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

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/
2. Welcome to pyca/cryptography
   Режим доступа: https://cryptography.io/en/latest/
   Qt for Python
   Режим доступа: https://doc.qt.io/qtforpython-6/index.html
3. SQLite
   Режим доступа: https://www.sqlite.org/docs.html
4. The Transport Layer Security (TLS) Protocol Version 1.3
   Режим доступа: https://datatracker.ietf.org/doc/html/rfc8446
5. Руководство по выживанию — TLS/SSL и сертификаты SSL (X.509)
   Режим доступа:
   https://www.opennet.ru/docs/RUS/ldap_apacheds/tech/ssl.html
6. OpenSSL Documentation.
   Режим доступа: https://docs.openssl.org/master/
7. bcrypt 4.3.0.
   Режим доступа: https://pypi.org/project/bcrypt/
8. 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())
```