**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**



ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

**КУРСОВОЙ ПРОЕКТ**

по дисциплине: Компьютерные сети

тема: «Приложение для безопасной передачи данных (TLS)»

Автор работы \_\_\_\_\_\_\_\_\_\_\_\_\_ Игнатьев Артур Олегович ПВ-223

(подпись)

Руководитель проекта \_\_\_\_\_\_\_\_\_\_\_\_\_ Федотов Евгений Александрович

(подпись)

Оценка \_\_\_\_\_\_

Белгород 2025 г

Содержание

Введение…………………………………………………………………………...3

1. Анализ протоколов передачи данных…………………………………………5

1.1. Обзор протоколов передачи данных……………………………………5

1.2. Сравнение протоколов…………………………………………………..6

1.3. Криптографические механизмы в TLS…………………………………7

1.4. Роль сертификатов в TLS………………………………………………..8

2. Разработка клиент-серверного приложения…………………………………..9

2.1 Разработка сервера………………………………………………………9

2.2 Разработка клиента……………………………………………………..11

2.3. Генерация сертификатов……………………………………………….13

2.4 Архитектура приложения………………………………………………14

2.5. Преимущества и ограничения…………………………………………14

Вывод о проделанной работе……………………………………………………15

Список литературы………………………………………………………………16

Приложения………………………………………………………………………17

Приложение 1. Исходный код generate\_cert.py……………………………17

Приложение 2. Исходный код server.py……………………………………17

Приложение 3. Исходный код client.py…………………………………….22

Введение

Ежедневно по всему миру совершается более 2,2 тысяч кибератак, таких как перехват данных, атаки “человек посередине” (MITM) и несанкционированный доступ, что приводит к многочисленным утечкам персональных данных. Защита данных при передачи через сеть в настоящее время является необходимостью. Для этой цели используются различные методы шифрования. Протокол TLS (Transport Layer Security) стал стандартом для обеспечения безопасности сетевых соединений и используется в веб-браузерах, мессенджерах и других приложениях. Приложение с использованием этого протокола гарантирует конфиденциальность, аутентификацию и целостность данных при обмене данными между клиентом и сервером.

Целью данной курсовой работы является разработка клиент-серверного приложения для безопасной передачи данных, использующего протокол TLS для шифрования. Приложение должно обеспечивать аутентификацию пользователей, защиту данных от перехвата, управление сертификатами и поддержку различных криптографических алгоритмов.

Для выполнения работы были поставлены следующие задачи:

1. Изучить протоколы передачи данных и обосновать выбор TLS.
2. Разработать серверную часть приложения обеспечивающую безопасное соединения, аутентификацию.
3. Разработать клиентскую часть приложения с графическим интерфейсом для взаимодействия с сервером.
4. Реализовать функционал для генерации и управления сертификатами.
5. Обеспечить работу сообщений в реальном времени с сохранением истории переписки.

В работе был использован язык программирования Python, библиотеки cryptography для работы с сертификатами, ssl для настройки TLS, PyQt6 для создания графического интерфейса и SQLite для хранения данных. Эти инструменты были выбраны из-за простоты, кроссплатформенности, надёжностью и широкой поддержкой сообщества.

1. Анализ протоколов передачи данных

1.1. Обзор протоколов передачи данных

Обеспечение безопасности передачи данных в сети является необходимостью в настоящее время. Для защиты информации от перехвата, модификации и несанкционированного доступа используются различные протоколы, такие как TLS (Transport Layers Security), SSL (Secure Sockets Layer), SSH (Secure Shell) и IPsec.

TLS — это криптографический протокол, обеспечивающий шифрование данных, аутентификацию сторон и проверку целостности сообщений. Он широко применяется в веб-браузерах, мессенджерах и других приложениях. Последняя версия, TLS 1.3, предлагает улучшенную безопасность и производительность по сравнению с предыдущими версиями. Основные преимущества TLS:

1. Поддержка современных шифров, таких как AES и ChaCha20;
2. Механизмы Perfect Forward Secrecy (PFS), защищающие данные даже при компрометации ключей.
3. Гибкость в настройке криптографических алгоритмов.
4. Широкая совместимость с различными платформами и библиотеками.

SSL - предшественник TLS, разработанный в 1990-х годах. Несмотря на схожие принципы работы, SSL устарел из-за уязвимостей, таких как атака POODLE и слабые шифры. Использование SSL в современных приложениях не рекомендуется, так как он не соответствует текущим стандартам безопасности.

SSH - предназначен для безопасного удалённого доступа и передачи файлов. Он обеспечивает шифрование и аутентификацию, но ориентирован на специфические задачи, такие как управление серверами. SSH менее подходит для приложений реального времени, таких как чаты, из-за отсутствия гибкости в интеграции с пользовательскими интерфейсами.

IPsec - используется для защиты данных на уровне сетевого протокола. Он эффективен для создания VPN и защиты корпоративных сетей, но сложен в настройке для приложений с графическим интерфейсом. Кроме того, IPsec требует значительных ресурсов для управления ключами и туннелям.

1.2. Сравнение протоколов

При выборе протокола были рассмотрены следующие критерии:

1. Уровень безопасности: способность противостоять атакам, таким как MITM, и поддержка современных шифров.
2. Простота интеграции: совместимость с Python и клиент-серверной архитектурой.
3. Производительность: скорость установления соединения и обработки данных.
4. Гибкость: возможность настройки параметров безопасности.
5. Распространённость: поддержка сообществом и наличие документации.

Сравнительная таблица:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Протокол | Безопасность | Интеграция | Производительность | Гибкость | Распространённость |
| TLS | Высокая | Простая | Высокая | Высокая | Широкая |
| SSL | Низкая | Средняя | Низкая | Средняя | Устаревшая |
| SSH | Высокая | Сложная | Средняя | Низкая | Средняя |
| IPsec | Высокая | Сложная | Средняя | Средняя | Средняя |

TLS 1.3 был выбран для приложения по следующим причинам: Исключение устаревших шифров (MD5, SHA-1) и поддержка PFS. Модуль ssl в Python позволяет легко настроить TLS для сокетов. TLS 1.3 сокращает время установления соединения за счёт упрощённого рукопожатия. Поддержка различных наборов шифров, таких как DEFAULT@SECLEVEL=2, обеспечивает баланс между безопасностью и совместимостью. TLS используется в большинстве современных приложений, что делает его изучение актуальным.

1.3. Криптографические механизмы в TLS

Криптографические алгоритмы используемые в приложении:

1. RSA (2048 бит) - применяется для генерации ключей и подписи сертификатов. RSA выбран за его надёжность и поддержку в TLS. Размер ключа 2048 бит обеспечивает достаточную стойкость при приемлемой производительности.
2. SHA-256 - используется для хеширования при подписи сертификатов. Этот алгоритм устойчив к коллизиям и рекомендован для TLS 1.3.
3. AES - применяется в составе наборов шифров TLS 1.3 для шифрования данных. AES выбран за высокую скорость и криптографическую стойкость.
4. Bcrypt – не часть TLS, но алгоритм используется для хеширования паролей пользователей. Bcrypt обеспечивает защиту от атак перебора благодаря адаптивной сложности.

Данные алгоритмы были выбраны, так как являются оптимальным соотношением безопасности, производительности и поддерживаются библиотекой cryptography.

1.4. Роль сертификатов в TLS

Сертификаты X.509 обеспечивают аутентификацию сторон. В приложении сервер использует самоподписанный сертификат, сгенерированный с помощью скрипта generate\_cert.py. Клиенты могут создавать свои сертификаты.

Причина использования самоподписанных сертификатов, это отсутствие необходимости доверительного центра сертификации (CA). Однако рекомендуется использовать сертификаты выданные доверенным CA, что бы повысить доверие к серверу и клиентам.

2. Разработка клиент-серверного приложения

2.1. Разработка сервера

Для обеспечения безопасности сервер использует протокол TLS 1.3, настроенный через модуль ssl в Python. Настройка TLS реализована с использованием самоподписного сертификата и ключа, сгенерированных с помощью библиотеки cryptography. Сертификат привязан к IP-адресу 127.0.0.1, для локального использования, но вместо него может быть привязан к IP-адресу удалённого сервера. Использовалась настройка DEFAULT@SECLEVEL=2, которая обеспечивает поддержку только безопасных шифров, рекомендованных для TLS 1.3. Это позволяет избежать уязвимостей, связанных с устаревшими алгоритмами, такими как MD5 или SHA-1.

Аутентификация реализована через базу данных SQLite, где хранятся логины и хеши паролей. Для хеширования паролей выбран алгоритм bcrypt по следующим причинам:

1. Автоматически генерирует и интегрирует соль для каждого пароля, что предотвращает использование радужных таблиц и делает атаки по словарю менее эффективными.
2. Позволяет изменять сложность хеширования, делая его более устойчивым к атакам методом подбора даже при увеличении вычислительной мощности.
3. Использует итеративное хеширование и рассчитан на длительную работу, что затрудняет перебор хешей.
4. Легко интегрируется в различные языки программирования и платформы.
5. Защищает пароли от несанкционированного доступа, предотвращая утечку данных.

Сервер обрабатывает две команды для аутентификации:

1. LOGIN - проверяет логин и пароль пользователя, сравнивая введённый пароль с хешем в базе данных.
2. REGISTER - создаёт новую запись в базе данных, если логин уникален, и сохраняет хеш пароля

В случае успешной аутентификации клиент добавляется в список активных подключений, что позволяет серверу отслеживать онлайн-пользователей.

Сервер поддерживает обмен сообщениями в реальном времени с сохранением истории. Сообщения хранятся в таблице messages базы данных SQLite с указанием логина отправителя, текста сообщения и временной метки. При подключении нового клиента сервер отправляет уже написанные сообщения пользователю. Сообщения рассылаются всем подключённым клиентам, кроме отправителя, с помощью механизма широковещательной рассылки broadcast\_message. Для синхронизации доступа к списку клиентов используется блокировка threading.Lock, что предотвращает ошибки в многопоточной среде.

Клиенты могут загружать свои сертификаты на сервер с помощью команды UPLOAD\_CERT. Сертификаты сохраняются в таблице certificates базы данных SQLite. Это позволяет серверу хранить информацию о сертификатах пользователей для будущей проверки подлинности.

Для обработки нескольких клиентов одновременно сервер использует модуль threading. Каждое новое подключение обрабатывается в отдельном потоке, что обеспечивает масштабируемость. Модуль был выбран, так как он проще в реализации по сравнению с асинхронным программированием и обеспечивает независимую обработку клиентских запросов без блокировки основного потока сервера.

Для отладки и мониторинга работы сервера реализована система логирования с использованием модуля logging. Логи включают информацию о подключениях, аутентификации, ошибках и отправке сообщений. Это помогает отслеживать работу приложения и выявлять потенциальные проблемы.

2.2. Разработка клиента

Клиентская часть реализована с использованием библиотеки PyQt6 для создания графического интерфейса. Она обеспечивает удобное взаимодействие пользователя с сервером, включая вход, регистрацию, отправку сообщений и управление сертификатами.

Графический интерфейс включает три формы: вход, регистрация и чат. PyQt6 был выбран по следующим причинам:

1. Кроссплатформенность, позволяющая запускать приложение на Windows, macOS и Linux.
2. Богатый набор виджетов (QTextEdit, QListWidget, QPushButton), упрощающий создание функционального интерфейса.
3. Поддержка стилей через CSS-подобный синтаксис, что позволило создать современный дизайн с полупрозрачным фоном и кастомными кнопками.

Интерфейс реализован без стандартной рамки окна (Qt.FramelessWindowHint), с поддержкой перетаскивания, минимизации и максимизации. Это придаёт приложению современный вид и улучшает пользовательский опыт. Формы включают:

1. Форма входа - поля для ввода логина и пароля, кнопки для входа и перехода к регистрации.
2. Форма регистрации - поля для логина, пароля и его подтверждения с валидацией.
3. Форма чата - текстовое поле для сообщений, список онлайн-пользователей и кнопки для отправки сообщений, генерации сертификатов и выхода.

Клиент устанавливает соединение с сервером через TLS 1.3, используя модуль ssl. Перед подключением проверяется наличие сертификата сервера server.crt. Настройка context.verify\_mode = ssl.CERT\_REQUIRED гарантирует, что соединение будет установлено только при наличии действительного сертификата. Это предотвращает подключение к ненадёжным серверам и защищает от атак MITM.

Функция generate\_certificate позволяет клиентам создавать самоподписанные сертификаты с использованием библиотеки cryptography. Сертификат содержит логин пользователя в поле Common Name и имеет срок действия один год. После генерации сертификат кодируется в Base64 и отправляется на сервер через команду UPLOAD\_CERT.

Клиент отправляет сообщения на сервер через защищённое TLS-соединение. Для получения сообщений и обновления списка онлайн-пользователей используется отдельный поток threading, что предотвращает зависание интерфейса. Полученные сообщения отображаются в текстовом поле, а список пользователей — в виджете QListWidget. Для обработки событий (например, обновления сообщений) используется сигнал-слот-механизм PyQt6.

Клиент обрабатывает различные ошибки, такие как:

1. Отсутствие сертификата сервера.
2. Неверный логин или пароль.
3. Разрыв соединения с сервером.
4. Ошибки при генерации или загрузке сертификатов.

Ошибки отображаются пользователю через диалоговые окна QMessageBox, что упрощает диагностику проблем. Логирование с использованием модуля logging помогает разработчику отслеживать ошибки и состояние приложения.

При регистрации клиент проверяет:

1. Уникальность логина (сервер возвращает REG\_FAILED, если логин занят).
2. Минимальную длину пароля (6 символов).
3. Соответствие логина формату (буквы, цифры, подчёркивание).

2.3. Генерация сертификатов

Скрипт generate\_cert.py отвечает за создание самоподписанного сертификата и ключа для сервера. Используется библиотека cryptography с алгоритмом RSA (2048 бит) и хешем SHA-256. Причины выбора:

1. RSA-2048 обеспечивает достаточную криптографическую стойкость.
2. SHA-256 является стандартом для TLS 1.3 и устойчив к атакам на коллизии.
3. Библиотека cryptography проста в использовании и широко поддерживается.

2.4 Архитектура приложения

Архитектура приложения основана на модели клиент-сервер:

1. Сервер - слушает подключения на порту 443, обрабатывает команды LOGIN, REGISTER, UPLOAD\_CERT и рассылает сообщения.
2. Клиент - подключается к серверу, отправляет команды и сообщения, отображает данные в графическом интерфейсе.
3. База данных - хранит информацию о пользователях, сообщениях и сертификатах.

Коммуникация между клиентом и сервером происходит через сокеты с TLS-шифрованием. Формат сообщений включает префиксы, что упрощает их обработку.

2.5. Преимущества и ограничения

Преимущества разработанного приложения:

1. Безопасность соединения благодаря TLS 1.3.
2. Простота использования за счёт интуитивного интерфейса.
3. Поддержка аутентификации, хранения сообщений и управления сертификатами.
4. Лёгкость масштабирования за счёт многопоточности.

Ограничения:

1. Использование самоподписанных сертификатов.
2. Отсутствие end-to-end шифрования сообщений.
3. Ограниченная функциональность (нет поддержки передачи файлов).

Вывод о проделанной работе

В рамках курсовой работы было разработано клиент-серверное приложение для безопасной передачи данных с использованием протокола TLS 1.3. Основная цель - создание системы, обеспечивающей шифрование данных, аутентификацию пользователей и управление сертификатами, — была достигнута. Приложение позволяет пользователям обмениваться сообщениями в реальном времени через защищённое соединение, регистрироваться, входить в систему и генерировать сертификаты.

Разработанное приложение демонстрирует основные принципы безопасной передачи данных с использованием TLS. Оно может использоваться в образовательных целях для изучения сетевой безопасности, криптографии и разработки клиент-серверных систем. Полученные навыки работы с Python, TLS, cryptography и PyQt6 являются хорошим опытом для дальнейших проектов в области информационной безопасности и сетевых технологий. Приложение выполняет поставленные задачи, но имеет потенциал для улучшения, что делает его хорошей основой для дальнейшей доработки.

Список литературы

1. Python 3.13.4 documentation

Режим доступа: <https://docs.python.org/3/>

1. Welcome to pyca/cryptography

Режим доступа: <https://cryptography.io/en/latest/>

Qt for Python

Режим доступа: <https://doc.qt.io/qtforpython-6/index.html>

1. SQLite

Режим доступа: <https://www.sqlite.org/docs.html>

1. The Transport Layer Security (TLS) Protocol Version 1.3

Режим доступа: <https://datatracker.ietf.org/doc/html/rfc8446>

1. Руководство по выживанию — TLS/SSL и сертификаты SSL (X.509)

Режим доступа: <https://www.opennet.ru/docs/RUS/ldap_apacheds/tech/ssl.html>

1. OpenSSL Documentation.

Режим доступа: <https://docs.openssl.org/master/>

1. bcrypt 4.3.0.

Режим доступа: <https://pypi.org/project/bcrypt/>

1. Руководство по сетевому программированию на Python.

Режим доступа: <https://metanit.com/python/network/>

Приложения

Приложение 1. Исходный код generate\_cert.py

from cryptography import x509

from cryptography.x509.oid import NameOID

from cryptography.hazmat.primitives import hashes

from cryptography.hazmat.primitives.asymmetric import rsa

from cryptography.hazmat.primitives import serialization

import datetime

import ipaddress

private\_key = rsa.generate\_private\_key(

public\_exponent=65537,

key\_size=2048

)

subject = x509.Name([

x509.NameAttribute(NameOID.COMMON\_NAME, "127.0.0.1")

])

builder = (

x509.CertificateBuilder()

.subject\_name(subject)

.issuer\_name(subject)

.not\_valid\_before(datetime.datetime.now(datetime.UTC))

.not\_valid\_after(datetime.datetime.now(datetime.UTC) + datetime.timedelta(days=365))

.serial\_number(x509.random\_serial\_number())

.public\_key(private\_key.public\_key())

.add\_extension(

x509.SubjectAlternativeName([x509.IPAddress(ipaddress.ip\_address("127.0.0.1"))]),

critical=False

)

)

certificate = builder.sign(private\_key, hashes.SHA256())

with open("server.crt", "wb") as f:

f.write(certificate.public\_bytes(serialization.Encoding.PEM))

with open("server.key", "wb") as f:

f.write(private\_key.private\_bytes(

encoding=serialization.Encoding.PEM,

format=serialization.PrivateFormat.TraditionalOpenSSL,

encryption\_algorithm=serialization.NoEncryption()

))

Приложение 2. Исходный код server.py

import ssl

import socket

import threading

import sqlite3

import bcrypt

import datetime

import logging

import sys

logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')

clients = []

clients\_lock = threading.Lock()

def create\_server\_context():

context = ssl.SSLContext(ssl.PROTOCOL\_TLS\_SERVER)

context.minimum\_version = ssl.TLSVersion.TLSv1\_3

context.maximum\_version = ssl.TLSVersion.TLSv1\_3

context.set\_ciphers('DEFAULT@SECLEVEL=2')

context.load\_cert\_chain(certfile="server.crt", keyfile="server.key")

logging.info(f"Создан серверный SSL-контекст: min={context.minimum\_version}, max={context.maximum\_version}")

logging.debug(f"Доступные шифры: {context.get\_ciphers()}")

return context

def init\_db():

conn = sqlite3.connect("users.db")

cursor = conn.cursor()

cursor.execute("""

CREATE TABLE IF NOT EXISTS users (

login TEXT PRIMARY KEY,

password\_hash TEXT NOT NULL

)

""")

cursor.execute("""

CREATE TABLE IF NOT EXISTS messages (

id INTEGER PRIMARY KEY AUTOINCREMENT,

login TEXT NOT NULL,

message TEXT NOT NULL,

timestamp TEXT NOT NULL

)

""")

cursor.execute("""

CREATE TABLE IF NOT EXISTS certificates (

login TEXT PRIMARY KEY,

certificate TEXT NOT NULL

)

""")

conn.commit()

conn.close()

logging.info("База данных инициализирована")

def register\_user(login, password):

conn = sqlite3.connect("users.db")

cursor = conn.cursor()

password\_hash = bcrypt.hashpw(password.encode(), bcrypt.gensalt()).decode()

try:

cursor.execute("INSERT INTO users (login, password\_hash) VALUES (?, ?)", (login, password\_hash))

conn.commit()

logging.info(f"Пользователь {login} зарегистрирован")

return True

except sqlite3.IntegrityError:

logging.warning(f"Попытка регистрации существующего логина: {login}")

return False

finally:

conn.close()

def authenticate\_user(login, password):

conn = sqlite3.connect("users.db")

cursor = conn.cursor()

cursor.execute("SELECT password\_hash FROM users WHERE login = ?", (login,))

result = cursor.fetchone()

conn.close()

if result and bcrypt.checkpw(password.encode(), result[0].encode()):

logging.info(f"Аутентификация успешна для {login}")

return True

logging.warning(f"Неудачная аутентификация для {login}")

return False

def save\_certificate(login, certificate):

conn = sqlite3.connect("users.db")

cursor = conn.cursor()

try:

cursor.execute("INSERT OR REPLACE INTO certificates (login, certificate) VALUES (?, ?)", (login, certificate))

conn.commit()

logging.info(f"Сертификат сохранен для {login}")

return True

except Exception as e:

logging.error(f"Ошибка при сохранении сертификата для {login}: {e}")

return False

finally:

conn.close()

def save\_message(login, message):

conn = sqlite3.connect("users.db")

cursor = conn.cursor()

timestamp = datetime.datetime.now().isoformat()

cursor.execute("INSERT INTO messages (login, message, timestamp) VALUES (?, ?, ?)", (login, message, timestamp))

conn.commit()

conn.close()

logging.info(f"Сообщение сохранено: {login}: {message}")

def get\_message\_history():

conn = sqlite3.connect("users.db")

cursor = conn.cursor()

cursor.execute("SELECT login, message, timestamp FROM messages ORDER BY timestamp LIMIT 50")

messages = cursor.fetchall()

conn.close()

logging.info("История сообщений получена")

return messages

def broadcast\_message(message, sender\_socket, login):

with clients\_lock:

for client\_socket, client\_login in clients:

if client\_socket != sender\_socket:

try:

client\_socket.send(f"{login}: {message}".encode())

logging.debug(f"Сообщение отправлено клиенту {client\_login}: {login}: {message}")

except Exception as e:

logging.error(f"Ошибка при отправке сообщения клиенту {client\_login}: {e}")

def broadcast\_user\_list():

with clients\_lock:

user\_list = ",".join(client[1] for client in clients)

for client\_socket, \_ in clients:

try:

client\_socket.send(f"USER\_LIST:{user\_list}".encode())

logging.debug(f"Список пользователей отправлен: {user\_list}")

except Exception as e:

logging.error(f"Ошибка при отправке списка пользователей: {e}")

def handle\_client(client\_socket, addr):

try:

logging.debug(f"Ожидание данных от {addr}")

data = client\_socket.recv(1024).decode()

if not data:

logging.warning(f"Пустые данные от {addr}, закрытие соединения")

client\_socket.close()

return

parts = data.split(":", 2)

command = parts[0]

logging.info(f"Получена команда от {addr}: {command}")

if command == "LOGIN":

login, password = parts[1], parts[2]

if authenticate\_user(login, password):

client\_socket.send("AUTH\_SUCCESS".encode())

with clients\_lock:

clients.append((client\_socket, login))

logging.info(f"Клиент {login} ({addr}) подключился, TLS: {client\_socket.version()}, Шифр: {client\_socket.cipher()}")

for login\_, msg, \_ in get\_message\_history():

try:

client\_socket.send(f"{login\_}: {msg}".encode())

logging.debug(f"Отправлено историческое сообщение клиенту {login}: {login\_}: {msg}")

except Exception as e:

logging.error(f"Ошибка при отправке истории клиенту {login}: {e}")

broadcast\_user\_list()

else:

client\_socket.send("AUTH\_FAILED".encode())

logging.warning(f"Неудачная аутентификация для {login} от {addr}")

client\_socket.close()

return

elif command == "REGISTER":

login, password = parts[1], parts[2]

if register\_user(login, password):

client\_socket.send("REG\_SUCCESS".encode())

logging.info(f"Успешная регистрация {login} от {addr}")

else:

client\_socket.send("REG\_FAILED".encode())

logging.warning(f"Неудачная регистрация {login} от {addr}")

client\_socket.close()

return

elif command == "UPLOAD\_CERT":

login, certificate = parts[1], parts[2]

if save\_certificate(login, certificate):

client\_socket.send("CERT\_SUCCESS".encode())

logging.info(f"Сертификат успешно загружен для {login} от {addr}")

else:

client\_socket.send("CERT\_FAILED".encode())

logging.error(f"Ошибка загрузки сертификата для {login} от {addr}")

client\_socket.close()

return

else:

client\_socket.send("INVALID\_COMMAND".encode())

logging.warning(f"Недопустимая команда от {addr}: {command}")

client\_socket.close()

return

while True:

try:

logging.debug(f"Ожидание сообщения от {addr} ({login})")

message = client\_socket.recv(1024).decode()

if not message:

logging.warning(f"Пустое сообщение от {addr} ({login}), закрытие соединения")

break

logging.info(f"Получено сообщение от {addr} ({login}): {message}")

save\_message(login, message)

broadcast\_message(message, client\_socket, login)

except Exception as e:

logging.error(f"Ошибка при получении сообщения от {addr} ({login}): {e}")

break

except Exception as e:

logging.error(f"Общая ошибка с клиентом {addr}: {e}")

finally:

with clients\_lock:

for client in clients:

if client[0] == client\_socket:

clients.remove(client)

logging.info(f"Клиент {client[1]} ({addr}) отключился")

break

client\_socket.close()

broadcast\_user\_list()

def main():

init\_db()

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

context = create\_server\_context()

server\_socket = context.wrap\_socket(server\_socket, server\_side=True)

server\_socket.bind(('127.0.0.1', 443))

server\_socket.listen(5)

logging.info("Сервер запущен на 127.0.0.1:443")

logging.debug(f"Версия Python: {sys.version}, Версия OpenSSL: {ssl.OPENSSL\_VERSION}")

while True:

try:

client\_socket, addr = server\_socket.accept()

logging.info(f"Новое подключение: {addr}, TLS: {client\_socket.version()}, Шифр: {client\_socket.cipher()}")

threading.Thread(target=handle\_client, args=(client\_socket, addr)).start()

except Exception as e:

logging.error(f"Ошибка при принятии подключения: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

Приложение 3. Исходный код client.py

import sys

import ssl

import socket

import threading

import os

import re

import base64

import logging

import datetime

from PyQt6.QtWidgets import (

QApplication, QWidget, QVBoxLayout, QHBoxLayout, QLabel,

QLineEdit, QPushButton, QTextEdit, QListWidget, QMessageBox

)

from PyQt6.QtCore import Qt, pyqtSignal, QObject, QPoint

from PyQt6.QtGui import QIcon

from cryptography import x509

from cryptography.hazmat.primitives import serialization, hashes

from cryptography.hazmat.primitives.asymmetric import rsa

from cryptography.x509.oid import NameOID

logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')

class Communicate(QObject):

update\_text = pyqtSignal(str)

update\_users = pyqtSignal(list)

class TLSChatClient(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("TLS Чат")

self.resize(600, 400)

self.client\_socket = None

self.cert\_path = "server.crt" if os.path.exists("server.crt") else None

self.current\_login = None

self.current\_password = None

self.comm = Communicate()

self.comm.update\_text.connect(self.update\_message\_text)

self.comm.update\_users.connect(self.update\_user\_list)

self.is\_maximized = False

self.old\_pos = None

self.setWindowIcon(QIcon('icon.png'))

self.setStyleSheet("""

QWidget#MainWindow {

background-color: rgba(30, 30, 30, 230);

border-radius: 10px;

color: #ffffff;

font-family: 'Segoe UI', Arial, sans-serif;

font-size: 14px;

}

QWidget#TitleBar {

background-color: rgba(20, 20, 20, 240);

border-top-left-radius: 10px;

border-top-right-radius: 10px;

}

QWidget#Content {

background-color: rgba(30, 30, 30, 230);

}

QLineEdit {

background-color: rgba(50, 50, 50, 200);

border: 1px solid #555555;

border-radius: 10px;

padding: 8px;

color: #ffffff;

}

QLineEdit:focus {

border: 1px solid #1e90ff;

}

QPushButton {

background-color: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #1e90ff, stop:1 #4682b4);

border: none;

border-radius: 10px;

padding: 10px;

color: #ffffff;

font-weight: bold;

}

QPushButton:hover {

background-color: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #4682b4, stop:1 #1e90ff);

}

QPushButton:pressed {

background-color: #1c6ea4;

}

QPushButton#TitleBarButton {

background-color: transparent;

border: none;

border-radius: 0px;

padding: 5px;

min-width: 30px;

max-width: 30px;

}

QPushButton#TitleBarButton:hover {

background-color: rgba(255, 255, 255, 50);

}

QPushButton#CloseButton:hover {

background-color: rgba(255, 0, 0, 100);

}

QTextEdit {

background-color: rgba(40, 40, 40, 200);

border: 1px solid #555555;

border-radius: 10px;

padding: 5px;

color: #ffffff;

}

QListWidget {

background-color: rgba(40, 40, 40, 200);

border: 1px solid #555555;

border-radius: 10px;

padding: 5px;

color: #ffffff;

}

QLabel {

color: #ffffff;

font-weight: bold;

}

QMessageBox {

background-color: rgba(40, 40, 40, 200);

border: 1px solid #555555;

border-radius: 10px;

padding: 5px;

color: #ffffff;

}

""")

self.setAttribute(Qt.WidgetAttribute.WA\_TranslucentBackground)

self.setWindowFlags(Qt.WindowType.FramelessWindowHint | Qt.WindowType.Window)

self.setObjectName("MainWindow")

self.main\_layout = QVBoxLayout()

self.main\_layout.setSpacing(0)

self.main\_layout.setContentsMargins(0, 0, 0, 0)

self.setLayout(self.main\_layout)

self.title\_bar = QWidget()

self.title\_bar.setObjectName("TitleBar")

self.title\_bar.setFixedHeight(40)

title\_layout = QHBoxLayout()

title\_layout.setContentsMargins(0, 0, 0, 0)

self.title\_bar.setLayout(title\_layout)

self.title\_label = QLabel("TLS Чат")

title\_layout.addWidget(self.title\_label)

title\_layout.addStretch()

self.minimize\_btn = QPushButton("🗕")

self.minimize\_btn.setObjectName("TitleBarButton")

self.minimize\_btn.clicked.connect(self.showMinimized)

title\_layout.addWidget(self.minimize\_btn)

self.maximize\_btn = QPushButton("🗖")

self.maximize\_btn.setObjectName("TitleBarButton")

self.maximize\_btn.clicked.connect(self.toggle\_maximize)

title\_layout.addWidget(self.maximize\_btn)

self.close\_btn = QPushButton("🗙")

self.close\_btn.setObjectName("TitleBarButton")

self.close\_btn.setObjectName("CloseButton")

self.close\_btn.clicked.connect(self.close)

title\_layout.addWidget(self.close\_btn)

self.main\_layout.addWidget(self.title\_bar)

self.content\_widget = QWidget()

self.content\_widget.setObjectName("Content")

self.content\_layout = QHBoxLayout()

self.content\_layout.setContentsMargins(0, 0, 0, 0)

self.content\_widget.setLayout(self.content\_layout)

self.main\_layout.addWidget(self.content\_widget)

self.init\_login\_form()

self.init\_register\_form()

self.init\_chat\_form()

self.show\_login\_form()

logging.debug(f"Версия Python: {sys.version}, Версия OpenSSL: {ssl.OPENSSL\_VERSION}")

def mousePressEvent(self, event):

if event.button() == Qt.MouseButton.LeftButton and self.title\_bar.underMouse():

self.old\_pos = event.globalPosition().toPoint()

super().mousePressEvent(event)

def mouseMoveEvent(self, event):

if self.old\_pos is not None:

delta = event.globalPosition().toPoint() - self.old\_pos

self.move(self.pos() + delta)

self.old\_pos = event.globalPosition().toPoint()

super().mouseMoveEvent(event)

def mouseReleaseEvent(self, event):

self.old\_pos = None

super().mouseReleaseEvent(event)

def toggle\_maximize(self):

if self.is\_maximized:

self.showNormal()

self.maximize\_btn.setText("🗖")

self.is\_maximized = False

else:

self.showMaximized()

self.maximize\_btn.setText("🗗")

self.is\_maximized = True

def closeEvent(self, event):

self.logout()

event.accept()

def init\_login\_form(self):

self.login\_form = QWidget()

layout = QVBoxLayout()

layout.setContentsMargins(10, 10, 10, 10)

self.login\_form.setLayout(layout)

layout.addWidget(QLabel("Логин:"))

self.login\_entry = QLineEdit()

layout.addWidget(self.login\_entry)

layout.addWidget(QLabel("Пароль:"))

self.password\_entry = QLineEdit()

self.password\_entry.setEchoMode(QLineEdit.EchoMode.Password)

layout.addWidget(self.password\_entry)

btn\_layout = QHBoxLayout()

login\_btn = QPushButton("Вход")

login\_btn.clicked.connect(self.login)

register\_btn = QPushButton("Регистрация")

register\_btn.clicked.connect(self.show\_register\_form)

btn\_layout.addWidget(login\_btn)

btn\_layout.addWidget(register\_btn)

layout.addLayout(btn\_layout)

layout.addStretch()

def init\_register\_form(self):

self.register\_form = QWidget()

layout = QVBoxLayout()

layout.setContentsMargins(10, 10, 10, 10)

self.register\_form.setLayout(layout)

layout.addWidget(QLabel("Логин:"))

self.reg\_login\_entry = QLineEdit()

layout.addWidget(self.reg\_login\_entry)

layout.addWidget(QLabel("Пароль:"))

self.reg\_password\_entry = QLineEdit()

self.reg\_password\_entry.setEchoMode(QLineEdit.EchoMode.Password)

layout.addWidget(self.reg\_password\_entry)

layout.addWidget(QLabel("Повторите пароль:"))

self.reg\_password2\_entry = QLineEdit()

self.reg\_password2\_entry.setEchoMode(QLineEdit.EchoMode.Password)

layout.addWidget(self.reg\_password2\_entry)

btn\_layout = QHBoxLayout()

register\_btn = QPushButton("Зарегистрироваться")

register\_btn.clicked.connect(self.register)

back\_btn = QPushButton("Назад")

back\_btn.clicked.connect(self.show\_login\_form)

btn\_layout.addWidget(register\_btn)

btn\_layout.addWidget(back\_btn)

layout.addLayout(btn\_layout)

layout.addStretch()

def init\_chat\_form(self):

self.chat\_form = QWidget()

layout = QHBoxLayout()

layout.setContentsMargins(10, 10, 10, 10)

self.chat\_form.setLayout(layout)

left = QVBoxLayout()

self.message\_text = QTextEdit()

self.message\_text.setReadOnly(True)

left.addWidget(QLabel("Сообщения:"))

left.addWidget(self.message\_text)

input\_layout = QHBoxLayout()

input\_layout.addWidget(QLabel("Ввод:"))

self.input\_entry = QLineEdit()

self.input\_entry.returnPressed.connect(self.send\_message)

input\_layout.addWidget(self.input\_entry)

left.addLayout(input\_layout)

cert\_btn = QPushButton("Сгенерировать сертификат")

cert\_btn.clicked.connect(self.generate\_certificate)

left.addWidget(cert\_btn)

left\_widget = QWidget()

left\_widget.setLayout(left)

layout.addWidget(left\_widget, stretch=3)

right = QVBoxLayout()

right.addWidget(QLabel("Пользователи онлайн:"))

self.user\_list = QListWidget()

right.addWidget(self.user\_list)

send\_btn = QPushButton("Отправить")

send\_btn.clicked.connect(self.send\_message)

logout\_btn = QPushButton("Выйти")

logout\_btn.clicked.connect(self.logout)

right.addWidget(send\_btn)

right.addWidget(logout\_btn)

right\_widget = QWidget()

right\_widget.setLayout(right)

layout.addWidget(right\_widget, stretch=1)

def clear\_main\_layout(self):

while self.content\_layout.count():

widget = self.content\_layout.takeAt(0).widget()

if widget:

widget.setParent(None)

def show\_login\_form(self):

self.clear\_main\_layout()

self.content\_layout.addWidget(self.login\_form)

def show\_register\_form(self):

self.clear\_main\_layout()

self.content\_layout.addWidget(self.register\_form)

def show\_chat\_form(self):

self.clear\_main\_layout()

self.content\_layout.addWidget(self.chat\_form)

def update\_message\_text(self, message):

if message:

self.message\_text.append(message)

def update\_user\_list(self, users):

self.user\_list.clear()

for user in users:

if user and user != self.current\_login:

self.user\_list.addItem(user)

def create\_ssl\_context(self):

if not self.cert\_path or not os.path.exists(self.cert\_path):

raise FileNotFoundError("Файл server.crt не найден.")

context = ssl.SSLContext(ssl.PROTOCOL\_TLS\_CLIENT)

context.minimum\_version = ssl.TLSVersion.TLSv1\_3

context.maximum\_version = ssl.TLSVersion.TLSv1\_3

context.set\_ciphers('DEFAULT@SECLEVEL=2')

context.load\_verify\_locations(self.cert\_path)

context.verify\_mode = ssl.CERT\_REQUIRED

logging.info(f"Создан SSL-контекст: min={context.minimum\_version}, max={context.maximum\_version}")

logging.debug(f"Доступные шифры: {context.get\_ciphers()}")

return context

def generate\_certificate(self):

if not self.current\_login:

QMessageBox.critical(self, "Ошибка", "Сначала войдите в чат!")

return

try:

key = rsa.generate\_private\_key(

public\_exponent=65537,

key\_size=2048

)

subject = x509.Name([

x509.NameAttribute(NameOID.COMMON\_NAME, self.current\_login)

])

cert = x509.CertificateBuilder().subject\_name(

subject

).issuer\_name(

subject

).public\_key(

key.public\_key()

).serial\_number(

x509.random\_serial\_number()

).not\_valid\_before(

datetime.datetime.utcnow()

).not\_valid\_after(

datetime.datetime.utcnow() + datetime.timedelta(days=365)

).add\_extension(

x509.SubjectAlternativeName([x509.DNSName(self.current\_login)]),

critical=False

).sign(key, hashes.SHA256())

cert\_file = f"{self.current\_login}\_client.crt"

with open(cert\_file, "wb") as f:

f.write(cert.public\_bytes(serialization.Encoding.PEM))

cert\_data = base64.b64encode(cert.public\_bytes(serialization.Encoding.PEM)).decode()

context = self.create\_ssl\_context()

temp\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

temp\_socket = context.wrap\_socket(temp\_socket, server\_hostname="127.0.0.1")

temp\_socket.connect(('127.0.0.1', 443))

temp\_socket.send(f"UPLOAD\_CERT:{self.current\_login}:{cert\_data}".encode())

response = temp\_socket.recv(1024).decode()

temp\_socket.close()

if response == "CERT\_SUCCESS":

QMessageBox.information(self, "Успех", f"Сертификат сохранен как {cert\_file}")

logging.info(f"Сертификат сгенерирован и отправлен для {self.current\_login}")

else:

QMessageBox.critical(self, "Ошибка", "Не удалось загрузить сертификат на сервер")

logging.error(f"Ошибка загрузки сертификата на сервер для {self.current\_login}")

except Exception as e:

QMessageBox.critical(self, "Ошибка", f"Не удалось сгенерировать сертификат: {e}")

logging.error(f"Ошибка генерации сертификата: {e}")

def login(self):

if not self.cert\_path:

QMessageBox.critical(self, "Ошибка", "Файл server.crt не найден в текущей директории!")

return

login = self.login\_entry.text()

password = self.password\_entry.text()

if not login or not password:

QMessageBox.critical(self, "Ошибка", "Введите логин и пароль!")

return

try:

context = self.create\_ssl\_context()

self.client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

self.client\_socket = context.wrap\_socket(self.client\_socket, server\_hostname="127.0.0.1")

self.client\_socket.connect(('127.0.0.1', 443))

logging.info(f"Фактическая версия TLS после подключения: {self.client\_socket.version()}")

logging.debug(f"Шифры, использованные в соединении: {self.client\_socket.cipher()}")

self.client\_socket.send(f"LOGIN:{login}:{password}".encode())

response = self.client\_socket.recv(1024).decode()

if response == "AUTH\_SUCCESS":

self.current\_login = login

self.current\_password = password

self.message\_text.append("Успешное подключение!")

self.show\_chat\_form()

threading.Thread(target=self.receive\_messages, daemon=True).start()

self.login\_entry.clear()

self.password\_entry.clear()

logging.info(f"Успешный вход: {login}")

else:

QMessageBox.critical(self, "Ошибка", "Неверный логин или пароль")

self.client\_socket.close()

self.client\_socket = None

logging.warning(f"Неудачная аутентификация для {login}")

except Exception as e:

QMessageBox.critical(self, "Ошибка", f"Не удалось подключиться: {e}")

self.client\_socket = None

logging.error(f"Ошибка входа: {e}")

def register(self):

login = self.reg\_login\_entry.text()

password = self.reg\_password\_entry.text()

password2 = self.reg\_password2\_entry.text()

if not login or not password or not password2:

QMessageBox.critical(self, "Ошибка", "Заполните все поля!")

return

if password != password2:

QMessageBox.critical(self, "Ошибка", "Пароли не совпадают!")

return

if not re.match(r"^[a-zA-Z0-9\_]{3,}$", login):

QMessageBox.critical(self, "Ошибка", "Логин должен быть минимум 3 символа и содержать только буквы, цифры или \_")

return

if len(password) < 6:

QMessageBox.critical(self, "Ошибка", "Пароль должен быть минимум 6 символов!")

return

try:

context = self.create\_ssl\_context()

temp\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

temp\_socket = context.wrap\_socket(temp\_socket, server\_hostname="127.0.0.1")

temp\_socket.connect(('127.0.0.1', 443))

logging.debug(f"Регистрация: TLS версия: {temp\_socket.version()}, шифр: {temp\_socket.cipher()}")

temp\_socket.send(f"REGISTER:{login}:{password}".encode())

response = temp\_socket.recv(1024).decode()

temp\_socket.close()

if response == "REG\_SUCCESS":

QMessageBox.information(self, "Успех", "Регистрация успешна! Войдите с новым логином.")

self.reg\_login\_entry.clear()

self.reg\_password\_entry.clear()

self.reg\_password2\_entry.clear()

self.show\_login\_form()

logging.info(f"Успешная регистрация: {login}")

else:

QMessageBox.critical(self, "Ошибка", "Логин уже занят!")

logging.warning(f"Неудачная регистрация: {login}")

except Exception as e:

QMessageBox.critical(self, "Ошибка", f"Не удалось зарегистрироваться: {e}")

logging.error(f"Ошибка регистрации: {e}")

def send\_message(self):

if not self.client\_socket:

QMessageBox.critical(self, "Ошибка", "Сначала подключитесь к серверу!")

return

message = self.input\_entry.text()

if message:

try:

self.client\_socket.send(b"")

logging.debug(f"Отправка сообщения: {message}")

self.client\_socket.send(message.encode())

self.message\_text.append(f"Вы: {message}")

self.input\_entry.clear()

logging.info(f"Сообщение отправлено: {message}")

except (BrokenPipeError, ConnectionError, OSError) as e:

QMessageBox.critical(self, "Ошибка", f"Соединение разорвано: {e}")

self.client\_socket = None

self.show\_login\_form()

logging.error(f"Ошибка отправки сообщения: {e}")

except Exception as e:

QMessageBox.critical(self, "Ошибка", f"Не удалось отправить сообщение: {e}")

self.client\_socket = None

self.show\_login\_form()

logging.error(f"Ошибка отправки сообщения: {e}")

def receive\_messages(self):

while self.client\_socket:

try:

message = self.client\_socket.recv(1024).decode()

if not message:

logging.warning("Пустое сообщение, разрыв соединения")

break

logging.debug(f"Получено сообщение: {message}")

if message.startswith("USER\_LIST:"):

users = message[len("USER\_LIST:"):].split(",")

self.comm.update\_users.emit(users)

else:

self.comm.update\_text.emit(message)

except (ConnectionError, OSError) as e:

logging.error(f"Ошибка соединения при получении сообщения: {e}")

break

except Exception as e:

logging.error(f"Ошибка при получении сообщения: {e}")

break

if self.client\_socket:

try:

self.client\_socket.close()

logging.info("Сокет закрыт в receive\_messages")

except Exception as e:

logging.error(f"Ошибка при закрытии сокета: {e}")

self.client\_socket = None

self.comm.update\_text.emit("Соединение разорвано")

self.show\_login\_form()

logging.info("Соединение с сервером разорвано")

def logout(self):

if self.client\_socket:

try:

self.client\_socket.close()

logging.info("Сокет закрыт при выходе")

except Exception as e:

logging.error(f"Ошибка при закрытии сокета: {e}")

self.client\_socket = None

self.message\_text.clear()

self.user\_list.clear()

self.show\_login\_form()

logging.info(f"Выход из чата: {self.current\_login}")

self.current\_login = None

self.current\_password = None

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

client = TLSChatClient()

client.show()

sys.exit(app.exec())