Реферат

Выпускная квалификационная работа содержит 106 с., 16 рис., 21 таблиц, 38 источников, 1 приложение.

ЗАЩИЩЁННЫЙ АСИНХРОННЫЙ ОБМЕН, НЕДОВЕРЕННЫЙ ПОСРЕДНИК, СКВОЗНОЕ ШИФРОВАНИЕ, ЭЛЕКТРОННАЯ ПОДПИСЬ, ЦИФРОВАЯ ВАЛЮТА, УСТОЙЧИВОСТЬ К БЛОКИРОВКЕ, КРИПТОВАЛЮТНЫЕ ТРАНЗАКЦИИ

Объект исследования – процессы защищённого асинхронного информационного обмена через недоверенные посредники.

Предмет исследования – модели, протоколы и программные средства такого обмена для сообщений и криптовалютных транзакций.

Цель работы – спроектировать и реализовать транспортно-независимую систему безопасного асинхронного обмена сообщениями и криптовалютными транзакциями через недоверенного посредника.

Построена модель угроз по методике ФСТЭК России и сформированы требования. Спроектирована четырёхслойная архитектура и реализован прототип на языке Go.

Практическая значимость состоит в применимости прототипа для защищённой координации трансграничных криптовалютных расчётов.

Содержание

Мультимедийная презентация 15 кадров (диск CD-R 700 Mb)

Общее количество листов иллюстративной части: 15 листов.

Введение

Развитие трансграничной электронной торговли и ограничение традиционных каналов международных расчётов привели к росту применения криптовалют, прежде всего стейблкоинов, во внешнеэкономической деятельности. С 1 сентября 2024 года в Российской Федерации действует экспериментальный правовой режим, допускающий использование цифровой валюты во внешнеторговых расчётах под надзором Банка России. Частный бизнес, ведущий такие расчёты, вынужден координировать платёжные операции – согласовывать и подписывать транзакции, обмениваться платёжными реквизитами – по каналам, которые должны оставаться доступными даже при блокировке привычных средств связи.

Существующие средства обмена либо требуют доверенного централизованного посредника и одновременной доступности обеих сторон, либо перекладывают передачу транзакционных артефактов на небезопасные каналы общего назначения. Централизованный координатор образует единую точку блокировки, изъятия и компрометации, а ручная передача через мессенджеры или электронную почту не обеспечивает сквозной защиты, сокрытия метаданных и устойчивости к цензуре. В условиях 2025–2026 годов, когда фильтрация трафика и блокировки усиливаются, для бизнеса это становится вопросом непрерывности расчётов, а не только приватности.

В работе предлагается система защищённого асинхронного обмена через недоверенного посредника – произвольный носитель, способный лишь хранить байты (облачная папка, расшаренный документ, WebDAV-хранилище). Носитель не наделяется доверием: конфиденциальность, целостность, подлинность и анонимность получателя обеспечиваются криптографией поверх него, а непрерывность при блокировке – работой

поверх нескольких взаимозаменяемых носителей. Обмен криптовалютными транзакциями поддерживается как штатный прикладной сценарий.

Актуальность работы определяется тремя факторами. Кто и где: решение предназначено частному бизнесу, ведущему трансграничные расчёты стейблкоинами в рамках экспериментального правового режима. Почему сейчас: в 2024–2026 годах одновременно легализован крипто-канал внешнеторговых расчётов и усилены сетевые блокировки, что обостряет потребность в устойчивом защищённом канале координации. Что будет лучше: предлагаемая система устраняет доверенного посредника и сохраняет работоспособность при блокировке отдельного носителя, скрывая при этом граф общения сторон.

Цель работы – спроектировать и реализовать транспортно-независимую систему безопасного асинхронного обмена сообщениями и криптовалютными транзакциями через недоверенного посредника. Для достижения цели поставлены задачи:

– провести анализ предметной области и существующих решений, выявить нерешённую задачу;

– построить модель угроз и сформировать требования к системе;

– спроектировать транспортно-независимую слоистую архитектуру и формат защищённого сообщения;

– реализовать программный прототип: абстракцию носителя, криптографический слой и протокол обмена;

– реализовать прикладной сценарий кооперативного подписания криптовалютной мультиподписи;

– выполнить тестирование и оценить свойства безопасности и устойчивости;

– провести экономическое обоснование и проработать вопросы безопасности и охраны труда.

Научная новизна заключается в формальной модели недоверенного носителя с минимальным контрактом и в транспортно-независимом протоколе асинхронного защищённого обмена, архитектура которого расширяема одновременно по двум осям – по носителю и по типу полезной нагрузки.

Практическая значимость состоит в применимости прототипа для защищённой координации трансграничных криптовалютных расчётов в условиях ограничений и в переиспользуемости ядра системы. Диссертация состоит из введения, шести глав, заключения, списка использованных источников и приложения.

1 Нормативные ссылки

В настоящей выпускной квалификационной работе использованы ссылки на следующие нормативные документы:

1 ГОСТ Р ИСО/МЭК 27001-2021. Информационная технология. Методы и средства обеспечения безопасности. Системы менеджмента информационной безопасности. Требования.

2 ГОСТ Р ИСО/МЭК 27002-2021. Информационные технологии. Методы и средства обеспечения безопасности. Свод практик в области мер обеспечения информационной безопасности.

3 ГОСТ Р 34.10-2012. Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи.

4 ГОСТ Р 34.11-2018. Информационная технология. Криптографическая защита информации. Функция хэширования.

5 ГОСТ 7.1-2003. Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание. Общие требования и правила составления.

6 ГОСТ 7.32-2017. Система стандартов по информации, библиотечному и издательскому делу. Отчёт о научно-исследовательской работе. Структура и правила оформления.

7 ГОСТ 12.0.003-2015. Система стандартов безопасности труда. Опасные и вредные производственные факторы. Классификация.

8 ГОСТ 12.1.030-81. Система стандартов безопасности труда. Электробезопасность. Защитное заземление, зануление.

9 СанПиН 1.2.3685-21. Гигиенические нормативы и требования к обеспечению безопасности и (или) безвредности для человека факторов среды обитания.

2 Термины и определения

В настоящей выпускной квалификационной работе применяются следующие термины с соответствующими определениями:

1 **Информационная безопасность** – состояние защищённости информации, при котором обеспечиваются её конфиденциальность, целостность и доступность.

2 **Конфиденциальность информации** – обязательное для выполнения лицом, получившим доступ к информации, требование не передавать её третьим лицам без согласия обладателя информации.

3 **Шифрование** – обратимое преобразование информации с использованием криптографического ключа в целях сокрытия её содержания.

4 **Криптографический ключ** – параметр алгоритма криптографического преобразования, обеспечивающий выбор одного преобразования из совокупности возможных для данного алгоритма.

5 **Сквозное шифрование** – шифрование данных на стороне отправителя с расшифрованием только на стороне получателя, при котором промежуточные узлы передачи не имеют доступа к открытому тексту.

6 **Прямая секретность** – свойство криптографического протокола, при котором компрометация долговременного ключа не позволяет раскрыть содержание сообщений, переданных до момента компрометации.

7 **Электронная подпись** – информация в электронной форме, присоединённая к подписываемой информации или иным образом связанная с ней и используемая для определения лица, подписавшего информацию.

8 **Цифровая валюта** – совокупность электронных данных, которые могут быть приняты в качестве средства платежа или инвестиций и в отношении которых отсутствует обязанное перед их обладателем лицо.

3 Сокращения

БЖД – безопасность жизнедеятельности

ВКР – выпускная квалификационная работа

ВЭД – внешнеэкономическая деятельность

ИБ – информационная безопасность

ПДн – персональные данные

ЦФА – цифровые финансовые активы

ЭП – электронная подпись

ЭПР – экспериментальный правовой режим

AEAD – Authenticated Encryption with Associated Data, аутентифицированное шифрование с присоединёнными данными

DH – Diffie–Hellman, протокол выработки общего ключа Диффи – Хеллмана

DPI – Deep Packet Inspection, глубокая инспекция пакетов

ERC-20 – стандарт токенов в сети Ethereum

EVM – Ethereum Virtual Machine, виртуальная машина Ethereum

HKDF – HMAC-based Key Derivation Function, функция вывода ключа на основе HMAC

MITM – Man-in-the-Middle, атака «человек посередине»

PSBT – Partially Signed Bitcoin Transaction, частично подписанная транзакция Bitcoin

RPC – Remote Procedure Call, удалённый вызов процедур

TUI – Text User Interface, текстовый интерфейс пользователя

4 Анализ методов передачи криптовалютных сообщений

4.1 Задача защищённого асинхронного обмена через недоверенного посредника

Рассматриваемая задача состоит в том, чтобы обеспечить двум или более сторонам защищённый обмен сообщениями и криптовалютными транзакционными данными в условиях, когда стороны не могут или не хотят опираться на доверенного централизованного посредника, не обязаны находиться в сети одновременно, а канал связи может частично или полностью блокироваться. Обмен ведётся через недоверенный посредник – произвольный носитель, способный лишь хранить и выдавать именованные порции байтов: каталог облачного хранилища, расшаренный документ, ресурс по протоколу WebDAV.

Носитель рассматривается как активный противник: он может читать, изменять, удалять, переупорядочивать, повторять и блокировать данные, а также анализировать метаданные обращений. Поэтому все свойства безопасности – конфиденциальность, целостность, подлинность, защита от повтора и анонимность получателя – должны достигаться исключительно криптографическими средствами поверх носителя. Сам носитель не наделяется ни одной функцией доверия. Такая постановка близка к исследуемой в области защищённого обмена сообщениями [1], но усилена требованием транспортной независимости и работы поверх «голого» хранилища.

Обобщённая модель обмена через недоверенного посредника приведена на рисунке 1.

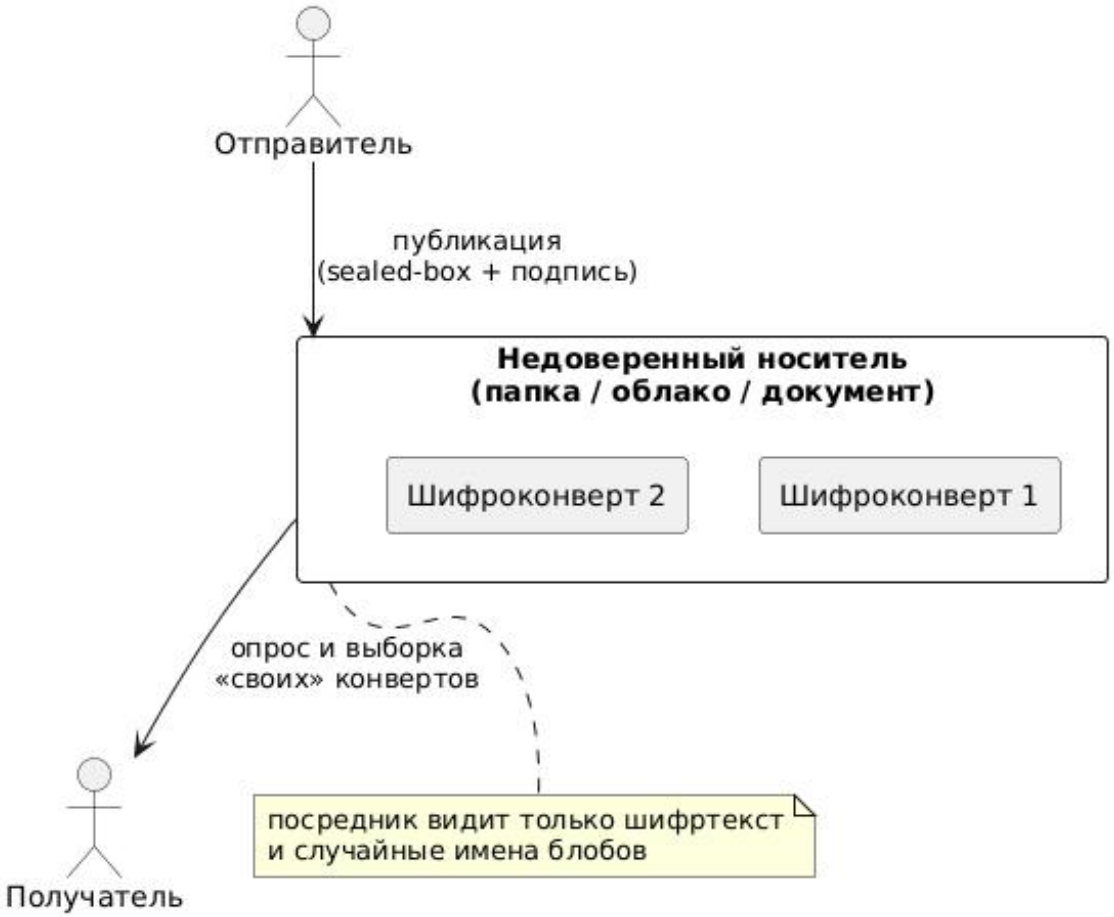

Рисунок 1 – Обобщённая модель обмена через недоверенного посредника

Из модели вытекают шесть ключевых свойств, по которым далее оцениваются существующие решения: транспортная независимость (работа поверх произвольного носителя), асинхронность, сквозное шифрование, анонимность получателя на носителе, устойчивость к блокировке и поддержка криптовалютных транзакций в качестве полезной нагрузки.

Прежде чем оценивать существующие решения, необходимо строго определить критерии сравнения. Шесть перечисленных свойств трактуются в работе следующим образом.

Транспортная независимость – способность протокола функционировать поверх любого носителя, удовлетворяющего минимальному контракту хранения именованных порций байтов, без предъявления к нему требований доверия, доступности конкретного сетевого

протокола или серверной логики. Свойство считается выполненным полностью, если смена носителя не затрагивает ни формат сообщения, ни криптографические гарантии. Свойство считается выполненным частично, если решение допускает несколько транспортов, но привязано к собственной серверной инфраструктуре.

Асинхронность – возможность доставки сообщения адресату, который в момент отправки не находится в сети: сообщение сохраняется на носителе и забирается получателем при следующем обращении. Свойство критично для трансграничной координации, где стороны разнесены по часовым поясам и не гарантируют одновременного присутствия.

Сквозное шифрование – обеспечение конфиденциальности и целостности содержимого исключительно на конечных устройствах участников, так что ни носитель, ни любой промежуточный наблюдатель не получают доступа к открытому тексту. Анонимность получателя на носителе – невозможность для оператора носителя установить, какому из участников адресован конкретный блоб, и построить граф общения по адресной информации.

Устойчивость к блокировке – сохранение работоспособности канала при недоступности, цензурировании или компрометации отдельного носителя или сетевого маршрута. Поддержка криптовалютных транзакций – наличие штатного механизма передачи транзакционных артефактов (реквизитов и подписей) как полезной нагрузки, а не только текстовых сообщений. Перечисленные критерии используются далее как единая шкала оценки для всех рассматриваемых классов решений и сводятся в сравнительную таблицу в конце раздела.

4.2 Концепция тайника и хранилища как транспорт

В основе предлагаемого подхода лежит классическая концепция мёртвого тайника: отправитель оставляет сообщение в заранее условленном

месте, а получатель забирает его позже, причём стороны не встречаются и не находятся на связи одновременно. Перенос этой схемы в цифровую среду означает, что «местом» становится произвольное хранилище, способное лишь принимать и выдавать именованные порции байтов. Такая модель естественно даёт асинхронность и снимает требование одновременного присутствия сторон.

Выбор «голого» хранилища в качестве транспорта обоснован несколькими практическими соображениями. Во-первых, облачные папки, ресурсы WebDAV и расшаренные документы повсеместны и уже используются бизнесом для легитимных задач, поэтому обращение к ним не выглядит как использование средства связи и обладает низкой заметностью. Во-вторых, выборочная блокировка такого хранилища затруднена, поскольку затронула бы и легитимный деловой трафик. В-третьих, при недоступности одного хранилища его легко заменить другим, что и лежит в основе устойчивости к блокировке.

Требование к хранилищу при этом минимально – хранить и выдавать именованные байты, не выполняя никакой протокольной логики и не наделяясь доверием. Именно эта минимальность формализуется далее как контракт носителя и обеспечивает транспортную независимость: один и тот же протокол работает поверх любого хранилища, удовлетворяющего контракту.

Современные защищённые мессенджеры обеспечивают сильное сквозное шифрование. Эталоном является протокол Signal, сочетающий асимметричное согласование ключей X3DH [2] для установления сессии с адресатом, отсутствующим в сети, и алгоритм двойного храповика Double Ratchet [3] для прямой секретности и восстановления после компрометации. Систематизация исследований по защищённому обмену [1] показывает, что эти механизмы надёжно покрывают конфиденциальность и аутентичность,

однако вопросы доставки и устойчивости к цензуре остаются за рамками криптографического ядра.

С точки зрения транспорта массовые решения привязаны к собственной инфраструктуре. Signal использует централизованные серверы и привязку к идентификатору пользователя. Briar строит одноранговую сеть поверх Tor и локальных каналов. Session и Cwtch опираются на onion-маршрутизацию или собственную сеть сервис-нод. SimpleX отказывается от идентификаторов пользователей, но требует специализированных релейных серверов своего протокола. Ни одно из этих решений не работает поверх произвольного недоверенного хранилища и не предназначено для передачи криптовалютных транзакционных артефактов.

Криптографическое ядро протокола Signal заслуживает отдельного рассмотрения, поскольку его примитивы пригодны для заимствования. Протокол X3DH [2] решает задачу установления общего секрета с адресатом, отсутствующим в сети: получатель заранее публикует на сервере набор предключей – долговременный, подписанный среднесрочный и пул одноразовых, – а отправитель комбинирует несколько операций Диффи – Хеллмана над этими ключами и собственной эфемерной парой, получая стойкий начальный секрет даже без интерактивного рукопожатия. Это обеспечивает асинхронность установления сессии и неявную аутентификацию сторон.

Алгоритм двойного храповика [3] развивает начальный секрет в ходе диалога: симметричный храповик порождает уникальный ключ для каждого сообщения, обеспечивая прямую секретность, а храповик Диффи – Хеллмана периодически обновляет корневой ключ свежими эфемерными парами, давая восстановление после компрометации (post-compromise security). В совокупности эти механизмы дают сильные гарантии для интерактивного диалога, но опираются на сервер предключей и предполагают относительно

регулярный обмен, что плохо согласуется с моделью редкой записи в произвольное хранилище.

С точки зрения сокрытия метаданных массовые мессенджеры различаются существенно. Signal минимизирует хранимые метаданные и применяет механизм запечатанного отправителя, однако сохраняет централизованную точку и привязку к телефонному номеру. SimpleX отказывается от глобальных идентификаторов пользователей, адресуя сообщения по однонаправленным очередям, но требует собственных релейных серверов. Briar и Cwtch переносят коммуникацию в одноранговую сеть поверх Tor, что повышает устойчивость к цензуре, но не работает в офлайн-модели хранилища и требует одновременной достижимости узлов. Ни в одном из решений транспорт не сведён к контракту хранения байтов, что и отличает их от рассматриваемой задачи.

Таким образом, криптографические примитивы защищённого обмена хорошо изучены и пригодны для заимствования, однако существующие мессенджеры не отделяют протокол от транспорта и не закрывают прикладной сценарий криптовалютных расчётов.

4.3 Системы с защитой метаданных и обмен через хранилище

Отдельное направление образуют системы с защитой метаданных. Система Vuvuzela [4] обеспечивает масштабируемый обмен сообщениями, устойчивый к анализу трафика, за счёт серверов-миксеров и добавления фиктивного трафика. Однако это требует развёртывания специализированной инфраструктуры и неприменимо к модели произвольного носителя. Аналогичные академические системы метаданной приватности также опираются на выделенные серверы и миксенты.

Передача через хранилище в «ручном» виде – шифрование файла средствами PGP или age с последующей загрузкой в облако – обеспечивает конфиденциальность содержимого, но не образует протокола: отсутствуют

адресные ящики, подтверждения доставки, сборка мусора, анонимность получателя, защита от повтора и диалоговая семантика. Инструменты для разоблачителей на базе Tor-сервисов ориентированы на одностороннюю передачу через выделенную инфраструктуру организации, а не на двусторонний асинхронный обмен.

Среди систем защиты метаданных можно выделить несколько технических направлений. Сети-миксеры, к которым относится Vuvuzela [4], перемешивают сообщения на цепочке серверов и маскируют реальный трафик добавлением шума по принципам дифференциальной приватности, скрывая сам факт и интенсивность общения. Сети с отказоустойчивостью на основе DC-нетей (dining cryptographers) обеспечивают теоретико-информационную анонимность отправителя, но плохо масштабируются. Протоколы приватного информационного поиска (PIR) позволяют получателю забирать сообщение, не раскрывая серверу, какое именно, ценой значительных вычислительных затрат. Луковая маршрутизация (Tor и производные) скрывает сетевой маршрут, но не защищает от глобального анализа трафика.

Общая черта перечисленных направлений – зависимость от выделенной, специально развёрнутой и согласованно работающей инфраструктуры (миксеров, сервис-нод, релеев), которая сама становится точкой отказа и блокировки. Это прямо противоречит требованию транспортной независимости: рассматриваемая задача предполагает работу поверх уже имеющегося у сторон произвольного хранилища, не наделённого никакой протокольной логикой.

Прикладные инструменты подтверждают тот же вывод. Средства односторонней передачи файлов поверх onion-сервисов решают задачу разовой выдачи данных через выделенную точку, но не образуют двустороннего асинхронного диалога с подтверждениями и сборкой мусора. Децентрализованные протоколы обмена сообщениями поверх одноранговых

сетей переносят коммуникацию в собственный сетевой слой и требуют участия узлов сети, а не работают поверх произвольного хранилища. Ни один из подходов не сводит транспорт к контракту хранения байтов и не закрывает прикладной сценарий криптовалютных транзакций.

Следовательно, целостного протокола асинхронного защищённого обмена поверх «голого» недоверенного хранилища, пригодного для повседневной инфраструктуры, в открытом доступе не выявлено.

В сфере криптовалют ключевой кооперативной операцией является подписание транзакции несколькими владельцами – мультиподпись. Для сети Bitcoin стандарт BIP-174 [5] определяет формат частично подписанной транзакции (PSBT), которым стороны обмениваются для последовательного сбора подписей. Для сетей на базе Ethereum распространены смарт-контрактные кошельки Safe, где транзакция подписывается владельцами по схеме типизированных структурированных данных EIP-712 [6].

Кошелёк Safe реализован как смарт-контракт, хранящий список владельцев и порог m из n требуемых подписей. Для исполнения перевода формируется структура SafeTx с реквизитами (адрес назначения, сумма, данные вызова, служебные поля и монотонно растущий nonce). По правилам EIP-712 [6] из неё и доменного разделителя контракта вычисляется типизированный хеш, который независимо подписывает каждый владелец своим ключом. Собранные подписи передаются в метод execTransaction, который проверяет их соответствие порогу и исполняет перевод. Перевод стейблкоина при этом кодируется как вызов метода transfer стандарта ERC-20 во вложенных данных транзакции.

В сети Bitcoin аналогичную роль играет формат частично подписанной транзакции PSBT [5]: участники последовательно дополняют единую структуру своими входами, подписями и служебными данными, пока транзакция не станет готовой к публикации. В обоих случаях стандарт фиксирует структуру обмениваемого артефакта и порядок его дополнения, но

оставляет открытым вопрос о том, как именно этот артефакт доставляется между подписантами.

Существенно, что названные стандарты задают только формат подписываемого артефакта, но намеренно не определяют защищённый транспорт его передачи. На практике PSBT и подписи Safe пересылают произвольными средствами – через мессенджеры, электронную почту, QR-коды или съёмные носители, что не обеспечивает ни сквозной защиты, ни сокрытия самого факта согласования сделки. Раскрытие реквизитов или подмена адреса назначения на этапе передачи ведёт к прямому имущественному ущербу, а раскрытие самого факта подготовки сделки – к деанонимизации контрагентов. Этот разрыв между стандартизованным форматом и небезопасной практикой передачи и устраняется в настоящей работе.

4.4 Правовой и отраслевой контекст

Прикладной контекст работы – частный бизнес, ведущий трансграничные расчёты криптовалютой. Согласно Федеральному закону № 259-ФЗ [7], цифровая валюта в Российской Федерации не является законным средством платежа, а расчёты ею по общему правилу запрещены (статья 14). Вместе с тем с 1 сентября 2024 года действует экспериментальный правовой режим, допускающий использование цифровой валюты во внешнеторговых расчётах под надзором Банка России [8]. На практике подавляющая часть таких операций приходится на стейблкоины.

Отсюда следует разграничение, определяющее правовую чистоту работы. Проектируемая система передаёт сообщения и транзакционные артефакты (реквизиты, подписи), но не осуществляет расчётов, не хранит средства и не является обменником или платёжным сервисом. Следовательно, она не подпадает под запрет статьи 14 закона № 259-ФЗ и выступает

вспомогательным защищённым каналом координации при внешнеэкономической деятельности с криптовалютой.

Экспериментальный правовой режим [8] вводит ограниченный во времени и круге участников порядок, при котором отдельные операции с цифровой валютой во внешнеторговой деятельности допускаются под надзором Банка России. Это создаёт легальную нишу для бизнеса, ведущего трансграничные расчёты, но не отменяет общего ограничительного режима закона № 259-ФЗ [7] и не снимает потребности в защищённой координации: участникам по-прежнему необходимо согласовывать реквизиты и собирать подписи по каналам, которые остаются доступными при усилении сетевых ограничений. Преобладание стейблкоинов в таких операциях объясняется привязкой их курса к фиатной валюте, что снижает ценовой риск при отложенных во времени расчётах, характерных для асинхронной координации.

Важно отметить, что предлагаемая система не порождает самостоятельных обязанностей в части противодействия отмыванию доходов, поскольку не проводит операций и не распоряжается средствами, а лишь переносит данные между сторонами. Обязанности по идентификации контрагентов и контролю операций сохраняются за самими участниками внешнеэкономической деятельности в рамках применимого к ним регулирования. Тем самым правовая конструкция работы строится на чётком разграничении передачи данных и совершения расчётов.

Требования к защите информации формируются на основе методики оценки угроз безопасности информации ФСТЭК России [9], которая служит методическим каркасом модели угроз (раздел 2). Сертификация средства как средства криптографической защиты информации для рассматриваемого частного применения не требуется, однако архитектура предусматривает возможность замены криптографического набора на сертифицированный.

Сводное сравнение рассмотренных классов решений по шести ключевым свойствам приведено в таблице 1.

Таблица 1 – Сравнение классов решений по ключевым свойствам

Класс решения	Транспорт-независимость	Асинхронность	Анонимность	Устойчивость	Крипто-транзакции
Signal	нет	да	нет	нет	нет
Briar / Session / SimpleX	нет	частично	частично	частично	нет
Tor-сервисы	нет	да	частично	частично	нет
PGP/age + облако	да	да	нет	частично	нет
Vuvuzela	нет	да	да	нет	нет
Координация Safe/PSBT	частично	да	нет	нет	да
Предлагаемая система	да	да	да	да	да

Анализ показывает, что ни одно из существующих решений не сочетает одновременно транспортную независимость, асинхронность, сквозную защиту, анонимность получателя, устойчивость к блокировке и поддержку криптовалютных транзакций. Защищённые мессенджеры привязаны к собственному транспорту и не работают с транзакциями. Системы защиты метаданных требуют специальной инфраструктуры. Ручная передача через облако не образует протокола. Средства координации мультиподписи решают задачу формата, но не транспорта.

Для наглядности число выполненных свойств из шести по каждому классу решений приведено на рисунке 2.

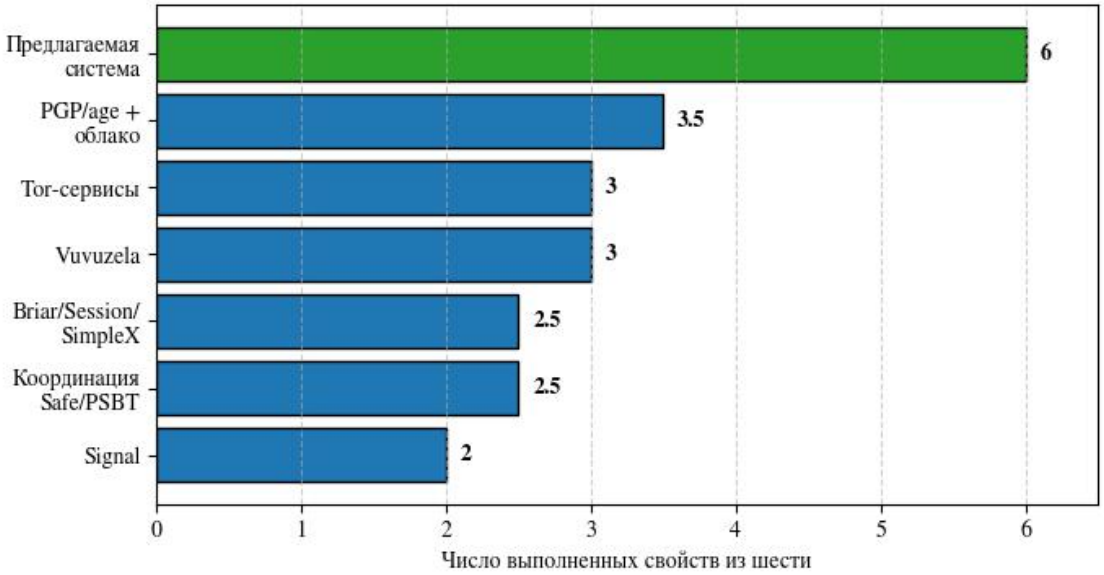

Рисунок 2 – Покрытие ключевых свойств классами решений

Отсюда вытекает нерешённая задача: разработать транспортно-независимый протокол асинхронного защищённого обмена поверх недоверенного носителя, в котором конфиденциальность, целостность, подлинность и анонимность получателя обеспечиваются криптографически, непрерывность достигается работой поверх нескольких взаимозаменяемых носителей, а тип полезной нагрузки является сменным – с реализацией криптовалютной мультиподписи в качестве штатного прикладного сценария. Решению этой задачи посвящены последующие разделы: модель угроз и проектирование (раздел 2), реализация (раздел 3) и экспериментальная оценка (раздел 4).

5 Проектирование системы передачи криптовалютных сообщений

5.1 Модель угроз и требования к системе

Модель угроз построена по структуре методического документа ФСТЭК России [9] и носит характер модели стадии проектирования. Защищаемый периметр – канал обмена с момента шифрования данных на устройстве отправителя, через хранение у недоверенного посредника, до расшифрования на устройстве получателя. Конечные устройства и закрытые ключи участников считаются доверенными и находятся вне периметра.

Защищаемые активы:

– содержимое сообщений;

– криптовалютные транзакционные данные (реквизиты, подписи);

– закрытые ключи участников;

– метаданные обмена (граф «кто – кому – когда», объёмы);

– подлинность отправителя и адресность получателя;

– доступность канала.

Ключевое негативное последствие – имущественный ущерб от подмены реквизитов транзакции и от компрометации ключей.

Границы доверия и поток данных между участниками и недоверенным носителем показаны на рисунке 3: шифрование и подпись выполняются на доверенных устройствах сторон, тогда как носитель находится вне границы доверия и является источником угроз T1–T7.

Рисунок 3 – Границы доверия и поток данных в модели угроз

Основной нарушитель – внешний оператор недоверенного носителя с полным доступом к хранилищу (чтение, запись, удаление, наблюдение паттернов). Дополнительно рассматриваются сетевой наблюдатель и недобросовестный легитимный участник. Целевой потенциал нарушителя – до среднего уровня включительно. Глобальный наблюдатель с анализом

трафика всей сети и компрометация доверенного устройства явно вынесены за область как остаточные риски.

Возможности нарушителей классифицированы по типам и потенциалу в соответствии с подходом методики ФСТЭК России [9]. Сводная характеристика нарушителей приведена в таблице 2.

Таблица 2 – Характеристика нарушителей и их возможностей

Тип нарушителя	Возможности	Потенциал	Учёт в модели
Оператор носителя	Полный доступ к хранилищу: чтение, запись, удаление, переупорядочивание и повтор блобов, наблюдение паттернов обращений	Базовый – средний	Основной
Сетевой наблюдатель	Перехват и анализ трафика между участником и носителем, блокирование соединений	Базовый – средний	Учитывается
Недобросовестный участник	Легитимный ключ и доступ к диалогу; попытка подмены реквизитов совместной сделки	Средний	Учитывается
Посторонний отправитель	Запись произвольных блобов в общий ящик без легитимных ключей	Базовый	Учитывается
Глобальный наблюдатель	Одновременный анализ трафика и обращений ко всем носителям сети	Высокий	Остаточный риск
Нарушитель на доверенном устройстве	Доступ к закрытым ключам и открытому тексту на конечном устройстве участника	Высокий	Вне периметра

Перечень актуальных угроз и базовых мер защиты приведён в таблице 3.

Таблица 3 – Актуальные угрозы и базовые меры защиты

Идент.	Угроза	Нарушитель	Базовая мера защиты
T1	Раскрытие содержимого сообщений	Носитель, наблюдатель	Сквозное AEAD-шифрование
T2	Раскрытие метаданных, деанонимизация получателя	Носитель	Случайные имена блобов, единый ящик, дополнение размеров
T3	Подмена сообщения или транзакции	Носитель, наблюдатель	AEAD и электронная подпись
T4	Подделка сообщения от чужого имени	Носитель, посторонний	Электронная подпись Ed25519

Окончание таблицы 3

T5	Повтор ранее переданного сообщения	Носитель	Уникальный идентификатор сообщения и множество принятых
T6	Переадресация подписанного содержимого	Носитель	Подпись покрывает открытый ключ получателя
T7	Блокировка или удаление сообщений	Носитель, оператор сети	Несколько носителей, неотличимость трафика
T8	Недобросовестный со-подписант (подмена реквизитов)	Участник	Независимая сверка хеша транзакции до подписи

Помимо парируемых угроз T1–T8, модель фиксирует остаточные угрозы, выходящие за границы защищаемого периметра. Угроза T9 – компрометация долговременного ключа получателя – раскрывает ранее принятые им сообщения. Эта угроза снижается переходом к схеме с предключами и двойным храповиком. Угроза T10 – компрометация доверенного конечного устройства участника – лежит вне периметра и парируется средствами защиты оконечных систем. Угроза T11 – глобальный наблюдатель, анализирующий тайминги и объёмы обращений ко всем носителям, – парируется лишь частично (фиктивный трафик, фиксированное расписание) и отнесена к направлениям развития. Эти угрозы не закрываются полностью и явно вынесены за область гарантий для целевого нарушителя.

На основе угроз сформированы требования к системе, реализуемые в проектируемой архитектуре:

– сквозное шифрование полезной нагрузки аутентифицированным шифром; носитель не получает доступа к открытому тексту;

– целостность и подлинность каждого сообщения через AEAD и электронную подпись отправителя;

– защита от повтора и от переадресации сообщения другому получателю;

– анонимность получателя на носителе и минимизация утечки метаданных;

– непрерывность обмена при блокировке отдельного носителя;

– криптоагильность – возможность замены набора алгоритмов без изменения протокола.

Перечисленные требования формализованы в виде идентифицируемых пунктов, чтобы обеспечить их трассировку к угрозам и к результатам тестирования. Полный перечень функциональных и нефункциональных требований приведён в таблице 4.

Таблица 4 – Требования к системе

Идент.	Требование	Тип
ТР-1	Сквозное шифрование полезной нагрузки аутентифицированным шифром	Функциональное
ТР-2	Электронная подпись отправителя и проверка целостности каждого сообщения	Функциональное
ТР-3	Защита от повтора ранее переданных сообщений	Функциональное
ТР-4	Привязка подписи к открытому ключу получателя (защита от переадресации)	Функциональное
ТР-5	Анонимность получателя на носителе и минимизация метаданных	Функциональное
ТР-6	Подтверждение доставки и сборка мусора доставленных конвертов	Функциональное
ТР-7	Работа поверх нескольких взаимозаменяемых носителей	Функциональное
ТР-8	Сменный тип полезной нагрузки за единым интерфейсом	Функциональное
ТР-9	Кооперативное подписание криптовалютной мультиподписи Safe	Функциональное
ТР-10	Криптоагильность: смена набора алгоритмов без изменения протокола	Нефункциональное
ТР-11	Минимизация внешних зависимостей криптографического ядра	Нефункциональное
ТР-12	Транспортная независимость: минимальный контракт носителя	Нефункциональное

Связь функциональных требований с угрозами модели прослеживается в таблице 5. Нефункциональные требования ТР-10 – ТР-12 носят архитектурный характер и обеспечивают применимость и развиваемость решения, а не парирование конкретной угрозы.

Таблица 5 – Трассировка требований к угрозам

Требование	Закрываемые угрозы
TP-1	T1
TP-2	T3, T4
TP-3	T5
TP-4	T6
TP-5	T2
TP-6	Операционная целостность диалога
TP-7	T7
TP-9	T8

5.2 Слоистая архитектура и оси расширяемости

Система построена как четыре слоя с двумя точками расширения: сменный носитель снизу и сменный тип полезной нагрузки сверху. Слой носителя (L0) хранит байты. Слой обмена (L1) задаёт формат конверта и логику ящика. Криптографический слой (L2) обеспечивает шифрование и подпись. Слой нагрузки (L3) определяет прикладные типы сообщений. Слоистая структура показана на рисунке 4.

Рисунок 4 – Слоистая архитектура системы

Ключевой принцип разделения ответственности: носитель не наделяется доверием, поэтому конфиденциальность, целостность, подлинность и анонимность обеспечиваются слоями L1 и L2, а непрерывность при блокировке – слоем L0 за счёт работы поверх нескольких носителей. Расширяемость по двум осям повторяет приём «интерфейс и сменные реализации» как на уровне носителя, так и на уровне типа нагрузки.

Ключевые проектные решения, принятые при разработке архитектуры, и их обоснование сводятся к следующему:

– минимальный контракт носителя из четырёх операций – наименьший общий знаменатель для папки, облака и расшаренного документа, что обеспечивает транспортную независимость (ТР-12) при наименьших требованиях к хранилищу;

– схема «подписать, затем зашифровать» вместо обратной – чтобы личность отправителя и его подпись были скрыты от носителя внутри шифртекста, а не доступны вместе с открытой подписью (ТР-2, ТР-5);

– sealed-box с эфемерным ключом вместо долговременного согласования – для прямой секретности со стороны отправителя без интерактивного рукопожатия, что согласуется с асинхронной моделью;

– однобайтовый идентификатор криптонабора в заголовке конверта – для криптоагильности (ТР-10) без изменения формата и протокола;

– реализация непрерывности на уровне L0 (мульти-носитель), а не в протоколе – чтобы логика ящика оставалась независимой от числа и типа хранилищ (ТР-7);

– вынесение зависимости от блокчейн-библиотек только в слой полезной нагрузки – чтобы криптографическое ядро оставалось минимальным по зависимостям (ТР-11).

Носитель описывается минимальным контрактом из четырёх операций – публикация блоба, чтение, перечисление и удаление. Это наименьший общий знаменатель для папки, облака и расшаренного

документа. Публикация атомарна с точки зрения наблюдателя перечисления (блоб виден целиком либо отсутствует), удаление идемпотентно, отсутствующий ящик перечисляется как пустой. Имена блобов – непрозрачные идентификаторы. Верхние слои используют случайные имена, чтобы носитель не извлекал из них информации.

Реализованы носители на локальной файловой системе (в том числе примонтированном облаке), на протоколе WebDAV и составной носитель. Составной носитель зеркалирует запись на несколько хранилищ и объединяет их перечисления с дедупликацией: блокировка или отказ одного хранилища не теряет сообщение и не нарушает доставку. Так требование непрерывности под блокировкой реализуется на уровне L0, не затрагивая протокол.

Семантика составного носителя определяется двумя политиками. При записи блоб зеркалируется на все доступные хранилища. Операция считается успешной, если запись прошла хотя бы на кворум хранилищ, что допускает временную недоступность части из них без потери сообщения. При чтении и перечислении результаты всех доступных хранилищ объединяются, а дубликаты с одинаковым именем устраняются, поскольку имя блоба детерминированно и одинаково на всех зеркалах. Удаление выполняется на всех хранилищах и идемпотентно: повторное удаление отсутствующего блоба не является ошибкой.

Такая модель обеспечивает доступность по принципу «достаточно одного работающего хранилища для чтения и кворума для записи» и устойчива к частичным отказам и выборочной блокировке. Поскольку имена блобов случайны, а содержимое зашифровано, добавление или удаление хранилищ не требует миграции данных и не раскрывает носителю структуру обмена. Согласованность является итоговой (eventually consistent): расхождение наборов блобов между зеркалами устраняется при последующих операциях записи и сборки мусора.

Схема работы составного носителя с зеркальной записью и слиянием при чтении приведена на рисунке 5.

Рисунок 5 – Составной носитель – зеркальная запись и слияние при чтении

5.3 Криптографический слой

Каждый участник имеет две пары ключей с разделением по назначению: пару согласования ключей на эллиптической кривой Curve25519 [10] и пару подписи Ed25519 [11]. Алгоритмы скрыты за интерфейсом криптонабора, выбираемого однобайтовым идентификатором в конверте, что обеспечивает криптоагильность. Базовый набор использует согласование ключей X25519, вывод ключа HKDF [12], аутентифицированный шифр XChaCha20-Poly1305 [13] и подпись Ed25519. Корректность и стойкость этих примитивов рассмотрены в [14].

Шифрование к получателю выполняется по схеме sealed-box с эфемерным ключом: отправитель генерирует одноразовую пару, вычисляет

общий секрет с долговременным ключом получателя и выводит из него ключ шифрования по формуле (1).

$$k = \mathrm{HKDF}(\mathrm{DH}(e_{priv}, R_{pub}), \mathrm{info}), \quad (1)$$

Эфемерный закрытый ключ уничтожается сразу после вывода, что даёт прямую секретность со стороны отправителя: последующая компрометация отправителя не раскрывает ранее отправленные сообщения. Остаточный риск – компрометация долговременного ключа получателя. Полная прямая секретность (схема с предключами в духе X3DH и двойного храповика) обозначена как направление развития. Последовательность операций при отправке и приёме приведена на рисунке 6.

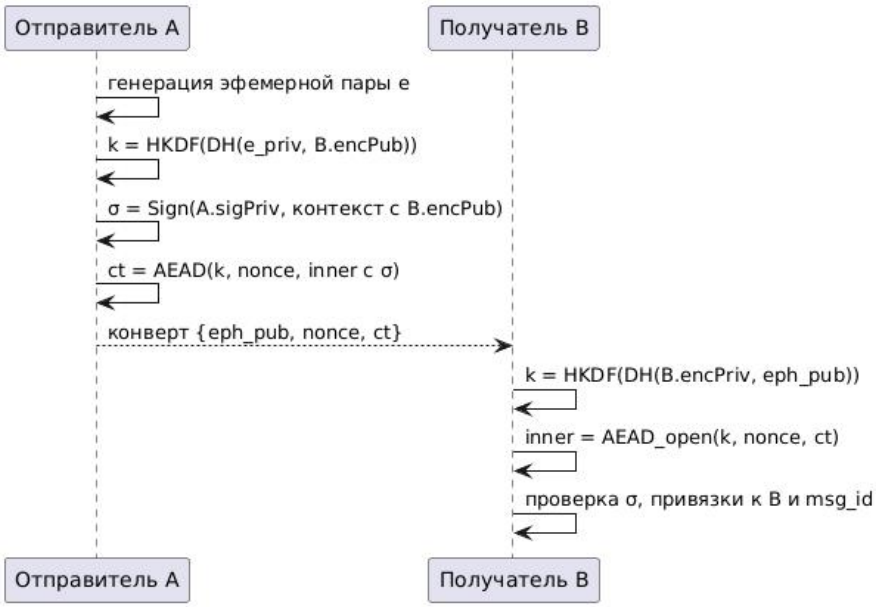

Рисунок 6 – Последовательность операций схемы sealed-box с подписью

Параметры базового криптографического набора и роль каждого примитива сведены в таблице 6. Набор подобран из современных, широко проверенных алгоритмов с реализациями в стандартной криптографической библиотеке языка реализации.

Таблица 6 – Параметры базового криптографического набора

Примитив	Алгоритм	Параметр	Назначение
Согласование ключей	X25519 (Curve25519)	ключ 32 байта	Вычисление общего секрета с получателем
Вывод ключа	HKDF-SHA-256	ключ 32 байта	Получение ключа шифрования из общего секрета
Аутентифицированное шифрование	XChaCha20-Poly1305	nonce 24 байта, тег 16 байт	Шифрование и контроль целостности тела с привязкой заголовка
Электронная подпись	Ed25519	подпись 64 байта	Подлинность отправителя и привязка к получателю

Расширенный nonce шифра XChaCha20-Poly1305 (24 байта) допускает безопасную случайную генерацию nonce без риска коллизий на практических объёмах, что важно при отсутствии общего состояния между независимыми отправками. Тег аутентификации покрывает как шифртекст, так и неаутентифицированный, но связанный заголовок конверта (associated data), что исключает подмену версии или идентификатора криптонабора.

Личность участника описывается двумя парами ключей с разделением назначения: пара согласования ключей и пара подписи. Разделение исключает использование одного ключа в двух криптографических ролях и упрощает возможную независимую ротацию. Закрытые ключи хранятся только на устройстве участника и составляют его секретный профиль. Открытые ключи вместе с идентификатором криптонабора образуют публичную карточку контакта, которой стороны обмениваются для начала переписки.

Доверие к карточке контакта устанавливается вне канала через отпечаток – короткое представление открытых ключей, удобное для сверки голосом или при личной встрече. Поскольку носитель не наделяется доверием и не выполняет функций удостоверяющего центра, привязка открытого ключа к реальному участнику обеспечивается именно внеполосной сверкой отпечатков, что защищает от подмены контакта на этапе знакомства. Ротация ключей выполняется выпуском новой карточки и

её повторной сверкой. Криптоагильность позволяет при ротации одновременно сменить и набор алгоритмов.

5.4 Версионирование формата и согласование криптонабора

Заголовок конверта содержит поле версии формата и однобайтовый идентификатор криптонабора, которые передаются открыто и входят в связанные данные при шифровании, что исключает их незаметную подмену. Идентификатор криптонабора прописан в карточке контакта, поэтому отправитель заранее знает, каким набором алгоритмов шифровать сообщение конкретному получателю. Явного интерактивного согласования не требуется. Получатель принимает конверт лишь при совпадении идентификатора набора с собственным, иначе блоб трактуется как чужой.

Такая схема обеспечивает криптоагильность без усложнения протокола: ввод нового набора алгоритмов сводится к выделению ему свободного идентификатора и публикации обновлённой карточки контакта, а поле версии формата позволяет в будущем расширять структуру конверта с сохранением совместимости. Для рассматриваемого частного применения используется единственный современный набор, а возможность замены, в том числе на отечественные алгоритмы, остаётся архитектурным резервом.

Конверт – единственный блоб на носителе со случайным именем. Применяется схема «подписать, затем зашифровать»: отправитель подписывает контекст, затем всё внутреннее содержимое (включая подпись и открытый ключ отправителя) шифруется к получателю. Поэтому носитель не видит ни содержимого, ни личности отправителя. Структура конверта приведена в листинге 1.

Выбор порядка «подписать, затем зашифровать», а не обратного, принципиален для анонимности отправителя. При обратном порядке подпись и открытый ключ подписи оказались бы вне шифртекста и были бы доступны носителю, что позволило бы связать блобы по автору и построить граф

общения. В принятой схеме подпись и идентификатор отправителя помещены внутрь шифртекста и раскрываются только легитимному получателю после успешного расшифрования. Внешний слой аутентифицированного шифрования при этом обеспечивает целостность как шифртекста, так и открытого заголовка, входящего в связанные данные, а подпись внутри – неотрекаемую подлинность отправителя с привязкой к конкретному получателю.

Листинг 1 – Структура шифроконверта

```
Заголовок (AAD):  magic(4) | version(1) | suite_id(1)
Тело:             eph_pub | nonce | ciphertext   (каждое – длина +
данные)

ciphertext = AEAD(k, nonce, inner, aad = Заголовок)
inner = { msg_id(16) | sender_sig_pub | ts | conv_id | seq |
          payload_type | payload | подпись }
```

Структура конверта с разделением на открытый заголовок, тело sealed-box и зашифрованную внутреннюю запись наглядно показана на рисунке 7.

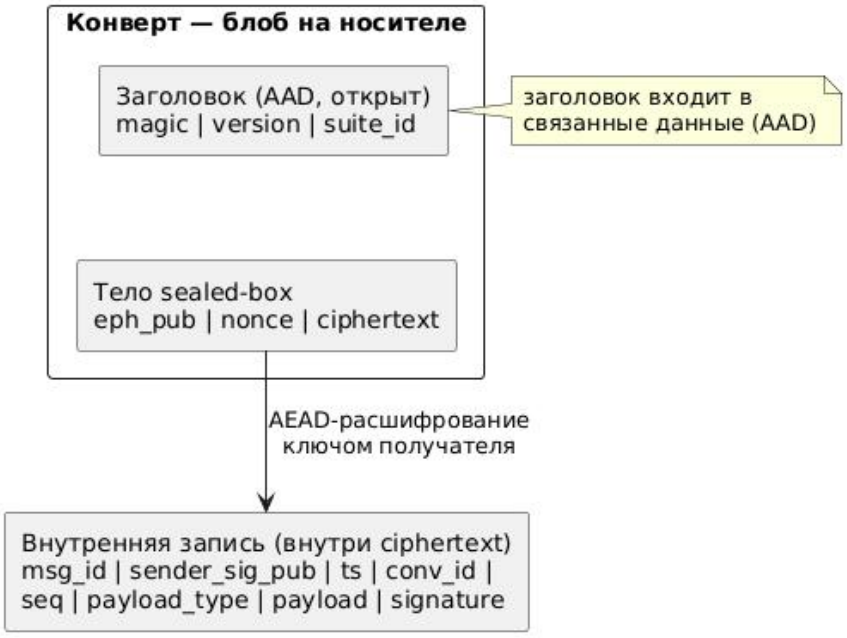

Рисунок 7 – Структура шифроконверта по уровням

Подпись вычисляется над контекстом, в который включён открытый ключ получателя, что исключает переадресацию подписанного содержимого другому адресату, по формуле (2).

$$\sigma = \mathrm{Sign}(sk_S, D \| s \| R_{pub} \| id \| m), \quad (2)$$

где D – доменный разделитель;

s – идентификатор криптонабора;

R_{pub} – открытый ключ получателя;

id – идентификатор сообщения;

m – полезная нагрузка.

Назначение и размер каждого поля конверта приведены в таблице 7. Поля заголовка передаются открыто и служат связанными данными при шифровании. Поля внутренней записи передаются только в зашифрованном виде.

Таблица 7 – Поля шифроконверта

Поле	Размер	Назначение
magic	4 байта	Сигнатура формата конверта
version	1 байт	Версия формата конверта
suite_id	1 байт	Идентификатор криптонабора (криптоагильность)
eph_pub	перем.	Эфемерный открытый ключ отправителя (sealed-box)
nonce	24 байта	Одноразовое значение для AEAD
ciphertext	перем.	Шифртекст внутренней записи
msg_id	16 байт	Уникальный идентификатор сообщения (защита от повтора)
sender_sig_pub	перем.	Открытый ключ подписи отправителя
ts	8 байт	Метка времени отправки
conv_id	16 байт	Идентификатор диалога
seq	8 байт	Порядковый номер сообщения в диалоге
payload_type	перем.	Тип полезной нагрузки
payload	перем.	Полезная нагрузка
signature	64 байта	Подпись контекста с привязкой к получателю

Логика ящика устроена так, что все участники публикуют конверты в общий каталог со случайными именами, а получатель распознаёт «свои»

конверты пробным расшифрованием. Подтверждения доставки оформляются такими же конвертами и неотличимы от сообщений. Защита от повтора обеспечивается множеством принятых идентификаторов сообщений, а контроль непрерывности – идентификатором диалога и порядковым номером. Доставленные конверты удаляются по подтверждению (сборка мусора).

Жизненный цикл конверта в ящике проходит несколько состояний, показанных на рисунке 8. После формирования конверт публикуется на носитель под случайным именем. При опросе получатель перечисляет блобы и пробно расшифровывает каждый. Успешно открытый и прошедший проверку подписи конверт принимается, если его идентификатор отсутствует во множестве уже принятых, иначе он отбрасывается как повтор. На принятый конверт получатель публикует подтверждение, после чего исходный конверт удаляется сборкой мусора. Разрыв в порядковых номерах в пределах диалога сигнализирует о возможной потере или удалении сообщения носителем.

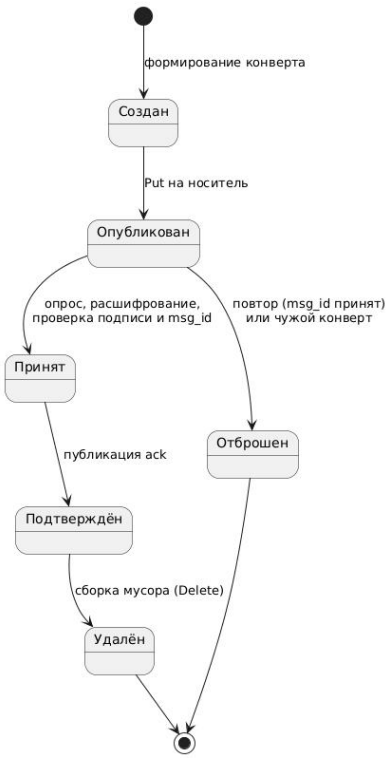

Рисунок 8 – Жизненный цикл конверта в ящике

5.5 Анонимность получателя и минимизация метаданных

Анонимность получателя на носителе обеспечивается совокупностью решений. Все участники публикуют конверты в единый общий ящик под случайными именами, не несущими адресной информации. Идентификатор диалога и порядковый номер находятся внутри шифртекста и носителю недоступны. Получатель распознаёт «свои» конверты пробным расшифрованием каждого нового блоба, поэтому на стороне носителя отсутствует какая-либо метка адресата, по которой можно было бы построить граф общения.

Подтверждения доставки оформляются такими же конвертами, как и обычные сообщения, и неотличимы от них на носителе, что не выдаёт факт прочтения. Остаточная утечка метаданных сводится к наблюдаемым носителем таймингам и объёмам обращений. Для её снижения предусмотрены дополнение размеров блобов до фиксированных классов, фиксированное расписание обращений и возможность фиктивного трафика. Полное парирование глобального наблюдателя (угроза T11) этими средствами не достигается и отнесено к направлениям развития.

Сообщения группируются в диалоги идентификатором диалога, а порядок внутри диалога задаётся монотонным порядковым номером. Это позволяет получателю упорядочивать принятые сообщения и обнаруживать пропуски или удаления по разрывам нумерации – косвенный признак вмешательства носителя. Семантика приёма «не более одного раза» обеспечивается множеством принятых идентификаторов сообщений: повторно встреченный идентификатор отбрасывается.

Подтверждения доставки выделены в отдельный служебный тип полезной нагрузки и обрабатываются тем же протоколом, что и прикладные сообщения. По получении подтверждения исходный конверт удаляется сборкой мусора, что удерживает размер ящика небольшим и ограничивает стоимость последующих опросов. Такое единообразное оформление

управляющих и прикладных сообщений упрощает протокол и сохраняет их неотличимость на носителе.

Слой нагрузки (L3) определяет прикладные типы сообщений за единым интерфейсом, что делает тип нагрузки сменным без изменения нижних слоёв. Реализованы текстовый тип и криптовалютный тип – кооперативное подписание мультиподписи Safe, рассматриваемое в разделе 3 как штатный прикладной сценарий.

Криптоагильность слоя L2 позволяет при необходимости заменить базовый набор алгоритмов на отечественный: электронную подпись по ГОСТ Р 34.10-2012 [15] и соответствующие функции хэширования и шифрования, реализованные сертифицированным средством, без изменения формата конверта и протокола. Для рассматриваемого частного применения такая замена не требуется.

Нефункциональные и правовые требования учитывают, что при обработке персональных данных обязанности оператора возникают у конечных участников в силу Федерального закона № 152-ФЗ [16], тогда как оператор носителя ввиду сквозного шифрования не имеет доступа к данным. Общие принципы защиты информации определяются Федеральным законом № 149-ФЗ [17], а организационная рамка управления информационной безопасностью – ГОСТ Р ИСО/МЭК 27001-2021 [18].

6 Реализация

6.1 Язык реализации и структура программных модулей

Прототип реализован на языке Go [19]. Выбор обусловлен наличием в стандартной библиотеке и в пакете golang.org/x/crypto выверенных криптографических примитивов (Ed25519, Curve25519, ChaCha20-Poly1305, HKDF), статической типизацией, простой кросс-компиляцией и встроенными средствами тестирования. Тяжёлая зависимость go-ethereum, необходимая для криптовалютного сценария, изолирована в отдельном подпакете, благодаря чему ядро системы остаётся компактным и легко аудируемым.

Структура программных модулей повторяет слоистую архитектуру: пакеты carrier, crypto, drop и message соответствуют слоям L0–L3, а исполняемые программы вынесены в каталог cmd. Связи между модулями показаны на рисунке 9.

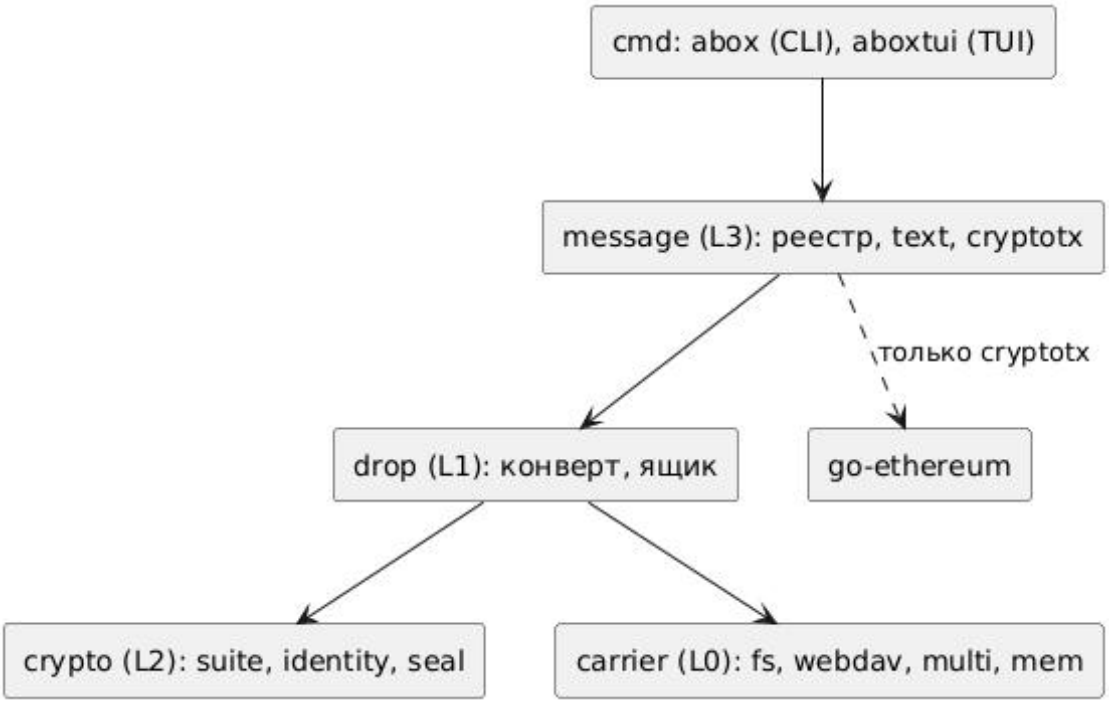

Рисунок 9 – Структура программных модулей и их зависимости

Ответственность каждого программного пакета приведена в таблице 8. Разбиение на пакеты прямо соответствует слоям архитектуры, что упрощает аудит и независимое тестирование.

Таблица 8 – Программные пакеты и их ответственность

Пакет	Слой	Ответственность
carrier	L0	Контракт носителя и реализации mem, fs, multi, webdav
crypto	L2	Криптонабор, личности, sealed-box и подпись
drop	L1	Формат конверта и почтовый ящик: публикация, опрос, подтверждение, сборка мусора
message	L3	Реестр типов нагрузки, текстовый и сервисный типы
message/cryptotx	L3	Кооперативное подписание Safe: EIP-712, ERC-20, execTransaction
keyfile	–	Формат файлов личности и карточки контакта
cmd/abox	–	Командная утилита
cmd/aboxtui	–	Терминальный интерфейс

Слой носителя задан интерфейсом из четырёх методов (листинг 2). Такой минимальный контракт позволяет подключать произвольные хранилища, не затрагивая верхние слои.

Листинг 2 – Интерфейс носителя

```
type Carrier interface {
    Put(name string, data []byte) error // атомарная публикация
    Get(name string) ([]byte, error)    // ErrNotExist, если нет
    List() ([]string, error)            // нет ящика → пусто
    Delete(name string) error           // идемпотентно
}
```

Реализованы четыре носителя. In-memory носитель хранит блобы в защищённой мьютексом ассоциативной таблице и применяется в тестах для детерминированных прогонов. Файловый носитель отображает ящик на каталог, а атомарность публикации обеспечивает записью во временный файл с последующим переименованием, что исключает наблюдение частично записанного блоба. Этот же носитель применим к примонтированному облачному хранилищу. Составной носитель зеркалирует запись на несколько хранилищ с заданным кворумом и объединяет их перечисления с

дедупликацией по имени, обеспечивая работоспособность при блокировке части хранилищ.

WebDAV-носитель работает поверх облачной коллекции (например, диска с доступом по WebDAV) и наиболее показателен с точки зрения устойчивости к нестабильному недоверенному транспорту. Запрос PUT атомарен с точки зрения наблюдателя перечисления, поэтому приём «во временный файл» не требуется. Транзиентные сбои – сетевые ошибки, код 429 (слишком много запросов) и коды 5xx – обрабатываются повторами с экспоненциальной выдержкой, причём заголовок Retry-After имеет приоритет над расчётной выдержкой (листинг 3). Создание коллекции методом MKCOL терпимо к кодам 405 и 423, означающим, что коллекция уже существует или временно заблокирована, а перечисление методом PROPFIND трактует отсутствующую коллекцию как пустой ящик согласно контракту носителя.

Листинг 3 – Повтор запросов WebDAV с учётом Retry-After

```
func (c *WebDAV) do(method, rawURL string, body []byte,
    headers map[string]string) (*http.Response, error) {
    var lastErr error
    var wait time.Duration
    for attempt := 0; attempt <= davMaxRetries; attempt++ {
        if attempt > 0 {
            backoff := davRetryBackoff << (attempt - 1)
            if backoff > davMaxBackoff { backoff = davMaxBackoff }
            if wait > backoff { backoff = wait } // Retry-After
приоритетен
            time.Sleep(backoff)
            wait = 0
        }
        resp, err := c.hc.Do(c.newRequest(method, rawURL, body,
headers))
        if err != nil { lastErr = err; continue }
        if resp.StatusCode == http.StatusTooManyRequests ||
            resp.StatusCode >= 500 {
            wait = parseRetryAfter(resp.Header.Get("Retry-After"))
            resp.Body.Close()
            lastErr = fmt.Errorf("webdav: %s: HTTP %d", method,
resp.StatusCode)
            continue
        }
```

```
        return resp, nil
    }
    return nil, lastErr
}
```

6.2 Реализация криптографического слоя

Криптонабор реализует интерфейс с операциями согласования ключей, вывода ключа, аутентифицированного шифрования и подписи. Базовый набор собран из примитивов X25519, HKDF-SHA-256, XChaCha20-Poly1305 и Ed25519. Личность хранит две пары ключей и порождает публичную карточку с отпечатком для внеполосной сверки.

Операция шифрования к получателю реализована функцией sealed-box с эфемерным ключом (листинг 4): эфемерный закрытый ключ и промежуточный общий секрет затираются сразу после использования.

Листинг 4 – Шифрование к получателю по схеме sealed-box

```
func SealTo(s Suite, recipientEncPub, plaintext, aad []byte) (*Sealed,
error) {
    ephPriv, ephPub, err := s.GenerateKEMKey()
    if err != nil { return nil, err }
    shared, err := s.DH(ephPriv, recipientEncPub)
    zero(ephPriv)
    if err != nil { return nil, err }
    key := s.KDF(shared, []byte("deaddrop/seal/v1"))
    zero(shared)
    nonce := randomBytes(s.NonceSize())
    ct := s.Seal(key, nonce, plaintext, aad)
    zero(key)
    return &Sealed{EphPub: ephPub, Nonce: nonce, Ciphertext: ct}, nil
}
```

Слой обмена собирает конверт по схеме «подписать, затем зашифровать»: формируется внутренняя запись с идентификатором сообщения, открытым ключом подписи отправителя, идентификатором диалога, порядковым номером, типом и телом нагрузки и подписью контекста (структура – листинг 1). Затем запись шифруется к получателю

функцией SealTo, а к телу добавляется открытый заголовок, играющий роль присоединённых данных.

Почтовый ящик публикует конверт под случайным шестнадцатеричным именем, а при приёме перечисляет новые блобы и пробует расшифровать каждый своим ключом. Успешное снятие аутентифицированного шифра означает «адресовано мне». Далее проверяются подпись, привязка к собственному ключу получателя и новизна идентификатора сообщения. Принятые идентификаторы запоминаются для защиты от повтора, а доставленные конверты удаляются по приходу подтверждения.

Почтовый ящик предоставляет компактный программный интерфейс над слоями L0–L2 (листинг 5): создание ящика поверх носителя и личности, регистрация контактов, отправка нагрузки заданного типа, опрос входящих и подтверждение принятых. Опрос возвращает уже расшифрованные и проверенные сообщения, скрывая от вызывающего кода криптографические детали.

Листинг 5 – Программный интерфейс почтового ящика

```
type Mailbox struct { /* носитель, личность, контакты, принятые msg_id
*/ }

func NewMailbox(c carrier.Carrier, id *crypto.Identity) *Mailbox
func (m *Mailbox) AddContact(c crypto.Contact)
func (m *Mailbox) Send(to crypto.Contact, typ string, payload []byte)
([17]byte, error)
func (m *Mailbox) Receive() ([]Inbound, error) // опрос, расшифровка,
проверка
func (m *Mailbox) Ack(in Inbound) error        // подтверждение и
сборка мусора
```

Такой интерфейс одинаково используется командной утилитой и терминальным приложением: цикл приёма сводится к периодическому вызову опроса с последующим подтверждением каждого принятого

сообщения, а отправка – к единственному вызову с указанием контакта, типа и тела нагрузки.

6.3 Криптовалютный сценарий – кооперативная мультиподпись

Прикладной тип нагрузки реализует кооперативное подписание транзакции смарт-контрактного кошелька Safe в сети на базе Ethereum. В системе сосуществуют два набора ключей: канальные ключи DeadDrop (Ed25519 и X25519), защищающие передачу, и кошельковые ключи владельцев Safe (secp256k1), которыми подписывается транзакция. Хеш транзакции вычисляется по схеме типизированных данных EIP-712. Поток кооперативного подписания приведён на рисунке 10.

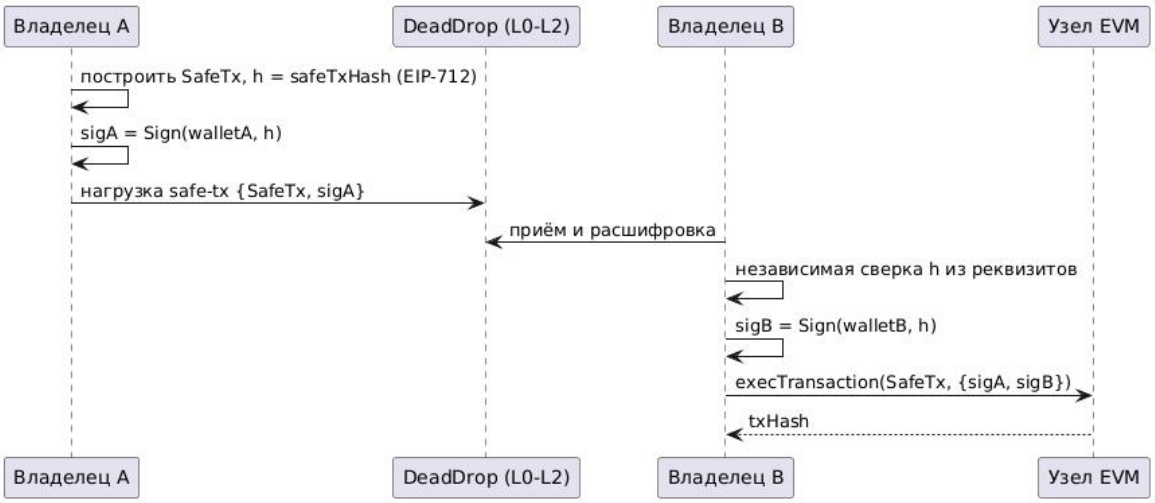

Рисунок 10 – Кооперативное подписание мультиподписи Safe через недоверенный носитель

Перевод стейблкоина строится как транзакция Safe, направленная не получателю напрямую, а контракту токена, с нулевым переводом базовой валюты и вложенными данными вызова метода transfer стандарта ERC-20 (листинг 6). Поле nonce кошелька защищает от повторного исполнения уже проведённой транзакции на стороне сети.

Листинг 6 – Построение транзакции перевода стейблкоина

```
func NewStablecoinTransfer(chainID uint64, safe, token, to
common.Address,
    amount *big.Int, nonce uint64) *SafeTx {
    return &SafeTx{
        ChainID: chainID, Safe: safe,
        To:    token,                    // вызов направлен контракту
токена
        Value: big.NewInt(0),
        Data:  ERC20Transfer(to, amount), // calldata transfer(to,
amount)
        Operation: 0,                    // CALL
        Nonce: nonce,
        Signatures: map[common.Address][]byte{},
    }
}
```

Данные вызова токена формируются как селектор метода transfer, за которым следуют дополненные до машинного слова адрес получателя и сумма перевода (листинг 7). Селектор вычисляется как первые четыре байта хеша сигнатуры метода и для transfer равен 0xa9059cbb.

Листинг 7 – Формирование calldata перевода токена ERC-20

```
func ERC20Transfer(to common.Address, amount *big.Int) []byte {
    out := append([]byte{}, erc20TransferSelector...) // 0xa9059cbb
    out = append(out, addrWord(to)...)                // адрес, 32
байта
    out = append(out, uintWord(amount)...)            // сумма, 32
байта
    return out
}
```

Подписание владельцем реализовано так, что подпись приводится к формату, ожидаемому контрактом Safe (листинг 8). Перед добавлением своей подписи каждый владелец независимо пересобирает хеш транзакции из её реквизитов и проверяет уже имеющиеся подписи, что реализует защиту от недобросовестного со-подписанта (угроза Т8). По достижении порога подписи сортируются по адресам и собираются в вызов execTransaction. Публикация в сеть выполняется отдельным узлом и находится вне защищённого канала.

Листинг 8 – Подписание транзакции Safe владельцем

```
func (tx *SafeTx) SignWith(key *ecdsa.PrivateKey) (common.Address,
error) {
    sig, err := ethcrypto.Sign(tx.Hash().Bytes(), key) // 65 байт, v
in {0,1}
    if err != nil { return common.Address{}, err }
    sig[64] += 27                                          // формат
подписи Safe
    addr := ethcrypto.PubkeyToAddress(key.PublicKey)
    tx.Signatures[addr] = sig
    return addr, nil
}
```

Перед добавлением подписи каждый владелец проверяет уже собранные подписи функцией VerifyOwners: каждая подпись восстанавливается в адрес и сверяется с адресом, под которым она хранится. Несоответствие означает поддельную или искажённую подпись и отвергается (листинг 9). Этим реализуется защита от недобросовестного со-подписанта (угроза T8).

Листинг 9 – Проверка подписей владельцев Safe

```
func (tx *SafeTx) VerifyOwners() ([]common.Address, error) {
    owners := make([]common.Address, 0, len(tx.Signatures))
    for addr, sig := range tx.Signatures {
        rec, err := tx.RecoverSigner(sig)
        if err != nil { return nil, err }
        if rec != addr {
            return nil, fmt.Errorf("подпись под %s восстановилась
в %s", addr, rec)
        }
        owners = append(owners, addr)
    }
    return owners, nil
}
```

По достижении порога подписи объединяются в формате, который ожидает метод execTransaction контракта Safe: они сортируются по адресу владельца по возрастанию и конкатенируются (листинг 10). Затем формируется вызов execTransaction с реквизитами транзакции и агрегированными подписями. Публикация вызова в сеть выполняется отдельно и находится вне защищённого канала.

Листинг 10 – Агрегирование подписей для execTransaction

```
func (tx *SafeTx) AggregatedSignatures() []byte {
    addrs := make([]common.Address, 0, len(tx.Signatures))
    for a := range tx.Signatures { addrs = append(addrs, a) }
    sort.Slice(addrs, func(i, j int) bool {
        return bytes.Compare(addrs[i].Bytes(), addrs[j].Bytes()) < 0
    })
    var out []byte
    for _, a := range addrs { out = append(out, tx.Signatures[a]...) }
    return out
}
```

6.4 Реестр типов нагрузки и сервисные сообщения

Слой полезной нагрузки построен вокруг двух интерфейсов: тип нагрузки описывает себя строковым тегом и умеет сериализоваться, а обработчик умеет декодировать данные своего тега и обрабатывать их. Реестр сопоставляет тег типа с обработчиком, так что приём сообщения сводится к выбору обработчика по тегу, прочитанному из расшифрованной внутренней записи. Добавление нового прикладного типа выполняется регистрацией обработчика и не затрагивает нижние слои, чем реализуется требование сменности нагрузки (ТР-8).

Реализованы текстовый тип, служебный тип подтверждений и криптовалютный тип кооперативного подписания Safe с тегом deaddrop/safe-tx. Единообразное оформление всех типов как полезной нагрузки одного формата конверта обеспечивает их неотличимость на носителе и единый путь обработки.

Личность и карточка контакта сохраняются в формате JSON отдельным пакетом, общим для обоих интерфейсов. Файл личности содержит идентификатор криптонабора и обе пары ключей, включая закрытые, и хранится только у владельца. Карточка контакта содержит идентификатор криптонабора и только открытые ключи и предназначена для передачи корреспонденту. Отпечаток, вычисляемый по открытым ключам

карточки, служит для внеполосной сверки подлинности контакта и защищает от его подмены на этапе знакомства.

Реализованы два интерфейса. Командная утилита abox обеспечивает генерацию личности, обмен текстом и сценарий мультиподписи (предложение и ко-подпись). Терминальный интерфейс aboxtui представляет защищённый чат с живым приёмом сообщений и действием ко-подписи Safe по сочетанию клавиш: входящая транзакция проверяется и отображается реквизитами, после чего владелец подтверждает подпись. Оба интерфейса используют одно и то же ядро и общий формат файлов личности и контакта.

Набор команд утилиты abox приведён в таблице 9. Файл личности содержит закрытые ключи и хранится только у владельца. Карточка контакта содержит открытые ключи и идентификатор криптонабора и передаётся корреспонденту для начала переписки, после чего отпечатки сверяются по доверенному внеполосному каналу.

Таблица 9 – Команды утилиты abox

Команда	Назначение
keygen	Генерация личности (пары согласования ключей и подписи) в файл
contact	Вывод публичной карточки контакта по файлу личности
send	Отправка текстового сообщения корреспонденту через дроп
recv	Приём, расшифровка и проверка входящих сообщений
safe-propose	Формирование, подпись и публикация транзакции Safe
safe-cosign	Приём, независимая сверка и ко-подпись транзакции Safe

Работа командной утилиты в сценарии генерации личностей и обмена текстом показана на рисунке 11, а в сценарии кооперативного подписания мультиподписи Safe – на рисунке 12.

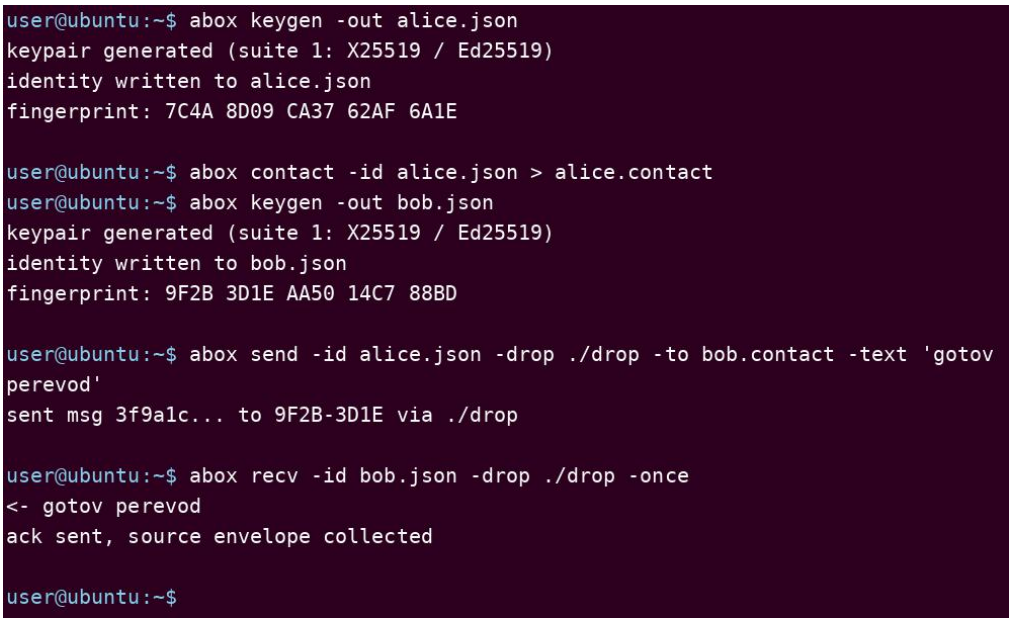

Рисунок 11 – Командная утилита abox: генерация личностей и обмен текстом

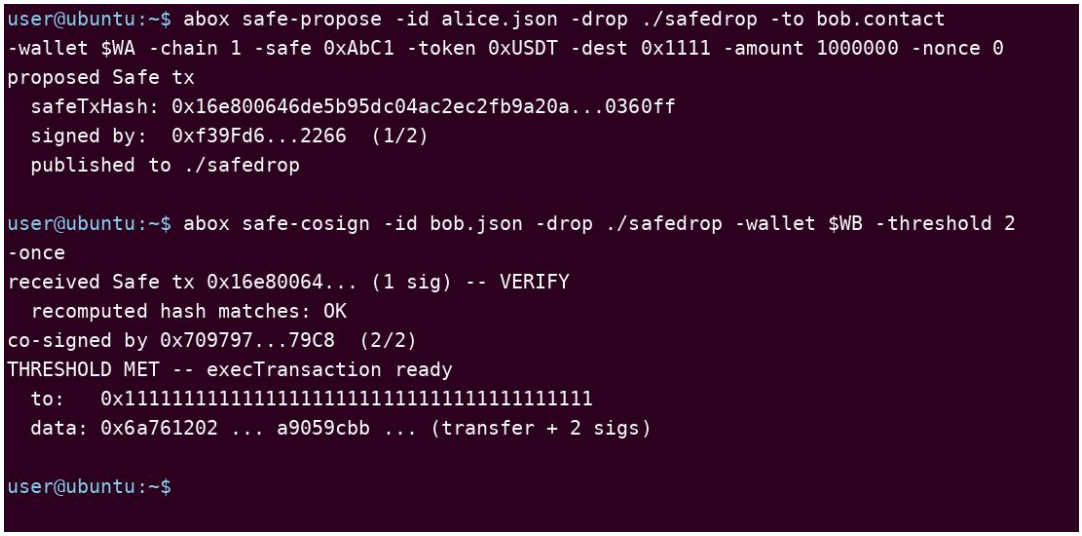

Рисунок 12 – Командная утилита abox: предложение и ко-подпись транзакции Safe

Терминальный интерфейс построен по архитектуре «модель – обновление – представление»: фоновый опрос ящика порождает события входящих сообщений, которые отображаются в ленте чата, а ко-подпись поступившей транзакции Safe выполняется по нажатию управляющего сочетания клавиш с предварительным показом реквизитов перевода для контроля оператором.

Вид терминального интерфейса во время чата с поступившей на ко-подпись транзакцией приведён на рисунке 13.

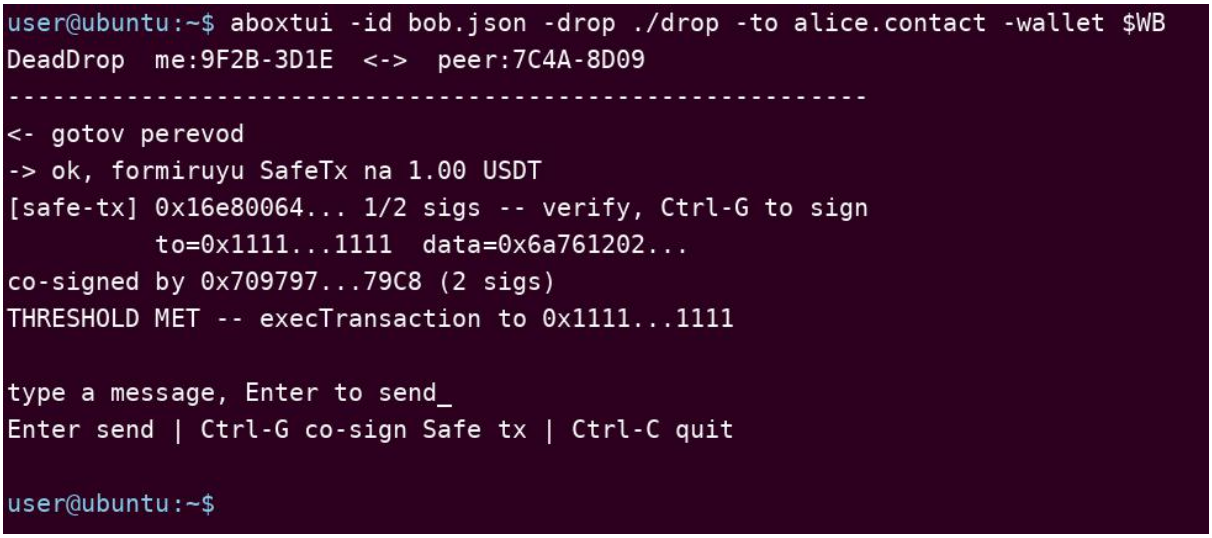

Рисунок 13 – Терминальный интерфейс aboxtui: чат и ко-подпись Safe

6.5 Меры безопасности реализации

При реализации приняты меры, снижающие риск ошибок, типичных для криптографического кода. Промежуточные секреты – эфемерный закрытый ключ, общий секрет согласования и выведенный ключ шифрования – затираются в памяти сразу после использования, что ограничивает время их присутствия и снижает риск утечки при дампе памяти. Все случайные величины (значения nonce, имена блобов, идентификаторы сообщений) получаются из криптографически стойкого генератора.

Любая неудача при расшифровании или проверке подписи трактуется единообразно как «адресовано не нам», без раскрытия причины отказа, что лишает наблюдателя оракула, по которому можно было бы различать типы ошибок. Секретные данные не попадают в журналы и в пользовательский вывод – отображаются только отпечатки и открытые сведения. Криптографические примитивы взяты из выверенных реализаций

стандартной библиотеки и пакета golang.org/x/crypto, а не реализованы заново, что исключает целый класс ошибок собственной реализации.

Криптографическое ядро системы (слои L0–L2) опирается только на стандартную библиотеку языка и на пакет проверенных криптографических примитивов golang.org/x/crypto, не привлекая сторонних зависимостей. Тяжёлая библиотека go-ethereum, необходимая для вычисления хеша EIP-712, работы с адресами и подписи secp256k1, используется исключительно в подпакете криптовалютной нагрузки и в исполняемых программах. Тем самым выполнено нефункциональное требование минимизации зависимостей ядра (ТР-11): аудит и возможная сертификация криптографической части не затрагиваются прикладной зависимостью.

Сборка и прогон тестов выполняются штатными средствами языка – командами go build для всех пакетов и go test для всего дерева. Требуется версия языка 1.25 или выше. Кросс-компиляция в статический исполняемый файл позволяет разворачивать утилиту без внешних зависимостей времени выполнения, что удобно для целевого сценария работы в ограниченной среде.

7 Тестирование и оценка результатов

7.1 Методика тестирования

Тестирование выполнено на трёх уровнях. Модульные и контрактные тесты проверяют каждый слой в отдельности: единый контракт-тест прогоняется против всех реализаций носителя, отдельные тесты – против криптографического слоя и протокола обмена. Интеграционные сценарии используют in-memory носитель, что делает прогон детерминированным и быстрым. Криптовалютный сценарий проверяется на корректности вычисления хеша и подписи, а полный прогон кооперативного подписания – на локальной тестовой сети.

Критерии приёмки заданы свойствами безопасности из модели угроз:

– сообщение, не адресованное стороне, не должно расшифровываться;

– искажённый или поддельный конверт должен отвергаться;

– повтор не должен приниматься повторно;

– подписанное содержимое не должно переадресовываться;

– контрольный пример мультиподписи должен давать корректный вызов on-chain.

Тестовая среда не требует внешней инфраструктуры: контрактные и интеграционные тесты используют in-memory носитель, что обеспечивает детерминированность и быстроту прогона и исключает влияние сети на результат. Файловый и WebDAV-носители проверяются отдельно – первый против каталога, второй против совместимого WebDAV-сервера. Криптовалютный сценарий проверяется как на уровне вычислений (хеш и подпись), так и полным прогоном кооперативного подписания на локальной тестовой сети EVM. Соответствие проверок требованиям и угрозам прослеживается в таблице 10.

Таблица 10 – Трассировка проверок к требованиям и угрозам

Требование	Проверка	Угроза	Результат
ТР-1	Round-trip конверта, чужой получатель	Т1	пройден
ТР-2	Искажение конверта, подделка подписи	Т3, Т4	пройден
ТР-3	Защита от повтора	Т5	пройден
ТР-4	Привязка подписи к ключу получателя	Т6	пройден
ТР-5	Случайные имена, единый ящик	Т2	пройден
ТР-6	Подтверждение и сборка мусора	–	пройден
ТР-7	Зеркалирование и слияние при отказе	Т7	пройден
ТР-9	Хеш, подпись и сборка execTransaction	Т8	пройден

Результаты функционального тестирования сведены в таблицу 11. Все тесты пройдены.

Таблица 11 – Результаты функционального тестирования

Проверка	Что подтверждается	Результат
Контракт носителя	Put/Get/List/Delete на mem, fs, multi, webdav	пройден
Зеркалирование и слияние	Кворум записи, чтение при отказе части хранилищ	пройден
Round-trip конверта	Шифрование и расшифрование сообщения	пройден
Чужой получатель	Сообщение не расшифровывается посторонним	пройден
Искажение конверта	Изменённый шифртекст отвергается (Т3)	пройден
Подделка подписи	Конверт с неверной подписью отвергается (Т4)	пройден
Защита от повтора	Повторный конверт не принимается (Т5)	пройден
Подтверждение и сборка мусора	Доставленный конверт удаляется	пройден
Хеш SafeTx	Детерминизм и чувствительность к реквизитам	пройден
Подпись и восстановление	Адрес владельца восстанавливается из подписи	пройден
Сборка execTransaction	Корректный селектор и порядок подписей	пройден

Отдельно подтверждено, что отправитель не доставляет сообщение самому себе, поскольку его собственные конверты не расшифровываются его ключом, а подтверждения неотличимы от обычных сообщений на стороне носителя.

7.2 Тестирование устойчивости носителя

Свойство непрерывности под блокировкой проверено на составном носителе. В сценарии зеркалирования сообщение публикуется на несколько хранилищ, после чего часть из них объявляется недоступной. Подтверждается, что сообщение по-прежнему читается через оставшиеся хранилища, а его доставка не нарушается. Слияние перечислений с дедупликацией по имени гарантирует, что одно и то же сообщение, присутствующее на нескольких зеркалах, доставляется получателю однократно.

Для WebDAV-носителя проверена корректная обработка нестабильного транспорта: повтор запросов при кодах 429 и 5xx с учётом заголовка Retry-After, терпимость создания коллекции к кодам 405 и 423 и трактовка отсутствующей коллекции как пустого ящика. Эти проверки подтверждают, что отказ или временная блокировка отдельного хранилища не приводит к потере сообщений и не нарушает работу протокола, чем реализуется требование ТР-7 и парируется угроза Т7 в пределах, заданных моделью.

В контрольном примере участник А формирует и подписывает перевод стейблкоина с мультиподписного кошелька с порогом два, передаёт транзакцию через файловый носитель, а участник В независимо сверяет хеш, ставит вторую подпись и собирает вызов execTransaction. Сокращённый вывод приведён в листинге 11.

Листинг 11 – Вывод контрольного примера ко-подписи Safe

```
proposed Safe tx
  safeTxHash:
0x16e800646de5b95dc04ac2ec2fb9a20a6656b5225746d7cc0ac86f4f4c0360ff
  signed by:  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
received Safe tx ... (1 sig) — VERIFY
co-signed by 0x70997970C51812dc3A010C7d01b50e0d17dc79C8; now 2/2
THRESHOLD MET — execTransaction ready
  to:   0x1111111111111111111111111111111111111111
  data: 0x6a761202 ... a9059cbb ... (transfer + две подписи)
```

Селектор вызова 0x6a761202 соответствует методу execTransaction контракта Safe, во вложенных данных присутствует селектор перевода токена 0xa9059cbb и две конкатенированные подписи владельцев. Это подтверждает корректность вычисления хеша по EIP-712, формата подписей и сборки итогового вызова. На всём протяжении носитель не получает доступа к реквизитам перевода и к составу подписантов.

Накладные расходы протокола складываются из постоянной добавки на каждое сообщение и из стоимости криптографических операций. Постоянная добавка образована открытым заголовком конверта, эфемерным открытым ключом, значением nonce и тегом аутентификации, а также служебными полями внутренней записи – открытым ключом подписи, идентификаторами сообщения и диалога, порядковым номером, меткой времени и подписью. Суммарно это порядка двухсот с небольшим байтов на сообщение независимо от длины полезной нагрузки. Для текстовой переписки и компактных транзакционных артефактов такая добавка пренебрежимо мала относительно типичного объёма блоба облачного хранилища.

Распределение постоянных накладных расходов по полям конверта приведено на рисунке 14. Основной вклад дают подпись (64 байта) и открытые ключи.

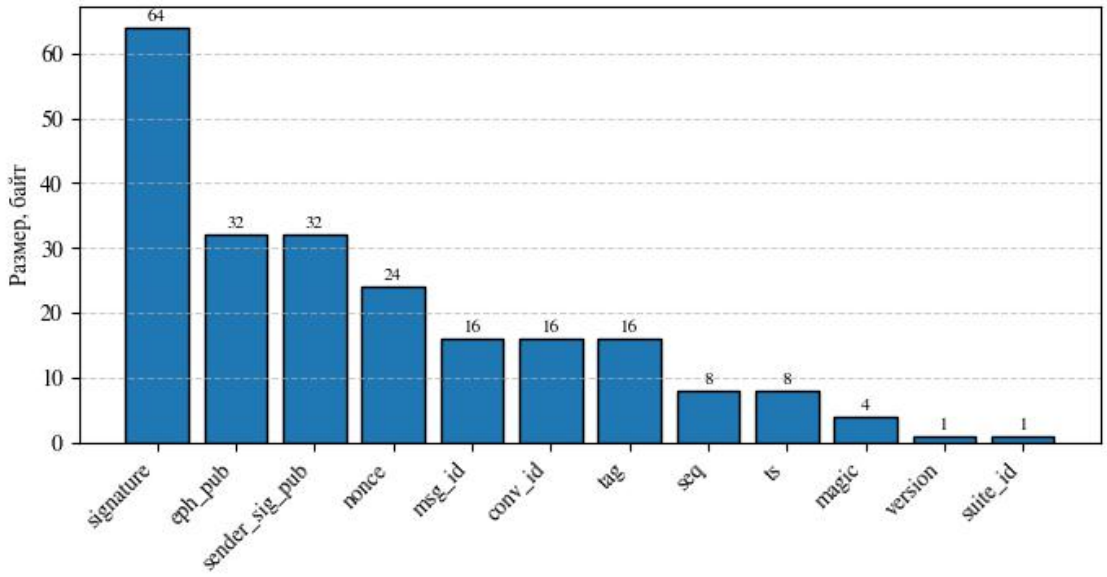

Рисунок 14 – Распределение постоянных накладных расходов конверта по полям

Стоимость криптографических операций при отправке и приёме одного сообщения составляют одно согласование ключей на кривой, один вывод ключа и одна операция аутентифицированного шифрования, а также формирование и проверка одной подписи. На современном процессоре эти операции занимают доли миллисекунды и пренебрежимо малы по сравнению с задержкой обращения к носителю по сети.

Определяющий вклад в задержку доставки вносит не вычисление, а модель опроса: получатель забирает сообщения при периодическом обращении к носителю, поэтому средняя задержка доставки составляет около половины интервала опроса. Уменьшение интервала снижает задержку, но увеличивает частоту обращений и тем самым заметность трафика – это осознанный компромисс между оперативностью и скрытностью. Стоимость одного опроса линейна по числу неподтверждённых блобов в ящике, поскольку получатель пробно расшифровывает каждый новый блоб. Своевременное подтверждение и сборка мусора удерживают ящик небольшим и тем ограничивают эту стоимость.

Линейная зависимость средней задержки доставки от интервала опроса показана на рисунке 15. Она позволяет подобрать интервал под требуемый компромисс между оперативностью и скрытностью.

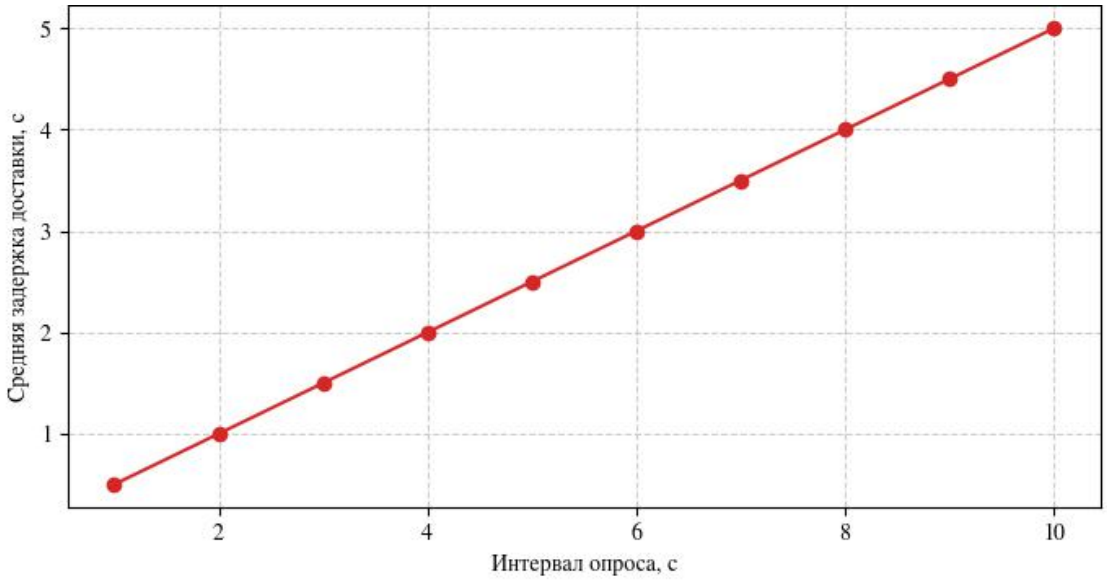

Рисунок 15 – Средняя задержка доставки в зависимости от интервала опроса

7.3 Оценка свойств безопасности

Сопоставление свойств безопасности, реализующих их механизмов и закрываемых угроз из модели ФСТЭК [9] приведено в таблице 12.

Таблица 12 – Свойства безопасности и закрываемые угрозы

Свойство	Механизм	Угроза	Подтверждено
Конфиденциальность	Сквозное AEAD-шифрование	T1	да
Анонимность получателя	Случайные имена, единый ящик, пробное расшифрование	T2	да
Целостность	AEAD и электронная подпись	T3	да
Подлинность отправителя	Подпись Ed25519	T4	да
Защита от повтора	Идентификатор сообщения и множество принятых	T5	да
Защита от переадресации	Подпись покрывает ключ получателя	T6	да
Доступность под блокировкой	Мульти-носитель, неотличимость трафика	T7	частично

Окончание таблицы 12

Целостность транзакции	Независимая сверка хеша подписантом	Т8	да

Итоговое сопоставление сформированных требований с реализующими их механизмами и способом подтверждения приведено в таблице 13. Все двенадцать требований реализованы. Функциональные требования подтверждены тестами, а архитектурные нефункциональные – построением системы.

Таблица 13 – Выполнение требований к системе

Требование	Реализующий механизм	Подтверждение
ТР-1	Сквозное AEAD-шифрование (sealed-box)	Round-trip, чужой получатель
ТР-2	Подпись Ed25519 и AEAD	Искажение, подделка подписи
ТР-3	Множество принятых идентификаторов	Защита от повтора
ТР-4	Подпись покрывает ключ получателя	Привязка к получателю
ТР-5	Случайные имена, единый ящик	Анонимность получателя
ТР-6	Подтверждения и сборка мусора	Ack и сборка мусора
ТР-7	Мульти-носитель: кворум и слияние	Зеркалирование при отказе
ТР-8	Реестр типов нагрузки	Текст и Safe как сменные типы
ТР-9	Кооперативное подписание Safe	Контрольный пример execTransaction
ТР-10	Идентификатор криптонабора в заголовке	По построению
ТР-11	Изоляция go-ethereum в слое L3	По построению
ТР-12	Контракт носителя из четырёх операций	Контракт-тест на четырёх носителях

Устойчивость к блокировке достигается работой поверх нескольких взаимозаменяемых носителей: при недоступности одного хранилища обмен продолжается через остальные, а потеря сообщений обнаруживается по подтверждениям и разрывам нумерации. Прямая секретность обеспечивается со стороны отправителя за счёт эфемерных ключей.

Честно фиксируются ограничения, выходящие за рамки модели:

– глобальный наблюдатель, анализирующий тайминги и объёмы обращений ко всем носителям, полностью не парируется и требует фиктивного трафика и фиксированного расписания;

– компрометация долговременного ключа получателя раскрывает ранее принятые им сообщения, что решается переходом к схеме с предключами и двойным храповиком;

– безопасность конечного устройства считается заданной.

Эти направления намечены как развитие и не обесценивают достигнутые свойства для целевого нарушителя.

8 Экономическое обоснование разработки

8.1 Расчёт затрат на машинное время

Разработка велась на одной рабочей станции мощностью 90 Вт, поэтому стоимость дисплейного времени входит в процессорное. Стоимость одного часа машинного времени определяется по формуле (3).

$$C_{п} = \frac{C_{кВт} \cdot P}{1000}, \tag{3}$$

где $C_{кВт}$ – тариф на электроэнергию для юридических лиц;

P – мощность ЭВМ, Вт.

При мощности 90 Вт стоимость часа машинного времени составляет:

$$C_{п} = \frac{6{,}5 \cdot 90}{1000} = 0{,}585 \text{ руб.}$$

За 752 рабочих часа (94 рабочих дня по 8 часов) затраты на машинное время равны:

$$M = 752 \cdot 0{,}585 \approx 440 \text{ руб.}$$

Эта величина учитывается в составе накладных расходов.

8.2 Трудоёмкость разработки

Разработка выполнена одним инженером-программистом. Трудоёмкость оценивается по этапам жизненного цикла прототипа: от

анализа предметной области до документирования. Распределение трудозатрат по этапам приведено в таблице 14.

Таблица 14 – Трудоёмкость разработки по этапам

Этап работ	Трудоёмкость, чел.-дни
Анализ предметной области и моделирование угроз	12
Проектирование архитектуры и формата сообщения	14
Реализация слоёв носителя, криптографии и протокола (L0–L3)	30
Реализация сценария кооперативного подписания Safe	10
Разработка интерфейсов командной строки и терминала (CLI/TUI)	8
Тестирование и отладка	12
Документирование	8
Итого	94

Суммарная трудоёмкость составляет 94 человеко-дня, что при среднем числе рабочих дней в месяце, равном 21, соответствует приблизительно 4,5 месяца работы одного специалиста.

8.3 Себестоимость и капитальные затраты на разработку

Себестоимость разработки складывается из материальных затрат, основной заработной платы инженера-разработчика, страховых взносов и накладных расходов. Часовая тарифная ставка разработчика определяется по формуле (4).

$$C_ч = \frac{O_к}{Ф_{рв}}, \tag{4}$$

где $O_к$ – месячный оклад инженера-программиста уровня middle;

$Ф_{рв}$ – плановый фонд рабочего времени за месяц.

Часовая ставка составляет:

$$C_ч = \frac{150000}{176} \approx 852 \text{ руб.}$$

Основная заработная плата за весь объём работ рассчитывается по формуле (5).

$$ЗП_{осн} = C_ч \cdot T_{ож} \cdot T_д, \quad (5)$$

где $T_{ож}$ – трудоёмкость разработки, рабочих дней;

$T_д$ – продолжительность рабочей смены, ч.

Основная заработная плата составляет:

$$ЗП_{осн} = 852 \cdot 94 \cdot 8 \approx 640907 \text{ руб.}$$

Страховые взносы во внебюджетные фонды приняты по общему тарифу 30 % и равны:

$$C_{вз} = 640907 \cdot 0{,}30 \approx 192272 \text{ руб.}$$

Амортизационные отчисления по рабочей станции за период разработки рассчитываются по формуле (6).

$$A = \frac{C_{пер} \cdot H_A \cdot K_M}{1200}, \quad (6)$$

где $C_{пер}$ – первоначальная стоимость рабочей станции;

H_A – годовая норма амортизации, %;

$К_М$ – количество рабочих месяцев.

Амортизационные отчисления составляют:

$$А = \frac{120000 \cdot 20 \cdot 5}{1200} = 10000 \text{ руб.}$$

Накладные расходы складываются из затрат на машинное время и амортизационных отчислений:

$$Н = 440 + 10000 = 10440 \text{ руб.}$$

Затраты на лицензии программного обеспечения отсутствуют – язык реализации и все используемые библиотеки распространяются по открытым лицензиям. Материальные затраты приведены в таблице 15.

Таблица 15 – Материальные затраты на разработку

Наименование	Количество, шт.	Цена за ед., руб.	Стоимость, руб.
Рабочая станция (ноутбук)	1	120 000	120 000
Аппаратный токен для тестирования подписи	2	3 000	6 000
Аренда облачного носителя (тестовый стенд, 5 мес.)	5	600	3 000
Итого			129 000

Полная себестоимость разработки приведена в таблице 16.

Таблица 16 – Сумма затрат на разработку

Вид затрат	Сумма, руб.
Материальные затраты	129 000
Основная заработная плата	640 907
Страховые взносы (30 %)	192 272
Накладные расходы (машинное время и амортизационные отчисления)	10 440
Итого себестоимость (С)	972 619

Капитальные затраты на разработку и внедрение определяются по формуле (7) как сумма себестоимости и затрат на внедрение – опытную эксплуатацию, настройку носителей и обучение пользователей, – оценённых в 60 000 рублей.

$$К_к = С + З_{вн}, \quad (7)$$

Капитальные затраты составляют:

$$К_к = 972619 + 60000 = 1032619 \text{ руб.}$$

Структура себестоимости разработки показана на рисунке 16: преобладают расходы на оплату труда – основная заработная плата вместе со страховыми взносами формируют около 86 % себестоимости, что характерно для наукоёмких программных разработок без закупки проприетарного оборудования и лицензий.

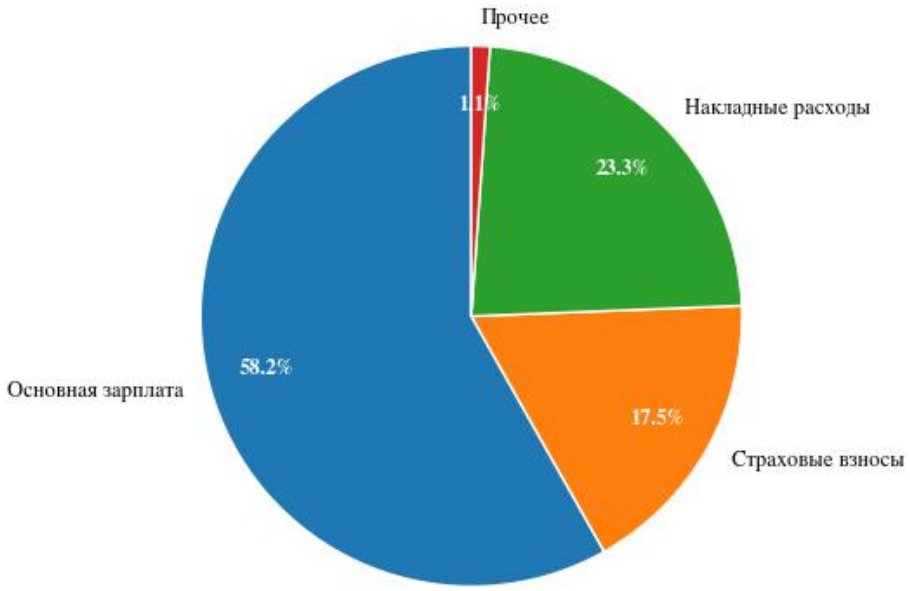

Рисунок 16 – Структура себестоимости разработки

8.4 Годовая экономия и экономический эффект

Экономический эффект оценивается относительно базового варианта, в котором целевое предприятие координирует трансграничные платежи через сторонний платный защищённый сервис с фиксированной абонентской платой 50 000 рублей в месяц, или 600 000 рублей в год. Годовая экономия от внедрения собственного решения определяется по формуле (8).

$$Э_р = (P_1 - P_2) + \Delta P_п, \quad (8)$$

где P_1 – эксплуатационные расходы до внедрения (абонентская плата за сторонний канал);

P_2 – эксплуатационные расходы после внедрения (аренда носителя и сопровождение);

$\Delta P_п$ – экономия от снижения простоя расчётов при блокировках.

Сопоставление годовых эксплуатационных затрат приведено в таблице 17.

Таблица 17 – Сравнение годовых эксплуатационных затрат

Статья затрат, руб./год	Сторонний сервис	Собственное решение
Абонентская плата за канал координации	600 000	0
Аренда носителя (облако/WebDAV)	0	12 000
Сопровождение решения	0	60 000
Итого годовых затрат	600 000	72 000

Годовая экономия составляет:

$$Э_р = (600000 - 72000) + 120000 = 648000 \text{ руб.}$$

Ожидаемый экономический эффект от внедрения определяется по формуле (9).

$$Э = Э_р - E_н \cdot K_к, \quad (9)$$

где $E_н$ – нормативный коэффициент экономической эффективности.

Экономический эффект составляет:

$$Э = 648000 - 0{,}33 \cdot 1032619 \approx 307236 \text{ руб.}$$

Расчётный коэффициент экономической эффективности определяется по формуле (10).

$$E_р = \frac{Э_р}{K_к}, \quad (10)$$

Коэффициент составляет:

$$E_р = \frac{648000}{1032619} \approx 0{,}63$$

Поскольку неравенство E_р ≥ E_н выполняется, внедрение разрабатываемого решения экономически эффективно. Срок окупаемости капитальных затрат определяется по формуле (11).

$$T = \frac{1}{E_р} = \frac{K_к}{Э_р}, \quad (11)$$

Срок окупаемости составляет:

$$T = \frac{1032619}{648000} \approx 1{,}6 \text{ года}$$

Это соответствует приблизительно 19 месяцам. Помимо прямой экономии на стороннем канале эффект включает трудно формализуемые составляющие: снижение риска простоя расчётов при блокировке единственного канала, отсутствие привязки к поставщику и сохранение полного контроля над ключами на стороне предприятия, что уменьшает регуляторные и репутационные риски. Таким образом, расчётный коэффициент экономической эффективности E_p = 0,63 превышает нормативный 0,33, и разработка прототипа экономически оправдана для целевого профиля заказчика.

9 Безопасность и экологичность

9.1 Значение и задачи безопасности жизнедеятельности

Охрана труда имеет выраженную социально-экономическую значимость: сохранение здоровья и работоспособности персонала прямо влияет на производительность и снижает издержки, связанные с заболеваемостью и текучестью кадров. Для офисной работы с ПЭВМ в финансовом и IT-секторе ведущими являются не травмирующие, а вредные производственные факторы длительного действия – зрительное и психоэмоциональное напряжение, гиподинамия, нагрузка на опорно-двигательный аппарат. По данным Росстата за 2024 год, уровень производственного травматизма в сфере информации и связи остаётся одним из самых низких среди отраслей, однако доля работников, занятых в условиях, не отвечающих гигиеническим нормативам по напряжённости труда, заметна, что и определяет приоритеты охраны труда в отрасли.

Обязанности по обеспечению безопасных условий и охраны труда возложены на работодателя статьёй 214 Трудового кодекса Российской Федерации [20]: организация безопасных рабочих мест, проведение специальной оценки условий труда в соответствии с Федеральным законом № 426-ФЗ [21], информирование работников об условиях труда и обеспечение режима труда и отдыха. Цели информатизации – повышение производительности и снижение рутинной нагрузки – согласуются с задачами охраны труда: автоматизация рутинных и ответственных операций уменьшает напряжённость трудового процесса и тем самым даёт сопряжённый экономический и социальный эффект.

9.2 Анализ условий труда и мероприятия по защите от воздействия вредных производственных факторов

Разработка ориентирована на отдел трансграничных расчётов предприятия внешнеэкономической деятельности. Для функционирования системы достаточно двух-четырёх специалистов-операторов с квалификацией в области информационной безопасности и финансовых операций. Их задача – подготовка, сверка и кооперативное подписание криптовалютных переводов, а также контроль доставки сообщений через носители. Режим работы – односменный, продолжительность смены 8 часов при пятидневной рабочей неделе. Рабочие места постоянные, размещены в офисном помещении. Перемещения специалистов в пределах помещения минимальны, основное время работа ведётся сидя за ПЭВМ.

Сводные данные об условиях труда на рабочем месте оператора ПЭВМ приведены в таблице 18.

Таблица 18 – Условия труда

Вредные производственные факторы	Характер воздействия на здоровье и работоспособность человека	Предельно допустимый уровень	Фактическое значение
Температура воздуха, °C	Тепловой дискомфорт, снижение работоспособности	22–24 (холодный период), 23–25 (тёплый)	23 (холодный), 24 (тёплый)
Относительная влажность воздуха, %	Сухость слизистых оболочек, утомление	40–60	45
Скорость движения воздуха, м/с	Сквозняк, переохлаждение	0,1	0,1
Освещённость рабочей поверхности, лк	Зрительное утомление, снижение остроты зрения	300–500 (общее)	400
Уровень шума, дБА	Снижение слуха, повышение артериального давления	50	45
Напряжённость ЭМП от ПЭВМ (5 Гц – 2 кГц), В/м	Утомление, головные боли	25	12

Окончание таблицы 18

Напряжённость трудового процесса	Психоэмоциональное напряжение, монотонность	Класс 2 (допустимый)	Класс 3.1 (до внедрения)

Превышений предельно допустимых уровней по физическим факторам на рабочем месте не выявлено: микроклимат поддерживается системой отопления и приточно-вытяжной вентиляции в пределах оптимальных значений для категории работ Iа по СанПиН 1.2.3685-21 [22], уровень шума не превышает 50 дБА согласно требованиям к условиям труда [23], напряжённость электромагнитного поля от современных жидкокристаллических мониторов существенно ниже норматива. Освещение совмещённое: естественное боковое дополняется искусственным общим равномерным, нормируемая освещённость 300–500 лк обеспечивается светодиодными светильниками с коэффициентом пульсации не выше нормируемого по СП 52.13330.2016 [24]. Разряд зрительной работы – высокой точности при работе с экраном.

Эргономика рабочего места соответствует требованиям к рабочему месту при выполнении работ сидя по ГОСТ 12.2.032-78 [25]: регулируемые по высоте кресло и подставка, размещение монитора на расстоянии 0,6–0,7 м от глаз, органы управления в пределах зоны досягаемости. Единственным фактором, превышающим допустимый класс до внедрения разработки, является напряжённость трудового процесса (класс 3.1 по классификации ГОСТ 12.0.003-2015 [26]), обусловленная высокой ответственностью за корректность платёжных операций и необходимостью одновременного контроля нескольких ненадёжных каналов связи. Снижение именно этого фактора и составляет основной вклад разработки в улучшение условий труда, что подтверждается оценкой ниже.

В качестве индивидуального задания выполнена оценка напряжённости трудового процесса оператора отдела расчётов по методу, изложенному в Руководстве Р 2.2.2006-05 [27]. Оценивается мужчина-

специалист, занятый подготовкой и сопровождением криптовалютных переводов в течение восьмичасовой смены. Показатели напряжённости по пяти группам факторов и их классы условий труда до и после внедрения разработанной системы приведены в таблице 19.

Таблица 19 – Показатели напряжённости трудового процесса

Показатели напряжённости трудового процесса	До внедрения, класс	После внедрения, класс
1. Интеллектуальные нагрузки		
1.1 Содержание работы	Решение сложных задач с выбором по известным алгоритмам – 3.1	Решение задач по чётким инструкциям – 2
1.2 Восприятие сигналов и их оценка	Восприятие с последующим сопоставлением и проверкой – 3.1	Восприятие с последующей коррекцией действий – 2
1.3 Распределение функций по сложности задания	Обработка, проверка и контроль выполнения задания – 3.1	Обработка и выполнение задания – 2
1.4 Характер выполняемой работы	Работа в условиях дефицита времени – 3.1	Работа по установленному графику – 2
2. Сенсорные нагрузки		
2.1 Длительность сосредоточенного наблюдения, % смены	51–75 – 3.1	26–50 – 2
2.2 Плотность сигналов и сообщений за 1 час	До 75 – 2	До 75 – 2
2.3 Число объектов одновременного наблюдения	До 5 – 1	До 5 – 1
2.4 Размер объекта различения, мм	5–1,1 более 50 % смены – 2	5–1,1 более 50 % смены – 2
2.5 Работа с оптическими приборами, % смены	До 25 – 1	До 25 – 1
2.6 Наблюдение за экраном видеотерминала, ч	5–6 – 3.1	3–4 – 2
2.7 Нагрузка на слуховой анализатор	Разборчивость 90–70 % – 2	Разборчивость 90–70 % – 2
2.8 Нагрузка на голосовой аппарат, ч в неделю	До 16 – 1	До 16 – 1
3. Эмоциональные нагрузки		
3.1 Степень ответственности, значимость ошибки	Ответственность за функциональное качество, ошибка ведёт к финансовым потерям – 3.2	Ответственность снижена за счёт автоматической сверки – 3.1

Окончание таблицы 19

3.2 Степень риска для собственной жизни	Исключена – 1	Исключена – 1
3.3 Ответственность за безопасность других лиц	Исключена – 1	Исключена – 1
3.4 Число конфликтных ситуаций за смену	1–3 – 2	1–3 – 2
4. Монотонность нагрузок		
4.1 Число элементов для реализации задания	Более 10 – 1	Более 10 – 1
4.2 Продолжительность простых заданий, с	Более 100 – 1	Более 100 – 1
4.3 Время активных действий, % смены	20 и более – 1	20 и более – 1
4.4 Монотонность производственной обстановки, %	Менее 75 – 1	Менее 75 – 1
5. Режим работы		
5.1 Фактическая продолжительность рабочего дня, ч	8–9 – 2	8–9 – 2
5.2 Сменность работы	Односменная без ночной смены – 1	Односменная без ночной смены – 1
5.3 Регламентированные перерывы	Недостаточно регламентированы – 2	Достаточны, предусмотрены – 1
Итоговый класс условий труда по напряжённости	3.1 (вредный)	2 (допустимый)

Итоговый класс условий труда по показателям напряжённости определяется по числу показателей высших классов согласно Р 2.2.2006-05 [27]. До внедрения разработки число показателей класса 3.1 и выше превышает порог, и итоговый класс составляет 3.1 (вредный, первой степени). После внедрения автоматическая сверка хеша транзакции, кооперативное подписание и устойчивая многоканальная доставка снимают дефицит времени, снижают долю сосредоточенного наблюдения и значимость ошибки оператора. Число показателей вредного класса опускается ниже порога, и итоговый класс снижается до 2 (допустимый). Таким образом, внедрение системы переводит условия труда оператора по фактору напряжённости из вредного класса в допустимый.

9.3 Обеспечение электробезопасности

Электропитание рабочих мест осуществляется от трёхфазной сети переменного тока напряжением 380/220 В частотой 50 Гц с глухозаземлённой нейтралью по системе TN-S. Распределение выполнено кабелем марки ВВГнг-LS сечением 5×6 мм². Применяемое оборудование (ПЭВМ, мониторы, сетевое оборудование) относится к классу защиты I и II. Классификация помещения, в котором предполагается внедрение результатов работы, приведена в таблице 20.

Таблица 20 – Классификация помещений по опасности поражения электрическим током

Подразделение организации (помещение)	Факторы, определяющие опасность поражения электрическим током	Класс помещения
Рабочее место оператора отдела расчётов	Сухое отапливаемое помещение, температура воздуха ≤ 35 °C, отсутствие токопроводящих полов и пыли, влажность не повышена	Помещение без повышенной опасности
Серверное помещение	Отсутствие повышенной влажности и токопроводящих полов, кондиционирование, температура ≤ 35 °C	Помещение без повышенной опасности

Согласно Правилам устройства электроустановок [28] оба помещения относятся к категории без повышенной опасности. Для защиты при косвенном прикосновении применяется комплекс мер:

– защитное заземление корпусов оборудования;

– защитное автоматическое отключение питания (зануление);

– уравнивание потенциалов в пределах помещения;

– применение устройств защитного отключения по дифференциальному току;

– двойная или усиленная изоляция оборудования класса II;

– использование сверхнизкого напряжения в цепях периферии;

– защитное электрическое разделение питающих цепей;

– ограничение доступа к токоведущим частям ограждениями и оболочками.

Эксплуатация электроустановок ведётся в соответствии с Правилами по охране труда при эксплуатации электроустановок [29]. Применяемое низковольтное оборудование соответствует требованиям технического регламента ТР ТС 004/2011 [30]. Защита от статического электричества обеспечивается заземлением и поддержанием нормируемой влажности воздуха, защита от атмосферного электричества – общим молниезащитным контуром здания.

9.4 Пожарная безопасность

Категорирование помещений по взрыво- и пожароопасности приведено в таблице 21.

Таблица 21 – Категории помещений и класс зон по взрыво- и пожароопасности

Подразделение организации (помещение)	Категория помещения по взрыво- и пожароопасности	Класс зоны
Офисное помещение отдела расчётов	В4 (пониженная пожароопасность)	П-IIа
Серверное помещение	В4 (пониженная пожароопасность)	П-IIа

Категории помещений определены по Федеральному закону № 123-ФЗ [31]: горючая нагрузка представлена изоляцией кабелей, корпусами оборудования, мебелью и бумагой, что соответствует категории В4 пониженной пожароопасности и классу зоны П-IIа. Обнаружение пожара обеспечивается автоматической пожарной сигнализацией с дымовыми оптико-электронными извещателями. В качестве первичных средств пожаротушения для электрооборудования под напряжением до 1000 В применяются углекислотные огнетушители ОУ-3 и ОУ-5. Эксплуатация ведётся с соблюдением Правил противопожарного режима [32], пути

эвакуации запроектированы по СП 1.13130.2020 [33]. Предусмотрены следующие мероприятия по пожарной безопасности:

– максимально возможное применение негорючих и трудногорючих материалов отделки;

– применение электрооборудования в исполнении, соответствующем классу зоны;

– применение быстродействующих устройств защитного отключения;

– применение оборудования, удовлетворяющего требованиям электростатической безопасности;

– оснащение помещений первичными средствами пожаротушения (ОУ-3, ОУ-5);

– применение автоматической пожарной сигнализации с дымовыми извещателями;

– применение строительных конструкций с регламентируемыми пределами огнестойкости;

– обеспечение нормируемого числа, размеров и исполнения путей эвакуации;

– обучение работающих требованиям пожарной безопасности и проведение тренировок по эвакуации.

9.5 Безопасность жизнедеятельности в чрезвычайных ситуациях

Чрезвычайные ситуации, как правило, выходят за пределы отдельного рабочего места, поэтому рассматривается предприятие в целом. Для Краснодарского края характерны природные чрезвычайные ситуации в виде сильных ливней и подтоплений, шквалистого ветра, а также сейсмическая активность силой до 6–7 баллов. Степень опасности самого технологического процесса (офисная работа с ПЭВМ) в нормальном режиме низкая и возрастает до средней при авариях систем электроснабжения. Потенциальная опасность соседних объектов оценивается с учётом их удалённости,

преобладающего направления ветра и рельефа местности. Предлагаются следующие меры:

– установление режима защиты персонала при угрозе и возникновении чрезвычайной ситуации, оповещение и эвакуация по заранее разработанным планам;

– повышение устойчивости объекта за счёт резервирования электропитания и дублирования каналов связи и хранения данных;

– планирование и проведение спасательных и аварийно-восстановительных работ силами нештатных формирований;

– выполнение инженерно-технических мероприятий гражданской обороны для снижения возможных потерь и ущерба.

Организация мероприятий ведётся в соответствии с Федеральным законом № 68-ФЗ [34] и Федеральным законом № 28-ФЗ [35]. Свойство устойчивости разработанной системы к блокировке и многоканальное хранение данных дополнительно повышают готовность предприятия к нарушению связи в условиях чрезвычайных ситуаций.

9.6 Охрана окружающей среды

Воздействие офисной деятельности предприятия на компоненты окружающей среды незначительно и оценивается по схеме «источник – процесс – отходы». В атмосферу производственные выбросы не поступают. Источниками являются лишь тепловыделение оборудования и вентиляционные потоки, специальная очистка не требуется. В гидросферу поступают только хозяйственно-бытовые сточные воды, отводимые в городскую канализацию по договору с водоканалом, что регулируется требованиями Федерального закона № 7-ФЗ [36].

Основное внимание уделяется обращению с твёрдыми отходами в соответствии с Федеральным законом № 89-ФЗ [37]. Образуются отходы бумаги и картона (V класс опасности по федеральному классификационному

каталогу отходов [38]), отработанные люминесцентные лампы (I класс) и вышедшая из строя вычислительная техника и её компоненты (II–IV класс). Утилизация устаревшей вычислительной техники выполняется по решению комиссии из сотрудников предприятия: пригодные узлы и драгоценные металлы передаются специализированным лицензированным организациям, пластиковые корпуса направляются в твёрдые коммунальные отходы, опасные компоненты подлежат раздельному сбору и захоронению на специализированных полигонах. Разработанное программное средство дополнительного воздействия на окружающую среду не оказывает, поскольку работает на уже имеющемся оборудовании и не требует выделенной инфраструктуры.

Заключение

В выпускной квалификационной работе решена актуальная задача проектирования и реализации транспортно-независимой системы безопасного асинхронного обмена сообщениями и криптовалютными транзакциями через недоверенного посредника. Поставленная цель достигнута, все семь задач выполнены.

В первом разделе проведён анализ предметной области и существующих решений – защищённых мессенджеров, систем, устойчивых к анализу трафика, и форматов кооперативного подписания транзакций. Показано, что ни одно из них не покрывает одновременно транспортную независимость, асинхронность, анонимность получателя на носителе, устойчивость к блокировке и поддержку криптовалютных транзакций как полезной нагрузки, что и определяет нерешённую задачу.

Во втором разделе построена модель угроз с недоверенным носителем в роли активного противника, сформированы требования и спроектирована слоистая транспортно-независимая архитектура. Определён формат защищённого сообщения по схеме «подписать, затем зашифровать» с привязкой подписи к открытому ключу получателя, что исключает переадресацию, и заложена криптоагильность через идентификатор криптонабора.

В третьем разделе реализован программный прототип на языке Go. Реализованы абстракция носителя с вариантами для оперативной памяти, файловой системы, протокола WebDAV и мультиносителя, криптографический слой на основе X25519, HKDF-SHA-256, XChaCha20-Poly1305 и Ed25519, а также протокол ящика с публикацией, опросом, подтверждением и сборкой мусора. Реализован прикладной сценарий кооперативного подписания мультиподписи Safe для перевода стейблкоина, а также интерфейсы командной строки и терминала.

В четвёртом разделе выполнено тестирование на трёх уровнях, все одиннадцать функциональных и противоборствующих проверок пройдены. Контрольный пример кооперативного подписания подтвердил корректность вычисления хеша по EIP-712, формата подписей и сборки вызова execTransaction. Сопоставление свойств безопасности с угрозами показало покрытие восьми из восьми угроз модели, из них семь – полностью.

В пятом разделе выполнено экономическое обоснование: капитальные затраты на разработку и внедрение прототипа оценены в 1,03 млн рублей, годовая экономия относительно платного стороннего канала координации – 648 тыс. рублей, расчётный коэффициент экономической эффективности – 0,63 при нормативном 0,33, срок окупаемости – около 1,6 года.

В шестом разделе рассмотрены вопросы безопасности и экологичности проекта. По результатам индивидуального задания внедрение системы снижает класс условий труда оператора по напряжённости трудового процесса с 3.1 (вредный) до 2 (допустимый), а помещения отнесены к категории без повышенной опасности и к категории В4 по пожароопасности.

Перспективы развития работы намечены по нескольким направлениям. Это переход к схеме с предключами и двойным храповиком для полной прямой секретности, добавление фиктивного трафика и фиксированного расписания обращений для противодействия глобальному наблюдателю, реализация криптонабора на основе отечественных алгоритмов ГОСТ при появлении соответствующего требования, а также расширение набора типов полезной нагрузки и реализаций носителя. По результатам выполненной работы цель достигнута, поставленные задачи решены, разработанный прототип готов к применению для защищённой координации трансграничных расчётов в условиях 2025–2026 годов.

Список использованных источников

1. Unger N., Dechand S., Bonneau J. et al. SoK: Secure Messaging // 2015 IEEE Symposium on Security and Privacy. – IEEE, 2015. – P. 232–249. – DOI: 10.1109/SP.2015.22.

2. Marlinspike M., Perrin T. The X3DH Key Agreement Protocol. – Open Whisper Systems, 2016. – 11 p. – URL: https://signal.org/docs/specifications/x3dh/ (дата обращения: 29.05.2026).

3. Perrin T., Marlinspike M. The Double Ratchet Algorithm. – Open Whisper Systems, 2016. – 35 p. – URL: https://signal.org/docs/specifications/doubleratchet/ (дата обращения: 29.05.2026).

4. van den Hooff J., Lazar D., Zaharia M., Zeldovich N. Vuvuzela: Scalable Private Messaging Resistant to Traffic Analysis // Proceedings of the 25th Symposium on Operating Systems Principles (SOSP '15). – ACM, 2015. – P. 137–152. – DOI: 10.1145/2815400.2815417.

5. Chow A. BIP-174: Partially Signed Bitcoin Transaction Format. – Bitcoin Improvement Proposals, 2017. – URL: https://github.com/bitcoin/bips/blob/master/bip-0174.mediawiki (дата обращения: 29.05.2026).

6. Buterin V., Sandford R., Steele L. et al. EIP-712: Typed structured data hashing and signing. – Ethereum Improvement Proposals, 2017. – URL: https://eips.ethereum.org/EIPS/eip-712 (дата обращения: 29.05.2026).

7. Российская Федерация. Законы. О цифровых финансовых активах, цифровой валюте и о внесении изменений в отдельные законодательные акты Российской Федерации: Федер. закон от 31.07.2020 № 259-ФЗ (в действующей редакции). – Доступ из справ.-правовой системы «КонсультантПлюс».

8. Информация Банка России от 12.03.2025 о сделках с криптовалютами в рамках экспериментального правового режима. – М.: Банк России, 2025. – URL: https://www.cbr.ru (дата обращения: 29.05.2026).

9. Методика оценки угроз безопасности информации: утв. ФСТЭК России 5 февраля 2021 г. – М.: ФСТЭК России, 2021. – 83 с.

10. Bernstein D. J. Curve25519: New Diffie-Hellman Speed Records // Public Key Cryptography – PKC 2006. – Berlin: Springer, 2006. – (LNCS; vol. 3958). – P. 207–228. – DOI: 10.1007/11745853_14.

11. Bernstein D. J., Duif N., Lange T., Schwabe P., Yang B.-Y. High-speed high-security signatures // Journal of Cryptographic Engineering. – 2012. – Vol. 2, № 2. – P. 77–89. – DOI: 10.1007/s13389-012-0027-1.

12. Krawczyk H., Eronen P. HMAC-based Extract-and-Expand Key Derivation Function (HKDF): RFC 5869. – IETF, 2010. – 14 p. – DOI: 10.17487/RFC5869.

13. Nir Y., Langley A. ChaCha20 and Poly1305 for IETF Protocols: RFC 8439. – IETF, 2018. – 46 p. – DOI: 10.17487/RFC8439.

14. Boneh D., Shoup V. A Graduate Course in Applied Cryptography. – Version 0.6. – 2023. – 900 p. – URL: https://toc.cryptobook.us/ (дата обращения: 29.05.2026).

15. ГОСТ Р 34.10-2012. Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи. – М.: Стандартинформ, 2013. – 22 с.

16. Российская Федерация. Законы. О персональных данных: Федер. закон от 27.07.2006 № 152-ФЗ (в действующей редакции). – Доступ из справ.-правовой системы «КонсультантПлюс».

17. Российская Федерация. Законы. Об информации, информационных технологиях и о защите информации: Федер. закон от

27.07.2006 № 149-ФЗ (в действующей редакции). – Доступ из справ.-правовой системы «КонсультантПлюс».

18. ГОСТ Р ИСО/МЭК 27001-2021. Информационная технология. Методы и средства обеспечения безопасности. Системы менеджмента информационной безопасности. Требования. – М.: Стандартинформ, 2021. – 35 с.

19. Donovan A. A., Kernighan B. W. The Go Programming Language. – Boston: Addison-Wesley, 2015. – 380 p. – ISBN 978-0-13-419044-0.

20. Российская Федерация. Кодексы. Трудовой кодекс Российской Федерации от 30.12.2001 № 197-ФЗ (в действующей редакции). – Доступ из справ.-правовой системы «КонсультантПлюс».

21. Российская Федерация. Законы. О специальной оценке условий труда: Федер. закон от 28.12.2013 № 426-ФЗ (в действующей редакции). – Доступ из справ.-правовой системы «КонсультантПлюс».

22. СанПиН 1.2.3685-21. Гигиенические нормативы и требования к обеспечению безопасности и (или) безвредности для человека факторов среды обитания. – М.: Роспотребнадзор, 2021. – 469 с.

23. СП 2.2.3670-20. Санитарно-эпидемиологические требования к условиям труда. – М.: Роспотребнадзор, 2020. – 60 с.

24. СП 52.13330.2016. Естественное и искусственное освещение. Актуализированная редакция СНиП 23-05-95*. – М.: Минстрой России, 2016. – 105 с.

25. ГОСТ 12.2.032-78. Система стандартов безопасности труда. Рабочее место при выполнении работ сидя. Общие эргономические требования. – М.: Издательство стандартов, 1979. – 9 с.

26. ГОСТ 12.0.003-2015. Система стандартов безопасности труда. Опасные и вредные производственные факторы. Классификация. – М.: Стандартинформ, 2016. – 10 с.

27. Р 2.2.2006-05. Руководство по гигиенической оценке факторов рабочей среды и трудового процесса. Критерии и классификация условий труда. – М.: Роспотребнадзор, 2005. – 142 с.

28. Правила устройства электроустановок (ПУЭ). – 7-е изд. – М.: Энергосервис, 2003.

29. Об утверждении Правил по охране труда при эксплуатации электроустановок: приказ Минтруда России от 15.12.2020 № 903н. – Зарегистрировано в Минюсте России 30.12.2020 № 61957.

30. ТР ТС 004/2011. Технический регламент Таможенного союза «О безопасности низковольтного оборудования»: утв. решением Комиссии Таможенного союза от 16.08.2011 № 768.

31. Российская Федерация. Законы. Технический регламент о требованиях пожарной безопасности: Федер. закон от 22.07.2008 № 123-ФЗ (в действующей редакции). – Доступ из справ.-правовой системы «КонсультантПлюс».

32. Об утверждении Правил противопожарного режима в Российской Федерации: пост. Правительства РФ от 16.09.2020 № 1479 (в действующей редакции). – Собр. законодательства РФ. – 2020. – № 39. – Ст. 6056.

33. СП 1.13130.2020. Системы противопожарной защиты. Эвакуационные пути и выходы. – М.: МЧС России, 2020. – 60 с.

34. Российская Федерация. Законы. О защите населения и территорий от чрезвычайных ситуаций природного и техногенного характера: Федер. закон от 21.12.1994 № 68-ФЗ (в действующей редакции). – Доступ из справ.-правовой системы «КонсультантПлюс».

35. Российская Федерация. Законы. О гражданской обороне: Федер. закон от 12.02.1998 № 28-ФЗ (в действующей редакции). – Доступ из справ.-правовой системы «КонсультантПлюс».

36. Российская Федерация. Законы. Об охране окружающей среды: Федер. закон от 10.01.2002 № 7-ФЗ (в действующей редакции). – Доступ из справ.-правовой системы «КонсультантПлюс».

37. Российская Федерация. Законы. Об отходах производства и потребления: Федер. закон от 24.06.1998 № 89-ФЗ (в действующей редакции). – Доступ из справ.-правовой системы «КонсультантПлюс».

38. Об утверждении Федерального классификационного каталога отходов: приказ Росприроднадзора от 22.05.2017 № 242 (в действующей редакции). – Зарегистрировано в Минюсте России 08.06.2017 № 47008.

Приложение А

Фрагменты исходного кода прототипа

Листинг А.1 – carrier/carrier.go

```
// Package carrier is layer L0 of DeadDrop: the untrusted-carrier
abstraction.
//
// A Carrier is a flat namespace of named blobs ("a drop") with four
operations
// — Put, Get, List, Delete. It is the lowest common denominator
across the
// media DeadDrop can ride on: a local folder, a mounted cloud, WebDAV,
or
// several of those composed for resilience. The carrier is assumed
UNTRUSTED:
// it may read, alter, delete, reorder, replay or block blobs. All
security
// (confidentiality, integrity, authenticity, anonymity) is provided
by the
// upper layers (drop, crypto), never by the carrier; this layer only
stores and
// retrieves bytes and provides continuity under blocking via
composition.
package carrier

import (
      "errors"
      "strings"
)

// ErrNotExist is returned by Get when a blob is absent. Delete treats
a missing
// blob as success (idempotent), and List of a missing drop yields an
empty
// slice without error — callers cannot distinguish "empty" from
"unreadable"
// through List alone, by design.
var ErrNotExist = errors.New("carrier: blob does not exist")

// Carrier is a flat namespace of named blobs with atomic publish.
//
// Contract:
//   - Put publishes data under name atomically from a List observer's
point of
//     view (the blob appears whole or not at all); an existing blob
is
```

```
//     overwritten.
//   - Get returns the blob, or ErrNotExist if absent.
//   - List returns the blob names currently in the drop; a
missing/empty drop
//     yields an empty slice and a nil error.
//   - Delete removes a blob and is idempotent (absent blob is
success).
//
// Names are opaque leaf identifiers (see ValidName); the upper layers
use
// random names so that the carrier learns nothing from them.
type Carrier interface {
      Put(name string, data []byte) error
      Get(name string) ([]byte, error)
      List() ([]string, error)
      Delete(name string) error
}

// ErrBadName is returned for a name that is unsafe as a storage leaf.
var ErrBadName = errors.New("carrier: invalid blob name")

// ValidName reports whether name is a safe leaf identifier: non-empty,
not "."
// or "..", and free of path separators and NUL. This is a security
boundary —
// it prevents a crafted name from escaping the drop directory in
filesystem-
// backed carriers (path traversal).
func ValidName(name string) bool {
      if name == "" || name == "." || name == ".." {
            return false
      }
      if len(name) > 255 {
            return false
      }
      if strings.ContainsAny(name, "/\\\x00") {
            return false
      }
      return true
}
```

Листинг А.2 – carrier/mem.go

```
package carrier

import "sync"
```

```
// Mem is an in-memory Carrier. It is safe for concurrent use and is
primarily
// for tests and as the reference implementation of the contract.
type Mem struct {
	mu    sync.RWMutex
	blobs map[string][]byte
}

// NewMem returns an empty in-memory carrier.
func NewMem() *Mem {
	return &Mem{blobs: make(map[string][]byte)}
}

func (m *Mem) Put(name string, data []byte) error {
	if !ValidName(name) {
		return ErrBadName
	}
	cp := make([]byte, len(data))
	copy(cp, data)
	m.mu.Lock()
	m.blobs[name] = cp
	m.mu.Unlock()
	return nil
}

func (m *Mem) Get(name string) ([]byte, error) {
	m.mu.RLock()
	data, ok := m.blobs[name]
	m.mu.RUnlock()
	if !ok {
		return nil, ErrNotExist
	}
	cp := make([]byte, len(data))
	copy(cp, data)
	return cp, nil
}

func (m *Mem) List() ([]string, error) {
	m.mu.RLock()
	out := make([]string, 0, len(m.blobs))
	for name := range m.blobs {
		out = append(out, name)
	}
	m.mu.RUnlock()
	return out, nil
}

func (m *Mem) Delete(name string) error {
	m.mu.Lock()
```

```
	delete(m.blobs, name)
	m.mu.Unlock()
	return nil
}
```

Листинг А.3 – carrier/fs.go

```
package carrier

import (
	"errors"
	"io/fs"
	"os"
	"path/filepath"
	"strings"
)

// tmpPrefix marks in-progress writes. Real blob names never start
with a dot
// (the upper layers use hex), so List skips dot-prefixed entries and
never
// exposes a half-written file.
const tmpPrefix = ".ddtmp-"

// FS is a Carrier backed by a local directory (the "drop"). The
directory may
// itself be a mounted cloud. Publish is atomic via write-to-temp +
rename
// within the same directory.
type FS struct {
	root string
}

// NewFS returns a filesystem carrier rooted at dir. The directory is
created
// lazily on first Put.
func NewFS(dir string) *FS {
	return &FS{root: dir}
}

func (c *FS) Put(name string, data []byte) error {
	if !ValidName(name) {
		return ErrBadName
	}
	if err := os.MkdirAll(c.root, 0o700); err != nil {
		return err
	}
```

```
	tmp, err := os.CreateTemp(c.root, tmpPrefix+"*")
	if err != nil {
		return err
	}
	tmpName := tmp.Name()
	// Best-effort cleanup if we bail out before the rename succeeds.
	defer func() { _ = os.Remove(tmpName) }()
	if _, err := tmp.Write(data); err != nil {
		tmp.Close()
		return err
	}
	if err := tmp.Close(); err != nil {
		return err
	}
	return os.Rename(tmpName, filepath.Join(c.root, name))
}

func (c *FS) Get(name string) ([]byte, error) {
	if !ValidName(name) {
		return nil, ErrBadName
	}
	data, err := os.ReadFile(filepath.Join(c.root, name))
	if errors.Is(err, fs.ErrNotExist) {
		return nil, ErrNotExist
	}
	return data, err
}

func (c *FS) List() ([]string, error) {
	ents, err := os.ReadDir(c.root)
	if errors.Is(err, fs.ErrNotExist) {
		return nil, nil // missing drop lists as empty, per contract
	}
	if err != nil {
		return nil, err
	}
	out := make([]string, 0, len(ents))
	for _, e := range ents {
		if e.IsDir() {
			continue
		}
		if strings.HasPrefix(e.Name(), ".") {
			continue // temp/hidden files are not blobs
		}
		out = append(out, e.Name())
	}
	return out, nil
}
```

```
func (c *FS) Delete(name string) error {
	if !ValidName(name) {
		return ErrBadName
	}
	err := os.Remove(filepath.Join(c.root, name))
	if errors.Is(err, fs.ErrNotExist) {
		return nil // idempotent
	}
	return err
}
```

Листинг А.4 – carrier/multi.go

```
package carrier

import "errors"

// Multi composes several carriers for continuity under blocking (T9,
TR-F-5).
//
// Writes are mirrored to every backing carrier; a Put succeeds when
at least
// `quorum` of them accept it, so blocking or failure of some carriers
neither
// loses the message nor fails the publish. Reads merge the listings
of all
// reachable carriers (an unreachable one is skipped); Get returns the
first
// copy found. Because the upper layers use unique random blob names,
the same
// blob carries the same name on every carrier, so listing-level
deduplication
// is exact.
type Multi struct {
	carriers []Carrier
	quorum   int
}

// NewMulti composes carriers with the given write quorum (number of
mirrors
// that must accept a Put for it to succeed). quorum is clamped to
// [1, len(carriers)]; a typical value is 2.
func NewMulti(quorum int, carriers ...Carrier) *Multi {
	if quorum < 1 {
		quorum = 1
	}
	if quorum > len(carriers) {
```

```
            quorum = len(carriers)
      }
      return &Multi{carriers: carriers, quorum: quorum}
}

func (m *Multi) Put(name string, data []byte) error {
      if !ValidName(name) {
            return ErrBadName
      }
      var ok int
      var errs []error
      for _, c := range m.carriers {
            if err := c.Put(name, data); err != nil {
                  errs = append(errs, err)
                  continue
            }
            ok++
      }
      if ok >= m.quorum {
            return nil
      }
      return errors.Join(errs...)
}

func (m *Multi) Get(name string) ([]byte, error) {
      var errs []error
      for _, c := range m.carriers {
            data, err := c.Get(name)
            if err == nil {
                  return data, nil
            }
            if errors.Is(err, ErrNotExist) {
                  continue
            }
            errs = append(errs, err)
      }
      // Reachable carriers that lacked the blob agreed it is absent;
only report
      // ErrNotExist when none were merely unreachable, so a transient
outage is
      // not mistaken for a definitive miss.
      if len(errs) > 0 {
            return nil, errors.Join(errs...)
      }
      return nil, ErrNotExist
}

func (m *Multi) List() ([]string, error) {
      seen := make(map[string]struct{})
```

```
	var errs []error
	var anyOK bool
	for _, c := range m.carriers {
		names, err := c.List()
		if err != nil {
			errs = append(errs, err)
			continue
		}
		anyOK = true
		for _, n := range names {
			seen[n] = struct{}{}
		}
	}
	if !anyOK && len(errs) > 0 {
		return nil, errors.Join(errs...)
	}
	out := make([]string, 0, len(seen))
	for n := range seen {
		out = append(out, n)
	}
	return out, nil
}

func (m *Multi) Delete(name string) error {
	var errs []error
	for _, c := range m.carriers {
		if err := c.Delete(name); err != nil {
			errs = append(errs, err)
		}
	}
	return errors.Join(errs...)
}
```

Листинг A.5 – carrier/webdav.go

```
package carrier

import (
	"bytes"
	"encoding/xml"
	"fmt"
	"io"
	"net/http"
	"net/url"
	"path"
	"strconv"
	"strings"
	"sync"
```

```
	"time"
)

// WebDAV is a Carrier backed by a WebDAV collection (e.g.
Yandex.Disk). The
// drop is one collection on the server; blobs are files directly
under it. A
// WebDAV PUT is atomic from a listing observer's perspective, so no
temp+rename
// dance is needed. Auth is the full Authorization header value (e.g.
// "Basic <base64(user:app-password)>").
type WebDAV struct {
	endpoint string // trimmed origin, e.g.
"https://webdav.yandex.ru"
	dropAbs  string // absolute collection path on the server, e.g.
"/deaddrop"
	auth     string
	hc       *http.Client

	mu      sync.Mutex
	created bool // the drop collection has been ensured this process
}

const (
	davMaxRetries    = 6
	davRetryBackoff  = 500 * time.Millisecond
	davMaxBackoff    = 8 * time.Second
	davMaxRetryAfter = 30 * time.Second
	davReqTimeout    = 90 * time.Second
)

// NewWebDAV constructs a WebDAV carrier. endpoint is the server
origin; drop is
// the collection path under which blobs live; auth is the full
Authorization
// header value sent on every request.
func NewWebDAV(endpoint, auth, drop string) *WebDAV {
	return &WebDAV{
		endpoint: strings.TrimRight(endpoint, "/"),
		dropAbs:  "/" + strings.Trim(drop, "/"),
		auth:     auth,
		hc: &http.Client{
			Timeout: davReqTimeout,
			Transport: &http.Transport{
				MaxIdleConns:        32,
				MaxIdleConnsPerHost: 32,
				IdleConnTimeout:     90 * time.Second,
				ForceAttemptHTTP2:   true,
			},
```

```
            },
      }
}

func (c *WebDAV) url(absPath string) string {
      segs := strings.Split(strings.Trim(absPath, "/"), "/")
      for i, s := range segs {
            segs[i] = url.PathEscape(s)
      }
      return c.endpoint + "/" + strings.Join(segs, "/")
}

func (c *WebDAV) blobURL(name string) string { return c.url(c.dropAbs
+ "/" + name) }

// do issues one request with auth and retries transient failures
(network
// errors, 429, 5xx). A 429/5xx Retry-After header is honoured in
preference to
// the exponential backoff. The response body is owned by the caller.
func (c *WebDAV) do(method, rawURL string, body []byte, headers
map[string]string) (*http.Response, error) {
      var lastErr error
      var wait time.Duration
      for attempt := 0; attempt <= davMaxRetries; attempt++ {
            if attempt > 0 {
                  backoff := davRetryBackoff << (attempt - 1)
                  if backoff > davMaxBackoff {
                        backoff = davMaxBackoff
                  }
                  if wait > backoff {
                        backoff = wait
                  }
                  time.Sleep(backoff)
                  wait = 0
            }
            var rdr io.Reader
            if body != nil {
                  rdr = bytes.NewReader(body)
            }
            req, err := http.NewRequest(method, rawURL, rdr)
            if err != nil {
                  return nil, err
            }
            req.Header.Set("Authorization", c.auth)
            for k, v := range headers {
                  req.Header.Set(k, v)
            }
            if body != nil {
```

```
                req.ContentLength = int64(len(body))
            }
            resp, err := c.hc.Do(req)
            if err != nil {
                lastErr = err
                continue
            }
            if resp.StatusCode == http.StatusTooManyRequests ||
resp.StatusCode >= 500 {
                wait = parseRetryAfter(resp.Header.Get("Retry-After"))
                resp.Body.Close()
                lastErr = fmt.Errorf("webdav: %s: HTTP %d", method,
resp.StatusCode)
                continue
            }
            return resp, nil
      }
      return nil, lastErr
}

func parseRetryAfter(v string) time.Duration {
      v = strings.TrimSpace(v)
      if v == "" {
            return 0
      }
      var d time.Duration
      if secs, err := strconv.Atoi(v); err == nil {
            d = time.Duration(secs) * time.Second
      } else if t, err := http.ParseTime(v); err == nil {
            d = time.Until(t)
      }
      if d < 0 {
            return 0
      }
      if d > davMaxRetryAfter {
            return davMaxRetryAfter
      }
      return d
}

// ensure creates the drop collection (and its parents) once. MKCOL is
not
// recursive; an existing collection is reported by Yandex as 405 or,
when in
// use, 423 — both mean "already there".
func (c *WebDAV) ensure() error {
      c.mu.Lock()
      defer c.mu.Unlock()
      if c.created {
```

```
            return nil
      }
      cur := ""
      for _, seg := range strings.Split(strings.Trim(c.dropAbs, "/"),
"/") {
            if seg == "" {
                  continue
            }
            cur += "/" + seg
            resp, err := c.do("MKCOL", c.url(cur), nil, nil)
            if err != nil {
                  return err
            }
            code := resp.StatusCode
            resp.Body.Close()
            if code/100 != 2 && code != http.StatusMethodNotAllowed &&
code != http.StatusLocked {
                  return fmt.Errorf("webdav: MKCOL %s: HTTP %d", cur,
code)
            }
      }
      c.created = true
      return nil
}

func (c *WebDAV) Put(name string, data []byte) error {
      if !ValidName(name) {
            return ErrBadName
      }
      if err := c.ensure(); err != nil {
            return err
      }
      resp, err := c.do(http.MethodPut, c.blobURL(name), data,
            map[string]string{"Content-Type": "application/octet-
stream"})
      if err != nil {
            return err
      }
      defer resp.Body.Close()
      if resp.StatusCode/100 != 2 {
            return fmt.Errorf("webdav: PUT %s: HTTP %d", name,
resp.StatusCode)
      }
      return nil
}

func (c *WebDAV) Get(name string) ([]byte, error) {
      if !ValidName(name) {
            return nil, ErrBadName
```

```
	}
	resp, err := c.do(http.MethodGet, c.blobURL(name), nil, nil)
	if err != nil {
		return nil, err
	}
	defer resp.Body.Close()
	if resp.StatusCode == http.StatusNotFound {
		return nil, ErrNotExist
	}
	if resp.StatusCode != http.StatusOK {
		return nil, fmt.Errorf("webdav: GET %s: HTTP %d", name,
resp.StatusCode)
	}
	return io.ReadAll(resp.Body)
}

func (c *WebDAV) Delete(name string) error {
	if !ValidName(name) {
		return ErrBadName
	}
	resp, err := c.do(http.MethodDelete, c.blobURL(name), nil, nil)
	if err != nil {
		return err
	}
	defer resp.Body.Close()
	if resp.StatusCode == http.StatusNotFound || resp.StatusCode/100
== 2 {
		return nil
	}
	return fmt.Errorf("webdav: DELETE %s: HTTP %d", name,
resp.StatusCode)
}

const davPropfindBody = `<?xml version="1.0" encoding="utf-8"?>` +
	`<propfind xmlns="DAV:"><prop><resourcetype/></prop></propfind>`

type davMultistatus struct {
	XMLName   xml.Name      `xml:"multistatus"`
	Responses []davResponse `xml:"response"`
}

type davResponse struct {
	Href     string `xml:"href"`
	Propstat []struct {
		Prop struct {
			ResourceType struct {
				Collection *struct{} `xml:"collection"`
			} `xml:"resourcetype"`
		} `xml:"prop"`
```

```
	} `xml:"propstat"`
}

func (r davResponse) isCollection() bool {
	for _, ps := range r.Propstat {
		if ps.Prop.ResourceType.Collection != nil {
			return true
		}
	}
	return false
}

// List returns the leaf blob names in the drop collection. A missing
collection
// (404) lists as empty, per the Carrier contract. Sub-collections and
the
// collection's own self-entry are skipped.
func (c *WebDAV) List() ([]string, error) {
	resp, err := c.do("PROPFIND", c.url(c.dropAbs),
[]byte(davPropfindBody),
		map[string]string{"Depth": "1", "Content-Type":
`application/xml; charset="utf-8"`})
	if err != nil {
		return nil, err
	}
	defer resp.Body.Close()
	if resp.StatusCode == http.StatusNotFound {
		return nil, nil
	}
	if resp.StatusCode != http.StatusMultiStatus {
		return nil, fmt.Errorf("webdav: PROPFIND %s: HTTP %d",
c.dropAbs, resp.StatusCode)
	}
	var ms davMultistatus
	if err := xml.NewDecoder(resp.Body).Decode(&ms); err != nil {
		return nil, fmt.Errorf("webdav: PROPFIND parse: %w", err)
	}
	want := cleanPath(c.dropAbs)
	out := make([]string, 0, len(ms.Responses))
	for _, r := range ms.Responses {
		if cleanPath(r.Href) == want || r.isCollection() {
			continue
		}
		out = append(out, path.Base(cleanPath(r.Href)))
	}
	return out, nil
}
```

```
// cleanPath normalises a WebDAV href to an unescaped, slash-trimmed
absolute
// path so the collection self-entry can be matched against the
request.
func cleanPath(href string) string {
      hp := href
      if u, err := url.Parse(href); err == nil && u.Path != "" {
            hp = u.Path
      }
      if un, err := url.PathUnescape(hp); err == nil {
            hp = un
      }
      return "/" + strings.Trim(hp, "/")
}
```