевода состоит из следующих действий: 1. Уменьшить баланс технологического счёта списания на значение, указанное в транзакции;	Технологические счета пользователей до перевода (поля **balance, nonce, storageRoot, codeHash**), где: – **balance** - количество ТРЕ на данном технологическом счёте;	Технологические счета пользователей после совершения перевода (поля **balance, nonce, storageRoot, codeHash**), где:	Компонент исполнения прикладных задач, компонент адресации участников сетевого взаимодействия.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
			2. Если идентификатор ключа пользователя получателя не существует, создать его; 3. Увеличить баланс технологического счёта начисления на значение, указанное в транзакции; 4. Если принимающий идентификатор ключа пользователя – это идентификатор ключа смарт-контракта, выполнить код смарт-контракта до полного его исполнения или пока не закончится Газ; 5. В случае, если смарт-контракт имеет внутреннее хранилище данных, то отправитель платит за хранение новой информации, помещённой туда.	– **nonce** или account nonce - количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора смарт-контракта. Данное поле необходимо для предотвращения атаки повторного воспроизведения (replay attack); – **storageRoot** - хеш корневого элемента Merkle Patricia дерева, которое кодирует содержимое хранилища данного технологического счёта. Хеш берется от байтовой конкатинации ключа и значения корневого узла;	– **balance** - количество TPE на данном технологическом счёте; – **nonce** или account nonce - количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора смарт-контракта. Данное поле необходимо для предотвращения атаки повторного воспроизведения (replay attack);	

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
				– **codeHash** – хеш EVM кода данного технологического счёта (в случае смарт-контракта).	– **storageRoot** - хеш корневого элемента Merkle Patricia дерева, которое кодирует содержимое хранилища данного технологического счёта. Хеш берется от байтовой конкатинации ключа и значения корневого узла; – **codeHash** – хеш EVM кода данного технологического счёта (в случае смарт-контракта).	
10.	010	TPE достаточно для перевода	TPE достаточно на счёте списания достаточно для перевода, если выполняются оба условия:	Транзакция, формат данных описан в п.6.2. Технологический счёт пользователя после совершения перевода (поля **balance, nonce, storageRoot, codeHash**), где:	Результаты проверки достаточности TPE для перевода.	Компонент исполнения прикладных задач.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
			– Значение баланса технологического счёта списания больше или равно цене обработки транзакции, указанной в транзакции; – Значение **gasLimit** транзакции больше или равно (Газ за обработку транзакции + Газ за выполнение кода смарт-контракта + Газ за хранение данных). Данное поле описано в п.3 – см. Таблица 11.	– **balance** - количество TPE на данном технологическом счёте; – **nonce** или account nonce - количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора смарт-контракта. Данное поле необходимо для предотвращения атаки повторного воспроизведения (replay attack);		

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
				– **storageRoot** - хеш корневого элемента Merkle Patricia дерева, которое кодирует содержимое хранилища данного технологического счёта. Хеш берется от байтовой конкатинации ключа и значения корневого узла; – **codeHash** – хеш EVM кода данного технологического счёта (в случае смарт-контракта).		
11.	011	Откатить перевод	Если на технологическом счёте списания недостаточно TPE для совершения перевода, изменения, совершённые на шаге 009 откатываются назад.	Отрицательный результат проверки достаточности TPE для перевода. Технологические счета пользователей после совершения перевода (поля **balance, nonce, storageRoot, codeHash**), где: – **balance** - количество TPE на данном технологическом счёте;	Технологические счета пользователей с отмененными изменениями в части перевода (поля **balance, nonce, storageRoot, codeHash**), где: – **balance** - количество TPE на данном технологическом счёте;	Компонент исполнения прикладных задач, компонент адресации участников сетевого взаимодействия.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
				– **nonce** или account nonce - количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора смарт-контракта. Данное поле необходимо для предотвращения атаки повторного воспроизведения (replay attack); – **storageRoot** - хеш корневого элемента Merkle Patricia дерева, которое кодирует содержимое хранилища данного технологического счёта. Хеш берется от байтовой конкатинации ключа и значения корневого узла;	– **nonce** или account nonce - количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора смарт-контракта. Данное поле необходимо для предотвращения атаки повторного воспроизведения (replay attack);	

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
				– **codeHash** – хеш EVM кода данного технологического счёта (в случае смарт-контракта).	– **storageRoot** - хеш корневого элемента Merkle Patricia дерева, которое кодирует содержимое хранилища данного технологического счёта. Хеш берется от байтовой конкатинации ключа и значения корневого узла; – **codeHash** – хеш EVM кода данного технологического счёта (в случае смарт-контракта).	
12.	012	Начислить вознаграждение за обработку транзакции	Увеличить баланс технологического счёта майнера на значение вознаграждения за обработку транзакции.	Результаты проверки достаточности TPE для перевода. Значение вознаграждения за обработку транзакции. Технологический счёт майнера до начисления вознаграждения (поля **balance, nonce, storageRoot, codeHash**), где:	Технологический счёт майнера после начисления вознаграждения (поля **balance, nonce, storageRoot, codeHash**), где:	Компонент ведения РР, компонент исполнения прикладных задач.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
				– **balance** - количество TPE на данном технологическом счёте; – **nonce** или account nonce - количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора смарт-контракта. Данное поле необходимо для предотвращения атаки повторного воспроизведения (replay attack);	– **balance** - количество TPE на данном технологическом счёте; – **nonce** или account nonce - количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора смарт-контракта. Данное поле необходимо для предотвращения атаки повторного воспроизведения (replay attack);	

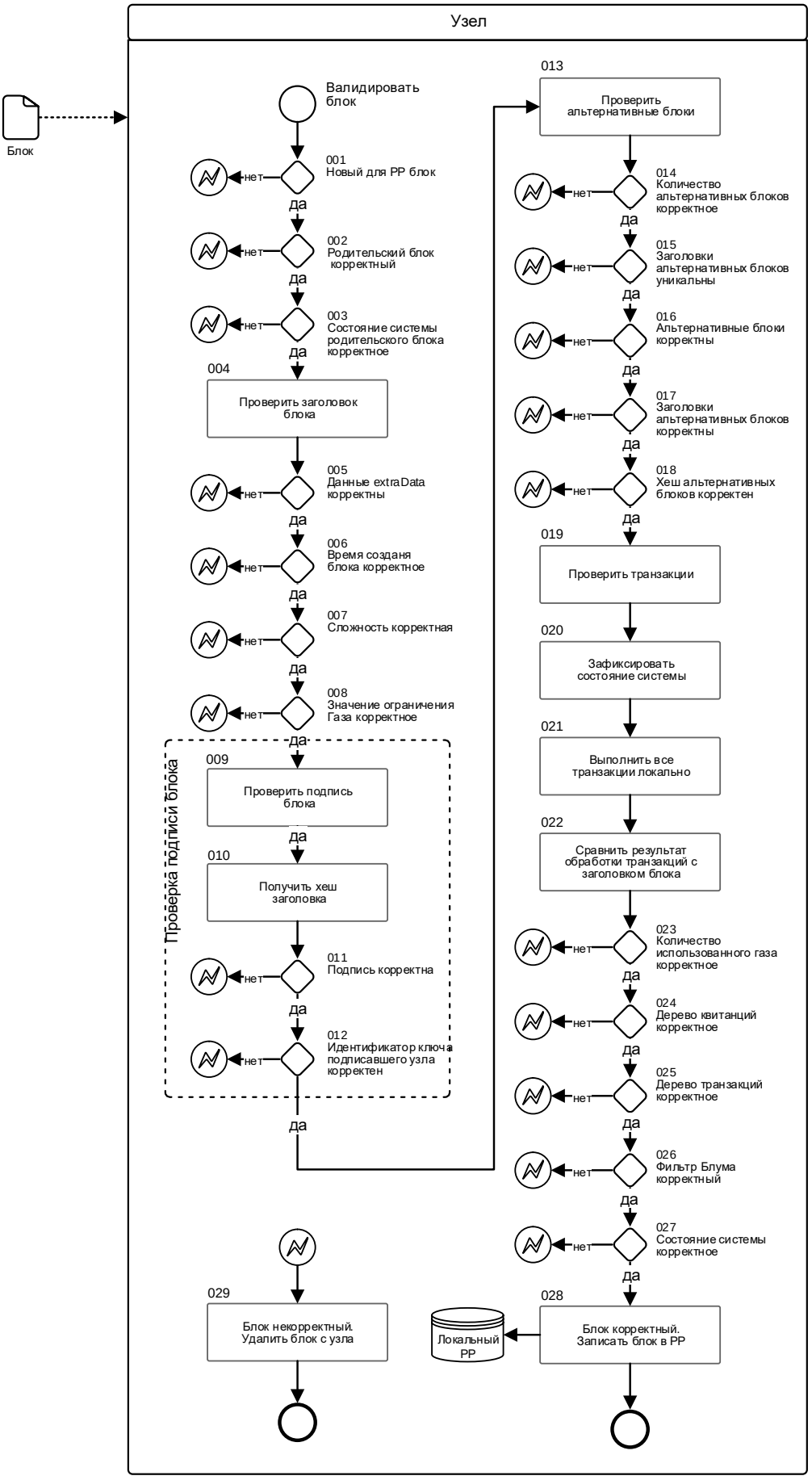
№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
				– **storageRoot** - хеш корневого элемента Merkle Patricia дерева, которое кодирует содержимое хранилища данного технологического счёта. Хеш берется от байтовой конкатинации ключа и значения корневого узла; – **codeHash** – хеш EVM кода данного технологического счёта (в случае смарт-контракта).	– **storageRoot** - хеш корневого элемента Merkle Patricia дерева, которое кодирует содержимое хранилища данного технологического счёта. Хеш берется от байтовой конкатинации ключа и значения корневого узла; – **codeHash** – хеш EVM кода данного технологического счёта (в случае смарт-контракта).	
13.	013	Вернуть стоимость неиспользованного Газа	Увеличить баланс технологического счёта списания на значение оплаты неиспользованного Газа.	Значение оплаты неиспользованного Газа. Технологический счёт майнера после начисления вознаграждения (поля **balance, nonce, storageRoot, codeHash**), где:	Технологический счёт пользователя (поля **balance, nonce, storageRoot, codeHash**), где: – **balance** - количество TPE на данном технологическом счёте;	Компонент исполнения прикладных задач, компонент адресации участников сетевого взаимодействия.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
				– **balance** - количество TPE на данном технологическом счёте; – **nonce** или account nonce - количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора смарт-контракта. Данное поле необходимо для предотвращения атаки повторного воспроизведения (replay attack);	– **nonce** или account nonce - количество транзакций, посланных с данного идентификатора ключа пользователя. В случае идентификатора смарт-контракта – количество смарт-контрактов, созданных с этого идентификатора смарт-контракта. Данное поле необходимо для предотвращения атаки повторного воспроизведения (replay attack);	

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
				– **storageRoot** - хеш корневого элемента Merkle Patricia дерева, которое кодирует содержимое хранилища данного технологического счёта. Хеш берется от байтовой конкатинации ключа и значения корневого узла; – **codeHash** – хеш EVM кода данного технологического счёта (в случае смарт-контракта).	– **storageRoot** - хеш корневого элемента Merkle Patricia дерева, которое кодирует содержимое хранилища данного технологического счёта. Хеш берется от байтовой конкатинации ключа и значения корневого узла; – **codeHash** – хеш EVM кода данного технологического счёта (в случае смарт-контракта).	
14.	014	Включить транзакцию в список обработанных транзакций	Включить транзакцию в список обработанных транзакций.	Транзакция до включения в список обработанных транзакций, формат данных описан в п.6.2.	Транзакция, включенная в список обработанных транзакций, список обработанных транзакций. Формат данных описан в п.6.2.	Компонент исполнения прикладных задач.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
15.	015	Увеличить значение использованного Газа для блока за обработанную транзакцию	Увеличить значение использованного газа для блока за обработанную транзакцию.	Значение использованного Газа. Блок до увеличения значения использованного Газа за обработанную транзакцию, формат данных описан в п. 7.2.1.	Блок после увеличения значения использованного Газа за обработанную транзакцию, формат данных описан в п. 7.2.1.	Компонент исполнения прикладных задач.
16.	016	Удалить транзакцию из списка транзакций для включения в блок	Транзакция удаляется из списка транзакций для включения в блок.	Транзакция, включенная в список обработанных транзакций, список обработанных транзакций. Формат данных описан в п.6.2.	Список транзакций без обработанной транзации, формат данных описан в п.6.2.	Компонент ведения РР.
17.	017	Список транзакций и данные по транзакциям для блока подготовлены	Результатом обработки транзакций для включения в блок является: – Состояние системы, – Дерево транзакций, – Дерево квитанций, – Количество использованного Газа, – Из дерева квитанций получаются записи для фильтра Блума.	Отрицательный результаты проверки хотя бы одного из шагов 003, 004.	Данные о состоянии системы, зафиксированные в поле **stateRoot** заголовка блока. Данное поле описано в п.4 – см. Таблица 12. Дерево транзакций, дерево квитанций. Формат данных описан в п.7.2.2. Количество использованного Газа.	Компонент исполнения прикладных задач.

ПРИЛОЖЕНИЕ И. ОПИСАНИЕ АЛГОРИТМА ВАЛИДАЦИЯ БЛОКА



Пояснения к алгоритму приведены – см. Таблица 22.

Таблица 22 – Описание алгоритма валидации блока

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
1.	001	Новый для РР блок	Осуществляется проверка является ли блок новым для РР (проверка по хешу блока).	Отправленный блок, записанный в локальный РР, формат данных описан в п.7.2.	Результаты проверки, является ли блок новым для РР.	Компонент ведения РР.
2.	002	Родительский блок корректный	Осуществляется проверка по указанному хешу родительского блока и номеру блока.	Родительский блок, блок, записанный в локальный РР. Формат данных описан в п.7.2.	Результаты проверки корректности родительского блока.	Компонент ведения РР.
3.	003	Состояние системы родительского блока корректное	Осуществляется проверка того, что из **StateRoot** родительского блока можно восстановить состояние системы.	Данные о состоянии системы, зафиксированные в поле **stateRoot** заголовка родительского блока. Данное поле описано в п.4 – см. Таблица 12.	Результаты проверки корректного состояния системы родительского блока.	Компонент ведения РР.
4.	004	Проверить заголовок блока	Осуществляется проверка полей заголовка блока.	Блок, записанный в локальный РР, формат данных полей заголовка блока описан в п.7.2.1.	Результаты проверки заголовка блока.	Компонент ведения РР.
5.	005	Данные extraData корректны	Данные, указанные в **extraData** корректны, если 32 байта <= extraData < 97 байта. Данное поле описано в п.13 – см. Таблица 12.	Блок, записанный в локальный РР, формат данных полей заголовка блока описан в п.7.2.1.	Результаты проверки корректности данных указанных в поле **extraData.** Данное поле описано в п.13 – см. Таблица 12.	Компонент ведения РР.
6.	006	Время создания блока корректное	Время создания блока корректное, если больше времени создания родительского блока.	Данные о времени создания блока, времени создания родительского блока.	Результаты проверки корректности времени создания блока.	Компонент ведения РР.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
7.	007	Сложность корректная	Сложность корректная, если – отличается от сложности родительского блока; – шаг изменения сложности (относительно сложности родительского блока) не меньше значения, вычисляемого по алгоритму изменения значения сложности.	Блок, записанный в РР, родительский блок. Формат данных описан в п. 7.2.1.	Результаты проверки корректности сложности блока.	
8.	008	Значение ограничения Газа корректное	Значение Газа корректное, если шаг изменения не меньше значения, вычисляемого по алгоритму изменения значения стоимости Газа.	Блок, формат данных описан в п. 7.2.1.	Результаты проверки корректности значения ограничения Газа.	Компонент ведения РР.
9.	009	Проверить подпись блока	Осуществляется проверка подписи блока (шаги 010-012).	Блок, формат данных описан в п. 7.2.1. Подпись блока.	Хеш заголовка блока. Результаты проверки корректности подписи. Результаты проверки корректности идентификатора ключа подписавшего узла.	Компонент ведения РР, компонент обеспечения сетевого взаимодействия между узлами РР.
10.	010	Получить хеш от заголовка	Вычисляется хеш от заголовка блока без последних 65 байт поля **ExtraData**. Данное поле описано в п.13 – см. Таблица 12.	Блок, записанный в локальный РР, формат данных полей заголовка блока описан в п.7.2.1.	Хеш заголовка блока.	Компонент ведения РР.
11.	011	Подпись корректна	Подпись корректна, если из подписи и хеша заголовка блока	Подпись блока, хеш заголовка блока.	Результаты проверки корректности подписи.	Компонент ведения РР.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
			можно восстановить публичный ключ подписавшего узла.			
12.	012	Идентификатор ключа подписавшего узла корректен	Идентификатор ключа подписавшего узла корректен, если: – Идентификатор ключа узла добавлен в Whitelist узлов; – Роль узла валидатор и майнер (выставлены флаги FLAG_CAN_READ и FLAG_CAN_MINE). Более подробно данные флаги описаны в документе ЖТЯИ.00107-01 91 01 «СКЗИ Мастерчейн. Руководство администратора информационной безопасности»; – Ключ не просрочен.	Идентификатор ключа подписавшего узла, Whitelist узлов, формат описан в п.3.3.7. Справочник узлов, формат данных описан в п.4.2.6 документа ЖТЯИ.00107-01 91 01 «СКЗИ Мастерчейн. Руководство администратора информационной безопасности».	Результаты проверки корректности идентификатора ключа подписавшего узла.	Компонент ведения РР, компонент обеспечения сетевого взаимодействия между узлами РР.
13.	013	Проверить альтернативные блоки	Осуществляется проверка включенных альтернативных блоков (шаги 014 – 018).	Включенные альтернативные блоки, формат данных описан в п.7.2.3.	Результаты проверки корректности количества альтернативных блоков. Результаты проверки уникальности заголовков альтернативных блоков.	Компонент ведения РР.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
					Результаты проверки корректности альтернативных блоков. Результаты проверки корректности заголовков альтернативных блоков. Результаты проверки корреткности хеша альтернативных блоков.	
14.	014	Количество альтернативных блоков корректное	Заголовков альтернативных блоков не может быть больше двух.	Включенные альтернативные блоки, формат данных описан в п.7.2.3.	Результаты проверки корректности количества альтернативных блоков.	Компонент ведения PP.
15.	015	Заголовки альтернативных блоков уникальны	Хеши заголовков альтернативных блоков не должны совпадать. Вычисление хеша заголовка альтернативного блока аналогично вычислению хеша заголовка блока.	Включенные альтернативные блоки, формат данных описан в п.7.2.3.	Результаты проверки уникальности заголовков альтернативных блоков.	Компонент ведения PP.
16.	016	Альтернативные блоки корректны	Альтернативные блоки корректны, если: – не добавлены в PP, как обычные блоки;	Включенные альтернативные блоки, формат данных описан в п.7.2.3. Родительские блоки включенных альтернативных блоков, блок, записанный в	Результаты проверки корректности альтернативных блоков.	Компонент ведения PP.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
			– родительские блоки альтернативных блоков находятся в РР среди последних шести блоков до валидируемого блока.	локальный РР, формат данных описан в п.7.2.		
17.	017	Заголовки альтернативных блоков корректны	Осуществляется проверка корректности заголовков альтернативных блоков: – Формат заголовков; – Решение POW заголовков; – Подпись заголовков.	Включенные альтернативные блоки, формат данных описан в п.7.2.3.	Результаты проверки корректности заголовков альтернативных блоков.	Компонент ведения РР.
18.	018	Хеш альтернативных блоков корректен	Вычисляется хеш от списка альтернативных блоков в проверяемом блоке. Этот хеш должен совпадать с хешом, указанном в заголовке блока.	Хеш заголовка блока. Хеш включенных альтернативных блоков.	Результаты проверки корреткности хеша альтернативных блоков.	Компонент ведения РР.
19.	019	Проверить транзакции	Осуществляется проверка списка транзакций блока (шаги 020 – 021).	Список транзакций блока, формат данных описан в п.6.2.	Зафиксированные данные о состоянии системы, соответствуют данным о состоянии системы, указанным в поле **stateRoot** заголовка блока. Данное поле описано в п.4 – см. Таблица 12. Дерево транзакций, дерево квитанций. Формат данных описан в п.7.2.2.	Компонент ведения РР, компонент исполнения прикладных задач.

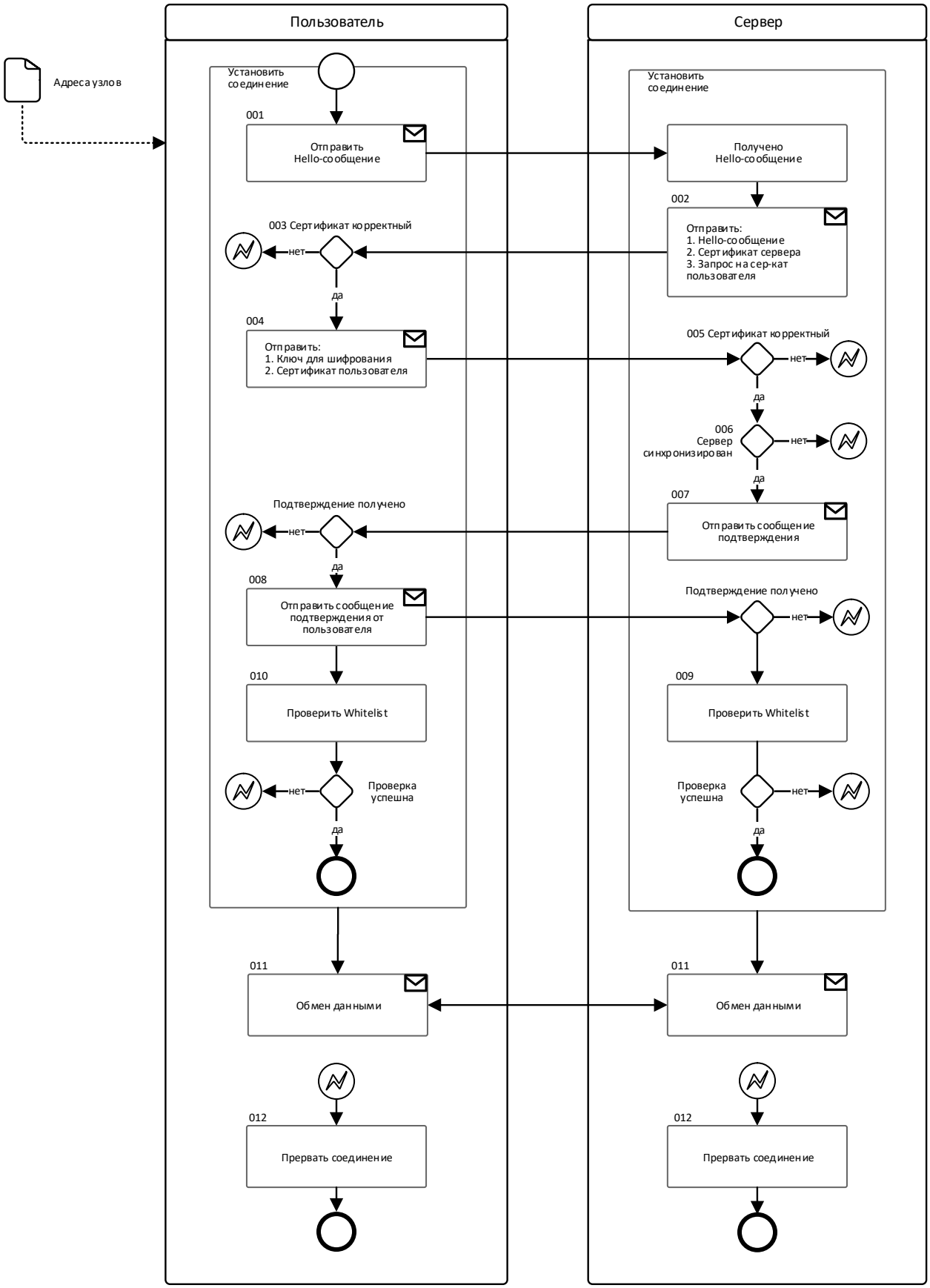
№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
					Количество использованного Газа.	
20.	020	Зафиксировать состояние системы	Фиксирование состояния системы: берётся состояние системы из родительского блока.	Данные о состоянии системы, указанные в поле **stateRoot** заголовка родительского блока. Данное поле описано в п.4 – см. Таблица 12.	Зафиксированные данные о состоянии системы, соответствуют данным о состоянии системы, указанным в поле **stateRoot** заголовка блока. Данное поле описано в п.4 – см. Таблица 12.	Компонент ведения РР.
21.	021	Выполнить все транзакции локально	Все транзакции из блока выполняются локально. Результатом выполнения транзакций является: – Состояние системы; – Дерево транзакций; – Дерево квитанций; – Количество использованного Газа; – Из дерева квитанций получаются записи для фильтра Блума.	Список транзакций блока, формат данных описан в п.6.2.	Зафиксированные данные о состоянии системы, соответствуют данным о состоянии системы, указанным в поле **stateRoot** заголовка блока. Данное поле описано в п.4 – см. Таблица 12. Дерево транзакций, дерево квитанций. Формат данных описан в п.7.2.2. Количество использованного Газа.	Компонент исполнения прикладных задач.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
22.	022	Сравнить результат обработки транзакций с заголовками блока	Сравниваются значения, полученные на шаге 021 со значениями, указанными в заголовке блока (шаги 023-027).	Зафиксированные данные о состоянии системы, соответствуют данным о состоянии системы, указанным в поле **stateRoot** заголовка блока. Данное поле описано в п.4 – см. Таблица 12. Дерево транзакций, дерево квитанций. Формат данных описан в п.7.2.2. Количество использованного Газа.	Результат сравнения обработки транзакций, формат данных описан в п.6.2, с заголовком блока, формат данных описан в п. 7.2.1.	Компонент ведения РР.
23.	023	Количество использованного газа корректное	Сравнивается значение, полученное на шаге 021 со значением, указанным в заголовке блока для использованного Газа.	Количество использованного Газа, полученное на шаге 021. Количество использованного Газа, указанное в заголовке блока, формат данных полей заголовка блока описан в п.7.2.1.	Результаты проверки корректности количества использованного Газа.	Компонент ведения РР.
24.	024	Дерево квитанций корректное	Сравнивается корневой элемент дерева квитанций, полученный на шаге 021 со значением, указанным в заголовке блока.	Дерево квитанций, полученное на шаге 021, формат данных описан в п.7.2.2. Значение поля **receiptsRoot** в заголовке блока, формат данных полей заголовка блока описан в п.7.2.1.	Результаты проверки корректности дерева квитанций.	Компонент ведения РР.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
25.	025	Дерево транзакций корректное	Сравнивается значение, полученное на шаге 021 со значением, указанным в заголовке блока для корневого элемента дерева транзакций.	Дерево транзакций, полученное на шаге 021, формат данных описан в п.7.2.2. Значение поля **transactionsRoot** в заголовке блока, формат данных полей заголовка блока описан в п.7.2.1.	Результаты проверки корректности дерева транзакций.	Компонент ведения РР.
26.	026	Фильтр Блума корректный	Сравнивается значение, полученное на шаге 021 со значением, указанным в заголовке блока для фильтра Блума.	Дерево квитанций, полученное на шаге 021, формат данных описан в п.7.2.2. Значение поля **logsBloom** (или bloom) в заголовке блока, формат данных полей заголовка блока описан в п.7.2.1.	Результаты проверки корректности фильтра Блума.	Компонент ведения РР.
27.	027	Состояние системы корректное	Сравнивается значение, полученное на шаге 021 со значением, указанным в заголовке блока для состояния системы.	Данные о состоянии системы, полученные на шаге 021, данные о состоянии системы, указанные в поле **stateRoot** заголовка блока. Данное поле описано в п.4 – см. Таблица 12.	Результаты проверки корректности состояния системы.	Компонент ведения РР.
28.	028	Блок корректный. Записать блок в РР	При успешном прохождении всех проверок блок считается валидным и записывается в локальный РР.	Блок, прошедший проверки, формат данных описан в п.7.2.	Локальный РР, с записанным блоком, формат данных описан в п.7.2.	Компонент ведения РР.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
29.	029	Блок некорректный. Удалить блок с узла	Если при валидации блок не проходит хоть одну проверку, процесс валидации прерывается, блок считается некорректным и удаляется с узла.	Блок, не прошедший хотя бы одну из проверок, формат данных описан в п.7.2.	Удаленный с узла блок, формат данных описан в п.7.2.	Компонент ведения РР.

ПРИЛОЖЕНИЕ К. ОПИСАНИЕ АЛГОРИТМА TLS-СОЕДИНЕНИЕ



Пояснения к алгоритму приведены – см. Таблица 23.

Таблица 23 – Описание алгоритма TLS-соединения

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
1.	001	Отправить Hello-сообщение (Получено Hello-сообщение)	Пользователь отправляет Hello-сообщение, в котором перечислены криптографические данные: 1.Версия TLS; 2.Список режимов симметричного шифрования, поддерживаемые пользователем, в порядке предпочтения пользователем; 3.Случайная последовательность байтов, которая используется при последующих вычислениях.	Адреса узлов. Данные для Hello-сообщения (отправленное Hello-сообщение), формат данных описан – см. **0**.	Отправленное (полученное) Hello-сообщение, формат данных описан – см. **0**.	Компонент обеспечения сетевого взаимодействия между узлами РР.
2.	002	Отправить: 1.Hello-сообщение 2.Сертификат сервера 3.Запрос на сертификат пользователя	Сервер отправляет: 1.Ответ Hello-сообщение, который содержит: режим симметричного шифрования, выбранный сервером из списка, предоставленного пользователем; идентификатор сессии; случайную последовательность байтов (отличную от шага 001.3); 2.Сертификат сервера; 3.Запрос на сертификат пользователя.	Полученное Hello-сообщение, формат данных описан – см. **0**.	Hello-сообщение, формат данных описан – см. **0**. Сертификат сервера, запрос на сертификат пользователя, формат данных описан в Приложении Р. Протокол TLS	Компонент обеспечения сетевого взаимодействия между узлами РР.

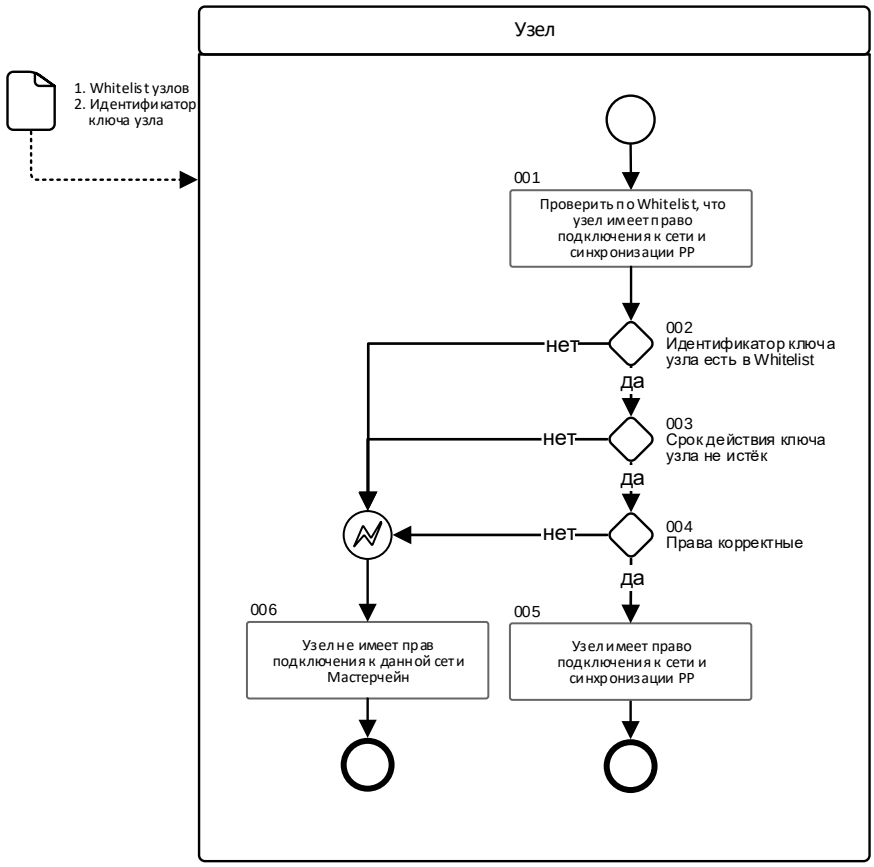
№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
3.	003	Сертификат корректный	Пользователь проверяет сертификат сервера.	Сертификат сервера, формат данных описан в Приложении Р. Протокол TLS	Результаты проверки корректности сертификата сервера.	Компонент обеспечения сетевого взаимодействия между узлами РР.
4.	004	Отправить: 1.Ключ для шифрования 2.Сертификат пользователя	Пользователь отправляет: 1.Случайную последовательность байтов, зашифрованную открытым ключом сервера, которая позволит пользователю и серверу вычислить секретный ключ, который будет использоваться для шифрования последующих сообщений; 2.Случайную последовательность байтов (шаг 002.1), зашифрованную закрытым ключом пользователя; 3.Сертификат пользователя.	Сертификат сервера, формат данных описан в Приложении Р. Протокол TLS.	Ключ для шифрования, сертификат пользователя, формат данных описан в Приложении Р. Протокол TLS.	Компонент обеспечения сетевого взаимодействия между узлами РР.
5.	005	Сертификат корректный	Сервер проверяет сертификат, присланный пользователем.	Сертификат пользователя, формат данных описан в Приложении Р. Протокол TLS.	Результаты проверки корректности сертификата пользователя.	Компонент обеспечения сетевого взаимодействия между узлами РР.
6.	006	Сервер синхронизирован	Сервер проверяет, что локальный РР синхронизирован.	Локальный РР, формат данных описан в п.7.2.	Результаты проверки того, что локальный РР синхронизирован.	Компонент обеспечения сетевого взаимодействия между узлами РР.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
7.	007	Отправить сообщение подтверждения	Сервер посылает сообщение подтверждения, которое зашифровано секретным ключом (шаг 004.1). Данное сообщение означает, что часть рукопожатия со стороны сервера завершена. TLS сессия установлена.	Ключ для шифрования, сертификат пользователя, формат данных описан в Приложении Р. Протокол TLS.	Сообщение подтверждения со стороны сервера, формат данных описан в Приложении Р. Протокол TLS.	Компонент обеспечения сетевого взаимодействия между узлами РР.
8.	008	Отправить сообщение подтверждения от пользователя	Пользователь посылает сообщение подтверждения, которое зашифровано секретным ключом (шаг 004.1). Данное сообщение означает, что часть рукопожатия со стороны пользователя завершена.	Сообщение подтверждения со стороны сервера, формат данных описан в Приложении Р. Протокол TLS.	Сообщение подтверждения со стороны пользователя, формат данных описан в Приложении Р. Протокол TLS.	Компонент обеспечения сетевого взаимодействия между узлами РР.
9.	009	Проверить Whitelist	Проверка идентификатора ключа пользователя сервера в Whitelist узлов. С сервером будет установлено соединение, если выполнены все условия: 1.Идентификатор ключа пользователя есть в Whitelist узлов;	Идентификатор ключа пользователя сервера, Whitelist узлов, формат описан в п.3.3.7. Справочник узлов, формат данных описан в документе ЖТЯИ.00107-01 91 01 «СКЗИ Мастерчейн. Руководство администратора информационной безопасности».	Результаты провер**к**и идентификатора ключа пользователя сервера в Whitelist узлов.	Компонент обеспечения сетевого взаимодействия между узлами РР.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
			2.Для идентификатора ключа пользователя выставлен флаг FLAG_CAN_READ (синхронизация цепочки разрешена). Более подробно данный флаг описан в документе ЖТЯИ.00107-01 91 01 «СКЗИ Мастерчейн. Руководство администратора информационной безопасности»; 3.Срок жизни ключа не истёк. Подробно алгоритм проверки Whitelist при TLS описан – см.**0**.			
10.	010	Проверить Whitelist	Проверка идентификатора ключа пользователя серверав Whitelist узлов. С пользователем будет установлено соединение, если выполнены все условия: 1.Идентификатор ключа пользователя есть в Whitelist узлов;	Идентификатор ключа пользователя сервера, Whitelist узлов, формат описан в п.3.3.7. Справочник узлов, формат данных описан в документе ЖТЯИ.00107-01 91 01 «СКЗИ Мастерчейн. Руководство администратора информационной безопасности».	Результаты провер**к**и идентификатора ключа пользователя сервера в Whitelist узлов.	Компонент обеспечения сетевого взаимодействия между узлами РР.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
			2.Для идентификатора ключа пользователя выставлен флаг FLAG_CAN_READ (синхронизация цепочки разрешена). Более подробно данный флаг описан в документе ЖТЯИ.00107-01 91 01 «СКЗИ Мастерчейн. Руководство администратора информационной безопасности»; 3.Срок жизни ключа не истёк. Подробно алгоритм проверки Whitelist при TLS описан – см.**0**.			
11.	011	Обмен данными	Сессия установлена. Далее, в рамках этой сессии сервер и пользователь обмениваются данными, которые симметрично шифруются с помощью общего секретного ключа (шаг 004.1).	Данные для передачи, формат описан в Приложении Р. Протокол TLS.	Полученные данные, формат описан в Приложении Р. Протокол TLS.	Компонент обеспечения сетевого взаимодействия между узлами РР.
12.	012	Прервать соединение	Разрыв соединения.	Отрицательные результаты проверок хотя бы одного из шагов 003, 005-010.	Сообщение об ошибке.	Компонент обеспечения сетевого взаимодействия между узлами РР.

ПРИЛОЖЕНИЕ Л. ОПИСАНИЕ АЛГОРИТМА WHITELIST ПРОВЕРКА ПРИ TLS



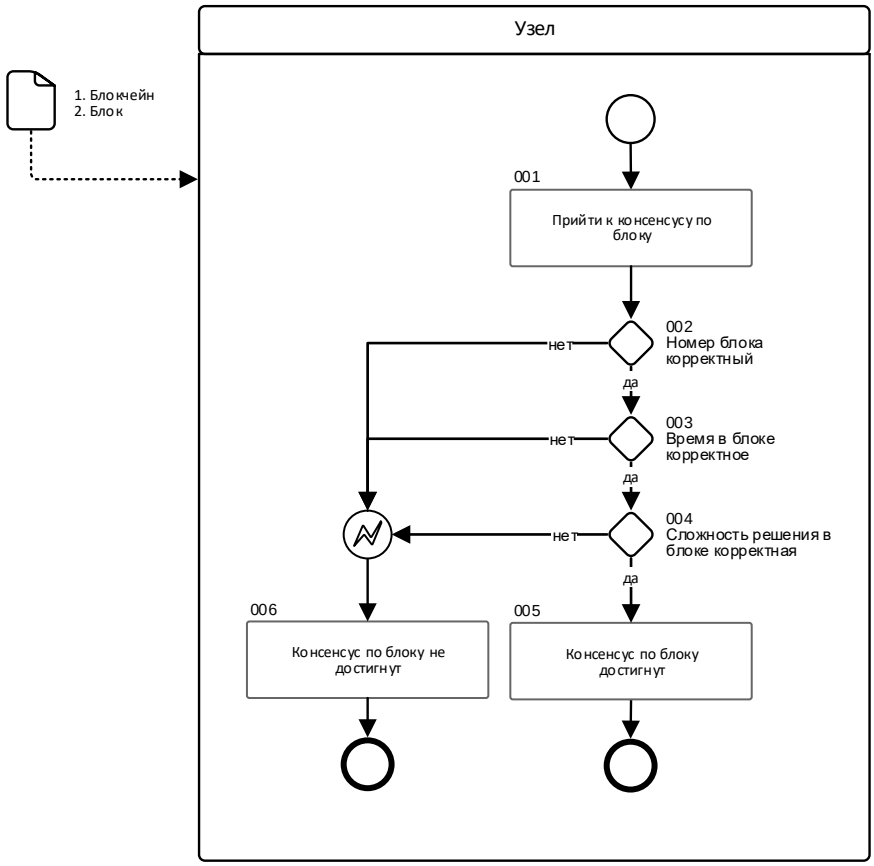
Пояснения к алгоритму приведены – см. Таблица 24.

Таблица 24 – Описание алгоритма Whitelist проверки при TLS

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
1.	001	Проверить по Whitelist, что узел имеет право подключения к сети и синхронизации РР	Проверить по Whitelist, что узел с данным идентификатором имеет право подключения к сети и синхронизации РР.	Полученный от узла идентификатор ключа узла, Whitelist узлов, формат описан в п.3.3.7.	Результаты проверки по Whitelist, что узел имеет право подключения к сети и синхронизации РР.	Компонент обеспечения сетевого взаимодействия между узлами РР.
2.	002	Идентификатор ключа узла есть в Whitelist	Проверка методами Whitelist смарт-контракта, что идентификатор ключа узла есть в вписке узлов. Функция *isNode.*	Идентификатор ключа узла, Whitelist узлов, формат описан в п.3.3.7.	Результат проверки того, что идентификатор ключа узла есть в Whitelist.	Компонент обеспечения сетевого взаимодействия между узлами РР.
3.	003	Срок действия ключа узла не истёк	Проверка методами Whitelist смарт-контракта, что срок действия данного ключа не истёк. Функция *isExpired.*	Идентификатор ключа узла, Whitelist узлов, формат описан в п.3.3.7.	Результат проверки того, что срок действия ключа узла не истёк.	Компонент обеспечения сетевого взаимодействия между узлами РР.

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
4.	004	Права корректные	Права корректные, если роль узла валидатор (выставлен флаг FLAG_CAN_READ). Более подробно данный флаг описан в документе ЖТЯИ.00107-01 91 01 «СКЗИ Мастерчейн. Руководство администратора информационной безопасности». Проверка методами Whitelist смарт-контракта, что данный флаг выставлен. Функции *isFlagSet* и *getMask*.	Справочник узлов, формат данных описан в п.3.3.7.	Результат проверки корректности прав.	Компонент обеспечения сетевого взаимодействия между узлами РР.
5.	005	Узел имеет право подключения к сети и синхронизации РР	Если все проверки успешны, узел с данным идентификатором ключа имеет право подключаться к данной сети Мастерчейн и синхронизировать РР.	Положительные результаты проверок шагов 001-004.	Результаты проверок идентификатора ключа пользователя сервера в Whitelist узлов.	Компонент обеспечения сетевого взаимодействия между узлами РР.
6.	006	Узел не имеет прав подключения к данной сети Мастерчейн	Если хоть одна проверка не успешна, узел с данным идентификатором ключа не имеет прав подключения к данной сети Мастерчейн.	Отрицательные результаты проверки хотя бы одного из шагов 002-004.	Сообщение об ошибке.	Компонент обеспечения сетевого взаимодействия между узлами РР.

ПРИЛОЖЕНИЕ М. ОПИСАНИЕ АЛГОРИТМА КОНСЕНСУС



Пояснения к алгоритму приведены – см. Таблица 25.

Таблица 25 – Описание алгоритма консенсуса по блоку

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
1.	001	Прийти к консенсусу по блоку	Прийти к консенсусу по блоку.	Блок, формат данных описан в п.7.2.	Блок, включенный в РР, формат данных описан в п.7.2.	Компонент ведения РР
2.	002	Номер блока корректный	Номер блока корректный, если равен номеру родительского блока + 1 (проверяются значения поля **number** в заголовке родительского блока и проверяемого блока). Данное поле описано в п.9 – см. Таблица 12.	Блок, записанный в РР, родительский блок. Формат данных описан в п.7.2.1.	Результаты проверки корректности номера блока.	Компонент ведения РР
3.	003	Время в блоке корректное	Время создания блока корректное, если больше времени создания родительского блока (проверяются значения поля **timestamp** в заголовке родительского блока и проверяемого блока). Данное поле описано в п.12 – см. Таблица 12.	Блок, записанный в РР, родительский блок. Формат данных описан в п. 7.2.1.	Результаты проверки корректности времени в блоке.	Компонент ведения РР
4.	004	Сложность решения в блоке корректная	Сложность решения (поле **difficulty** в заголовке блока) вычисляется из уровня сложности предыдущего (родительского) блока и **timestamp.** Данные поля описаны в п.п. 8, 12 – см. Таблица 12.	Блок, записанный в РР, родительский блок. Формат данных описан в п. 7.2.1.	Результаты проверки корректности сложности решения.	Компонент ведения РР

№ п/п	№ шага	Действие	Описание	Входные данные	Выходные данные	Компонент, принимающий участие
5.	005	Консенсус по блоку достигнут	Если все проверки успешны, по данному блоку достигнут консенсус.	Положительные результаты проверки шагов 002-004.	Блок, включенный в РР, формат данных описан в п.7.2.	Компонент ведения РР
6.	006	Консенсус по блоку не достигнут	Если хоть одна проверка не успешна, консенсус не может быть достигнут по данному блоку.	Отрицательные результаты проверки хотя бы одного из шагов 002-004.	Блок, не включаемый в РР, формат данных описан в п.7.2.	Компонент ведения РР

ПРИЛОЖЕНИЕ Н. ВЗАИМОСВЯЗЬ КОМПОНЕНТ СКЗИ МАСТЕРЧЕЙН С ПАКЕТАМИ И ФАЙЛАМИ

№ п/п	Компонент	Пакет	Файл	Описание файла
1.	**Компонент ведения РР**	**bmt**	bmt.go	Реализация дерева хешей Меркла (BMT). Предлставлено 2 реализации: - Hasher - оптимизированная реализация с минимальным набором контролирующих структур для координации параллельных рутин. Предоставляет интерфейс ChunkHash такой же как, стандартный hash. Hash интерфейс Golang; - ChunkHash – эталонная реализация.
			bmt_r.go	Дополнение к реализации hasher (RefHasher).
		consensus	consensus.go	PoW - интерфейс консенсуса, основанного на proof-of-work. Предоставляет функцию Hashrate. Engine - интерфейс, предоставляющий следующие функции алгоритма консенсуса: - Author() - возвращает адрес аккаунта, создавший данный блок. Может отличаться от coinbase в заголовке блока, если консенсус основан на подписях; - VerifyHeader() - проверяет, что заголовок блока соответствует правилам консенсуса. Так же может проверяться криптографическая печать блока; - VerifyHeaders() - то же самое, что и VerifyHeader, но проверяет несколько блоков конкурентно. Метод возвращает канал для остановки операций и канал для результатов асинхронной верификации;

№ п/п	Компонент	Пакет	Файл	Описание файла
				- VerifyUncles() - проверяет, что uncles блока соответствуют правилам используемого алгоритма консенсуса; - VerifySeal() - проверяет, что криптографическая печать блока верна согласно правилам используемого алгоритма консенсуса; - Prepare() - инициализирует поля консенсуса в заголовке блока согласно правилам используемого алгоритма консенсуса; - Finalize() - запускает любые пост-транзакционные модификации состояния (например, вознаграждение за блок) и собирает финальный блок - Seal() - генерирует новый блок для подданных входящих данных; - APIs() - возвращает набор доступных API, которые предоставляет используемый алгоритм консенсуса. ChainReader - интерфейс, предоставляющий методы для доступа к локальному PP во время header & uncle верификации: - Config() - получает конфигурацию цепи PP; - CurrentHeader() - получает текущий заголовок PP; - GetHeader() - получает заголовок блока по заданному хэшу и номеру; - GetHeaderByNumber() - получает заголовок блока по заданному номеру; - GetHeaderByHash() - получает заголовок блока по заданному хэшу; - GetBlock() - получает блок по заданному хэшу и номеру.
			errors.go	Набор ошибок, возникающих при работе с функциями пакета consensus.

№ п/п	Компонент	Пакет	Файл	Описание файла
			clique	
			clique/api.go	Обработка API вызовов в RPC при работе в режиме POA.
			clique/clique.go	Алгоритм консенсуса POA.
			clique/snapshot.go	Код обработки снепшотов (состояние консенсусу на конкретный момент времени).
			ethash	
			ethash/algorithm.go	Код для инициализации данных, необходимых для работы ethhash.
			ethash/algorithm_go1.7.go	Обработчики, специфичные для go1.7.0.
			ethash/algorithm_go1.8.go	Обработчики, специфичные для go1.8.0.
			ethash/consensus.go	Код для прихода к консенсусу по данным с использованием POW.
			ethash/ethash.go	Ethash - алгоритм POW консенсуса. Основан на алгоритме Dagger-Hashimoto. Основная работа алгоритма состоит в следующем: 1. Существует стартовое значение (seed), которое может быть вычислено для каждого блока, обрабатывая заголовки блоков, вплоть до текущего; 2. Из стартового значения вычисляется 16 МБ (мегабайт) данных для псевдослучайного кэша;

№ п/п	Компонент	Пакет	Файл	Описание файла
				3. Из полученного кэша генерируется 1 ГБ (гигабайт) данных датасета, со следующим свойством: каждой сущности датасета сопоставляется зависимость от небольшого количества сущностей из кэша. Полные клиенты и майнеры хранят датасеты на каждом узле. Датасеты растут во времени линейно; 4. Операции майнинга включают выборку случайных массивов датасета и последующего их совместного хэширования. Проверка может быть осуществлена используя небольшое количество памяти, используя кэш для воссоздания конкретных кусков датасета, поэтому проверяющему необходимо только хранить кэш. Большой датасет обновляется каждые 30000 (тридцать тысяч) блоков, поэтому подавляющее число операций майнера будет состоять в чтении датасета, а не во внесении изменений в него. - config - конкретная конфигурация алгоритма; - caches - карта кэшей, содержащихся в памяти, для текущей эпохи. Необходима для избежания частого обновления ethash; - datasets - наборы данных, содержащиеся в памяти, для текущей эпохи. Необходимы для избежания частого обновления ethash; - fdataset - наборы данных для следующей эпохи; - rand - генератор псевдослучайных чисел с заданными сидами для генерации соли (nonce); - threads - количество потоков, доступных для майнинга; - update - канал для обновления параметров майнинга;

№ п/п	Компонент	Пакет	Файл	Описание файла
				- hashrate - счётчик текущего хэшрейта. Config - конфигурация алгоритма Ethash: - CacheDir - адрес директории кэша в файловой системе; - CachesInMem - количество частей кэша в памяти; - CachesOnDisk - количество частей кэша на диске; - DatasetDir - адрес директории.
			ethash/sealer.go	Код, решающий POW для заданных данных.
			misc	
			misc/forks.go	VerifyForkHashes - функция, проверяющая что блоки, подходящие под условия хард-форков сети имеют правильные хеши для исключения перехода клиентов на другие части цепи.
		core	block_validator.go	Отвечает за валидацию заголовков блоков, отвергнутых блоков (uncles/ommers) и обработанных состояний.
			blockchain.go	Содержит структуру BlockChain (каноничный РР, записанный в базу данных с генезис-блоком) и функции для работы с ней: импорт (import), возврат к предыдущему состоянию (revert), реорганизация РР. Импорт включает в себя двухэтапную валидацию. На первом этапе обрабатываются включенные в блок транзакции, на втором валидируется полученное состояние. Также, BlockChain предоставляет функцию GetBlock, которая возвращает блоки из ЛЮБОГО чейна, включенного в базу данных, тогда как функция GetBlockByNumber всегда возвращает блоки из каноничного чейна.

№ п/п	Компонент	Пакет	Файл	Описание файла
			blocks.go	Содержит набор специально занесённых хешей блоков, которые вручную были помечены как bad hashes в сети Ethereum (обычно это хард-форк блоки).
			chain_indexer.go	ChainIndexer выполняет работу по пост-процессингу секций (каноничного PP) одинаковой длины. ChainIndexer имеет связь с PP через систему событий ChainEventLoop, запускаемую в отдельной goroutine. ChainIndexerBackend интерфейс предоставляет функции для обработки сегментов PP в бэкграунде и записывает результаты в базу данных.
			chain_makers.go	GenerateChain создаёт цепочку из n блоков. Родителем первого блока будет назначен предоставленный parent блок. База данных может использоваться для хранения независимых состояний и должна содержать state trie родителя. Блоки, созданные с помощью GenerateChain не содержат значений valid proof of work. Чтобы вставить их в BlockChain, необходимо использовать механизм FakePow или другую реализацию невалидирующего Proof of Work.
			database_util.go	Высокоуровневые функции для работы с базой данных. Возвращают данные, упакованные в конкретные объекты. - encodeBlockNumber кодирование номера блока в big endian uint64; - GetCanonicalHash() получение хеша номера блока; - GetHeadHeaderHash() получение хеша канонического заголовка текущего блока; GetHeadBlockHash() получение хеша заголовка блока, полученного в режиме синхронизации full; - GetHeadFastBlockHash() получение хеша заголовка последнего блока, полученного в режиме синхронизации fast.

№ п/п	Компонент	Пакет	Файл	Описание файла
				- GetHeaderRLP() получение заголовка блока в RLP-кодированом виде или nil, если заголовок блока не найден; - GetHeader() получение заголовка блока, соответствующего переданому хешу или nil, если блок не найден по переданному хешу; - GetBodyRLP() получение тела блока в RLP-кодированом виде; - GetBody() получение тела блока (транзакции, альтернативные блоки), соответствующего переданому хешу или nil, если блок не найден; - GetTd() получение значения полной сложности блока (total difficulty), соответствующее переданому хешу или nil, если блок не найден; - GetBlock() получение блока целиком, соответствующего переданому хешу, собирая его из сохранённого заголовка и тела блока. Возвращает nil если заголовок или тело блока не найдено. Замечание: так как загрузка заголовка и блока производится параллельно, заголовок и его хеш могут быть сохранены в базе данных, но тело ещё может не быть пока доступно. - GetBlockReceipts() получение по хешу блока квитанций (receipts), созданных транзакциями, включенными в блок; - GetTxLookupEntry() получение позиционных метаданных, связанных с транзакцией для извлечения транзакции или получения по хешу; - GetTransaction() получение определенной транзакции из базы данных вместе с дополнительными позиционными метаданными; - GetReceipt() получение квитанции для транзакции из базы данных вместе с дополнительными позиционными метаданными;

№ п/п	Компонент	Пакет	Файл	Описание файла
				- GetBloomBits() получение фильтр Блума по предоставленому хешу, индексу и секции; - WriteCanonicalHash() запись хеша заданного блока; - WriteHeadHeaderHash() запись хеша заголовка последнего полного блока; - WriteHeadBlockHash() запись хеша последнего полного блока; - WriteHeadFastBlockHash() запись хеша последнего быстрого блока; - WriteHeader() сериализация и запись заголовка блока в базу данных; - WriteBody() сериализация и запись тела блока в базу данных (вызывает WriteBodyRLP); - WriteBodyRLP() запись сериализованного тела блока в базу данных; - WriteTd() сериализация и запись значения общей сложности (total difficulty); - WriteBlock() сериализация и запись блока в базу данных, заголовок отдельно от тела блока; - WriteBlockReceipts() запись всех квитанций транзакций, принадлежащих блоку как единый массив квитанций. Используется при реорганизации цепи для планирования отброшенных транзакций; - WriteTxLookupEntries() запись метаданных каждой транзакции блока, что обеспечивает возможность поиска транзакции и квитанции по хешу; - WriteBloomBits() запись сжатого фильтра Блума, принадлежащего предоставленной секции и индексу;

№ п/п	Компонент	Пакет	Файл	Описание файла
				- DeleteCanonicalHash() удаление из базы данных ключ-значения номер блока, хеш; - DeleteHeader() удаление всех данных, ассоциированных с хешем заголовка блока; - DeleteBody() удаление всех данных, ассоциированных с хешем тела блока; - DeleteTd() удаление всех данных общей сложности (total difficulty) блока по хешу блока; - DeleteBlock() удаление всех данные блока по хешу блока; - DeleteBlockReceipts() удаление данных о всех квитанциях блока по хешу блока; - DeleteTxLookupEntry() удаление квитанций всех транзакций блока по хешу блока; - PreimageTable() получение экземпляра базы данных с префиксом ключа для записей прообраза; - WritePreimages() запись предоставленного набора прообразов в базу данных. «number» - это номер текущего блока и используется только для сообщений отладки; - GetBlockChainVersion() получение версии PP из базы данных; - WriteBlockChainVersion() запись версии PP в базу данных; - WriteChainConfig() запись настроек PP в базу; - GetChainConfig() получение настроек PP по предоставленному хешу;

№ п/п	Компонент	Пакет	Файл	Описание файла
				- FindCommonAncestor() получение последнего общего предка двух заголовков блока.
			error.go	Набор ошибок, возвращаемых при работе с PP.
			events.go	Различные структуры, реализующие события: - TxPreEvent - события от транзакций, попавших в пул транзакций; - PendingLogsEvent – события от транзакций попаших в pending block, но не включённых в цепочку; - PendingStateEvent – событие об итоговом состоянии pending-блока; - NewMinedBlockEvent - публикуется, когда блок импортирован в PP; - RemovedTransactionEvent - события об откаченных транзакциях (в случае пересогласования цепочки); - RemovedLogsEvent - события об откаченных (в случае пересогласования цепочки) событиях (объявленных в смарт-контрактах).
			evm.go	ChainContext - интерфейс, предоставляющий возможность получить заголовки и параметры консенсуса из существующего PP, чтобы использовать эти значения для обработки транзакции.
			fees.go	Переменная BlockReward.
			gaspool.go	Реализация типа данных GasPool, необходимого при выполнении транзакций блока. Функции для работы с GasPool: AddGas, SubGas.
			gen_genesis.go	Создание genesis-блока на основе JSON-описания.

№ п/п	Компонент	Пакет	Файл	Описание файла
			gen_genesis_account.go	Создание состояний смарт-контрактов и индентификаторов публичных ключей genesis-блока на основе JSON-описания.
			state_processor.go	Выполняет переход из одного состояния PP в другое.
			state_transition.go	stage transition - изменения, сделанные при применении транзакции к текущему состоянию PP. State Transitioning Model выполняет всю необходимую работу для перехода к новому валидному состоянию PP: 1. Обработка нонса; 2. Обработка предоплаты Газа; 3. Создание нового объекта состояния, если адрес получателя \0*32; 4. Передача ценности: если создается смарт-контракт a) попытка выполнить данные транзакции; b) если действительно, использовать результат, как код для нового объекта состояния; конец 5. Запуск секции Script; 6. Получение нового кореня состояния.

№ п/п	Компонент	Пакет	Файл	Описание файла
			tx_journal.go	Лог транзакций, позволяющий сохранить созданные локально и невыполненные транзакции, чтобы избежать их уничтожения при перезапуске узла.
			tx_list.go	Код для работы со структурой данных, в которой сохраняются транзакции.
			tx_pool.go	Реализация пула транзакций.
			types.go	Интерфейсы Validator и Processor.
		miner	agent.go	Описание интерфейсов для расчёта POW локально на CPU.
			miner.go	Код для формирования и контроля выполнения задачи для POW.
			remote_agent.go	Код, обеспечивающий управление и мониторинг майнером через HTTP и IPC API.
			unconfirmed.go	Код для работы с неподтверждёнными блоками (которые были найдены локально, но ещё не были приняты\отвергнуты сетью).
			worker.go	Код блок (и заголовок) для которого будет решаться POW. Формирует подпись блока, для которого найден POW.
		rlp	decode.go	Логика декодера.
			doc.go	Комментарий.
			encode.go	Логика кодирования.

№ п/п	Компонент	Пакет	Файл	Описание файла
			raw.go	Обработчик закодированных данных.
			typecache.go	Логика управления операциями кодирования.
		trie	encoding.go	Код кодирования данных.
			errors.go	Описание ошибок.
			hasher.go	Код для хеширования в дереве.
			iterator.go	Код итератора для перемещения по дереву.
			node.go	Код для работы с узлами деревьев
			proof.go	Код для исполнения доказательств Меркла, для проверки наличия хеша данных в дереве.
			secure_trie.go	Код для ускоренной проверки.
			sync.go	Актуализация дерева по поступающим данным.
			trie.go	Описание функций и интерфейсов для работы с деревом.
2.	**Компонент обеспечения сетевого взаимодействия между узлами PP**	**compression**	compression/rle/read_write.go	Код для сжатия данных.
		ethdb	database.go	LDBDatabase - структура, хранящая подключение к базе данных levelDB, различные метрики и средство поиска. Так же файл содержит низкоуровневые функции: Put, Get, Has, Delete.

№ п/п	Компонент	Пакет	Файл	Описание файла
			interface.go	Database - интерфейс, инкапсулирующий все операции с базой данных. Все функции могут использоваться конкурентно. Batch - интерфейс записи к основной базе данных, для вызова функции Write. Не может использоваться конкурентно.
			memory_database.go	Реализация in-memory базы данных. Запрещено использование в продакшене для хранения данных.
		node	api.go	Код, обеспечивающий управление и мониторинг узла через HTTP и IPC API.
			config.go	Функции для преобразования настроек узла в объект и для взаимодействия с ним.
			defaults.go	Умолчательные настройки узла.
			doc.go	В данном файле только текстовый комментарий.
			errors.go	Описание ошибок.
			node.go	Основной код узла, в котором происходит запуск API-интерфейсов, запуск сетевой подсистемы для общения с другими узлами, чтение настроек, подключение базы и крипто-контейнера.
			service.go	Код для описания сервисов. Сервисы - абстракция, которая используется для запуска параллельных тредов процесса, таких как, например, тред, обслуживающий HTTP API и тред для сетевого взаимодействия с другими узлами.

№ п/п	Компонент	Пакет	Файл	Описание файла
		P2P	dial.go	Структура dialstate используется для управления tcp вызовами к другим нодам и discovery-опросами других нод по udp. Структура dialTask используется для каждой ноды, которую вызывает узел.
			message.go	Структура P2P сообщения? определена в Msg.
			metrics.go	Содержит метрики и таймеры, используемые на сетевом уровне.
			peer.go	Peer нужна для представления данных удалённого узла, подключенного к локальному узлу.
			peer_error.go	Набор ошибок при работе с сетью.
			protocol.go	Описание для абстракции протокола взаимодействия между узлами. В данном случае это rlpx, завёрнутый в TLS.
			rlpx.go	RLPx - транспортный протокол, используемый сетевыми соединениями. Этот протокол работает «внутри» TLS.
			server.go	Server управляет сетевыми соединениями.
			discover	
			database.go	Структура nodeDB содержит интерфейс к levelDB (на файловой системе) и хранит данные о всех известных узлу других узлах.

№ п/п	Компонент	Пакет	Файл	Описание файла
			node.go	Структура Node содержит представление узла в сети (IP, TCP- и UDP-порты, индентификатор публичного ключа узла) и функции, обеспечивающие интерфейс к этой структуре.
			ntp.go	Код для контроля временных меток в протоколе.
			table.go	Таблица (находится в RAM) узлов с которыми взаимодействует узел и интерфейс для работы с ней.
			udp.go	Описание UDP-протокола.
			nat	
			nat.go	Код автоматической подстройки при работе «за NAT».
			natpmp.go	Для подстройки в случае nat-pmp протокола.
			natupnp.go	Для случая UPnP протокола.
			netutil	
			error.go	Интерфейс для сообщений об ошибках парсеров.
			net.go	Парсеры IP-адресов и прочая логика, используемая для диагностики сети.
			toobig_notwindows.go	Логика обработки пакетов неожиданного размера.
			toobig_windows.go	Логика обработки пакетов неожиданного размера.

№ п/п	Компонент	Пакет	Файл	Описание файла
		whitelist-utils	-	Набор скриптов для создания приватной сети Мастерчейн.
3.	**Компонент исполнения прикладных задач**	**accounts**	**abi**	
			abi/abi.go	ABI содержит информацию о контексте смарт-контракта и выполняемых методах (функциях). Это позволяет проводить проверку типов выполняемых функций и подготавливать данные соответствующе. - Constructor - конструктор контракта; - Methods - карта методов контракта; - Events - карта событий контракта.
			abi/argument.go	Argument содержит имя аргумента и его тип. Типы используются при упаковке и тестировании аргументов.
			abi/doc.go	Пакет abi содержит реализацию Ethereum ABI (Application Binary Interface). Ethereum ABI статическое, строго типизированое, компилируемое. ABI содержит базовые правила приведения типов (знаковые в беззнаковые и обратно). Не поддерживается преобразование массивов беззнаковых значений в массив знаковых значений.
			abi/error.go	Описание для обработчика ошибок, возникающих при работе abi.
			abi/event.go	Event - событие, потенциально вызванное LOG механизмом виртуальной машины EVM. Содержит информацию (input) о данном выводе (output). Могут быть анонимными и не анонимными. - Name - имя события;

№ п/п	Компонент	Пакет	Файл	Описание файла
				- Anonymous - флаг анонимности; - Inputs - массив информации, подлежащий сохранению, как причина события.
			abi/method.go	Реализация абстрактного метода виртуальной машины EVM. Метод может быть типа Const - в этом случае, не требуется изменения данных при вызове данного метода. Это может быть просто симулировано, используя локальную EVM. Например, метод Balance() нужен только для получения информации из хранилища и не требует отправки транзакции в сеть. В другом случае, метод Transact() требует отправки транзакции в сеть и должен быть оплачен. - Name - имя метода; - Const - флаг необходимости изменений; - Inputs - массив аргументов метода; - Outputs - массив возвращаемых значений.
			abi/numbers.go	Согласование чисельных типов данных для работы abi.
			abi/pack.go	Согласование байтовых массивов для работы abi.
			abi/reflect.go	Согласование типов данных из строкового ABI описания.
			abi/type.go	Type - отражение (reflection) аргумента поддерживаемого типа.
			abi/unpack.go	Unpacker - интерфейс, позволяющий иметь абстракцию между событиями и методами и надлежащим образом «распаковывать» их.

№ п/п	Компонент	Пакет	Файл	Описание файла
			abi/bind	
			abi/bind/auth.go	NewKeyedTransactor - функция, создающая подписанта транзакции (signer) из приватного ключа.
			abi/bind/backend.go	ContractBackend - интерфейс, определяющий интерфейсы для взаимодействия с контрактом в режиме чтения-записи; ContractCaller - интерфейс, определяющий методы, нужные для взаимодействия с контрактом в режиме чтения; ContractTransactor - интерфейс, определяющий методы, нужные для взаимодействия с контрактом в режиме записи; DeployBackend - интерфейс, предоставляющий функции нужные WaitMined и WaitDeployed (см. abi/bind/util.go); PendingContractCaller - интерфейс, предоставляющий функции вызова (contract calls) для взаимодействия с ожидающим состоянием (pending state).
			abi/bind/base.go	BoundContract - обёртка, которая отражает контракт по конкретному адресу в сети. Содержит коллекцию методов контракта (ABI), интерфейсы ввода-вывода для взаимодействия с PP; CallOpts - набор опций для настройки вызова (call) контракта; TransactOpts - набор авторизационных данных для создания корректной транзакции. Включает в себя указание конкретной функции подписи транзакции (SignerFn), соль, адрес отправителя, объем отправляемой валюты (в wei), цену газа, лимит газа и контекст транзакции для поддержки её возможной отмены;

№ п/п	Компонент	Пакет	Файл	Описание файла
				SignerFn - колбэк функции, вызываемый в случае требования контрактом метода для подписания транзакции ДО отправки в сеть.
			abi/bind/bind.go	Bind - функция, которая создаёт Go-обёртку для ABI контракта.
			abi/bind/template.go	tmplContract - структура, содержащая данные для биндинга к конкретному контракту. Включает ABI, EVM байткод, конструктор, функции вызовов (call) и функции транзакций (write); tmplMethod - обёртка для метода из ABI, содержащая предобработанные и кэшированные поля данных.
			abi/bind/util.go	WaitMined - функция, ожидающая исполнение транзакции; WaitDeployed - функция, ожидающая исполнение транзакции, создающей контракт и возвращающая адрес контракта, когда транзакция будет смайнена.
			abi/bind/backends	
			abi/bind/backends/simulated.go	SimulatedBackend - структура, реализующая bind.ContractBackend, симулируя существование PP. Её назначение - тестировать вызовы контракта. Содержит фейковый генезис и PP, хранимые в памяти; NewSimulatedBackend - функция, создающая SimulatedBackend для задач тестирования.
		cmd	abigen/main.go	Утилита для генерации ABI для предоставленного файла исходного кода смарт-контракта.

№ п/п	Компонент	Пакет	Файл	Описание файла
			bootnode/main.go	Реализация бутноды - узла сети, взаимодействующего только по P2P и предоставляющего другим узлам список доступных узлов.
			evm	
			evm/compiler.go	Код для компилятора EVM assembler в EVM байт-код.
			evm/disasm.go	Встроенный дизассемблер байткода.
			evm/json_logger.go	Инструмент логирования работы EVM в формате json.
			evm/main.go	Консольная утилита для выполнения EVM-кода.
			evm/runner.go	Код для мониторинга состояний EVM при выполнении кода.
			evm/staterunner.go	Код для трассировки состояний EVM.
			evm/internal/compiler	
			evm/internal/compiler/compiler.go	Исходный кода компилятора EASM (Ethereum assembler) в бинарный код.
			geth	
			geth/accountcmd.go	Функции управления аккаунтами: - account list - вывод в консоль всех доступных аккаунтов; - account new - создание нового аккаунта; - account update - изменение пароля аккаунта.

№ п/п	Компонент	Пакет	Файл	Описание файла
			geth/bugcmd.go	Функция bug предоставляет пользователю возможность сообщить об ошибке разработчикам.
			geth/chaincmd.go	Функции для работы с цепочкой: - init- инициализация нового PP с помощью предоставленного genesis.json; - impor - импорт блоков в PP из файла. Блоки должны быть предоставлены в виде файла в RLP-кодированном виде. Возможно использование одного файла с несколькими RLP-кодированными блоками, либо использование нескольких файлов; - export - экспорт заданных блоков из PP в файл; - removedb - удаление PP в заданной директории; - dump - дамп конкретного блока из хранилища.
			geth/config.go	Парсер конфигурационного TOML-файла или флагов.
			geth/consolecmd.go	Реализация функций интерактивной Javascript-консоли.
			geth/main.go	Точка входа в приложение meth. Объединяет в себе все возможные флаги запуска и запускает полную ноду. Модульная архитектура позволяет подключить файлы с дополнительными функциями.
			geth/misccmd.go	- Реализация дополнительных функций ноды: - makecache - создание верификационного кэша ethash (для тестирования); - makedag - создание direct acyclic graph (DAG) (для тестирования); - version - вывод версии приложения и завершение приложения.

№ п/п	Компонент	Пакет	Файл	Описание файла
			geth/monitorcmd.go	Реализация функций мониторинга ноды: - Monitor - отображение метрик конкретной ноды.
			geth/usage.go	Содержит шаблон и генератор описания использования команды meth.
			geth/testdata	Директория с тестовыми данными для эталонного geth Ethereum.
			puppeth	
			rlpdump/main.go	rlpdump - утилита для вывода RLP-данных в отформатированном виде.
			utils	
			utils/cmd.go	Код, запускающий логику в различных режимах (в соответствии с передаваемыми флагами).
			utils/customflags.go	Дополнительный обработчик передаваемых с флагом значений.
			utils/fdlimit_freebsd.go	Дополнительный обработчик флагов для файловой системы freebsd.
			utils/fdlimit_unix.go	Дополнительный обработчик флагов для файловой системы unix.
			utils/fdlimit_windows.go	Дополнительный обработчик флагов для файловой системы windows.
			utils/flags.go	Набор флагов и парсер.
		common	big.go	Описание типов Big*.
			bytes.go	Набор функций для работы с hex-представлением данных.

№ п/п	Компонент	Пакет	Файл	Описание файла
			debug.go	Код конструирующий.
			format.go	Код форматирует данные для удобства чтения человеком.
			path.go	Код, формирующий «баннер» узла.
			size.go	Код форматирует данные для удобства чтения человеком.
			test_utils.go	json-парсер.
			types.go	Библиотека типов.
			types_template.go	Шаблон для функций конвертации типов.
			bitutil	
			bitutil/bitutil.go	Код для ускорения битовых операций.
			bitutil/compress.go	Код для сжатия данных.
			bitutil/compress_fuzz.go	Код для тестирования операций кодирования-декодирования.
			compiler	
			compiler/solidity.go	Код для использования внешнего компилятора Solidity при запросе компиляции через API.
			hexutil	

№ п/п	Компонент	Пакет	Файл	Описание файла
			hexutil/hexutil.go	Код для работы со строковым представлением HEX-данных.
			hexutil/json.go	json-парсер.
			math	
			math/big.go	Набор функций для работы с большими числами.
			math/integer.go	Набор функций для работы для конфертации целых чисел в различные типы.
			mclock	
			mclock/mclock.go	Код для получения времени.
			number	
			number/int.go	Библиотека математических операций с целыми числами.
		contracts	**whitelist**	
			whitelist/contract.go	Описание интерфейса для взаимодействия со смарт-контрактами цепочки из кода узла.
			whitelist/whitelist.go	Настройки, необходимые для работы смарт-контракта.
		eth	api.go	PublicEthereumAPI - предоставляет API для доступа к информации о ноде Мастерчейн.

№ п/п	Компонент	Пакет	Файл	Описание файла
				PublicMinerAPI - предоставляет API для контроля майнинговых операций. Предоставляет методы только для операций с данными, не имеющих риска с точки зрения безопасности при публичном доступе к этим данным.
			api_backend.go	EthApiBackend реализует ethapi.Backend для полнофункциональных нод.
			backend.go	Ethereum - структура, реализующая функционал сервиса Ethereum: - config *Config - chainConfig *params.ChainConfig - shutdownChan chan bool // Канал для прекращения работы Ethereum; - stopDbUpgrade func() error // Остановка PP, последовательное обновление ключа базы данных; - txPool *core.TxPool - blockchain *core.BlockChain - protocolManager *ProtocolManager - lesServer LesServer - chainDb ethdb.Database // База данных PP; - eventMux *event.TypeMux - engine consensus.Engine - accountManager *accounts.Manager - bloomRequests chan chan *bloombits.Retrieval // Канал получения запросов на получение данных фильтров Блума;

№ п/п	Компонент	Пакет	Файл	Описание файла
				- bloomIndexer *core.ChainIndexer Блум индексатор, работающий при импорте блоков; - ApiBackend *EthApiBackend - Miner *miner.Miner - gasPrice *big.Int - etherbase common.Address - networkId uint64 - netRPCService *ethapi.PublicNetAPI - lock sync.RWMutex // Предохраняет вариативные поля (например, цену Газа и etherbase).
			bind.go	ContractBackend - структура, содержащая указатели на набор API, реализующих прямые вызовы внутренней логики Мастерчейн. Необходима для поддержки взаимодействия контрактов с субпротоколом eth. - Eapi - обёртка объектов Мастерчейн для доступа к метаданным; - Bcapi - обёртка объектов PP для доступа к данным в цепи; - Txapi - обёртка пула транзакций для доступа к данным транзакций.

№ п/п	Компонент	Пакет	Файл	Описание файла
			bloombits.go	Bloom bits - битовый массив в заголовке блока, по которому осуществляется поиск, используя Фильтр Блума. Фильтр Блума - это вероятностная структура данных, позволяющая проверять принадлежность элемента к множеству. При этом существует возможность получить ложноположительное срабатывание (элемента в множестве нет, но структура данных сообщает, что он есть), но не ложноотрицательное. В то время, когда блок сгенерирован или верифицирован, адрес любого логируемого контракта и все indexed значения из логов, созданных при выполнении транзакций добавляются в фильтр, который включается в заголовок блока. Полные логи транзакций не включаются в блок для экономии места. Когда приложение ищет все события по конкретному контракту или конкретные значения полей indexed, узел быстро считывает заголовки всех блоков, проверяя, может ли фильтр содержать релевантные данные. Если фильтр блока содержит такие данные, узел выполняет заново транзакции из данного блока, восстанавливая логи на узле и возвращает конкретные результаты приложению. BloomIndexer - структура, реализующая core.ChainIndexer, создавая обновляемый индекс bloom bits для заголовков блоков.
			config.go	Config - структура, содержащая конфигурацию узла Мастерчейн: - Genesis *core.Genesis `toml:",omitempty"` - генезис-блок (нулевой), который будет записан в PP, если база данных пуста;

№ п/п	Компонент	Пакет	Файл	Описание файла
				- NetworkId uint64 - идентификатор сети, проверяется при подключении к другим участникам сети. Позволяет разворачивать публичные и приватные сети; - SyncMode downloader.SyncMode - все доступные методы кроме FullSync удалены; - SkipBcVersionCheck - настройка, отменяющая проверку версии базы данных; - NodeKeyContainer - ID контейнера КриптоПро приватного ключа узла Мастерчейн; - Etherbase - адрес по умолчанию для получения вознаграждения (TPE) если данный узел смайнил блок; - MinerThreads - количество доступных для майнинга потоков процессора; - ExtraData - дополнительные данные, записанные в генезис-блок PP; - GasPrice - цена Газа для вычислительных операций в смарт-контрактах; - Ethash - настройки Ethash POW алгоритма, необходимого для майнинга; - TxPool - настройки пула транзакций; - GPO gasprice.Config // Опции для Gas Price Oracle (модуль, который предсказывает (вычисляет) цену на Газ); - EnablePreimageRecording // Разрешение отслеживания SHA3 прообразов в M.

№ п/п	Компонент	Пакет	Файл	Описание файла
			db_upgrade.go	upgradeDeduplicateData() проверяет версию базы PP и запускает процесс обновления, если необходимо. Возвращает стоп-функцию, которая блокирует обращения к базе до завершения процесса обновления.
			gen_config.go	Генерирует объект Config из файла в формате toml. Конкретные настройки можно сохранить в toml-файл, полезно при доставке решений клиентам.
			handler.go	ProtocolManager - структура, содержащая в себе различные подпротоколы и исполнители, которые обеспечивают функционирование узла в целом. Среди них: - txpool - объект пула транзакций; - blockchain - объект PP; - chaindb - объект базы данных, хранящей PP; - chainconfig - объект ChainConfig (конфигурация PP); - downloader - объект загрузчика блоков; - fetcher - объект сборщика данных о доступных блоках; - peers - набор известных узлу пиров; - SubProtocols - массив P2P протоколов, доступных узлу.
			metrics.go	Набор метрик, собираемых подсистемой eth.

№ п/п	Компонент	Пакет	Файл	Описание файла
			peer.go	Репрезентация конкретного узла в подсистеме eth. Экземпляр структуры peer содержит сетевые данные о конкретном узле, версию протокола, идентификатор, хеш заголовочного блока пира, сложность, массив транзакций и блоков, известных узлу. Так же, в файле содержится набор функций для работы с узлом: хендшейк, чтение статуса удалённого узла, запрос конкретных данных (блоки, заголовки и другое).
			protocol.go	Константы версий протокола и коды сообщений.
			sync.go	syncer() - функция структуры ProtocolManager, ответственная за периодическую синхронизацию узла с сетью. Получает анонсы и загружает хеши и блоки. Процесс выбора лучшего пира (best peer) для синхронизации представляет собой выборку из общего списка пиров по полной сложности (total difficulty). synchronize() - функция, производящая синхронизацию локального чейна с удалённым узлом в режиме FullSync.
			downloader	
			downloader/api.go	PublicDownloaderAPI - структура, содержащая API, дающее информацию о текущем статусе синхронизации. Предоставляет только функции для обращения к данным, которые могут быть доступны всем (anyone) без риска нарушения безопасности.
			downloader/downloader.go	Режимы синхронизации:

№ п/п	Компонент	Пакет	Файл	Описание файла
				- full - загрузка всех блоков и валидация всех транзакций в PP; - Downloader - структура, реализующая функционал по загрузке блоков; - BlockChain - интерфейс, инкапсулирующий фукнции, необходимые для полной синхронизации (full or fast).
			downloader/events.go	Набор событий, используемых для управления процессом загрузки блоков.
			downloader/fakepeer.go	FakePeer - структура, реализующая пир без сетевых функций, но включающая в себя загрузчик блоков (downloader). Применима для тестовых нужд и в операциях по синхронизации с существующей локальной базой данных.
			downloader/metrics.go	Набор различных метрик, собираемых подсистемой Downloader.
			downloader/modes.go	Набор констант режимов синхронизации.
			downloader/peer.go	Peer - интерфейс, инкапсулирующий функции, необходимые для синхронизации с удалённым пиром. peer - структура, представляющая конкретный активный пир, с которого происходит получение хешей и блоков. peerSet - структура, хранящая информацию об активных пирах, участвующих в процедуре загрузке блоков.
			downloader/queue.go	Queue - структура, хранящая очереди заданий по получению данных по конкретным хешам.

№ п/п	Компонент	Пакет	Файл	Описание файла
			downloader/statesync.go	Набор структур данных, для передачи запросов по загрузке блоков, объединяющих несколько запросов в один пакет.
			downloader/types.go	Набор структур данных, используемых для загрузки блоков.
			fetcher	
			fetcher/fetcher.go	Fetcher - структура, содержащая исполнительные элементы, отвечающие за накопление объявлений о блоках (block announcements) от других пиров и планирование их загрузки.
			fetcher/metrics.go	Набор различных метрик, собираемых подсистемой Fetcher.
			filters	
			filters/api.go	Описание встроенных фильтров для заданных типов событий и интерфейсов для взыимодействия с этими фильтрами.
			filters/filter.go	Пакет filters реализует систему поиска для блоков, транзакций и событий. Filter - структура для получения и фильтрации логов. Backend - интерфейс, инкапсулирующий функции над Filter.
			filters/filter_system.go	EventSystem - структура, реализующая подписки на события, обрабатывающая события и передающая их в подписку, в соответствии с заданными критериями. Subscription - объект данной структуры создаётся, когда клиент регистрирует себя на ожидание конкретного события (подписка на события).

№ п/п	Компонент	Пакет	Файл	Описание файла
			gasprice	
			gasprice/gasprice.go	Oracle - структура, хранящая цену газа. Текущая цена газа вычисляется исходя из цены газа некоторого количества последних блоков. Oracle рекомендует (suggest) цену газа, но не выставляет жёсткое условие.
		event	event.go	Код реализующий основной функционал для работы с событиями.
			feed.go	Код реализующий «подписку на фиды» событий: информация о событии возвращается не по запросу, а отправляется по факту обнаружения в интерфейс, через который установлена подписка.
			subscription.go	Код реализующий «подписку» на конкретные события: информация о событии хешируется по факту обнаружения. Используется для того, чтобы не искать события каждый раз по всей цепочке.
		internal	/ethapi	Описание интерфейсов, которые используются для взаимодействия между компанентами системы.
			/ethapi/addrlock.go	Блокировка mutex при процедуре создания транзакции.
			/ethapi/api.go	Функции для операций доступных через RPC.
			/ethapi/backend.go	Описание интерфейса для получения данных, необходимых для работы RPC и взаимодействия узлов между собой.
			/ethapi/tracer.go	Библиотека для контроля исполнения Javascript при работе консоли Мастерчейн.

№ п/п	Компонент	Пакет	Файл	Описание файла
			/jsre	Система интерпретации Javascript, используемая при работе в консоли Мастерчейн.
			/jsre/deps	Директория с JS зависимостями необходимыми для работы консоли.
			/jsre/completion.go	Логика автодополнений в консоли.
			/jsre/jsre.go	RunTime Environment для Java Script.
			/jsre/pretty.go	Инструменты для форматирования вывода.
			/web3ext/web3ext.go	Описание интерфейсов для согласования структуры между Java Script RunTime Environment консоли и СКЗИ Мастерчейн.
4.	**Компонент взаимодействия с криптосредствами**	**crypto**	**sha3/**	
			sha3/doc.go	Комментарий.
			sha3/hashes.go	Описание интерфейсов.
			sha3/keccakf.go	Алгоритм KECCAK (основа sha3).
			sha3/keccakf_amd64.go	Интерфейс к оптимизированному модулю.
			sha3/register.go	Код для подготовки интерфейсов.
			sha3/sha3.go	Функции подготовки данных.
			sha3/shake.go	Описание работы стримов (абстракция go-lang) для ввода\вывода данных.

№ п/п	Компонент	Пакет	Файл	Описание файла
			sha3/xor.go	Режим для xor-операций.
			sha3/xor_generic.go	Описание режима работы xor.
			sha3/xor_unaligned.go	Описание режима работы xor.
			sha3/LICENSE	Лицензия на sha3.
			sha3/PATENTS	Патенты sha3.
		crypto-csp	common.go	Базовые операции.
			crypto.go	Описание функций, используемых для крипто-преобразований в СКЗИ Мастерчейн.
			cspc.go	Функции, обеспечивающие уровень абстракции между CSP и кодом СКЗИ Мастерчейн.
			curve.go	Функции для работы с кривой, создающей подпись. Используются для подготовки данных для CSP.
			hash.go	Интерфейс для сертифицированного хеширования в CSP.
			params.go	Параметры кривой.
			public.go	Набор функций для работы с публичным ключом.
			testing_utils.go	Набор тестов для работы с CSP.

№ п/п	Компонент	Пакет	Файл	Описание файла
5.	**Компонент адресации участников сетевого взаимодействия**	**accounts**	accounts.go	Содержит следующие структуры данных: Account - представляет аккаунт Мастерчейн, расположенный по заданному адресу [address, URL]. Wallet - представляет программное или аппаратное хранилище (кошелёк), которое может содержать один или несколько аккаунтов. Backend – «поставщик кошелька», который может содержать набор аккаунтов, которые могут подписывать транзакции по запросу и функций, выполняющих эту работу. WalletEvent - событие, создаваемое backend аккаунта, при выполнении следующих действий: - WalletArrived - подключение нового кошелька через USB или файловую систему; - WalletOpened - успешное открытие кошелька; - WalletDropped - закрытие (удаление) кошелька.
			errors.go	Набор переменных, содержащий коды ошибок, возникающих при работе с аккаунтами и их описание.
			manager.go	Содержит структуру данных Manager и функции для управления аккаунтами. Manager - управляющая структура для общения с различными backends для подписания транзакций.
			url.go	URL - структура данных, представляющая идентификатор места хранения ключа:

№ п/п	Компонент	Пакет	Файл	Описание файла
				- Scheme - наименование протокола связи; - Path - адрес, согласно протокола.
			keystore	
			keystore/account_cache.go	accountCache - реестр всех аккаунтов, находящихся в хранилище. Файл так же содержит функции работы с реестром аккаунтов.
			keystore/file_cache.go	fileCache - кэш файлов, обработанных при последнем обращении к хранилищу.
			keystore/key.go	Key - структура, хранящая данные об аккаунте: - Id - идентификатор аккаунта; - Address - адрес аккаунта; - KeyContainer - имя контейнера КриптоПро, хранящего приватный ключ; - Passphrase - пароль для контейнера. keyStore - интерфейс, предоставляющий функции чтения и записи ключа с/на диск.
			keystore/keystore.go	KeyStore - структура, отображающая хранилище аккаутов на жёстком диске. Содержит список кошельков, интерфейс к хранилищу, реестр аккаунтов (accountCache), список разблокированных в данное время адресов.
			keystore/keystore_plain.go	keyStorePlain - структура, хранящая адрес хранилища ключей (keystore) в файловой системе.

№ п/п	Компонент	Пакет	Файл	Описание файла
			keystore/keystore_wallet.go	keystoreWallet реализует интерфейс accounts.Wallet для конкретного аккаунта в хранилище ключей.
			keystore/watch.go	watcher - наблюдатель за директорией хранилища ключей. Реагирует на события, возникающие на уровне файловой системы.
			keystore/watch_fallback.go	Реализация watcher для неподдерживаемых платформ.
			keystore/testdata	Директория данных для тестов accounts.
6.	**Компонент взаимодействия с пользователями**	**ethclient**	ethclient.go	Интерфейс для функций клиента RPC.
			signer.go	Функции для подписи данных через RPC интерфейс.
		console	bridge.go	Bridge - коллекция Javascript-методов, связывающих js runtime и Go RPC. OpenWallet - обёртка для personal.openWallet, позволяющая обрабатывать сообщения об ошибках или запросы PIN-кода для аппаратных ключей; UnlockAccount - обёртка для personal.unlockAccout. Использует строку ввода пароля, не отображающую символы для получения ключевой фразы и вызова оригинального RPC-метода разблокировки аккаунта; Sign - обёртка для personal.sign. Использует строку ввода пароля, не отображающую символы для получения ключевой фразы и вызова оригинального RPC-метода подписи транзакции; Sleep - блокирует консоль на заданное количество секунд; SleepBlocks - блокирует консоль на заданное количество блоков. Опционально можно задать таймаут для разблокировки;

№ п/п	Компонент	Пакет	Файл	Описание файла
				Send - реализует web3-метод send.
			console.go	Config - коллекция конфигураций для настройки Javascript-консоли; Console - интерактивное окружение для выполнения Javascript-кода.
			prompter.go	UserPrompter - интерфейс, предоставляющий методы необходимые консоли для запроса ввода у пользователя: - PromptInput - отображает переданный пользователю запрос и запрашивает ввод какого-либо текста, возвращая ввод пользователю; - PromptPassword - отображает переданный пользователю запрос и запрашивает ввод какого-либо текста, возвращая ввод пользователю. При этом, вводимый пользователем текст не отображается на экране; - PromptConfirm - отображает переданный пользователю запрос и запрашивает булевый выбор, возвращая этот выбор; - SetHistory - устанавливает глубину истории вводимых команд; - AppendHistory - добавляет объект в историю ввода. Должен быть вызван, тогда и только тогда, когда выполнилась валидная команда; - SetWordCompleter - устанавливает функцию дополнения ввода, которую prompter будет вызывать для выборки возможных значений для автодополнения команды, когда пользователь нажмёт Tab.
			testdata	Тестовые данные для тестирования пакета console.
		rpc	client.go	Код клиента для подключения консоли к RPC (HTTP/IPC) узла. Узел можно запускать в режиме RPC-клиента.

№ п/п	Компонент	Пакет	Файл	Описание файла
			doc.go	Комментарий.
			errors.go	Описание ошибок.
			http.go	HTTP-сервер для HTTP RPC.
			inproc.go	Код подключения клиента.
			ipc.go	IPC-сервер для IPC RPC.
			ipc_unix.go	Режим IPC для unix-систем.
			ipc_windows.go	Режим IPC для win-систем.
			json.go	JSON-кодек для обработки RPC-сообщений.
			server.go	Код RPC-сервера.
			subscription.go	Система «подписки» на события (через RPC).
			types.go	Описание типов и интерфейсов.
			utils.go	Код для нормализации данных и управления RPC-сессиями.
			websocket.go	Код для работы RPC через веб-сокеты.
7.	**Компонент мониторинга**	**ethstats**	ethstats.go	Пакет для внутреннего мониторинга сетевой активности.
		log	**term/**	

№ п/п	Компонент	Пакет	Файл	Описание файла
			doc.go	Комментарий.
			format.go	Описание формата представления данных.
			handler.go	Код, описывающий разные режимы работы провайдера логов.
			handler_glog.go	Код для запуска провайдеров с индивидуальными настройками.
			handler_go13.go	Настройки логгера под конкретную версию Go.
			handler_go14.go	Настройки логгера под конкретную версию Go.
			logger.go	Код инициализации для использования логгера в СКЗИ Мастерчейн.
			root.go	Описание дерева провайдеров логов.
			syslog.go	Интерфейс для передачи логов в формате syslog.
8.	-	**mastertests**	-	Тесты функций СКЗИ Мастерчейн, например, развёртывание контрактов, восстановление публичного ключа из подписи.
		img	-	Пакет для статических изображений для wiki в GitHub-репозитории, не содержит функций.
		params	bootnodes.go	Пустой объект, в котором могут быть заданы стартовые узлы (bootnodes). Сейчас они передаются при старте узла.
			config.go	Разнообразные константы для разных режимов работы.

№ п/п	Компонент	Пакет	Файл	Описание файла
			denomination.go	- Wei = 1; - Ada = 1e3; - Babbage = 1e6; - Shannon = 1e9; - Szabo = 1e12; - Finney = 1e15; - Ether = 1e18; - Einstein = 1e21; - Douglas = 1e42.
			gas_table.go	Таблица с заданными ценами за газ для операций EVM для разных режимов работы.
			network_params.go	Параметр для структуры данных.
			protocol_params.go	Параметры протокола PP.
			version.go	Описание объекта, в котором сохраняется версия сборки узла.
		tests	-	Набор тестов, при компиляции не используется.
		vendor	-	Набор библиотек, используемых платформой Мастерчейн.
		build	-	Содержит скрипты для сборки СКЗИ Мастерчейн, не используется при работе СКЗИ Мастерчейн.

№ п/п	Компонент	Пакет	Файл	Описание файла
		containers	-	Пакет содержит файлы контейнеров Docker и Vagrant, предназначенных для запуска Мастерчейн в изолированных контейнерах.

ПРИЛОЖЕНИЕ О. ОПИСАНИЕ ФУНКЦИЙ КРИПТОАДАПТЕРА

№ п/п	Функция криптоадаптера	Назначение	Входные параметры	Выходные параметры	Функции криптопровайдера КриптоПро CSP
1.	Gost34112012_256(data)	Вычисление хеш-функции согласно стандарту ГОСТ 34.11-2012 256 бит.	Byte[] data – массив байтов, от которых необходимо посчитать хеш-функцию.	Byte[] – массив байтов – вычисленный хеш. Длина 32 байта.	1. CryptAcquireContext() – получение дескриптора криптопровайдера; 2. CryptCreateHash() – создание объекта хеширования; 3. CryptHashData() – вычисление хеша; 4. CryptGetHashParam() – получение хеша в виде массива байтов из объекта хеширования.
2.	Gost34112012_256Hash(data)	Вычисление хеш-функции согласно стандарту ГОСТ 34.11-2012 256 бит.	Byte[] data – массив байтов, от которых необходимо посчитать хеш-функцию.	Hash – тип обертка над byte[].	1. CryptAcquireContext() – получение дескриптора криптопровайдера; 2. CryptCreateHash() – создание объекта хеширования; 3. CryptHashData() – вычисление хеша; 4. CryptGetHashParam() – получение хеша в виде массива байтов из объекта хеширования.
3.	LoadContainerInfo(filename)	Получение имени ключевого контейнера из файла, хранящегося в файловой системе.	String filename – имя файла, в котором находится информация о имени необходимого для работы ключевого контейнера.	1. String – имя ключевого контейнера; 2. Error – тип ошибки, в случае успеха nil.	Функции КриптоПро CSP не используются.

№ п/п	Функция криптоадаптера	Назначение	Входные параметры	Выходные параметры	Функции криптопровайдера КриптоПро CSP
4.	SaveContainerInfo(filename)	Сохранение имени ключевого контейнера в файл, хранящийся в файловой системе.	String filename – имя файла, в котором находится информация о имени необходимого для работы ключевого контейнера	Error – тип ошибки, в случае успеха nil.	Функции КриптоПро CSP не используются.
5.	CreateNewKeysContainer(passphrase)	Создание нового ключевого контейнера. Имя контейнера задается системой. В качестве имени используется hex представление открытого ключа, хранящегося в ключевом контейнере.	String passphrase – пин код ключевого контейнера.	String – имя ключевого контейнера.	1. CryptAcquireContext() – получение дескриптора криптопровайдера; 2. CryptSetProvParam() – установка пин-кода ключевого контейнера; 3. CryptGenKey() – генерация ключа.
6.	DeleteKeysContainer(container, passphras)	Удаление ключевого контейнера.	1. String container – имя ключевого контейнера; 2. String passphrase – пин код ключевого контейнера.	void	1. CryptAcquireContext() – получение дескриптора криптопровайдера; 2. CryptSetProvParam() – установка пин-кода ключевого контейнера; 3. CryptSetProvParam() – удаление ключевого контейнера.
7.	UpdateKeysContainerPin(container, passphrase, newPassphrase)	Обновления пин-кода ключевого контейнера.	1. String container – имя ключевого контейнера; 2. String passphrase – текущий пин код ключевого контейнера;	void	1. CryptAcquireContext() – получение дескриптора криптопровайдера; 2. CryptSetProvParam() – установка пин-кода ключевого контейнера;

№ п/п	Функция криптоадаптера	Назначение	Входные параметры	Выходные параметры	Функции криптопровайдера КриптоПро CSP
			3. String newPassphrase – новый пин ключевого контейнера.		3. CryptSetProvParam() – обновление пин-кода ключевого контейнера.
8.	GetPublicKey(container, passphrase)	Получение публичного ключа по ключевому контейнеру.	1. String container – имя ключевого контейнера; 2. String passphrase – пин код ключевого контейнера.	Byte[] – массив байтов – значение открытого ключа. Длина 64 байта.	1. CryptAcquireContext() – получение дескриптора криптопровайдера; 2. CryptSetProvParam() – установка пин-кода ключевого контейнера; 3. CryptGetUserKey() – получение объекта ключа; 4. CryptExportKey() – экспорт открытого ключа.
9.	Sign(container, passphrase, hash)	Получение подписи хеша на ключе по ключевому контейнеру. В качестве алгоритма подписи используется ГОСТ 34.10-2012 256 бит.	1. String container – имя ключевого контейнера; 2. String passphrase – пин код ключевого контейнера; 3. Byte[] hash – хеш в виде массива байтов, который необходимо подписать.	Byte[] – массив байтов – значение вычисленной подписи. Длина 65 байтов: 64 байта подписи по ГОСТ 34.10.2012 256 бит + 1 байт четности для функции восстановления открытого ключа.	1. CryptAcquireContext() – получение дескриптора криптопровайдера; 2. CryptSetProvParam() – установка пин-кода ключевого контейнера; 3. CryptCreateHash() – создание объекта хеширования; 4. CryptSetHashParam() – установка фактического значения хеша для дальнейшей подписи; 5. CryptSignHash() – вычисление подписи.

№ п/п	Функция криптоадаптера	Назначение	Входные параметры	Выходные параметры	Функции криптопровайдера КриптоПро CSP
10.	Ecrecover(hash, signature)	Получение публичного ключа в виде массива байтов по оригинальному сообщению и его подписи.	1. Byte[] hash – оригинальный хеш в виде байт массива; 2. Byte[] signature – подпись от hash в виде байт массива.	Byte[] – массив байтов – значение открытого ключа в сериализованном виде. Длина 65 байт: 1 байт со значением «4» (служебная информация о сериализации) + 64 байта открытого ключа.	CryptVerifySignature()– проверка подписи.
11.	SigToPub(hash, signature)	Получение публичного ключа в виде специального типа по оригинальному сообщению и его подписи.	1. Byte[] hash – оригинальный хеш в виде байт массива; 2. Byte[] signature – подпись от hash в виде байт массива.	1. *PublicKey – тип обертка над открытым ключом; 2. Error – тип ошибки, в случае успеха nil.	Функции КриптоПро CSP не используются.
12.	PubkeyBytesToAddress(Byte[] key)	Преобразование публичного ключа в тип адреса.	Byte[] key – открытый ключ в виде массива байтов.	common.Address – тип адреса.	Функции КриптоПро CSP не используются.
13.	PublicKeyBytes(pubKey)	Преобразование публичного ключа из специального типа в массив байтов.	*PublicKey pubKey – тип обертка над открытым ключом.	Byte[] – открытый ключ в виде массива байтов.	Функции КриптоПро CSP не используются.
14.	InitTLS(container, passphrase)	Создание SSP мандатов от лица клиента (для исходящего соединения) и от лица сервера (для входящего соединения).	1. String container – имя ключевого контейнера, ключ которого используется для создания самоподписанного сертификата для создания безопасного канала;	Error – тип ошибки, в случае успеха nil.	1. InitSecurityInterfaceA() – инициализация таблицы функций SSP; 2. AcquireCredentialsHandle() – создание удостоверения пакета безопасности.

№ п/п	Функция криптоадаптера	Назначение	Входные параметры	Выходные параметры	Функции криптопровайдера КриптоПро CSP
			2. String passphrase – пин код ключевого контейнера.		Для создания удостоверений безопасности в системе на основе открытого ключа, лежащего в ключевом контейнере ноды, создается самоподписанный сертификат. Для его создания используются следующие функции: 1. CryptEncodeObject() – кодирование информации, необходимой для создания сертификата в формате ASN.1; 2. CryptGenRandom() – получение псевдослучайного числа для генерации серийного номера сертификата; 3. CryptExportPublicKeyInfo() – экспорт открытого ключа в виде BLOB для дальнейшего помещения в сертификат; 4. CryptSignAndEncodeCertificate() – создание объекта сертификата, его подпись и кодирование в формате ASN.1; 5. CertCreateCertificateContext() – создание контекста сертификата для

№ п/п	Функция криптоадаптера	Назначение	Входные параметры	Выходные параметры	Функции криптопровайдера КриптоПро CSP
					использования в рамках обмена информации по протоколу tls; 6. CertSetCertificateContextProperty() – установка ключевого контейнера для контекста сертификата.
15.	TlsClientHello(node)	Формирование Hello сообщения клиента для отправки на сервер.	Byte[] node – открытый ключ удаленного узла, с которой происходит установление соединения.	ConnectionContext – контекст соединения. Инкапсулирует в себе мандат соединения, сессию и пакет ответа для сервера.	InitializeSecurityContext() – создание клиентского контекста безопасности. Возвращает handshake-сообщения, которые клиент должен передать серверу, и обрабатывает ответные handshake-сообщения.
16.	TlsServerHello(packetIn)	Обработка Hello сообщения клиента и формирование ответного Hello сообщения сервера для отправки на клиент.	Byte[] packetIn – Hello сообщение от клиента.	ConnectionContext – контекст соединения. Инкапсулирует в себе мандат соединения, сессию и пакет ответа для клиента.	AcceptSecurityContext() – создание серверного контекста безопасности. Возвращает handshake-сообщения, которые сервер должен передать клиенту, и обрабатывает ответные handshake-сообщения.

№ п/п	Функция криптоадаптера	Назначение	Входные параметры	Выходные параметры	Функции криптопровайдера КриптоПро CSP
17.	TlsClientStep(packetIn)	Обработка сообщения в рамках процедуры handshake от сервера и формирование ответного сообщения от клиента. Функция вызывается до получения ответа SEC_E_OK.	Byte[] packet – пакет в рамках обмена handshake сообщениями от сервера.	void Пакет результата содержится в контексте соединения.	InitializeSecurityContext() – создает клиентский контекст безопасности. Возвращает handshake-сообщения, которые клиент должен передать серверу, и обрабатывает ответные handshake-сообщения.
18.	TlsServerStep(packetIn)	Обработка сообщения в рамках процедуры handshake от клиента и формирование ответного сообщения от сервера. Функция вызывается до получения ответа SEC_E_OK.	Byte[] packet – пакет в рамках обмена handshake сообщениями от клиента.	void Пакет результата содержится в контексте соединения.	AcceptSecurityContext() – создает серверный контекст безопасности. Возвращает handshake-сообщения, которые сервер должен передать клиенту, и обрабатывает ответные handshake-сообщения.
19.	TlsEncrypt(packetIn)	Зашифровывание сообщения от одной удаленной стороны к другой в рамках обмена сообщениями по протоколу tls.	Byte[] packet – пакет для зашифровывания.	void Пакет результата содержится в контексте соединения.	EncryptMessage() – зашифровывание сообщения с использованием сессионного ключа, выработанного совместно с удаленной стороной, и вычисляет имитовставку.

№ п/п	Функция криптоадаптера	Назначение	Входные параметры	Выходные параметры	Функции криптопровайдера КриптоПро CSP
20.	TlsDecrypt(packetIn)	Расшифровывание сообщения от одной удаленной стороны к другой в рамках обмена сообщениями по протоколу tls.	Byte[] packet – пакет для расшифровывания.	void Пакет результата содержится в контексте соединения	DecryptMessage() – расшифровывание сообщения, проверка его целостность и уведомление о получении запроса на переустановку связи или разрыв соединения.
21.	GetCertificate(string container, string passphrase)	Получение сертификата в двоичном представлении по имени контейнера для передачи стороннему пользователю, для осуществления функции шифрования на сертификате.	1. string container – имя ключевого контейнера; 2. string passphrase – пин код ключевого контейнера.	CertDescriptor: []byte – массив байтов, двоичное представление сертификата; int – размер массива; int – код ошибки; string – текст ошибки.	1. CertOpenSystemStore() – открытие хранилища сертификатов; 2. CertFindCertificateInStore() – поиск необходимого сертификата; 3. GetCertificateContext() – получение контекста сертификата; 4. CertFreeCertificateContext() – освобождение памяти контекста сертификата.
22.	EncryptWithCertificate(container, passphrase, msg, certificate)	Шифрование сообщения отправителем на сертификате приватного ключа получателя.	1. string container – имя ключевого контейнера; 2. string passphrase – пин код ключевого контейнера, 3. byte[] msg – сообщение, 4. byte[] certificate – двоичное представление сертификата.	BlobDescriptor: []byte – массив байтов, двоичное представление зашифрованного сообщения; int – размер массива; int – код ошибки; string – текст ошибки.	1. CertOpenSystemStore() – открытие хранилища сертификатов; 2. CertCreateCertificateContext() – создание контекста сертификата по двоичному представлению; 3. CryptEncryptMessage() – зашифровывание сообщения на контексте сертификата;

№ п/п	Функция криптоадаптера	Назначение	Входные параметры	Выходные параметры	Функции криптопровайдера КриптоПро CSP
					4. CertFreeCertificateContext() – освобождение памяти контекста сертификата.
23.	DecryptWithCertificate(container, passphrase, msg)	Расшифровывание сообщения получателем, зашифрованного отправителем на сертификате приватного ключа получателя.	1. string container – имя ключевого контейнера; 2. string passphrase – пин код ключевого контейнера; 3. byte[] msg – зашифрованное сообщение.	BlobDescriptor: []byte – массив байтов, двоичное представление расшифрованного сообщения; int – размер массива; int – код ошибки; string – текст ошибки.	1. CertOpenSystemStore() – открытие хранилища сертификатов; 2. CryptDecryptMessage() – расшифровывание сообщения на контексте сертификата, поиск нужного сертификата осуществляется по ссылке на хранилище автоматически; 3. CertFreeCertificateContext() – освобождение памяти контекста сертификата.

ПРИЛОЖЕНИЕ П. ОПИСАНИЕ RLP КОДИРОВАНИЯ

RLP используется для кодирования (сериализации) произвольно вложенных массивов двоичных данных (бинарных массивов). Назначение RLP – кодирование структуры.

RLP принимает на вход два типа данных:

- Строка (байтовый массив);
- Список строк.

При RLP кодировании никакие специальные кодировки не используются, и не подразумевается никаких знаний о содержании строк.

RLP кодирование (или сериализация) производится следующим образом:

1. Если входные данные – это один байт, значение которого находится в диапазоне [0x00, 0x7f], тогда этот байт (префикс) является его собственным RLP кодированием.
2. Если входные данные – это строка 0-55 байт, RLP кодировка таких данных: первый байт 0x80 + длина строки, за которым следует строка. Значение первого байта (префикса) кодировки находится в диапазоне [0x80, 0xb7].
3. Если входные данные – это строка длиной более 55 байт, RLP кодировка таких данных: первый байт (префикс) 0x80 + длина строки в двоичном формате, затем длина строки, за которым следует строка. Значение первого байта кодировки находится в диапазоне [0xb8, 0xbf].
4. Если входные данные – это список, суммарной длиной 0-55 байт, RLP кодировка таких данных: первый байт (префикс) 0xc0 плюс длина данных, затем RLP кодировки элементов списка. Значение первого байта кодировки находится в диапазоне [0xc0, 0xf7].
5. Если входные данные – это список, суммарной длиной более 55 байт, RLP кодировка таких данных: первый байт (префикс) 0xf7 плюс размер в байтах длины данных в двоичном формате, затем длина данных, затем RLP кодировки элементов списка. Значение первого байта кодировки находится в диапазоне [0xf8, 0xff].

RLP кодирование:

```
def rlp_encode(input):
    if isinstance(input,str):
        if len(input) == 1 and ord(input) < 0x80: return input
        else: return encode_length(len(input), 0x80) + input
    elif isinstance(input,list):
        output = ''
        for item in input: output += rlp_encode(item)
        return encode_length(len(output), 0xc0) + output

def encode_length(L,offset):
    if L < 56:
         return chr(L + offset)
    elif L < 256**8:
        BL = to_binary(L)
         Return chr(len(BL) + offset + 55) + BL
    else:
         raise Exception("input too long")

def to_binary(x):
```

```
    if x == 0:
        return ''
    else:
        return to_binary(int(x / 256)) + chr(x % 256)
```

Декодирование происходит следующим образом:

1. По первому байту (префикс) определяется тип данных, которые декодируются, размер и смещение.
2. В зависимости от типа данных и смещения декодировать данные соответствующим образом.
3. Декодировать оставшуюся часть ввода.

Входные данные для RLP декодирования – это массив двоичных данных.

Правила декодирования типов данных и смещение:

1. Значение первого байта (префикса) находится в диапазоне [0x00, 0x7f]. Тип данных: строка, длиной один байт. Значение префикс соответствует значению этого байта.
2. Значение первого байта (префикса) находится в диапазоне [0x80, 0xb7]. Тип данных: строка, длина строки: первый байт минус 0x80, строка следует за первым байтом.
3. Значение первого байта (префикса) находится в диапазоне [0xb8, 0xbf]. Тип данных: строка. Размер длины строки в байтах: первый байт минус 0xb7, длина строки следует за префиксом, сама строка следует за длиной строки.
4. Значение первого байта (префикса) находится в диапазоне [0xc0, 0xf7]. Тип данных: список, длина всех элементов списка: первый байт минус 0xc0, за префиксом идут конкатенированные элементы списка в RLP кодировке.
5. Значение первого байта (префикса) находится в диапазоне [0xf8, 0xff]. Тип данных: список. Размер длины всех элементов списка: первый байт минус 0xf7, длина всех элементов списка следует за префиксом, затем идут конкатенированные элементы списка в RLP кодировке.

RLP декодирование:

```
def rlp_decode(input):
    if len(input) == 0:
        return
    output = ''
    (offset, dataLen, type) = decode_length(input)
    If type is str:
        output = instantiate_str(substr(input, offset, dataLen))
    elif type is list:
        output = instantiate_list(substr(input, offset, dataLen))
    output + rlp_decode(substr(input, offset + dataLen))
    return output

def decode_length(input):
    length = len(input)
    if length == 0:
        raise Exception("input is null")
    prefix = ord(input[0])
    if prefix <= 0x7f:
        return (0, 1, str)
    elif prefix <= 0xb7 and length > prefix - 0x80:
        strLen = prefix - 0x80
        return (1, strLen, str)
    elif prefix <= 0xbf and length > prefix - 0xb7 and length > prefix - 0xb7 +
to_integer(substr(input, 1, prefix - 0xb7)):
```

```
        lenOfStrLen = prefix - 0xb7
        strLen = to_integer(substr(input, 1, lenOfStrLen))
        return (1 + lenOfStrLen, strLen, str)
    elif prefix <= 0xf7 and length > prefix - 0xc0:
            listLen = prefix - 0xc0;
        return (1, listLen, list)
    elif prefix <= 0xff and length > prefix - 0xf7 and length > prefix - 0xf7 +
to_integer(substr(input, 1, prefix - 0xf7)):
        lenOfListLen = prefix - 0xf7
        listLen = to_integer(substr(input, 1, lenOfListLen))
        return (1 + lenOfListLen, listLen, list)
    else:
        raise Exception("input don't conform RLP encoding form")

def to_integer(b):
    length = len(b)
    if length == 0:
        raise Exception("input is null")
    elif length == 1:
        return ord(b[0])
    else:
        return ord(substr(b, -1)) + to_integer(substr(b, 0, -1)) * 256
```

ПРИЛОЖЕНИЕ Р. ПРОТОКОЛ TLS

Протокол TLS предназначен для обеспечения криптографическими средствами аутентификации отправителя (клиента) и адресата (сервера), контроля целостности и шифрования данных информационного обмена. Безопасность транспортного уровня обеспечивается с помощью того, что поверх протокола TCP/IP устанавливается зашифрованный канал, внутри которого передаются данные по прикладному протоколу. Фактически данные по прикладному протоколу передаются по TCP/IP, но они зашифрованы. Расшифровать передаваемые данные может только участники, между которыми установлено соединение. Для всех остальных, кто получит передаваемые пакеты, эта информация будет бессмысленной, т.к. они не смогут ее расшифровать.

Иерархия информационного обмена включает в себя сессии, соединения и поток сообщений в соединении. Поток сообщений при большой длине разбивается на фрагменты с пакетной передачей фрагментов. В одной сессии может быть реализовано несколько соединений, произвольно разнесенных по времени. В каждом соединении может быть обработан необходимый поток сообщений.

Сессия характеризуется следующими атрибутами:

- идентификатор сессии (случайное число, 32 байта, задается сервером при открытии сессии);
- метод компрессии;
- сертификат сервера (опционально);
- сертификат клиента (опционально);
- спецификация алгоритмов и параметров защиты (алгоритмы шифрования и MAC, криптографические параметры);
- master secret (используется при генерации ключей шифрования, ключей MAC, векторов инициализации);
- флаг, разрешающий/запрещающий новые соединения в сеансе.

Сертификаты представляются в стандарте X509. v3. Спецификация алгоритмов и параметров защиты может меняться в течение сессии.

Соединение характеризуется следующими атрибутами:

- client_random – случайные 32 байта, задаваемые клиентом;
- server_random– случайные 32 байта, задаваемые сервером;
- client write MAC secret (ключ клиента для вычисления значения ключевой хэш-функции);
- server write MAC secret (ключ сервера для вычисления значения ключевой хэш-функции);
- client write key (ключ, используемый для шифрования данных клиентом и расшифрования их сервером);

- server write key (ключ, используемый для шифрования данных сервером и расшифрования их клиентом);
- client write IV, server write IV (векторы инициализации, используемые клиентом и сервером соответственно);
- порядковый номер соединения (поддерживается независимо для передаваемых и принимаемых сообщений).

Вектор инициализации задается для первого фрагмента сообщения в соединении; для последующих фрагментов вектор инициализации формируется из конечного блока зашифрованного текста предыдущего фрагмента.

Порядковые номера соединений поддерживаются независимо для передаваемых и принимаемых сообщений. При смене сессии, изменении спецификации алгоритмов и параметров защиты нумерация соединений начинается с 0; Диапазон нумерации: 0 ÷ 264-1.

Соединение ассоциируется с одной сессией.

Алгоритм преобразования информации при обмене с использованием протокола TLS включает следующие операции:

1) прием от протокола верхнего уровня потока не интерпретируемых данных в блоках произвольного размера;
2) фрагментация принятых с верхнего уровня данных в структурированные блоки (фрагменты) протокола TLS. Размер фрагмента – не более 214 байт;
3) компрессия фрагментов (опционально);
4) вычисление значения ключевой хэш-функции (MAC) от конкатенации ключа хэш-функции, типа компрессии, длины компрессированного фрагмента, компрессированного фрагмента и заданной константы;
5) конкатенация фрагмента и результата вычисления значения хэш-функции от него (расширенный фрагмент);
6) зашифрование расширенного фрагмента (опционально);
7) добавление открытого заголовка, содержащего тип сообщения (один байт), версию протокола TLS (два байта) и длину компрессированного фрагмента.

При приеме информации применяется обратная последовательность операций.

В протоколе TLS используются следующие типы сообщений:

- Hello message (ClientHello, ServerHello);
- Change cipher specs message (изменение спецификации алгоритмов и параметров защиты);
- Key exchange message (передача ключа обмена ключами шифрования и MAC клиента, сервера);
- Alert message (предупреждение, оповещение о фатальной ошибке);

- Application_data message (передача данных);
- Finished message (сообщение о возможности работы в созданной сессии).

Протокол TLS является двухуровневым и действует над транспортным протоколом. К первому уровню относятся TLS Handshake Protocol, TLS Change Cipher Spec и TLS Alert Protocol. Ко второму уровню относится TLS Record Protocol.

TLS Handshake Protocol обеспечивает инициализацию сессии (соединения) выполнением следующих операций:

1) клиент и сервер договариваются об используемых в сессии алгоритмах и параметрах защиты, обмениваются случайными величинами client_random, server_random, договариваются, будут или нет новые соединения;
2) производится обмен сертификатами для аутентификации клиента и сервера (по заданным опциям);
3) клиент генерирует случайную величину pre_master secret, шифрует ее и передает серверу.
4) клиент и сервер по pre_master secret, client_random и server_random формируют master secret (набор необходимой ключевой информации) сессии.

Обмен данными между условными клиентом и сервером состоит из следующих шагов:

1) Клиент отправляет hello-сообщение, в котором перечислены криптографические данные;
2) Сервер отвечает hello-сообщением, указывая: режим симметричного шифрования, идентификатор сессии - случайную последовательность байтов сертификат сервера, запрос на сертификат клиента;
3) Клиент проверяет сертификат сервера и отправляет случайную последовательность данных для вычисления секретного ключа, сертификат клиента;
4) Сервер проверяет сертификат клиента, после чего определяет состояние распределенного реестра, он должен быть синхронизирован;
5) В случае успеха, выполняется обмен сообщениями, подтверждающими установку TLS соединения;
6) Выполняется проверка ключа сервера и клиента в Whitelist узлов;
7) В случае успеха предыдущих пунктов, начинается обмен данными прикладного уровня;
8) По завершению обмена, соединение разрывается.

В СКЗИ Мастерчейн для установки TLS соединения между узлом СКЗИ Мастерчейн и смежными платформенными компонентами в ролях клиента и сервера используются следующие прикладные объектные идентификаторы (OID):

- 1.2.643.6.57.1.1.1.2: [TLS Client] Узел Мастерчейн;
- 1.2.643.6.57.1.1.1.1: [TLS Server] Узел Мастерчейн;
- 1.2.643.6.57.1.1.1.3: [TLS Server] API узла Мастерчейн.

ПЕРЕЧЕНЬ ТЕРМИНОВ

Термин	Описание
Блок	Набор транзакций и метаданных, который успешно прошёл процедуры валидации и консенсуса.
Ключевой контейнер	Способ хранения закрытых ключей, реализованный в КриптоПро.
Коин	Виртуальная монета, которая используется в качестве цифровых денег с возможностью приобретения товаров и услуг, а также иных расчетов между пользователями.
Компонент КриптоПро	Средство криптографической защиты информации «КриптоПро CSP» версия 5.0, либо ПАКМ «КриптоПро HSM» v. 2.0, при условии наличия действующего сертификата соответствия ФСБ России. Программный компонент СКЗИ Мастерчейн, используемый в качестве средства, реализующего криптографические алгоритмы.
Консенсус	Процесс согласования состояния РР между узлами, с применением процедуры валидации.
Майнинг	Деятельность по созданию нового или нахождению текущего блока.
Нонс	Случайный набор данных, используемый в качестве вводных данных при хешировании блока транзакций.
Одноранговая сеть	Одноранговая, децентрализованная или пиринговая (англ. Peer-to-peer, P2P — равный к равному) сеть — это оверлейная компьютерная сеть, основанная на равноправии участников. Часто в такой сети отсутствуют выделенные серверы, а каждый узел (пир) является как пользователем, так и выполняет функции сервера.
Пир	Узел в P2P сети, с которым установлено соединение для синхронизации данных.
Пользователь	Субъект, подключающиеся к узлу локально или через API для выполнения своих функций.
Процедура валидации	Перечень проверок при добавлении транзакции в блок, включающий: 1. Аутентификацию участника-отправителя (проверку подлинности); 2. Проверку целостности; 3. Проверку корректности транзакции (соответствие логике работы Мастерчейн и соответствие логики работы смарт-контракта, если транзакция обращена к нему); 4. Проверку на противоречивые состояния.

Термин	Описание
СКЗИ Мастерчейн	Скомпилированный исполняемый файл, запускаемый в программном окружении. Все компоненты СКЗИ Мастерчейн реализуются исходным кодом СКЗИ Мастерчейн. После компиляции, все они реализованы в одном бинарном файле.
Распределенный Реестр (РР)	Распределенная база данных с управлением конкурентным доступом на основе многоверсионности с необратимым ходом событий (времени), построенная на основе журнала транзакций с массовой репликацией в одноранговой сети участников.
Расчетные единицы (Токены)	Расчетные единицы, используемые для обмена активами между участниками.
Сайзинг	Подбор оптимальной конфигурации аппаратного обеспечения.
Сертификат участника	Сертификат открытого ключа подписи участника.
Сеть Мастерчейн (Сеть)	Одноранговая сеть, образуемая экземплярами СКЗИ Мастерчейн запущенными в заданном программном окружении.
СКЗИ Мастерчейн	Скомпилированный исполняемый файл, запускаемый в программном окружении. Все компоненты СКЗИ реализуются исходным кодом СКЗИ. После компиляции, все они реализованы в одном бинарном файле.
Смарт-контракт	Объект, представляющий собой экземпляр программного класса, с заданным набором свойств (атрибутов) и операций (методов) над ними, состояние которого хранится в РР. Методы смарт-контракта могут выполнять транзакции в сети Мастерчейн, при этом Составителем транзакции является Участник, вызвавший данный метод.
Сокет	Программный интерфейс для обеспечения обмена данными между процессами.
Составитель транзакции	Участник, создавший транзакцию.
Счёт участника	Сущность в БД Мастерчейн, обладающая следующими атрибутами: – уникальный идентификатор счёта участника; – признак счёта по умолчанию; – код актива; – владелец счёта участника (уникальный идентификатор участника); – баланс.

Термин	Описание
Транзакция	Запрос на изменение логического состояния РР, который содержит в себе все необходимые данные для выполнения, идентификации, аутентификации и авторизации участников – Составителя, Отправителя и Получателя запроса. Данный запрос выполняется с использованием технологии двухфазного исполнения.
Узел	Экземпляр СКЗИ Мастерчейн, подключённый к сети Мастерчейн.
Участник	Субъект имеющий зарегистрированный уникальный индентификатор в сети Мастерчейн

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

Сокращение	Описание
АБС	Автоматизированная банковская система
АФТ	Академия финансовых технологий
БД	База данных
БП	Бизнес процессы
ДДС	Децентрализованная депозитарная система
ДМЗ	Демилитаризованная зона
ОЗУ	Оперативное запоминающее устройство (оперативная память)
ОС	Операционная система
СКЗИ	Средства криптографической защиты информации
СРЕ	Специализированные расчетные единицы
СУБД	Система управления базами данных
ТРЕ	Технологические расчетные единицы
ЦПУ	Центральный процессор
API	Application Programming Interface
CPU	Central Processing Unit
CSP	Cryptography Service Provider
DHT	Distributed Hash Table
EVM	Ethereum Virtual Machine
IPC	Inter Process Communications
JS	Java Script
JSON	Java Script Object Notation
NAT	Network Address Translation
PIN	Personal Identification Number

Сокращение	Описание
POA	Proof of Authority
RAM	Random Access Memory
RLP	Recursive Length Prefix
RPC	Remote Procedure Call
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User DatagramProtocol

Лист регистрации изменений									
Изм.	Номера листов (страниц)				Всего листов (страниц) в докум.	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
	изменённых	заменённых	новых	аннулированных					