МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет вычислительной техники

Кафедра «Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Вычислительные машины, комплексы, системы и сети»

БАКАЛАВРСКАЯ РАБОТА

на тему

ГИБРИДНАЯ КРИПТОГРАФИЧЕСКАЯ СХЕМА ДЛЯ ШИФРОВАНИЯ ФАЙЛОВ В TELEGRAM

Студент 03.06.2026 Кащеев Илья Олегович
(подпись, дата) (ФИО полностью)

Руководитель 03.06.2026 СорокинА.А.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр 03.06.2026 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 г. № 14)

Заведующий кафедрой _____ МитрохинМ.А.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 11 июня 2026г. № 8)

Секретарь ГЭК _____ ГуринЕ.И.
(подпись) (фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«07» апреля 2026 г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Кащеев Илья Олегович гр. 22ВВВ1 факультета вычислительной техники.

2.Тема работы: «Гибридная криптографическая схема для шифрования файлов в Telegram»

Тема утверждена приказом ПГУ № 347/о от "7" апреля 2026 г.

3. Руководитель бакалаврской работы доцент кафедры ВТ ПГУ Сорокин А.А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)
Разработать кросс-платформенную систему защищённой передачи файлов через мессенджер Telegram. Система должна реализовывать сквозное шифрование файлов на стороне клиента (алгоритм AES-256-GCM), обмен сеансовыми ключами по схеме RSA-OAEP-SHA256, цифровую подпись отправителя RSA-PSS-SHA256 по схеме encrypt-then-sign, защиту приватного ключа паролем по стандарту PKCS8, JWT-аутентификацию для управления доступом к API, ограничение частоты, верификацию Chat ID через Telegram Bot API и логирование всех операций. Серверная часть разрабатывается на FastAPI и разворачивается на облачной платформе Railway. Клиентская часть реализуется в виде кросс-платформенного GUI-приложения на tkinter.

5. Перечень подлежащих разработке вопросов
Провести анализ существующих решений в области защищённой передачи файлов и обосновать актуальность разработки. Определить требования к системе и спроектировать клиент-серверную архитектуру с разделением криптографических операций и серверной логики. Выполнить выбор языка программирования, криптографических алгоритмов, библиотек и среды разработки. Разработать криптографический модуль и шифрования ключа паролем. Разработать серверное приложение на FastAPI с JWT-аутентификацией, защитой от перезаписи ключей, rate limiting и верификацией Chat ID. Разработать кросс-платформенное клиентское приложение с графическим интерфейсом. Развернуть серверную часть на облачной платформе Railway. Провести функциональное тестирование по сценариям использования и оценить работоспособность системы.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	32	07.04.2026 – 11.04.2026	
2	Анализ аналогов, актуальности, постановка задачи	24	11.04.2026 – 14.04.2026	
3	Выбор и настройка средств разработки	4	14.04.2026 – 15.04.2026	
4	Проектирование архитектуры приложения	30	15.04.2026 – 19.04.2026	
5	Разработка криптографического модуля и серверной части	100	19.04.2026 – 04.05.2026	
6	Разработка клиентского приложения и GUI	55	04.05.2026 – 11.05.2026	
7	Тестирование и развёртывание на Railway	54	11.05.2026 – 18.05.2026	
8	Оформление пояснительной записки	25	18.05.2026 – 21.05.2026	

Дата выдачи задания " 07 " апреля 2026г.

Руководитель бакалаврской работы 07.04.2026 Сорокин А.А.
(подпись, дата)

Задание к исполнению принял студент 07.04.2026 Кащеев И.О.
(подпись, дата)

Работу к защите допустить
Декан факультета 10.06.2026 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 80 листов, 27 рисунков, 5 таблиц, 16 источников, 3 приложения.

ЗАЩИЩЁННАЯ ПЕРЕДАЧА ФАЙЛОВ, СКВОЗНОЕ ШИФРОВАНИЕ, ЦИФРОВАЯ ПОДПИСЬ, TELEGRAM BOT API, JWT-АУТЕНТИФИКАЦИЯ, ГРАФИЧЕСКИЙ ИНТЕРФЕЙС, БАЗА ДАННЫХ

Цель работы заключается в разработке кросс-платформенной системы защищённой передачи файлов через Telegram с реализацией сквозного шифрования, цифровой подписи и безопасного управления ключами.

Объектом исследования являются процессы защищённой передачи файлов в мессенджерах с использованием криптографических методов и клиент-серверной архитектуры.

В работе применены методы гибридного шифрования AES-256-GCM и RSA-OAEP-SHA256, цифровой подписи RSA-PSS-SHA256 по схеме encrypt-then-sign, защиты ключа паролем PKCS8+PBKDF2, ограничения частоты запросов методом скользящего окна и функционального тестирования по сценариям использования.

В работе проведён анализ существующих решений и выявлены их ограничения, разработана система Secure Telegram Sender с шифрованием на стороне клиента, верификацией подписи отправителя, JWT-аутентификацией, защитой от перезаписи ключей, ratelimiting, верификацией Chat ID и развёртыванием на платформе Railway. Тестирование охватывает 10 сценариев использования.

					ПГУ 1.090301.04.001 ПЗ			
Изм.	Лист	№докум.	Подпись	Дата				
Разраб.		Кащеев И.О		03.06.2026	Гибридная криптографическая схема для шифрования файлов в Telegram Пояснительная записка	Лит.	Лист	Листов
Провер.		Сорокин А. А.		03.06.2026			2	80
Н. Контр.		Бычков А.С.		03.06.2026		ФВТ, каф. ВТ гр. 22ВВВ1		
Утверд.		Митрохин М.А.						

Содержание

Перечень сокращений и обозначений 5
Введение 7
1 Анализ предметной области 9
1.1 Анализ решения поставленной задачи 9
1.2 Анализ актуальности разработки 10
1.3 Анализ аналогичных решений 11
1.4 Постановка задачи 14
1.5 Описание архитектуры проекта 15
2 Выбор инструментов разработки 17
2.1 Выбор языка программирования 17
2.2 Выбор криптографических алгоритмов 17
2.3 Выбор серверного фреймворка 18
2.4 Выбор СУБД 18
2.5 Выбор среды разработки 18
3 Реализация 21
3.1 Структура проекта 21
3.2 Реализация криптографического модуля 21
3.2.1 Шифрование приватного ключа паролем 21
3.2.2 Форматбинарногопакета 22
3.2.3 AES-256-GCM шифрование 22
3.3 Реализация JWT-аутентификации 22
3.4 Защита регистрации от перезаписи ключа 23
3.5 Реализацияratelimiting 24
3.6 Логирование 24
3.7 Верификация Chat ID 24
3.8 Реализация клиентского приложения 25
3.8.1 Управление паролем и ключами 25
3.8.2 Процесс отправки файла 25
3.8.3 DPI-масштабирование 25
3.9 Установка и запуск 25

4 Руководство пользователя.....27
4.1 Требования к системе.....27
4.2 Первый запуск – создание ключей.....27
4.3 Последующие запуски – ввод пароля.....29
4.4 Регистрация в системе.....29
4.5 Отправка зашифрованного файла.....31
4.6 Расшифровка полученного файла.....33
4.7 Обновление ключевой пары.....35
5 Тестирование системы.....36
5.1 Типичные ошибки и пути их решения.....37
5.2 Рекомендации к дальнейшей разработке.....38
6 Безопасность жизнедеятельности.....39
6.1 Правила безопасности при использовании ПК.....39
6.2 Электробезопасность.....40
Заключение.....41
Список использованных источников.....42
Приложение А – UML-диаграммы.....44
Приложение Б – Листинг фрагментов программы.....49
Приложение В – Презентация.....76

Перечень сокращений и обозначений

В настоящей работе применяются следующие сокращения и обозначения

БД – база данных

СУБД – система управления базами данных

AES (Advanced Encryption Standard) – симметричный алгоритм блочного шифрования данных с длиной ключа 256 бит

AES-GCM (AES Galois/Counter Mode) – режим работы AES, обеспечивающий шифрование и аутентификацию данных посредством тега целостности

API (Application Programming Interface) – программный интерфейс взаимодействия компонентов системы

asyncio – стандартная библиотека Python для написания конкурентного кода с использованием синтаксиса async/await

Bot API – интерфейс программирования приложений мессенджера Telegram для создания и управления ботами

E2EE (End-to-End Encryption) – сквозное шифрование, при котором данные доступны исключительно отправителю и получателю

FastAPI – высокопроизводительный веб-фреймворк для создания HTTP API на языке Python, основанный на стандарте ASGI

GUI (Graphical User Interface) – графический интерфейс пользователя

HTTP (Hyper Text Transfer Protocol) – протокол передачи данных в сети Интернет

httpx – асинхронная HTTP-клиентская библиотека для Python

IDE (Integrated Development Environment) – интегрированная среда разработки

JSON (JavaScript Object Notation) – текстовый формат обмена данными

JWT (JSON Web Token) – открытый стандарт (RFC 7519) для передачи информации в виде JSON-объекта, защищённого цифровой подписью

MITM (Man-in-the-Middle) – атака типа «человек посередине», при которой злоумышленник перехватывает канал связи

OAEP (Optimal Asymmetric Encryption Padding) – схема дополнения для RSA-шифрования, обеспечивающая семантическую безопасность

PBKDF2 (Password-Based Key Derivation Function 2) – функция формирования ключа из пароля, устойчивая к атакам перебора

PEM (Privacy Enhanced Mail) – формат кодирования криптографических объектов на основе Base64

PKCS8 – стандарт синтаксиса для хранения закрытых ключей, в том числе с шифрованием паролем

PSS (Probabilistic Signature Scheme) – вероятностная схема формирования цифровой подписи на основе RSA

PyCharm – интегрированная среда разработки от JetBrains для языка Python

Python-dotenv – библиотека для загрузки переменных окружений из файла .env

Railway – облачная платформа для развёртывания серверных приложений

Ratelimiting – ограничение частоты запросов к серверу для защиты от злоупотреблений

RSA – асимметричный криптографический алгоритм для шифрования данных и формирования цифровой подписи

slidingwindow – алгоритм скользящего окна для подсчёта событий за фиксированный временной интервал

SQLite – встраиваемая реляционная СУБД, хранящая всю базу данных в одном файле

tkinter – стандартная библиотека Python для создания графических интерфейсов

TLS (Transport Layer Security) – криптографический протокол защиты транспортных соединений

UML (Unified Modeling Language) – унифицированный язык моделирования

uvicorn – высокопроизводительный ASGI-сервер для Python-приложений

WAL (Write-Ahead Logging) – режим журналирования SQLite для параллельных операций

Введение

В современном мире передача конфиденциальных файлов через публичные мессенджеры стала повседневной практикой как в корпоративной, так и в личной среде. Telegram занимает одно из лидирующих положений среди мессенджеров благодаря открытому Bot API и широким возможностям автоматизации. Однако стандартный режим работы Telegram не обеспечивает сквозного шифрования файлов, передаваемых через ботов: файлы хранятся на серверах платформы.

Анализ практики применения Telegram-ботов показывает несколько системных уязвимостей. Во-первых, токен бота нередко хранится непосредственно в клиентском приложении, что при компрометации любого устройства ведёт к полной утрате контроля над ботом. Во-вторых, отсутствие механизмов аутентификации пользователей позволяет злоумышленнику перезаписать чужой публичный ключ. В-третьих, без цифровой подписи получатель не может верифицировать подлинность отправителя и убедиться в целостности полученного файла.

В данной выпускной квалификационной работе разрабатывается система защищённой передачи файлов с использованием Telegram в качестве транспортного канала. Система реализует комплекс мер: сквозное шифрование AES-256-GCM, обмен ключами RSA-OAEP-SHA256, цифровую подпись RSA-PSS-SHA256, JWT-аутентификацию, защиту приватного ключа паролем по стандарту PKCS8, ratelimiting методом скользящего окна, верификацию Chat ID через Telegram Bot API и логирование всех операций.

Объектом исследования является процесс защищённой передачи файлов в мессенджерах с использованием криптографических методов и клиент-серверной архитектуры.

Предметом исследования являются методы и программные средства реализации сквозного шифрования, аутентификации пользователей, управления криптографическими ключами и обеспечения безопасности серверной инфраструктуры.

Целью данной работы является разработка кросс-платформенной системы защищённой передачи файлов через Telegram, реализующей E2EE, цифровую подпись, JWT-аутентификацию и централизованное управление открытыми ключами с безопасным хранением токена бота на сервере.

Для достижения поставленной цели необходимо решить следующие задачи:

- провести анализ предметной области и существующих решений в области защищённой передачи файлов;
- изучить архитектуру Telegram Bot API и возможности его использования в качестве транспортного канала;
- разработать криптографический модуль с поддержкой AES-256-GCM, RSA-OAEP, RSA-PSS и шифрования ключа паролем;
- реализовать JWT-аутентификацию с выдачей токена при регистрации и его проверкой при отправке файлов;
- разработать защиту эндпоинта регистрации от перезаписи чужого ключа;
- реализовать ratelimiting методом скользящего окна без сторонних зависимостей;
- реализовать логирование событий в файл и базу данных;
- добавить верификацию Chat ID через Telegram Bot API при регистрации;
- разработать кросс-платформенное клиентское приложение с графическим интерфейсом;
- провести тестирование и оценку разработанного решения.

Практическая значимость работы состоит в создании готового к применению инструмента, устраняющего ключевые уязвимости типовых решений на основе Telegram-ботов.

1 Анализ предметной области

Для разработки качественного программного решения, обеспечивающего защищённую передачу файлов, необходимо детально изучить предметную область, охватывающую принципы криптографической защиты данных, архитектуру современных мессенджеров и существующие подходы к управлению ключами.

Мессенджеры являются одним из ключевых инструментов цифровой коммуникации. Telegram занимает особое место среди них благодаря открытому Bot API. Для передачи файлов через ботов используется метод sendDocument, который загружает файл на серверы Telegram. Таким образом, при стандартном использовании Bot API содержимое файлов доступно инфраструктуре платформы [1].

Анализ практики разработки Telegram-ботов выявил типичные архитектурные ошибки. Наиболее распространённой является хранение токена бота в клиентском приложении. При компрометации любого устройства злоумышленник получает полный доступ к боту. Второй проблемой является отсутствие аутентификации: без проверки идентичности запрашивающего клиента любой желающий может перезаписать чужой публичный ключ. Третьей проблемой является отсутствие ratelimiting, что открывает возможность для атак перебором.

1.1 Анализ решения поставленной задачи

В данной работе предлагается реализовать механизм сквозного шифрования файлов с передачей через Telegram. Все криптографические операции выполняются исключительно на стороне клиента – сервер получает уже зашифрованный пакет и не имеет доступа к содержимому файла.

Для шифрования содержимого файлов используется AES-256-GCM. Режим GCM обеспечивает аутентифицированное шифрование: 16-байтовый тег позволяет обнаружить любое изменение содержимого. Сеансовый ключ

шифруется открытым ключом получателя по схеме RSA-OAEP-SHA256. Аутентификация отправителя обеспечивается цифровой подписью RSA-PSS-SHA256 по схеме encrypt-then-sign.

JWT-аутентификация реализована для управления доступом к API. После регистрации сервер выдаёт токен со сроком действия 24 часа. Эндпоинт /send принимает запросы только при наличии действительного токена. Приватный ключ пользователя на диске защищён паролем: используется BestAvailableEncryption (AES-256-CBC + PBKDF2).

Токен бота хранится исключительно на сервере в переменной окружения TELEGRAM_BOT_TOKEN, что исключает его компрометацию через клиентские устройства.

1.2 Анализ актуальности разработки

Актуальность разработки обусловлена несколькими факторами. Во-первых, несмотря на широкое распространение Telegram как корпоративного инструмента, передача файлов через Bot API лишена встроенных механизмов E2EE. Файлы хранятся на серверах платформы и доступны при компрометации инфраструктуры.

Во-вторых, токены Telegram-ботов регулярно обнаруживаются в открытом доступе в репозиториях GitHub. Предлагаемая архитектура, при которой токен хранится только на сервере в переменной окружения, полностью устраняет данный вектор атаки.

В-третьих, без ограничения частоты запросов серверные приложения уязвимы для атак перебором и DoS. Ratelimiting методом скользящего окна, реализованный в данной системе без внешних зависимостей, демонстрирует самодостаточный подход к защите.

В-четвёртых, рост числа атак MITM делает механизм верификации отправителя посредством цифровой подписи особенно востребованным.

1.3 Анализ аналогичных решений

Для определения требований к разрабатываемой системе проведён анализ существующих решений в области защищённой передачи файлов.

Signal реализует E2EE по умолчанию по протоколу Signal (DoubleRatchet + X3DH). Является монолитным приложением: пользователь не может управлять собственными ключами вне приложения или интегрировать его с Bot API. Интерфейс сервиса показан на рисунке 1.1

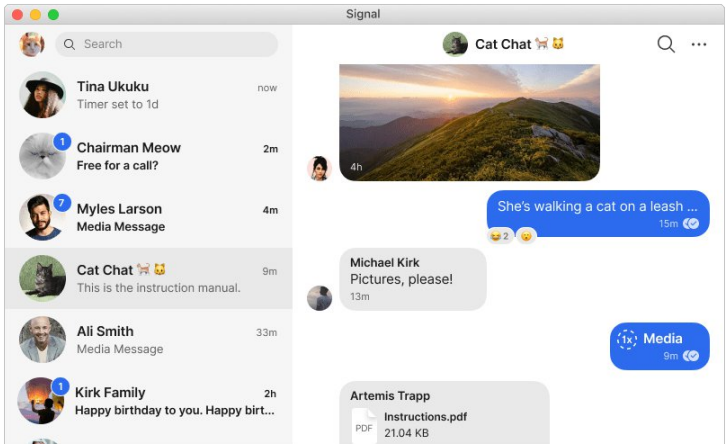


Рисунок 1.1 – Интерфейс мессенджера Signal

Telegram Secret Chats – встроенный механизм E2EE на базе MTProto 2.0. Ключевым ограничением является недоступность через Bot API: все файлы, передаваемые через ботов, обрабатываются в облачном режиме без E2EE. Интерфейс чата показан на рисунке 1.2

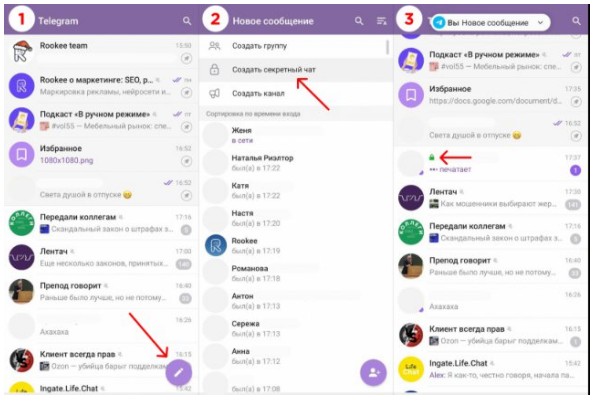


Рисунок 1.2 – Интерфейс Telegram Secret Chats

GPG реализует стандарт OpenPGP и предоставляет полный контроль над ключами. Существенные недостатки: отсутствие удобного кросс-платформенного GUI, высокий порог входа, необходимость отдельного канала для передачи файлов. Интерфейс приложения показан на рисунке 1.3

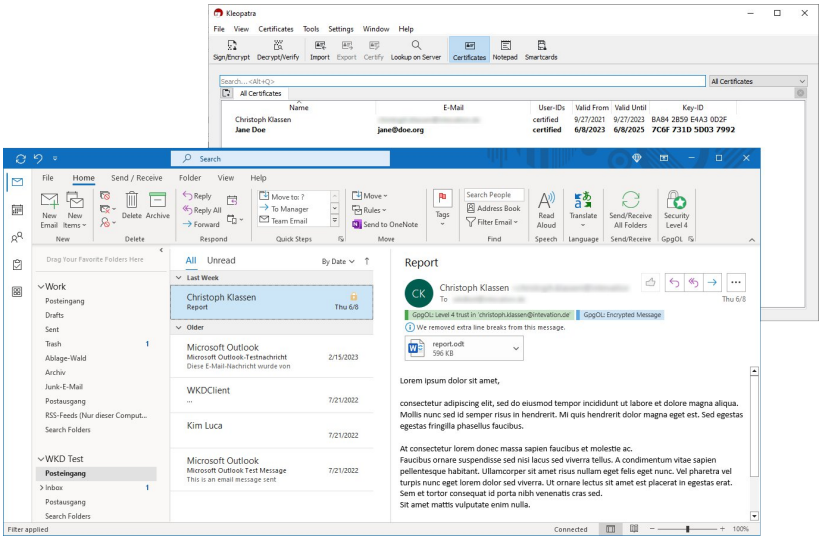


Рисунок 1.3 – Интрефейс GNUPrivacyGuard

VeraCrypt обеспечивает надёжное симметричное шифрование контейнеров. Не решает задачу аутентификации отправителя и требует ручной передачи пароля по безопасному каналу. Интерфейс инструмента показан на рисунке 1.4

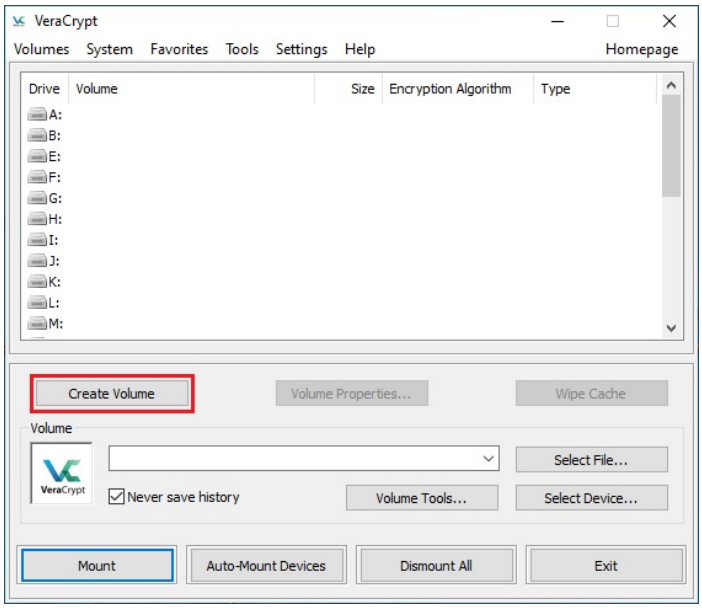


Рисунок 1.4 – ИнтрефейсVeraCrypt

ProtonMailDrive предоставляет E2EE хранилище файлов. Недостатки: привязка к экосистеме ProtonMail, отсутствие Telegram-интеграции, платные тарифы. Интерфейс приложения показан на рисунке 1.5

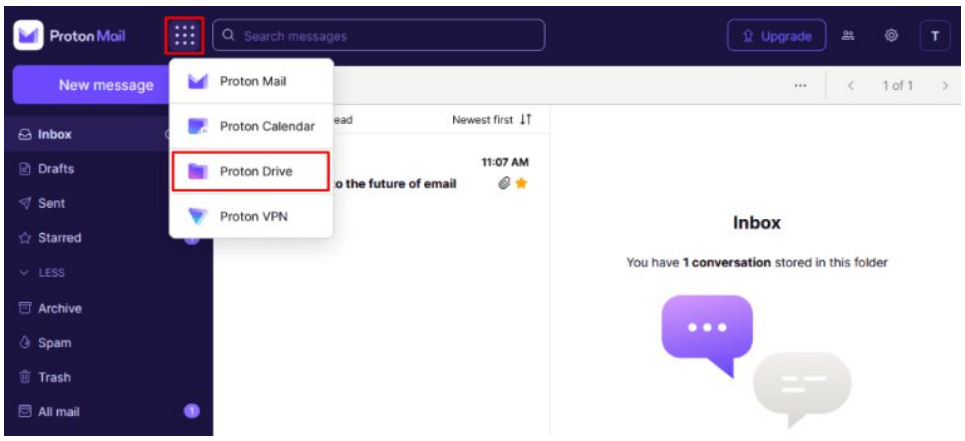


Рисунок 1.5 – Интрефейс Proton Mail Drive

Сводное сравнение рассмотренных аналогов представлено в таблице 1.1.

Таблица 1.1 – Сравнение аналогичных решений

Решение	Преимущества	Недостатки
Signal	E2EE по умолчанию, открытый код, нет серверного хранения ключей	Нет интеграции с Bot API, нельзя управлять собственными ключами
Telegram SecretChats	E2EE на уровне MTProto 2.0, встроен в Telegram	Недоступен через Bot API, нельзя передавать файлы через ботов зашифровано
GPG + email/FTP	Полный контроль над ключами, стандарт OpenPGP, подпись	Сложность настройки, нет удобного GUI, требует отдельного канала передачи
VeraCrypt + облако	Сильное симметричное шифрование	Нет аутентификации отправителя, ручная передача пароля
Решение	Преимущества	Недостатки

ProtonMailDrive	E2EE, удобный интерфейс	Нет Telegram-интеграции, привязка к экосистеме ProtonMail, платные тарифы

Сравнительный анализ функциональных возможностей представлен в таблице 1.2.

Таблица 1.2 – Сравнение функциональных возможностей

Функция	Моя система	Signal	GPG+FTP	Telegram SC
E2EE файлов	AES-256-GCM	Да	Да	Нет (Bot API)
Цифровая подпись	RSA-PSS-SHA256	Нет явной	Да (OpenPGP)	Нет
JWT-аутентификация	Да	N/A	N/A	N/A
Защита токена бота	Только на сервере	N/A	N/A	N/A
Ratelimiting	Slidingwindow	N/A	N/A	N/A
Логирование	Файл + БД	Нет	Нет	Нет
Пароль на ключ	PKCS8+PBKDF2	Нет	Да	Нет
Кросс-платформа	macOS/Win/Linux	Да	Да	Да

Проведённый анализ показал, что ни одно из рассмотренных решений не сочетает все необходимые средства защиты.

1.4 Постановка задачи

На основании проведённого анализа сформулированы следующие требования к разрабатываемой системе.

Функциональные требования к клиентскому приложению:

- генерация RSA-2048 ключей с сохранением приватного ключа, зашифрованного паролем (PKCS8);
- диалог ввода пароля при запуске с тремя попытками;
- регистрация с передачей открытого ключа и Chat ID, получение JWT-токена;
- обновление ключа с подписью нового публичного ключа старым приватным;
- шифрование AES-256-GCM и подпись RSA-PSS-SHA256 по схеме encrypt-then-sign;
- передача пакета на сервер с JWT-токеном; расшифровка с верификацией подписи.

Функциональные требования к серверному приложению:

- хранение открытых ключей и Chat ID в SQLite (WAL); выдача JWT при регистрации;
- проверка old_key_signature при попытке перезаписи ключа;
- верификация Chat ID через Telegram Bot API (getChat);
- rate limiting: 5/60сдля /register, 20/60сдля /send, 60/60сдля /user;
- JWT-аутентификация для /send; логирование в файл и таблицу send_log.

1.5 Описание архитектуры проекта

Клиентское приложение реализовано на языке Python с использованием стандартной библиотеки tkinter для создания графического интерфейса. Клиент полностью автономен в части криптографии: все операции шифрования, подписи и верификации выполняются локально без передачи приватного ключа или незашифрованного содержимого на сервер. Клиент хранит два постоянных файла: private_key.pem – приватный ключ пользователя, зашифрованный паролем по стандарту PKCS8, и auth_token.txt – JWT-токен, полученный при регистрации. Взаимодействие с сервером осуществляется через три HTTP-эндпоинта: POST /register для регистрации, GET /user/{username} для получения публичного ключа и POST /send для передачи зашифрованного пакета.

Серверное приложение реализовано на фреймворке FastAPI с ASGI-сервером uvicorn. Архитектура включает четыре логических слоя: middleware rate limiting (HTTP 429 при превышении лимитов), JWT-аутентификация через FastAPI Depends, бизнес-логика эндпоинтов и хранилище данных SQLite с режимом WAL При регистрации клиент отправляет POST /register, сервер проверяет Chat ID через Telegram Bot API и возвращает JWT-токен сроком 24 часа.

При отправке файла клиент выполняет криптографические операции локально и передаёт зашифрованный пакет через POST /send с JWT в заголовке Authorization. Сервер верифицирует токен, логирует операцию в server.log и таблицу send_log и доставляет пакет в Telegram через asyncio Bot API. На рисунке 1.6 представлена UML-диаграмма компонентов системы:

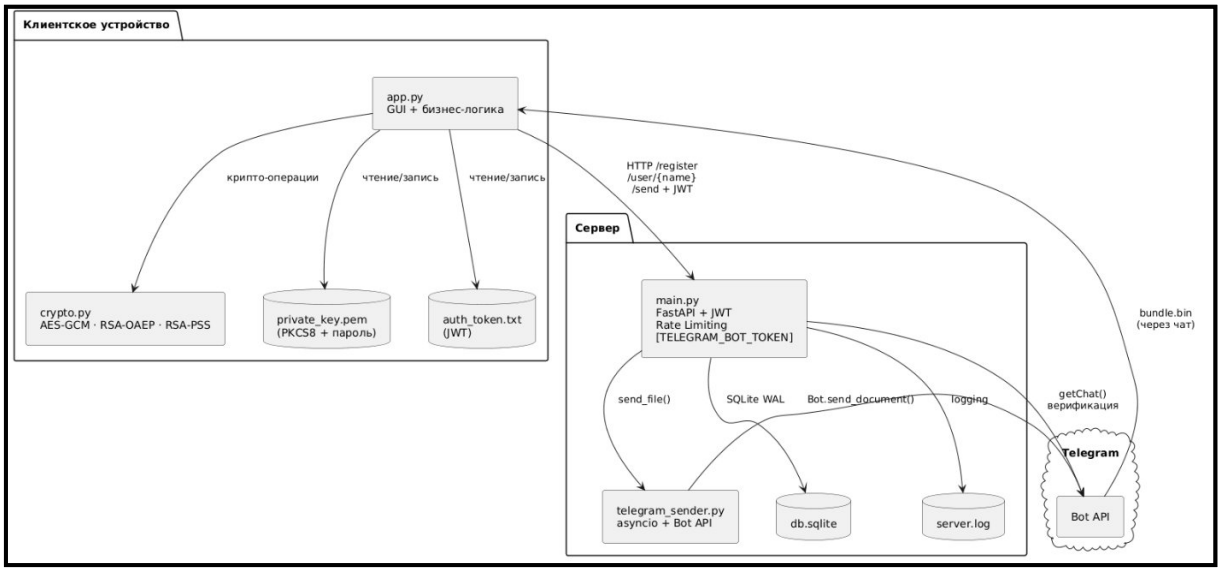


Рисунок 1.6 - Диаграмма компонентов системы

2 Выбор инструментов разработки

При разработке системы защищённой передачи файлов необходимо обоснованно подойти к выбору языка программирования, криптографических алгоритмов и библиотек, фреймворков серверной части и средств разработки.

2.1 Выбор языка программирования

Для реализации как клиентской, так и серверной части выбран Python версии 3.11. Данный выбор обоснован рядом факторов.

Python обеспечивает кросс-платформенное выполнение кода на macOS, Windows и Linux без перекомпиляции. Экосистема Python предоставляет высококачественные библиотеки для всех задач: cryptography для криптографии, FastAPI для серверной части, python-telegram-bot [2] для Bot API [3], tkinter для GUI.

В сравнении с Java Python обеспечивает более быструю разработку и меньший объём шаблонного кода. По сравнению с C# обеспечивает полную кросс-платформенность. По сравнению с C++ значительно упрощает разработку GUI при достаточной производительности для криптографических операций, которые в библиотеке cryptography реализованы на C через OpenSSL.

2.2 Выбор криптографических алгоритмов

AES-256-GCM выбран для симметричного шифрования файлов. Режим GCM обеспечивает аутентифицированное шифрование с тегом 16 байт. В отличие от CFB, устойчив к атакам подмены байт. Рекомендован NIST SP 800-38D [4,5].

RSA-OAEP-SHA256 выбран для шифрования сеансового ключа. Схема OAEP обеспечивает семантическую безопасность RSA в отличие от устаревшей PKCS#1 v1.5. Длина ключа 2048 бит соответствует рекомендациям NIST до 2030 года [6].

RSA-PSS-SHA256 выбран для цифровой подписи. Схема PSS является рекомендованной PKCS#1 v2.1 схемой с доказуемой стойкостью. Используется максимальная длина соли.

BestAvailableEncryption из cryptography выбран для защиты ключа на диске: AES-256-CBC с деривацией PBKDF2-HMAC-SHA256, стандарт PKCS#8 [7].

Схема encrypt-then-sign: сначала шифрование, затем подпись над шифртекстом. Это исключает атаки, при которых злоумышленник мог бы переупаковать подписанные данные в другой шифртекст [8].

2.3 Выбор серверного фреймворка

Для серверной части выбран FastAPI 0.104.1. Фреймворк обеспечивает асинхронную обработку через ASGI, автоматическую валидацию данных через Pydantic [9, 10], встроенный Dependency Injection для JWT-аутентификации.

В качестве ASGI-сервера используется uvicorn 0.24.0. Для JWT – PyJWT 2.8.0 [11]. Для верификации Chat ID – httpx (асинхронные HTTP-запросы к Telegram Bot API). Для загрузки переменных окружения из файла .env используется библиотека python-dotenv 1.0.0.

2.4 Выбор СУБД

Для хранения данных выбрана SQLite: не требует отдельного серверного процесса, встроена в Python и обеспечивает достаточную производительность для ожидаемой нагрузки.

Для корректной работы при параллельных запросах используются: режим WAL (параллельное чтение без блокировок) и threading.local (каждый поток получает собственное соединение). В базе создаются таблицы users и send_log [12].

2.5 Выбор среды разработки

Для разработки системы использовалась интегрированная среда разработки PyCharm от JetBrains [13]. Данная IDE специализирована для языка Python и

обеспечивает глубокую интеграцию со всеми применяемыми в проекте инструментами и библиотеками. По сравнению с универсальными редакторами, такими как VS Code, PyCharm предоставляет более глубокую поддержку Python без необходимости установки дополнительных расширений: инспекции кода, поиск неиспользуемых импортов и потенциальных ошибок работают «из коробки» сразу после установки.

Ключевые возможности PyCharm, использованные в проекте:

- интеллектуальный редактор кода с автодополнением, подсветкой синтаксиса и рефакторингом, специфичным для Python;
- интеграция с виртуальными окружениями venv: PyCharm автоматически обнаруживает окружения в папках client/ и server/ и переключается между ними без ручной настройки;
- встроенный отладчик Python с поддержкой точек останова, просмотра переменных и пошагового выполнения — активно использовался при отладке криптографических операций и проверке корректности формирования бинарного пакета;
- интеграция с системой контроля версий Git для управления историей изменений проекта;
- встроенный терминал для одновременного запуска сервера и клиента в раздельных вкладках без переключения между окнами;
- поддержка FastAPI: распознавание аннотаций Pydantic, подсказки по типам и навигация между эндпоинтами непосредственно из редактора.

Для тестирования HTTP API использовался терминальный вывод сервера uvicorn совместно с файлом server.log. При каждом входящем запросе uvicorn выводит HTTP-метод, путь и код ответа, что позволяло оперативно отслеживать корректность работы эндпоинтов и выявлять ошибки на этапе разработки. Функциональное тестирование проводилось непосредственно через клиентское приложение по сценариям, описанным в разделе 5. Функция «Run/Debug Configurations» позволила настроить параллельный запуск серверной и

клиентской частей с разными рабочими каталогами в рамках одной IDE, что показано на рисунке 2.1.

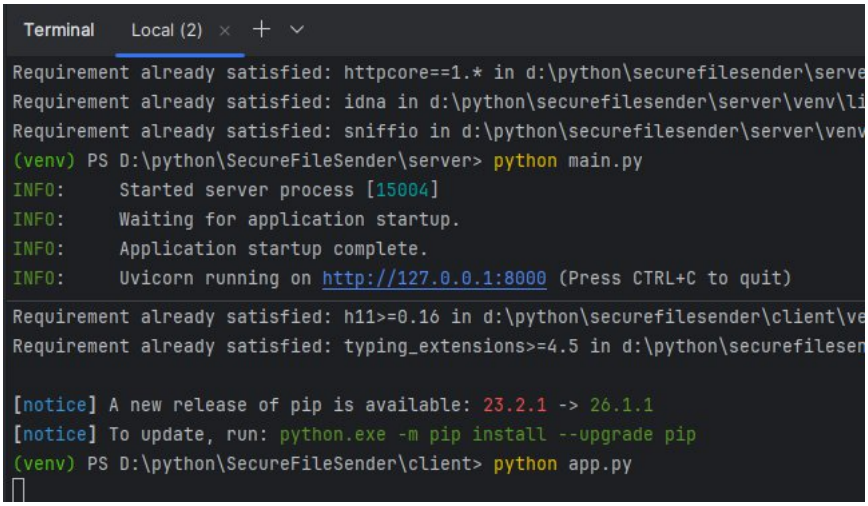


Рисунок 2.1 – Параллельный запуск

3 Реализация

В данном разделе описывается структура программного проекта, детали реализации всех компонентов системы, а также особенности обеспечения безопасности и надёжности.

3.1 Структура проекта

Проект разделён на две независимые части: клиент и сервер, каждая со своими зависимостями.

В клиентской части находятся два основных файла. Первый – app.py, это само приложение с графическим интерфейсом: окно с вкладками, кнопки, диалоги ввода пароля. Второй – crypto.py, здесь сосредоточена вся криптография: генерация ключей, шифрование файлов, подпись и упаковка данных в пакет. Помимо кода там хранятся два рабочих файла: private_key.pem – зашифрованный паролем приватный ключ пользователя, и auth_token.txt – JWT-токен который приложение получает при регистрации и использует при каждой отправке.

В серверной части три модуля. Главный – main.py, это сам сервер: принимает запросы, проверяет токены, ограничивает частоту запросов, верифицирует Chat ID. Второй – telegram_sender.py, занимается только одним – отправляет зашифрованный файл в Telegram получателю. Третий – копия crypto.py с клиента, нужна серверу чтобы проверять подписи при смене ключей. Данные хранятся в файле db.sqlite – там публичные ключи пользователей и история отправок. Все события сервера пишутся в server.log

3.2 Реализация криптографического модуля

Криптографический модуль crypto.py является центральным компонентом системы и не имеет зависимостей от GUI или сетевого взаимодействия.

3.2.1 Шифрование приватного ключа паролем

Функция serialize_private принимает необязательный параметр password. При его наличии используется BestAvailableEncryption – AES-256-CBC +

PBKDF2-HMAC-SHA256. Результатом является PEM-файл PKCS8 с заголовком ENCRYPTED PRIVATE KEY [14].

```
def serialize_private(priv, password: bytes | None = None) -> bytes:
    encryption = (
serialization.BestAvailableEncryption(password)
        if password else serialization.NoEncryption()
    )
    return priv.private_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PrivateFormat.PKCS8,
encryption_algorithm=encryption)
```

3.2.2 Форматбинарногопакета

Форматпакета: [4 байта: len(enc_key)][enc_key][4 байта: len(sig)][sig][4 байта: len(filename)][filename][encrypted…]. Добавлено поле с оригинальным именем файла, что позволяет при расшифровке автоматически восстанавливать исходное имя и расширение файла без необходимости переименования вручную.

3.2.3 AES-256-GCM шифрование

Для каждого вызова encrypt_file генерируется случайный nonce 12 байт через os.urandom(12). Результат: nonce + шифртекст + тег GCM (16 байт). При расшифровке несоответствие тега выбрасывает InvalidTag.

3.3 Реализация JWT-аутентификации

Функция _create_token формирует JWT с полями sub (имя пользователя), iat (время выдачи) и exp (время истечения) и подписывает его алгоритмом HS256 [15]. Функция get_current_user используется как FastAPIDepends-зависимость: автоматически извлекает токен из заголовка Authorization и возвращает имя пользователя. Последовательность взаимодействия компонентов при аутентификации представлена на рисунке 3.1. Таким образом, при каждом обращении к эндпоинту /send сервер автоматически проверяет токен и извлекает имя отправителя, которое фиксируется в логах и отображается получателю в подписи к файлу.

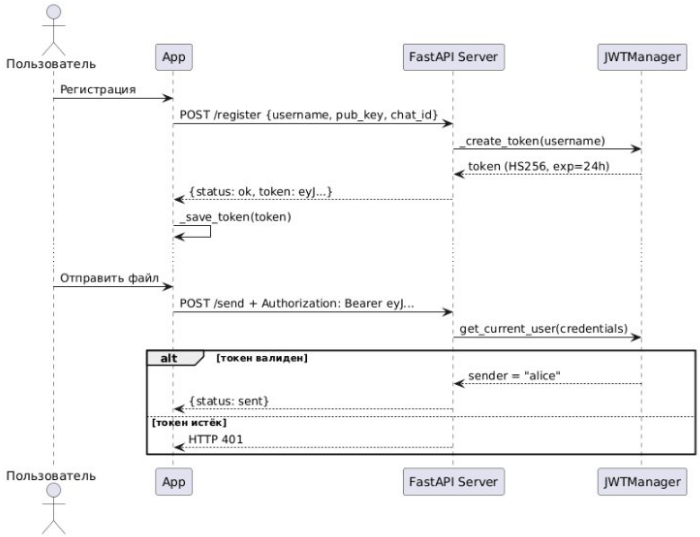


Рисунок 3.1 – Диаграмма последовательностей (JWT-аутентификация)

3.4 Защита регистрации от перезаписи ключа

При попытке зарегистрировать занятое имя без old_key_signature сервер возвращает HTTP 409. При наличии подписи – загружает старый публичный ключ из БД и вызывает verify_signature. В клиенте кнопка «Обновить ключ» автоматически подписывает новый публичный ключ старым приватным, формируя поле old_key_signature в формате base64 и передавая его на сервер вместе с новым ключом. Последовательность взаимодействия компонентов при обновлении ключа представлена на рисунке 3.2.

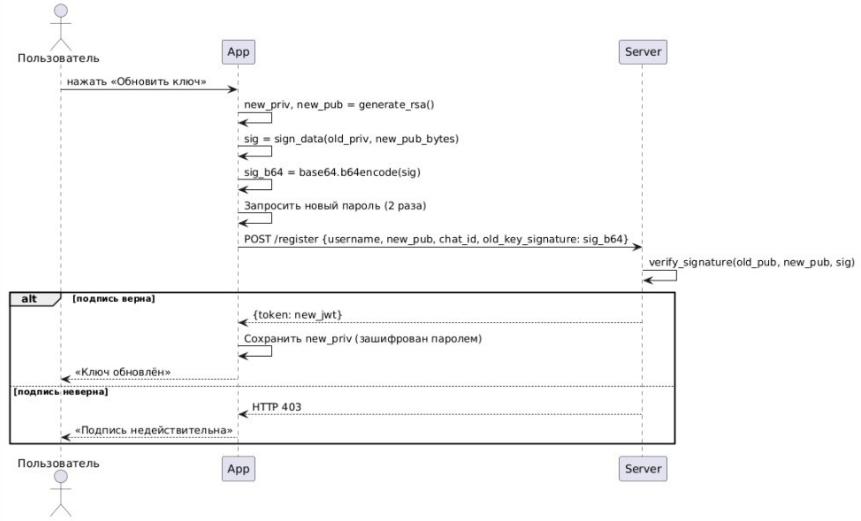


Рисунок 3.2 – Диаграмма последовательностей (обновление ключа)

3.5 Реализацияratelimiting

Ratelimiting реализован как FastAPImiddleware на основе алгоритма скользящего окна. Для каждой пары (IP, путь) хранится список временных меток. При превышении лимита возвращается HTTP 429. threading.Lock обеспечивает потокобезопасность; time.monotonic() защищает от перевода системных часов.

```
def _check_rate_limit(ip: str, path: str) -> None:
    now = time.monotonic()
    key = f"{ip}:{path}"
    with _rate_lock:
        _rate_store[key] = [t for t in _rate_store[key]
                            if now - t < window]
        if len(_rate_store[key]) >= limit:
            raise HTTPException(429, ...)
        _rate_store[key].append(now)
```

3.6 Логирование

Файловый лог server.log формируется через стандартный модуль logging с FileHandlerиStreamHandler. Логируются: регистрации, превышения ratelimit, ошибки проверки Chat ID, выдача токенов и каждая отправка файла. Таблица send_log в SQLite хранит записи об отправках без содержимого файлов.

2026-05-06 12:34:01 INFO New register: username=aliceip=127.0.0.1

2026-05-06 12:34:01 INFO Token issued: username=alice

2026-05-06 12:35:22 INFO Sent: from=alice to=bob size=204800 ip=127.0.0.1

2026-05-06 12:36:00 WARN Rate limit: ip=192.168.1.5 path=/register

3.7 Верификация Chat ID

При регистрации сервер отправляет асинхронный GET-запрос к методу getChat Telegram Bot API через библиотеку httpx с таймаутом 5 секунд. Данная проверка позволяет убедиться, что указанный Chat ID действительно существует в Telegram и пользователь написал боту хотя бы одно сообщение – без этого условия Telegram не позволяет боту отправлять сообщения пользователю. При отсутствии переменной окружения TELEGRAM_BOT_TOKEN проверка пропускается с предупреждением в лог – это обеспечивает возможность

локальной разработки без настройки бота. При временной недоступности серверов Telegram функция возвращает True, не блокируя регистрацию, что предотвращает отказ в обслуживании при сбоях сети. При явном ответе Telegram вида {"ok": false} регистрация отклоняется с HTTP 400 и информативным сообщением о необходимости написать боту @FastSecureFilesBot перед регистрацией.

3.8 Реализация клиентского приложения

Клиентское приложение реализует GUI посредством tkinter с виджетами ttk, организованный в трёх вкладках: «Настройки», «Отправить», «Расшифровать».

3.8.1 Управление паролем и ключами

При первом запуске _generate_keys_with_password запрашивает пароль через simpledialog.askstring с маскировкой (show="*") и его подтверждение. При несовпадении приложение завершается. При последующих запусках _load_keys_with_password даёт три попытки ввода.

3.8.2 Процесс отправки файла

При нажатии кнопки «Выбрать файл и отправить» проверяется наличие токена. Диалог выбора файла открывается в главном потоке, тяжёлые операции выполняются в daemon-потоке threading.Thread. Временный файл создаётся через tempfile и удаляется в блоке finally.

3.8.3 DPI-масштабирование

На macOS: tkscaling 2.0. На Windows: ctypes.windll.shcore.SetProcessDpiAwareness(1) + динамический расчёт масштаба. На Linux: дополнительные действия не требуются.

3.9 Установка и запуск

Серверная часть развёрнута на облачной платформе Railway по адресу https://secure-telegram-sender-production.up.railway.app. Для подключения клиента к удалённому серверу в файле app.py указывается адрес Railway вместо

локального. Переменные окружения TELEGRAM_BOT_TOKEN и JWT_SECRET задаются в разделе Variables панели управления Railway [16].

Для локального запуска сервера переменные окружения хранятся в файле server/.env и загружаются автоматически при наличии файла через библиотеку python-dotenv.

Установка клиентской части:

```
cdclient
python3 -mvenvvenv
source venv/bin/activate
pip install -r requirements.txt
pythonapp.py
```

В PyCharm обе части удобно запускать через RunConfigurations: создаётся две конфигурации с разными рабочими каталогами (client/ и server/), которые запускаются одновременно в разных вкладках терминала IDE.

4 Руководство пользователя

В данном разделе описывается порядок работы с системой защищённой передачи файлов для конечного пользователя. Руководство охватывает все этапы: от первого запуска и регистрации до отправки и расшифровки файлов. Для корректной работы системы необходимо предварительно написать боту @FastSecureFilesBot в Telegram любое сообщение – это обязательное условие, без которого бот не сможет доставлять файлы в чат пользователя.

4.1 Требования к системе

Для работы с клиентским приложением необходимо выполнение следующих условий:

- операционная система: macOS 10.14+, Windows 10/11 или Linux (Ubuntu 20.04+);
- Python 3.11 или выше с установленными зависимостями из client/requirements.txt;
- доступ к серверу системы; сервер развёрнут на облачной платформе Railway и доступен без дополнительной настройки со стороны пользователя;
- наличие аккаунта Telegram, Chat ID (можно узнать у бота @userinfobot) и предварительно отправленное сообщение боту @FastSecureFilesBot – без этого шага Telegram не позволит боту писать пользователю.

Для запуска серверной части дополнительно требуется токен Telegram-бота, полученный через @BotFather, а также учётная запись на платформе Railway для управления переменными окружения.

4.2 Первый запуск – создание ключей

Перед первым запуском необходимо убедиться, что все зависимости установлены командой pipinstall -r requirements.txt в папке client/. При первом запуске приложения файл private_key.pem отсутствует. Приложение автоматически определяет это и отображает информационное окно с

предложением создать новую пару криптографических ключей, как показано на рисунке 4.1. Генерация ключей выполняется однократно – при последующих запусках приложение загружает уже существующий ключ.

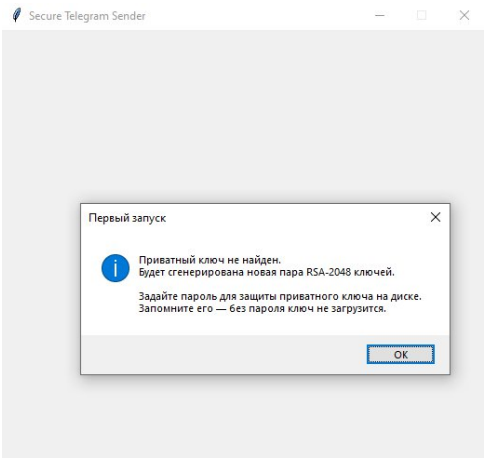


Рисунок 4.1 – Информационное окно при первом запуске

После нажатия «ОК» приложение запрашивает пароль для защиты приватного ключа. Пароль вводится в поле с маскировкой символов (рисунок 4.2). Рекомендуется использовать надёжный пароль длиной не менее 12 символов.

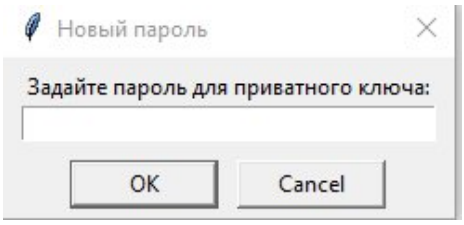


Рисунок 4.2 – Диалог создания пароля для приватного ключа

Затем следует подтверждение пароля (рисунок 4.3). При несовпадении паролей приложение уведомляет об ошибке и завершает работу – необходимо запустить его повторно. При успешном создании пара RSA-2048 ключей генерируется автоматически, приватный ключ сохраняется в файл private_key.pem в зашифрованном виде.

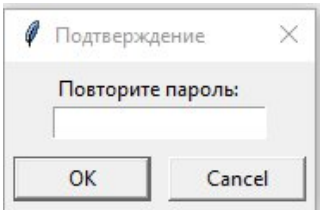


Рисунок 4.3 – Диалог подтверждения пароля

4.3 Последующие запуски – ввод пароля

При каждом следующем запуске приложение запрашивает пароль для расшифровки приватного ключа (рисунок 4.4). Предоставляется три попытки ввода. При трёх неудачных попытках приложение завершает работу с рекомендацией удалить private_key.pem и зарегистрироваться заново.

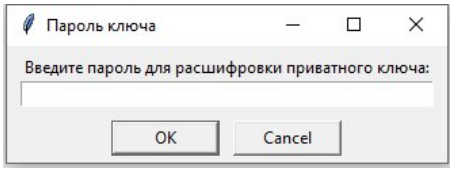


Рисунок 4.4 – Диалог ввода пароля при запуске

4.4 Регистрация в системе

После успешного ввода пароля открывается главное окно приложения с тремя вкладками. Для начала работы необходимо выполнить регистрацию на вкладке «Настройки» (рисунок 4.5).

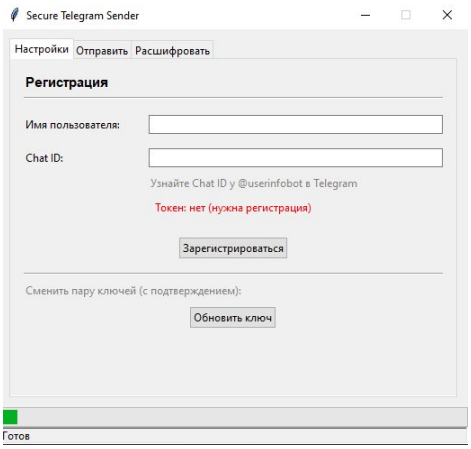


Рисунок 4.5 – Вкладка «Настройки»

Порядок регистрации:

- в поле «Имя пользователя» ввести уникальный никнейм (только латинские буквы, цифры, дефис и точка);
- в поле «Chat ID» ввести числовой идентификатор своего Telegram-аккаунта. Для его получения необходимо написать боту @userinfobot в Telegram — он ответит числом вида 123456789;
- нажать кнопку «Зарегистрироваться».

При успешной регистрации в нижней строке состояния появляется сообщение «Зарегистрирован как «имя»», а статус токена обновляется до «Токен: получен» (рисунок 4.6). JWT-токен автоматически сохраняется в файл auth_token.txt.

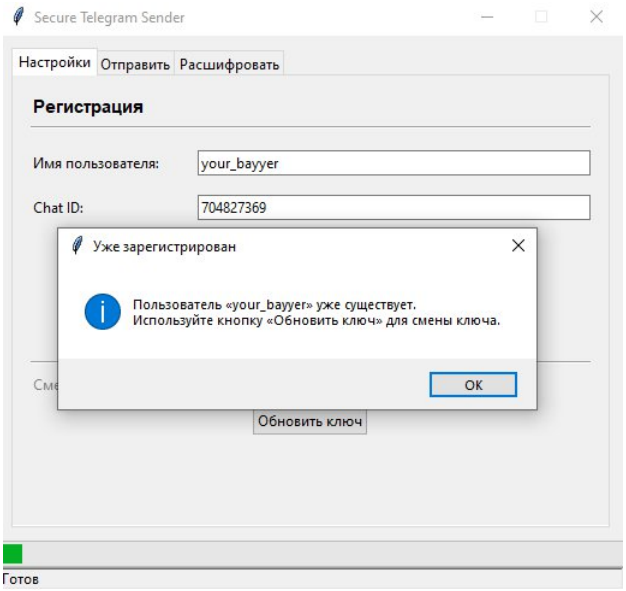


Рисунок 4.6 – Успешная регистрация (токен получен)

Если имя пользователя уже занято (рисунок 4.7), сервер возвращает соответствующее сообщение с кодом 409. В этом случае следует выбрать другое имя или воспользоваться кнопкой «Обновить ключ» для смены ключевой пары. Данная защита реализована намеренно: сервер не позволяет перезаписать чужой публичный ключ без криптографического подтверждения владения старым ключом, что исключает возможность выдать себя за другого пользователя

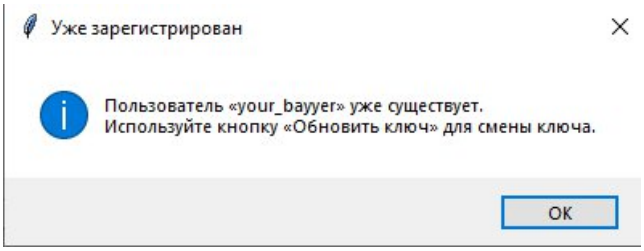


Рисунок 4.7 – Сообщение об ошибке при занятом имени

4.5 Отправка зашифрованного файла

Для отправки файла необходимо перейти на вкладку «Отправить» (рисунок 4.8).

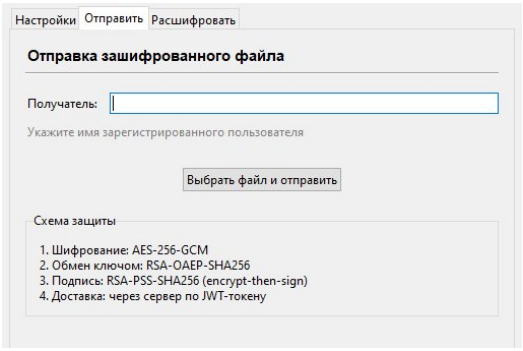


Рисунок 4.8 – Вкладка «Отправить»

Порядок отправки:

- в поле «Получатель» ввести имя пользователя, которому адресован файл. Пользователь должен быть предварительно зарегистрирован в системе;
- нажать кнопку «Выбрать файл и отправить»;
- в открывшемся системном диалоге выбрать файл для отправки (рисунок 4.9).

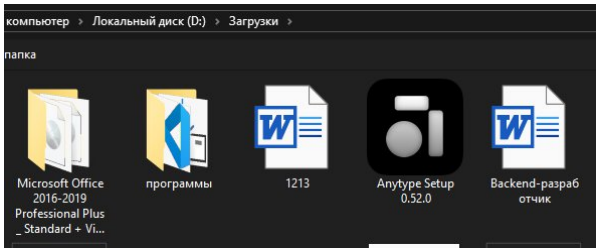


Рисунок 4.9 – Системный диалог выбора файла

После выбора файла приложение автоматически выполняет следующие операции: получает публичный ключ получателя с сервера, шифрует файл алгоритмом AES-256-GCM, шифрует сеансовый ключ открытым ключом получателя (RSA-OAEP), подписывает зашифрованные данные своим приватным ключом (RSA-PSS) и передаёт пакет на сервер.

Прогресс операции отображается в строке состояния (рисунок 4.10). По завершении появляется окно с подтверждением успешной отправки (рисунок 4.11).

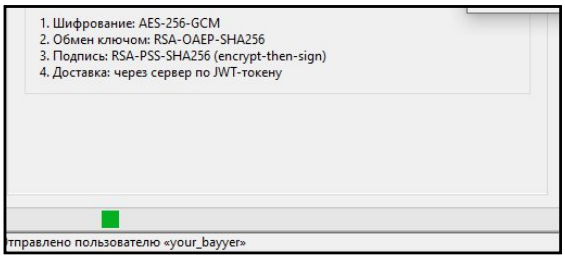


Рисунок 4.10 – Прогресс отправки в строке состояния

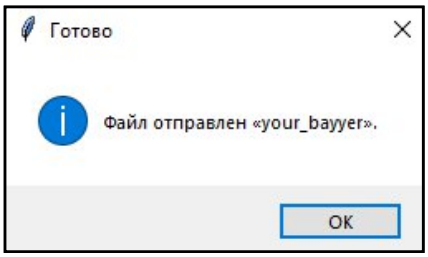


Рисунок 4.11 – Подтверждение успешной отправки

Получатель получит в Telegram сообщение от бота с прикреплённым зашифрованным файлом .bin (рисунок 4.12). Файл необходимо скачать для последующей расшифровки.

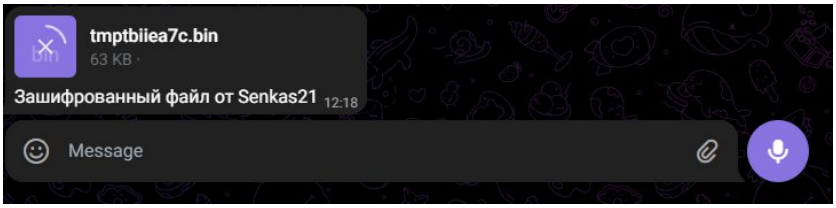


Рисунок 4.12 – Зашифрованный файл в Telegram получателя

Если JWT-токен истёк (срок действия 24 часа), приложение отображает сообщение «Токен истёк, зарегистрируйтесь заново» (рисунок 4.13). В этом случае необходимо вернуться на вкладку «Настройки» и нажать «Зарегистрироваться» повторно.

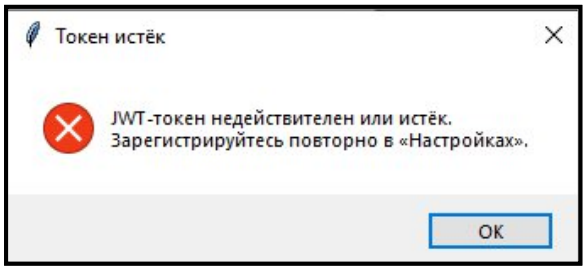


Рисунок 4.13 – Сообщение об истечении токена

4.6 Расшифровка полученного файла

Для расшифровки необходимо перейти на вкладку «Расшифровать» (рисунок 4.14).

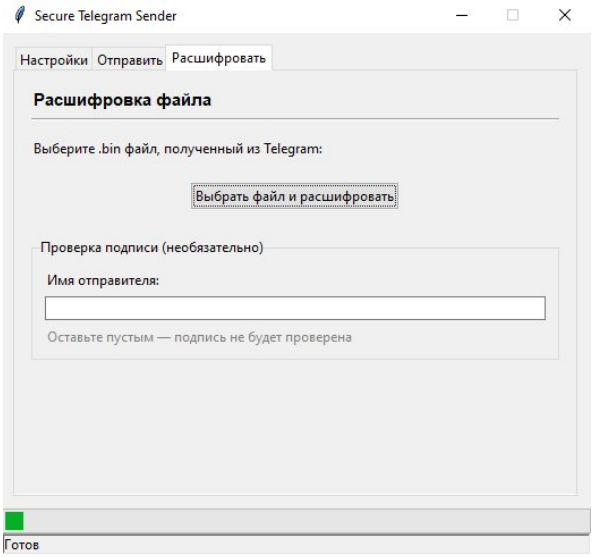


Рисунок 4.14 – Вкладка «Расшифровать»

Порядок расшифровки:

- скачать зашифрованный .bin файл из Telegram на компьютер;
- нажать кнопку «Выбрать файл и расшифровать»;
- в диалоге выбрать скачанный .bin файл;

~ при необходимости проверки подлинности отправителя — ввести его имя пользователя в поле «Имя отправителя».

Если имя отправителя указано, система автоматически запросит его публичный ключ с сервера и проверит цифровую подпись. При успешной проверке появляется статус «Подпись верна» (рисунок 4.15).

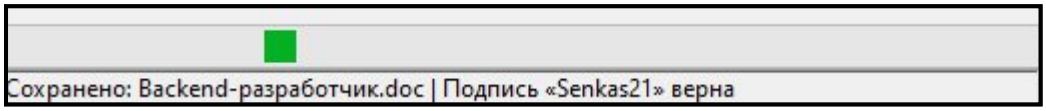


Рисунок 4.15 – Успешная верификация подписи

Если подпись не прошла проверку – расшифровка отменяется и отображается предупреждение (рисунок 4.16). Это означает, что файл мог быть изменён или отправлен другим пользователем.

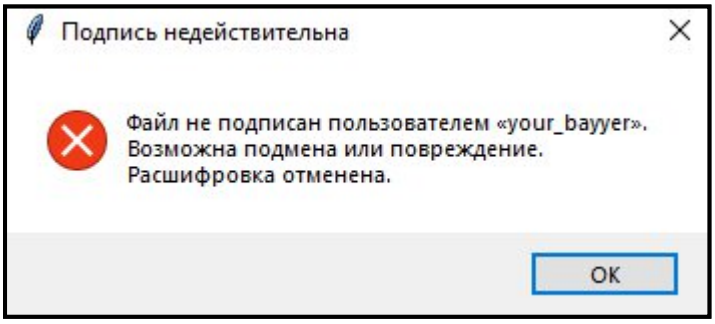


Рисунок 4.16 – Ошибка верификации подписи

файл сохраняется в той же папке что и .bin файл с оригинальным именем и расширением (рисунок 4.17) – таким же каким он был при отправке.

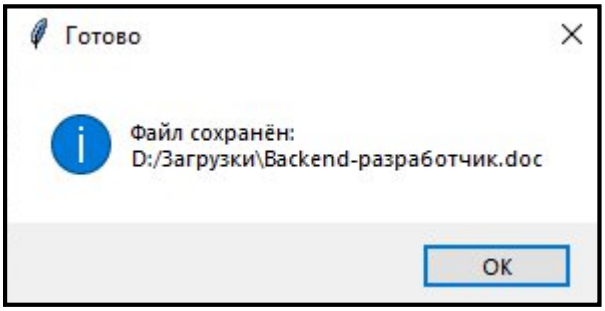


Рисунок 4.17 – Успешная расшифровка файла

4.7 Обновление ключевой пары

При необходимости смены ключей (например, если пароль был скомпрометирован) используется кнопка «Обновить ключ» (рисунок 4.18) на вкладке «Настройки». Для выполнения операции необходим текущий приватный ключ – система автоматически подписывает новый публичный ключ старым, подтверждая право на обновление.

Внимание: после смены ключей файлы, зашифрованные для старого ключа, не смогут быть расшифрованы новым ключом.

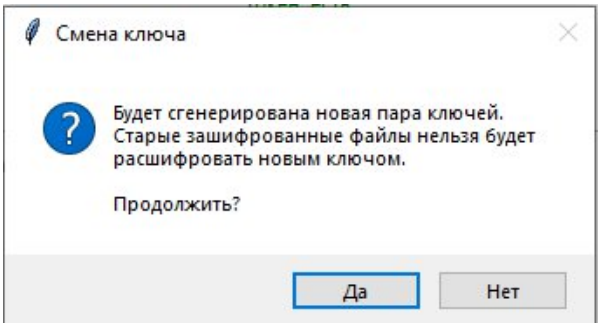


Рисунок 4.18 – Диалог подтверждения смены ключа

5 Тестирование системы

Тестирование системы проводилось по десяти сценариям, охватывающим как штатные варианты использования, так и попытки атак и нештатные ситуации.

Сценарий 1: первая регистрация. Пользователь alice задал пароль при первом запуске, ввёл имя и Chat ID. Сервер проверил Chat ID через getChat, создал запись в БД и вернул JWT-токен. Токен сохранён в auth_token.txt, статус в GUI обновился до «Токен: получен».

Сценарий 2: базовая отправка и расшифровка. Alice отправила файл размером 1 МБ пользователю bob. Файл доставлен в Telegram в зашифрованном виде. Bob расшифровал файл, указав имя отправителя alice. Подпись успешно верифицирована, содержимое совпало с оригиналом.

Сценарий 3: попытка перезаписи ключа. Злоумышленник попытался зарегистрировать имя alice без поля old_key_signature. Сервер вернул HTTP 409 с информативным сообщением. Исходный ключ alice в базе данных не изменился.

Сценарий 4: обновление ключа легитимным владельцем. Alice нажала «Обновить ключ» и подтвердила действие. Клиент подписал новый публичный ключ старым приватным и передал на сервер. Сервер верифицировал подпись, обновил запись в БД и выдал новый JWT.

Сценарий 5: обнаружение повреждения файла. В зашифрованный bundle был внесён случайный байт. При расшифровке библиотека cryptography выбросила исключение InvalidTag. Файл _decrypted не был создан, пользователю отображено сообщение об ошибке.

Сценарий 6: неверная подпись. Bundle, подписанный пользователем alice, получатель попытался верифицировать как от пользователя charlie. Функция verify_signature вернула False. Расшифровка была отменена с выводом диагностического сообщения.

Сценарий 7: ratelimiting. К эндпоинту /register было направлено 10 последовательных запросов. Первые 5 обработаны успешно, начиная с 6-го

возвращался HTTP 429. В server.log зафиксированы предупреждения. Через 60 секунд запросы возобновились в штатном режиме.

Сценарий 8: истёкший JWT-токен. Значение JWT_EXPIRE_SECONDS временно установлено в 1 секунду. После ожидания попытка отправки файла вернула HTTP 401. Клиент отобразил сообщение «Токен истёк, зарегистрируйтесь заново».

Сценарий 9: кросс-платформенная совместимость. Файл, зашифрованный на macOS, успешно расшифрован на Windows и Linux. Файл, зашифрованный на Windows, успешно расшифрован на Linux. Корректность бинарного формата пакета подтверждена.

Сценарий 10: неверный пароль ключа. При запуске введён неверный пароль три раза подряд. Приложение завершило работу с сообщением «Если вы забыли пароль – удалите private_key.pem и зарегистрируйтесь заново». Файл ключа не был повреждён.

5.1 Типичные ошибки и пути их решения

В процессе эксплуатации системы пользователи могут столкнуться с рядом типовых ситуаций, требующих диагностики. В таблице 5.1 приведены наиболее распространённые ошибки, их причины и способы устранения.

Таблица 5.1 – Типичные ошибки и решения

Сообщение об ошибке	Причина	Решение
Нет соединения с сервером	Сервер не запущен	Запустить python main.py в папке server/
Пользователь не найден	Получатель не зарегистрирован	Получатель должен запустить клиент и зарегистрироваться
Токен истёк	Прошло более 24 часов	Нажать «Зарегистрироваться» повторно
Сообщение об ошибке	Причина	Решение

Подпись недействительна	Файл изменён или указан неверный отправитель	Проверить имя отправителя или запросить файл заново
Ошибка расшифровки (InvalidTag)	Файл повреждён при передаче	Запросить повторную отправку
Неверный пароль	Введён неверный пароль ключа	Ввести правильный пароль (3 попытки)

5.2 Рекомендации к дальнейшей разработке

В части безопасности рекомендуется перевести взаимодействие клиента с сервером на HTTPS для защиты JWT-токенов и открытых ключей от перехвата. Для продуктивного развёртывания следует реализовать ротацию JWT_SECRET и добавить двухфакторную аутентификацию при смене ключа.

В части функциональности рекомендуется реализовать групповую отправку файлов нескольким получателям одновременно с шифрованием для каждого, добавить CLI-интерфейс для автоматизации, а также историю отправок в клиентском приложении.

Сервер развёрнут на облачной платформе Railway, что обеспечивает постоянную доступность без необходимости поддерживать собственную инфраструктуру. В качестве дальнейшего улучшения рекомендуется подключить постоянный том хранилища (Volume) в Railway для сохранения базы данных между передеплоями.

6 Безопасность жизнедеятельности

6.1 Правила безопасности при использовании ПК

При разработке и эксплуатации системы защищённой передачи файлов пользователь проводит значительное время за персональным компьютером. Это обусловливает необходимость соблюдения требований безопасности труда, направленных на сохранение здоровья и работоспособности.

Одним из основных вредных факторов является зрительное напряжение, возникающее при работе с графическим интерфейсом приложения: вводе криптографических параметров, отслеживании статуса отправки и расшифровки файлов. Длительное фокусирование взгляда на экране вызывает усталость глазных мышц и может привести к снижению остроты зрения. Для профилактики рекомендуется располагать монитор на расстоянии 50–70 см от глаз, устанавливать яркость и контрастность в соответствии с освещённостью помещения, а каждые 40–50 минут выполнять упражнения для глаз и делать перерывы продолжительностью не менее 10 минут.

Вторым значимым фактором является статическая нагрузка на мышцы шеи, спины и запястий при длительной работе с клавиатурой и мышью. Рабочее место следует организовать в соответствии с эргономическими нормами: кресло должно обеспечивать поддержку поясничного отдела, а высота стола - позволять держать предплечья горизонтально. Монитор располагается так, чтобы верхний край экрана находился на уровне глаз или чуть ниже.

Следует также учитывать электрическую безопасность: всё оборудование должно быть заземлено, провода не должны иметь повреждений изоляции, а розетки - признаков перегрева или нестабильного контакта. Поскольку система работает с конфиденциальными криптографическими ключами и JWT-токенами, рабочую станцию необходимо блокировать при каждом отходе от неё, а пароль от приватного ключа хранить только в памяти или в надёжном менеджере паролей.

6.2 Электробезопасность

Безопасная эксплуатация компьютерного оборудования, на котором функционирует система защищённой передачи файлов, предполагает соблюдение требований электробезопасности. Согласно ГОСТ 12.1.019-2017 «Электробезопасность. Общие требования», все электроустановки должны иметь защитное заземление, а токоведущие части - надёжную изоляцию. Перед началом работы необходимо убедиться в отсутствии видимых повреждений кабелей питания и корпусов устройств. Не допускается эксплуатация оборудования с нарушенной изоляцией, а также подключение к одной розетке нескольких мощных потребителей через удлинители без защиты от перегрузки.

При обнаружении запаха гари, искрения или нестандартного нагрева оборудование необходимо немедленно отключить от сети и сообщить ответственному лицу. Серверная часть системы, функционирующая в непрерывном режиме на платформе Railway, размещена в профессиональном дата-центре, где соблюдение норм электробезопасности обеспечивается провайдером в соответствии с международными стандартами надёжности.

Заключение

В ходе выполнения данной выпускной квалификационной работы разработана кросс-платформенная система защищённой передачи файлов через Telegram, устраняющая ключевые уязвимости типовых решений.

Реализован криптографический модуль с AES-256-GCM, RSA-OAEP-SHA256, RSA-PSS-SHA256 и защитой ключа паролем PKCS8+PBKDF2. Разработано серверное приложение на FastAPI с JWT-аутентификацией, защитой от перезаписи ключей, ratelimiting, верификацией Chat ID и двухуровневым логированием. Разработан кросс-платформенный GUI на tkinter для macOS, Windows и Linux.

Тестирование по десяти сценариям подтвердило корректность всех реализованных механизмов: шифрования, верификации подписи, ratelimiting, JWT-аутентификации, защиты от перезаписи ключа и кросс-платформенной совместимости.

Разработанная система представляет практическую ценность как инструмент защищённого обмена файлами и демонстрирует применение современных криптографических стандартов в реальном приложении.Система успешно развёрнута на облачной платформе Railway и протестирована в условиях реального использования с участием нескольких пользователей.

Список использованных источников

1. Telegram Bot API Documentation. URL: https://core.telegram.org/bots/api (дата обращения: 02.05.2026).

2. Python-telegram-bot Documentation. URL: https://python-telegram-bot.readthedocs.io (дата обращения: 04.05.2026).

3. httpx Documentation. URL: https://www.python-httpx.org (дата обращения: 04.05.2026).

4. NIST FIPS 197. Advanced Encryption Standard (AES). – NIST, 2001. – 51 с.

5. NIST SP 800-38D. Recommendation for Block Cipher Modes of Operation: GCM. – NIST, 2007. – 39 с.

6. RFC 5958 – PKCS #8. URL: https://www.rfc-editor.org/rfc/rfc5958 (дата обращения: 06.05.2026).

7. Документация библиотеки cryptography. URL: https://cryptography.io/en/latest/ (дата обращения: 06.05.2026).

8. Ferguson N., Schneier B., Kohno T. Cryptography Engineering. – Wiley, 2010. – 384 с.

9. FastAPI Documentation. URL: https://fastapi.tiangolo.com (дата обращения: 06.05.2026).

10. Pydantic v2 Documentation. URL: https://docs.pydantic.dev (дата обращения: 06.05.2026).

11. PyJWT Documentation. URL: https://pyjwt.readthedocs.io (дата обращения: 07.05.2026).

12. SQLite WAL Mode. URL: https://www.sqlite.org/wal.html (дата обращения: 07.05.2026).

13. PyCharm Documentation. URL: https://www.jetbrains.com/pycharm/documentation/ (дата обращения: 08.05.2026).

14. RFC 5958 – PKCS #8. URL: https://www.rfc-editor.org/rfc/rfc5958 (дата обращения: 08.05.2026).

15. RFC 7519 – JSON Web Token (JWT). URL: https://www.rfc-editor.org/ (дата обращения: 08.05.2026).

16. Railway Documentation. URL: https://docs.railway.app (дата обращения: 10.05.2026).

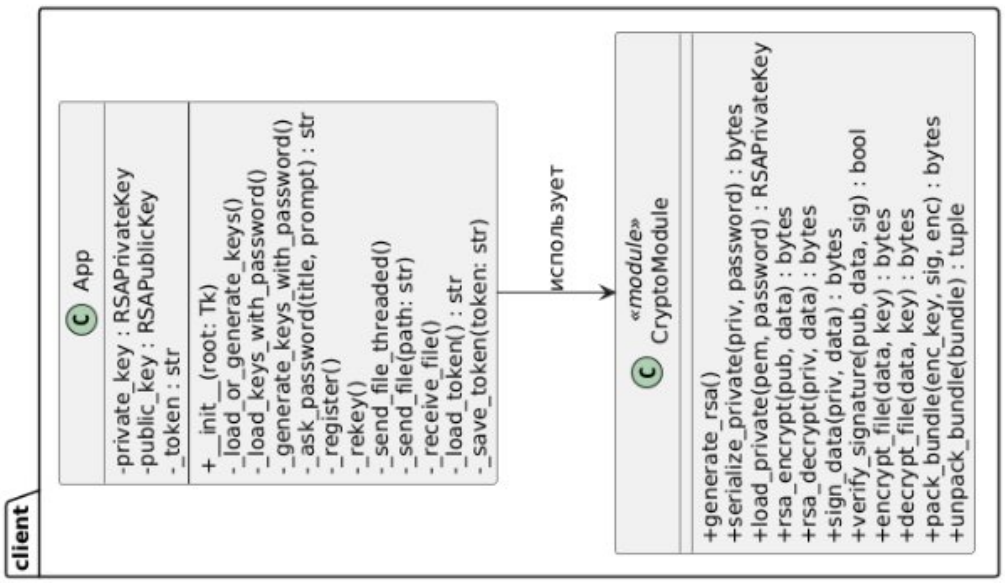
Приложение А

(обязательное)

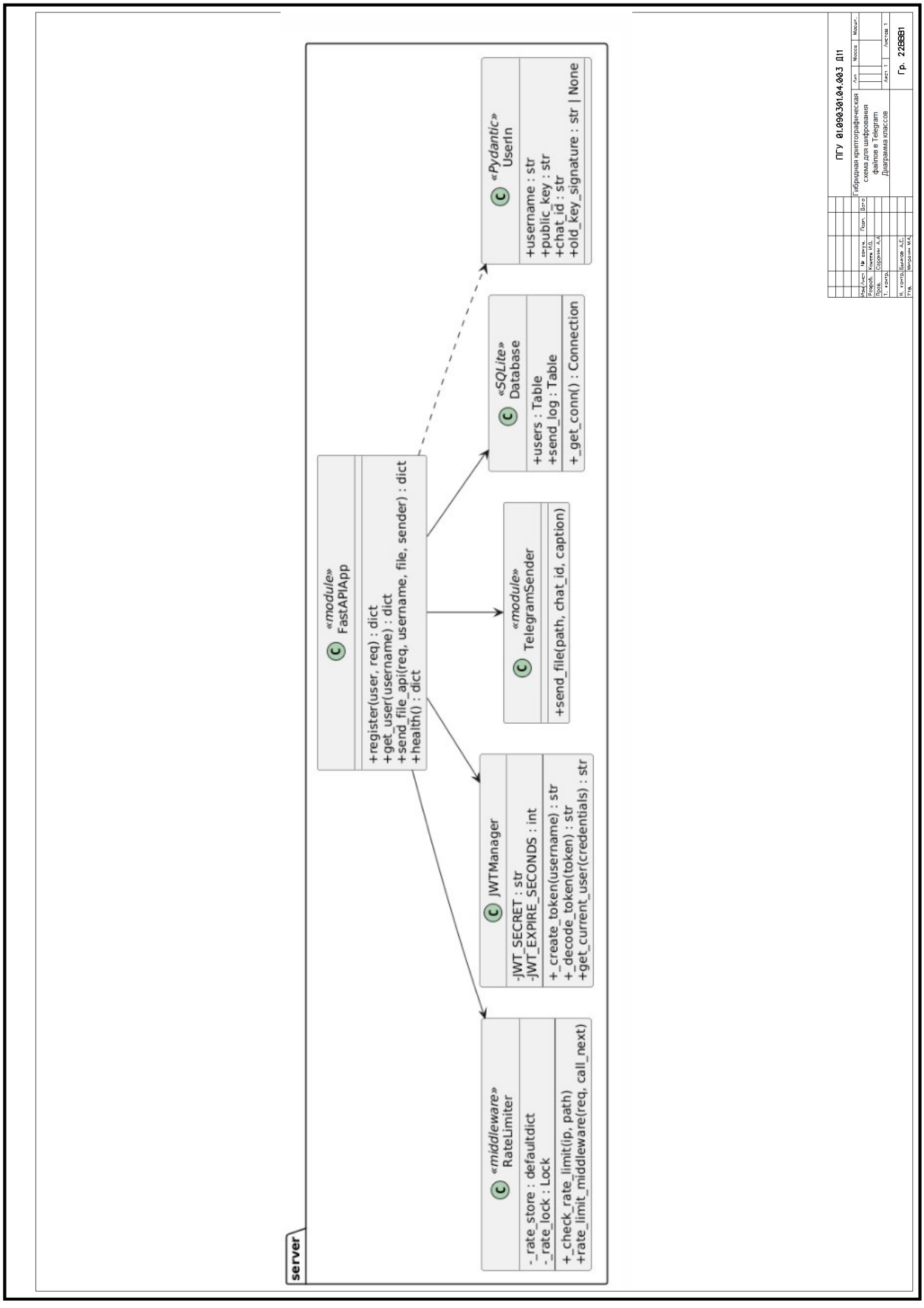
Гибридная криптографическая схема для шифрования файлов в Telegram

UML-диаграммы

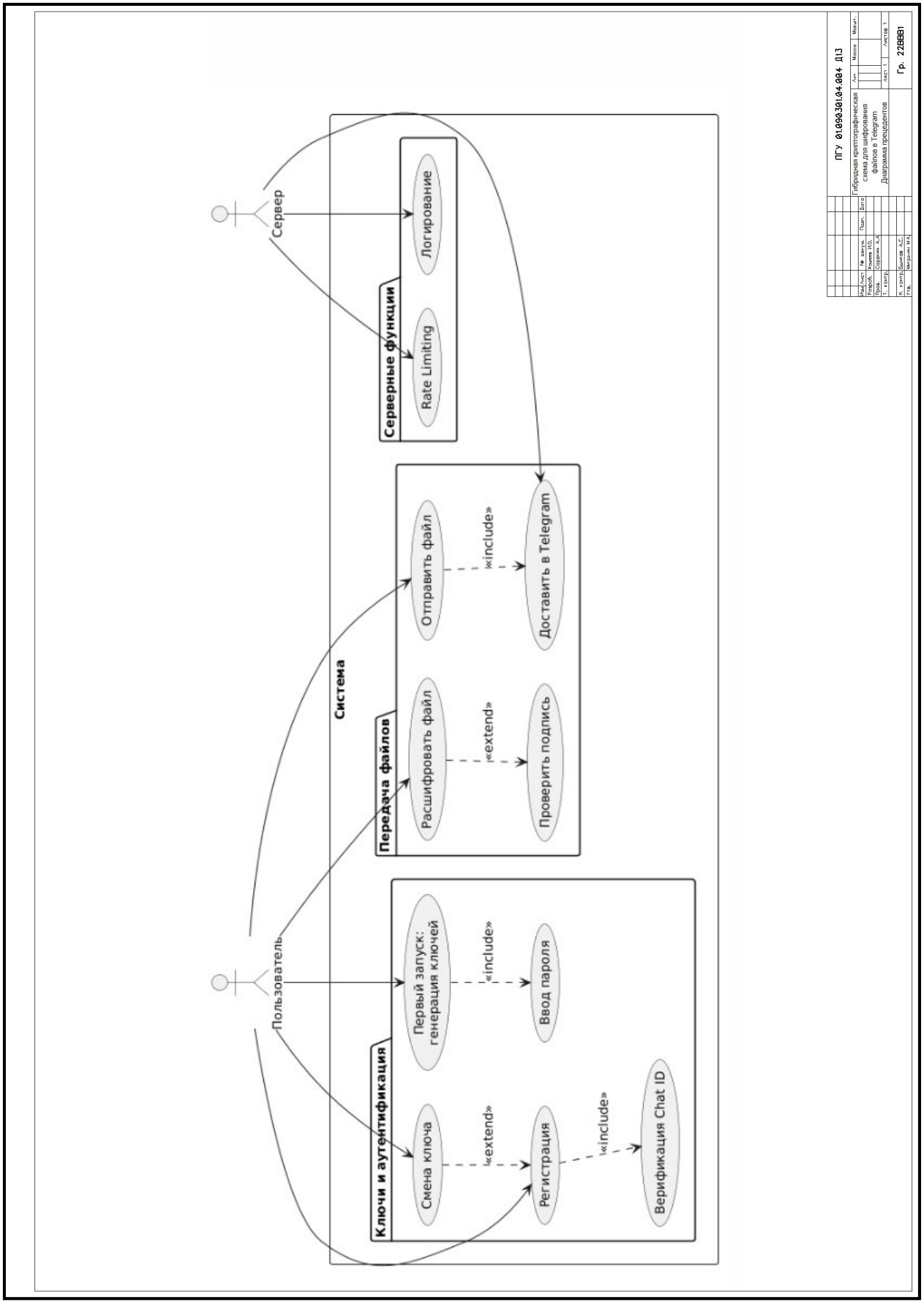
UML-диаграмма классов: клиентская часть



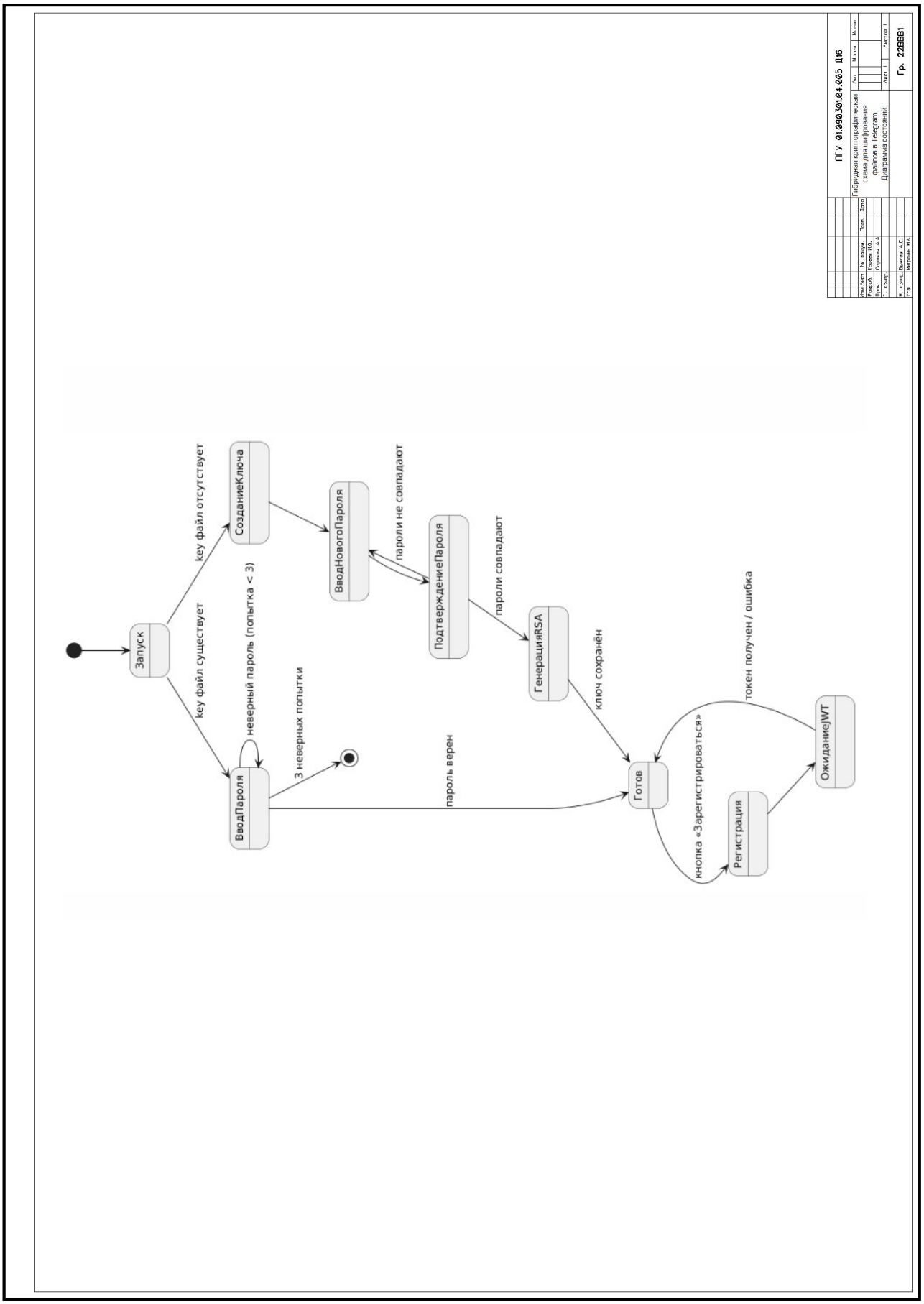
UML-диаграмма классов: серверная часть



UML-диаграмма прецендентов



UML-диаграмма состояний (запуск)



Приложение Б

(обязательное)

Гибридная криптографическая схема для шифрования файлов в Telegram

Листинг фрагментов программы

Файл client/app.py

```
"""
app.py — GUI-клиент Secure Telegram Sender v2.
Кросс-платформенный: macOS, Windows, Linux.

"""

import base64
import os
import platform
import tempfile
import threading
import tkinter as tk
from tkinter import filedialog, messagebox, simpledialog, ttk

import requests

from crypto import (
decrypt_file,
encrypt_file,
generate_rsa,
generate_session_key,
load_private,
load_public,
pack_bundle,
rsa_decrypt,
rsa_encrypt,
serialize_private,
serialize_public,
sign_data,
unpack_bundle,
verify_signature,
)

SERVER = "https://secure-telegram-sender-production.up.railway.app"
PRIVATE_KEY_FILE = "private_key.pem"
TOKEN_FILE = "auth_token.txt"


# ── DPI
──────────────────────────────────────────────────────────────────
──


def _configure_dpi(root: tk.Tk) -> None:
    system = platform.system()
    if system == "Darwin":
        try:
root.tk.call("tk", "scaling", 2.0)
        except Exception:
            pass
elif system == "Windows":
```

```
        try:
            import ctypes
ctypes.windll.shcore.SetProcessDpiAwareness(1)
        except Exception:
            pass
        try:
root.tk.call("tk", "scaling", root.winfo_fpixels("1i") / 72)
        except Exception:
            pass


# ── Приложение
─────────────────────────────────────────────────────────────────


class App:
    def __init__(self, root: tk.Tk) -> None:
self.root = root
self.root.title("Secure Telegram Sender")
self.root.geometry("520x460")
self.root.resizable(False, False)

        _configure_dpi(root)

self._token: str = self._load_token()
self.private_key, self.public_key = self._load_or_generate_keys()
self._build_ui()

    # ── Токен
─────────────────────────────────────────────────────────────────


    def _load_token(self) -> str:
        if os.path.exists(TOKEN_FILE):
            with open(TOKEN_FILE, "r") as f:
                return f.read().strip()
        return ""

    def _save_token(self, token: str) -> None:
self._token = token
        with open(TOKEN_FILE, "w") as f:
f.write(token)

    # ── Ключиспаролем
─────────────────────────────────────────────────────


    def _ask_password(self, title: str, prompt: str) -> str | None:
"""Диалог ввода пароля. Возвращает строку или None если отменено."""
pwd = simpledialog.askstring(title, prompt, show="*",
parent=self.root)
        return pwd

    def _load_or_generate_keys(self):
        if os.path.exists(PRIVATE_KEY_FILE):
```

```
            return self._load_keys_with_password()
        return self._generate_keys_with_password()

    def _load_keys_with_password(self):
"""Загружает ключи с паролем. При неверном пароле предлагает
повторить."""
for attempt in range(3):
pwd = self._ask_password(
"Пароль ключа",
f"Введите пароль для расшифровки приватного ключа"
f"{' (попытка ' + str(attempt + 1) + '/3)' if attempt > 0 else
''}:",
)
ifpwdisNone:
                # Пользователь нажал Отмена
messagebox.showerror(
                    "Ошибка", "Без пароля приложение не может
работать."
)
self.root.destroy()
                raise SystemExit

            try:
                with open(PRIVATE_KEY_FILE, "rb") as f:
priv = load_private(f.read(), password=pwd.encode() if pwd else
None)
                return priv, priv.public_key()
            except Exception:
                if attempt == 2:
messagebox.showerror(
"Ошибка",
"Неверный пароль. Приложение закрывается.\n"
                        "Если вы забыли пароль — удалите
private_key.pem "
                        "и зарегистрируйтесь заново.",
                    )
self.root.destroy()
raiseSystemExit
continue

    def _generate_keys_with_password(self):
"""Генерирует новую пару ключей, запрашивает пароль для защиты."""
messagebox.showinfo(
            "Первый запуск",
            "Приватный ключ не найден.\n"
            "Будет сгенерирована новая пара RSA-2048 ключей.\n\n"
            "Задайте пароль для защиты приватного ключа на диске.\n"
            "Запомните его — без пароля ключ не загрузится.",
)
pwd = self._ask_password("Новый пароль", "Задайте пароль для
приватного ключа:")
if pwd is None:
```

```
messagebox.showerror("Ошибка", "Пароль обязателен.")
self.root.destroy()
            raise SystemExit

pwd2 = self._ask_password("Подтверждение", "Повторите пароль:")
ifpwd != pwd2:
messagebox.showerror("Ошибка", "Пароли не совпадают. Перезапустите
приложение.")
self.root.destroy()
            raise SystemExit

priv, pub = generate_rsa()
        with open(PRIVATE_KEY_FILE, "wb") as f:
f.write(serialize_private(priv, password=pwd.encode() if pwd else
None))
        return priv, pub

    # ── UI
─────────────────────────────────────────────────────────────────

    def _build_ui(self) -> None:
nb = ttk.Notebook(self.root)
nb.pack(fill="both", expand=True, padx=10, pady=10)

tab_s = ttk.Frame(nb)
nb.add(tab_s, text="Настройки")
self._build_settings_tab(tab_s)

tab_send = ttk.Frame(nb)
nb.add(tab_send, text="Отправить")
self._build_send_tab(tab_send)

tab_recv = ttk.Frame(nb)
nb.add(tab_recv, text="Расшифровать")
self._build_recv_tab(tab_recv)

self.status_var = tk.StringVar(value="Готов")
ttk.Label(
self.root, textvariable=self.status_var, relief="sunken", anchor="w"
).pack(fill="x", side="bottom")

self.progress = ttk.Progressbar(self.root, mode="indeterminate")
self.progress.pack(fill="x", side="bottom")

    def _build_settings_tab(self, parent: ttk.Frame) -> None:
        pad = {"padx": 15, "pady": 8}

ttk.Label(parent, text="Регистрация",
                  font=("", 11, "bold")).grid(
            row=0, column=0, columnspan=2, sticky="w", padx=15,
pady=(15, 5))
ttk.Separator(parent, orient="horizontal").grid(
```

```
            row=1, column=0, columnspan=2, sticky="ew", padx=15,
pady=(0, 10))

ttk.Label(parent, text="Имяпользователя:").grid(
            row=2, column=0, sticky="w", **pad)
self.username_entry = ttk.Entry(parent, width=30)
self.username_entry.grid(row=2, column=1, sticky="ew", **pad)

ttk.Label(parent, text="Chat ID:").grid(row=3, column=0, sticky="w",
**pad)
self.chat_id_entry = ttk.Entry(parent, width=30)
self.chat_id_entry.grid(row=3, column=1, sticky="ew", **pad)

ttk.Label(
            parent,
            text="Узнайте Chat ID у @userinfobotв Telegram",
            foreground="gray",
).grid(row=4, column=1, sticky="w", padx=15, pady=(0, 8))

        # Статустокена
self.token_status_var = tk.StringVar(
value="Токен: есть" ifself._tokenelse "Токен: нет (нужна
регистрация)"
)
ttk.Label(parent, textvariable=self.token_status_var,
                  foreground="green" if self._token else
"red").grid(
            row=5, column=0, columnspan=2, padx=15, pady=(0, 8))

ttk.Button(
            parent, text="Зарегистрироваться",
command=self._register
).grid(row=6, column=0, columnspan=2, pady=15)

        # Сменаключа
ttk.Separator(parent, orient="horizontal").grid(
            row=7, column=0, columnspan=2, sticky="ew", padx=15,
pady=(0, 8))
ttk.Label(parent, text="Сменить пару ключей (с подтверждением):",
foreground="gray").grid(
            row=8, column=0, columnspan=2, sticky="w", padx=15)
ttk.Button(
            parent, text="Обновитьключ", command=self._rekey
).grid(row=9, column=0, columnspan=2, pady=8)

parent.columnconfigure(1, weight=1)

    def _build_send_tab(self, parent: ttk.Frame) -> None:
ttk.Label(parent, text="Отправка зашифрованного файла",
font=("", 11, "bold")).pack(anchor="w", padx=15, pady=(15, 5))
ttk.Separator(parent, orient="horizontal").pack(
            fill="x", padx=15, pady=(0, 10))
```

```
        frame = ttk.Frame(parent)
frame.pack(fill="x", padx=15, pady=8)
ttk.Label(frame, text="Получатель:").pack(side="left", padx=(0, 10))
self.target_entry = ttk.Entry(frame, width=28)
self.target_entry.pack(side="left", fill="x", expand=True)

ttk.Label(
parent,
text="Укажите имя зарегистрированного пользователя",
foreground="gray",
).pack(anchor="w", padx=15, pady=(0, 15))

ttk.Button(
parent, text="Выбрать файл и отправить",
command=self._send_file_threaded,
).pack(pady=10)

        info = ttk.LabelFrame(parent, text="Схемазащиты",
padding=10)
info.pack(fill="x", padx=15, pady=10)
ttk.Label(
            info,
            text="1. Шифрование: AES-256-GCM\n"
                 "2. Обменключом: RSA-OAEP-SHA256\n"
                 "3. Подпись: RSA-PSS-SHA256 (encrypt-then-sign)\n"
"4. Доставка: через сервер по JWT-токену",
justify="left",
).pack(anchor="w")

    def _build_recv_tab(self, parent: ttk.Frame) -> None:
ttk.Label(parent, text="Расшифровкафайла",
                  font=("", 11, "bold")).pack(anchor="w", padx=15,
pady=(15, 5))
ttk.Separator(parent, orient="horizontal").pack(
            fill="x", padx=15, pady=(0, 10))

ttk.Label(
parent, text="Выберите .bin файл, полученный из Telegram:"
).pack(anchor="w", padx=15, pady=(5, 10))

ttk.Button(
            parent, text="Выбратьфайлирасшифровать",
            command=self._receive_file,
).pack(pady=10)

sig_frame = ttk.LabelFrame(
            parent, text="Проверкаподписи (необязательно)",
padding=10)
sig_frame.pack(fill="x", padx=15, pady=15)
ttk.Label(sig_frame, text="Имяотправителя:").pack(anchor="w",
pady=(0, 5))
```

```
self.sender_entry = ttk.Entry(sig_frame, width=30)
self.sender_entry.pack(fill="x", pady=(0, 5))
ttk.Label(
sig_frame,
text="Оставьте пустым — подпись не будет проверена",
foreground="gray",
        ).pack(anchor="w")

    # ── Статус
─────────────────────────────────────────────────────────────

def _set_status(self, text: str) -> None:
self.status_var.set(text)
self.root.update_idletasks()

    def _start_progress(self) -> None:
self.progress.start(12)

    def _stop_progress(self) -> None:
self.progress.stop()

    # ── Регистрация
──────────────────────────────────────────────────────

    def _register(self) -> None:
        username = self.username_entry.get().strip()
chat_id = self.chat_id_entry.get().strip()

        if not username or not chat_id:
messagebox.showerror("Ошибка", "Заполните имя пользователя и
ChatID")
return
        if not chat_id.lstrip("-").isdigit():
messagebox.showerror("Ошибка", "Chat ID долженбытьчислом")
            return

        try:
            r = requests.post(
                SERVER + "/register",
                json={
                    "username": username,
                    "public_key": serialize_public(self.public_key),
                    "chat_id": chat_id,
                },
                timeout=10,
)
ifr.status_code == 409:
                # Пользователь уже существует — предлагаем
обновление с подписью
messagebox.showinfo(
                    "Уже зарегистрирован",
f"Пользователь «{username}» уже существует.\n"
```

```
                    "Используйте кнопку «Обновить ключ» для смены
ключа.",
)
                return
r.raise_for_status()
            data = r.json()
self._save_token(data["token"])
self.token_status_var.set("Токен: получен")
self._set_status(f"Зарегистрированкак «{username}»")
messagebox.showinfo(
                "Готово",
                f"Пользователь «{username}» зарегистрирован.\n"
f"JWT-токен сохранён (действует 24 часа).",
)
        except requests.ConnectionError:
messagebox.showerror(
"Ошибка",
                "Нет соединения с сервером.\n"
                "Убедитесь, что сервер запущен на
http://127.0.0.1:8000",
)
        except requests.HTTPError as e:
messagebox.showerror("Ошибкасервера",
                                    f"{e.response.status_code}:
{e.response.text}")
        except Exception as e:
messagebox.showerror("Ошибка", str(e))

def _rekey(self) ->None:
        """Обновление ключа: подписываем новый публичный ключ старым
приватным."""
username = self.username_entry.get().strip()
        if not username:
messagebox.showerror("Ошибка", "Введите имя пользователя")
return

        if not messagebox.askyesno(
"Смена ключа",
"Будет сгенерирована новая пара ключей.\n"
            "Старые зашифрованные файлы нельзя будет расшифровать
новым ключом.\n\n"
            "Продолжить?",
        ):
return

        # Сохраняем старый приватный ключ для подписи
old_private_key = self.private_key

        # Генерируемновуюпару
new_priv, new_pub = generate_rsa()

# Подписываем новый публичный ключ старым приватным
```

```
new_pub_bytes = serialize_public(new_pub).encode()
        signature = sign_data(old_private_key, new_pub_bytes)
        sig_b64 = base64.b64encode(signature).decode()

# Запрашиваем новый пароль
messagebox.showinfo(
            "Новый пароль",
            "Задайте пароль для нового приватного ключа.",
)
pwd = self._ask_password("Новый пароль", "Пароль для нового ключа:")
if pwd is None:
            return
pwd2 = self._ask_password("Подтверждение", "Повторите пароль:")
ifpwd != pwd2:
messagebox.showerror("Ошибка", "Пароли не совпадают")
return

chat_id = self.chat_id_entry.get().strip() or "0"

        try:
            r = requests.post(
                SERVER + "/register",
                json={
                    "username": username,
                    "public_key": serialize_public(new_pub),
                    "chat_id": chat_id,
                    "old_key_signature": sig_b64,
                },
                timeout=10,
            )
r.raise_for_status()
data = r.json()

            # Сохраняем новый ключ
with open(PRIVATE_KEY_FILE, "wb") as f:
f.write(serialize_private(new_priv,
                         password=pwd.encode() if pwd else None))
self.private_key = new_priv
self.public_key = new_pub

self._save_token(data["token"])
self.token_status_var.set("Токен: получен (новыйключ)")
self._set_status("Ключобновлён")
messagebox.showinfo("Готово", "Ключ успешно обновлён на сервере.")
except requests.HTTPError as e:
messagebox.showerror("Ошибка", f"{e.response.status_code}:
{e.response.text}")
        except Exception as e:
messagebox.showerror("Ошибка", str(e))

    # ── Отправка
```

```
    def _send_file_threaded(self) -> None:
        if not self._token:
messagebox.showerror(
"Нет токена",
                "Вы не зарегистрированы или токен истёк.\n"
                "Перейдите в «Настройки» и зарегистрируйтесь.",
)
            return
        path =
filedialog.askopenfilename(title="Выберитефайлдляотправки")
        if not path:
            return
threading.Thread(target=self._send_file, args=(path,),
daemon=True).start()

    def _send_file(self, path: str) -> None:
        target = self.target_entry.get().strip()
        if not target:
messagebox.showerror("Ошибка", "Укажите получателя")
return

self._start_progress()
tmp_path = None
try:
self._set_status("Получение публичного ключа...")
r = requests.get(SERVER + "/user/" + target, timeout=5)
r.raise_for_status()
            data = r.json()

            pub = load_public(data["public_key"])

self._set_status("Шифрование и подпись...")
session_key = generate_session_key()
            with open(path, "rb") as f:
                raw = f.read()

            encrypted = encrypt_file(raw, session_key)
enc_key = rsa_encrypt(pub, session_key)
            signature = sign_data(self.private_key, encrypted)
            bundle = pack_bundle(enc_key, signature, encrypted,
os.path.basename(path))

            with tempfile.NamedTemporaryFile(suffix=".bin",
delete=False) as tmp:
tmp.write(bundle)
tmp_path = tmp.name

self._set_status("Передача на сервер...")
with open(tmp_path, "rb") as f:
                resp = requests.post(
                    SERVER + "/send",
```

```
                    params={"username": target},
                    files={"file": (os.path.basename(path) + ".bin",
f,
                                    "application/octet-stream")},
                    headers={"Authorization": f"Bearer
{self._token}"},
                    timeout=30,
                )

            if resp.status_code == 401:
messagebox.showerror(
"Токен истёк",
                    "JWT-токен недействителен или истёк.\n"
"Зарегистрируйтесь повторно в «Настройках».",
                )
self._set_status("Ошибка: токенистёк")
return

resp.raise_for_status()
self._set_status(f"Отправленопользователю «{target}»")
messagebox.showinfo("Готово", f"Файлотправлен «{target}».")

except requests.ConnectionError:
messagebox.showerror("Ошибка", "Нетсоединенияссервером")
self._set_status("Ошибкаотправки")
        except requests.HTTPError as e:
messagebox.showerror("Ошибкасервера",
                                 f"{e.response.status_code}:
{e.response.text}")
self._set_status("Ошибкаотправки")
        except Exception as e:
messagebox.showerror("Ошибка", str(e))
self._set_status("Ошибкаотправки")
        finally:
self._stop_progress()
            if tmp_path and os.path.exists(tmp_path):
                try:
os.unlink(tmp_path)
                except Exception:
                    pass

    # ── Расшифровка
─────────────────────────────────────────────────────

    def _receive_file(self) -> None:
        path = filedialog.askopenfilename(
            title="Выберитезашифрованныйфайл",
            filetypes=[("Binary", "*.bin"), ("Всефайлы", "*.*")],
        )
        if not path:
            return
```

```
self._start_progress()
        try:
            with open(path, "rb") as f:
                bundle = f.read()

enc_key, signature, encrypted, original_filename =
unpack_bundle(bundle)

            sender = self.sender_entry.get().strip()
            if sender:
self._set_status("Проверкаподписи...")
                try:
                    r = requests.get(SERVER + "/user/" + sender,
timeout=5)
r.raise_for_status()
sender_pub = load_public(r.json()["public_key"])
                except Exception:
messagebox.showerror(
                        "Ошибка",
f"Не удалось получить ключ пользователя «{sender}»",
)
                    return

                if not verify_signature(sender_pub, encrypted,
signature):
messagebox.showerror(
                        "Подпись недействительна",
f"Файл не подписан пользователем «{sender}».\n"
                        "Возможна подмена или повреждение.\n"
"Расшифровка отменена.",
                    )
self._set_status("Подпись не прошла проверку")
return
self._set_status("Подпись верна — расшифровка...")
else:
self._set_status("Расшифровка...")

session_key = rsa_decrypt(self.private_key, enc_key)
            decrypted = decrypt_file(encrypted, session_key)

            if original_filename:
out_path = os.path.join(os.path.dirname(path), original_filename)
            else:
out_path = os.path.splitext(path)[0] + "_decrypted"
            with open(out_path, "wb") as f:
f.write(decrypted)

            status = "Сохранено: " + os.path.basename(out_path)
            if sender:
                status += f" | Подпись «{sender}» верна"
self._set_status(status)
messagebox.showinfo("Готово", f"Файлсохранён:\n{out_path}")
```

```
        except requests.ConnectionError:
messagebox.showerror("Ошибка", "Нетсоединенияссервером")
        except Exception as e:
messagebox.showerror("Ошибкарасшифровки", str(e))
self._set_status("Ошибкарасшифровки")
        finally:
self._stop_progress()


# ── Запуск
──────────────────────────────────────────────────────────────────────


if __name__ == "__main__":
    root = tk.Tk()
    app = App(root)
root.mainloop()
```

Файл crypto.py (Используется на клиенте и на сервере)

```
"""
crypto.py — криптографические примитивы.
- RSA-OAEP-SHA256 для обмена ключами
- RSA-PSS-SHA256 для подписи файлов
- AES-256-GCM для шифрования файлов
- Шифрование приватного ключа паролем (BestAvailableEncryption)
"""

import os
import struct

from cryptography.exceptions import InvalidSignature
from cryptography.hazmat.primitives import hashes, serialization
from cryptography.hazmat.primitives.asymmetric import padding, rsa
from cryptography.hazmat.primitives.ciphers.aead import AESGCM


# ── RSA ──────────────────────────────────────────────────────────────
──


def generate_rsa():
    private = rsa.generate_private_key(public_exponent=65537,
key_size=2048)
    return private, private.public_key()


def serialize_public(pub) -> str:
    return pub.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo,
).decode()


def serialize_private(priv, password: bytes | None = None) -> bytes:
    """
Сериализуетприватныйключ.
Если password задан — шифрует с BestAvailableEncryption (AES-256-CBC
+ PBKDF2).
Если password=None — сохраняет без шифрования (не рекомендуется).
"""
    encryption = (
serialization.BestAvailableEncryption(password)
        if password
        else serialization.NoEncryption()
    )
    return priv.private_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PrivateFormat.PKCS8,
encryption_algorithm=encryption,
```

```
    )


def load_public(pem: str):
    return serialization.load_pem_public_key(pem.encode())


def load_private(pem: bytes, password: bytes | None = None):
"""
    Загружает приватный ключ.
password — байты пароля или None для незашифрованного ключа.
    Выбрасывает ValueError при неверном пароле.
"""
    return serialization.load_pem_private_key(pem,
password=password)


def rsa_encrypt(pub, data: bytes) -> bytes:
    return pub.encrypt(
        data,
padding.OAEP(
mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None,
        ),
    )


def rsa_decrypt(priv, data: bytes) -> bytes:
    return priv.decrypt(
        data,
padding.OAEP(
mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None,
        ),
    )


# ── RSA-PSS ──────────────────────────────────────────────────────────

_PSS_PADDING = padding.PSS(
mgf=padding.MGF1(hashes.SHA256()),
salt_length=padding.PSS.MAX_LENGTH,
)


def sign_data(priv, data: bytes) -> bytes:
    return priv.sign(data, _PSS_PADDING, hashes.SHA256())
```

```
def verify_signature(pub, data: bytes, signature: bytes) -> bool:
    try:
pub.verify(signature, data, _PSS_PADDING, hashes.SHA256())
        return True
    except InvalidSignature:
        return False


# ── AES-256-GCM
──────────────────────────────────────────────────────────────


def generate_session_key() -> bytes:
    return os.urandom(32)


def encrypt_file(data: bytes, key: bytes) -> bytes:
    nonce = os.urandom(12)
ct = AESGCM(key).encrypt(nonce, data, None)
    return nonce + ct


def decrypt_file(data: bytes, key: bytes) -> bytes:
    return AESGCM(key).decrypt(data[:12], data[12:], None)


# ── Бинарныйпакет
──────────────────────────────────────────────────────────────
def pack_bundle(enc_key: bytes, signature: bytes, encrypted: bytes,
filename: str = "") -> bytes:
fname_bytes = filename.encode("utf-8")
    return (
struct.pack(">I", len(enc_key)) + enc_key
        + struct.pack(">I", len(signature)) + signature
        + struct.pack(">I", len(fname_bytes)) + fname_bytes
        + encrypted
    )


def unpack_bundle(bundle: bytes):
    offset = 0

enc_key_len = int.from_bytes(bundle[offset:offset + 4], 'big')
    offset += 4
enc_key = bundle[offset:offset + enc_key_len]
    offset += enc_key_len

sig_len = int.from_bytes(bundle[offset:offset + 4], 'big')
    offset += 4
    signature = bundle[offset:offset + sig_len]
    offset += sig_len

fname_len = int.from_bytes(bundle[offset:offset + 4], 'big')
```

```
    offset += 4
    filename = bundle[offset:offset + fname_len].decode("utf-8")
    offset += fname_len

    encrypted = bundle[offset:]

    return enc_key, signature, encrypted, filename
```

Файл server/telegram_sender.py

```
import os
from telegram import Bot


async def send_file(file_path: str, chat_id: str, caption: str =
""):
    token = os.getenv("TELEGRAM_BOT_TOKEN")

    if not token:
        raise ValueError("TELEGRAM_BOT_TOKEN незадан")

    async with Bot(token=token) as bot:
        with open(file_path, "rb") as f:
            await bot.send_document(
chat_id=chat_id,
                document=f,
                caption=caption or os.path.basename(file_path),
            )
```

Файл server/main.py

```
"""
server/main.py — FastAPI-сервер Secure Telegram Sender.

Улучшения v2:
  - JWT-аутентификация: /register выдаёт токен, /send требует его
  - Защита /register от перезаписи чужого ключа через подпись старым
ключом
  - Ratelimiting (slidingwindow, без сторонних зависимостей)
  - Логирование всех действий в server.log
  - Проверка ChatID через TelegramBotAPI при регистрации
"""

import base64
import logging
import os
import re
import sqlite3
import tempfile
import threading
import time
from collections import defaultdict
from datetime import datetime, timezone

import httpx
import jwt
from fastapi import FastAPI, HTTPException, UploadFile, File,
Depends, Request
from fastapi.responses import JSONResponse
from fastapi.security import HTTPBearer,
HTTPAuthorizationCredentials
from pydantic import BaseModel, field_validator
from telegram_sender import send_file
from crypto import load_public, verify_signature
if os.path.exists(".env"):
    from dotenv import load_dotenv
load_dotenv()

# ── Логирование
─────────────────────────────────────────────────────

logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s  %(levelname)-8s  %(message)s",
    handlers=[
logging.FileHandler("server.log", encoding="utf-8"),
logging.StreamHandler(),
    ],
)
log = logging.getLogger("secure_sender")
```

```
print("TOKEN CHECK:", "OK" if os.getenv("TELEGRAM_BOT_TOKEN") else
"NOT FOUND")
print("ALL ENV:", {k: v for k, v in os.environ.items() if 'TOKEN' in
k or 'JWT' in k})
# ── Конфигурация
──────────────────────────────────────────────────────────

JWT_SECRET = os.getenv("JWT_SECRET", "change-me-in-production-use-
long-random-string")
JWT_ALGORITHM = "HS256"
JWT_EXPIRE_SECONDS = 86400  # 24 часа


# ── Приложение
──────────────────────────────────────────────────────────

app = FastAPI(title="Secure Sender Server", version="2.0")
security = HTTPBearer()

# ── Rate Limiting (sliding window)
──────────────────────────────────────

_rate_lock = threading.Lock()
_rate_store: dict = defaultdict(list)

RATE_LIMITS = {
    "/register": (5, 60),
    "/send":     (20, 60),
    "/user":     (60, 60),
}

def _check_rate_limit(ip: str, path: str) -> None:
    limit, window = None, None
    for prefix, (lim, win) in RATE_LIMITS.items():
        if path.startswith(prefix):
            limit, window = lim, win
            break
    if limit is None:
        return
    now = time.monotonic()
    key = f"{ip}:{path}"
    with _rate_lock:
        _rate_store[key] = [t for t in _rate_store[key] if now - t <
window]
        if len(_rate_store[key]) >= limit:
            raise HTTPException(
status_code=429,
                detail=f"Слишкоммногозапросов. Подождите {window}
секунд.",
            )
        _rate_store[key].append(now)
```

```
@app.middleware("http")
async def rate_limit_middleware(request: Request, call_next):
ip = request.client.host if request.client else "unknown"
    try:
        _check_rate_limit(ip, request.url.path)
    except HTTPException as e:
log.warning("Rate limit: ip=%s path=%s", ip, request.url.path)
        return JSONResponse(status_code=e.status_code,
content={"detail": e.detail})
    return await call_next(request)

# ── Базаданных
──────────────────────────────────────────────────────────────

DB_PATH = os.path.join(os.path.dirname(__file__), "db.sqlite")
_local = threading.local()

def _get_conn() -> sqlite3.Connection:
    if not hasattr(_local, "conn"):
        _local.conn = sqlite3.connect(DB_PATH,
check_same_thread=False)
        _local.conn.row_factory = sqlite3.Row
        _local.conn.execute("PRAGMA journal_mode=WAL")
        _local.conn.execute("""
            CREATE TABLE IF NOT EXISTS users (
                username      TEXT PRIMARY KEY,
public_key    TEXT NOT NULL,
chat_id       TEXT NOT NULL,
registered_at TEXT NOT NULL
            )
        """)
        _local.conn.execute("""
            CREATE TABLE IF NOT EXISTS send_log (
                id         INTEGER PRIMARY KEY AUTOINCREMENT,
                sender     TEXT NOT NULL,
recipient  TEXT NOT NULL,
sent_at    TEXT NOT NULL,
file_size  INTEGER
            )
        """)
        _local.conn.commit()
    return _local.conn

# ── JWT
──────────────────────────────────────────────────────────────
──

def _create_token(username: str) -> str:
    payload = {
        "sub": username,
        "iat": int(time.time()),
        "exp": int(time.time()) + JWT_EXPIRE_SECONDS,
```

```
    }
    return jwt.encode(payload, JWT_SECRET, algorithm=JWT_ALGORITHM)

def _decode_token(token: str) -> str:
    try:
        payload = jwt.decode(token, JWT_SECRET,
algorithms=[JWT_ALGORITHM])
        return payload["sub"]
    except jwt.ExpiredSignatureError:
        raise HTTPException(status_code=401, detail="Токенистёк.
Зарегистрируйтесьзаново.")
    except jwt.InvalidTokenError:
        raise HTTPException(status_code=401,
detail="Недействительныйтокен.")

def get_current_user(
    credentials: HTTPAuthorizationCredentials = Depends(security),
) -> str:
    return _decode_token(credentials.credentials)

# ── Проверка Chat ID
──────────────────────────────────────────────────────

async def _verify_chat_id(chat_id: str) -> bool:
    token = os.getenv("TELEGRAM_BOT_TOKEN")
    if not token:
log.warning("TELEGRAM_BOT_TOKEN незадан — проверкаchat_idпропущена")
        return True
    try:
        async with httpx.AsyncClient(timeout=5.0) as client:
            r = await client.get(
f"https://api.telegram.org/bot{token}/getChat",
                params={"chat_id": chat_id},
            )
            return r.json().get("ok", False)
    except Exception as e:
log.error("Ошибкапроверкиchat_id %s: %s", chat_id, e)
        returnTrue  # При недоступности Telegram не блокируем

# ── Схемы
──────────────────────────────────────────────────────────────────
─

classUserIn(BaseModel):
username: str
public_key: str
chat_id: str
old_key_signature: str | None = None  # base64,
нужнатолькоприперерегистрации

    @field_validator("username")
    @classmethod
```

```
    def username_valid(cls, v: str) -> str:
        v = v.strip().lstrip("@")
        if not v or len(v) > 64:
            raiseValueError("username пустой или слишком длинный")
if not re.match(r"^[\w\-\.]+$", v):
            raiseValueError("username содержит недопустимые
символы")
return v

    @field_validator("chat_id")
    @classmethod
    def chat_id_valid(cls, v: str) -> str:
        v = v.strip()
        if not v.lstrip("-").isdigit():
            raise ValueError("chat_idдолженбытьчислом")
        return v

    @field_validator("public_key")
    @classmethod
    def pubkey_valid(cls, v: str) -> str:
        if "BEGIN PUBLIC KEY" not in v:
            raiseValueError("Некорректный публичный ключ PEM")
return v

# ── Endpoints
─────────────────────────────────────────────────────────────────

@app.post("/register")
async def register(user: UserIn, request: Request):
    conn = _get_conn()
ip = request.client.host if request.client else "unknown"

    existing = conn.execute(
        "SELECT public_key FROM users WHERE username=?",
(user.username,)
).fetchone()

ifexisting:
        # Пользователь уже существует — требуем подпись старым
ключом
if not user.old_key_signature:
            raise HTTPException(
status_code=409,
                detail=(
"Пользовательужесуществует. Для обновления ключа "
                    "предоставьте old_key_signature — подпись нового
публичного "
                    "ключа старым приватным ключом (base64)."
),
            )
        try:
old_pub = load_public(existing["public_key"])
```

```
sig_bytes = base64.b64decode(user.old_key_signature)
            if not verify_signature(old_pub,
user.public_key.encode(), sig_bytes):
                raise HTTPException(
status_code=403,
                    detail="Подписьстарымключомнедействительна.
Обновлениеотклонено.",
                )
        except HTTPException:
            raise
        except Exception as e:
            raise HTTPException(status_code=400,
detail=f"Ошибкапроверкиподписи: {e}")
log.info("Re-register: username=%s ip=%s", user.username, ip)
    else:
log.info("New register: username=%s ip=%s", user.username, ip)

    # Проверка Chat ID через Telegram
    if not await _verify_chat_id(user.chat_id):
        raise HTTPException(
status_code=400,
            detail=(
f"Chat ID {user.chat_id} ненайден в Telegram. "
                "Убедитесь, что вы написали боту хотя бы одно
сообщение."
),
        )

    now = datetime.now(timezone.utc).isoformat()
conn.execute(
        "INSERT OR REPLACE INTO users "
        "(username, public_key, chat_id, registered_at) VALUES
(?,?,?,?)",
        (user.username, user.public_key, user.chat_id, now),
    )
conn.commit()

    token = _create_token(user.username)
log.info("Token issued: username=%s", user.username)
    return {
        "status": "ok",
        "username": user.username,
        "token": token,
        "expires_in": JWT_EXPIRE_SECONDS,
    }


@app.get("/user/{username}")
def get_user(username: str):
    row = _get_conn().execute(
        "SELECT public_key, chat_id FROM users WHERE username=?",
(username,)
```

```
).fetchone()
    if row is None:
        raise HTTPException(status_code=404,
detail="Пользовательненайден")
    return {"public_key": row["public_key"], "chat_id":
row["chat_id"]}


@app.post("/send")
async def send_file_api(
    request: Request,
    username: str,
    file: UploadFile = File(...),
    sender: str = Depends(get_current_user),  # JWT обязателен
):
    conn = _get_conn()
ip = request.client.host if request.client else "unknown"

    row = conn.execute(
        "SELECT chat_id FROM users WHERE username=?", (username,)
).fetchone()
    if row is None:
        raise HTTPException(status_code=404,
detail="Получательненайден")

    content = await file.read()
file_size = len(content)

    with tempfile.NamedTemporaryFile(delete=False, suffix=".bin") as
tmp:
tmp.write(content)
tmp_path = tmp.name

    try:
        await send_file(tmp_path, row["chat_id"],
                caption=f"Зашифрованныйфайлот {sender}")
    finally:
os.unlink(tmp_path)

    # Логируембезсодержимого
    now = datetime.now(timezone.utc).isoformat()
conn.execute(
        "INSERT INTO send_log (sender, recipient, sent_at,
file_size) "
        "VALUES (?,?,?,?)",
        (sender, username, now, file_size),
    )
conn.commit()
log.info("Sent: from=%s to=%s size=%d ip=%s", sender, username,
file_size, ip)
```

```
    return {"status": "sent", "from": sender, "to": username,
"size": file_size}


@app.get("/health")
def health():
    return {"status": "ok", "time":
datetime.now(timezone.utc).isoformat()}


if __name__ == "__main__":
    import uvicorn
    port = int(os.getenv("PORT", 8000))
uvicorn.run(app, host="0.0.0.0", port=port)
```

Приложение В

(обязательное)

Гибридная криптографическая схема для шифрования файлов в Telegram

Презентация

БАКАЛАВРСКАЯ РАБОТА

Гибридная криптографическая схема для шифрования файлов в Telegram

Автор работы: Кащеев Илья Олегович

Руководитель: доцент кафедры ВТ Сорокин А.А.

ПГУ, 2026

Актуальность и цель работы

Отсутствие E2EE

Файлы через Telegram Bot API хранятся на серверах платформы в незашифрованном виде

Небезопасные токены

Токены ботов часто хранятся в клиентском коде, компрометация любого устройства даёт доступ к боту

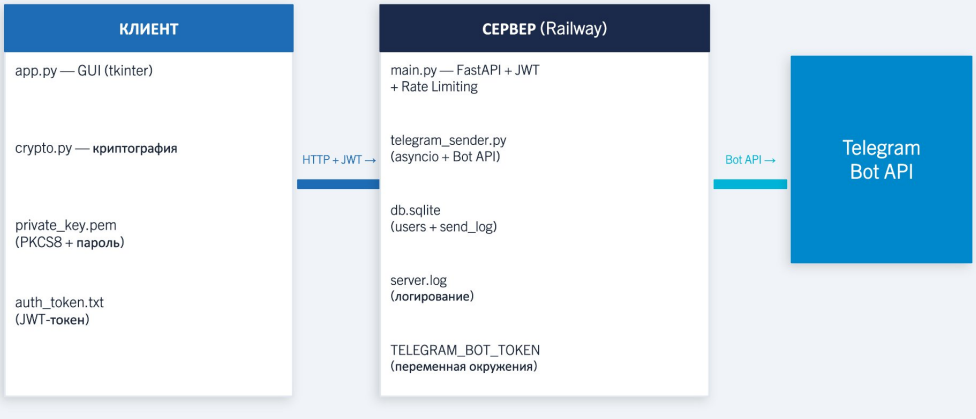
Нет аутентификации

Без механизмов проверки любой может перезаписать чужой публичный ключ и выдать себя за другого

ЦЕЛЬ РАБОТЫ

Разработка кросс-платформенной системы защищённой передачи файлов через Telegram с реализацией E2EE, цифровой подписи, JWT-аутентификации и безопасного хранения токена бота

Архитектура системы



Токен бота хранится только на сервере — клиент с Telegram напрямую не взаимодействует

Криптографическая схема защиты

1	AES-256-GCM Шифрование файла	Генерируется уникальный session_key (32 байта) и nonce (12 байт). Файл шифруется с тегом аутентификации 16 байт — любое изменение шифртекста обнаруживается (InvalidTag)
2	RSA-OAEP-SHA256 Шифрование ключа	session_key шифруется открытым ключом получателя. Расшифровать может только владелец соответствующего приватного ключа (RSA-2048)
3	RSA-PSS-SHA256 Цифровая подпись	Зашифрованные данные подписываются приватным ключом отправителя. Схема encrypt-then-sign исключает атаки на основе выбранного шифртекста
4	PKCS8 + PBKDF2 Защита ключа на диске	Приватный ключ хранится зашифрованным паролем: AES-256-CBC + PBKDF2-HMAC-SHA256. При запуске приложение запрашивает пароль (3 попытки)

Механизмы безопасности сервера

JWT-аутентификация

При регистрации выдаётся токен HS256 (24 часа). Эндпоинт /send требует Bearer токен — сервер знает кто отправляет файл

Защита от подмены ключа

При перерегистрации требуется old_key_signature — подпись нового ключа старым. HTTP 409 без подписи, HTTP 403 при неверной подписи

Rate Limiting

Sliding window без зависимостей: /register 5/60с, /send 20/60с, /user 60/60с. threading.Lock + time.monotonic()

Логирование

Все события в server.log + таблица send_log в SQLite. Отправитель, получатель, размер — без содержимого файлов

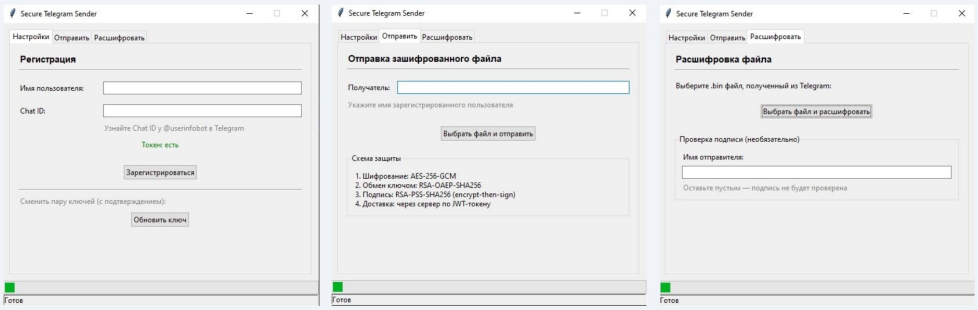
Верификация Chat ID

При регистрации запрос getChat к Telegram Bot API через httpx. Нельзя зарегистрироваться с несуществующим Chat ID

Деплой на Railway

Сервер развёрнут на облачной платформе Railway. Токен и JWT_SECRET в переменных окружения. Доступен 24/7 без локального сервера

Клиентское приложение — интерфейс



При первом запуске — диалог создания пароля для приватного ключа (3 попытки при последующих запусках)

Полный цикл отправки и получения файла

1	GET /user/{получатель} → pub_key
2	session_key = os.urandom(32)
3	encrypted = AES-256-GCM(file, key)
4	enc_key = RSA-OAEP(key, pub_key)
5	sig = RSA-PSS(encrypted, priv)
6	bundle = pack(enc_key, sig, name, encrypted)
7	POST /send + Bearer JWT → Railway → Telegram
8	**Получатель:** unpack → verify sig → RSA-decrypt → AES-decrypt → **файл**

Тестирование системы

Сц. 1-2 Базовый сценарий — ✓ Успешно

Регистрация → отправка 1 МБ → расшифровка с верификацией подписи. Содержимое совпало с оригиналом

Сц. 3-4 Защита ключей — ✓ Успешно

HTTP 409 без old_key_signature. Обновление ключа с подписью — сервер верифицировал, выдал новый JWT

Сц. 5-6 Целостность данных — ✓ Успешно

Изменение байта в bundle → InvalidTag. Неверное имя отправителя → verify_signature=False, отмена расшифровки

Сц. 7 Rate Limiting — ✓ Успешно

6-й запрос к /register → HTTP 429. Через 60 сек запросы возобновились. Записи в server.log

Сц. 8 JWT-аутентификация — ✓ Успешно

Истёкший токен (1 сек) → HTTP 401. Клиент отобразил «Токен истёк, зарегистрируйтесь заново»

Сц. 9-10 Кросс-платформа — ✓ Успешно

macOS→Windows→Linux: файлы расшифрованы корректно. Неверный пароль 3 раза → безопасное завершение

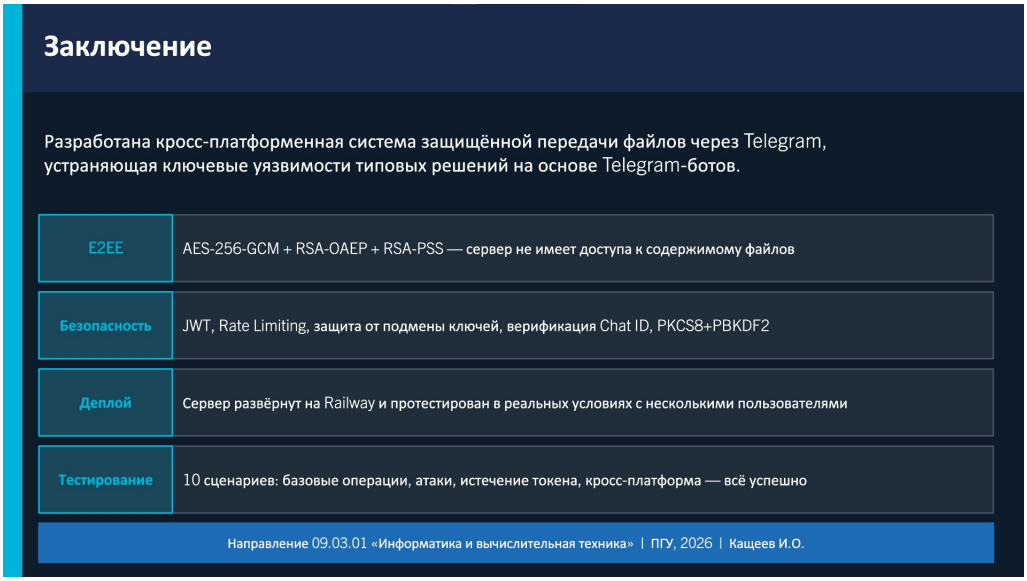
Результаты разработки

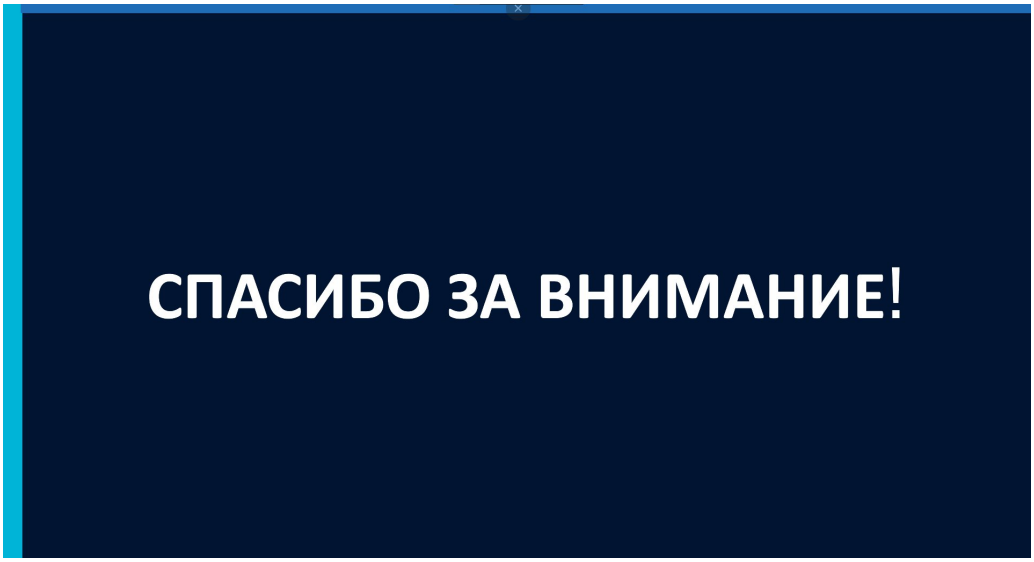
Реализовано:

- Криптографический модуль: AES-256-GCM, RSA-OAEP, RSA-PSS, PKCS8+PBKDF2
- Серверное приложение на FastAPI: JWT, Rate Limiting, верификация Chat ID, логирование, защита от подмены ключей
- Кросс-платформенный GUI (tkinter): macOS, Windows, Linux
- Бинарный пакет с оригинальным именем файла: автоматическое восстановление при расшифровке

Достигнуто:

- Сервер заделпоен на Railway — доступен без локального запуска
- 10 сценариев тестирования пройдено успешно
- Токен Telegram-бота только на сервере: защита от компрометации через клиент
- Исходный код опубликован на GitHub


Заключение
Разработана кросс-платформенная система защищённой передачи файлов через Telegram, устраняющая ключевые уязвимости типовых решений на основе Telegram-ботов.
E2EE
AES-256-GCM + RSA-OAEP + RSA-PSS — сервер не имеет доступа к содержимому файлов
Безопасность
JWT, Rate Limiting, защита от подмены ключей, верификация Chat ID, PKCS8+PBKDF2
Деплой
Сервер развёрнут на Railway и протестирован в реальных условиях с несколькими пользователями
Тестирование
10 сценариев: базовые операции, атаки, истечение токена, кросс-платформа — всё успешно
Направление 09.03.01 «Информатика и вычислительная техника» | ПГУ, 2026 | Кащеев И.О.


СПАСИБО ЗА ВНИМАНИЕ!