МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование кафедры)
01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки, специальности)
Системное программирование и компьютерные технологии
(направленность (профиль)/специализация

БАКАЛАВРСКАЯ РАБОТА

на тему Алгоритм безопасной передачи данных между пользователями в сети при помощи криптографической подписи

Студент	А.А. Кузьмин	
	(И.О. Фамилия)	(личная подпись)
Руководитель	А.В. Шляпкин	
	(И.О. Фамилия)	(личная подпись)
Консультанты	М.А. Четаева	
	(И.О. Фамилия)	(личная подпись)

Допустить к защите

Заведующий кафедрой к.т.н., доцент, А.В. Очеповский _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)
«_____»_____20_____г.

Тольятти 2018

АННОТАЦИЯ

Дипломная работа посвящена вопросу разработки автоматизированной системы безопасной передачи данных между пользователями в сети при помощи криптографической подписи с открытым и закрытым ключами.

Данная дипломная работа состоит из пояснительной записки на 48 страниц, включая введение на 1 страницу, 3 таблицы, 19 рисунков, списка 20 источников, включая 5 источников на иностранном языке. Ключевым вопросом в дипломной работе является безопасность передачи данных в сети Интернет между пользователями. Особое внимание уделяется криптографической системе шифрования RSA и платформе для создания децентрализованных онлайн-сервисов на базе блокчейна - Ethereum. Программный код написан с помощью языков программирования JavaScript и Solidity и программной платформы Node.js. Дипломная работа может быть разделена на следующие логические части: шифрование, виды шифрования и блокчейн.

В дипломной работе подробно описывается шифрование. Сначала мы изучаем его роль в криптографии и безопасности в сети Интернет. Рассматриваем виды шифрования, такие как симметричные и асимметричные. В отдельной части дипломной работы подробно рассказывается о принципе работы технологии блокчейна Ethereum, почему была выбрана именно эта платформа и как работает смарт-контракт. Были сравнены некоторые конкурентные мессенджеры – WhatsApp, Telegram, Viber, которые используют end-to-end шифрование, с решением, получившимся в результате проделанной работы.

Примененная методика, подтверждает, использование публичного (децентрализованного) хранения данных безопасно, так как взломать приватный ключ Ethereum практически невозможно.

ABSTRACT

The topic of the bachelor's thesis is Algorithm for secure data exchange between users on the network using a cryptographic signature.

The bachelor's thesis consists of an explanatory note on 48 pages, including introduction, 19 figures, 3 tables, a list of 20 references including 5 foreign references and 1 appendix. We touched upon the problem of secure data transmission in unreliable communication channels on the Internet. Particular attention was paid to the cryptographic system RSA and Ethereum - open blockchain platform that lets anyone build and use decentralized applications that run on blockchain technology. The program code was written using the JavaScript and Solidity programming languages, and the software platform Node.js. The bachelor's thesis may be divided into several logically connected parts, which are cryptography, types of encryption, and blockchain technology.

The bachelor's thesis describes encryption in detail. We studied the role of cryptography and security on the Internet. We considered types of encryption, such as symmetric and asymmetric encryption. The reader's attention is also drawn to Ethereum-technology, why this platform was chosen, and how the smart contract works. Some competitive messengers were compared - WhatsApp, Telegram, and others which use end-to-end encryption and cloud storage.

The results of the study showed that using public (decentralized) data storage is safe because it is almost impossible to crack the private Ethereum key.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В выпускной квалификационной работе (ВКР) будет рассмотрено решение задач безопасной передачи данных между пользователями в сети при помощи криптографической подписи с открытым и закрытым ключами.

Актуальность данной работы связана с тем, что складывается тенденция к увеличению количества атак и взломов тайных переписок. Методы и технологии удаленных сетевых атак регулярно совершенствуются, и существующие алгоритмы и системы шифрования не всегда полностью защищают конфиденциальную информацию. Но развитие технологий блокчейна, которые обладают высокой криптографической стойкостью, так же не стоят на месте. Эти обстоятельства делают разработку и внедрение системы безопасной передачи данных при помощи криптографической подписи с открытым и закрытым ключами весьма актуальными.

Цель**:** разработка автоматизированной системы безопасной передачи данных в сети между пользователями при помощи криптографической подписи с открытым и закрытым ключами.

Предмет исследования: алгоритмы шифрования данных.

Задачи работы:

- сравнить существующие системы по обмену мгновенными сообщениями;
- проанализировать и сравнить существующие алгоритмы и системы шифрования данных;
- изучить блокчейн платформу Ethereum и смарт-контракты;
- разработать на базе платформы Ethereum децентрализованную систему передачи и хранения данных.

ГЛАВА 1. СОВРЕМНННЫЕ СРЕДСТВА ОБМЕНА ИНФОРМАЦИЕЙ В СЕТИ ИНТЕРНЕТ И АЛГОРИТМЫ ШИФРОВАНИЯ

1.1 Программы по обмену мгновенными сообщениями (мессенджеры)

Мессенджер – это программа, мобильное приложение или веб-сервис для мгновенного обмена информацией в реальном времени через сеть Интернет. В настоящее время мессенджеры стали одним из главных способов коммуникации. С их помощью обмениваются не только сообщениями, но и фотографиями, музыкой, голосовыми сообщениями и даже документами. Нередко, передаваемая информация требует строгой конфиденциальности и недоступности третьим лицам. Поэтому разработчики приложений обмена данными обеспечивают их различными алгоритмами и системами шифровая данных. Самые популярные и распространенные мессенджеры – это Viber, WhatsApp, Telegram и другие.

1.2 Сравнение популярных мессенджеров

Каждый популярный мессенджер даёт пользователям возможность сохранять сообщения во избежание потери данных. Приложения, которые не имеют такой функции, например, Wickr, Signal, Confide, не набирают больше миллиона активных пользователей в день и остаются нишевыми.

Популярные приложения, такие как WhatsApp, Viber и Line, полагаются на Apple iCloud и Google Drive для хранения истории сообщений пользователей и предотвращения потери данных в случае утери или поломки смартфонов. Такие резервные копии не используют оконечное (end-to-end) шифрование и расшифровываются каждый раз, когда пользователь начинает использовать новый телефон и восстанавливает всю историю своих переписок. Хотя в некоторых случаях можно отказаться от резервного копирования, неизвестно точно, откажется ли от такой функции собеседники. Таким образом нет абсолютной гарантии, что отправленные и полученные сообщения не зашифрованы оконечно в облачном хранилище.

Невозможно узнать, что действительно шифруется при резервном копировании. Поэтому личные данные уязвимы для хакеров и злоумышленников, которые могут получить к ним доступ через облачное хранилище. Согласно информации представителей WhatsApp, большая часть оконечно зашифрованных переписок в мессенджере в итоге копируется и хранится в облаке в незашифрованном виде.

Иные мессенджеры, такие как Wickr, Signal, Confide, не делают никаких резервных копий, благодаря чему являются более безопасными. Но вместе с безопасностью появляются ограничения и недостатки. В том числе невозможность восстановить историю всех чатов при поломке, утрате или при смене смартфона. Поэтому большинство пользователей выбирают приложения, предоставляющие такие возможности и с меньшей вероятностью станут использовать мессенджеры, которые не популярны.

Telegram пошел обоими путями. В мессенджере реализованы два вида чатов – секретные и облачные. Секретные чаты поддерживают end-to-end шифрование и не включаются в резервные копии. Облачные чаты тоже зашифрованы, но копируются в облачное хранилище, которое принадлежит самому Telegram[6]. Пользователи всегда знают, какие из зашифрованных сообщений хранятся в облаке, а какие нет. В отличие от нишевых приложений, трафик между пользователями облачных и секретных чатов неотделим (используется одно и то же шифрование, но у сервера есть ключ шифрования от облачных чатов). Но, как и в случае с непопулярными мессенджерами, восстановить оконечно зашифрованные сообщения невозможно.

Но развивающиеся технологии блокчейн позволяют хранить данных публично, но безопасно. Мессенджер, основанный на такой технологии, позволит избежать хранения информации на серверах.

1.3 Шифрование данных (криптография)

Криптография – это преобразование информации в другую форму или код таким образом, что доступ к ней сможет получить только тот пользователь, у которого есть секретный ключ – ключ дешифрования.

Шифрование данных является одним из самых распространенных и эффективных методов защиты информации. Существуют два основных типа шифрования: симметричное, использующее закрытый(секретный) ключ, и ассиметричное - с открытым ключом.

Целью шифрования является обеспечение конфиденциальности данных. Криптография применяется для хранения важной информации в ненадежных источниках и передачи ее незащищенным каналам связи. Так как подавляющее большинство информации хранится на компьютерах и передается по сети Интернет, она уязвима к различного рода атакам. Изначально шифрование использовалось только для передачи конфиденциальной информации. Для шифрования применялся метод DES, который был разработан в 70-х годах. Сейчас он считается устаревшим. В настоящее время используется более сложные и надежные алгоритмы[2].

Данные, требующие шифровки, называются открытым текстом. После шифрования такой текст выглядит как набор символов, не несущих в себе смысла. Расшифровать и просмотреть исходную информацию можно только при наличии подходящего ключа. Симметричное шифрование требует использования одного и того же ключа как для защиты, так и для последующей расшифровки информации. При этом отправитель файла и его получатель должны обменяться ключом. Такая система шифрования работает быстрее ассиметричной. Ассиметричная криптография использует два разных ключа – открытый(публичный) и закрытый. Публичный ключ может быть передан всем, в то врем как закрытый остается защищенным.

Шифрование может обеспечить высокий уровень защиты, но оно не дает абсолютной гарантии, что данные не будут похищены злоумышленниками. Самый распространенный способ атаки - это брут-форс или подборка ключа,

которая производится до тех пор, пока не будет найдено совпадение. Надежность защиты прямо пропорциональна размеру ключа: чем ключ длиннее, тем больше времени и ресурсов затратит взломщик на его подбор. Существуют альтернативные методы взлома – атака по сторонним каналам и криптоанализ. Атака по сторонним каналам производится на уже зашифрованный файл. Успех такой атаки возможен, если при проектировании системы были допущены ошибки. Криптоанализ же производит поиск уязвимостей в самом коде. С развитием криптоанализа, появились методики, позволяющие дешифровать закрытый текст не имея ключа. Они основаны на математическом анализе перехваченных данных.

Организации вынуждены прибегать к криптографии, так как сотрудники не редко используют съемные носители информации и загружают файлы в облачные хранилища. Из-за этого, данные выходят из-под контроля защитной системы организации и возникает риск их утери и перехвата злоумышленниками. Шифрование позволяет предотвратить кражу информации и не допустить внедрение вредоносного программного обеспечения, но сохранить всю информацию доступной доверенным лицам.

1.3.1 Симметричное шифрование

Симметричное шифрование - это метод криптографии, в котором один ключ отвечает за шифрование и дешифрование данных. Участвующие стороны разделяют этот ключ, пароль или кодовую фразу, и они могут использовать его для дешифрования или шифрования любых сообщений, которые они хотят. Согласно проекту защиты открытых веб-приложений, некоторые из наиболее распространенных алгоритмов, используемых для симметричной криптографии, включают в себя стандарт шифрования данных (DES), который использует 56-битные ключи. Тройной DES, который трижды применяет алгоритм DES с различными ключами; и стандарт расширенного шифрования (AES), алгоритм, который Национальный институт стандартов и технологий

Соединенных Штатов рекомендует использовать для безопасного хранения и передачи данных.

Симметричные ключевые шифры или алгоритмы, используемые для шифрования и дешифрования, привлекательны для организаций, потому что они недороги, несмотря на уровень защиты, который они предоставляют. Действительно, аутентификация встроена в симметричную криптографию, поскольку стороны не могут расшифровывать данные, зашифрованные одним симметричным ключом, используя другой симметричный ключ. Центр знаний IBM отмечает, что симметричные ключевые шифры также меньше по размеру. Это свойство помогает минимизировать задержку времени, связанную с шифрованием и дешифровкой данных.

Но симметричное шифрование не является совершенным. Ключи в этом методе криптографии живут вечно, что означает, что организации должны инвестировать в ведение журнала и аудит ключей в течение их жизненного цикла. Это также означает, что, если симметричный ключ потерян, организации не могут его вспомнить. Вместо этого они должны шифровать и расшифровывать данные с помощью другого ключа, как только они восстановят свои данные в незашифрованном виде.

Учитывая бизнес-расходы, связанные с потерей симметричного ключа, компаниям необходимо проявлять большую осторожность, чтобы заинтересованные стороны надежно обменяли свой ключ. На основе базы знаний Венафи один ответ - это система содержания под стражей, посредством которой хранители получают части ключа из модуля аппаратной безопасности (HSM) или физического вычислительного устройства, которое управляет ключами. Затем они защищают эти компоненты и отправляют их получателям, которые вводят соответствующие ключевые фрагменты в HSM для формирования ключа. Успешный ввод всех компонентов позволяет участвующим сторонам шифровать и расшифровывать данные с помощью завершенного ключа[19].

В качестве альтернативы, хранитель может получить симметричный ключ, обернутый асимметричным хранилищем ключей. Затем хранитель отправляет это хранилище ключей получателю, который загружает хранилище ключей в HSM. Модуль, в свою очередь, разворачивает хранилище ключей, тем самым позволяя получателю шифровать и расшифровывать сообщения. Конечно, этот метод имеет свои пределы. Если получателю всегда нужен другой ключ для шифрования симметричного ключа, все может выйти из-под контроля и привести к бесконечному циклу ключей в зависимости от дополнительных ключей.

Когда все сказано и сделано, организациям нужен способ контролировать свои ключи. Этот процесс может стать ресурсоемким, поскольку, если нескольким сторонам необходимо установить свои собственные защищенные каналы связи друг с другом с использованием симметричного шифрования, им понадобятся собственные ключи для каждого канала. Именно поэтому в интересах организаций автоматизировать управление ключами.

1.3.1.1 Data Encryption Standard (DES)

DES - симметричный алгоритм шифрования, в котором один ключ используется как для шифрования, так и для расшифрования данных. DES разработан фирмой IBM и утвержден правительством США в 1977 году как официальный стандарт. Алгоритм DES широко использовался при хранении и передаче данных между различными вычислительными системами; в почтовых системах, в электронных системах чертежей и при электронном обмене коммерческой информацией. Стандарт DES реализовывался как программно, так и аппаратно. Предприятиями разных стран был налажен массовый выпуск цифровых устройств, использующих DES для шифрования данных. Все устройства проходили обязательную сертификацию на соответствие стандарту.

Алгоритм шифрования DES использует ключ длиной 56 бит. В то время считалось, что опробовать все 72 055 594 037 927 936 возможных ключей (семь

с 16 нулями) было бы невозможно, потому что компьютеры настолько мощными. В 1998 году Electronic Frontier Foundation (EFF) создал специальную машину, которая могла расшифровать сообщение, опробовав все возможные ключи менее чем за три дня. Машина стоила менее 250 000 долларов США и искала более 88 миллиардов ключей в секунду[1].

Размер блока в DES - 64 бита, для шифрования использует ключ с длиной 56 бит, количество раундов – 16. DES является классической сетью Фейштеля с двумя ветвями. За несколько раундов алгоритм преобразует 64-битный входной блок данных в 64-битный выходной блок. Стандарт DES построен на комбинированном использовании перестановки, замены и гаммирования. Данные для шифрования должны быть представлены в двоичном виде.

Процесс шифрования каждого 64-битового блока исходных данных можно разделить на три этапа:

1. начальная подготовка блока данных;
2. 16 раундов "основного цикла";
3. конечная обработка блока данных.

На первом этапе выполняется начальная перестановка 64-битного исходного блока данных. При начальной перестановке биты блока данных определенным образом переупорядочиваются, что придает некоторую "хаотичность" исходному сообщению, понижая возможность использования хкриптоанализа статистическими методами. Одновременно с начальной перестановкой блока данных выполняется начальная перестановка 56 бит ключа. В каждом из циклов используется соответствующий 48-битный частичный ключ. Ключи получаются по определенному алгоритму, используя каждый из битов начального ключа по нескольку раз. В каждом раунде 56-битный ключ делится на две 28-битовые части. Затем части сдвигаются влево на один или два бита зависимо от номера раунда. После сдвига определенным образом выбирается 48 из 56 битов. Из-за того, что при этом не только выбирается подмножество битов, но и изменяется их порядок, эта операция

называется "перестановка со сжатием". Ее результатом является набор из 48 битов. В среднем каждый бит исходного 56-битного ключа используется в 14 из 16 подключей, хотя не все биты используются равное количество раз.

На втором этапе блок делится на две ветви по 32 бита каждая и выполняется основной цикл преобразования, организованный по сети Фейштеля и состоящий из 16 одинаковых раундов. При этом в каждом раунде получается промежуточное 64-битное значение, которое затем обрабатывается в следующем раунде.

Вначале правая часть блока увеличивается до 48 битов, используя таблицу, которая определяет перестановку плюс расширение на 16 битов. Эта операция приводит размер правой части в соответствие с размером ключа для выполнения операции XOR. За счет выполнения этой операции быстрее возрастает зависимость всех битов результата от битов исходных данных и ключа.

После выполнения перестановки с расширением для полученного 48-битного значения выполняется операция XOR с 48-битным подключом. Затем полученное 48-битное значение передается на вход блока подстановки, результат которой - 32-битное значение. Подстановка выполняется в восьми блоках подстановки. При выполнении этой операции 48 битов данных разделяются на восемь 6-битовых подблоков, каждый из которых по своей таблице замен заменяется четырьмя битами. Подстановка с помощью S-блоков является одним из важнейших этапов DES. Таблицы замен для этой операции специально спроектированы так, чтобы обеспечивать максимально возможную безопасность. В результате этого этапа получаются восемь 4-битовых блоков, которые вновь объединяются в единое 32-битовое значение.

Далее полученное 32-битовое значение обрабатывается с помощью независящей от используемого ключа перестановки. Целью перестановки является максимальное переупорядочивание битов, чтобы в следующем цикле шифрования каждый бит с большой вероятностью обрабатывался другим блоком перестановки.

Результат перестановки объединяется с помощью операции XOR с левой половиной первоначального 64-битового блока данных. Далее левая и правая части меняются местами, и начинается следующий раунд.

На втором этапе блок делится на две части (ветви) по 32 бита каждая. Правая ветвь преобразуется, используя некоторую функцию и соответствующий частичный ключ, который получается из основного ключа шифрования по специальному алгоритму преобразования ключей. Далее производится обмен данными между левой и правой ветвями блока. Это повторяется в цикле 16 раз.

Наконец, на последнем третьем этапе производится перестановка результата, полученного после шестнадцати шагов основного цикла. Эта перестановка обратна начальной перестановке.

После выполнения всех шагов блок данных считается полностью зашифрованным и можно переходить к шифрованию следующего блока сообщения.

1.3.1.2 Advanced Encryption Standard (AES)

Advanced Encryption Standard (AES) представляет собой симметричный шифр блочного типа, выбранный правительством США для защиты секретной информации и реализованный в программном и аппаратном обеспечении во всем мире для шифрования конфиденциальных данных.

Национальный институт стандартов и технологий (NIST) начал разработку AES в 1997 году, когда объявил о необходимости использования алгоритма преемника для устаревшего алгоритма Data Encryption Standard (DES), который стал уязвимым для брут-форс атак.

В качестве расширенного стандарта AES был выбран алгоритм Rijndael, разработанный бельгийскими криптографами Винсентом Рэйменом и Йоаном Дайменом и отличавшийся повышенной безопасностью, производительностью и гибкостью.

Алгоритм Rijndael представляет собой симметричный блочный шифр, который поддерживает размеры ключей 128, 192 и 256 бит, причем данные обрабатываются в 128-битных блоках, однако, помимо критериев проектирования AES, размеры блоков могут быть зеркальными для ключей[3]. Rijndael использует переменное количество раундов, в зависимости от размера ключа и блока, следующим образом:

- 9 раундов, если размер ключа и блока составляет 128 бит;
- 11 раундов, если размер ключа и блока составляет 192 бита;
- 13 раундов, если размер ключа и блока составляет 256 бит.

Rijndael - это шифр линейного преобразования подстановки, не требующий сети Фейстеля. Он использует тройные сдержанные обратимые равномерные преобразования (слои). В частности, это:

- Линейное преобразование;
- Нелинейное преобразование;
- Преобразование ключа.

Еще до первого раунда выполняется простой уровень добавления ключа, что добавляет безопасности. После этого - раунды Nr-1, а затем финальный раунд. Преобразования образуют форму при запуске, но до завершения всего процесса.

Форму можно рассматривать как массив, структурированный с 4 строками, а номер столбца - длина блока, деленная на длину бит (например, деленная на 32). Ключ шифрования аналогичным образом представляет собой массив с 4 строками, но длина ключа делится на 32, чтобы указать количество столбцов. Блоки могут быть интерпретированы как одномерные массивы 4-байтовых векторов.

Точные преобразования происходят следующим образом: субтрансформация байтов нелинейна и работает на каждом из байтов формы независимо - обратимая таблица подстановок состоит из двух преобразований. Трансформация сдвига видит, что форма смещена по переменным смещениям. Значения смещения сдвига зависят от длины блока формы. Преобразование с

помощью функции, которая смешивает данные внутри каждого столбца формы, видит, что столбцы формы берут полиномиальные характеристики по значениям поля Галуа (28), умноженные на x4 + 1 (по модулю) с фиксированным многочленом. Наконец, происходит преобразование в форму с помощью уникального ключа, который применяется в каждом отдельном раунде, и функции XOR. Расписание ключей помогает ключу шифрования определять уникальные ключи посредством расширения ключа и выбор раунда.

В целом, структура Rijndael демонстрирует высокую степень модульной конструкции, которая должна сделать модификацию для противодействия любой атаке, даже с учетом будущих технологий, намного проще, чем с использованием устаревших алгоритмов.

1.3.2 Асимметричное шифрование

Асимметричная криптография, также известная как криптография с открытым ключом, использует общедоступные и закрытые ключи для шифрования и дешифрования данных. Ключи - это просто большие числа, которые были соединены вместе, но не идентичны (асимметричны). Один ключ из пары может быть известен всем, он называется открытым ключом. Другой ключ из пары хранится в секрете, он называется закрытым ключом. Любой из ключей может использоваться для шифрования сообщения; для дешифрования используется противоположный ключ от того, который используется для шифрования сообщения.

Многие протоколы, такие как SSH, OpenPGP, S/MIME и SSL/TLS, основаны на асимметричной криптографии для функций шифрования и цифровой подписи. Он также используется в программах, таких как браузеры, которым необходимо установить безопасное соединение по небезопасной сети, например, в Интернете, или для проверки цифровой подписи. Надёжность шифрования напрямую зависит от размера ключа, а удвоение длины ключа обеспечивает экспоненциальное увеличение прочности, хотя и снижает

производительность. По мере увеличения вычислительной мощности и выявления более эффективных алгоритмов факторинга увеличивается и способность увеличивать и увеличивать число также возрастает.

При асимметричном шифровании для обеспечения конфиденциальности, целостности, аутентичности и отказоустойчивости, пользователи и системы должны быть уверены, что открытый ключ является подлинным, что он принадлежит заявленному лицу или субъекту и что он не был подделан или заменен злоумышленниками. Не существует идеального решения проблемы аутентификации с открытым ключом. Наиболее распространенным подходом является инфраструктура открытых ключей (PKI), в которой доверенные сертификационные центры сертифицируют права собственности на пары ключей и сертификаты, но продукты шифрования, основанные на модели Pretty Good Privacy (включая OpenPGP), полагаются на децентрализованную модель аутентификации, называемой веб-службой доверия, которая опирается на индивидуальные одобрения связи между пользователем и открытым ключом.

Уитфилд Диффи и Мартин Хеллман, исследователи из Стэнфордского университета, впервые публично предложили асимметричное шифрование в своей статье 1977 года «Новые направления в криптографии». За несколько лет до Диффи и Хеллмана эта концепция была независимо и секретно предложена Джеймсом Эллисом, который работал в штаб-квартире правительственных коммуникаций (GCHQ), британской разведывательной и охранной организации. Асимметричный алгоритм, описанный в документе Диффи-Хеллмана, использует специальные числа, для создания ключей дешифрования.

RSA (Rivest-Shamir-Adleman), наиболее широко используемый асимметричный алгоритм, встроен в протокол SSL/TLS, который используется для обеспечения безопасности связи по компьютерной сети. RSA получает свою безопасность от вычислительной сложности факторизации больших целых чисел, которые являются произведением двух больших простых чисел. Умножение двух больших простых чисел легко, но сложность определения исходных чисел из суммарного факторинга - является основой безопасности

криптографии с открытым ключом. Время, затрачиваемое на фактор продукта двух достаточно больших простых чисел, считается слишком большим для основной части атакующих, за исключением национальных государственных субъектов, которые могут иметь доступ к достаточной вычислительной мощности. RSA-ключи обычно имеют длину 1024 или 2048 бит, но эксперты считают, что в ближайшем будущем могут быть взломаны 1024-битные ключи, поэтому правительство и индустрия переходят на минимальную длину ключа 2048 бит[7].

Эллиптическая кривая криптографии (ECC) завоевывает популярность у многих экспертов по безопасности в качестве альтернативы RSA для реализации криптографии с открытым ключом. ECC - это метод шифрования с открытым ключом, основанный на теории эллиптических кривых, который может создавать более быстрые, более мелкие и более эффективные криптографические ключи. ECC генерирует ключи через свойства уравнения эллиптической кривой. Чтобы взломать ECC, нужно вычислить дискретный логарифм эллиптической кривой, и оказывается, что это значительно более сложная задача, чем факторинг. В результате размеры ключей ECC могут быть значительно меньше, чем требуемые RSA, но обеспечивают эквивалентную безопасность с меньшей вычислительной мощностью и потреблением ресурсов батареи, что делает его более подходящим для мобильных приложений, чем RSA.

Цифровые подписи основаны на асимметричной криптографии и могут предоставлять заверения в отношении происхождения, идентификации и статуса электронного документа, транзакции или сообщения, а также подтверждения информированного согласия подписавшего. Для создания цифровой подписи программное обеспечение подписи (например, программа электронной почты) создает односторонний хэш электронных данных, которые должны быть подписаны. Закрытый ключ пользователя используется для шифрования хэша, возвращая значение, уникальное для хешированных данных. Зашифрованный хэш наряду с другой информацией, такой как алгоритм

хеширования, формирует цифровую подпись. Любое изменение данных даже в одном бите приводит к другому значению хэш-функции. Этот атрибут позволяет другим проверять целостность данных, используя открытый ключ подписывающего лица для дешифрования хэша. Если дешифрованный хэш соответствует второму вычисленному хэшу тех же данных, он доказывает, что данные не изменились с момента его подписания. Если эти два хэша не совпадают, данные либо каким-то образом подделаны (что указывает на отказ целостности), либо подпись была создана с помощью закрытого ключа, который не соответствует открытому ключу, представленному подписывающим лицом (с указанием отказа аутентификации).

Цифровая подпись также мешает подписавшей стороне отказаться от того, что она что-то подписала (свойство отказоустойчивости). Если подписавшая сторона отрицает действительную цифровую подпись, их секретный ключ либо был скомпрометирован, либо они лгут. Во многих странах, цифровые подписи имеют одинаковую юридическую силу с более традиционными формами подписей[12].

1.3.2.1 Протокол Диффи – Хеллмана

Первая публикация данного алгоритма появилась в семидесятых годах XX века в статье Уитфилда Диффи и Мартина Хеллмана, в которой вводились основные понятия криптографии с открытым ключом. Алгоритм Диффи - Хеллмана не применяется для шифрования данных или формирования электронной подписи. Его предназначение – в распределении ключей. Он позволяет двум или более пользователям обменяться без посредников ключом, который может быть использован для симметричного шифрования. Это была первая криптосистема, которая позволяла защищать информацию без необходимости использования секретных ключей, передаваемых по защищенным каналам. Схема открытого распределения ключей, предложенная Диффи и Хеллманом, произвела настоящую революцию в шифровании, так как

решала основную проблему классической криптографии – проблему распределения ключей.

Алгоритм основан на трудности вычисления дискретных логарифмов. В этом алгоритме, как и во многих других алгоритмах с открытым ключом, вычисления выполняются по модулю некоторого большого простого числа P. Сначала специальным образом подбирается некоторое натуральное число A, меньшее P. Если нужно зашифровать значение X, то вычисляется

$$Y = A^X \bmod P. \quad (1.1)$$

При этом, имея X, вычислить Y легко. Обратная задача вычисления X из Y является достаточно сложной. Экспонента X и называется дискретным логарифмом Y. Таким образом, зная о сложности вычисления дискретного логарифма, число Y можно открыто передавать даже по незащищенному каналу связи, так как при большом модуле P исходное значение X подобрать будет практически невозможно. Алгоритм Диффи – Хеллмана для формирования ключа основан на этом математическом факторе.

Пусть два пользователя, пользователь 1 и пользователь 2, желают сформировать общий ключ для алгоритма симметричного шифрования. Вначале они должны выбрать большое простое число P и некоторое специальное число A, которое $1 < A < P-1$, такое, что все числа из интервала [1, 2, ..., P-1] могут быть представлены как различные степени A mod P. Всем абонентам системы эти числа должны быть известны и могут выбираться открыто. Это будут общие параметры[4].

Затем пользователь 1 выбирает число X_1, такое, что $X_1<P$, которое нужно формировать с помощью генератора случайных чисел. Это будет закрытый ключ первого пользователя, и он должен храниться в секрете. На основе закрытого ключа пользователь 1 вычисляет число:

$$Y_1 = A^{X_1} \bmod P \quad (1.2)$$

которое он отправляет второму пользователю. Второй абонент поступает аналогично, генерируя X_2 и вычисляя

$$Y_2 = A^{X_2} \bmod P \quad (1.3)$$

Этот результат отправляется первому пользователю.

После этого у пользователей есть следующая информация:

Таблица 1 – Параметры ключей шифрования

	Общие параметры	Открытый ключ	Закрытый ключ
1й пользователь	P, A	Y_1	X_1
2й пользователь		Y_2	X_2

Из чисел Y_1 и Y_2, а также из личных закрытых ключей каждый пользователь может сгенерировать общий секретный ключ Z для сеанса симметричного шифрования. Первый пользователь:

$$Z = (Y_2)^{X_1} \bmod P \quad (1.4)$$

Никто, кроме первого пользователя, не может этого сделать, так как число X_1 секретно. Второй пользователь может получить такое же число Z, используя свои закрытый ключ и открытый ключ пользователя 1:

$$Z = (Y_1)^{X_2} \bmod P \quad (1.5)$$

Если весь протокол формирования общего секретного ключа выполнен верно, значения Z у одного и второго абонента должны получиться одинаковыми. Причем, что самое важное, злоумышленник, не зная секретных чисел X_1 и X_2, не сможет вычислить Z. Не зная X_1 и X_2, он может попытаться вычислить Z, используя только передаваемые открыто значения P, A, Y_1 и Y_2.

Безопасность формирования общего ключа в алгоритме Диффи - Хеллмана исходит из того факта, что, хотя относительно легко высчитать экспоненты по модулю простого числа, очень трудно вычислить дискретные логарифмы. Задача считается неразрешимой для больших простых чисел размером сотни и тысячи бит, так как требует колоссальных затрат вычислительных ресурсов.

Первый и второй пользователи могут использовать число Z в качестве секретного ключа как для шифрования, так и для расшифрования данных.

Таким же образом любая пара абонентов может вычислить секретный ключ, известный только им.

1.3.2.2 Алгоритм шифрования RSA

RSA – криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел. RSA является первым алгоритмом шифрования с открытым ключом. Название системы происходит от первых букв фамилий ее авторов – Рональд Ривест, Ади Шамир и Леонард Адлеман – трех ученых из Массачусетского технологического института.

После изучения опубликованной в 1976 году статьи Уитфилда Диффи и Мартина Хеллмана «Новые направления в криптографии», которая заложила основы криптографии с открытым ключом, Ривест, Шамир и Адлеман приступили к поискам математической функции, которая позволяла бы реализовать модель системы, описанной в статье. После работы над более чем 40 возможными вариантами, ученым удалось обнаружить алгоритм, основывающийся на том, насколько легко находить большие простые числа и насколько сложно раскладывать на множители произведение двух больших простых чисел.

Для шифрования используется простая операция возведения в степень по модулю N. Для расшифрования необходимо вычислить функцию Эйлера от числа N, для этого необходимо знать разложение числа N на простые множители (в этом состоит задача факторизации). В криптографической системе RSA открытый и закрытый ключи состоят из пары целых чисел. Закрытый ключ хранится в секрете, а открытый сообщается другому участнику, или где-то публикуется[15].

Система базируется на следующих фактах:

- При известных числах b и d вычисление числа a из сравнения

$$a \equiv b^d \ mod \ n. \tag{2.1}$$

По составному модулю n – это простая задача;

- Вычисление неизвестного числа b при известных числах d и a из сравнения по составному модулю n

$$a \equiv b^d \ mod \ n \tag{2.2}$$

является трудной задачей;

- Если известно, что p и q простые числа и n = pq, то вычислить n легко, а найти разложение n на простые множители трудно;
- Если известно разложение n = pq на простые множители, то задача вычисления числа b из уравнения

$$A \equiv b^d \ mod \ n \tag{2.3}$$

выполнима.

Теоретической основой криптосистемы RSA является теорема Эйлера: для любых натуральных и взаимно простых чисел n и a справедливо равенство

$$a^{\varphi(n)} \equiv 1 \ mod \ n. \tag{2.4}$$

Здесь $\varphi(n)$ есть функция Эйлера – количество взаимно простых с n натуральных чисел от 1 до n.

Из теории чисел известно, что если p и q простые числа, а n = pq, то

$$\varphi \ n \ = \ p-1 \ \ q-1 \ . \tag{2.5}$$

Кроме того, из теоремы Эйлера следует, что если некоторое число e взаимно просто с $\varphi(n)$, то уравнение

$$de \equiv 1 \ mod \ \varphi \ n \ , \tag{2.6}$$

или иначе

$$de = k \ \varphi \ n \ + 1, \tag{2.7}$$

однозначно разрешается относительно d. Решение легко определяется расширенным алгоритмом Евклида.

Итак, если известно, что

$$de \equiv 1 \, mod \, \varphi \ n \ , \tag{2.8}$$

а x – передаваемая информация, то справедливо равенство

$$(x^e)^d \, mod \, n \equiv \ \ x^{k\varphi \ n \ +1} \ \ mod \, n \ \equiv \ \ x^{\varphi \ n \ \ \ k} \, x \, mod \, n \equiv x. \tag{2.9}$$

Фактически, последнее соотношение является основой для формулировки системы RSA.

1.3.2.3 Принцип работы системы RSA

Сначала происходит генерация пары ключей – открытого и закрытого. Генерация осуществляется следующим способом:

1. Выбираются два простых больших числа p и q, при этом они не равны.
2. Вычисляется модуль числа:

$$N = p * q. \tag{3.1}$$

3. Вычисляется значение функции Эйлера от модуля числа N:

$$\varphi \ N \ = \ p-1 \ * \ q-1 \ . \tag{3.2}$$

4. Выбирается некоторое число e – открытая экспонента – которое лежит в интервале 1 < q < $\varphi \ N$ и является взаимно простым со значением функции $\varphi \ N$.
5. Вычисляется число d – секретная экспонента. Причем оно является мультипликативно обратным к числу e по модулю $\varphi \ N$.

$$d * e = 1 \ mod \varphi \ N \ \ . \tag{3.3}$$

В результате получается пара ключей: *(e, N)* – открытый ключ и *(d, N)* – закрытый ключ.

Пользователь A и пользователь B обмениваются сообщениями в интернете. Чтобы поддерживать переписку в секрете, они используют шифрование. Пользователь B заранее сгенерировал пару ключей, а затем передал открытый ключ пользователю A, который отправляет зашифрованное сообщение[17].

Шифрование: Пользователь A шифрует сообщение m при помощи открытого ключа второго пользователя *(e, N)* и отправляет его:

$$c = e\ m\ = m^e\ mod(N). \quad (3.4)$$

Расшифрование: Приняв зашифрованное сообщение, пользователь B расшифровывает его, используя закрытый ключ *(d, N)*:

$$m = d\ c\ = c^d\ mod(N). \quad (3.5)$$

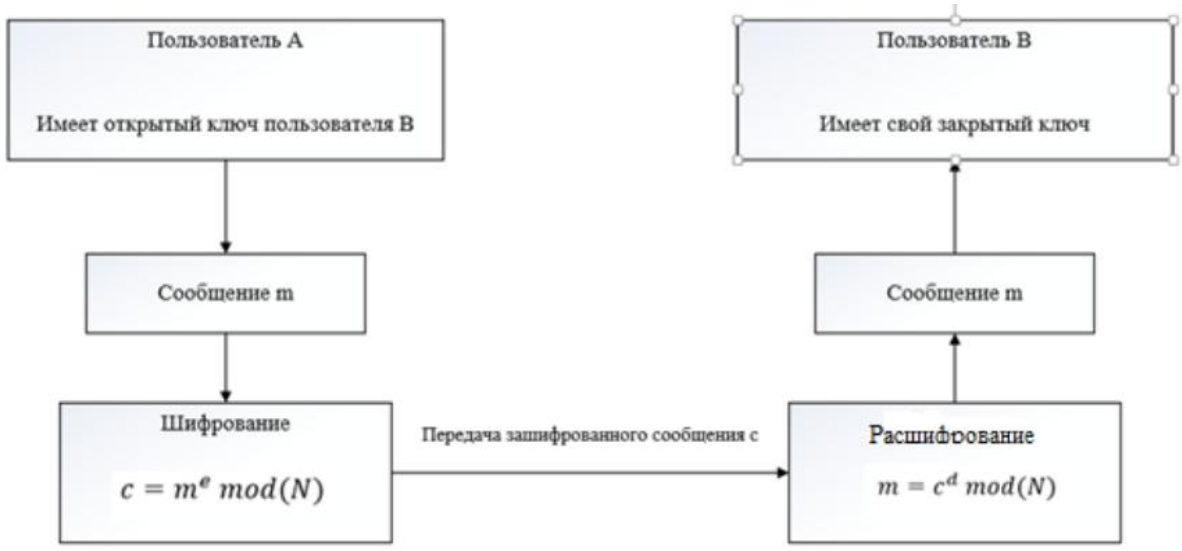

Рисунок 1.1– Принцип работы RSA

1.3.2.4 Пример шифрования и расшифрования RSA

Нужно зашифровать сообщение «RSA». Обозначим каждую букву их порядковыми номерами в английском алфавите. R – 18, S – 19, A – 1. Далее следуем алгоритму:

1. Выбираем простые числа (для простоты вычислений возьмем небольшие): $p = 3$, $q = 11$.

2. Вычисляем модуль N:

$$N = p * q = 3 * 11 = 33. \quad (4.1)$$

3. Находим функцию Эйлера от модуля числа N:

$$\varphi\ N\ =\ 3-1\ *\ 11-1\ = 2-10 = 20. \quad (4.2)$$

4. Выбираем открытую экспоненту: e = 7.

5. Вычисляем открытую экспоненту:

$$d * 7 = 1\ mod(20), d\ =\ 3. \quad (4.3)$$

Получившимся открытым ключом (7,33) шифруем каждую букву исходного сообщения:

$$\begin{aligned} c1 &= 18^7\ \ mod\ 33 = 6; \\ c2 &= 19^7\ \ mod\ 33 = 13; \\ c3 &= 1^7\ \ mod\ 33 = 1. \end{aligned} \quad (4.4)$$

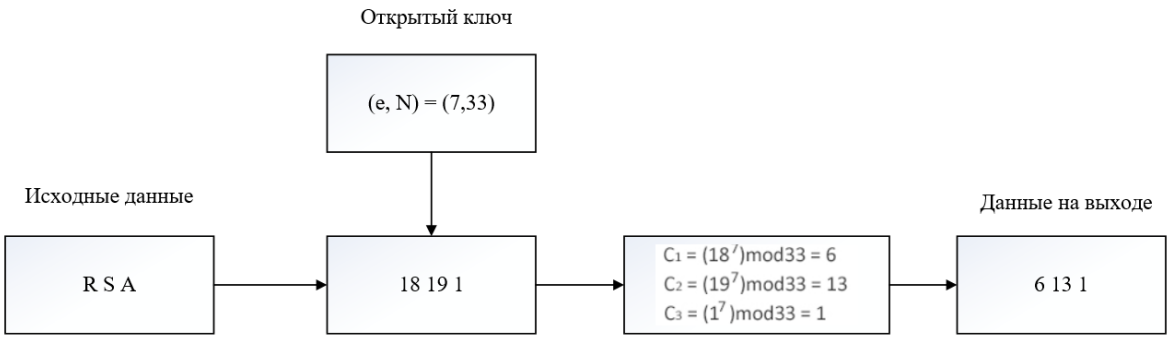

Рисунок 1.2 – Пример шифрования RSA

Чтобы расшифровать поученное сообщение, используем закрытый ключ (3, 33):

$$\begin{aligned} m1 &= 6^3\ \ mod\ 33 = 18; \\ m2 &= 13^3\ \ mod\ 33 = 19; \\ m3 &= 1^3\ \ mod\ 33 = 1. \end{aligned} \quad (4.5)$$

Получилось исходное сообщение.

1.4 Формирование требований к новой технологии

Используя блокчейн платформу Ethereum, мессенджеры могут стать максимально безопасными в эксплуатации. Взлом секретного ключа Ethereum не представляется возможным, даже с учетом современных технологий, так как на его подбор уйдет колоссальное количество времени.

Децентрализованный подход хранения данных в блокчейне гарантирует, невмешательство третьих лиц, чего не могут обеспечить централизованные подходы резервного копирования информации – облачные хранилища, серверы. При этом нет необходимости в реализации двух видов чатов – облачных и зашифрованных end-to-end алгоритмом. Все сообщения шифруются с помощью одного вида шифрования, а доступ к ним можно получить с любого устройства в реальном времени. Соответственно можно восстановить все переписки, фотографии, документы и прочие файлы, полученные и отправленные ранее.

В качестве алгоритма шифрования выбран RSA. Благодаря использованию открытого и закрытого ключей, криптосисетема является наиболее надежной и криптостойкой среди рассмотренных.

ГЛАВА 2. РАЗРАБОТКА СИСТЕМЫ БЕЗОПАСНОЙ ПЕРЕДАЧИ ДАННЫХ В СЕТИ ИНТЕРНЕТ

2.1 Блокчейн технология Ethereum

Сегодня наши личные данные, пароли и финансовая информация в основном хранятся в облаках и серверах, принадлежащих таким компаниям, как Amazon, Facebook, Apple или Google. Такой способ хранения данных имеет ряд удобств, так как эти компании развертывают команды специалистов, которые помогают хранить и защищать эти данные, и устранять затраты, связанные с хостингом и временем безотказной работы.

Но несмотря на все удобства, существует большой минус – это уязвимость. Злоумышленники могут получить нежелательный доступ к вашим файлам без вашего ведома, атакуя или влияя на стороннюю службу, то есть они могут украсть, раскрыть или изменить важную информацию.

Брайан Бехлендорф, создатель веб-сервера Apache даже назвал централизованное решение «первородным грехом» Интернета. Некоторые, такие как Бехлендорф, утверждают, что Интернет всегда был децентрализован, и раздробленное движение возникло вокруг использования новых инструментов, в том числе технологии блокчейн, для достижения этой цели.

Ethereum - одна из новейших технологий для присоединения к этому движении, цель которого - использовать блокчейн для замены третьих сторон в Интернете - тех, которые хранят данные, передают ипотечные кредиты, отслеживают сложные финансовые инструменты и т.д.

Ethereum может стать «мировым компьютером», который децентрализует, или даже демократизирует, существующую модель клиент-сервера. С Ethereum серверы и облака заменяются тысячами так называемых

«узлов», которыми управляют волонтеры со всего мира (таким образом формируя «мировой компьютер»).

Технология Ethereum позволит использовать эту же функциональность для людей во всем мире, что позволит им конкурировать за предоставление услуг поверх этой инфраструктуры.

Все современные приложения от банкинга до фитнеса и приложений для обмена сообщениями полагаются на компанию (или другую стороннюю службу) для хранения информации о кредитной карте, истории покупок и других персональных данных - где-то, как правило, на серверах, контролируемых третьими лицами.

Идея Ethereum заключается в том, что один объект больше не будет контролировать и внезапно блокировать приложение, работающее на технологии блокчейн. Только пользователь может вносить изменения, а не какой-либо другой объект.

Ethereum сочетает в себе контроль над тем, какой информацией люди обменивались в прошлом, с простотой в доступе к информации, к которой мы привыкли в эпоху цифровых технологий. Каждый раз, когда происходит сохранение изменения, добавление или удаление информации, каждый узел в сети делает изменения.

Ethereum – это адаптируемая и гибкая блокчейн платформа с открытым исходным кодом, которая позволяет создавать и использовать децентрализованные приложения, работающие на технологии блокчейн. Ethereum, вероятно, является самой развитой, но сложной системой, когда-либо созданной[18].

Несмотря на сложность протокола и механизмов безопасности, которые были разработаны на сегодняшний день, полный узел Ethereum состоит из трех основных частей:

- блокчейн;
- одноранговая сеть (peer-to-peer);
- виртуальная машина.

2.1.1 Блокчейн

Блокчейн - это не что иное, как цепочка блоков, привязанных друг к другу в определенном порядке. Причем, цепочка обладает таким свойством, что разбиение любого одного блока b также разрушит всех его преемников b + 1, b + 2, ... b + n.

В каждом блоке сохраняется ряд транзакций вместе с хэшем предыдущего блока и доказательством выполнения работы (proof-of-work) текущего блока. Доказательство выполнения работы - результат интенсивного вычисления, которое находит первое число (называемое nonce), которое вместе с содержимым блока возвращает определенный хэш. Такой хэш обычно начинается с числа 0, которое увеличивается в соответствии с параметром, называемым сложностью. Чем выше сложность, тем больше времени требуется, чтобы найти хэш, начиная с более высокого числа 0.

Поскольку каждый блок связан с предыдущем блоком (хэш предыдущего блока является частью содержимого текущего блока), очень сложно разбить цепочку. Разрыв цепи означает подделку специального блока в любом положении в цепочке, так что все последующие блоки остаются неизменными. Если бы, по абсурду, такой блок существовал, было бы чрезвычайно сложно убедить остальную сеть, что такая цепочка является легитимной.

Блокчейн Ethereum, по сути, является машиной состояний, работающей посредством транзакций. Определение машины состояний подразумевает, что этот механизм считывает входные данные и, основываясь на них, переходит в новое состояние.

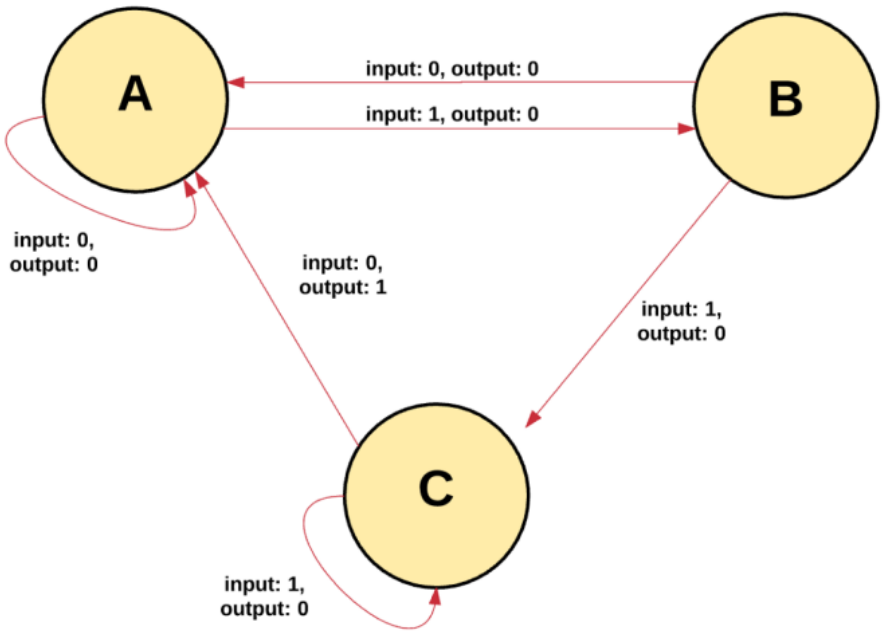

Рисунок 2.1 – Парадигма блокчейна Ethereum

В случае с машиной состояний Ethereum, начальной точкой является «состояние генезиса». Оно подобно чистому листу или пустому бланку до того, пока в сети не произойдут какие-либо транзакции. После выполнения транзакций состояние генезиса переходит в конечное состояние. В любой момент времени это конечное состояние является текущим состоянием Ethereum.

Рисунок 2.2 – Состояние блокчейна

Состояние Ethereum включает в себя огромное количество транзакций. Эти транзакции группируются в «блоки». Блок содержит группы транзакций, а каждый блок связан с предыдущим, тем самым образуя цепочку.

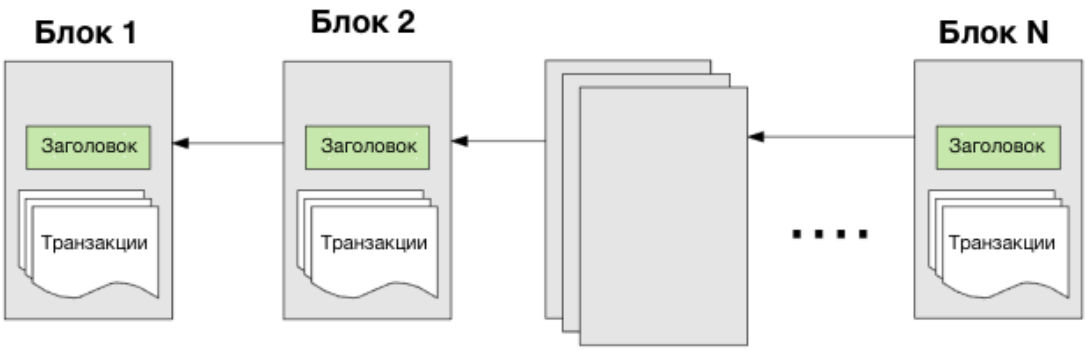

Рисунок 2.3 – Блоки транзакций

Чтобы вызвать переход сети из одного состояния в другое, транзакция должна быть признана действительной, для чего она должна пройти через процесс валидации (проверки и утверждения), известный как майнинг. Майнингом - это процесс, в котором группа узлов сети (компьютеров) расходует свои вычислительные ресурсы на генерацию блока действительных транзакций.

Любой вычислительный узел сети (их также называют «нодами»), объявляющий себя в качестве майнера, может претендовать на создание и валидацию блока транзакций. Многие ноды со всего мира одновременно пытаются создавать и валидировать блоки. Каждый майнер при записи блока в блокчейн предоставляет математическое «доказательство», которое действует как гарантия: если доказательство существует, то блок должен быть действительным.

Для того чтобы добавить блок к основному блокчейну, майнер должен подтвердить его раньше всех его конкурентов. Процесс проверки каждого блока путём предоставления нодами математического доказательства называется доказательством выполнения работы (proof-of-work)[20].

Так как блокчейн одноэлементный механизм записи транзакций с совместно используемым состоянием, то корректное текущее состояние – это единственная глобальная истина, которая должна приниматься всеми. Наличие нескольких состояний (или цепочек) разрушило бы всю систему, так как в таком случае было бы невозможно договориться о том, какое состояние сети должно считаться истинным.

Каждый раз, когда формируется несколько вариантов цепочек, образуется разветвление (fork). Обычно разветвлений стараются избегать, так как они разрушают систему и ставят пользователей перед выбором, какой цепочке они должны доверять.

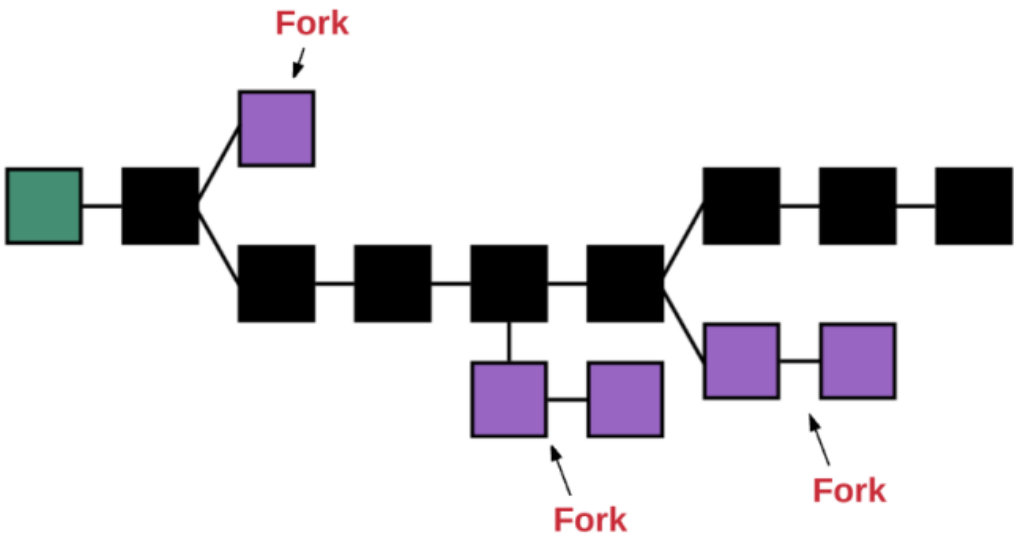

Рисунок 2.4 – Разветвления

Для того чтобы определить, какой путь наиболее действителен и предотвратить образование нескольких цепочек, в Ethereum используется механизм, известный как «протокол GHOST».

Согласно протоколу GHOST, мы должны выбирать тот путь, на котором было совершено самое большое количество вычислений. Один из способов определить этот путь – по номеру последнего блока, который представляет собой общее количество блоков на текущем пути, исключая генезисный. Чем больше номер блока, тем длиннее проделанный путь и тем выше количество усилий, затраченных нодами для того, чтобы достичь этой точки. Используя такое рассуждение, можно договориться о действительной версии текущего состояния.

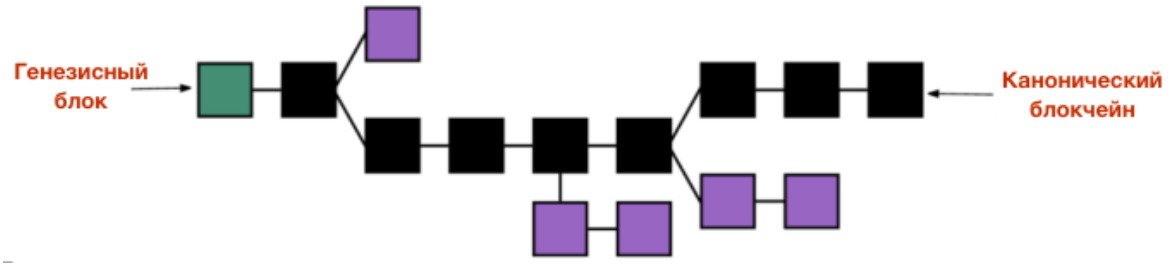

Рисунок 2.5 – Каноническое состояние

2.1.2 Смарт-контракты

Смарт-контракт или «умный контракт» – это компьютерный алгоритм, который контролирует выполнение обязательств сторон в процессе обмена активами в технологии блокчейн. Это договоры, за исполнением которых следит компьютер.

Смарт-контакты - это закодированные логики, которые изменяют цифровые активы при срабатывании определенных событий. Соответственно, это своего рода программное обеспечение, которое использует условие «если это так, то затем» утверждения, где «если» являются основными требованиями к запуску «затем». Когда умный контракт работает на блокчейне, он автоматически запускается, когда выполняются все необходимые условия. Вся структура основана и проверена множеством подключенных компьютеров. Это гарантирует, что умные контракты:

- безопасные;
- открытые;
- заслуживают доверия;
- почти лишены любых возможных ошибок.

Однако это также означает, что, когда процесс запускается, все компьютеры, задействованные в этой огромной сети, должны выполнять одну и ту же операцию, что объясняет дороговизну этого процесса.

Большинство блокчейн-платформ в основном ограничены. Но Ethereum отличается. Смарт-контракты позволяют их разработчикам создавать и

запускать любые операции, которые они желают, создавая тысячи универсальных приложений[20].

Виртуальная машина Ethereum (EVM) – полное по Тьюрингу программное обеспечение, способное запускать любое приложение. Новшество Ethereum заключается в том, что любая выполняемая внутри него операция одновременно выполняется каждым отдельным узлом во всей сети. Эти высокофункциональные операционные базы контрактов Ethereum объясняют его способность создавать блоксхемы намного быстрее, проще и продуктивнее.

Особенностью работы смарт-контрактов на платформе Ethereum является то, что все они имеют свои собственные адреса в блокчейне. То есть, соответствующий код не вставляется в каждый контракт. Вместо этого узел запускает определенную транзакцию, которая создает и прикрепляет уникальный адрес к контракту. После этой первичной транзакции контракт превращается в неотделимую единицу блокчейна, адрес которой никогда не меняется. Затем умный контракт будет действовать без остановок до успешного завершения операции.

2.2 Реализация системы

2.2.1 Создание ключей и их использование для транзакций

Создание идентификатора Ethereum и использование его для отправки в блокчейн. Идентификатор - это объект с приватным ключом и соответствующим открытым ключом и его адресом. Чтобы создать новый идентификатор, вызывается функция createIdentity, которая возвращает ее.

```
const EthCrypto = require('eth-crypto');

const identity = EthCrypto.createIdentity();

console.dir(identity);
/* > {
    address: '0x3f243FdacE01Cfd9719f7359c94BA11361f32471',
    privateKey:
'0x107be946709e41b7895eea9f2dacf998a0a9124acbb786f0fd1a826101581a07',
    publicKey: 'bf1cc3154424dc22191941d9f4f50b063a2b663a2337e5548abea633c1d06ece...'
} */
```

Рисунок 3.1 – Создание нового идентификатора

Компоненты идентификатора:

- Приватный ключ (privateKey), который никогда не должен раскрываться никому. Его можно использовать для подписания и расшифровки сообщений и для создания его открытого ключа (publicKey).
- Открытый ключ раскрывается всякий раз, когда что-то подписывается с помощью privateKey. Также принято отправлять publicKey другим людям, чтобы они могли зашифровать данные с ним, которые затем могут быть расшифрованы только с помощью правильного privateKey. Существует два способа представления publicKey сжатым и несжатым. EthCrypto всегда создает несжатый ключ, который начинается с 0x04. Сжатые ключи начинаются с 0x03 или 0x02. Чтобы представить ключ, мы отделяем начало 04 от него и внутренне добавляем его при выполнении криптографических вызовов.
- Адрес вычисляется из последних 20 байтов keccak-256 хэширования publicKey. Он используется для представления идентификатора. Не существует способа вычислить publicKey с адреса. Это означает, что всякий раз, когда нужно зашифровать данные для кого-то, мы сначала должны получить publicKey. Существует два способа представления адреса. Обычный адрес имеет строчный регистр и представляет собой только 20 байтов хэша. Формат контрольной суммы содержит прописные буквы, которые предназначены для обнаружения ошибок, когда адрес вводится вручную[9].

Транзакция Ethereum в основном представляет собой json-объект с определенными значениями.

```
const rawTransaction = {
        from: identity.address, // адрес отправителя
        to: '0x86Fa049857E0209aa7D9e616F7eb3b3B78ECfdb0', // адрес получателя
        value: 1000000000000000000, // количество wei, которое мы хотим отправить
(=1 эфир)
        nonce: 0, // инкрементное tx-число. добавляет +1 для каждой транзакции
        gasPrice: 5000000000,
        gasLimit: 21000 // нормальный gasLimit для транзакций без кода
    };
```

Рисунок 3.2 – Создание транзакции

Прежде чем транзакция может быть отправлена на узел, она должна быть подписана с помощью privateKey и переведена в шестнадцатеричную строку.

```
const serializedTx = EthCrypto.signTransaction(
        rawTransaction,
        identity.privateKey
    );
    console.log(serializedTx);
    // > 'f86c808504a817c80082ea609463dcee1fd1d814858acd4172bb20e1...'
```

Рисунок 3.3 – Подпись с privateKey

Теперь строка транзакции может быть отправлена в блокчейн. Для целей тестирования создается локальная тестовая цепочка и проверяется там. Чтобы создать локальную тестовую сеть, используется ganache-cli и подключается к экземпляру web3, чтобы взаимодействовать с ним.

```
const Web3 = require('web3');
const ganache = require('ganache-cli');

// создание web3-экземпляра
const web3 = new Web3();

// создание провайдера ganache
const ganacheProvider = ganache.provider({
// устанавливка баланса идентификатора на 10 эфиров
accounts: [{
secretKey: identity.privateKey,
balance: web3.utils.toWei('10', 'ether')
}]
});

// установить ganache на web3 в качестве провайдера
web3.setProvider(ganacheProvider);
```

Рисунок 3.4 – Создание локальной тестовой сети

sendSignedTransaction вызывается, чтобы отправить подписанную транзакцию в тестовую цепочку. Ganache немедленно выполнит транзакцию, и receipt вернется обратно. Чтобы убедиться, что транзакция сработала, проверим баланс адреса получателей.

```
const receipt = await web3.eth.sendSignedTransaction(serializedTx);
const balance = await
web3.eth.getBalance('0x86Fa049857E0209aa7D9e616F7eb3b3B78ECfdb0');
console.log(balance);
// > '100000000000000000'
```

Рисунок 3.5 – Отправка транзакции

2.2.2 Подпись и проверка данных с помощью Solidity

Подпишем данные с помощью JavaScript и проверим подпись в смарт-контракте Solidity.

Сначала создаем два идентификатора, создателя и получателя. Затем запускаем локальную тестовую сеть. В тестовой сети даем creatorIdentity баланс 10 эфира. Также предоставляем один эфир получателю, поэтому у нас достаточно газа для отправки транзакций.

Прежде чем мы сможем заключить контракт с локальным блокчейном, нужно скомпилировать Solidity-код в байт-код, используя JavaScript-версию solc-компилятора:

```
const path = require('path');
const SolidityCli = require('solidity-cli');
const contractPath = path.join(__dirname, '../../contracts/DonationBag.sol');
const compiled = await SolidityCli.compileFile(contractPath);
const compiledDonationBag = compiled[':DonationBag'];

const createCode = EthCrypto.txDataByCompiled(
    JSON.parse(compiledDonationBag.interface), // abi
    compiledDonationBag.bytecode, // байткод
    [creatorIdentity.address] // конструктор-аргумент
);

console.dir(compiledDonationBag);
```

Рисунок 4.1 – Компиляция в байткод

Теперь, когда есть байт-код контракта, мы можем отправить транзакцию для создания нового экземпляра в локальной тестовой цепочке.

```
// отпрака в локальную цепочку
const receipt = await web3.eth.sendSignedTransaction(serializedTx);
const contractAddress = receipt.contractAddress;

console.log(contractAddress);
```

Рисунок 4.2 – Отправка транзакции

Теперь контракт находится в блокчейне. Чтобы проверить, правильно ли он развернут, можно вызвать функцию. Прежде чем мы сможем подписывать пожертвования, нужно отправить какое-то значение в контракт.

При подписании сообщения, мы не подписываем адрес получателя напрямую, а только хэш, который составлял некоторые обобщенные данные:

- Prefix: чтобы гарантировать, что создатель не может быть обманут в случайного подписания действительной транзакции Ethereum, мы подставляем подписанные данные с чем-то уникальным для нашей системы.
- contractAddress: возможно, у создателя имеется более одного экземпляра контракта, загруженного в блокчейн. В этом случае подписи могут быть воспроизведены в других экземплярах. В качестве предотвращения этой атаки мы также добавляем адрес контрактов к подписанному хэшу.
- receiverAddress: подписывая этот адрес, создатель доказывает, что данный адрес должен получить пожертвование.

Теперь у приемника есть подпись от создателя, которую он может отправить в контракт, чтобы потребовать пожертвования[5].

Если все пошло правильно, у получателя должно быть больше эфира.

```
const receiverBalance = await web3.eth.getBalance(recieverIdentity.address);
console.dir(receiverBalance);
// '1999802840000000000'
```

Рисунок 4.3 – Проверка баланса

2.2.3 Шифрование и подпись сообщений

С помощью ключей Ethereum можно не только взаимодействовать с блокчейном, но и использовать их для безопасного обмена сообщениями по взаимным ненадежным каналам. Идентификаторы Ethereum будут использоваться для отправки сообщений.

Сначала мы создаем два идентификатора – пользователь A и пользователь B. Пользователь A отправляет сообщение «Hello, B». Прежде чем отправить, нужно обеспечить, чтобы только абонент B мог прочитать сообщение, и чтобы он был уверен, что сообщение пришло действительно от первого пользователя. Чтобы это сделать, сначала нужно подписать сообщение с помощью секретного ключа абонента A, а затем зашифровать сообщение и подписать с помощью открытого ключа пользователя B.

```
const signature = EthCrypto.sign(
    A.privateKey,
    EthCrypto.hash.keccak256(secretMessage)
);
const payload = {
    message: secretMessage,
    signature
};
const encrypted = await EthCrypto.encryptWithPublicKey(
    B.publicKey, privateKey
    JSON.stringify(payload)
);

const encryptedString = EthCrypto.cipher.stringify(encrypted);
```

Рисунок 5.1 – Шифрование и подпись сообщения

Когда получатель принимает сообщение, он начинает с расшифровки его своим секретным ключом, а затем проверяет подпись.

```
const encryptedObject = EthCrypto.cipher.parse(encryptedString);

const decrypted = await EthCrypto.decryptWithPrivateKey(
    B.privateKey,
    encryptedObject
);
const decryptedPayload = JSON.parse(decrypted);

const senderAddress = EthCrypto.recover(
    decryptedPayload.signature,
    EthCrypto.hash.keccak256(payload.message)
);

console.log(
    'Сообщение получено от ' +
    senderAddress +
    ': ' +
    decryptedPayload.message
);
```

Рисунок 5.2 – Расшифровка

Теперь, когда пользователь B получил сообщение, он также может ответить. Чтобы сделать это, он должен восстановить открытый ключ пользователя A.

ГЛАВА 3. РАСЧЕТ ЗАТРАТ

3.1 Gas в сети Ethereum

Gas – это внутренняя валюта сети Ethereum, которая используется для заключения сделки и контрактов. Gas является единицей, которая измеряет объем вычислительных усилий, которые потребуются для выполнения определенных операций. Оплата за вычисления происходит всегда, независимо от того, прошла транзакция или нет. Даже в случаях, когда транзакция отклоняется, ноды должны подтвердить и выполнить вычисления. Поэтому оплата за работу нод происходит независимо от успешности транзакции.

Цена на Gas за вычисления или контракт настроена для работы с Тьюринг полным происхождением Ethereum и его EVM. Идея состоит в том, чтобы ограничить бесконечные циклы. Так, например, 1 Gas может выполнять строку кода или некоторую команду. Если в учетной записи недостаточно эфира, чтобы выполнить транзакцию или отправить сообщение, то считается недействительным. Задумка заключается в том, чтобы прекратить атаки на отказ в обслуживании из бесконечных циклов, повысить эффективность кода и заставить злоумышленника платить за ресурсы, которые они используют, от пропускной способности до вычислений ЦП и хранения данных[13].

Чем сложнее команды, которые нужно выполнить, тем больше количество Gas’а придется заплатить. Например, если некто А хочет отправить пользователю В 1 Ether - общая сумма в 1.00001 Ether должна быть оплачена пользователем А. Однако если А хочет заключить контракт с В в зависимости от будущей цены на Ether, будет больше строк исполняемого кода и больше потребления энергии, размещенного в распределенной сети Ether, и, следовательно, пользователь А должен будет заплатить больше, чем 1 Gas, для выполнения транзакции.

Некоторые вычислительные шаги стоят дороже, чем другие, либо потому, что они являются дорогостоящими вычислительными, либо потому, что они увеличивают объем данных, которые должны храниться в блокчейне.

3.2 Аппроксимация

Рассчитать лимит Gas'а можно по следующей формуле:

$$gasLimit = G_{transaction} + G_{txdatanonzero} * dataByteLength. \quad (5.1)$$

$G_{transaction}$ – это стандартная плата за каждую транзакцию, равную 21000 Gas.

$G_{txdatanonzero}$ – плата за каждый ненулевой байт данных или кода для транзакции. Стоимость 68 Gas.

dataByteLength – размер данных в байтах.

В нашем случае, для того, чтобы определить рентабельность хранения данных в блокчейне Ethereum, требуется решить следующую задачу: какую цену Gas'а необходимо поставить для того, чтобы сообщение можно было считать доставленным в переделах определенного времени. То есть, нужно определить такой Gas price, при котором скорость подтверждения транзакции будет удовлетворять условие «не более N секунд». Воспользуемся графиком зависимости времени подтверждения транзакции от стоимости Gas'а.

Таблица 2 – Цена Gas'а и время подтверждения

Цена Gas'а	2	3	4	5	6	7	8	9	10
Время подтверждения (в секундах)	961,8	869,1	473,1	465,1	132,4	8,5	8	2	1,1

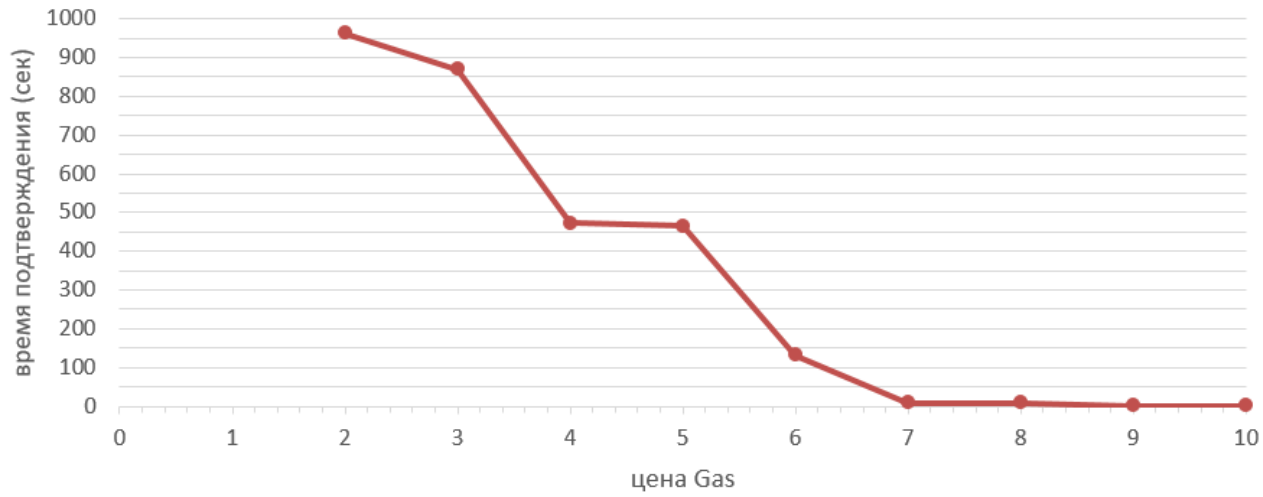

Рисунок 6.1 – Время подтверждения по цене Gas'а

Согласно графику, для того, чтобы сообщение было отправлено в течение 400 секунд, необходимо поставить стоимость Gas'а в 5,2. Наша цель заключается в том, чтобы обмен сообщениями был максимально дешев. Постоим кубическую регрессию, которая позволит определить оптимальные расходы.

Методом наименьших квадратов найдем кубическую функцию, с помощью которой проведем аппроксимацию.

Уравнение регрессии:

$$y = ax^3 + bx^2 + cx + d. \tag{5.2}$$

Коэффициенты a, b, c и d найдем из решения системы:

$$\begin{aligned} a \quad & x_i^3 + b \quad x_i^2 + c \quad x_i + nd = \quad y_i, \\ a \quad & x_i^4 + b \quad x_i^3 + c \quad x_i^2 + d \quad x_i = \quad x_i y_i; \\ a \quad & x_i^5 + b \quad x_i^4 + c \quad x_i^3 + d \quad x_i^2 = \quad x_i^2 y_i; \\ a \quad & x_i^6 + b \quad x_i^5 + c \quad x_i^4 + d \quad x_i^3 = \quad x_i^3 y_i. \end{aligned} \tag{5.3}$$

Опустим вычисления, искомая функция принимает вид:

$$y = 2{,}3364x^3 - 22{,}021x^2 - 146{,}16x + 1356{,}1. \tag{5.4}$$

Изобразим на графике функцию и проследим за стоимостью Gas'а в пределах 400 секунд.

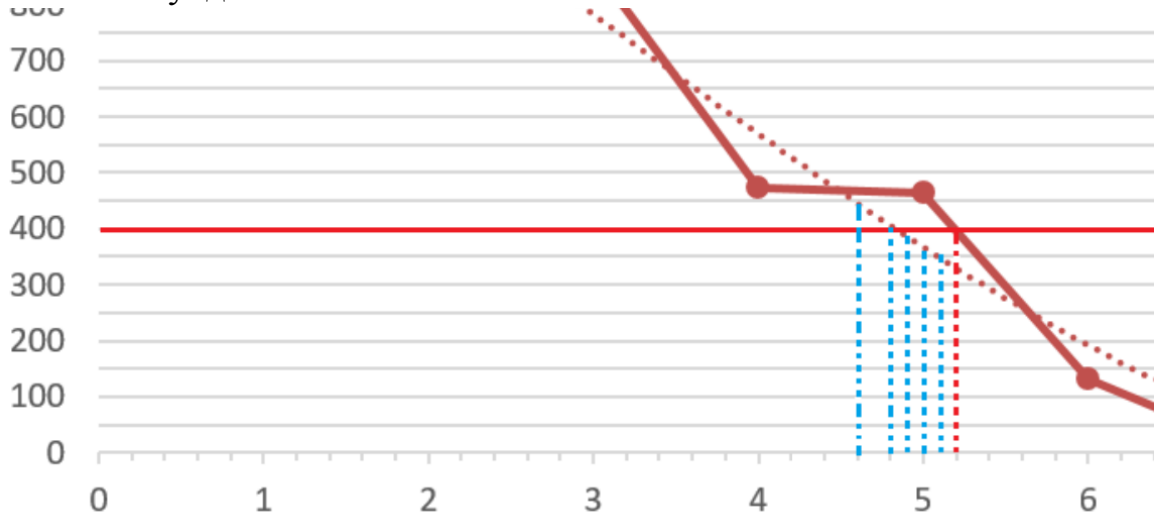

Рисунок 6.2 – Аппроксимация

Из графика видно, что для отправки сообщения можно затратить меньшее количество Gas’a – 5,1 вместо 5,2. Следовательно наблюдается экономия. Попробуем еще снизить стоимость транзакций.

Таблица 3 – Цена газа после аппроксимации

Цена Gas’s	5,1	5	4,9	4,6
Время подтверждения (в секундах)	350	360	380	450

Можем заметить, что при стоимости Gas’s в 4,6, время подтверждения переходит заданную границу в 400 секунд.

В конечном итоге, при всех преимуществах реализованной системы, для отправки одного и того же сообщения в других системах обмена сообщениями, мне не потратим денежных средств. В нашем случае, за сообщение придется заплатить, но при этом данные будут отправляться и храниться максимально безопасно, а доступ к ним можно получить с любого устройства, использую единую учетную запись.

ЗАКЛЮЧЕНИЕ

В ходе выполнения бакалаврской работы были рассмотрены различные алгоритмы шифрования, их теоретические аспекты. Особое внимание было уделено криптосистеме RSA, его математическим основам, продемонстрирован небольшой пример. Данный алгоритм был выбран для реализации системы безопасной передачи данных между пользователями в сети.

RSA является одним из самых надежных алгоритмов для шифрования с открытым и закрытым ключами и электронной подписью. Криптосистема используется в наиболее популярных продуктах, требующих высокого уровня безопасности, и протоколах, используемых сегодня, и может рассматриваться, как одна из основ для безопасного общения в сети Интернет.

Также была изучена платформа для создания децентрализованных приложений Ethereum, ее смарт-контракты, принцип их работы. Данная блокчейн технология является максимально безопасной для хранения данных, так как чтобы взломать секретный ключ Ethereum потребуются столетия. Благодаря этому, мессенджер основанный на этой технологии позволит избавиться от необходимости реализации двух видов чатов – секретные и облачных. Кроме этого, гарантируется невмешательство третьих лиц и злоумышленников в тайную переписку. Пользователи могут получить доступ к отправленным и полученным ранее файлам и сообщениям с любого устройства в реальном времени. Такая функция крайне полезна в случае утери или смене смартфона, или при использовании одной учетной записи на разных устройствах.

В ходе тестирования данной технологии, выяснилось, что хранение данных в блокчейне является нерентабельным из-за довольно высокой стоимости транзакций, но максимально надежным и безопасным.

Список используемой литературы

1. Интуит – Национальный Открытый институт. [Электронный ресурс]. - Режим доступа: https://www.intuit.ru/studies/courses/691/547/lecture/12377

2. Ишмухаметов, Ш.Т. Математические основы защиты информации: учеб. пособие / Ш.Т. Ишмухаметов, Р.Г. Рубцов — Казань: Казанский федер. Ун- т, 2012. — 138 с.

3. Как устроен AES [Электронный ресурс]// Хабрахабр – Режим доступа URL: https://habr.com/post/112733/

4. ОСНОВЫ ШИФРОВАНИЯ (ЧАСТЬ 1) - АЛГОРИТМ ДИФФИ-ХЕЛЛМАНА [Электронный ресурс] / Securitylab - 25.01.2016 - Режим доступа URL: https://www.securitylab.ru/analytics/478912.php

5. Пишем умный контракт на Solidity. [Электронный ресурс] / Хабрахабр – 07.10.2016 - Режим доступа URL: https://habr.com/post/312008/

6. Почему у Telegram не включено End-to-end шифрование по умолчанию [Электронный ресурс]// Medium - Режим доступа URL: https://medium.com/@tglive/telegram-end-to-end-e93554cb9e46

7. Пример алгоритма шифрования rsa [Электронный ресурс] // Infoprotect.net - Режим доступа URL: http://infoprotect.net/varia/algoritm_shifrovaniya_rsa_primer

8. Романьков, В.А. Введение в криптографию. Курс лекций / В.А. Романьков. — М.: ФОРУМ, 2012. — 240 с.

9. Руководство по Solidity [Электронный ресурс] / James Ray // GitHub - Режим доступа URL: https://github.com/ethereum/wiki/wiki/%D0%A0%D1%83%D0%BA%D0%BE%D0

%B2%D0%BE%D0%B4%D1%81%D1%82%D0%B2%D0%BE-%D0%BF%D0%BE-Solidity

10. Салий, В.Н. Криптографические методы и средства защиты информации: учеб. пособие / В.Н. Салий; Саратов: Саратовский гос. ун-т имени Н.Г. Чернышевского, 2012. — 41 с

11. Ященко, В.В. Введение в криптографию. / В.В. Ященко — 4-е изд., [доп.]. — М.: МЦНМО, 2012. — 348 с.

12. Asymmetric cryptography (public key cryptographt). [Электронный ресурс] /Margaret Rouse // SearchSecurity – 05.06.2015 - Режим доступа URL: https://searchsecurity.techtarget.com/definition/asymmetric-cryptography

13. Ethereum Yellow Paper [Электронный ресурс] / ethereum.github.io // Dr. Gavin Wood – 28.05.2018 - Режим доступа URL: https://ethereum.github.io/yellowpaper/paper.pdf

14. Everything you wanted to know about the next generation of public key crypto.[Электронный ресурс] / Nick Sullivan // Ars Technica – 25.10.2013 – Режим доступа: https://arstechnica.com/information-technology/2013/10/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/2/

15. RSA algorithm (Rivest-Shamir-Adleman). [Электронный ресурс] /Margaret Rouse // SearchSecurity – 15.05.2014 - Режим доступа URL: https://searchsecurity.techtarget.com/definition/RSA

16. Spy-funded privacy tools (like Signal and Tor) are not going to protect us from President Trump [Электронный ресурс] // Yasha Levine / Surveillance Valley – 09.12.2016 - Режим доступа URL: https://surveillancevalley.com/blog/government-backed-privacy-tools-are-not-going-to-protect-us-from-president-trump

17. Technology Box — портал посвященный вопросам информационной безопасности. [Электронный ресурс]. - Режим доступа: http://teh-box.ru/informationsecurity/algoritm-shifrovaniya-rsa-na-palcax.html

18. What is Ethereim? [Электронный ресурс] / Alyssa Hertig // CoinDesk – Режим доступа URL: https://www.coindesk.com/information/what-is-ethereum/

19. What is symmetric encryption.[Электронный ресурс] / David Bisson // Venafi Blog – 09.11.2017 – Режим доступа: https://www.venafi.com/blog/what-symmetric-encryption

20. White Paper [Электронный ресурс] / GitHub - Режим доступа URL: https://github.com/ethereum/wiki/wiki/White-Paper