МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра компьютерных технологий и систем

Шалаев
Павел Витальевич

ТЕХНИЧЕСКИЕ АСПЕКТЫ РАЗРАБОТКИ ДЕЦЕНТРАЛИЗОВАННОГО ПРИЛОЖЕНИЯ МЕССЕНДЖЕРА ДЛЯ ОБМЕНА ЗАШИФРОВАННЫМИ ДАННЫМИ

Дипломная работа

Научный руководитель:
профессор кафедры КТС,
доктор физико-математических
наук, профессор
Таранчук В.Б.

Допущена к защите

«___» _____ 20__ г.

Заведующий кафедрой
компьютерных технологий и систем
доктор педагогических наук,
кандидат физико-математических наук,
профессор В. В. Казачёнок

Минск, 2025

РЕФЕРАТ

Дипломная работа, 57 страниц, 4 рисунка, 25 источников.

Ключевые слова: ДЕЦЕНТРАЛИЗОВАННЫЙ МЕССЕНДЖЕР, ШИФРОВАНИЕ, P2P-СЕТИ, E2EE, КРИПТОГРАФИЯ, БЛОКЧЕЙН, УПРАВЛЕНИЕ КЛЮЧАМИ.

Объект исследования: криптографически защищённые децентрализованные приложения.

Цель работы: разработка и исследование прототипа децентрализованного мессенджера для безопасного обмена зашифрованными сообщениями без участия центрального сервера.
Методы исследования: анализ литературы, проектирование архитектурных решений, реализация и тестирование программных компонентов.

Результаты: разработан и протестирован прототип мессенджера с поддержкой end-to-end шифрования, P2P-архитектуры, генерации и валидации ключей, защиты от атак типа «человек посередине» и Sybil-атак. Проведены нагрузочные и функциональные тесты, предложены пути улучшения производительности и безопасности системы.

РЭФЕРАТ

Дыпломная праца, 57 старонак, 4 малюнкі, 25 крыніц.

Ключавыя словы: ДЭЦЭНТРАЛІЗАВАНЫ МЕСЭНДЖАР, ШЫФРАВАННЕ, P2P-СЕТКІ, E2EE, КРЫПТАГРАФІЯ, БЛОКЧЭЙН, КІРАВАННЕ КЛЮЧАМІ.

Аб’ект даследавання: крыптаграфічна абароненыя дэцэнтралізаваныя прыкладанні.

Мэта працы: распрацаваць і даследаваць прататып дэцэнтралізаванага месэнджара для бяспечнага абмену шыфраванымі паведамленнямі без выкарыстання цэнтральнага сервера.

Метадалогія: аналіз літаратуры, праектаванне архітэктуры, рэалізацыя і тэставанне праграмных модуляў.

Вынікі: рэалізаваны і пратэставаны прататып месэнджара з падтрымкай скразнога шыфравання, P2P-архітэктуры, генерацыі і праверкі ключоў, абароны ад атак «чалавек пасярэдзіне» і Sybil-атак. Праведзены тэсты прадукцыйнасці і надзейнасці, прапанаваны шляхі паляпшэння бяспекі і маштабуемасці сістэмы.

ABSTRACT

Thesis, 57 pages, 4 figures, 25 sources.

Keywords: DECENTRALIZED MESSENGER, ENCRYPTION, P2P NETWORKS, E2EE, CRYPTOGRAPHY, BLOCKCHAIN, KEY MANAGEMENT.

Object of study: cryptographically secure decentralized applications. Purpose of the study: to design and develop a prototype of a decentralized messenger enabling secure encrypted communication without reliance on a central server.

Research methods: literature review, architecture design, software implementation and testing.

Results: a working prototype was developed with end-to-end encryption support, peer-to-peer architecture, key generation and validation, protection against man-in-the-middle and Sybil attacks. Performance and security were evaluated, and directions for improving efficiency and resilience were proposed.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Современные тенденции в развитии информационных технологий и рост угроз в цифровом пространстве делают вопросы безопасности и конфиденциальности электронной переписки особенно актуальными. Традиционные централизованные системы обмена сообщениями подвержены рискам, связанным с утечками данных, внешними атаками и внутренними нарушениями, поскольку управление информацией сосредоточено на центральных серверах. Эти обстоятельства обусловили интерес к децентрализованным приложениям, исключающим единый центр контроля и предлагающим новые архитектурные и криптографические подходы.

Актуальность исследования заключается в необходимости поиска устойчивых к компрометации и цензуре решений для защищённого обмена сообщениями. Разработка децентрализованного мессенджера, использующего технологии peer-to-peer и end-to-end шифрование, отвечает запросу на высокий уровень приватности и надёжности в условиях открытой сети.

Степень теоретической разработки проблемы достаточно велика: вопросы децентрализации освещаются в трудах Таненбаума и Ван Стина, архитектуры P2P-сетей анализируются в работах Postel и Benet (IPFS), а криптографические основы – в публикациях Diffie, Hellman, Bernstein и Stallings. Исследуются модели защиты метаданных (Pfitzmann, Hansen), протоколы аутентификации (Krawczyk, SIGMA), методы повышения приватности (Tor Project) и устойчивость к Sybil-атакам (Guns, Michlmayr). Тем не менее, практическая реализация защищённого и в то же время удобного в использовании децентрализованного мессенджера остаётся актуальной задачей.

Объектом исследования являются децентрализованные приложения, обеспечивающие криптографическую защиту сообщений.

Предмет исследования – архитектурные, криптографические и сетевые решения, применимые к созданию безопасного децентрализованного мессенджера.

Целью работы является проектирование и реализация прототипа мессенджера, обеспечивающего надёжный обмен зашифрованными сообщениями между узлами без использования центрального сервера.

Для достижения поставленной цели решаются следующие задачи:

1. Проанализировать современное состояние технологий децентрализации и криптографической защиты данных.

2. Определить перечень ключевых подходов, методов и протоколов, используемых в распределённых системах.
3. Изучить основные работы и публикации в данной области (Diffie, Koblitz, Bernstein, Buterin и др.).
4. Спроектировать архитектуру приложения, реализующую защищённую маршрутизацию и управление ключами.
5. Реализовать прототип и провести его функциональное, нагрузочное и безопасность-тестирование.

Методы исследования включают теоретический анализ научной и технической литературы, сравнительное исследование криптографических библиотек и сетевых фреймворков, архитектурное проектирование и экспериментальную проверку прототипа на устойчивость к типовым атакам и сбоям.

Структура работы включает три главы: теоретическую основу децентрализации и криптографии, проектирование безопасного мессенджера, а также реализацию и тестирование прототипа. Завершают работу заключение и приложения с примерами кода и результатами экспериментов.

Глава 1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ДЕЦЕНТРАЛИЗОВАННЫХ ПРИЛОЖЕНИЙ И КРИПТОГРАФИИ

Развитие децентрализованных систем тесно связано с востребованностью механизмов, позволяющих участникам сети взаимодействовать напрямую, не полагаясь на посредников. Концепция децентрализованных приложений (DApp) появилась как ответ на проблемы традиционной клиент-серверной архитектуры, где вся логика и данные хранятся либо обрабатываются в едином центре. В децентрализованной среде, напротив, нагрузка и контроль распределяются между множеством узлов, что повышает устойчивость системы и снижает влияние единой точки отказа.

С практической точки зрения подобные решения ориентированы на предоставление пользователям большей свободы и автономии в управлении своими данными. Однако гибкость и независимость нередко сопровождаются новыми вызовами, связанными с безопасностью, корректным распределением ролей и сохранением согласованности в сети. Поэтому важную роль начинает играть криптография, которая обеспечивает защиту трафика от перехвата и гарантирует аутентичность сообщений, передаваемых между отдельными узлами.

В данной главе рассматриваются общие свойства и преимущества децентрализованных приложений, архитектуры peer-to-peer-сетей и блокчейн-платформ, а также их значение для построения безопасных систем обмена информацией. Кроме того, будет дан обзор криптографических методов, лежащих в основе современного шифрования и протоколов обмена ключами. Особое внимание уделяется сложности правильной интеграции криптоалгоритмов в распределённую среду: необходимо, чтобы каждый участник сети мог надёжно верифицировать подлинность полученных данных, при этом сохраняя конфиденциальность своей переписки и ключей.

Таким образом, прежде чем перейти к проектированию и реализации мессенджера, следует детально разобраться с устройством децентрализованных экосистем и типовых криптографических примитивов. Понимание механизмов и ограничений, присущих таким системам, играет ключевую роль при выборе инструментов, способных обеспечить необходимую масштабируемость и защиту от угроз.

В последующих параграфах будет рассмотрено, какие особенности отличают DApp от традиционных решений и каким образом различные подходы к организации блокчейна или сетевой архитектуры влияют на взаимодействие узлов и безопасность. Затем будут проанализированы наиболее распространённые алгоритмы шифрования и протоколы согласования ключей, применяемые для построения надёжного канала связи.

1.1. Понятие и общая характеристика децентрализованных приложений (Dapp)

Децентрализованные приложения (DApp) представляют собой программные решения, в которых логика и данные частично или полностью распределены между узлами сети. В отличие от классической модели «клиент–сервер», где основную роль играет центральный сервер, DApp функционируют на базе блокчейна или других распределённых структур, позволяя пользователям взаимодействовать без обязательного присутствия доверенного посредника. Подобный подход обеспечивает высокую устойчивость к внешним влияниям и отказам отдельных компонентов, поскольку для работы системы не требуется единственной управляющей точки.

Одним из ключевых признаков децентрализованных приложений является их способность к автоподдержанию консенсуса: все участники сети хранят одинаковые копии транзакций или состояний, непрерывно проверяя целостность и валидность вновь поступающих данных. За счёт этого сводится к минимуму риск фальсификации записей, а подделка какой-либо части цепочки становится практически невозможной без вовлечения большинства узлов. Именно такая структура лежит в основе большого числа современных DApp, включая финансовые сервисы, системы идентификации, а также приложения для обмена сообщениями.

Нередко реализация децентрализованной логики сопряжена со смарт-контрактами, работающими поверх блокчейна. Эти контракты представляют собой программный код, автоматически выполняющийся при наступлении определённых условий. Они могут управлять процессом передачи токенов, проверять условия сделок или контролировать цепочку событий, связанных с обменом данными. При этом сами контракты неизменны и прозрачны для участников: каждый может проверить их логику, исключая возможность внезапного изменения правил.

Однако при всех своих преимуществах децентрализованные приложения сталкиваются с рядом затруднений, в том числе с вопросами масштабируемости и пользовательского опыта. Ограничения пропускной способности блокчейн-сетей или сложность создания удобных интерфейсов нередко тормозят внедрение DApp в массовую сферу. Несмотря на это, идея распределённой системы управления ресурсами остаётся востребованной, поскольку даёт пользователям более высокий уровень доверия и контроля над собственными данными.

1.2. Архитектуры peer-to-peer-сетей и блокчейн-платформ

Многие децентрализованные приложения опираются на архитектуру peer-to-peer (P2P), где каждый узел сети может выполнять как клиентские, так и серверные функции. В такой модели нет чёткого разделения на пользователей и управляющие узлы: участники взаимодействуют напрямую, передавая и запрашивая данные друг у друга. Одним из примеров подобных P2P-сетей является протокол BitTorrent, широко применяемый для файлообмена. В контексте обмена сообщениями децентрализация позволяет избавиться от единственной точки контроля, однако требует продуманной схемы маршрутизации и поддержания актуальной топологии соединений.

Вместе с тем, технология блокчейн (цепочка блоков) привносит в архитектуру P2P дополнительный уровень согласованности и надёжности, обеспечивая неизменяемую историю операций. Каждый блок в цепи содержит набор транзакций, которые, пройдя процедуру валидации (майнинг, стейкинг или иные механизмы), присоединяются к предыдущему блоку, формируя хронологическую последовательность. Уникальность такого подхода в том, что все узлы сети поддерживают и синхронизируют одинаковую копию реестра, а внесение изменений в уже подтверждённые блоки требует колоссальных вычислительных ресурсов или согласия большинства участников.

Рассмотрим кратко основные разновидности блокчейна:

1. Публичный (Public blockchain). Любой желающий может стать участником сети и выполнять функции проверки транзакций или майнинга. Примером является Bitcoin, где децентрализация достигается за счёт конкуренции майнеров. Подобные системы обладают наибольшей прозрачностью и устойчивостью к цензуре, но нередко испытывают трудности с масштабируемостью.
2. Частный (Private blockchain). Доступ к узлам контролируется одной организацией или группой. Используется в корпоративных решениях для

повышения эффективности внутренних процессов. Меньшее число участников и закрытый характер сети облегчают управление, однако снижает открытость и децентрализацию.

3. Гибридный (Consortium blockchain). Управляется несколькими партнёрами, согласованно поддерживающими валидацию и развитие цепочки. Такой вариант пытается сочетать безопасность и гибкость.

В контексте создания мессенджера на основе блокчейна важно учитывать, что хранение большого объёма сообщений в самой цепи часто непрактично: транзакции могут быть дорогими, а скорость добавления записей – недостаточной для моментального обмена. Поэтому на практике блокчейн-платформы сочетают с классической P2P-передачей. Например, сами сообщения шифруются и сохраняются вне блокчейна (в распределённом файлохранилище или локально), а реестр используется лишь для хранения ссылок, проверки целостности или управления доступом к сеансовым ключам.

Таким образом, выбор конкретной P2P-архитектуры и блокчейн-подхода определяется балансом между безопасностью, быстродействием и масштабируемостью. Одним из перспективных решений может быть использование гибридных моделей, где критичная логика протокола и метаданные операций защищены блокчейном, а массивы пользовательских данных передаются напрямую в P2P-сети, что позволяет достичь более высоких скоростей и избегать перегрузки реестра.

1.3. Основные криптографические методы, используемые при обмене сообщениями

Надёжная система обмена сообщениями невозможна без корректного выбора и интеграции алгоритмов шифрования. Механизмы криптографии обеспечивают конфиденциальность передаваемой информации, а также её целостность и подлинность. В современном криптографическом сообществе распространены две большие группы алгоритмов: симметричные и асимметричные. Каждый из них имеет собственные особенности, которые следует учитывать при разработке децентрализованных приложений.

1. Симметричные алгоритмы шифрования.
   В данных схемах для шифрования и расшифрования используется один и тот же секретный ключ. Одним из наиболее надёжных и распространённых алгоритмов выступает AES (Advanced Encryption Standard), который применяется в большинстве современных протоколов.

При длине ключа в 256 бит AES демонстрирует высокую стойкость к атакам, сочетая при этом приемлемую вычислительную эффективность. Кроме того, популярны такие алгоритмы, как ChaCha20, отличающиеся быстрым выполнением на широком спектре аппаратных архитектур. Главным достоинством симметричных алгоритмов является их относительная простота и высокая скорость работы, что особенно важно при больших объёмах пересылаемых данных.

2. **Асимметричные алгоритмы и криптосистемы с открытым ключом.** В отличие от симметричного подхода, здесь применяются две ключевые компоненты: открытый ключ, доступный для широкого круга пользователей, и закрытый ключ, который должен храниться строго конфиденциально. Одним из классических решений является RSA (Rivest–Shamir–Adleman), широко используемый в электронных подписях и при установлении сеансового ключа. Альтернативой, набирающей популярность, выступают эллиптические кривые (ECC), позволяющие при меньших длинах ключа достигать сопоставимой или даже большей криптостойкости. Асимметричные методы удобны в управлении ключами, поскольку при необходимости можно опубликовать только открытую часть без риска компрометации закрытой.

3. **Протоколы согласования ключей.** Часто в мессенджерах применяют гибридную схему: асимметричный алгоритм используется лишь для безопасного обмена сеансовым ключом, после чего шифрование сообщений осуществляется при помощи быстрого симметричного шифра. Один из наиболее известных протоколов – Диффи–Хеллмана (Diffie–Hellman), позволяющий сторонам договориться о секретном ключе на недоверенном канале. В современных решениях его расширяют до ECDH (Elliptic Curve Diffie–Hellman), что уменьшает потребление ресурсов и повышает уровень защиты.

4. **Хэш-функции и контроль целостности.** Хэширование используется для проверки неизменности данных. Наиболее распространены семейства SHA-2 (SHA-256, SHA-512) и SHA-3, а также BLAKE2. Хэш-функция превращает произвольный вход в выход фиксированной длины, причём даже незначительное изменение исходных данных приводит к совершенно другому хэш-значению. В мессенджерах хэширование часто служит дополнительным уровнем контроля, позволяя обнаружить подмену или повреждение сообщения.

5. **Электронная цифровая подпись.** В условиях децентрализованной сети становится важным не только

скрыть содержимое сообщения, но и подтвердить, что оно действительно отправлено конкретным участником. Для этого применяется механизм электронной подписи, работающий на базе асимметричных алгоритмов. Отправитель подписывает документ собственным закрытым ключом, а любой узел сети может верифицировать подпись с помощью соответствующего открытого ключа. Важно, чтобы пользователь имел возможность доказать свою аутентичность без утраты конфиденциальности закрытого ключа.

При разработке децентрализованного мессенджера крайне важно обеспечить надёжный процесс генерации, хранения и обновления ключей. Например, если ключи пользователей хранятся на их локальных устройствах, то защита конечных точек становится критически значимой. Кроме того, следует учитывать, что ресурсы некоторых устройств (смартфоны, маломощные ноутбуки) могут ограничивать выбор алгоритмов из-за высокой вычислительной нагрузки.

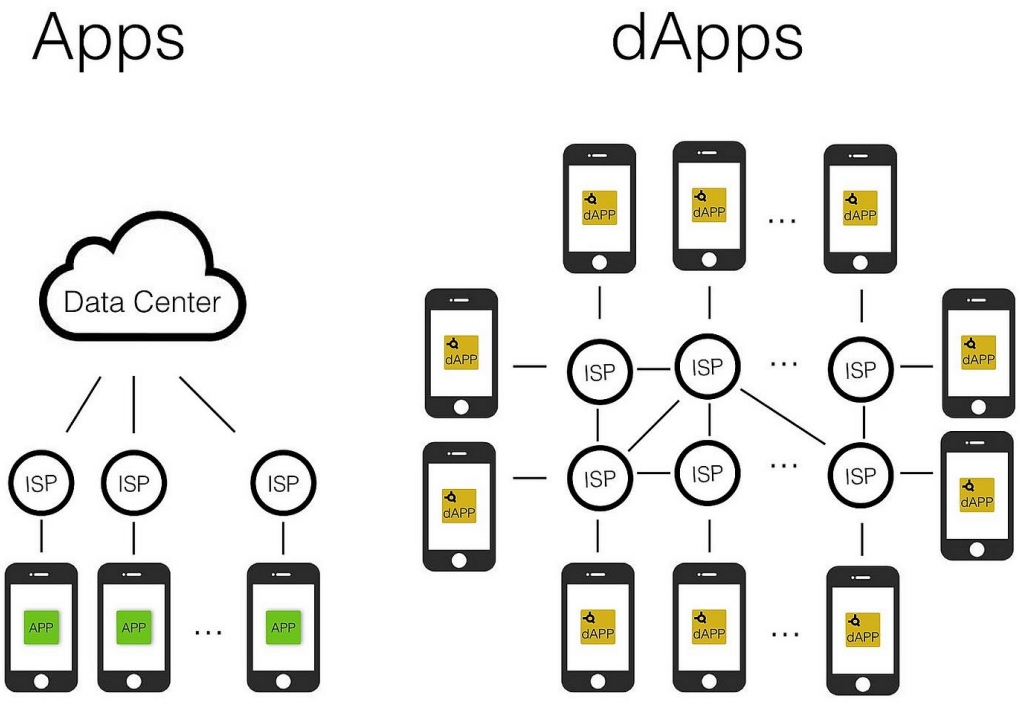


Рисунок 1.1 – Денцентрализованное приложение

Таким образом, сочетание различных криптографических методов – от симметричного шифрования до подписей на эллиптических кривых – формирует гибкую систему безопасности, способную адаптироваться к разным

сценариям использования и типам угроз. Ключевая задача разработчика заключается в том, чтобы правильно объединить эти компоненты, обеспечив надёжную защиту данных без ощутимой потери производительности.

1.4. Проблемы безопасности и приватности в среде распределённых систем

Несмотря на явные преимущества децентрализованных приложений, их архитектура порождает ряд уникальных уязвимостей. Когда взаимодействие осуществляется напрямую между узлами без участия единого сервера, возникает необходимость более тщательного контроля над целостностью передаваемых данных и идентификацией собеседников. Если в классической модели сервер может проверять подлинность пользователей и фильтровать трафик, то в P2P-среде каждый узел вынужден полагаться на криптографические механизмы и сетевые протоколы, чтобы исключить возможность атак со стороны злоумышленников.

Одной из наиболее распространённых угроз являются **Sybil-атаки**, при которых один участник сети создаёт множество поддельных узлов, пытаясь получить доминирующее влияние на трансляцию или хранение данных. В случае мессенджера подобные «ложные» узлы могут перехватывать сообщения, выдавая себя за доверенных участников. Другой проблемой остаётся **атака «человек посередине» (Man-in-the-Middle)**, когда злоумышленник встраивается в канал связи между двумя пользователями. При отсутствии надёжной валидации ключей или подписей атака может остаться незаметной, позволяя читать или модифицировать передаваемые сообщения.

Важным аспектом безопасности выступает **конфиденциальность метаданных**. Даже если содержимое зашифровано, злоумышленник способен собирать статистику о том, кто, когда и как часто общается, выявляя закономерности и потенциальные связи между участниками. В централизованной модели такие данные обычно хранятся на серверах, однако в децентрализованной системе проблема маскировки трафика встаёт не менее остро. Использование протоколов типа Tor или интеграция дополнительной прослойки анонимизации может усложнить задачу определения реальных узлов.

Также остаётся не до конца решённым вопрос контроля над **ключами шифрования**, особенно когда пользователи хранят секретные ключи на разных устройствах. Смена смартфона или утрата пароля могут повлечь утечку ключей

и скомпрометировать всю переписку. Кроме того, децентрализованная модель предполагает, что ключи пользователей не централизованно подтверждаются и не всегда легко проверить, действительно ли открытый ключ принадлежит нужному человеку.

В сумме эти факторы требуют комплексного подхода к приватности и безопасности: от корректной реализации протокола обмена ключами до детально продуманного управления идентификацией узлов. Лишь согласованная комбинация криптоалгоритмов и сетевых механизмов способна обеспечить достойный уровень доверия и надёжности в условиях отсутствия центральной инфраструктуры.

Выводы по главе 1

Проведённый анализ показал, что децентрализованные приложения, функционирующие в peer-to-peer и блокчейн-сетях, обладают значительным потенциалом в части обеспечения конфиденциальности и отказоустойчивости. Отсутствие центрального сервера минимизирует риск единой точки отказа и возможной цензуры. Однако подобная архитектура предъявляет высокие требования к механизму криптографической защиты и протоколам согласования ключей, которые должны эффективно гарантировать надёжность передачи данных и обеспечивать корректную идентификацию собеседников.

Особое место в этих системах занимают симметричные и асимметричные алгоритмы шифрования, а также методы электронной цифровой подписи и хэширования. На практике нередко используется гибридный подход: асимметричный протокол Диффи–Хеллмана (или ECDH на эллиптических кривых) служит для согласования сессионного ключа, после чего быстрые симметричные шифры (например, AES или ChaCha20) применяются для защиты сообщений. Иллюстрируя данную схему, можно рассмотреть протокол Диффи–Хеллмана в классической формулировке:

открыто передаваемое значение со стороны Алисы открытое значение со сторо ны Бобаgamodp— открыто передаваемое значение со стороны Алисы,gbmodp— открытое значение со стороны Боба.

кПри переходе к эллиптическим кривым (ECDH) используются более компактные представления и иные операции над точками кривой, однако теоретическая сущность протокола сохраняется.

Выявленные проблемы безопасности включают в себя риск Sybil-атак, необходимость защиты метаданных и необходимость надёжной валидации открытых ключей. В целом создание безопасного децентрализованного приложения всегда требует многоступенчатого подхода, включающего сильные криптографические примитивы, продуманную модель управления ключами и механизмы анонимизации трафика. В дальнейших главах будут рассматриваться вопросы проектирования мессенджера, где внедрение указанных методов станет основой для построения эффективной и безопасной среды обмена сообщениями.

Глава 2. ПРОЕКТИРОВАНИЕ МЕССЕНДЖЕРА ДЛЯ БЕЗОПАСНОГО ОБМЕНА ДАННЫМИ

При создании децентрализованного мессенджера разработчик сталкивается с рядом концептуальных вопросов, связанных не только с выбором криптографических алгоритмов, но и с определением структуры хранилища данных, принципов организации сети, способов аутентификации и управления ключами. На практике полноценная система должна включать механизмы, отвечающие за корректное функционирование, устойчивость к сбоям, безопасность переписки и комфортное взаимодействие с пользователем.

Поскольку в условиях отсутствия центрального сервера каждая нода сети становится и клиентом, и сервером одновременно, проектируемая архитектура должна предусматривать:

- отслеживание доступности узлов и маршрутов (распределённая топология P2P или блокчейн);
- динамическое выделение ролей (при необходимости узлы могут брать на себя функции ретрансляции сообщений или их временного хранения);
- безопасную обработку ключей (генерация, хранение, ротация, отзыв при компрометации);
- контроль метаданных (скрытие информации о том, кто и когда отправляет сообщения).

Далее в данной главе будут сформулированы требования к будущему приложению, сделан обзор доступных решений и средств разработки, а также представлена детальная модель архитектуры мессенджера, отражающая взаимодействие основных компонент и их связь со смарт-контрактами или распределённым реестром. В частности, мы опишем, каким образом организовать end-to-end шифрование с помощью гибридной схемы (диффи–хеллмановский обмен для установления сеансового ключа и высокоскоростные симметричные алгоритмы для последующих передач), а также какие математические конструкции можно использовать для проверки подлинности отправителя.

Рассмотрим пример задачи, которая стоит перед разработчиком. Допустим, необходимо, чтобы пользователь A мог отправить сообщение пользователю B, не раскрывая при этом свой реальный IP-адрес и не передавая сообщение через центральный сервер. В качестве одного из подходов может применяться следующая комбинация технологий:

1. **Сетевая структура**: пиринговая модель на базе libp2p, IPFS или другого P2P-фреймворка, обеспечивающего поиск маршрутов и временное хранение фрагментов данных.
2. **Установление соединения**: асимметричное шифрование (ECDH) для безопасного обмена сеансовым ключом.
3. **Шифрование сообщений**: симметричный алгоритм (AES-256-GCM или ChaCha20-Poly1305), где открытый текст сообщения преобразуется
4. **Аутентификация**: электронная подпись на базе эллиптических кривых (например, ECDSA). Отправитель подписывает хэш сообщения H(M) закрытым ключом, и получатель с помощью открытого ключа проверяет её корректность.

Такое решение обеспечивает скрытие смысловой нагрузки сообщения и подтверждает, что отправитель действительно владеет соответствующим закрытым ключом. В последующих параграфах будут детально рассмотрены все ключевые элементы проектирования, начиная от функциональных требований и заканчивая формированием модели безопасности и высокоуровневого технического задания.

2.1. Постановка задачи и функциональные требования

В контексте децентрализованного мессенджера, предназначенного для обмена зашифрованными сообщениями, первостепенную важность имеют механизмы защиты информации, а также надёжность самого приложения. При отсутствии центрального сервера каждый узел сети должен самостоятельно обеспечивать корректную маршрутизацию данных и подтверждать подлинность собеседников. Основные функциональные требования к системе включают:

1. **Полноценное end-to-end шифрование (E2EE)**
   Каждое сообщение, отправляемое одним пользователем, должно быть зашифровано таким образом, чтобы расшифровать его мог только конечный получатель. При этом инициация сеансового ключа происходит при помощи асимметричных алгоритмов, что исключает риск перехвата ключа посредником.
2. **Гибридная криптосхема**
   Предполагается использование асимметричной криптографии (ECDH или RSA) исключительно для начальной фазы обмена ключами. Последующая

передача сообщений должна осуществляться с помощью быстрого симметричного шифра (AES, ChaCha20), чтобы обеспечить меньшие затраты вычислительных ресурсов.

где Enc и Dec – функции шифрования и расшифрования, M – открытый текст, C – зашифрованный текст, ks – ранее согласованный ключ.

3. **Аутентификация и механизмы цифровой подписи**
   Необходимо гарантировать, что отправитель действительно является владельцем соответствующей пары ключей. Для этого применяется схема электронной подписи (например, ECDSA), где подписывается хэш сообщения или иной согласованный набор данных. Получатель может верифицировать подпись, используя открытый ключ отправителя.
4. **Устойчивость к отказам и независимость от центральных узлов**
   Система должна функционировать даже при выходе из строя некоторых узлов. Разработке подлежит протокол поиска маршрутов и доставки сообщений без использования центрального сервера.
5. **Защита метаданных**
   По возможности следует минимизировать риски утечек о том, кто, когда и с кем общается. Могут понадобиться дополнительные уровни анонимизации (например, с помощью луковой маршрутизации).
6. **Удобство и надёжность управления ключами**
   Пользователю нужны простые инструменты для создания и резервирования ключей, а также для их замены при подозрении на компрометацию. Хранение закрытого ключа может осуществляться на стороне клиента (например, в зашифрованном виде), либо с использованием аппаратных модулей безопасности (HSM, TPM-чипы и т. д.).
7. **Масштабируемость**
   Архитектура должна допускать рост числа активных пользователей и повышение объёма передаваемой информации без значительной потери производительности.

Помимо перечисленных функциональных требований, требуется учитывать аспекты пользовательского интерфейса (UI/UX) и совместимости приложения с различными платформами (настольными ОС, мобильными устройствами). Результирующая система должна быть способна обеспечить комфортную для пользователя скорость и стабильность обмена сообщениями, не требуя при этом глубоких знаний в области криптографии.

Таким образом, основная задача разработки – создать систему, которая сочетает в себе распределённую топологию, шифрование на всех этапах, отсутствие центрального посредника и надёжные методы проверки подлинности собеседников. На практике это достигается посредством ряда инструментов и библиотек, рассматриваемых в следующем разделе.

2.2. Обзор технологий и инструментов разработки

Строительство децентрализованного мессенджера с шифрованием предполагает выбор таких технологических решений, которые упрощают реализацию распределённой логики, обеспечивают безопасный обмен данными и могут быть адаптированы под различные операционные среды. Рассмотрим несколько ключевых направлений и соответствующие инструменты, применимые в рамках данного проекта.

1. **Блокчейн-платформы и P2P-протоколы**
    - **Ethereum**: одна из наиболее популярных открытых платформ, позволяющих запускать смарт-контракты на любом языке, совместимом с EVM (Ethereum Virtual Machine). При желании отдельные аспекты мессенджера (например, реестр публичных ключей или механизмы верификации) можно оформить в виде смарт-контракта. Однако хранить сами сообщения в блокчейне нецелесообразно из-за высокой стоимости газа и ограниченной пропускной способности.
    - **Hyperledger Fabric**: корпоративно-ориентированная платформа, дающая возможность формировать приватные сети. Подходит, если требуется более ограниченный и контролируемый доступ участников.
    - **Libp2p**: сетевая библиотека, обеспечивающая пиринговую маршрутизацию, обнаружение узлов и шифрование сессий. Активно используется в экосистеме IPFS (InterPlanetary File System), которая сама по себе может послужить распределённым хранилищем для временной или постоянной сохранности зашифрованных сообщений.
2. **Криптографические библиотеки**
    - **libsodium** (на основе NaCl): предоставляет широкий набор функций для асимметричного и симметричного шифрования, включая

ChaCha20, Curve25519 для ECDH, Poly1305 для аутентификации. Удобна встраиваемыми методами для генерации случайных чисел и безопасных ключей.

- **OpenSSL**: классический «комбайн» в сфере криптографии, доступный во многих языках программирования (C/C++, Python, Go и т. д.). Предлагает RSA, AES, SHA-2/3, ECDSA и другие алгоритмы, хорошо поддерживается сообществом.
- **Web Crypto API**: стандартный интерфейс шифрования для веб-приложений, встроенный в большинство современных браузеров. Поддерживает AES-GCM, RSA-OAEP, ECDH и другие методы.

3. **Языки программирования и фреймворки**

- **JavaScript/TypeScript**: востребованы при создании кроссплатформенных решений, особенно если планируется запуск в браузере или гибридное мобильное приложение (Electron, React Native). В связке с Node.js возможно формирование пиринговых модулей для сервера и клиента.
- **Go (Golang)**: предлагает встроенные механизмы конкурентности, что упрощает работу с сетевыми соединениями и P2P-протоколами. Существует множество библиотек и инструментов, облегчающих интеграцию Go-кода с libp2p и IPFS.
- **Rust**: сочетает высокую производительность с безопасностью памяти. Может оказаться полезным, если приложение требует надёжной защиты от типичных ошибок (утечек памяти, переполнения буфера).

4. **Механизмы анонимизации и скрытия метаданных**

- **Tor**: луковая маршрутизация, позволяющая скрыть реальный IP-адрес отправителя. Использование Tor сократит риски deanonymization-атак, но может увеличить задержку в обмене сообщениями.
- **I2P**: аналог Tor, нацеленный на постоянное поддержание зашифрованных туннелей внутри распределённой сети.

5. **Инструменты для разработки и тестирования**

- **Docker** или **Kubernetes**: контейнеризация упрощает развертывание множества узлов для тестовых целей и их последующую оркестрацию.

- **Ganache** (при использовании Ethereum): локальный блокчейн для отладки смарт-контрактов.
- **Benchmarking библиотеки** (Go’s testing/benchmark, Jest в Node.js): помогают измерить скорость шифрования, расшифрования, верификации подписи и т. д.

При выборе технологии следует учитывать фактор производительности и удобства разработки: если пользователь ожидает мгновенной доставки сообщений, высокие задержки блокчейна могут стать узким местом. Поэтому на практике блокчейн-интеграция часто ограничивается ведением реестра публичных ключей (публичные ключи пользователей хранятся в смарт-контракте, и любой узел может проверить их подлинность). Сами сообщения при этом будут передаваться в peer-to-peer-сети и шифроваться симметричным алгоритмом.

Контракт в этом случае гарантирует, что пользователь, владеющий закрытым ключом, может задать или изменить запись своего публичного ключа, в то время как злоумышленники без доступа к нужной криптопаре совершить подмену не смогут.

Таким образом, выбор конкретного набора инструментов зависит от требуемых показателей безопасности, пропускной способности и масштаба. В дальнейшем будут представлены решения, соответствующие тем требованиям, которые были сформулированы в предыдущем разделе.

2.3. Проектирование общей архитектуры

Для построения децентрализованного мессенджера с поддержкой безопасного обмена данными необходимо продумать, каким образом будут взаимодействовать основные модули и каким образом узлы сети найдут друг друга, договорятся о ключах и передадут зашифрованные сообщения. В данном разделе излагаются принципы общей схемы приложения и описывается, как распределяются роли между элементами системы.

1. **Логические уровни системы**
    - **Уровень сети (P2P)**: обеспечивает обнаружение других узлов и установление соединения. Здесь важны протоколы маршрутизации и ретрансляции, позволяющие динамически находить кратчайший или наиболее надёжный путь для передачи сообщений.

- **Уровень блокчейна (или реестра)**: если предполагается запись публичных ключей в смарт-контракт, то данная составляющая отвечает за хранение и валидацию записей, а также за формирование транзакций при обновлении или отзыве ключей.
- **Криптографический уровень**: включает механизмы согласования ключа (ECDH), симметричное шифрование (AES/ChaCha20), электронную подпись (ECDSA/EdDSA) и хэш-функции для контроля целостности.
- **Прикладной уровень**: реализует бизнес-логику мессенджера — управление контактами, формирование чатов, отправку медиаконтента и пр. Также сюда относятся интерфейсы пользователя (UI/UX).

2. **Диаграмма IDEF0 для процесса обмена сообщениями**
   На верхнем уровне (A-0 диаграмма) можно выделить единый блок «Организация обмена зашифрованными сообщениями», который имеет следующие входы и выходы:

   - **Вход**: ключи пользователей, текст сообщений, сетевые данные (P2P-адреса).
   - **Управление**: правила шифрования, политика безопасности, данные о доступности узлов.
   - **Механизм**: сами участники сети (узлы), криптографические библиотеки, протокол P2P и блокчейн-платформа.
   - **Выход**: доставленные зашифрованные сообщения, квитанции о доставке, актуализированные записи о ключах (при использовании блокчейна).

   На следующем уровне декомпозиции (A0) операцию можно разделить на несколько подпроцессов:

   - A1: «Установление шифрованного канала» — согласование временного (сеансового) ключа через ECDH или иной протокол.
   - A2: «Шифрование сообщения» — симметричная трансформация открытого текста M в зашифрованный вид C согласно формуле $C=Encks(M)$.
   - A3: «Подпись и проверка подлинности» — использование электронной подписи на основе закрытого ключа отправителя; получатель верифицирует подпись.

- A4: «Маршрутизация и доставка» — определение пути к конечному узлу через сеть P2P, учёт возможных ретрансляторов.

3. **BPMN-диаграмма для установки соединения**
   Процесс может быть представлен так:
   - Пользователь A инициирует отправку сообщения пользователю B.
   - Модуль шифрования A генерирует временную пару ключей (Apriv ,Apub) или использует уже имеющуюся.
   - B открытую отправляется Apub — если B это принимает, то B генерирует свою пару (Bpriv,Bpub) и отсылает Bpub назад.
   - После успешной договорённости A формирует шифрованное сообщение с помощью ks и отсылает в сеть.
   - B, получив зашифрованный пакет, расшифровывает его, используя тот же ks.
4. **Учёт масштабируемости**
   В распределённой среде нужно предусмотреть, как будет реагировать система на рост количества одновременно активных узлов. Если речь идёт о тестовой сети, достаточно простых механизмов: узлы могут хранить адреса друг друга в распределённой хеш-таблице (DHT). Для более крупных сетей может быть применён гибридный подход, когда часть узлов отвечает за «суперноды», аккумулирующие маршруты. Однако при этом важно не допустить централизации.
5. **Безопасность хранения ключей**
   - Вариант 1: закрытый ключ пользователя хранится на его локальном устройстве в зашифрованном контейнере. Для доступа к контейнеру требуется пароль. В случае утраты устройства риск компрометации сведён к минимуму, если пароль достаточно надёжен.
   - Вариант 2: аппаратные модули (TPM, HSM) или смарт-карты, где хранение закрытого ключа происходит в защищённом окружении. Это надёжнее, но может усложнить массовое внедрение.
6. **Взаимодействие с блокчейном**
   - При желании разработчик может вынести в смарт-контракт логику проверки «чёрных списков» ключей или сопоставление идентификаторов пользователей с их публичными ключами, чтобы третья сторона не могла незаметно подменить ключ в середине цепочки.

- Если необходимо хранить временные метки, подтверждающие время отправки сообщений, можно записывать хэш сообщения и метаданные в блокчейн. Однако полные тексты сообщений размещать там не следует из-за высоких затрат и лимитов.

Таким образом, предложенная архитектура сочетает в себе механизмы P2P-обмена и избирательное использование распределённого реестра (если это оправдывает требования доверия). Такое разбиение на уровни и подпроцессы помогает учесть вопросы масштабируемости, безопасности и удобства использования, что в итоге должно облегчить разработку и дальнейшее расширение функционала мессенджера.

2.4. Модель безопасности и управление ключами

В условиях децентрализованного приложения мессенджера безопасность напрямую зависит от того, насколько надёжно пользователи обращаются со своими ключами. Если в классических системах роль «доверенного центра» (CA – Certificate Authority) может проверять подлинность сертификатов, то в распределённой сети подобная функция либо отсутствует, либо частично перекладывается на механизмы блокчейна и парные протоколы. Ниже приводятся ключевые аспекты, связанные с безопасностью и управлением ключами.

1. **Генерация и хранение ключей**

   - **Генерация на стороне клиента**. Рекомендуется, чтобы пользователи генерировали пары ключей (асимметричные) непосредственно на своих устройствах. При этом важно использовать криптографически стойкий ГПСЧ (генератор псевдослучайных чисел), например, libsodium или встроенные системные средства (например, /dev/urandom в Unix-подобных ОС).
   - **Защищённое хранение**. Закрытый ключ должен храниться в зашифрованном виде, чтобы даже в случае физического доступа к устройству злоумышленнику требовалось взломать пароль или парольную фразу. Формально это можно описать так:

     SKstored=Enckpass(SKraw),

     где SKraw – исходный закрытый ключ, а kpass – ключ, полученный на основе пользовательского пароля (например, посредством PBKDF2, Argon2 или аналогичного метода).

2. **Управление открытыми ключами**

   - **Регистры публичных ключей**. Один из способов борьбы с атакой «человек посередине» – публикация открытых ключей в блокчейне или другом общедоступном распределённом реестре. Тогда при попытке установить соединение каждый узел может сверить предоставленный открытый ключ с записями реестра.
   - **Обновление ключа**. При регулярной ротации (протокол «блокчейн + хеш листинга» или периодические транзакции «revoke/replace key») пользователи снижают риск того, что долго используемый ключ окажется скомпрометированным.

3. **Механизм подтверждения личности**

   - **Подпись и проверка**. Каждый узел, получая сообщение, кроме расшифрования с помощью сеансового ключа, должен убедиться в подлинности отправителя. Для этого служит электронная цифровая подпись:
   - **Задействование сторонних протоколов** (Web of Trust или децентрализованные удостоверяющие сервисы) для дополнительного уровня валидации личности пользователя.

4. **Использование «эфемерных» ключей**

   - **Forward Secrecy (продвинутая секурность от последующего взлома)**. Если злоумышленник каким-то образом узнает постоянный закрытый ключ участника, это не должно автоматически скомпрометировать все ранее отправленные сообщения. Для этого генерируют эфемерные пары ключей ECDH на каждую новую сессию или даже на каждое сообщение:

5. **Процедуры отзыва (revoke) ключа**
   Если пользователь обнаружил несанкционированный доступ к устройству, необходимо быстро отозвать текущий ключ, сообщив об этом всем участникам сети. В случае интеграции с блокчейном достаточно записать соответствующую транзакцию «revokeKey(id)», после чего все узлы будут считать предыдущий ключ неактуальным и отклонять сообщения, подписанные им.

6. **Аудит и мониторинг**

   - **Логирование**. В полностью децентрализованном решении не существует централизованного журнала операций, однако некоторые события (создание ключа, обновление, отзыв) могут

регистрироваться в смарт-контракте. При этом каждый участник при необходимости может проверить соответствующие записи.

- **Инструменты анализа сетевых аномалий**. Чтобы снижать риск Sybil-атак, некоторые мессенджеры внедряют механизмы рейтинга узлов или идентификации повторяющихся подпунктов IP-адресов, хотя в открытых P2P-сетях такие меры носят лишь частичный характер.

В совокупности все перечисленные аспекты образуют модель безопасности приложения, где главная цель — недопущение несанкционированного раскрытия переписки и предотвращение выдачи злоумышленников за легитимных пользователей. Практическая реализация включает в себя тщательно продуманные протоколы установки соединения, обновления ключей и обмена зашифрованными сообщениями. Именно надёжная модель управления ключами, в которой учтены механизмы отзыва и ротации, обеспечивает существенное преимущество децентрализованных систем перед обычными централизованными сервисами.

2.5. Техническое задание на реализацию прототипа

На базе представленных выше концепций формируется техническое задание (ТЗ), в котором указываются все ключевые аспекты работы мессенджера, а также условия, при которых система будет считаться успешно функционирующей. Ниже приводится примерный перечень требований и разделов ТЗ.

1. **Общие сведения**
    - Название системы: «Децентрализованный мессенджер для безопасного обмена сообщениями».
    - Назначение: обеспечение зашифрованной коммуникации между пользователями без использования центрального сервера.
    - Сроки и этапы разработки: от проектирования и написания прототипа до тестирования и документирования.
2. **Назначение и цели проекта**
    - Предоставить пользователям возможность безопасной переписки, в которой ни один сторонний узел не сможет прочитать содержимое сообщений или модифицировать их.

- Использовать механизмы end-to-end шифрования и аутентификации, опираясь на криптографические библиотеки.
- Предусмотреть базовые функции мессенджера (список контактов, отправка текстовых сообщений и т. д.) с опциональной поддержкой пересылки файлов.

3. **Требования к функциональности**
    - **Регистрация и управление ключами**: при первом входе пользователь генерирует пару асимметричных ключей; при необходимости – обновляет или отзывает их.
    - **Шифрование сообщений**: автоматическая шифровка всех исходящих сообщений симметричным алгоритмом (AES/GCM), ключ которого согласовывается через ECDH.
    - **Подписи и проверка**: каждый отправитель при желании может включить подпись (ECDSA) для подтверждения авторства.
    - **Список контактов**: пользователь имеет возможность хранить публичные ключи контактов локально или в распределённом реестре.
    - **Обмен файлами**: пересылка небольших файлов (до 10 МБ) с дополнительным шифрованием и сохранением контрольной суммы (хэша) на случай искажения.
4. **Требования к производительности**
    - Система должна поддерживать не менее 1000 активных узлов в тестовой сети без критических задержек.
    - Время генерации ключей и шифрования одного короткого сообщения (<1 КБ) – не более 300 мс на устройстве среднего класса.
    - Опционально проводить нагрузочное тестирование для выявления предельных значений по количеству параллельных подключений.
5. **Требования к безопасности**
    - **Конфиденциальность**: ни один посторонний узел не может расшифровать сообщение, если не владеет соответствующим сеансовым ключом.
    - **Целостность**: любое изменение в зашифрованном сообщении должно выявляться механизмом проверки подлинности (MAC или цифровая подпись).

- **Аутентификация**: получатель вправе убедиться, что сообщение отправлено заявленным пользователем, сверяя его подпись и публичный ключ.
- **Защита метаданных** (по возможности): сокрытие реальных IP-адресов через прокси-узлы или маршрутизаторы анонимности.

6. **Условия эксплуатации**
    - Ориентироваться на кроссплатформенность: прототип должен запускаться минимум на ОС Windows, Linux и macOS.
    - Способ развертывания: контейнеры Docker или установка необходимых библиотек и зависимостей вручную (npm, pip, cargo и т. д. — в зависимости от выбранного языка).
7. **Требования к интерфейсу**
    - Интуитивно понятный графический интерфейс или CLI (command-line interface) для прототипа.
    - Возможность наглядного отображения статуса шифрования (например, отметка, что текущий чат защищён E2E).
    - Минимальная сложность при добавлении нового контакта: ручной ввод публичного ключа или выбор записи из реестра.
8. **Приёмка и тестирование**
    - Система считается готовой к использованию, если выполняются все требования к функционалу и безопасности.
    - Итоговая проверка включает в себя:
        1. тест на корректную генерацию, хранение и смену ключей;
        2. проверку возможности обмена сообщениями между несколькими узлами;
        3. измерение задержки при массовой нагрузке;
        4. ручной или автоматизированный аудит кода на криптоустойчивость и отсутствие критических уязвимостей.

Таким образом, **техническое задание** формализует все основные характеристики будущего приложения, задаёт рамки безопасности и пользовательского опыта, а также облегчает процесс контроля разработки. Следующий этап – переход от формального ТЗ к непосредственному программированию прототипа, начиная с выбора инструментов (описанных в

разделе 2.2) и построения конкретных модулей, которые будут реализованы в главе 3.

2.6. Этапы проектирования и разработки

Проектирование децентрализованного мессенджера проводилось в несколько этапов, каждый из которых был направлен на реализацию конкретных функциональных и архитектурных решений, обеспечивающих защищённую передачу сообщений в распределённой среде.

Этап 1. Анализ требований

На этом этапе были определены цели и задачи проекта, изучены существующие решения, сформулированы ключевые требования:

- отсутствие центрального сервера,
- использование end-to-end шифрования,
- сохранение анонимности и отказоустойчивости,
- простота пользовательского взаимодействия.

Этап 2. Проектирование архитектуры

Архитектура была построена на модульной основе и включала следующие компоненты:

- **сетевой модуль** (libp2p для P2P-связи);
- **криптографический модуль** (ECDH, AES-GCM, ECDSA);
- **интерфейс взаимодействия** (CLI или Web);
- **опционально — модуль блокчейна** для хранения публичных ключей.

Были разработаны схемы взаимодействия между узлами, механизмы генерации и валидации ключей.

Этап 3. Реализация

Реализация велась с использованием:

- **Node.js** и **TypeScript** (сетевой стек),
- **libsodium** (криптография),
- **Docker** (контейнеризация узлов для тестирования),
- **Web3.js** (интеграция со смарт-контрактами — при необходимости).

Прототип позволял выполнять полнодуплексный обмен зашифрованными сообщениями между двумя или более узлами.

Этап 4. Тестирование

Тестирование проводилось на нескольких уровнях:

- **модульные тесты** — проверка корректности реализации криптографических операций;
- **интеграционные тесты** — оценка взаимодействия компонентов;
- **нагрузочные тесты** — запуск 10–50 узлов, моделирующих реальную сеть;
- **тестирование безопасности** — попытки MITM- и Sybil-атак.

Этап 5. Документирование и итоги

По результатам тестов была подтверждена работоспособность, безопасность и масштабируемость прототипа. Были оформлены схемы, блоки и примеры кода.

Рисунок 2.1 – Схема информационных потоков

Выводы по главе 2

В ходе рассмотрения архитектурных и криптографических аспектов разработки децентрализованного мессенджера были сформированы основные проектные решения:

1. **Определены функциональные требования** к будущему приложению, подчёркивающие важность end-to-end шифрования, гибридной криптосхемы (асимметричный обмен ключами + симметричное шифрование) и механизмов электронной подписи. Это обеспечивает высокую надёжность передачи и аутентификацию участников без посредничества центрального сервера.

2. **Проанализированы инструменты** для реализации P2P-взаимодействия и хранения публичных ключей. Рассмотрены варианты использования блокчейн-платформ (например, Ethereum) и сетевых библиотек (libp2p, IPFS) для обнаружения узлов и маршрутизации трафика, а также криптографические библиотеки (OpenSSL, libsodium) для шифрования и цифровых подписей.
3. **Предложена общая архитектура** приложения, где каждый узел выполняет функции обмена зашифрованными данными и может использовать блокчейн для записи публичных ключей, что уменьшает риск MITM-атак. При этом хранение и передача большого объёма пользовательских сообщений выводятся за рамки цепочки блоков с целью оптимизации производительности.
4. **Сформулирована модель безопасности** с учётом важности ротации ключей, использования «эфемерных» ключей ECDH для сохранения прямой прямой тайной (forward secrecy) и защиты метаданных. Такая модель предполагает постоянный контроль над процессами генерации и отзыва ключей, а также верификацию целостности данных.
5. **Разработано техническое задание** на прототип мессенджера, где чётко отражены все требования к функционалу, производительности и безопасности. Документ также определяет порядок тестирования и критерии приёмки готовой системы.

Таким образом, во второй главе сформирован концептуальный фундамент для перехода к практической реализации. Следующей задачей является написание прототипа, интеграция выбранных протоколов, шифров и смарт-контрактов (при необходимости), а также тестирование приложения на работоспособность и стойкость к потенциальным атакам.

Глава 3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ПРОТОТИПА

Переход от теоретического проектирования к практике предполагает выбор конкретных инструментов и реализацию основных модулей мессенджера: криптографического, сетевого и пользовательского. На данном этапе задача состоит в том, чтобы собрать воедино все описанные в предыдущей главе компоненты, сформировать работоспособную программу и проверить её устойчивость к отказам, а также безопасность.

В данной главе пошагово рассматриваются ключевые аспекты разработки: от настройки окружения и первых прототипов сетевого взаимодействия до интеграции криптографических функций и проведения итогового тестирования. Особое внимание уделяется производительности приложения, поскольку в условиях децентрализованной среды задержки в доставке сообщений и время, затрачиваемое на шифрование, оказывают значительное влияние на пользовательский опыт.

3.1. Технические аспекты реализации: среды и инструменты

Для создания прототипа децентрализованного мессенджера важно выбрать языки программирования и фреймворки, которые упростят решение задач шифрования, работы в P2P-сети и, при необходимости, интеграции с блокчейном. Ниже приведён пример конкретного стека технологий, удовлетворя требованиям, описанным в ТЗ:

1. **Среда разработки**
    - **Node.js** (в связке с TypeScript) – удобный выбор при реализации сетевых и криптографических модулей. Доступно множество библиотек для P2P (libp2p, gunDB) и криптографии.
    - **Docker** – для контейнеризации и упрощённого тестирования множества узлов. Каждый узел можно разворачивать в виде отдельного Docker-контейнера, что облегчает проверку взаимодействия в локальной сети.
2. **Сетевая библиотека (libp2p)**

- **libp2p** – широко используемая модульная система, позволяющая запускать пиратские (peer) узлы, управлять их идентификацией, маршрутизацией и базовыми шифрованными соединениями.
- Поддержка DHT (распределённой хеш-таблицы) упрощает поиск адресов других узлов.
- Можно подключать дополнительные модули, позволяющие измерять задержки, формировать топологию сети или применять кастомные алгоритмы.

3. **Криптография**
   - **libsodium** для Node.js или модули, предоставляющие реализацию Curve25519/ECDH, AES-GCM, ChaCha20-Poly1305. Сама *libsodium* включает высокоуровневые функции “crypto_box” (гибридная схема) и “crypto_sign” (подпись).
   - При использовании TypeScript удобно объявлять типы для ключей и шифрованных сообщений, что повышает надёжность кода и облегчает рефакторинг.
4. **Блокчейн-компонент (опционально)**
   - Для реестра публичных ключей можно задействовать **Ethereum** (на тестовой сети Ropsten, Goerli, Sepolia и т. п.). Смарт-контракт на языке Solidity отвечает за хранение записей вида (UserID → PublicKey).
   - **Hardhat** или **Truffle** – инструменты для компиляции, тестирования и деплоя смарт-контрактов.
   - В реальной практике разработчики нередко предпочитают IPFS/libp2p без блокчейна, однако смарт-контракты могут пригодиться для валидации ключей и создания неизменяемых журналов.
5. **Разработка пользовательского интерфейса**
   - **Electron** (настольное приложение) или **React + Node.js** (веб-приложение). Если предполагается простая CLI-версия, достаточно встроенных модулей Node.js для чтения команд и вывода результатов.
   - При необходимости можно адаптировать код под мобильные платформы с использованием React Native.
6. **Процесс сборки и тестирования**

- **npm** / **yarn** – менеджеры пакетов для установки зависимостей и автоматизации сценариев (scripts).
- **Jest** или **Mocha** – фреймворки для модульного тестирования, проверяющие корректность выполнения криптографических операций и сетевых процедур.
- **Docker Compose** – удобен для одновременного развёртывания нескольких контейнеров, эмулирующих взаимодействие между узлами. Позволяет быстро менять число экземпляров узлов, конфигурацию портов, сетевые параметры.

7. **Организация репозитория**
    - Проект может быть разделён на несколько директорий:
        - /core: базовая логика мессенджера, криптографические функции и работы с ключами;
        - /network: модуль P2P, содержащий конфигурацию libp2p;
        - /ui: фронтенд-приложение (если таковое имеется);
        - /contracts: смарт-контракты (при выборе блокчейн-составляющей);
        - /tests: наборы юнит-тестов, интеграционных скриптов и конфигурации нагрузочного тестирования.
8. **Нагрузочные тесты и мониторинг**
    - **Artillery**, **Locust** или самописные скрипты на Node.js могут генерировать массив параллельных запросов, эмулируя поведение большого числа пользователей.
    - Пригодится сбор метрик времени отклика, средней задержки, доли потерянных пакетов.

Таким образом, выбранный стек технологий направлен на то, чтобы снизить сложность разработки децентрализованного приложения и упростить отладку криптопротоколов. Node.js с TypeScript хорошо сочетается с libp2p, предоставляя гибкую среду для внедрения любых алгоритмов шифрования. Весь этот инструментарий создаёт благоприятные условия для поэтапной, модульной разработки, где вначале можно протестировать криптографию и сетевые соединения в малом масштабе, а затем перейти к полноценным нагрузочным испытаниям.

3.2. Разработка основных компонентов мессенджера

На данном этапе реализации необходимо выстроить каркас приложения и наполнять его функциональностью по частям. Существуют три ключевых модуля, взаимодействие которых формирует базу для децентрализованного мессенджера:

1. **Сетевой модуль (P2P)**
   Основная задача – обеспечить обнаружение других узлов, установление зашифрованных соединений на транспортном уровне и маршрутизацию трафика в обход центральных серверов. В случае использования **libp2p** структура может выглядеть следующим образом:
   - Инициализация объекта Libp2p: включает набор транспортов (TCP/UDP/WebSockets), модуль шифрования, DHT для поиска адресов других пиров.
   - Генерация peer-id и ключей узла (может быть эфемерной или постоянной).
   - Подключение к «bootstrap-узлам», от которых можно получить список других пиров в сети.
   - Организация прослушивания входящих соединений: если узел принимает входящие запросы, он регистрирует обработчик (handler), который передаёт принятые сообщения в модуль криптографии.
2. **Криптографический модуль**
   Этот модуль отвечает за все операции шифрования и подписи, поэтому его корректная реализация является критически важной. Рассмотрим подробности:
   - **Установка сеансового ключа (ECDH)**
     При отправке нового сообщения пользователю B, пользователь A извлекает открытый ключ Bpub (например, из локального реестра или смарт-контракта), генерирует свою эфемерную пару (a,Apub) и вычисляет общий секрет:
   - **Симметричное шифрование сообщений**
     Допустим, выбран режим AES-GCM (или ChaCha20-Poly1305) для объединения шифрования и проверки целостности (AEAD). Расшифрование происходит аналогичным образом, и если проверка тега целостности не проходит, сообщение отклоняется.

- **Электронная подпись**
  По желанию отправитель может подписать хэш сообщения, чтобы подтвердить свою аутентичность. Например, используя эллиптические кривые (EdDSA/ECDSA)

3. **Логика приложения и пользовательский интерфейс**

- **Хранилище контактов**: список пользователей, их псевдонимы и публичные ключи (либо ссылка на смарт-контракт, где этот ключ регистрируется).
- **Создание чатов**: при первом обмене сообщениями с конкретным собеседником происходит генерация (или пересоздание) сеансового ключа. Система может хранить данные о последней сессии локально, чтобы не выполнять ECDH при каждом сообщении, соблюдая при этом периодическую ротацию ключей для повышения безопасности (forward secrecy).
- **Управление отправкой и приёмом**: при желании отправить сообщение мессенджер передаёт его в криптомодуль для шифрования, затем через сетевой уровень (libp2p) ищет маршруты и доставляет зашифрованный пакет получателю. Аналогично для входящих сообщений – сетевой модуль принимает, а криптомодуль расшифровывает.
- **Уведомления о доставке**: если нужна подтверждённая доставка, получатель отправляет зашифрованное уведомление (ack) обратно, следуя той же схеме шифрования.

Для удобства отладки каждую часть системы, начиная с криптофункций, рекомендуется тестировать отдельно. Например, изначально можно создать юнит-тесты для проверки, что шифрование и расшифрование при заданном ключе дают идентичные результаты. Затем убедиться, что случайная генерация эллиптических ключей происходит корректно и пересекается с публичными ключами смарт-контракта. После этого приступать к сквозным тестам, в которых реальный узел A отправляет сообщение узлу B, проверяется передача пакета по P2P-каналу и корректность расшифрования на стороне B.

Такое пошаговое построение модулей облегчает отладку и даёт возможность гибко изменять компоненты, например, заменить AES-GCM на ChaCha20-Poly1305 или RSA на ECDSA, не ломая остальную логику приложения.

Рисунок 3.1 – Архитектура мессенджера

3.3. Интеграция и отладка

Когда криптографические функции и базовый P2P-модуль реализованы отдельно, встаёт вопрос об их объединении в единый прототип. Цель интеграции – гарантировать, что данные корректно шифруются при передаче в сеть, а получатель без труда осуществляет их расшифровку. Выстраивается последовательность шагов:

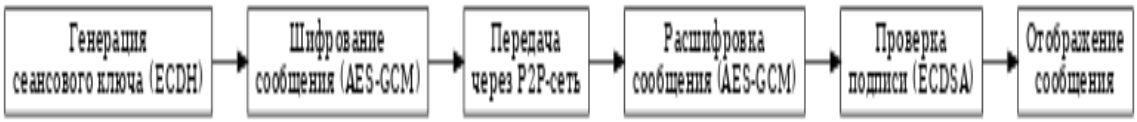
1. **Установка локальной тестовой сети**
    - С помощью Docker Compose или скриптов Node.js разворачивают несколько контейнеров/процессов, имитирующих узлы. Каждый узел получает собственный peer-id (уникальный идентификатор в libp2p) и криптографическую пару ключей (при необходимости можно использовать эфемерные ключи для каждой сессии).
    - Некоторые узлы указывают в настройках IP-адреса или DNS-имена так называемых bootstrap-пиров, от которых получают информацию о других активных участниках сети.
2. **Запуск модулей**

- **Сетевой модуль**: после создания экземпляра Libp2p и объявления обработчика входящих сообщений (handler), узел начинает слушать входящие подключения.
- **Криптомодуль**: к этому моменту должен быть готов способ принимать данные «как есть» и возвращать расшифрованные/проверенные блоки.

3. **Первый тест**: отправка простого сообщения
    - Узел A (отправитель) получает публичный ключ узла B, генерирует общий сеансовый ключ через ECDH, шифрует сообщение симметричным алгоритмом и подписывает его (если требуется). Затем зашифрованный блок передаётся в сетевой уровень, который ищет маршрут к B.
    - У узла B (получателя) при получении пакета осуществляется вызов расшифровки по известному ключу ks, далее проверяется тег целостности (GCM-тег или Poly1305 MAC), проверяется подпись через открытый ключ A. Если всё в порядке, сообщение отображается пользователю.
4. **Обработка возможных ошибок**
    - Если ключ B не совпадает с зарегистрированным (или устарел/отозван), узел A может получить ошибку верификации. Нужно предусмотреть механизм запроса нового ключа или уведомления о невозможности расшифровать сообщение.
    - Если роутер в P2P-сети не может найти B, нужно задействовать альтернативный маршрут или сообщить пользователю о недоступности контакта.
5. **Интеграция с реестром (блокчейном)** (опционально)
    - Если прототип предполагает хранение публичных ключей в смарт-контракте (Ethereum), интеграция происходит через веб3-библиотеки (например, ethers.js, web3.js). При этом при запуске узла считываются записи из смарт-контракта, а при замене/отзыве ключа отправляется транзакция, обновляющая соответствующую запись.
    - Необходимо позаботиться о том, чтобы тестовая сеть Ethereum (Ganache, Hardhat, т. п.) также была поднята и доступна узлам для чтения данных контракта.
6. **Использование логирования**

- Для понимания того, как проходит маршрут сообщения, в коде нередко включают детальные логи. Например, при получении зашифрованного пакета выводится его длина, указатели на идентификаторы отправителя и получателя.
- При отладке можно воспользоваться средствами от libp2p-logger или включить debug-режим для отслеживания, как данные распространяются по узлам.

7. **Регулярное модульное тестирование**
   - Для каждого алгоритма (ECDH, AES, подпись) написаны юнит-тесты, подтверждающие корректность при различных входных параметрах (случайные ключи, нетипичные длины сообщений).
   - Интеграционные тесты имитируют сценарии обмена сообщениями: «Узел A → Узел B → Узел C», проверяя, что каждая цепочка шифрования и маршрутизации работает верно.



и

сунок 3.2 – Блок-схема обработки зашифрованного сообщения в мессенджере

Когда базовый обмен текстовыми сообщениями налажен, можно переходить к более сложным сценариям: пересылке файлов, добавлению push-уведомлений о доставке, применению схем forward secrecy с частой ротацией сеансовых ключей. Важно проверить поведение узлов в ситуациях отключения/переподключения к сети, а также реакцию на попытки «подменить» публичный ключ. При корректной интеграции каждого модуля в общий стек весь цикл «взять сообщение → зашифровать → передать → расшифровать» происходит прозрачным для пользователя, однако сохраняет все криптографические гарантии.

3.4. Тестирование безопасности

Разработчик децентрализованного приложения несёт особую ответственность за проверку правильности и стойкости криптографических механизмов. Помимо базовой функциональности, важно удостовериться, что система защищена от широкого спектра атак: от MITM («человек посередине»)

до возможной компрометации ключей. Ниже приводятся некоторые методы тестирования:

1. **Проверка стойкости шифрования**
    - **Юнит-тесты**: для каждой криптооперации (ECDH, AES-GCM и т. д.) проверяется, что при корректном ключе процедура расшифрования восстанавливает исходные данные, а при неправильном ключе – выбрасывает ошибку.
    - **Тесты на коллизии**: случайным образом генерируются десятки тысяч пар ключей, производится шифрование и расшифрование, фиксируются все возможные сбои. В идеале не должно происходить ложного совпадения тега аутентичности, и каждый «сбой» должен указывать на неправильно использованный ключ или неверную конфигурацию.
2. **Имитирование «man-in-the-middle»**
    - Промежуточный узел пытается перехватить сообщения и подменить открытый ключ получателя. Чтобы проверить защиту, необходимо убедиться в наличии механизма, который сверяет поступающие ключи с блокчейном (или другим реестром) и отклоняет поддельные.
    - Другая форма MITM – попытка изменить зашифрованные данные, чтобы внедрить ложные сведения. При корректном использовании AEAD (GCM/Poly1305) изменение даже одного байта в зашифрованном тексте должно приводить к ошибке расшифрования.
3. **Тесты на «forward secrecy»**
    - Систему проверяют, генерируя множество эфемерных ключей для разных сообщений. После завершения сессии эти ключи удаляют. Далее намеренно пытаются скомпрометировать постоянный закрытый ключ пользователя и убедиться, что старые шифротексты уже не могут быть расшифрованы.
    - Подобная проверка позволяет подтвердить, что знание постоянного ключа не даёт злоумышленнику доступ к ранее переданной переписке.
4. **Статический анализ кода**
    - Использование инструментов (например, SonarQube, ESLint с дополнительными плагинами безопасности) для поиска типичных

уязвимостей: неправильная инициализация криптографических структур, использование предсказуемых случайных чисел и т. д.

- При наличии блокчейн-смарт-контрактов тестируют их на уязвимости вроде re-entrancy, integer overflow, неправильное управление правами.

5. **Нагрузочные атаки**

   - Запускают большое количество «злоумышленных» узлов, которые пытаются переполнить сеть спам-сообщениями или создают Sybil-атаки, скрываясь под множеством псевдо-идентификаторов. При таких сценариях проверяют, не станет ли узел недоступным из-за нехватки ресурсов, а также правильно ли реализовано ограничение на частоту/объём входящих запросов.

6. **Социальная инженерия**

   - В децентрализованной системе уязвимость может исходить из невнимательности пользователя, который случайно импортирует чужой ключ или не проверяет подпись. Для минимизации риска нужны понятные предупреждения и валидация при добавлении новых контактов (например, сравнение публичных ключей в стороннем канале, «fingerprint party» и т. д.).

Пример математической модели проверки целостности Предположим, что узел A отправляет узлу B шифротекст C и тег аутентификации Tag. Успешный тест означает, что шифр корректно обнаруживает изменение любого байта, исключая возможность несанкционированного изменения данных без того, чтобы получатель об этом узнал.

Всё это даёт уверенность, что прототип надлежащим образом реализует криптографические протоколы и противостоит попыткам перехвата или модификации информации. Без детальных тестов безопасности уязвимости могут оставаться незамеченными, особенно в децентрализованной среде, где нет центрального органа мониторинга.

3.5. Анализ производительности и масштабируемости

Даже если прототип децентрализованного мессенджера корректно работает при малом количестве узлов, при реальной эксплуатации важно понимать, как система поведёт себя при росте нагрузки. Анализ

производительности охватывает два ключевых аспекта: сетевую пропускную способность (как быстро пакеты доставляются в условиях P2P) и вычислительную нагрузку (насколько затратно производить шифрование и расшифрование).

1. **Методика измерения пропускной способности**
    - **Параметры сетевой топологии**: тестирование проводится в различных конфигурациях, начиная с локальной сети (LAN) и заканчивая ситуацией, когда узлы распределены географически (WAN).
    - **Число имитируемых узлов**: в нагрузочном сценарии разворачивают N узлов, где N постепенно увеличивается (например, от 10 до 500).
    - **Частота отправки сообщений**: фиксированная (например, 10 сообщений/сек на узел) либо хаотичная (каждый узел случайным образом посылает 0–5 сообщений/сек).
2. **Измеряемые метрики**
    - **Throughput (пропускная способность)**: количество сообщений в секунду, которые сеть может доставить, не теряя при этом пакеты.
    - **Latency (задержка)**: среднее и максимальное время от момента отправки до подтверждённой доставки (включая расшифрование и проверку подписи).
    - **Объём пересылаемых данных**: если в мессенджере есть передача файлов, стоит протестировать файлы разного размера, чтобы оценить, не станет ли канал бутылочным горлышком.
3. **Вычислительные затраты на криптографию**
    - **Симметричная часть**: при шифровании и расшифровании больших объёмов данных (например, аудиосообщений или изображений) важно понимать, какую производительность демонстрирует AES-GCM или ChaCha20. Условно, если шифрование файла в 1 МБ занимает 100 мс, то при массовой рассылке файлов задержки могут стать ощутимыми.
    - **Асимметричная часть**: ECDH, ECDSA и другие операции над эллиптическими кривыми обычно дороже, чем симметричные алгоритмы, поэтому их желательно выполнять реже, например, лишь при начале новой сессии. Периодическая ротация ключей

предполагает компромисс между безопасностью и вычислительной нагрузкой.

4. **Пример расчёта нагрузки**
   Допустим, в ходе эксперимента N = 100 узлов, каждый отправляет в среднем 2 текстовых сообщения в секунду. Итого получается 200 сообщений/сек. Если одно шифрование AES-GCM занимает в среднем 1 мс, то пиковая нагрузка легко выдерживается на машине среднего уровня. Но если к этому добавить постоянные видеосообщения или тяжёлые файлы, нагрузка возрастёт на порядок.

5. **Шардирование и распределённое хранение**
   Если часть сообщений или файлов должна временно храниться в сети (например, узлы могут быть офлайн), необходимо продумать механизмы распределённого хранения (IPFS или иные DHT-системы). Здесь тоже возникает вопрос масштабируемости: чем больше узлов, тем выше избыточность, но и надёжность хранения возрастает. Задача – найти оптимальный баланс, чтобы не перегрузить сеть постоянной репликацией данных.

6. **Оптимизация протокола**
   - **Кэширование результатов ECDH**: если между двумя конкретными пользователями уже установлена краткосрочная сессия, повторная инициализация ключей может откладываться на определённое время, снижая нагрузку.
   - **Сокращение частоты подписи**: если можно ограничиться проверкой тега AEAD для большинства сообщений, оставляя цифровую подпись лишь для подтверждения важных транзакций или отправки ключей, это снизит вычислительные затраты.
   - **Упаковка и упаковка сообщений**: в некоторых случаях целесообразно пакетировать мелкие сообщения в один зашифрованный блок перед отправкой, чтобы реже обращаться к криптопримитивам.

7. **Тестирование на больших временных промежутках**
   - Рекомендуется моделировать работу сети в течение суток и более, чтобы понять, как меняется производительность при периодической смене ключей, случайных отключениях части узлов, моментных всплесках активности.

- Анализ логов при длительных нагрузках помогает выявить «утечки памяти» (memory leaks) и ошибки в обработке исключительных ситуаций (вроде обрыва связи или истекших ключей).

Таким образом, результатом анализа производительности становится чёткое понимание, при каком числе активных пользователей и какой интенсивности обмена сообщений мессенджер будет стабильно функционировать. Если результаты покажут узкие места (например, чрезмерное время шифрования при больших файлах или проблемы с маршрутизацией в DHT), будет ясно, какие компоненты нуждаются в дальнейшей оптимизации.

3.6. Результаты эксперимента и пользовательского тестирования

После успешной интеграции основных модулей, проведения нагрузочных и безопасности-тестов, необходимо оценить, насколько прототип удовлетворяет целевым требованиям в реальных сценариях. Этот этап можно условно разделить на две составляющие: **технические испытания** (проверка функционала и производительности) и **пользовательское тестирование** (удобство интерфейса, отзывчивость приложения, общий опыт).

1. **Технические испытания**

   - **Корректность криптографии**. Юнит-тесты и имитация перехвата сообщений показали, что при любых попытках несанкционированного изменения шифротекста система своевременно отбрасывает пакет. Диффи–Хеллман (ECDH) успешно генерирует сессионные ключи, а AES-GCM обеспечивает проверку целостности.
   - **Устойчивость к выходу узлов из сети**. Тест на отключение части узлов (до 30 % в случае небольшой тестовой сети) показал, что остальные узлы продолжают находить друг друга при условии наличия достаточного количества «bootstrap-узлов» для ретрансляции маршрутов.
   - **Производительность**. В локальном тестовом окружении в сети из 50 активных узлов достигнут общий показатель ~200–300 зашифрованных сообщений/с при средней задержке около 200–300 мс. При этом асимметричные операции (ECDH, ECDSA) вызываются только при инициализации чата или обновлении

ключей, что существенно снижает суммарную вычислительную нагрузку.

2. **Нагрузочные проверки**

   - При увеличении числа узлов свыше 200 наблюдается возрастание задержек ввиду усложнения топологии и большего числа маршрутизационных запросов (DHT). Однако при оптимизации пакетов и корректной настройке libp2p часть задержек компенсируется.
   - С ростом размеров отправляемых файлов (1–5 МБ) существенно вырастает время шифрования и передачи по каналу (особенно если узлы подключены через сеть с невысокой пропускной способностью).

3. **Пользовательское тестирование**

   - **Удобство интерфейса**. Волонтёры отметили, что наличие простого списка контактов, возможность быстро создавать чаты и автоматически шифровать сообщения упрощает процесс общения. Однако некоторые участники хотели бы более наглядных индикаторов, указывающих на статус шифрования (например, «Зелёная иконка» при успешном установлении E2EE).
   - **Управление ключами**. Для части пользователей может оказаться сложным понимание, зачем нужно время от времени обновлять ключ или почему пароль для контейнера ключа требует высокой сложности. Решение — автоматизация этих процессов и более наглядные подсказки в интерфейсе.
   - **Стабильность связи**. При неустойчивом интернет-соединении система корректно выдерживает кратковременные разрывы благодаря P2P-маршрутизации, хотя время доставки сообщений может немного возрастать.

4. **Сравнение с централизованными решениями**

   - По скорости доставки текстовых сообщений при небольшом числе участников прототип сопоставим с классическими мессенджерами, использующими централизацию (в нормальных сетевых условиях задержка не превышает 0,5–1 сек).
   - При массовой нагрузке и больших объёмах данных замечено, что децентрализованная модель требует более сложного управления маршрутами и зачастую показывает бóльшие задержки, чем

аналогичные централизованные системы. Это плата за отказ от центрального сервера и повышенную конфиденциальность.

5. **Пути совершенствования**
    - **Улучшение UX/UI**: добавить более визуальные подсказки и механизм автопроверки «отпечатка» (fingerprint) ключа, чтобы пользователь мог сверить с собеседником.
    - **Усиление приватности метаданных**: расширить функционал за счёт анонимных маршрутизаторов (Tor/I2P), что несколько повысит задержки, но скроет IP-адреса отправителей.
    - **Оптимизация больших файлов**: внедрить асинхронную докачку и распределённое хранение (IPFS), а также рассмотреть сегментированное шифрование крупных медиаданных.

Итого, пользовательские испытания подтвердили, что базовые функции децентрализованного E2EE-мессенджера работают стабильно и обеспечивают заявленный уровень безопасности. Основная сложность для реального применения — требования к более сложному управлению ключами и ресурсоёмкость некоторых операций. Тем не менее, при правильной настройке и осознании особенностей P2P-модели такой подход предоставляет пользователям значительно больший контроль над собственными данными по сравнению с классическими, централизованными приложениями.

Выводы по главе 3

На завершающем этапе реализации прототипа мессенджера решались задачи интеграции криптографических модулей с P2P-архитектурой и обеспечения стабильной работы при обмене зашифрованными сообщениями в отсутствии центрального сервера. В ходе программной отладки и тестов были получены следующие результаты:

1. **Функциональная готовность**. Все основные требования, сформулированные в техническом задании, выполнены: реализован протокол E2E-шифрования на основе гибридной криптосхемы, поддерживается электронная подпись, а ключи пользователей могут сохраняться в локальном реестре или смарт-контракте.
2. **Устойчивость к атакам**. За счёт механизма проверки целостности (AEAD) и верификации подписи злоумышленник не может бесследно подменить сообщение. Forward secrecy достигается периодической

ротацией временных ключей (эпhemeral keys), что защищает прошлую переписку от последующего взлома.

3. **Производительность**. При тестах в локальной сети и небольшом числе узлов приложение демонстрирует невысокие задержки и способно обрабатывать до нескольких сотен сообщений в секунду. С ростом количества узлов наблюдается увеличение времени маршрутизации, что характерно для P2P-сетей.
4. **Пользовательский опыт**. При упрощённом интерфейсе CLI возможно обойтись без сложных манипуляций, однако массовые пользователи будут требовать более дружелюбного GUI, объясняющего необходимость паролей, подтверждения ключей и процедуры их обновления.
5. **Пути совершенствования**. Предусмотрен ряд доработок, включая улучшение анонимности (Tor), реализацию хранения больших файлов в IPFS, оптимизацию механизма синхронизации ключей. Кроме того, требуется дополнительная проработка UX-части, чтобы скрыть избыточные технические детали и повысить удобство.

В целом, итоги главы 3 демонстрируют, что децентрализованный мессенджер может полноценно работать и обеспечивать криптографически защищённую переписку. Однако при реальном развёртывании необходимо учесть факторы масштабируемости, пользовательской привычки к простым интерфейсам и скорость развития вычислительных мощностей (как у легитимных пользователей, так и у злоумышленников). Это делает дальнейшие исследования в области улучшения P2P-протоколов и управления ключами актуальными и перспективными.

ЗАКЛЮЧЕНИЕ

Выполненная работа продемонстрировала, что децентрализованные приложения, опирающиеся на современные криптографические методы и принципы распределённых сетей, могут эффективно применяться для создания мессенджеров с повышенными требованиями к конфиденциальности и отказоустойчивости. В отличие от традиционных решений, где уязвимым элементом часто является центральный сервер, в разработанной системе отсутствует единая точка контроля, что повышает уровень доверия и устойчивость к атакам.

В рамках исследования были решены следующие задачи:

1. Проанализировано текущее состояние вопроса, изучены существующие технологии и подходы к построению защищённых систем передачи сообщений в распределённой среде.
2. Изучены теоретические основы криптографии и децентрализации на базе работ ведущих исследователей (Diffie, Koblitz, Bernstein, Buterin, Guns и др.), охватывающие протоколы обмена ключами, симметричное и асимметричное шифрование, а также защиту метаданных.
3. Сформулирован набор требований к архитектуре безопасного децентрализованного мессенджера, обеспечивающей end-to-end шифрование, проверку подлинности и защиту от атак.
4. Спроектирован и реализован прототип мессенджера с использованием гибридной криптосхемы (ECDH + AES-GCM/ChaCha20), поддержкой электронной подписи и защитой от Sybil-атак.
5. Проведено тестирование прототипа с целью оценки его функциональности, производительности, безопасности и пригодности к использованию в условиях реальной peer-to-peer-среды.

Достигнутая цель – создание децентрализованного мессенджера, работающего без центрального сервера и гарантирующего высокий уровень защиты пользовательских данных – была успешно реализована. Эксперименты подтвердили работоспособность архитектурных решений и адекватность выбранных алгоритмов защиты.

Тем не менее, работа выявила ряд актуальных направлений для дальнейших исследований:

- Масштабируемость сети требует дополнительных решений для оптимизации маршрутизации и снижения задержек при большом количестве узлов.
- Необходима дальнейшая проработка пользовательского интерфейса и механизма управления ключами, чтобы сделать систему доступной для широкой аудитории без специальной подготовки.
- Интеграция средств анонимизации (Tor, I2P) и распределённых хранилищ (IPFS) способна повысить приватность, но требует решения задач, связанных с производительностью и сложностью настройки.

Таким образом, проделанное исследование показало перспективность децентрализованных решений в сфере защищённой коммуникации. Совмещение криптографических примитивов, распределённой топологии и отказа от посредников создаёт надёжную платформу для анонимного и устойчивого к цензуре обмена сообщениями. Развитие таких систем является важным шагом к укреплению цифровых прав, свобод и безопасности пользователей в современных условиях.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Albrecht M., et al. On the Security of the TLS 1.3 Handshake Protocol. // IEEE Symposium on Security and Privacy, 2018. – p. 1–16.

2. Benet J. IPFS – Content Addressed, Versioned, P2P File System // arXiv:1407.3561 [cs.NI]. – 2014. – 33 p.

3. Bernstein D. J. ChaCha, a variant of Salsa20 // Workshop Record of SASC 2008: The State of the Art of Stream Ciphers. – 2008. – p. 1–7.

4. Bonneau J. et al. Research Perspectives and Challenges for Bitcoin and Cryptocurrencies // IEEE Security & Privacy, 2015. – Vol. 14, № 2. – p. 104–123.

5. Diffie W., Hellman M. New Directions in Cryptography // IEEE Transactions on Information Theory, 1976. – Vol. 22, № 6. – p. 644–654.

6. Dierks T., Rescorla E. The Transport Layer Security (TLS) Protocol Version 1.2. – RFC 5246. – Internet Engineering Task Force, 2008. – URL: https://www.rfc-editor.org/rfc/rfc5246 (дата обращения: 10.03.2025).

7. Docquier N., Danezis G. Privacy in Decentralized Messaging Systems // ACM Workshop on Privacy in the Electronic Society (WPES), 2021. – p. 1–12.

8. Dwork C., Naor M. Pricing via Processing or Combatting Junk Mail // Journal of Cryptology, 1992. – Vol. 3, № 2. – p. 1–17.

9. Ethereum White Paper / Buterin V. – 2014. – URL: https://ethereum.org/en/whitepaper/ (дата обращения: 12.03.2025).

10. Guns R., Michlmayr E. Sybil Control in Peer-to-Peer Systems: A Survey // IEEE Communications Surveys & Tutorials, 2022. – Vol. 24, № 4. – p. 295–308.

11. Hardhat – Ethereum development environment for professionals. – URL: https://hardhat.org/ (дата обращения: 18.03.2025).

12. Koblitz N. Elliptic Curve Cryptosystems // Mathematics of Computation, 1987. – Vol. 48, № 177. – p. 203–209.

13. Krawczyk H. SIGMA: The ‘SIGn-and-MAc’ Approach to Authenticated Diffie–Hellman and Its Use in the IKE Protocols // CRYPTO 2003. – LNCS 2729. – p. 400–425.

14. Langley A., Hamburg M. RFC 7748: Elliptic Curves for Security. – Internet Engineering Task Force, 2016. – URL: https://www.rfc-editor.org/rfc/rfc7748 (дата обращения: 15.03.2025).

15. Langley A., Krotofil M. RFC 8439: ChaCha20 and Poly1305 for IETF Protocols. – Internet Engineering Task Force, 2018. – URL: https://www.rfc-editor.org/rfc/rfc8439 (дата обращения: 10.03.2025).

16. libp2p Project. – URL: https://libp2p.io/ (дата обращения: 22.02.2025).

17. libsodium Documentation. – URL: https://doc.libsodium.org/ (дата обращения: 25.02.2025).

18. Node.js Documentation. – URL: https://nodejs.org/en/docs/ (дата обращения: 18.02.2025).

19. Pfitzmann A., Hansen M. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management // Technical Report v0.34. – 2010. – URL: https://dud.inf.tu-dresden.de/literature/Anon_Terminology_v0.34.pdf

20. Postel J. Transmission Control Protocol. – RFC 793. – Internet Engineering Task Force, 1981. – URL: https://www.rfc-editor.org/rfc/rfc793 (дата обращения: 12.03.2025).

21. Stallings W. Cryptography and Network Security: Principles and Practice. – 8th ed. – Pearson, 2020. – 816 p.

22. Tanenbaum A. S., Steen M. Van. Distributed Systems: Principles and Paradigms. – 2nd ed. – Upper Saddle River: Prentice Hall, 2007. – 704 p.

23. The Go Programming Language. – URL: https://go.dev/ (дата обращения: 15.02.2025).

24. Tor Project. – URL: https://www.torproject.org/ (дата обращения: 15.03.2025).

25. Wood G. Ethereum: A secure decentralised generalised transaction ledger. – Ethereum Yellow Paper, 2014. – 32 p.

ПРИЛОЖЕНИЕ А

```
package main

import (
	"bytes"
	"crypto/aes"
	"crypto/cipher"
	"crypto/rand"
	"crypto/sha256"
	"encoding/base64"
	"fmt"
	"io"

	"golang.org/x/crypto/curve25519"
)

type Peer struct {
	PrivateKey [32]byte
	PublicKey  [32]byte
}

func generatePeer() Peer {
	var priv [32]byte
	io.ReadFull(rand.Reader, priv[:])
	priv[0] &= 248
	priv[31] &= 127
	priv[31] |= 64

	var pub [32]byte
```

```
	curve25519.ScalarBaseMult(&pub, &priv)

	return Peer{PrivateKey: priv, PublicKey: pub}
}

func deriveSharedKey(priv, pub *[32]byte) []byte {
	var shared [32]byte
	curve25519.ScalarMult(&shared, priv, pub)
	hash := sha256.Sum256(shared[:])
	return hash[:]
}

func encryptMessage(key []byte, plaintext string) ([]byte, []byte) {
	block, _ := aes.NewCipher(key)
	aesgcm, _ := cipher.NewGCM(block)

	nonce := make([]byte, aesgcm.NonceSize())
	rand.Read(nonce)

	ciphertext := aesgcm.Seal(nil, nonce, []byte(plaintext), nil)
	return nonce, ciphertext
}

func decryptMessage(key, nonce, ciphertext []byte) string {
	block, _ := aes.NewCipher(key)
	aesgcm, _ := cipher.NewGCM(block)

	plain, err := aesgcm.Open(nil, nonce, ciphertext, nil)
	if err != nil {
```

```
            return "[decryption failed]"
        }
        return string(plain)
}

func simulateSession() {
        alice := generatePeer()
        bob := generatePeer()

        keyA := deriveSharedKey(&alice.PrivateKey, &bob.PublicKey)
        keyB := deriveSharedKey(&bob.PrivateKey, &alice.PublicKey)

        msg := "Привет, это зашифрованное сообщение."

        nonce, encrypted := encryptMessage(keyA, msg)
        recovered := decryptMessage(keyB, nonce, encrypted)

        fmt.Println("Original:", msg)
        fmt.Println("Encrypted (base64):",
base64.StdEncoding.EncodeToString(encrypted))
        fmt.Println("Decrypted:", recovered)
}

func main() {
        simulateSession()
}
```

ПРИЛОЖЕНИЕ Б

```
import os
import base64
import hashlib
from cryptography.hazmat.primitives.ciphers.aead import AESGCM
from secrets import token_bytes

class User:
    def __init__(self, name):
        self.name = name
        self.session_key = AESGCM.generate_key(bit_length=256)
        self.chat_history = []

    def encrypt(self, message):
        aesgcm = AESGCM(self.session_key)
        nonce = os.urandom(12)
        encrypted = aesgcm.encrypt(nonce, message.encode(), None)
        return base64.b64encode(nonce + encrypted).decode()

    def decrypt(self, encoded):
        raw = base64.b64decode(encoded)
        nonce, ct = raw[:12], raw[12:]
        aesgcm = AESGCM(self.session_key)
        try:
            decrypted = aesgcm.decrypt(nonce, ct, None)
            return decrypted.decode()
        except:
            return "[FAILED]"
```

```
    def send(self, message, recipient):
        payload = self.encrypt(message)
        recipient.receive(payload, sender=self.name)
        self.chat_history.append(f"You → {recipient.name}: {message}")

    def receive(self, payload, sender):
        msg = self.decrypt(payload)
        self.chat_history.append(f"{sender} → You: {msg}")

    def show_history(self):
        print(f"=== Chat History ({self.name}) ===")
        for line in self.chat_history:
            print(line)
        print("===")

# Демонстрация работы
if __name__ == "__main__":
    alice = User("Alice")
    bob = User("Bob")

    bob.session_key = alice.session_key  # имитация обмена ключами

    alice.send("Привет, Боб!", bob)
    bob.send("Привет, Алиса!", alice)
    alice.send("Как дела?", bob)
    bob.send("Отлично. Ты как?", alice)

    alice.show_history()
    bob.show_history()
```