Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «**ИНФОРМАТИКА, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И СИСТЕМЫ УПРАВЛЕНИЯ**»

КАФЕДРА «**Программное обеспечение ЭВМ и информационные технологии**»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

«Метод создания уникальных сертификатов, подтверждающих окончание учебного заведения, на основе технологии невзаимозаменяемых токенов»

Студент группы **ИУ7-83Б**	_____	**Е.М. Соколов**
	(Подпись, дата)	(И.О. Фамилия)
Руководитель ВКР	_____	**Т.И. Вишневская**
	(Подпись, дата)	(И.О. Фамилия)
Нормоконтролер	_____	**Д.Ю. Мальцева**
	(Подпись, дата)	(И.О. Фамилия)

***2022** г.*

РЕФЕРАТ

Расчетно-пояснительная записка к выпускной квалификационной работе «Метод создания уникальных сертификатов, подтверждающих окончание учебного заведения, на основе технологии невзаимозаменяемых токенов» содержит 60 страниц, 4 части, 14 рисунков, 5 таблиц, 33 источника.

Ключевые слова: блокчейн-сеть, невзаимозаменяемые токены, консенсус сети, смарт-контракты, документы об образовании.

Объект разработки – метод создания сертификатов, подтверждающих окончание учебного заведения.

Цель работы – разработка и программная реализация метода создания уникальных сертификатов, подтверждающих окончание учебного заведения, с помощью блокчейн-сети, в которой содержаться невзаимозаменяемые токены.

Область применения – учебные заведения и образовательные онлайн-платформы, а также кадровые агентства, проверяющие резюме соискателей.

В первой части работы приведен обзор и сравнение существующих методов решения задачи выдачи сертификатов об образовании, средства разработки блокчейн-сети, а также алгоритмы консенсуса сети.

Во второй части описан процесс разработки метода создания уникальных сертификатов, подтверждающих окончание учебного заведения, с помощью блокчейн-сети и невзаимозаменяемых токенов.

В третьей части определены средства разработки, примененные при программной реализации метода, и приведена структура разработанного ПО.

В четвертой части проведены исследования доступности сертификата пользователям и зависимости времени финализации транзакций от числа узлов сети.

Поставленная цель была достигнута: был разработан и реализован метод создания уникальных сертификатов, подтверждающих окончание учебного заведения. Были рассмотрены преимущества и недостатки разработанного метода и предложены пути дальнейшего развития.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Документы об образовании играют важную роль в жизни каждого человека. Они необходимы для трудоустройства, повышения квалификации и т.п. В связи с этим встал вопрос о сохранности и подлинности данных документов. Документы об окончании государственных учебных заведений подвержены износу с течением времени; дипломы, выдаваемые образовательными онлайн-платформами, можно подделать в редакторе. Чтобы проверить подлинность документов об образовании необходимо сформировать запрос в государственный реестр или в организацию, выдавшую данный документ – на это все необходимо дополнительное время.

Использование блокчейн-сети с невзаимозаменяемыми токенами для подтверждения окончание учебного заведения позволит авторизованным пользователям (учебным организациям) вместе с классическими документами выпускать специальный сертификат в сети, подтверждающий классические документ. Любой участник сети сможет убедиться в подлинности выпущенного сертификата, а с течением времени невозможно будет изменить или подделать данные сертификата, не нарушив состояние блокчейн-сети [1]. Таким образом, использование данной технологии позволяет решить задачи, такие как утрата, подделывание и износ документов.

Целью работы является разработка и программная реализация метода создания невзаимозаменяемых токенов в блокчейн-сети, с помощью которых можно подтвердить окончание учебного заведения. В рамках работы решаются следующие задачи:

— проанализировать существующие способы выдачи сертификатов об окончании учебного заведения;
— разработать блокчейн-сеть, в которой реализован метод выдачи уникальных сертификатов;
— программно реализовать разработанный метод;

— исследовать зависимость времени финализации транзакции от количества участников сети, а также произвести сравнение с аналогами.

1 Аналитическая часть

В этом разделе производится анализ предметной области, рассматриваются существующие решения задачи выдачи сертификатов, подтверждающих окончание учебного заведения. Также в данном разделе выбираются средства разработки блокчейн-сети и алгоритм консенсунса сети, рассмотрены их положительные и отрицательные стороны. Также в этом разделе производится формализация постановки задачи и представлена диаграмма IDEF0.

1.1 Аналоги сертификатов, подтверждающих окончание учебного заведения

Рассмотрим существующие реализации выдачи сертификатов, подтверждающих окончание учебного заведения:

— сертификаты, выдаваемые образовательными онлайн-платформами, такими как Coursera [3];

— документы об окончания вуза РФ.

Сертификаты, выдаваемые образовательными онлайн-платформами, представляют из себя файл, в котором отражена информация о выпускнике и курсе, который он закончил. Такой документ подвержен фальсификациям, например, изменениям файла в редакторе. Таким образом, для проверки подлинности сертификата необходимо подать запрос в организацию, выдавшую его, и дождаться ее ответа.

В документе об окончании вуза РФ содержится информация о полученной специальности, сданных экзаменах и зачетах, итоговых оценках по профильным дисциплинам. Ответственность за сохранность документа лежит на выпускнике. Так, порча либо же утрата данного документа чревато проблемами при дальнейшем трудоустройстве, а восстановление документа происходит при обращении в специализированные органы и влечет за собой бюрократические задержки.

1.2 Средства разработки блокчейн-сети

Задача блокчейна – принимать от пользователей транзакции и однозначным и неоспоримым способом обрабатывать каждую из них. Результаты работы каждой транзакции записываются в общедоступную базу данных (англ. state database) на каждой машине блокчейн-сети. Любой участник может воспроизвести и перепроверить эту базу данных, если у него есть данные о ее первоначальном состоянии и журнал всех транзакций или блоков [2].

Чтобы добиться детерминированного выполнения транзакций, необходимо учесть, какая виртуальная машина будет использоваться в проекте. Также, любая логика, сложнее чем просто перевод токенов с адреса на адрес, требует использования определенного кода, который будет выполняться на заданных виртуальных машинах.

1.2.1 Виртуальная машина

Проанализируем три основных вида виртуальных машин для исполнения кода транзакций:

— специализированная виртуальная машина;

— стандартная виртуальная машина;

— обработка транзакций нативным кодом.

Специализированные вирутальные машина, как правило, ограничены и заточены строго под смарт-контракты своей платформы. Они порождают более предсказуемые результаты, являются наиболее безопасными и точно учитывают ресурсы, потраченные на обработку транзакций [4]. Примерами таких машин являются EVM (Ethereum) [5] и TVM (TON) [6].

Стандартная виртуальная машина реализуется с помощью WebAssembly. WebAssembly (WASM) — это веб-стандарт, используемый для создания кода, исполняемого на стороне клиента и более производительного, чем JavaScript [7]. Теоретически, смарт-контракты под WASM можно писать на любом языке, но для блокчейнов лучше подходят низкоуровневые языки, иначе получив-

шийся код будет плохо оптимизирован [8]. WASM также ведет учет потраченных на исполнение ресурсов. Имея большую функциональность контрактов, WebAssembly менее безопасна, чем специализированная виртуальная машина. Данный подход применяется в таких проектах, как EOS [9] и Parity Substrate [10], которые предлагает инфраструктуру для разработки блокчейн-сети, причем предоставляя разработчику выбор между технологической свободой и легкостью разработки.

Схему обработка транзакций нативным кодом можно представить в виде одного большого смарт-контракта, который обрабатывает все виды транзакций. Код проверяют валидаторы и голосуют за применение изменений, после чего новая логика начинает работать. При этом разработчики избавлены от большого числа ограничений и могут создавать код из набора готовых модулей. Пример проектов, использующих схему обработка транзакций нативным кодом: Cosmos [11].

1.2.2 Способ обновления кода контрактов

В современных блокчейнах задачи добавления и изменения функциональности системы решаются следующими способами:

— пользовательские смарт-контракты;

— нативный код, контролируемый валидаторами.

В схеме с использованием пользовательских смарт-контракты все участники могут создать один или несколько, соединенных в сложную систему, смарт-контрактов. Контракты можно размещать и обновлять без взаимодействия с валидаторами сети. Эта схема – наиболее гибкая [12]. Она позволяет строить системы контрактов произвольной сложности, но требует более сложной логики работы машины, поскольку код контрактов является недоверенным и может содержать все что угодно. Поэтому нужно, чтобы машина исполняла такой код крайне осторожно, ограничивая его по времени и запрашиваемым данным, и не позволяла бы влиять на консенсус сети. Примеры проектов, использующих пользовательские смарт-контракты: Ethereum [5], EOS [9], TON [6], Parity

Substrate (с модулем пользовательских смарт-контрактов WASM или EVM) [10].

Схему использования нативного кода можно представить в виде одного большого смарт-контракта, который обрабатывает все виды транзакций. Код проверяют валидаторы и голосуют за применение изменений, после чего новая логика начинает работать. При этом разработчики избавлены от большого числа ограничений и могут создавать код из набора готовых модулей. Данная схема делает более сложным изменение кода, который обрабатывает транзакции, но позволяет не делать дополнительные проверки безопасности. Валидаторы в данной схеме обязаны проверять изменения и не пропускать уязвимый код, иначе рискуют получить неработоспособный блокчейн. Однако разработчики получают в распоряжение гораздо больше возможностей и ресурсов, чем в схеме с пользовательскими контрактами. Эта схема особенно актуальна для «парачейнов» (небольших дочерних цепочек с отдельным функционалом, являющимися частями большой системы из множества блокчейнов) [13]. Данная схема нашла свое применение в следующих проектах: Application в Cosmos [11], runtime в Parity Substrate [10].

1.3 Алгоритмы консенсуса

В блокчейн-сетях задача о достижении консенсуса между недоверенными узлами сети есть трансформация задачи византийских генералах. В данной задаче группа генералов, которая командует византийской армией, окружает город. Некоторые генералы предпочитают напасть на город, другие – отступить. Важно отметить, что атака провалится, если только часть генералов решить атаковать. Таким образом, им необходимо достичь некоторого соглашения в своих действиях, что является проблематичным в распределенной среде. Аналогичная задача формулируется и для сетей блокчейн, поскольку в блокчейне нет центрального, главного узла, который гарантировал бы, что все участники владеют одной и той же информацией и находятся в консенсусе между собой.

1.3.1 Алгоритм PoW

Алгоритм Pow (англ. Proof of Work – «подтверждение работой») является стратегией консенсуса, используемой в сети Bitcoin [14]. В распределенной системе кто-то должен быть выбран для записи транзакций. Самый простой способ – случайный выбор. Однако такая стратегия выбора уязвима перед атаками. Таким образом, если участник хочет опубликовать блок транзакций, необходимо проделать большую работу, чтобы доказать, что участник скорее не пытается атаковать сеть. В общем случае, работа подразумевает под собой вычислительные затраты. В PoW каждый узел сети вычисляет хэш заголовка блока. Заголовок блока содержит в себе одноразовое число (англ. nonce), изменяя который, изменяется хэш заголовка. Консенсус требует, чтобы вычисленный хэш был меньше или равен некоторому заданному числу. Как только один из узлов получает необходимый хэш, он отправляет в сеть блок, чтобы остальные участники могли взаимно подтвердить корректность вычисленного хэша. Если блок успешно проходит валидацию, его добавляют в блокчейн [15].

В децентрализованных сетях валидные блоки могут быть сгенерированы одновременно – разные узлы подобрали подходящий nonce примерно в одно время. В результате могут возникнуть две разные ветки блокчейна. В то же время, маловероятно, что в двух конкурирующих ветках появятся новые блоки одновременно. В протоколе PoW более длинная цепочка признается истинной, а остальные отбрасываются. На рисунке 1 изображены две конкурирующие ветки блокчейна: ветви появились из-за одновременного появления блоков B4 и U4. Участники продолжают создавать новые блоки (подбирать nonce нового блока), пока не появится более длинная ветвь. Блоки B4 и B5 формируют более длинную цепочку и сеть приходит к консенсусу – все участники переключаются на эту ветвь и отбрасывают блок U4.

Генерация нового блока в PoW-консенсусе требует больших энергетических затрат, которые уходят впустую. Чтобы как-то компенсировать эти затра-

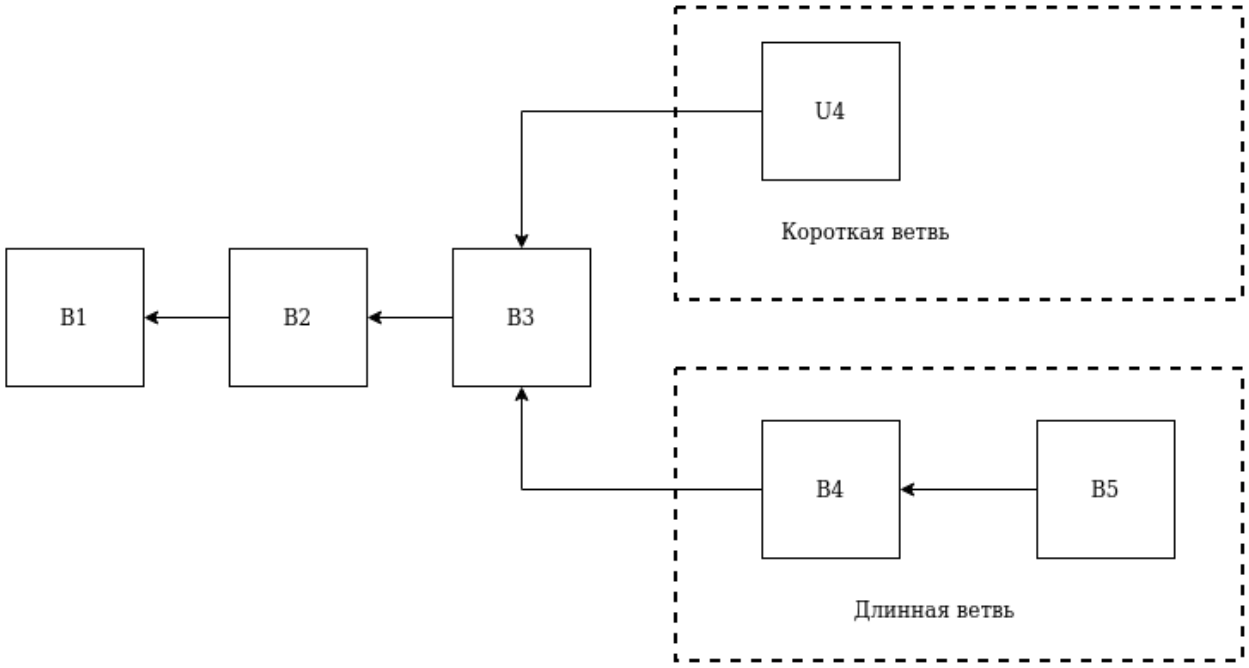

Рисунок 1 – Конкурирующие ветви блокчейна

ты некоторые проекты, как например [16], осуществляют поиск простых чисел, что может пригодится в математических исследованиях.

1.3.2 Алгоритм PoS

Протокол PoS (англ. Proof of Stake – «подтверждение долей») является энергосберегающим аналогом PoW. Участники должны подтвердить факт владения некоторого количества валюты. Считается, что участники с бóльшим капиталом в сети будут меньше заинтересованы в атаке. Выбор валидаторов, основанный на их капитале в сети, не является абсолютно честным, поскольку единственный богатейший участник сможет контролировать консенсус всей сети. Поэтому решение о принятии нового блока принимается на основании комбинации размера доли и некоторого значения. Так, в проекте Blackcoin [17] используется генератор псевдослучайных чисел, чтобы выбрать следующий блок, а в проекте Peercoin [18] отдается предпочтение возрасту денежных средств участника – обладатели более старых сбережений имеют больше шансов валидировать следующий блок.

В отличие от PoW PoS гораздо более эффективный и менее энергозатрат-

ный. Однако факт того, что энергетическая цена валидации одного блока условно равна нулю, подвергает опасности сеть в начале ее существования. Поэтому многие проекты стартуют, используя PoW, а затем переходят к PoS. Например, Ethereum [5] планирует переход от Ethash (разновидность PoW) к Casper (разновидность PoS).

1.3.3 Алгоритм DPoS

Основное отличие DPoS (англ. Delegated PoS – «делегированный PoS») от PoS заключается в том, что PoS является непосредственно демократичным, а DPoS – представительно демократичным [19]. Держатели долей выбирают своих представителей, которые будут непосредственно создавать и валидировать новые блоки. Таким образом, можно существенно уменьшить количество валидаторов в сети, что влечет за собой быстрое подтверждение транзакций. Делегаты могут изменять параметры сети: размер блока, время до принятия нового блока. Однако участники сети могут не беспокоится в бесчестности делегатов, поскольку последние могут быть легко смещены на очередном голосовании.

1.3.4 Алгоритм PBFT

Алгоритм PBFT (англ. Practical Byzantine Fault Tolerance – «Практическая византийская отказоустойчивость») – алгоритм репликации данных, разработанный для решения «византийских» ошибок [20]. Каждый новый блок в сети определяется в очередном раунде. В каждом раунде, опираясь, на определенные правила, выбирается «основной» узел, который ответственен за последовательность транзакций в блоке. Весь процесс валидации может быть разбит на три этапа: предподготовка, подготовка и фиксирование. Чтобы приступить к следующему этапу узлу необходимо получить голоса от двух третей всех остальных участников, что свидетельствует о следующем:

— PBFT устойчив к вредоносным узлам, если их количество не превышает треть участников сети;
— каждый узел системы должен быть известен остальным членам сети.

Аналогично PoS и DPoS, на основании PBFT выработан алгоритм dPBFT,

в котором путем голосования отбираются узлы, которые могут принимать участия в раундах валидации блоков [21].

1.3.5 Алгоритм Tendermint

В протоколе Tendermint новый блок, аналогично PBFT, валидируется в очередном раунде [22]. Для начала выбирается узел-кандидат, чей блок планируется валидировать. Сам раунд валидации можно разделить на три этапа:

1) этап предголосования: участники решают, перейти ли к следующему этапу валидации блока;
2) этап предфиксации: если на предыдущем этапе «за» проголосовало более двух третей участников, то в сеть транслируется информация об этом; если это подтверждает более двух третей узлов, то переходят к следующему этапу;
3) этап фиксации: узел фиксирует факт валидации блока и транслирует эту информацию в сеть. Если он набирает более двух третей подтверждений, то сеть принимает этот блок.

В отличие от PBFT, участники сети должны «заморозить» некоторую сумму на своих счетах, чтобы стать участником валидации.

1.4 Сравнительный анализ рассмотренных средств и алгоритмов

1.4.1 Средства разработки блокчейн-сети

Использование специализированной виртуальной машины (ВМ) позволяет порождать самые безопасные и ожидаемые результаты, но в то же время является самый ограниченным по возможным средствам разработки выбором. Нативный код, напротив, требует дополнительной проверки безопасности исполняемого кода, но зато почти неограничен в средствах разработки. Стандартна ВМ предоставляет компромисс между данными подходами. Отметим, что использование специализированной и стандартизированной ВМ позволяет вести учет затраченных ресурсов (чтобы, например, посчитать размер комиссии с валидации блока). В таблице 1 приведено сравнение виртуальных машин.

Использование смарт-контрактов – самая гибкая схема, поскольку позво-

Таблица 1 – Сравнение вирутальных машин

Характеристика	**Спец. ВМ**	**Стандарт. ВМ**	**Нативный код**
Безопасность	+	+−	−
Неограниченность средств	−	+−	+
Учет затраченных ресурсов	+	+	−

ляет проектировать произвольной сложности системы контрактов, однако требует более сложного устройства ВМ: требует от ВМ учет вычислительных ресурсов, проверку безопасности. Использование же нативного кода несколько усложняет проектирование систем контрактов, но зато предоставляют почти неограниченный функционал разработчикам. За проверку безопасности исполняемого нативного кода отвечают не ВМ, а валидаторы сети. В таблице 2 приведено сравнение способов обновления кода контрактов.

Таблица 2 – Сравнение способов обновления кода контрактов

Характеристика	**Смарт-контракты**	**Нативный код**
Безопасность	−	−
Неограниченность средств	−	+
Гибкость	+	−

1.4.2 Алгоритмы консенсуса

Рассмотренные алгоритмы консенсуса имеет как свои преимущества, так и свои недостатки. В таблице 3 приведено сравнение алгоритмов консенсуса по следующим характеристикам:

— идентификация узлов: PBFT и Tendemint требуют идентификации каждого участника сети, чтобы выбрать очередного валидатора; в PoW, PoS и DPoS узлы могут свободно подключаться к сети;

— энергоэффективность: в PoW для валидации блока необходимо вы-

полнить внушительный объем вычислений, который уйдет впустую; PoS и DPoS также подбирают хэш блока, но поскольку они спроектированы так, чтобы свести поиск к минимуму, они экономят значительное число ресурсов; в Tendermint и PBFT отсутствует подбор хэша, что существенно экономит энергию;

— допустимое число атакующих: в общем случае 51% хэшируемой мощности сети является порогом для захвата всей сети, однако стратегия PoW позволяет снизить этот порог до 25%; PBFT и Tendermint спроектированы выдерживать до трети атакующих от всего числа узлов;

Таблица 3 – Сравнение алгоритмов консенсуса

Характеристика	**PoW**	**PoS**	**DPoS**	**PBFT**	**Tendermint**
Идентификация узлов	нет	нет	нет	да	да
Энергоэффективность	нет	частичная	частичная	да	да
Допустимое число атакующих	<25% вычислительной мощности	<51% доли	<51% валидаторов	<33.3% византийских узлов	<33.3% византийских голосующих

PBFT и Tendermint – протоколы, основанные на разрешениях. Идентификация узла сети должна доступна каждому участнику сети, таким образом, эти протоколы находят свое применение в корпоративных и приватных сетях. PoW, PoS и DPoS подходят для публичных сетей.

1.5 Невзаимозаменяемые токены

Невзаимозаменяемые токены (англ. Non-Fungible Tokens, NFTs) – это тип криптовалюты, производной от Ethereum смарт-контрактов [23]. Если в классической криптовалюте денежная единица является эквивалента другой единице и неотличима от нее, то NFT, напротив, является уникальным и не может быть обменен на себе подобный (буквально, невзаимозаменяемый) [24]. Данное свойство NFT позволяет его использовать для идентификации кого-то или чего-то в сети. Так, применение NFT позволяет доказать факт владения некоторым цифровым ресурсом, например, изображением, аудио- или видеозаписи,

билетов на мероприятие и т.п.

Технология невзаимозаменямых токенов реализуется в блокчейн-сетях, которые гарантируют что информация о выдаче токена останется в истории сети, а проверка факта владения будет доступна любому частнику. Логика токена (его невзаимозаменяемость, прочие детали) прописывается в его смарт-контракте.

1.6 Формализация предлагаемого метода

Основываясь на проведенном анализе предметной области, задача по реализации метода создания уникальных сертификатов, подтверждающих окончание учебного заведения, с помощью технологии невзаимозаменяемых токенов может быть сформулирована следующим образом: необходимо спроектировать блокчейн-сеть на основе протокола консенсуса PBFT с использованием специализированной ВМ и смарт-контрактами для выдачи невзаимозаменяемых токенов. Входными данными являются документы об окончании учебного заведения, которые необходимо заверить невзаимозаменяемым токеном, а выходными данными – блокчейн-сеть, содержащая невзаимозаменяемый токен. На основе сформулированной задачи была построена диаграмма IDEF0, представленная на рисунке 2.

Выводы

В аналитическом разделе был проведен анализ предметной области, включающий в себя обзор существующих методов выдачи сертификатов, подтверждающих окончание учебного заведения, а также средства разработки блокчейн-сети и алгоритм консенсунса сети. В ходе анализа была сформулирована задача разработки метода, проанализированы существующие методы решения поставленной задачи и предложен вариант её решения.

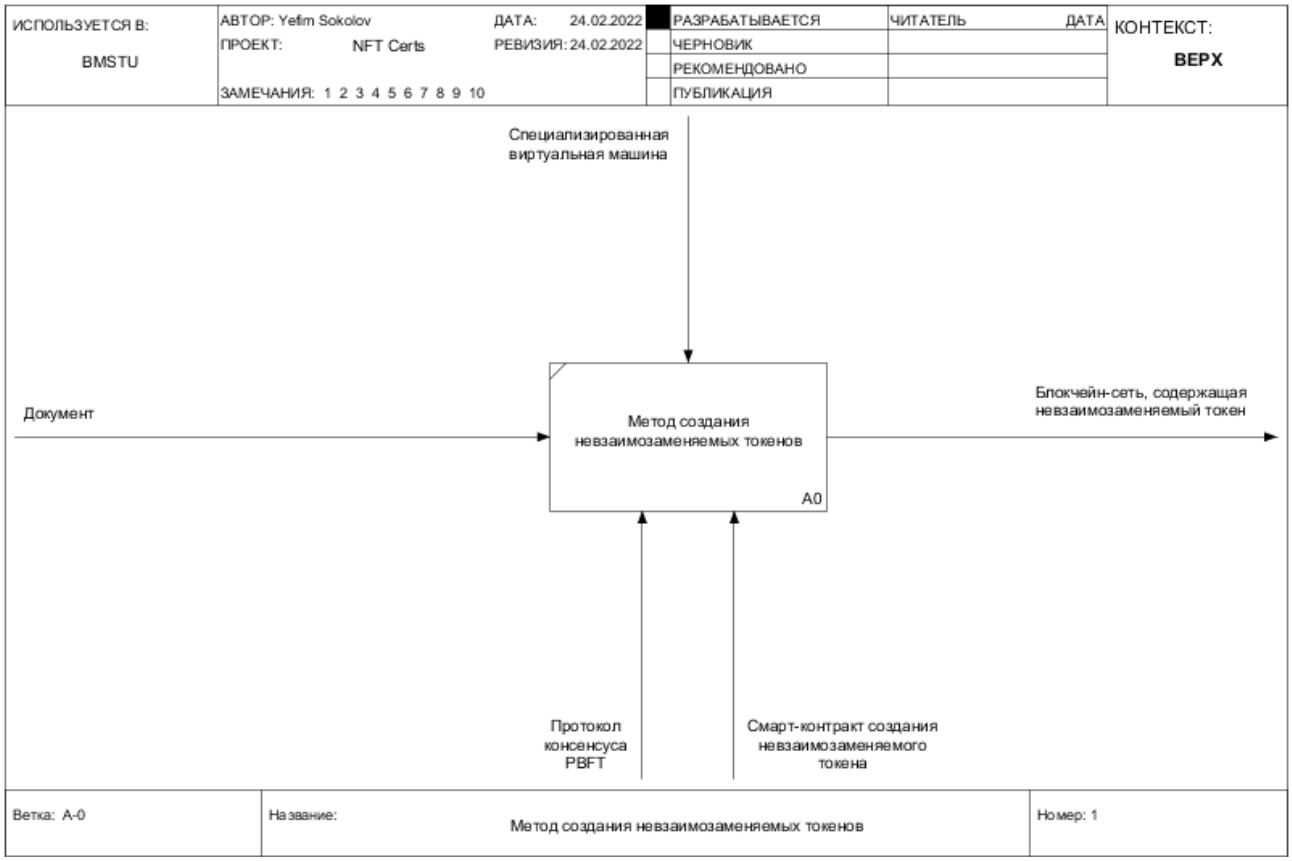

Рисунок 2 – Концептуальная модель системы в нотации IDEF0

2 Конструкторская часть

В данном разделе рассматривается детализированная концептуальная модель системы. Описываются этапы создания невзаимозаменяемых токенов в блокчейн-сети, детали достижения консенсуса в сети, и алгоритм проверки подлинности сертификатов. Также приведены основные сценарии функционирования системы.

2.1 Формализация задачи

2.1.1 Требования к разрабатываемую методу

Для разрабатываемого метода создания уникальных сертификатов, подтверждающих окончание учебного заведения, предложен следующий список требований:

— метод не должен допускать передачу сертификата другому пользователю;

— только ограниченный пул участников может создавать сертификаты;

— метод должен предоставлять возможность проверить подлинность сертификата;

— во всех вычислительных узлах в один момент времени данные о выданных сертификатах не противоречат друг другу.

2.1.2 Детализированная концептуальная модель системы

Построенная IDEF0 диаграмма 1 уровня с декомпозицией решения исходной задачи, формализованной в разделе 1, представлена на рисунке 3. На данной схеме формализованная задача из раздела 1 была декомпозирована на подзадачи A1-3. В следующих подразделах даны описания алгоритмов каждой декомпозированной задачи.

2.2 Создание уникального сертификата

Процесс создания уникального сертификата сводится к тому что, авторизованный на выдачу сертификатов участник (например, вуз) выполняет на специализированной виртуальной машине код смарт-контракта создания невза-

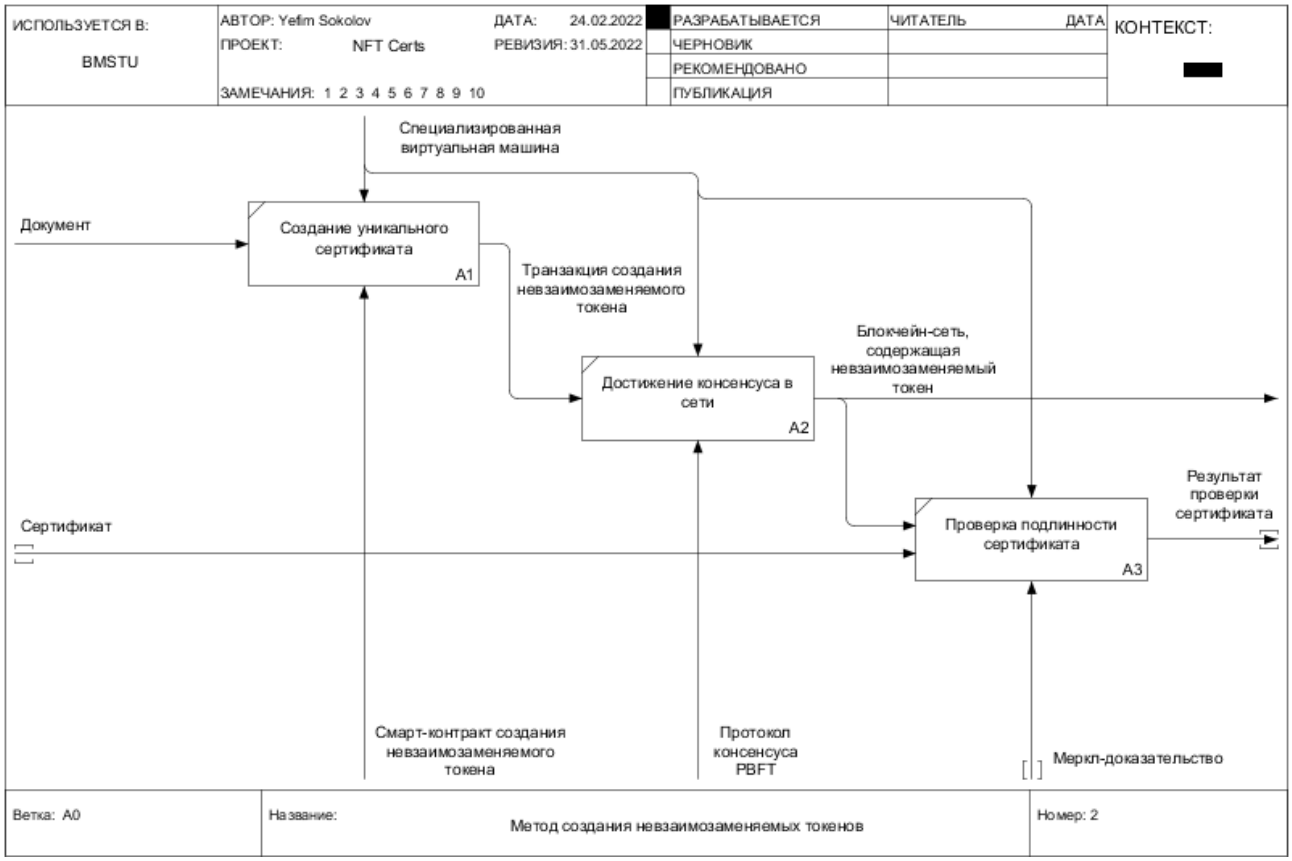

Рисунок 3 – Детализированная концептуальная модель системы в нотации IDEF0

имозаменяемого токена. Токен содержит в себе следующую информацию:

— идентификатор сертификата (токена);

— метаданные сертификата.

Идентификатором токена может быть целочисленное беззнаковое число, которое будет инкрементироваться для очередного сертификата, начиная с нуля или единицы. Под метаданными токена подразумевается полезная нагрузка, которая содержится в сертификате: ссылка на документ об окончании учебного заведения, наименование учебной организации и т.п. Заполнение метаданных токена - ответственность того, кто создает сертификат.

Факт выдачи сертификата заключается в том, что автор токена передает реципиенту идентификатор созданного сертификата, тем самым закрепляя факт владения сертификатом пользователем в сети.

Хранение в метаданных токена ссылки на документ, а не самого файла, и

передача только идентификатора токена обусловлены тем, что блокчейн сеть - это одноранговая сеть, что подразумевает многократную репликацию данных в узлах сети. Таким образом, мы добиваемся снижения затрат на память и уменьшаем нагрузку на сеть.

На рисунке 4 представлена схема алгоритма создания уникального сертификата.

2.3 Достижение консенсуса в сети

Задачу достижения консенсуса в распределенной сети разобьем на две подзадачи:

— производство блоков;

— финализация цепочек блоков (принятие решение о том, какая из ветвей блокчейна валидна).

2.3.1 Производство блоков

На основании проведенного в разделе 1 анализа, был сделан выбор в сторону алгоритма PBFT [20]. Поскольку в PBFT новый блок сети создается в очередном раунде, разделим реальное время на дискретные слоты s фиксированной длиной t (индекс слота можно считать, как $\frac{t_{UNIX}}{t}$). В каждый временной слот, только один участник (автор) может создать один и только один новый блок, автор блока выбирается как $s \bmod n$, где n – число авторизованных на создание блоков узлов.

2.3.2 Финализация цепочек блоков

Задача о финализации цепочек блоков заключается в детерминистическом принятии решения о том, какая из ветвей блокчейна является канонической, какие блоки являются подлинными, а не «византийскими». В качестве протокола финализации был выбран протокол GRANDPA [25].

Валидаторы протокола GRANDPA не создают блоки, а только голосуют за их цепочки. Таким образом, этот процесс может и должен происходить параллельно процессу производства блоков, то есть позволяет абстрагироваться от создания блоков, а также в будущем подменять алгоритм производства блоков,

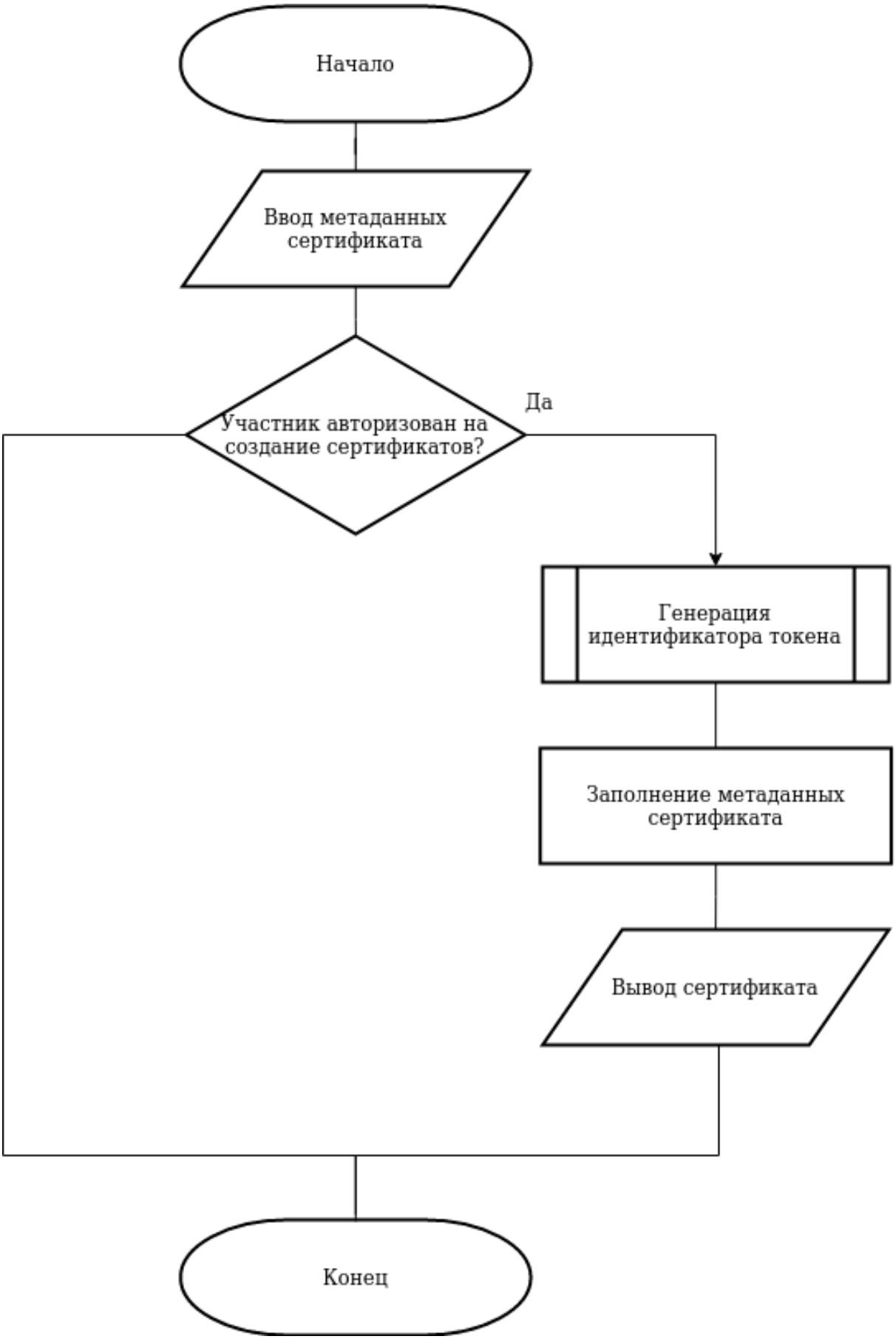

Рисунок 4 – Алгоритм создания уникального сертификата

не изменяя протокол финализации.

Как и алгоритм PBFT, GRANDPA требует, чтобы «честных» валидаторов было как минимум две трети от их общего числа. Важным преимуществом данного протокола является его возможность за один раунд голосования финализировать более одного блока.

Предполагается, что распространение информации о новом блоком в сети не превышает длительность временного слота t. Если за время t информация о новом блоке системы не дошла до всех участников сети, то возникают конкурирующие ветви блокчейн-сети, описанные в разделе 1. Чтобы разрешить конфликт ветвей, принято решение придерживаться правила более длинной ветви (англ. Longest chain rule) – более длинная цепочка блоков считается валидной [25]. На рисунке 5 приведен пример конфликта ветвей: черным отмечен уже финализированный блок, розовым – «лучшая» (самая длинная) цепочка, серым – блоки прочих веток.

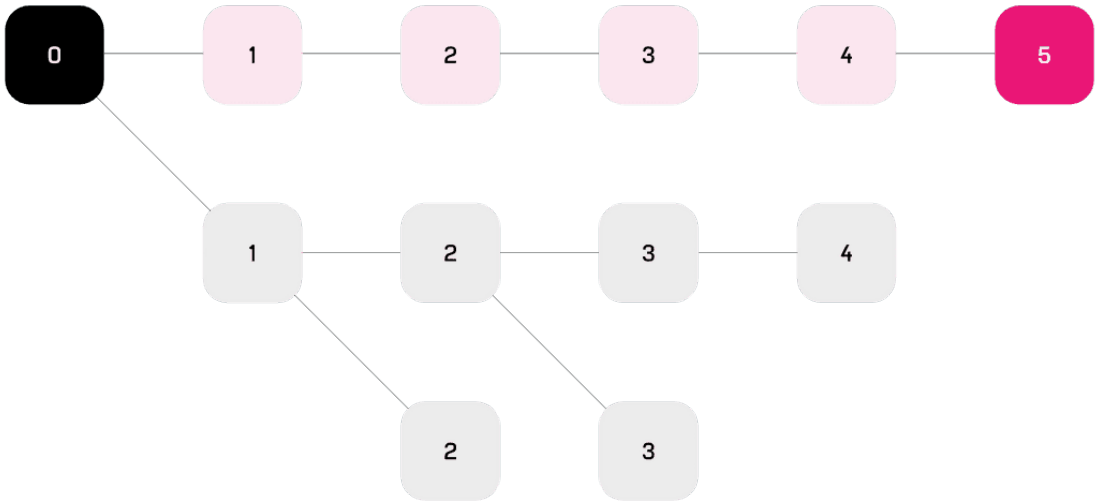

Рисунок 5 – Разрешение конфликта ветвей

Для обратной ситуации, когда в выделенный временной слот не было произведено блока, сеть может зависнуть, поскольку порядок авторов блоков строго регламентирован: автор a будет ждать завершения автора $a-1$. Для решения этой проблемы, введем некоторый предельный период времени T. Если автор

a не дождется блока от автора $a - 1$ за это время, то он в праве произвести собственный блок и цикл производства блоков возобновится.

Для своевременного достижения финализации сети, необходимо, чтобы узлы продолжали создавать блоки, даже если отсутствуют транзакции. Чтобы блокчейн-сеть не «засорялась» пустыми блоками, узлы могут отправлять «пустое сообщение» – подписанный хэш текущего блока, тем самым не производя новый блок, но сохраняя консенсус в сети.

2.4 Проверка подлинности сертификата

Решение задачи проверки подлинности сертификата заключается в доказательстве существования блока в блокчейн-сети, в котором был создан проверяемый сертификат. Чтобы узел сети не скачивал все блоки сети, чтобы проверить факт вхождения, было принято решение воспользоваться деревом Меркла.

Дерево Меркла - это сбалансированное бинарное дерево, в листьях которого содержатся хэши некоторых данных, причем в узлах вычисляется хэш от конкатенации хэшей узлов-потомков. Таким образом в корне дерева Меркл содержится хэш, в котором «учтены» все остальные хеши дерева. Такой корневой хэш, к тому же, учитывает порядок данных в листях дерева [26]. Использование дерева Меркла позволяет хранить в узлах только корень дерева, который будет отражать всю историю транзакций в блокчейн-сети. На рисунке 6 изображена иллюстрация дерева Меркла.

Для выполнения проверки вхождения в дерево Меркла необходимо предоставить «Меркл-докательство» (англ. Merkle proof), которое является массивом хешей, размером $\log_2 N$, где N – число листьев. Далее необходимо «восстановить» дерево Меркла: вычислить хэш от проверяемых данных, конкатенировать вычисленный хэш и хэш листа-брата, взятый из Меркл-доказательства, взять от результата конкатенации хэш и так далее вплоть до вычисления корневого хеша. Подтверждением факта вхождения в дерево Меркла является равенство «истинного» и вычисленного хэша в корне дерева.

На рисунке 7 представлена схема алгоритма доказательства включения в

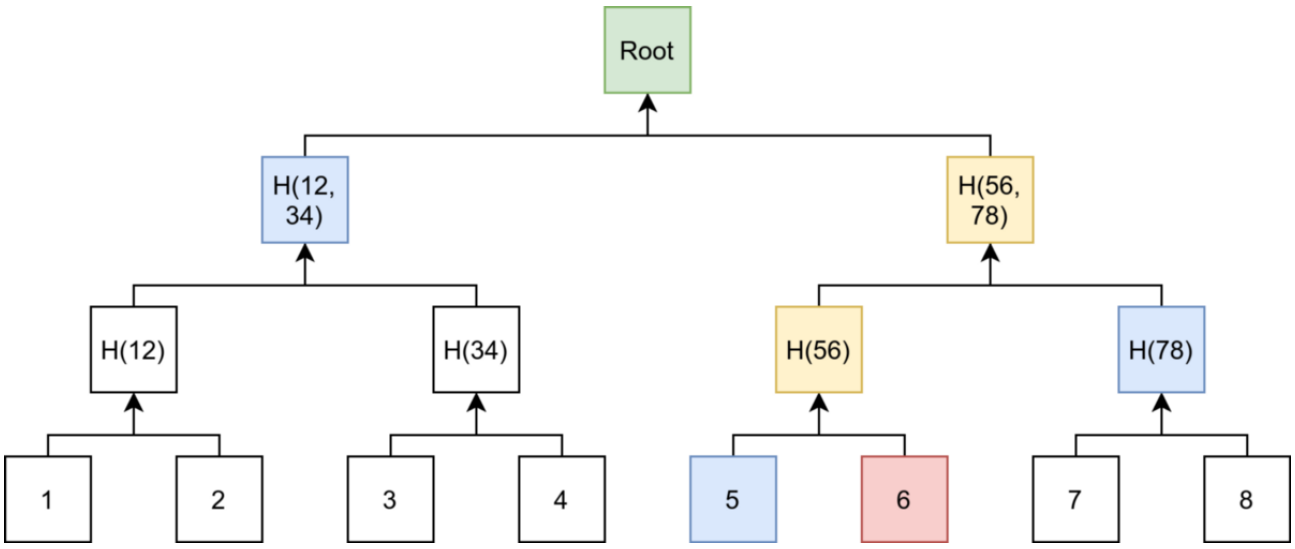

Рисунок 6 – Дерево Меркла

дерево Меркла.

2.5 Сценарии функционирования системы

Определим сценарии функционирования системы для наиболее часто используемых сценариев.

Сценарий для создания сертификата будет выглядеть следующим образом:

— авторизованный на создание сертификатов участник сети (учебное заведение) производит невзаимозаменяемый токен в сети;

— прочие авторизованные участники согласуют либо отвергают созданный сертификат;

— если сертификат «одобрен» большинством, то он выдается пользователю (выпускнику учебного заведения).

Сценарий для проверки подлинности выданного сертификата будет выглядеть следующим образом:

— выпускник предоставляет идентификатор своего сертификата;

— участник сети получает подтверждение подлинности или фальшивости данного сертификата.

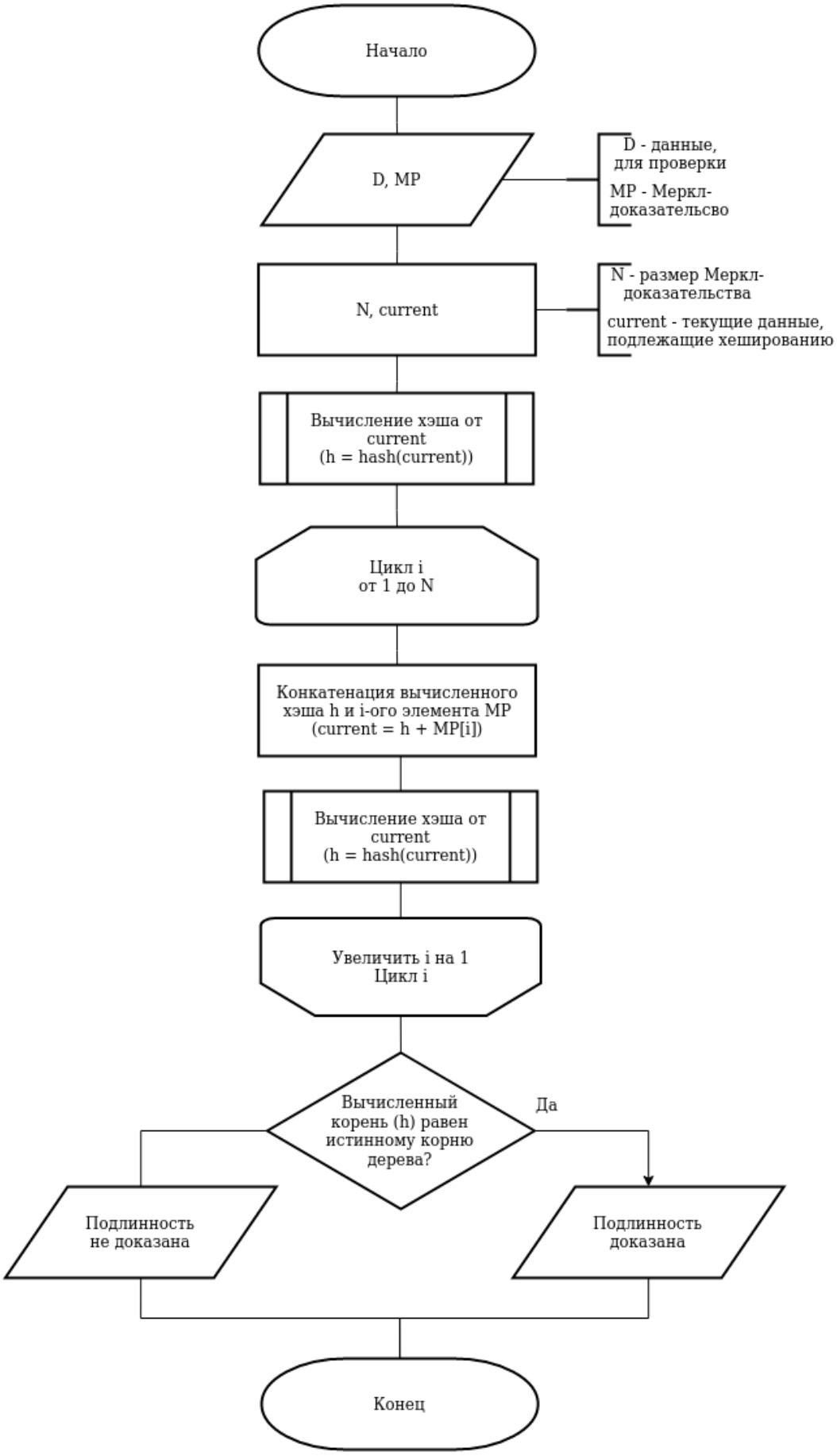

Рисунок 7 – Алгоритм доказательства включения в дерево Меркла

Выводы

Были представлены требования к разрабатываемому методу и детализированная концептуальная модель системы в нотации IDEF0. Приведены описание алгоритмов создания невзаимозаменяемых токенов, достижения консенсуса в сети и проверки подлинности сертификата. Приведены схемы разработанных алгоритмов.

3 Технологическая часть

Данный раздел содержит обоснование выбора средств программной реализации разработанного метода и описание программной реализации разрабатываемого метода. Приводится схема взаимодействия модулей разрабатываемого программного обеспечения. Описываются входные и выходные данные. Приводится методика тестирования модулей системы.

3.1 Выбор средств программной реализации

Для программной реализации поставленной необходимо прежде всего выбрать языки программирования как узлов, так и смарт-контрактов сети блокчейн. Также следует выбрать среду разработки проекта и библиотеку для написания узлов сети.

3.1.1 Выбор языка программирования

В качестве языка программирования узлов сети был выбран язык Rust по следующим причинам:

— в Rust отсутствует «сборщик мусора», что означает невозможность не детерминистических инцидентов (вызванных непосредственно языком) в ходе исполнения программы [27];
— данный язык программирования содержит большое количество библиотек, упрощающих разработку узлов сети блокчейн;
— имеется достаточный опыт программирования на этом языке, что сократит время написания программы.

В качестве языка программирования смарт-контрактов был выбран язык Solidity, поскольку:

— он был спроектирован и создан как язык для написания смарт-контрактов;
— данный язык исполняется на специализированной виртуальной машине EVM [5], абстрагируясь от реализации узлов сети [28].

3.1.2 Выбор среды разработки

В качестве среды разработки была выбрана редактор кода «VS Code» [29]. Этот выбор обусловлен следующими причинами:

— данный редактор кода свободно распространяется;

— он содержит множество доступных плагинов платформы, которые существенно облегчают и ускоряют процесс написание кода.

3.1.3 Выбор библиотеки для написания узлов сети

Для создания узлов блокчейн-сети был выбран фреймворк Substrate, поскольку он написан на языке Rust и позволяет создавать гибкие и настраиваемые блокчейны [10]. К тому же, данный фреймворк поддерживается активным сообществом разработчиков, что упрощает процесс разработки и изучения его документации.

3.2 Структура разработанного ПО

Блокчейн представляет из себя одноранговую сеть, то есть в ней отсутствуют «центральные» узлы. На рисунке 8 приведена схема полносвязной одноранговой сети, состоящей из 5 участников.

Чтобы выполнить поставленную задачу, необходимо реализовать узел сети блокчейн. Также следует написать смарт-контракт создания невзаимозаменяемых токенов в сети, который загружается в сеть. Для взаимодействия с сетью, требуется приложение-клиент.

Разработанная система состоит из следующих модулей:

— узел сети;

— смарт-контракты;

— браузерное клиентское приложение.

На каждом узле сети установлена специализированная виртуальная машина (в частности, EVM), которая обрабатывает транзакции пользователей: создание сертификата в сети и т.п. Помимо виртуальной машины узел также хранит актуальное состояние сети. Для взаимодействия с сетью необходимо,

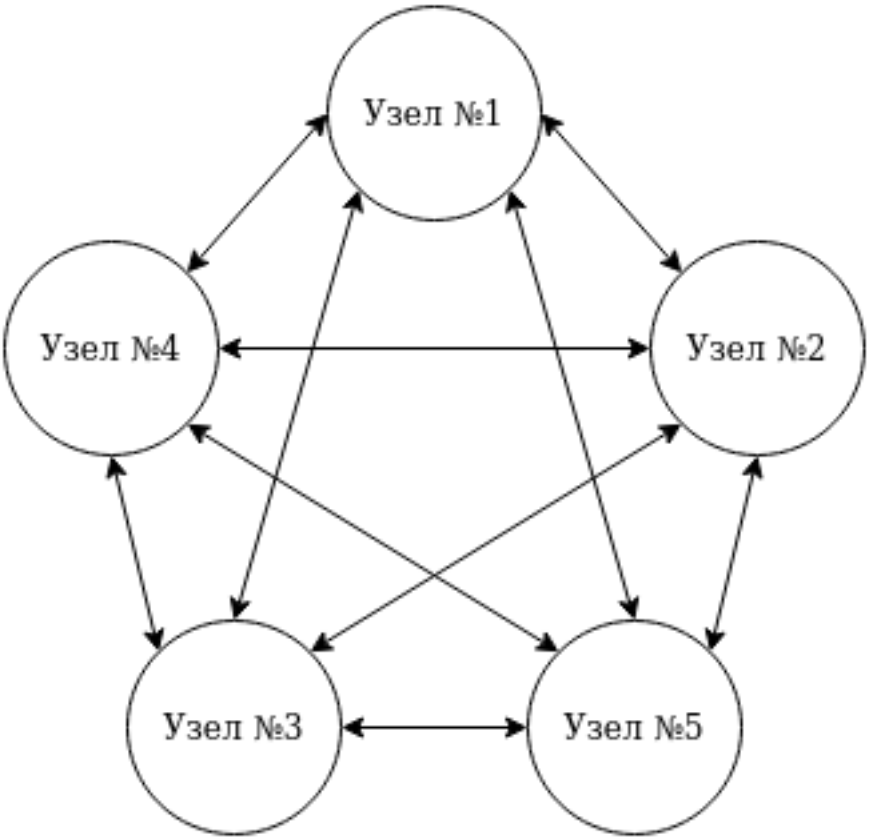

Рисунок 8 – Одноранговая сеть

чтобы узел имел открытый порт для внешнего подключения; стандартным протоколом «общения» является JSON-RPC [31]. На рисунке 9 приведена схема описанного взаимодействия.

3.2.1 Узел сети

Как было упомянуто выше, для реализации узла сети был использован фреймворк Substrate. Для имплементации протокола консенсуса и EVM, необходимо настроить т.н. «среду исполнения» Substrate-узла: требуется сконфигурировать «FRAME-паллеты» Substrate, в частности, pallet_aura и pallet_grandpa для имплементации протокола консенсуса, а также pallet_evm и pallet_ethereum для поддержки EVM и смарт-контрактов в сети. В приложении А на листинге 1 приведен код настройки узла сети.

3.2.2 Смарт-контракты

Каждое учебное заведение владеет собственной копией смарт-контракта, который производит невзаимозаменяемые токены. Чтобы не плодить бескон-

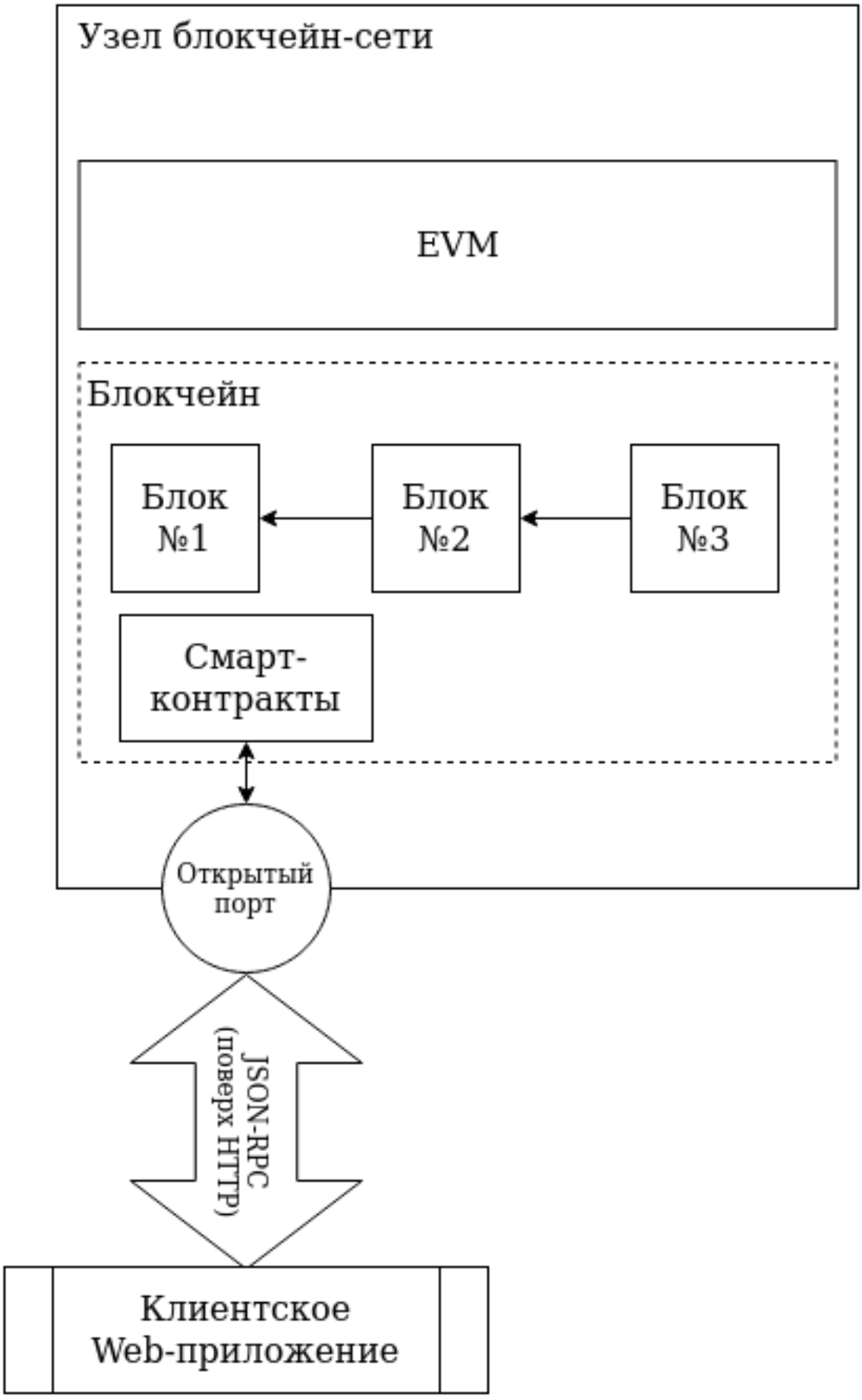

Рисунок 9 – Схема взаимодействия узла сети и клиентского приложения

трольно копии смарт-контрактов, был реализован контракт-фабрика, который производит однотипные копии контрактов. Контракт-фабрика хранит адрес шаблонного контракта, который имплементируется для каждого учебного заведения. В приложении А на листинге 2 представлен код контракта-фабрики.

Смарт-контракт учебного заведения отвечает за создание и чтение сертификатов, созданных заведением, которое им владеет. Данный смарт-контракт имеет следующие поля, указанные в таблице 4.

Таблица 4 – Поля смарт-контракта учебного заведения

Название и тип поля	Описание поля
Название учебного заведения: string storage	Удобное для восприятия пользователя название учебного заведения (не более 255 символов)
Идентификатор токенов: Counter	Счетчик-идентификатор созданных токенов (начиная с 1)
URI токенов: отображение uint256 на string	Отображение идентификатор токенов на ссылки подтверждаемых документов

Каждый созданный невзаимозаменяемый токен содержит в своих метаданных адрес владельца (выпускника), название учебной организации, которая выдала этот сертификат, а также ссылку на подтверждаемый документ. Хранение ссылки на документ вместо самого файла обусловлена дороговизной хранения данных в блокчейне: новым участникам сети придется скачивать больше данных, чтобы иметь актуальную версию блокчейна – повышается нагрузка на сеть и требуется больше свободного места на машине [2].

Поскольку выпуск сертификатов должен быть доступен только владельцу контракта (учебному заведению), необходимо сконфигурировать доступ к методам контракта, путем расширения стандартных контрактов OpenZeppelin, таких как Ownable и AccessControl [32].

В приложении А на листинге 3 приведен код смарт-контракта учебного заведения.

3.2.3 Клиентское приложение

С помощью библиотеки ethers.js было написано клиентское Web-приложение [33]. На каждой HTML-странице приложения содержится JS-скрипт, который при необходимости подключается к блокчейн-сети под адресом пользователя. Имея ABI смарт-контрактов и их адреса в сети, можно вызывать методы контрактов по протоколу JSON-RPC, которые доступны данному пользователю.

В приложении А на листинге 4 представлен код создания и чтения сертификата.

3.3 Методика тестирования разработанных модулей

Существуют три основных уровня тестирования разработанного программного обеспечения:

— Модульное тестирование — тестируется минимально возможный для тестирования компонент, например, отдельный класс или функция. Часто модульное тестирование осуществляется разработчиками программного обеспечения.

— Интеграционное тестирование — тестируются интерфейсы между компонентами, подсистемами или системами. При наличии резерва времени на данной стадии тестирование ведётся итерационно, с постепенным подключением последующих подсистем.

— Системное тестирование — тестируется интегрированная система на её соответствие требованиям.

Для тестирования разработанного ПО были написаны модульные тесты – с помощью среды Hardhat [30] были протестированы методы смарт-контрактов, в частности, инициирования смарт-контракта учебного заведения, а также создания и чтения невзаимозаменяемого токена. Также в рамках системного тестирования была протестирована блокчейн-сеть на предмет достижения кон-

сенсуса и финализации цепочек блоков.

3.4 Форматы входных и выходных данных

Чтобы создать смарт-контракт организации в сети, необходимо ввести названии заведения. В ответ будет получен 42-символьный шестнадцатеричный адрес созданного смарт-контракта.

Входными данными для создания невзаимозаменяемого токена являются ссылка на документ об образовании, адрес смарт-контракта учебной организации, и адрес выпускника. Выходными данными будут являться ID созданного сертификата. Для проверки сертификата необходимы ID токена и адрес смарт-контракта организации, которая выдала его. Вывод будет содержать метаданные сертификата, в случае если он существует, иначе сообщение об ошибке.

Входные данные:

— название учебного заведения;

— метаданные сертификата (ссылка на документ, адрес выпускника);

— адрес смарт-контракта учебной организации;

— ID сертификата.

Выходные данные:

— адрес созданного смарт-контракта учебного заведения;

— ID выпущенного сертификата;

— метаданные проверяемого сертификата.

3.5 Информация, необходимая для сборки и запуска

Минимальные технические требования для машины подобраны с учётом, минимальных системных требований к Substrate-проектам [10]:

— четырехъядерный центральный процессор;

— 8 гигабайт оперативной памяти;

— 50 гигабайт свободного места на HDD/SSD машины.

Выводы

В данном раздела было дано обоснование выбора средств реализации разрабатываемого метода. Была описана структура и схема взаимодействия моду-

лей разрабатываемого программного обеспечения. Было дано описание входных и выходных данных для различных сценариев использования ПО; методика тестирования узлов и смарт-контрактов сети. Также были представлены минимальные технические требования для запуска ПО.

4 Исследовательская часть

В данном разделе проверяется успешности чтения сертификата участниками сети. Приводится исследование времени финализации транзакций в сети с разным количеством узлов. Также выполнено сравнение реализованного метода с аналогами.

4.1 Проверка успешности чтения сертификата

Созданный сертификат принадлежит выпускнику и учебному заведению, что его выдало. Соответственно, прочитать этот сертификат могут только эти два участника сети. Чтобы подтвердить это утверждение был проведен соответствующий эксперимент.

Для проведения эксперимента потребовались следующие входные данные:

— адрес $addr$ смарт-контракта некоторой тестовой учебной организации (0x0x9f7F6a7C28e9d0733fD2d59071bA500B54430044);

— публичный ключ A выпускника №1 (0x6Be02d1d3665660d22FF9624b7BE0551ee1Ac91b);

— публичный ключ B выпускника №2 (0x8097c3C354652CB1EEed3E5B65fBa2576470678A).

В рамках эксперимента был создан смарт-контракт некоторой тестовой учебной организации под адресом $addr$. Далее был создан сертификат, выданный участнику A (рисунок 10). Далее участник A успешно «прочитал» выданный ему сертификат (скриншот прочтения сертификата изображен на рисунке 11). При попытке прочитать тот же самый сертификат участником B, вместо метаданных сертификата пользователь получил информацию об ошибке чтения, которая приведена на рисунке 12.

Проведя данный эксперимент, было получено подтверждение утверждения о том, что только владелец сертификата может его прочитать. Таким образом, можно судить о конфиденциальности сертификатов в сети: злоумышле-

Рисунок 10 – Создание уникального сертификата

ники не будут иметь доступ к сертификатам, а соответственно и документам, других пользователей.

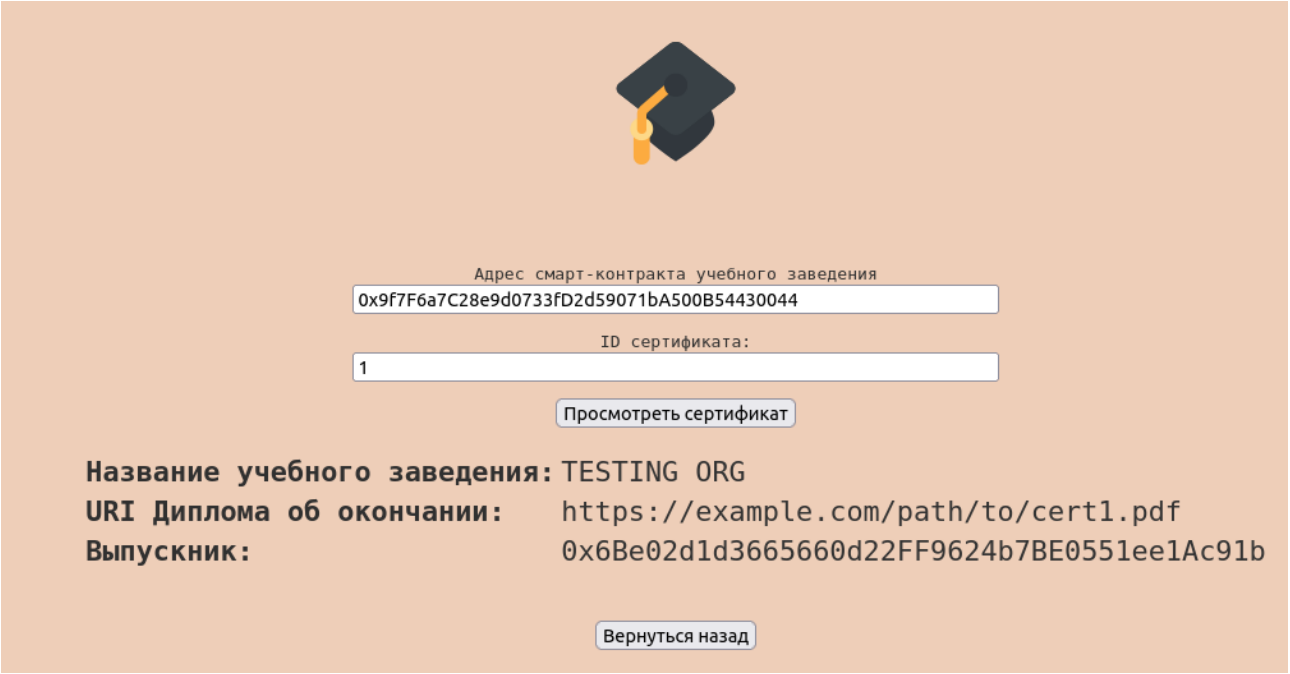

Рисунок 11 – Чтение сертификата выпускником-владельцем этого сертификата

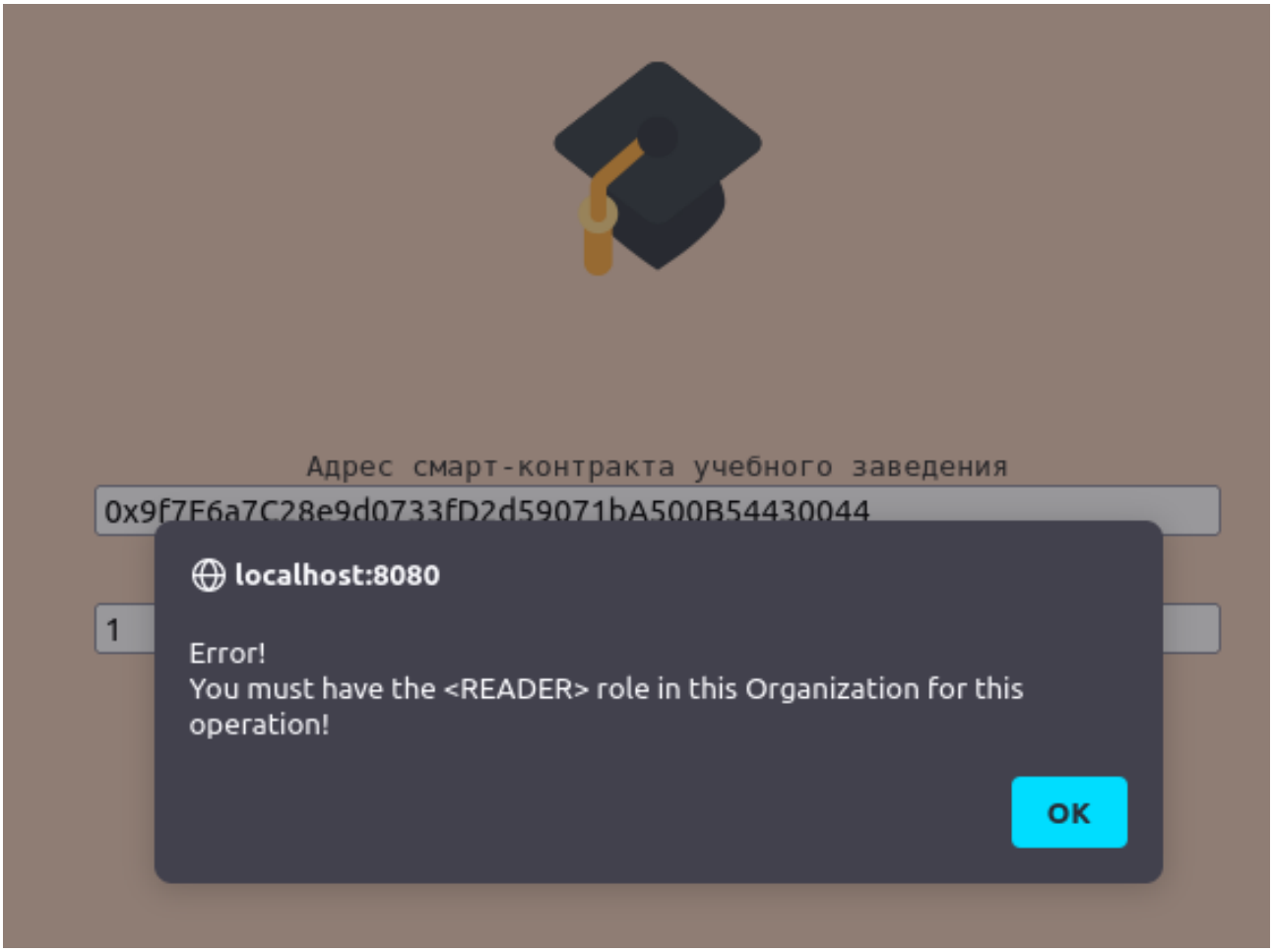

Рисунок 12 – Чтение сертификата другим участником сети

4.2 Проведение исследования времени финализации транзакций в сети

Было проведено исследование зависимости времени финализации транзакций, от количества участников-валидаторов в сети. Для этого были произведены замеры времени финализации транзакций в сети с 1, 2, 3, 4 и 5 валидаторами.

В качестве замеряемых транзакций были выбраны транзакции создания смарт-контракта учебного заведения и выдачи уникального сертификата выпускнику. Для каждого количества участников сети и было произведено по 5 замеров времени финализации каждой транзакции и взято соответствующее среднее значение. На рисунках 13 и 14 приведены столбиковые диаграммы результатов замеров времени финализации транзакции создания смарт-контракта учебного заведения и выдачи сертификата выпускнику, соответственно.

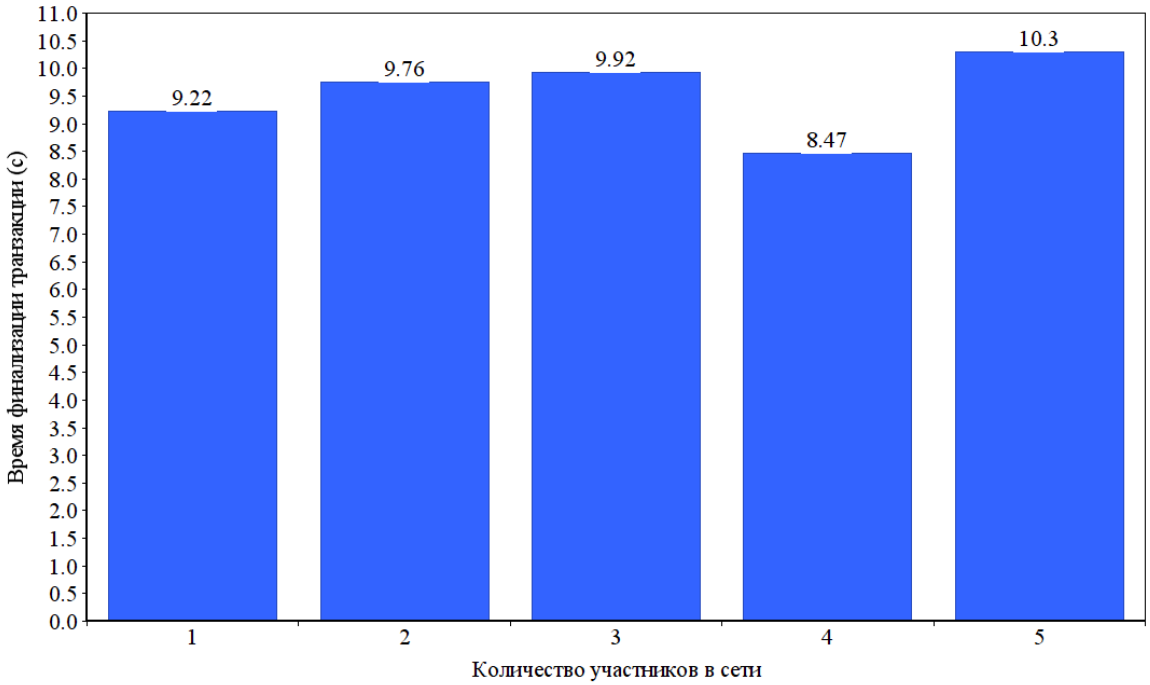

Рисунок 13 – Зависимость времени финализации транзакции создания смарт-контракта учебного заведения от количества участников в сети

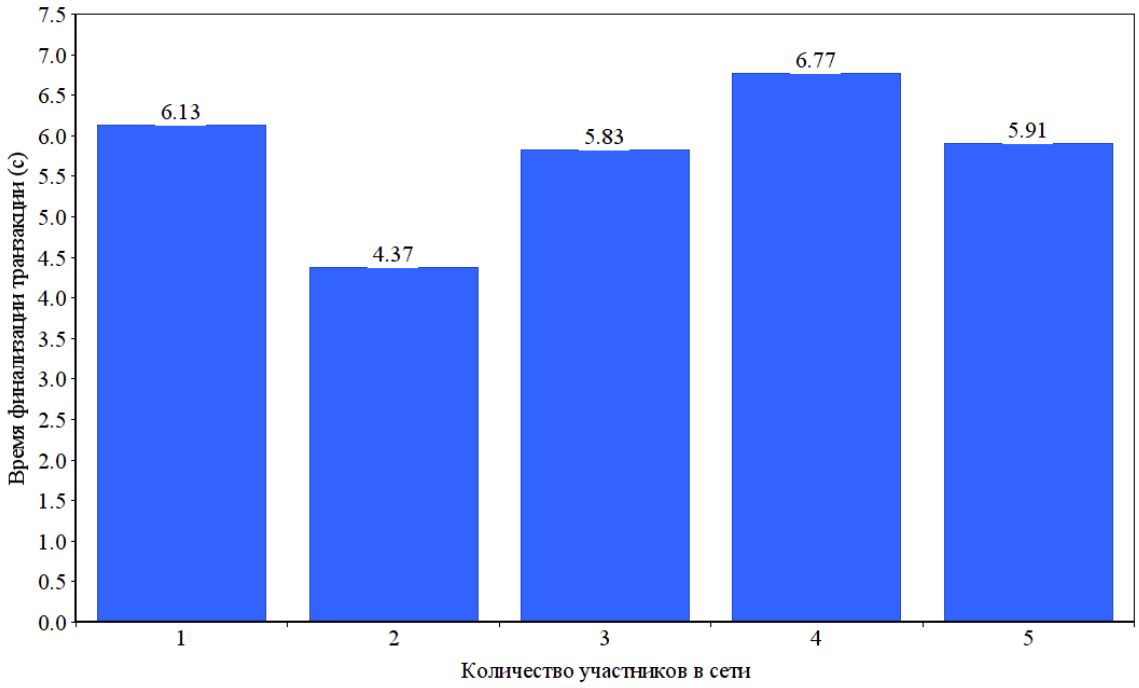

Рисунок 14 – Зависимость времени финализации транзакции создания уникального сертификата от количества участников в сети

Дисперсии полученных результатов зависимости времени финализации транзакции создания смарт-контракта учебного заведения и создания уникального сертификата от количества участников в сети составляют 0.5044 и 0.7769 соответственно. Данные значения дисперсии говорят о кучности полученных значений – полученные результаты релевантны.

На основании полученных результатов, можно сделать вывод, что время финализации транзакции как создания смарт-контракта учебного заведения, так и создания уникального сертификата не зависит от количества участников в сети, а зависит от временных интервалов производства новых блоков и их финализации, а также времени распространения информации о новом блоком среди валидаторов сети. Данный вывод позволяет судить о том, что в будущем разработанную сеть можно будет успешно масштабировать на большее число участников.

Также отметим, что среднее время создания смарт-контракта учебного заведения на 63% больше, чем время создания уникального сертификата. Это обусловлено тем, что для создания смарт-контракта виртуальная машина выполняет больше действий, нежели чем при создании невзаимозаменяемого токена.

4.3 Сравнение реализованного метода с аналогами

Было произведено сравнение разработанного метода с рассмотренными в аналитической части аналогами: сертификатами, выдаваемые образовательными онлайн-платформами, (Coursera, в частности), и документами об окончания вуза РФ.

Документы об окончания вуза РФ подвержены износу, их можно потерять. Чтобы их восстановить необходимо обращаться в уполномоченные органы. В то же время, предложенный метод персистентно хранит все сертификаты в блокчейне, лишая выпускника задач сохранения и восстановления выданных ему документов.

Сертификаты Coursera можно подделать (отредактировать) в текстовом редакторе. Предложенный же метод не подразумевает изменение созданных сертификатов, то есть однажды созданный сертификат больше нельзя отредактировать.

Чтобы проверить подлинность и сертификатов Coursera, и документов об окончании вуза РФ, необходимо сформировать официальный запрос и дождаться ответа на него. В свою очередь, реализованный метод позволяет получить подтверждение подлинности документов, не оформляя никаких запросов в уполномоченные организации, что существенно ускоряет время подтверждения.

На таблице 5 представлено сравнение реализованного метода с существующими реализациями выдачи сертификатов, подтверждающих окончание учебного заведения.

Таблица 5 – Сравнение реализованного метода с аналогами

Критерий / Реализация	Утрата	Износ	Подделывание	Проверка подлинности
Документы об окончания вуза РФ	-	-	+	-
Сертификаты Coursera	+	+	-	-
Предложенный метод	+	+	+	+

Выводы

В данном раздела были проведены исследования успешности чтения сертификата участниками сети, а также зависимости времени финализации транзакций в сети с разным количеством узлов. В ходе данных исследований было установлено, что:

— разработанные сертификаты конфиденциальны: личные данные в сертификате доступны только владельцу сертификата;

— время финализации транзакции не зависит от количества участников в сети, а зависит от временных интервалов производства новых блоков и их финализации, а также времени распространения информации о новом блоком среди валидаторов сети;

— среднее время создания смарт-контракта учебного заведения примерно в полтора раза больше, чем время создания уникального сертификата.

Также было произведено сравнение существующими аналогами выдачи сертификатов, подтверждающих окончание учебного заведения. Было установлено, что предложенный метод является предпочтительным.

ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной квалификационной работы были решены поставленные задачи:

— проанализированы существующие способы выдачи сертификатов об окончании учебного заведения;
— разработана блокчейн-сеть, в которой реализован метод выдачи уникальных сертификатов;
— программно реализован разработанный метод;
— исследована зависимость времени финализации транзакции от количества участников сети, а также произведено сравнение с аналогами; разработанное программное обеспечение полностью соответствует требованиям технического задания.

Был спроектирован и реализован метод создания невзаимозаменяемых токенов в блокчейн-сети, с помощью которых можно подтвердить окончание учебного заведения.

Достоинства разработанного программного обеспечения по сравнению существующими аналогами:

— защита от износа и подделывания выданных документов об образовании;
— возможность предоставить электронное подтверждение подлинности документа, не оформляя бюрократических запросов;
— неограниченное количество участников сети.

Основными недостатками разработанного программного обеспечения являются:

— большие вычислительные мощности, требуемые от машины;
— в случае если документ, хранящийся по ссылке в сертификате, скомпрометирован, тогда соответствующий сертификат тоже будет ложный.

Можно выделить следующие пути дальнейшего развития разработанной системы:

— переписывание кода смарт-контрактов на Rust;

— произвести внешний аудит разработанных смарт-контрактов;

— развить блокчейн-сеть до парачейна Polkadot [13].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Башир, И. Блокчейн: архитектура, криптовалюты, инструменты разработки, смарт-контракты / И. Башир ; перевод с английского М. А. Райтмана. – Москва : ДМК Пресс, 2019. – 538 с. – ISBN 978-5-97060-624-7. – Текст : электронный // Лань : электронно-библиотечная система. – URL: https://e.lanbook.com/book/123701 (дата обращения: 12.05.2022). — Режим доступа: для авториз. пользователей.

2. Цихилов, А. М. Блокчейн : Принципы и основы : практическое руководство / А. М. Цихилов. – Москва : Интеллектульная Литература, 2019. – 188 с. – ISBN 978-5-6042880-1-6. – Текст : электронный. – URL: https://znanium.com/catalog/product/1874893 (дата обращения: 12.05.2022). – Режим доступа: по подписке.

3. Coursera | Online Courses & Credentials from Top Educators [Электронный ресурс]. – Режим доступа: https://www.coursera.org/, свободный – (15.01.2022)

4. Zibin Zheng, Shaoan Xie – An Overview of Blockchain Technology: Architecture, Consensus, and Future Trend [Электронный ресурс]. – Режим доступа: https://www.researchgate.net/publication/318131748_An_Overview_of_Blockchain_Technology_Architecture_Consensus_and_Future_Trends, свободный – (15.01.2022)

5. Ethereum Virtual Machine (EVM) [Электронный ресурс]. – Режим доступа: https://ethereum.org/ru/developers/docs/evm/, свободный – (17.01.2022)

6. Telegram Open Network Virtual Machine [Электронный ресурс]. – Режим доступа: https://ton.org/tvm.pdf, свободный – (17.01.2022)

7. VM Matters: A Comparison of WASM VMs and EVMs in the Performance of Blockchain Smart Contracts [Электронный ресурс]. – Режим доступа: https://arxiv.org/pdf/2012.01032.pdf, свободный – (17.01.2022)

8. An Overview on Smart Contracts: Challenges, Advances and Platforms [Электронный ресурс]. – Режим доступа: https://arxiv.org/pdf/1912.10370.pdf, свободный – (17.01.2022)

9. EOSIO Development Documentation [Электронный ресурс]. – Режим доступа: https://developers.eos.io/, свободный – (17.01.2022)

10. Substrate Documentation [Электронный ресурс]. – Режим доступа: https://docs.substrate.io/v3/getting-started/overview/, свободный – (17.01.2022)

11. Cosmos Whitepaper [Электронный ресурс]. – Режим доступа: https://v1.cosmos.network/resources/whitepaper, свободный – (24.01.2022)

12. Табернакулов, А. Блокчейн на практике : практическое руководство / А. Табернакулов, Я. Койфманн. – Москва : Альпина Паблишер, 2019. – 260 с. – ISBN 978-5-9614-2382-2. – Текст : электронный. - URL: https://znanium.com/catalog/product/1874892 (дата обращения: 12.05.2022). – Режим доступа: по подписке.

13. Overview of Polkadot and its Design Considerations [Электронный ресурс]. – Режим доступа: https://eprint.iacr.org/2020/641.pdf, свободный – (24.01.2022)

14. Bitcoin: A Peer-to-Peer Electronic Cash System [Электронный ресурс]. – Режим доступа: https://www.ussc.gov/sites/default/files/pdf/training/annual-national-training-seminar/2018/Emerging_Tech_Bitcoin_Crypto.pdf,свободный – (12.02.2022)

15. Антонопулос, А. М. Осваиваем биткойн. Программирование блокчейна / А. М. Антонопулос ; перевод с английского А. В. Снастина. – Москва : ДМК Пресс, 2018. – 428 с. – ISBN 978-5-94074-965-3. – Текст : электронный // Лань : электронно-библиотечная система. – URL: https://e.lanbook.com/book/112924 (дата обращения: 12.05.2022). – Режим доступа: для авториз. пользователей.

16. Primecoin (XPM) Whitepaper [Электронный ресурс]. – Режим доступа: https://whitepaper.io/coin/primecoin, свободный – (12.02.2022)

17. BlackCoin (BLK) Whitepaper [Электронный ресурс]. – Режим доступа: https://whitepaper.io/coin/blackcoin, свободный – (12.02.2022)

18. PeerAssets Whitepaper [Электронный ресурс]. – Режим доступа: https://peerassets.github.io/WhitePaper/, свободный – (12.02.2022)

19. Delegated Proof of Stake with Downgrade: A Secure and Efficient Blockchain Consensus Algorithm with Downgrade Mechanism [Электронный ресурс]. – Режим доступа: https://www.researchgate.net/publication/335186093_Delegated_Proof_of_Stake_With_Downgrade_A_Secure_and_Efficient_Blockchain_Consensus_Algorithm_With_Downgrade_Mechanism, свободный – (12.02.2022)

20. Miguel Castro and Barbara Liskov – Practical Byzantine Fault Tolerance [Электронный ресурс]. – Режим доступа: https://pmg.csail.mit.edu/papers/osdi99.pdf, свободный – (12.02.2022)

21. Study of consensus protocols and improvement of the Delegated Byzantine Fault Tolerance (DBFT) algorithm [Электронный ресурс]. – Режим доступа: https://upcommons.upc.edu/bitstream/handle/2117/171243/Gavriel.Christofi-Thesis.pdf, свободный – (12.02.2022)

22. Tendermint: Consensus without Mining [Электронный ресурс]. – Режим доступа: https://tendermint.com/static/docs/tendermint.pdf, свободный – (12.02.2022)

23. Non-fungible tokens (NFT) | Ethereum Whitepaper [Электронный ресурс]. – Режим доступа: https://ethereum.org/en/nft/, свободный – (14.02.2022)

24. Non-Fungible Token (NFT): Overview, Evaluation, Opportunities

and Challenges [Электронный ресурс]. – Режим доступа: https://www.researchgate.net/publication/351656444_Non-Fungible_Token_NFT_Overview_Evaluation_Opportunities_and_Challenges, свободный – (14.02.2022)

25. GRANDPA: a Byzantine Finality Gadget – Alistair Stewart, Eleftherios Kokoris-Kogiahttps [Электронный ресурс]. – Режим доступа: https://arxiv.org/pdf/2007.01560.pdf, свободный – (19.04.2022)

26. Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis – Georg Becker [Электронный ресурс]. – Режим доступа: https://dokumen.tips/documents/merkle-signature-schemes-merkle-trees-and-their-cryptanalysis-therefore-alternative.html?page=3, свободный – (21.04.2022)

27. Язык программирования Rust [Электронный ресурс]. – Режим доступа: https://doc.rust-lang.ru/book/, свободный – (02.05.2022)

28. Язык программирования Solidity [Электронный ресурс]. – Режим доступа: https://docs.soliditylang.org/en/latest/, свободный – (02.05.2022)

29. Редактор кода VS Code [Электронный ресурс]. – Режим доступа: https://code.visualstudio.com/, свободный – (02.05.2022)

30. Hardhat | Ethereum development environment [Электронный ресурс]. – Режим доступа: https://hardhat.org/, свободный – (02.05.2022)

31. JSON-RPC API | Ethereum [Электронный ресурс]. – Режим доступа: https://ethereum.org/en/developers/docs/apis/json-rpc/, свободный – (07.05.2022)

32. OpenZeppelin Documentation [Электронный ресурс]. – Режим доступа: https://docs.openzeppelin.com/, свободный – (09.05.2022)

33. ethers.js Documentation [Электронный ресурс]. – Режим доступа: https://docs.ethers.io/v5/, свободный – (14.05.2022)

ПРИЛОЖЕНИЕ А

Листинг 1: Конфигурация узла сети блокчейн-сети

```
#![cfg_attr(not(feature = "std"), no_std)]
#![recursion_limit = "256"]


#[cfg(feature = "std")]
include!(concat!(env!("OUT_DIR"), "/wasm_binary.rs"));


use pallet_grandpa::{
    fg_primitives, AuthorityId as GrandpaId, AuthorityList as GrandpaAuthorityList,
};
use pallet_ethereum::{Call::transact, Transaction as EthereumTransaction};
use pallet_evm::{Account as EVMAccount, EnsureAddressTruncated, HashedAddressMapping, Runner};
use pallet_evm::FeeCalculator;


parameter_types! {
    pub const MaxAuthorities: u32 = 100;
}

impl pallet_aura::Config for Runtime {
    type AuthorityId = AuraId;
    type DisabledValidators = ();
    type MaxAuthorities = MaxAuthorities;
}

impl pallet_grandpa::Config for Runtime {
    type Event = Event;
    type Call = Call;

    type KeyOwnerProofSystem = ();

    type KeyOwnerProof =
    <Self::KeyOwnerProofSystem as KeyOwnerProofSystem<(KeyTypeId, GrandpaId)>>::Proof;

    type KeyOwnerIdentification = <Self::KeyOwnerProofSystem as KeyOwnerProofSystem<(
    KeyTypeId,
    GrandpaId,
    )>>::IdentificationTuple;

    type HandleEquivocation = ();

    type WeightInfo = ();
    type MaxAuthorities = ConstU32<32>;
}

```

```
parameter_types! {
    pub const ChainId: u64 = 100;
    pub BlockGasLimit: U256 = U256::from(u32::max_value());
    pub PrecompilesValue: FrontierPrecompiles<Runtime> = FrontierPrecompiles::<_>::new();
}

impl pallet_evm::Config for Runtime {
    type FeeCalculator = BaseFee;
    type GasWeightMapping = ();
    type BlockHashMapping = pallet_ethereum::EthereumBlockHashMapping<Self>;
    type CallOrigin = EnsureAddressTruncated;
    type WithdrawOrigin = EnsureAddressTruncated;
    type AddressMapping = HashedAddressMapping<BlakeTwo256>;
    type Currency = Balances;
    type Event = Event;
    type Runner = pallet_evm::runner::stack::Runner<Self>;
    type PrecompilesType = FrontierPrecompiles<Self>;
    type PrecompilesValue = PrecompilesValue;
    type ChainId = ChainId;
    type BlockGasLimit = BlockGasLimit;
    type OnChargeTransaction = ();
    type FindAuthor = FindAuthorTruncated<Aura>;
}

impl pallet_ethereum::Config for Runtime {
    type Event = Event;
    type StateRoot = pallet_ethereum::IntermediateStateRoot<Self>;
}

construct_runtime!(
pub enum Runtime where
Block = Block,
NodeBlock = opaque::Block,
UncheckedExtrinsic = UncheckedExtrinsic
{
    System: frame_system::{Pallet, Call, Config, Storage, Event<T>},
    // ...
    Aura: pallet_aura::{Pallet, Config<T>},
    Grandpa: pallet_grandpa::{Pallet, Call, Storage, Config, Event},
    // ...
    Ethereum: pallet_ethereum::{Pallet, Call, Storage, Event, Config, Origin},
    EVM: pallet_evm::{Pallet, Config, Call, Storage, Event<T>},
    // ...
}
);
```

Листинг 2: Код контракта-фабрики учебных заведений

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.2;
```

```
4
5  import "@openzeppelin/contracts/proxy/Clones.sol";
6  import "./Organization.sol";
7
8
9  contract OrganizationFactory {
10     address immutable _organizationImplementation;
11     address[] private _organizations;
12
13     constructor() {
14         _organizationImplementation = address(new Organization());
15     }
16
17     function createOrganization(string memory organizationName)
18     external
19     returns (address)
20     {
21         address cloned_address = Clones.clone(_organizationImplementation);
22         Organization(cloned_address)
23         .initOrganization(organizationName, msg.sender);
24         _organizations.push(cloned_address);
25         return cloned_address;
26     }
27
28     function getOrganizationAmount() public view returns (uint256) {
29         return _organizations.length;
30     }
31
32     function getOrganizationAddress(uint256 idx) public view returns (address) {
33         require(idx < _organizations.length, "getOrganizationAddress() —— Index out of
               range!");
34         return _organizations[idx];
35     }
36 }
```

Листинг 3: Код смарт-контракта учебного заведения

```
1  // SPDX-License-Identifier: MIT
2
3  pragma solidity ^0.8.2;
4
5  import "@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol";
6  import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
7  import "@openzeppelin/contracts-upgradeable/ access/AccessControlUpgradeable.sol";
8  import "@openzeppelin/contracts/utils/Counters.sol";
9  import "@openzeppelin/contracts/utils/math/SafeMath.sol";
10
11
12 contract Organization is ERC721Upgradeable, OwnableUpgradeable, AccessControlUpgradeable {
13     using Counters for Counters.Counter;
14
```

```
15      Counters.Counter private _diplomaIds;
16      string private _organizationName;
17      uint private constant MAX_NAME_LENGTH = 0xFF;
18      bool private _open;
19
20      mapping(uint256 => string) private _diplomaURIs;
21
22      bytes32 private constant READER_ROLE = keccak256("READER_ROLE");  // AccessControl role
23
24      event OrganizationCreated(address owner, string organizationName);
25      event DiplomaMinted(address owner, string organizationName, uint256 diplomaId);
26
27      function initOrganization(string memory organizationName, address _owner)
28      public
29      initializer
30      {
31          __ERC721_init("Organization", "ORG");
32          __Ownable_init();
33
34          require(bytes(organizationName).length <= MAX_NAME_LENGTH, "Organization name's too
                long!");
35          _organizationName = organizationName;
36
37          _grantRole(DEFAULT_ADMIN_ROLE, _owner);
38          _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
39          grantRole(READER_ROLE, _owner);
40          transferOwnership(_owner);
41          emit OrganizationCreated(_owner, organizationName);
42      }
43
44      modifier onlyReader() {
45          if (!_open) {
46              require(hasRole(READER_ROLE, msg.sender), "Restricted to <READER> role only!");
47          }
48          _;
49      }
50
51      function isReader(address account) public view returns (bool) {
52          if (_open) {
53              return true;
54          }
55          return hasRole(READER_ROLE, account);
56      }
57
58      function addReader(address account) public virtual onlyOwner {
59          grantRole(READER_ROLE, account);
60      }
61
62      function removeReader(address account) public virtual onlyOwner {
63          revokeRole(READER_ROLE, account);
64      }
```

```
65
66      function open() public virtual onlyOwner {
67          _open = true;
68      }
69
70      function close() public virtual onlyOwner {
71          _open = false;
72      }
73
74      // ----------------------------
75      // View functions -------------
76      // ----------------------------
77
78      function getOrganizationName() public view onlyReader returns (string memory) {
79          return _organizationName;
80      }
81
82      function getDiploma(uint256 idx) public view onlyReader returns (string memory) {
83          require((idx <= _diplomaIds.current()), "Index out of range!");
84          return _diplomaURIs[idx];
85      }
86
87      function getDiplomaAmount() public view onlyReader returns (uint256) {
88          return _diplomaIds.current();
89      }
90
91      function isOpen() public view returns (bool) {
92          return _open;
93      }
94
95      // Imposed by OpenZeppelin...
96      function supportsInterface(bytes4 interfaceId)
97      public
98      view
99      virtual
100     override(AccessControlUpgradeable, ERC721Upgradeable)
101     returns (bool)
102     {
103         return
104         interfaceId == type(IAccessControlUpgradeable).interfaceId ||
105         interfaceId == type(IERC721Upgradeable).interfaceId ||
106         interfaceId == type(IERC721MetadataUpgradeable).interfaceId ||
107         super.supportsInterface(interfaceId);
108     }
109
110     // ----------------------------
111     // Mint functions -------------
112     // ----------------------------
113
114     function mintDiploma(address recipient, string memory diplomaURI) public onlyOwner returns
            (uint256) {
```

```
115         require(bytes(diplomaURI).length > 0, "Cannot mint empty diploma!");
116
117         _diplomaIds.increment();
118         uint256 newDiplomaId = _diplomaIds.current();
119
120         _mint(recipient, newDiplomaId);
121         _diplomaURIs[newDiplomaId] = diplomaURI;
122         // _setTokenURI(newDiplomaId, diplomaURI);
123
124         emit DiplomaMinted(recipient, _organizationName, newDiplomaId);
125
126         return newDiplomaId;
127     }
128
129     function transferFrom(address from, address to, uint256 tokenId)
130     public
131     virtual
132     override
133     {
134         require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer caller is not
                owner nor approved");
135
136         _transfer(from, to, tokenId);
137         _grantRole(READER_ROLE, to);
138     }
139
140     function safeTransferFrom(address from, address to, uint256 tokenId, bytes memory _data)
141     public
142     virtual
143     override
144     {
145         require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer caller is not
                owner nor approved");
146
147         _safeTransfer(from, to, tokenId, _data);
148         _grantRole(READER_ROLE, to);
149     }
150 }
```

Листинг 4: Создание и чтение сертификата с помощью клиентского приложения

```
1 "use strict";
2
3 import organizationABI from "../data/OrganizationABI.json";
4 const { ethers } = require("ethers");
5
6
7 let organization;
8 let org;
```

```
9
10 function _checkAddress(addr) {
11     if (!ethers.utils.isAddress(addr)) {
12         window.alert("Oops!\nInvalid address!");
13         return false;
14     }
15     return true;
16 }
17
18 const App = {
19     provider: null,
20     signer: null,
21
22     start: async function () {},
23     createCert: async function () {
24         const contractAddr = document.getElementById("contractAddr").value;
25         const recipientAddr = document.getElementById("recipientAddr").value;
26         if (!_checkAddress(contractAddr)) {
27             return;
28         }
29         if (!_checkAddress(recipientAddr)) {
30             return;
31         }
32
33         const certURI = document.getElementById("certBody").value;
34         if (certURI.length == 0) {
35             window.alert("Error! Cannot mint empty NFT!");
36             return;
37         }
38
39         try {
40             organization = new ethers.Contract(contractAddr, organizationABI, this.provider);
41         } catch (err) {
42             window.alert("Error!\nInvalid Organization Contract Address");
43             return;
44         }
45
46         const status = document.getElementById("status");
47         status.innerHTML = Создаем" NFTсертификат—... пожалуйста, дождитесьзавершения .";
48
49         org = await organization.deployed();
50         const tx = await org.connect(App.signer).mintDiploma(recipientAddr, certURI);
51         await tx.wait();
52
53         const tokenId = await org.connect(App.signer).getDiplomaAmount();
54         status.innerHTML = "ID сертификата: " + tokenId;
55         return;
56     },
57     readPage: async function () {
58         const contractAddr = document.getElementById("contractAddr ").value.toString();
59         if (!_checkAddress(contractAddr)) {
```

```
60              return;
61          }
62
63          try {
64              organization = new ethers.Contract(contractAddr, organizationABI, this.provider);
65          } catch (err) {
66              window.alert("Error!\nInvalid Organization Contract Address");
67              return;
68          }
69
70          let signerAddr = await App.signer.getAddress();
71          if (!(await organization.connect(App.signer).isReader(signerAddr))) {
72              window.alert("Error!\nYou must have the <READER> role in this Organization for
                    this operation!");
73              return;
74          }
75
76          const tokenId = document.getElementById("tokenId").value;
77          if (tokenId > await organization.connect(App.signer).getDiplomaAmount()) {
78              window.alert("Error!\nInvalid DiplomaID!");
79              return;
80          }
81
82          const read = document.getElementById("read");
83          read.innerHTML = "<br><brЗагружаем> сертификат... пожалуйста, подождите."
84
85          let organizationName;
86          let diplomaURI;
87          let graduate;
88          try {
89              organizationName = await organization.connect(App.signer)
90              .getOrganizationName();
91              diplomaURI = await organization.connect(App.signer).getDiploma(tokenId);
92              graduate = await organization.connect(App.signer).ownerOf(tokenId);
93          } catch (err) {
94              window.alert("error: unable to View");
95              return;
96          }
97
98          let pageHTML = "<br><br><table class=\"tg\" style=\"max-width:400px;font-size:
                20px\"><tbody>";
99          pageHTML += "<tr><td class=\"tg-leftНазвание\"> учебногозаведения : </td><td
                class=\"tg-right\">" + organizationName + "</td></tr>";
100         pageHTML += "<tr><td class=\"tg-left\">URI Дипломаобокончании  : </td><td
                class=\"tg-right\">" + diplomaURI + "</td></tr>";
101         pageHTML += "<tr><td class=\"tg-leftВыпускник\">: </td><td class=\"tg-right\">" +
                graduate + "</td></tr>"
102         pageHTML += "</tbody></table>"
103         read.innerHTML = pageHTML;
104         return;
105     }
```

```
106 };
107
108
109 window.App = App;
110 window.addEventListener("load", async function () {
111     await window.ethereum.enable();
112
113     App.provider = new ethers.providers.Web3Provider(window.ethereum);
114     App.signer = App.provider.getSigner();
115     App.start();
116 });
```