

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР  
ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
КАФЕДРА ПРОГРАММНОЙ ИНЖЕНЕРИИ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
к курсовому проекту по дисциплине  
«Безопасность программ и данных»  
на тему: «Создание безопасного протокола обмена данными между  
клиентом и сервером для системы SILUR»

Руководитель:

\_\_\_\_\_ кафедры

\_\_\_\_\_

Выполнил:

ст. гр. ПИ–186

Моргунов А.Г.

ДОНЕЦК – 2021

## РЕФЕРАТ

Пояснительная записка к курсовому проекту содержит: 50 страницы, 41 рисунок, 11 источников, 4 приложения.

Цель работы – закрепить полученные знания в области криптографических алгоритмов, а также спроектировать и реализовать клиентское приложение для взаимодействия с сервером агрегации. Для достижения поставленной цели курсового проекта необходимо:

- проанализировать возможности Django;
- разработать комплекс модулей;
- осуществить техническое и рабочее проектирование сайта;
- реализовать эмуляторы, а также провести тестирования спроектированных программ;
- реализовать авторский протокол передачи данных.

Методы исследования – научные источники по агрегационным системам, сокет, методы, алгоритмы взаимодействия с сервером, возможности Django, криптографические алгоритмы.

Объект исследования – клиентское приложение – сайт для агрегационной системы.

Результаты работы – сайт, написанный при помощи framework Django на Python с поддержкой протоколов передачи данных сервера.

DJANGO, PYTHON, АРХИТЕКТУРА, САЙТ, SOCKET, АГРЕГАТОР, БАЗА ДАННЫХ, AES, RSA, Deffie-Hellman, КРИПТОГРАФИ

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	5
1 АНАЛИЗ СУЩЕСТВУЮЩИХ ПРОТОКОЛОВ .....	6
2 ПРОЕКТИРОВАНИЕ Клиентского ПРИЛОЖЕНИЯ.....	8
1.1 Web-клиент .....	8
1.2 Архитектура проекта .....	8
1.3 Переменные сессии.....	11
1.4 Основные требования к безопасности.....	11
3 ТЕОРИТИЧЕСКОЕ ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ КРИПТОГРАФИЧЕСКИХ АЛГОРИТМОВ В СИСТЕМЕ АГРЕГАЦИИ..	12
3.1 Алгоритм обмена ключом.....	12
3.2 Симметричный алгоритм шифрования AES в режиме ECB .....	12
3.3 Ассиметричный алгоритм шифрования RSA в режиме ОАЕР....	13
3.4 Описание функции хеширования SHA-256 .....	16
4 РАЗРАБОТКА web-клиента.....	17
4.1 Выбор средств реализации. Обоснование выбора .....	17
4.2 Описание библиотек .....	17
4.2.1 cryptography .....	17
4.3 Описание функций для криптографии .....	17
4.3.1 Реализация функций для работы AES .....	17
4.3.2 Реализация функций для работы RSA.....	18
4.3.4 Реализация безопасности на уровне сокетов .....	21
4.3.4.1 Класс Socket.....	21
4.3.4.2 Класс AESSocket .....	23
4.3.4.3 Класс RSASocketWriter .....	24
5 ОПИСАНИЕ АВТОРСКОГО ПРОТОКОЛА.....	25
5.1 Взаимодействие с сервером.....	25
5.2 Протокол передачи данных .....	25
5.2.1 Инициализация.....	26

5.2.2 Аутентификация .....	27
5.2.3 Достоинства и недостатки авторского протокола.....	32
ВЫВОДЫ.....	34
ПЕРЕЧЕНЬ ССЫЛОК .....	35
ПРИЛОЖЕНИЕ А. ТЕХНИЧЕСКОЕ ЗАДАНИЕ .....	36
ПРИЛОЖЕНИЕ Б. ЭКРАННЫЕ ФОРМЫ.....	39
ПРИЛОЖЕНИЕ В. ЛИСТИНГ КОДА.....	44

## ВВЕДЕНИЕ

Веб-сайт – совокупность файлов, документов, отраженных при помощи языка программирования таким образом, чтобы их видели пользователи сети Интернет. Другими словами, сайты включают в себя любую текстовую, графическую, аудио- или видеовизуальную информацию, собранную на странице или нескольких страницах.

Веб ресурсы в современном мире являются чрезвычайно важным аспектом информационной сферы. Многие разработчики и пользователи стали отказываться от десктопных приложений в пользу веб-приложений из-за того, что они доступны с любого устройства без установки.

В первом разделе описывается уже существующие протоколы.

Во втором разделе описывается архитектура проекта, классы.

В третьем разделе теоретическое описание используемых криптографических алгоритмов в системе агрегации.

В четвертом разделе описывается проектирование системы.

Целью курсового проекта является закрепление знаний, проектирования, реализация и понимание общей концепции работы агрегаторов сообщений в мессенджере Telegram.

## 1 АНАЛИЗ СУЩЕСТВУЮЩИХ ПРОТОКОЛОВ

Существующие протоколы для безопасной передачи данных:

SSL – разработан компанией Netscape Communications. Это криптографический протокол, который был опубликован в 1995 году. Во время аутентификации используется ассиметричные криптографические алгоритмы, а для обмена информацией симметричные. В 2015 году признан устаревшим [1].

TLS – развитием стандарта занимается IETF. TLS основан на спецификации протокола SSL 3.0, из-за чего имеет схожие подходы. Является намного более безопасным протоколом по сравнению с SSL. Обеспечена защита от многих криптографических атак [2].

Tox – протокол, который обеспечивает децентрализованный обмен сообщениями, а также осуществление аудио- и видеосвязи, в интернете с использованием ассиметричного шифрования. Основной особенностью является то, что при использовании протокола идентификатор пользователя создается локально, что позволяет обходиться без регистрации [3].

Signal – разработан компанией Open Whisper Systems. Обеспечивает сквозное шифрование при обмене сообщениями, аудио- и видеозвонках. Этот протокол поддерживается в приложениях: WhatsApp, Facebook Messenger, Skype [4].

Echo – это протокол, который предназначен для передачи сообщений в зашифрованном виде. Особенностью является возможность выбора из трех режимов передачи сообщения, которые могут использоваться в зависимости от нужд пользователя [5].

Bitmessage – это мессенджер, который использует протокол, использующий децентрализованную сеть. Особенностью является то, что зашифрованные сообщения рассылаются всем, причем сообщение не содержит адреса получателя. По этой причине все пользователи пытаются расшифровать все приходящие сообщения, но только получатель имеет ключ, которым можно расшифровать сообщение. Таким образом сохраняется анонимность пользователей [6].

MTProto – разработан компанией Telegram. Это протокол, который используется для обмена сообщениями в мессенджере Telegram. Особенностью данного алгоритма является комбинированное использование существующих криптографических алгоритмов и хеширования. Также опциональной возможностью является создание секретных чатов. При передаче сообщений в секретных чатах, не принимает участия сервер Telegram, все хранится на устройствах пользователей [7].

## 2 ПРОЕКТИРОВАНИЕ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ

### 2.1 Web-клиент

Приложение SILUR агрегирует сообщения из каналов-источников Telegram в другие каналы-приёмники.

В рамках данного курсового проекта разрабатывалась клиентская часть системы SILUR. Которая выполняет следующие работы: предоставляет пользователю интерфейс для взаимодействия с сервером, сохраняет пользовательскую сессию, получает информацию с сервера и отображает ее пользователю.

Для настройки агрегатора клиент использует API сервера, которое предоставляется для клиентов. Клиенты могут создавать новые каналы-источники, каналы-приемники и устанавливать отношение между ними.

Для взаимодействия с сервером в рамках курсового проекта разрабатывался авторский протокол.

### 2.2 Архитектура проекта

Архитектура проекта обусловлена его направленностью, а именно web направленностью. Для разработки был выбран фреймворк Django, что и определило основную архитектуру проекта.

Django использует архитектуру MVT — Model-View-Template. Model — это модель, которая представляет данные. При разработке в роли модели выступает сервер приложения. View — это представления, которые взаимодействуют с данными. Template — это шаблоны, с помощью которых генерируется пользовательский интерфейс (см. рис. ).



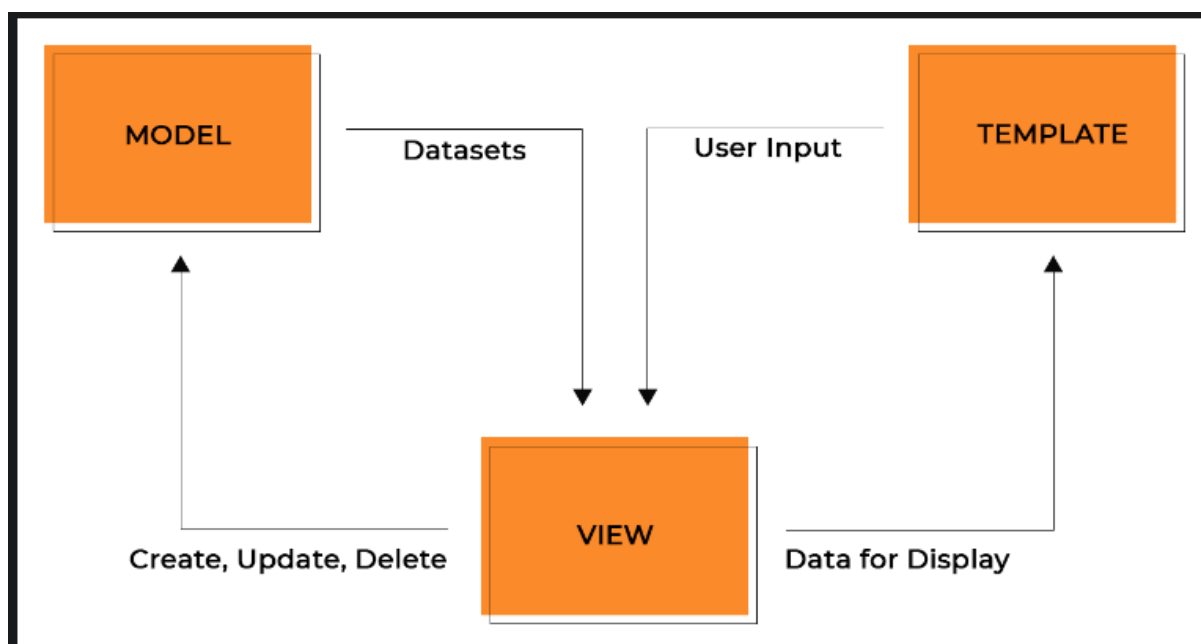


Рисунок 2.1 — Схема архитектуры MVT в Django

В программе есть несколько модулей:

- Модуль Django.
- Модуль сокетов.

Система содержит 5 основных модулей:

- 1) Модуль Django – обеспечивает основную функциональность клиента. Обрабатывает пользовательский ввод, информацию с сервера, обращается к сокетам для передачи данных между клиентом и сервером;
- 2) Модуль сокетов – предназначен для передачи данных между клиентом и сервером. Также шифрует данные, которые пересылаются.

Диаграмма классов для модуля сокетов приведена на рисунке .

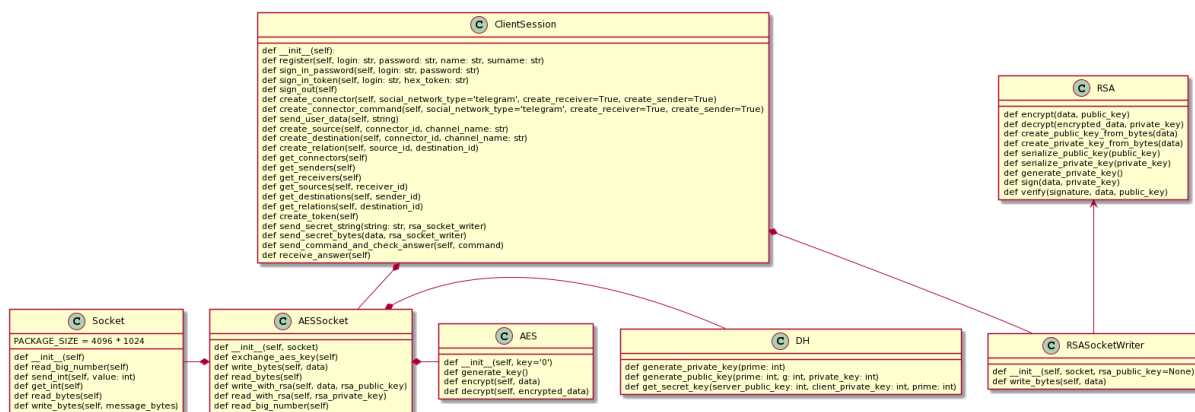


Рисунок 2.2 – Диаграмма классов

Модуль сокетов содержит:

- 1) AES – Реализация алгоритма AES
- 2) RSA- Реализация алгоритма RSA
- 3) DH-Реализация алгоритма DH
- 4) Socket-Рализация взаимодействия между сервером и клиентом
- 5) AESSocket- Сокеты с шифрованием AES
- 6) RSASokcetWriter- Сокеты с шифрованием RSA
- 7) ClientSession — Класс, который отвечает за создание сессии.

Главный класс, который содержит все остальные.

### 2.3 Переменные сессии

Для хранения данных приложение использует один из механизмов Django, а именно сессии. Они реализованы с помощью SQLite. Это переменные, которые хранятся в БД, при этом для каждой пользовательской сессии эти переменные принимают разные значения.

Таким способом сохраняется подписанный токен пользователя и его логин, что позволяет реализовать авторизацию пользователя без пароля.

### 2.4 Основные требования к безопасности

Клиент должен обеспечивать конфиденциальность и целостность данных пользователей при передаче на сервер. Для реализации требований безопасности в курсовом проекте был разработан авторский протокол передачи данных. Протокол включает в себе стадии инициализации, идентификации, авторизации и дальнейшей передачи данных.

Безопасность данных обеспечивается за счет шифрования данных при передаче их на сервер. Особо важные данные шифруются несколькими алгоритмами шифрования, что позволяет надежно обезопасить их. Также на каждую сессию генерируются новые сессионные ключи, которые обеспечивают безопасность основного количества данных, даже если один из сессионных ключей будет взломан.

### 3 ТЕОРИТИЧЕСКОЕ ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ КРИПТОГРАФИЧЕСКИХ АЛГОРИТМОВ В СИСТЕМЕ АГРЕГАЦИИ

#### 3.1 Алгоритм обмена ключом

Для передачи ключа используется алгоритм Deffie-Hellman, который подразумевает, что среда передачи данных является небезопасной и данные могут быть перехвачены.

Алгоритм:

Сервер генерирует  $g$ ,  $a$ ,  $p$  и вычисляет  $A$ .

$$A = g^a \bmod p$$

Далее пересылает клиенту  $A$ ,  $g$ ,  $p$ .

Клиент создает компоненту  $B$  и отправляет её серверу.

$$B = g^b \bmod p$$

Далее клиент и сервер получают один и тот же ключ.

$$A^b \bmod p = B^a \bmod p = g^{ab} \bmod p = K$$

Все данные, передаваемые по сети, являются публичными и по ним невозможно получить ключ в быстрый период времени. В дальнейшем данный ключ  $K$  используется в алгоритме AES.

#### 3.2 Симметричный алгоритм шифрования AES в режиме ECB

AES – симметричный алгоритм шифрования данных (использует общий ключ). В режиме ECB не использует вектор инициализаций, такое решение было принято для большей производительности при повышении криптостойкости. Т.к. сам по себе вектор инициализации не привносит большей криптостойкости, т.к. передается в открытом виде, в рамках протокола было решено использовать слоистую архитектуру шифрования. Архитектура подразумевает поверх безопасного канала передачи данных повторно обмениваться ключами и использовать новые ключи. Такой

подход кратно увеличивает криптостойкости и повышает производительность.

В рамках курсового проекта использовался AES размером 128 бит, что говорит о том, что алгоритм использует 10 раундов. Ключ специально был выбран минимальным для повышения быстродействия, а повышение криптостойкости решается слоёной архитектурой. В целом такой подход повышает латентность системы не теряя криптостойкости.

Алгоритм padding – PKCS, который дополняет недостающие байты символом, код которого – количество недостающих байтов. Если количество недостающих байтов  $\neq 0$ , то дополняются 16 байт padding.

### 3.3 Ассиметричный алгоритм шифрования RSA в режиме OAEP

RSA – ассиметричный алгоритм шифрования, является наиболее распространенным алгоритмом ассиметричного шифрования. Преимущества данного алгоритма – это криптостойкость основанная на теореме Эйлера и малой теореме Ферма.

Криптографические системы с открытым ключом используют так называемые односторонние функции, которые обладают следующим свойством:

Если известно  $x$ , то  $f(x)$ , вычислить просто.

Если известно  $y = f(x)$ , то для вычисления  $x$  не простого (эффективного) пути.

Под односторонностью понимается не математически доказанная однонаправленность, а практическая невозможность вычислить обратное значение, используя современные вычислительные средства, за обозримый интервал времени.

В основу криптографической системы с открытым ключом RSA положена сложность задачи факторизации произведения двух больших простых чисел. Для шифрования используется операция возведения в

степень по модулю большого числа. Для дешифрования (обратной операции) за разумное время необходимо уметь вычислять функцию Эйлера от данного большого числа, для чего необходимо знать разложение числа на простые множители.

Алгоритм генерации числа:

1. Выбирается 2 простых числа  $p$  и  $q$ .
2. Вычисляется  $n = p * q$ .
3. Вычисляется функция Эйлера от  $n$ .  $\varphi(n) = (p - 1) * (q - 1)$
4. Выбирается целое число  $e$  ( $1 < e < \varphi(n)$ ),  $e$  и  $\varphi(n)$

взаимнопростые.

5. Выбирается число  $d$ , мультипликативно обратное к числу  $e$  по модулю  $\varphi(n)$ , т.е.  $e * d \equiv 1 \bmod \varphi(n)$ .

6. Пара  $(e, n)$  – публичный ключ.

7. Пара  $(d, n)$  – приватный ключ.

Шифрование и расшифрование производится следующим образом.

Сообщение – целые числа от  $0$  до  $n - 1$  взаимнопростые с  $n$ , т. е.  $p$  и  $q$ .

Пусть  $m$  – сообщение, тогда шифрование производится следующим образом:  $c = m^e \bmod n$ . Расшифрование:  $c^d \bmod n = m^{ed} \bmod n = m^1 \bmod n = m$ .

Для проверки того, что расшифрованное сообщение не подменили существует алгоритм оптимального асимметричного шифрования с дополнением (ОАЕР) (см. рис. 3.1).

Для работы ОАЕР необходима однонаправленная hash функция (в рамках курсового проекта была выбрана SHA-256).

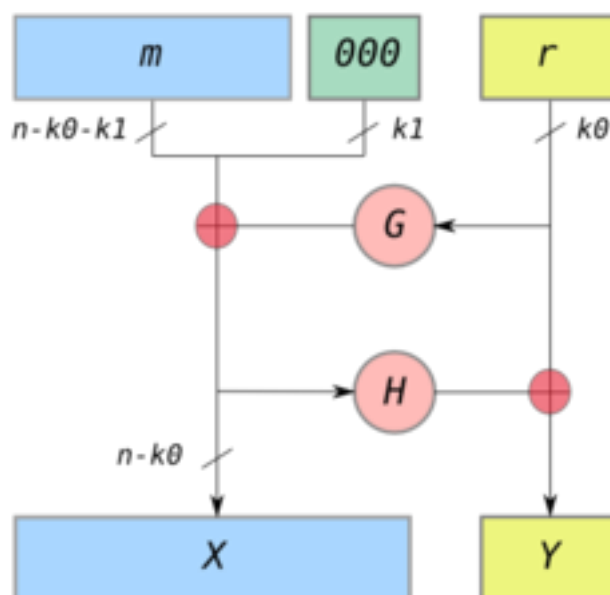


Рисунок 3.1 – Схема ОАЕР

$n$  – число бит для шифрования, длина  $m$ .

$k_1$  – число нулей.

$r$  – случайное число.

$k_0$  – длина  $r$  в битах.

$G, H$  – однонаправленные hash функции, в рамках курсового проекта – SHA-256.

Шифрование: к  $m$  дописывается  $k_1$  нулей. Генерируется случайное  $r$  длины  $k_0$ .  $G$  расширяет  $r$  до  $n - k_0$  размера. Вычисляется  $X = m00 \dots 0 \oplus G(r)$ .  $H$  ужимает  $n - k_0$  бит  $X$  до  $k_0$  бит.  $Y = r \oplus H(X)$ . Зашифрованный текст =  $X \vee Y$ .

После расшифрования RSA проверяем целостность данных следующим образом: восстанавливается случайная строка  $r = Y \oplus H(X)$ . Восстанавливается  $m00 \dots 0 = X \oplus G(r)$ . Последние  $k_1$  битов должны быть нулями, в противном случае расшифрованное сообщение не валидно (подменено или использовался неправильный ключ).

### 3.4 Описание функции хеширования SHA-256

Secure Hash Algorithm Version 2 — безопасный алгоритм хеширования, версия 2) — семейство криптографических алгоритмов — однонаправленных хеш-функций, включающее в себя алгоритмы SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/256 и SHA-512/224 [11].

### 3.5 Ассиметричный алгоритм шифрования для ЭЦП RSA стандарт PKCS1v15 вместе с SHA-256

Хеш-функции семейства SHA-2 построены на основе структуры Меркла — Дамгора.

Исходное сообщение после дополнения разбивается на блоки, каждый блок — на 16 слов. Алгоритм пропускает каждый блок сообщения через цикл с 64 или 80 итерациями (раундами). На каждой итерации 2 слова преобразуются, функцию преобразования задают остальные слова. Результаты обработки каждого блока складываются, сумма является значением хеш-функции. Тем не менее, инициализация внутреннего состояния производится результатом обработки предыдущего блока. Поэтому независимо обрабатывать блоки и складывать результаты нельзя.



## 4 РАЗРАБОТКА WEB-КЛИЕНТА

### 4.1 Выбор средств реализации. Обоснование выбора

Для написания клиентской части использовался язык Python с фреймворком Django. Выбор такой комбинации средств разработки обусловлен гибкостью языка, наличием большого количества пользовательских библиотек, которые могут решить любую тривиальную задачу, что должно ускорить разработку.

Также нужно отметить Django. Это чрезвычайно мощный и гибкий инструмент, который исключительно хорошо подходит под разрабатываемое приложение.

### 4.2 Описание библиотек

#### 4.2.1 cryptography

Библиотека для криптографии cryptography предоставляет все необходимые возможности для шифрования данных. Библиотека написана на языке Python, что позволяет разрабатывать приложение с большей скоростью. В библиотеке содержатся все необходимые криптографические алгоритмы. Существует отличная документация с множеством примеров.

### 4.3 Описание функций для криптографии

#### 4.3.1 Реализация функций для работы AES

На рисунке 4.1 предоставлена реализация шифрования AES.

```
def encrypt(self, data):
    encryptor = self.cipher.encryptor()
    aligned_data = align_to_size(data)
    encrypted_data = encryptor.update(aligned_data)
    encryptor.finalize()
    return encrypted_data
```

Рисунок 4.1 – Шифрование AES

Как можно видеть на рисунке 4.1 AES используется в ECB моде.

Расшифрование представлено на рисунке 4.2.

```
def decrypt(self, encrypted_data):
    decryptor = self.cipher.decryptor()
    decrypted_data = decryptor.update(encrypted_data)
    decryptor.finalize()
    decrypted_data_without_alignment = release_align(decrypted_data)
    return decrypted_data_without_alignment
```

Рисунок 4.2 – Расшифрование AES

#### 4.3.2 Реализация функций для работы RSA

```
@staticmethod
def generate_private_key():
    return rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
    )
```

Рисунок 4.3 – Генерация RSA-ключей

```

@staticmethod
def encrypt(data, public_key):
    return public_key.encrypt(
        data,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )

```

Рисунок 4.4 – Шифрование RSA

```

@staticmethod
def decrypt(encrypted_data, private_key):
    return private_key.decrypt(
        encrypted_data,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )

```

Рисунок 4.5 – Расшифрование RSA

```

@staticmethod
def create_public_key_from_bytes(data):
    return serialization.load_pem_public_key(data)

@staticmethod
def create_private_key_from_bytes(data):
    return serialization.load_pem_private_key(data, password=None)

```

Рисунок 4.6 – Загрузка RSA-ключей

```

@staticmethod
def serialize_public_key(public_key):
    return public_key.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo
    )

@staticmethod
def serialize_private_key(private_key):
    return private_key.private_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PrivateFormat.TraditionalOpenSSL,
        encryption_algorithm=serialization.NoEncryption()
    )

```

Рисунок 4.7 – Сохранение RSA-ключей

```

@staticmethod
def sign(data, private_key):
    return private_key.sign(
        data,
        padding.PKCS1v15(),
        hashes.SHA256()
    )

```

Рисунок 4.8 – Получение подписи данных

```

@staticmethod
def verify(signature, data, public_key):
    try:
        public_key.verify(
            signature,
            data,
            padding.PKCS1v15(),
            hashes.SHA256()
        )
        return True
    except Exception:
        return False

```

Рисунок 4.9 – Проверка подписи данных

#### 4.3.4 Реализация безопасности на уровне сокетов

##### 4.3.4.1 Класс Socket

Для создания прозрачности использования криптографии (где это возможно) было принято решение реализовать криптографические алгоритмы на уровне сокетов.

Первый шаг заключается в создании класса Socket для передачи данных по сети, не задумываясь о внутреннем устройстве сокетов.

Реализация write\_bytes приведена на рисунке 4.10

```

def write_bytes(self, message_bytes) -> None:
    self.send_int(len(message_bytes))
    size = len(message_bytes)
    sent_bytes = 0
    while size != sent_bytes:
        sent_bytes += self.socket.send(message_bytes[sent_bytes:])
    # print(f"Sent: {sent_bytes}")

```

Рисунок 4.10 – Реализация метода Send

```
def send_int(self, value: int) -> None:
    self.socket.send(value.to_bytes(length=4, byteorder='big'))
```

Рисунок 4.11 – Реализация метода send\_int

Алгоритм передачи данных, следующий: вычисление размера сообщения, перевод размера в сетевой порядок байт, отправка размера, отправка сообщения.

Метод read\_bytes симметричен (см. рис. 4.12).

```
def read_bytes(self) -> bytes:
    size = self.get_int()
    storage = bytearray()
    while len(storage) != size:
        storage += self.socket.recv(min(self.PACKAGE_SIZE, size - len(storage)))
    # print("Read")
    return bytes(storage)
```

Рисунок 4.12 – Реализация метода read\_bytes

```
def get_int(self) -> int:
    int_size = 4
    result = self.socket.recv(int_size)
    result = int.from_bytes(result, byteorder='big')
    return result
```

Рисунок 4.13 – Реализация метода get\_int

Алгоритм получения данных: прочитайте размер сообщения (4 байта – 32 бита), создайте буфер, переведите размер из сетевого порядка байт в порядок байт для платформы, читайте, пока объем считанной информации не будет равен размеру полученного сообщения.

#### 4.3.4.2 Класс AESSocket

AES-сокеты включают в себя Socket. Отличие AES-сокетов от Socket в следующем: в конструкторе AES-сокетов происходит обмен AES-ключами при помощи Deffie-Hellman алгоритма (см. рис. 4.16-4.18); при отправке данные предварительно шифруются (см. рис. 4.19); при получении данные расшифровываются (см. рис. 4.20).

```
def __init__(self, socket):
    self.socket = socket
    aes_key = get_key_with_dh(self.socket)
    self.aes = AES(aes_key)
```

Рисунок 4.14 – Конструктор AES-сокетов

```
def get_key_with_dh(socket):
    prime = socket.read_big_number()
    g = socket.read_big_number()
    server_public_key = socket.read_big_number()
    dh = DH() # Diffie Hellman
    client_private_key = dh.generate_private_key(prime)
    client_public_key = dh.generate_public_key(prime, g, client_private_key)
    socket.write_bytes(str(client_public_key).encode())
    key = dh.get_secret_key(server_public_key, client_private_key, prime)
    byte_key = key.to_bytes(16, 'big')
    return byte_key
```

Рисунок 4.15 – Генерации DH

```
def write_bytes(self, data):
    encrypted_data = self.aes.encrypt(data)
    self.socket.write_bytes(encrypted_data)
```

Рисунок 4.19 – Шифрование перед отправкой

```
def read_bytes(self):
    received_data = self.socket.read_bytes()
    data = self.aes.decrypt(received_data)
    return data
```

Рисунок 4.20 – Расшифрование перед получением

#### 4.3.4.3 Класс RSASocketWriter

RSASocketWriter алгоритмом схож с AESSocket т.к. RSA ассиметричен, то данный класс умеет только расшифровывать сообщения при получении.

При создании сокета происходит получение публичного RSA ключа (см. рис. 4.21).

```
def __init__(self, socket, rsa_public_key=None):
    self.socket = socket
    if rsa_public_key is None:
        received_public_key = self.socket.read_bytes()
        self.rsa_public_key = RSA.create_public_key_from_bytes(received_public_key)
    else:
        self.rsa_public_key = rsa_public_key
```

Рисунок 4.21 – Получение публичного ключа

```
def write_bytes(self, data):
    encrypted_data = RSA.encrypt(data, self.rsa_public_key)
    self.socket.write_bytes(encrypted_data)
```

Рисунок 4.22 – Шифрование перед отправкой

Хочется заметить, что AES и RSA сокеты работают поверх произвольного сокета, т.е. можно легко создать RSA сокет поверх AES сокета или любую другую комбинацию из произвольного числа сокетов.



## 5 ОПИСАНИЕ АВТОРСКОГО ПРОТОКОЛА

### 5.1 Взаимодействие с сервером

Для использования протокола необходимо соединение с сервером, в рамках курсового проекта была выбрана архитектура, описанная ниже. Для каждого пользователя создается собственное соединение с сервером.

### 5.2 Протокол передачи данных

Не существует идеального протокола для передачи данных. Многие действительно хорошие протоколы (MTProto и другие) привязаны к определенным приложениям или задачам компаний, что не позволяет произвольно использовать данные протоколы. Протоколы для безопасной передачи данных, которые используются для произвольных задач, слишком обобщенные, что плохо сказывается на безопасности, а также содержат много дополнительной информации, которая не всегда бывает нужна, что сказывается на производительности.

Исходя из этих недостатков можно сделать вывод, что для обеспечения высокопроизводительной безопасной передачи данных необходимо создавать собственный протокол исходя из бизнес-логики приложений или сервисов.

Существует стандартный подход к созданию протокола безопасной передачи данных, состоящий из двух частей: алгоритм работы протокола на этапе аутентификации и алгоритм работы протокола на этапе передачи данных.

Алгоритм работы протокола на этапе аутентификации необходим для того, чтобы предоставить доступ пользователю к ресурсу, при этом следует учитывать, что по умолчанию канал передачи данных небезопасный.

Алгоритм работы протокола на этапе передачи данных учитывает, что уже получены необходимые сведения (ключи) для передачи данных по защищенному каналу связи. В разработку данной части протокола входит проектирование структуры сообщения передачи данных, а также общий алгоритм шифрования данных, алгоритм проверки целостности и подлинности данных. Рассмотрим подробнее суть предлагаемого протокола для безопасной передачи данных в программном обеспечении для агрегации сообщений.

### 5.2.1 Инициализация

Первый шаг алгоритма аутентификации в клиент-серверном взаимодействии заключается в том, что пользователь подключается к серверу. После чего необходимо создать безопасный канал передачи данных.

Создание безопасного канала между клиентом и сервером заключается в том, чтобы сгенерировать ключи при помощи алгоритма Диффи-Хеллмана [8] на стороне клиента и сервера, после чего отправлять все данные, зашифрованные симметричным алгоритмом AES [9]. На рис. 1 представлена стадия инициализации авторского протокола.

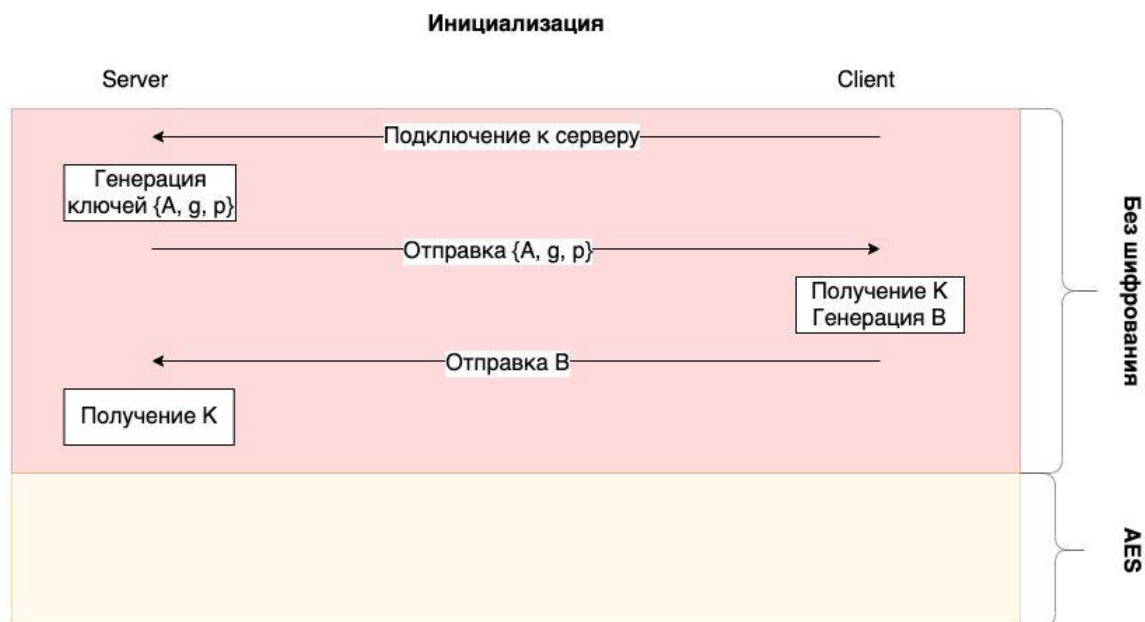


Рисунок 5.4 – Стадия инициализации

На рисунке 5.5 представлена проекция данного протокола в коде со стороны клиента.

```
def __init__(self):
    self.socket = Socket()
    self.aes_socket = AESSocket(self.socket)
```

Рисунок 5.5 – Клиентская сессия создает Socket и AESSocket

На рисунке 5.5 изображено поле `self.socket`, которое устанавливает соединение с сервером, а затем создается `AESSocket` из обычного сокета, что приводит к обмену AES ключами через Diffie-Hellman с сервером. Клиентская сессия создается при подключении пользователя к серверу.

### 5.2.2 Аутентификация

Первый «слой» безопасной передачи данных настроен на стадии инициализации, но он не является достаточно надежным, чтобы передавать критически важные данные: пароль или логин.

Стоит отметить, что следующие «слои» безопасности всегда генерируют новые ключи криптографических алгоритмов, для разных алгоритмов используют разные ключи для повышения криптостойкости.

Следующий шаг заключается в создании дополнительного «слоя» безопасности для передачи логина и пароля. На стороне сервера генерируются новые RSA [10] ключи ( $e$ ,  $d$ ,  $n$ ). Публичный ключ для шифрования принимается от сервера, предварительно расшифрованный алгоритмом AES. Таким образом, создается безопасный канал для передачи данных с клиента на сервер (использование криптографических алгоритмов AES и RSA). Через данный канал передается логин и пароль, после чего сервер отвечает клиенту успехом или неудачей, шифруя ответ только алгоритмом AES. В случае успеха обрабатывается пароль пользователя аналогичным образом. На рисунке 5.6 представлена стадия идентификации и аутентификации.

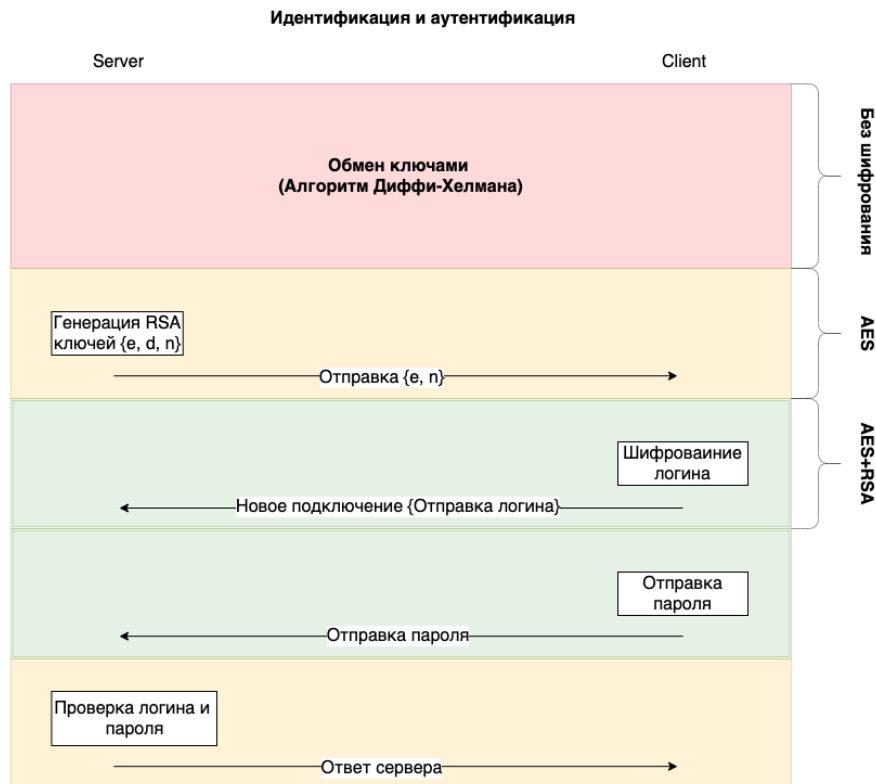


Рисунок 5.6 – Стадия идентификации и аутентификации

В коде это выражено следующим образом (см. рис. 5.7).

```
def sign_in_password(self, login: str, password: str):
    data = {
        'command': 'sign in',
        'method': 'password'
    }
    self.send_command_and_check_answer(data)
    rsa_socket_writer = RSASocketWriter(self.aes_socket)
    self.send_secret_string(login, rsa_socket_writer)
    self.send_secret_string(password, rsa_socket_writer)
    result = self.receive_answer()
    if result['status'] == 'ok':
        self.aes_socket.exchange_aes_key()
    return result
```

Рисунок 5.7 – Идентификация, аутентификация и авторизация

После успешной аутентификации необходимо создать сессионный ключ. Данный ключ генерируется при помощи алгоритма Диффи-Хеллмана поверх существующего AES соединения, в дальнейшем используется только последний ключ (см. рис. 5.8).

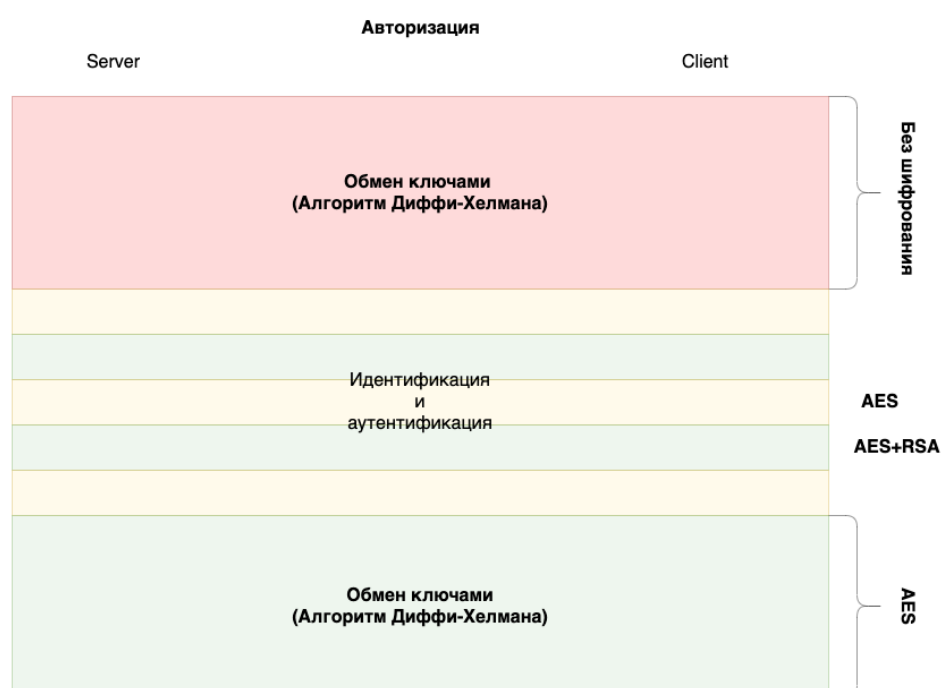


Рисунок 5.8 – Стадия авторизации

```
def exchange_aes_key(self):  
    aes_key = get_key_with_dh(self)  
    self.aes = AES(aes_key)
```

Рисунок 5.9 – Обмен ключами

Процесс авторизации закончен. Дальнейшая передача данных осуществляется по защищенному каналу.

Для получения доступа к данным необходимо взломать первый алгоритм Диффи-Хеллмана, публичный ключ RSA сервера и второй алгоритм Диффи-Хеллмана. Может показаться, что данные меры безопасности избыточны, но данный протокол рассчитан на то, что через несколько лет появятся вычислительные мощности в совокупности с современными алгоритмами и искусственным интеллектом, которые будут способны взломать алгоритмы RSA и Диффи-Хеллмана за минимальное время. То есть, через несколько лет данные, которые недостаточно хорошо зашифрованы, будут легкодоступными для злоумышленников. Но при использовании предложенного алгоритма, взлом можно отложить на будущее, что дает гарантию безопасности данных на долгий период.

Частным случаем аутентификации является восстановление сессии, но для этого необходимо её создать и сохранить на сервере и клиенте. Создание сессии доступно только аутентифицированным клиентам, для этого клиент отправляет запрос серверу с командой «создать сессию». После чего клиент и сервер генерируют одинаковый токен (Diffie-Hellman) и сохраняют его. Дополнительно клиент создает подпись и отправляет ключ серверу для проверки подписи при следующем восстановлении сессии. При этом клиент сохраняет только подписанный токен. (см. рис 5.10).

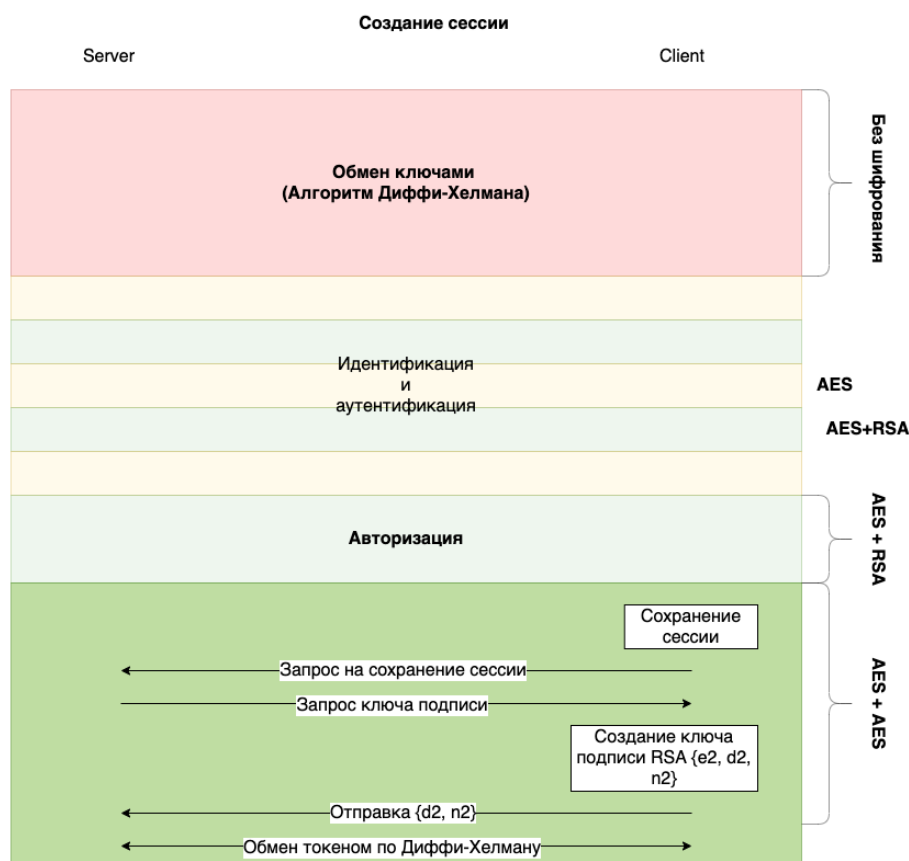


Рисунок 5.10 – Создание сессии

```
def create_session(request, login, password):
    new_session = ClientSession()
    response = new_session.sign_in_password(login, password)
    if response['status'] == 'ok':
        token = new_session.create_token()
        request.session['token'] = token
        request.session['login'] = login
    else:
        raise RuntimeError('Incorrect login or password')
    return new_session
```

Рисунок 5.11 – Создание сессии (токена) в коде

При восстановлении сессии отправляется логин и подписанный токен. Сессия восстанавливается только при успешном сравнении расшифрованного токена (при помощи ключа проверки подписи) пользователя с токеном из базы данных сервера (см. рис. 5.12).

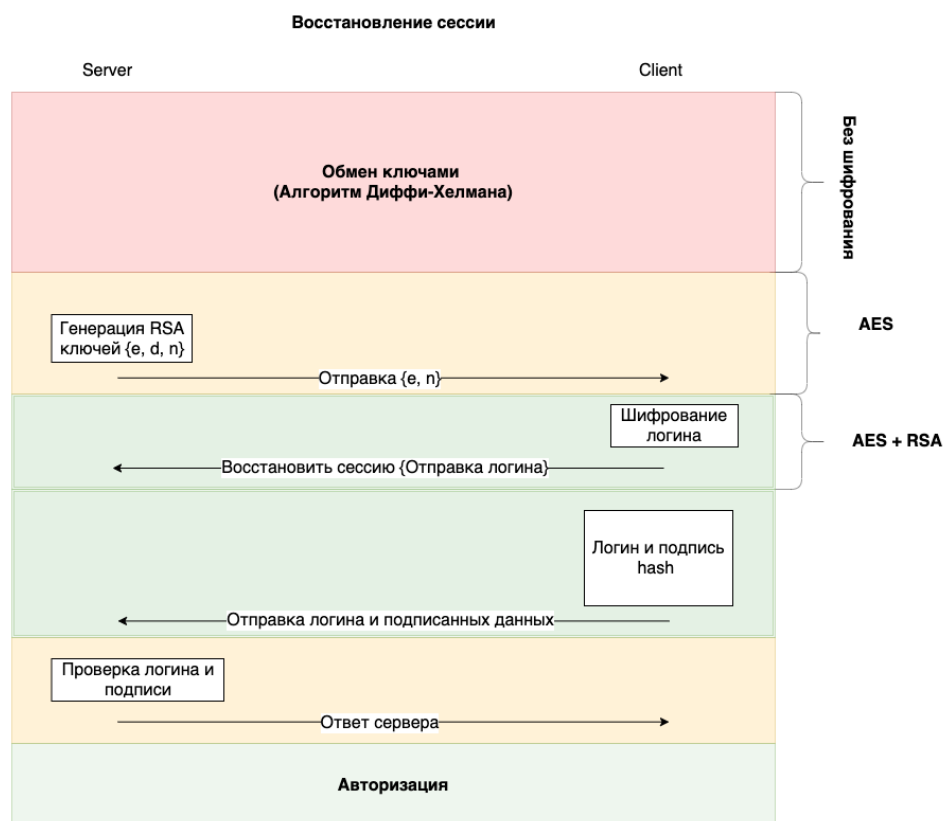


Рисунок 5.12 – Восстановление сессии

```

def restore_session(request):
    restored_session = ClientSession()
    login = request.session['login']
    token = request.session['token']
    restored_session.sign_in_token(login, token)
    return restored_session
  
```

Рисунок 5.13 – Восстановление сессии в коде

### 5.2.3 Достоинства и недостатки авторского протокола

В основе протокола лежит оригинальная комбинация симметричного алгоритма шифрования AES, протокола Диффи-Хеллмана для обмена 2048-битными RSA-ключами между двумя устройствами и SHA-256.

Достоинства предлагаемого протокола:



1. Первый этап шифрования устанавливается до аутентификации, что позволяет полностью скрыть информацию о пользователе.

2. Аутентификация происходит по защищенному каналу связи.

3. Для шифрования сообщений используется симметричное шифрование.

4. Авторизацию пользователи могут выполнить, как при помощи пароля, так и без пароля при помощи сессионных токенов.

Недостатки.

Недостатки протокола будут выявлены в процессе программной реализации и тестирования программного обеспечения в реальных условиях.

Также стоит заметить, что протокол привязан к приложению, в будущем структура приложения будет развиваться и увеличиваться, что скажется на структуре и алгоритмах шифрования сообщений, а также алгоритме аутентификации.

## ВЫВОДЫ

Во время выполнения курсового проекта были получены навыки в области криптографических алгоритмов и методов по защите данных.

На базе данных знаний была разработан авторский протокол передачи данных, используемый в системе агрегирования. Преимущества которого заключается в повышенной криптостойкости за счет уникальной комбинации существующих криптографических алгоритмов.

Результатом является клиентское приложение – сайт. При этом в реализации были учтены конфиденциальность пользователя, безопасность передачи данных, а также возможность дальнейшего развития – добавление большинства стилей и дальнейшая поддержка сервера. Авторский протокол передачи данных, идеально подходит для данной задачи и не добавляет дополнительной нагрузки, как в случае использования других протоколов передачи данных.

В дальнейшей данный проект можно расширить другими возможностями: добавление анимации на сайт, поддержка мобильных устройств, создание множества стилей, добавление темной и светлой темы.

## ПЕРЕЧЕНЬ ССЫЛОК

1. The Secure Sockets Layer (SSL) Protocol Version 3.0 [электронный ресурс]. – Режим доступа: <https://datatracker.ietf.org/doc/html/rfc6101> (дата обращения 05.11.2021).
2. The Transport Layer Security (TLS) Protocol Version 1.2 [электронный ресурс]. – Режим доступа: <https://datatracker.ietf.org/doc/html/rfc5246> (дата обращения 05.11.2021).
3. The Tox Reference [электронный ресурс]. – Режим доступа: <https://zetok.github.io/tox-spec/> (дата обращения 05.11.2021).
4. Signal [электронный ресурс]. – Режим доступа: <https://signal.org/docs/> (дата обращения 05.11.2021).
5. Echo Protocol [электронный ресурс]. – Режим доступа: <https://datatracker.ietf.org/doc/html/rfc862> (дата обращения 05.11.2021).
6. Bitmessage protocol specification [электронный ресурс]. – Режим доступа: [https://wiki.bitmessage.org/index.php/Protocol\\_specification](https://wiki.bitmessage.org/index.php/Protocol_specification) (дата обращения 05.11.2021).
7. MTProto Mobile Protocol [электронный ресурс]. – Режим доступа: <https://core.telegram.org/mtproto> (дата обращения 05.11.2021).
8. Diffie-Hellman Key Agreement Method [электронный ресурс]. – Режим доступа: <https://datatracker.ietf.org/doc/html/rfc2631> (дата обращения 05.11.2021).
9. The Advanced Encryption Standard (AES) Cipher Algorithm in the SNMP User-based Security Model [электронный ресурс]. – Режим доступа: <https://datatracker.ietf.org/doc/html/rfc3826> (дата обращения 05.11.2021).
10. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptograph Specifications Version 2.1 [электронный ресурс]. – Режим доступа: <https://datatracker.ietf.org/doc/html/rfc3447> (дата обращения 05.11.2021).
11. US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF) [электронный ресурс]. – Режим доступа: <https://datatracker.ietf.org/doc/html/rfc6234> (дата обращения 05.11.2021).

## ПРИЛОЖЕНИЕ А. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Утверждено

зав. кафедрой ПИ

Зори С.А.

«\_\_\_\_» \_\_\_\_\_ 2021 г.

**ТЕХНИЧЕСКОЕ ЗАДАНИЕ**  
на курсовой проект по дисциплине  
«Безопасность программ и данных»

студенту группы ПИ-186 Моргунову Арсению Геннадьевичу

Тема: “Создание безопасного протокола обмена данными между клиентом и сервером для системы SILUR”

Вариант № 5

Задание:

Разработать клиентскую часть протокола безопасной аутентификации и передачи данных между клиентом и сервером.

1. Создание безопасного подключения для любых данных.

Генерация ключей при помощи алгоритма Деффи-Хеллмана.

Шифрование данных при помощи симметричного алгоритма AES.

2. Создание ассиметричного шифрования (RSA) при передаче данных повышенной важности (при генерации ключей для передачи данных).

3. Реализовать возможность сохранения сессии пользователем, чтобы при последующем подключении не запрашивать пароль пользователя заново.

Для восстановления сессии использовать цифровую подпись при помощи ассиметричного алгоритма RSA.

4. В базе данных хранить логины и соответствующие им токены пользователей.

5. Разработать команды управления между клиентом и сервером для предметной области системы SILUR.

Пример команд:

Получение списка подписок.

Создание подписки.

Просмотреть историю публикации (передача содержимого сообщений: подпись, текст, файлы и т.д.).

6. Тестирование протокола в реальных условиях.

7. Создание понятного пользовательского интерфейса.

## График выполнения курсового проекта:

Неделя	Работа
1-2	Выдача задания и изучение задания
3	Анализ требований к системе и способов их реализации
4-5	Проектирование
6-7	Организация процессов
8	Поддержка режимов работы, проектирование общей структуры
9-13	Разработка программ
14	Оформление пояснительной записки
15-17	Защита курсового проекта

Дата выдачи задания

1.09.2021

Задание принял

Руководители проекта

Моргунов А.Г.

Чернышова А.В.

Ногтев Е.А.

## ПРИЛОЖЕНИЕ Б. ЭКРАННЫЕ ФОРМЫ



Рисунок Б.1 – Профиль



Рисунок Б.2 – Авторизация



Рисунок Б.3 – Регистрация

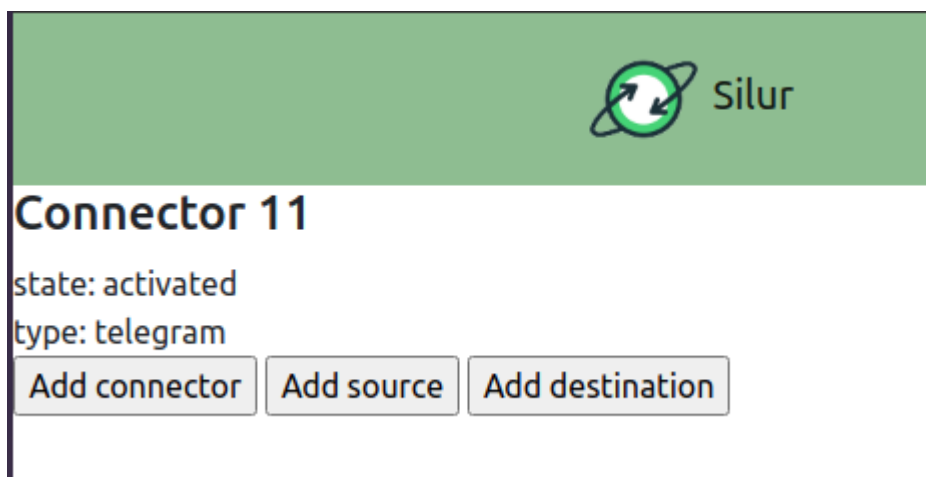


Рисунок Б.4 – Соединения

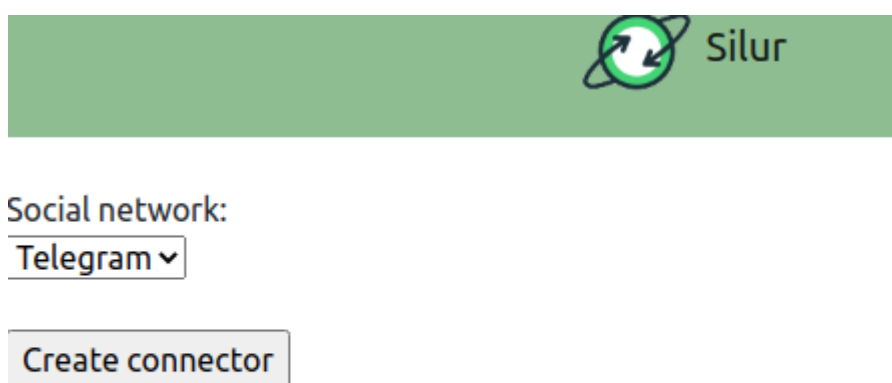



Рисунок Б.5 – Создание соединения






Silur

Connector:

Channel name:

Рисунок Б.6 – Создание источника



Silur

Connector:

Channel name:

Рисунок Б.7 – Создание приёмника



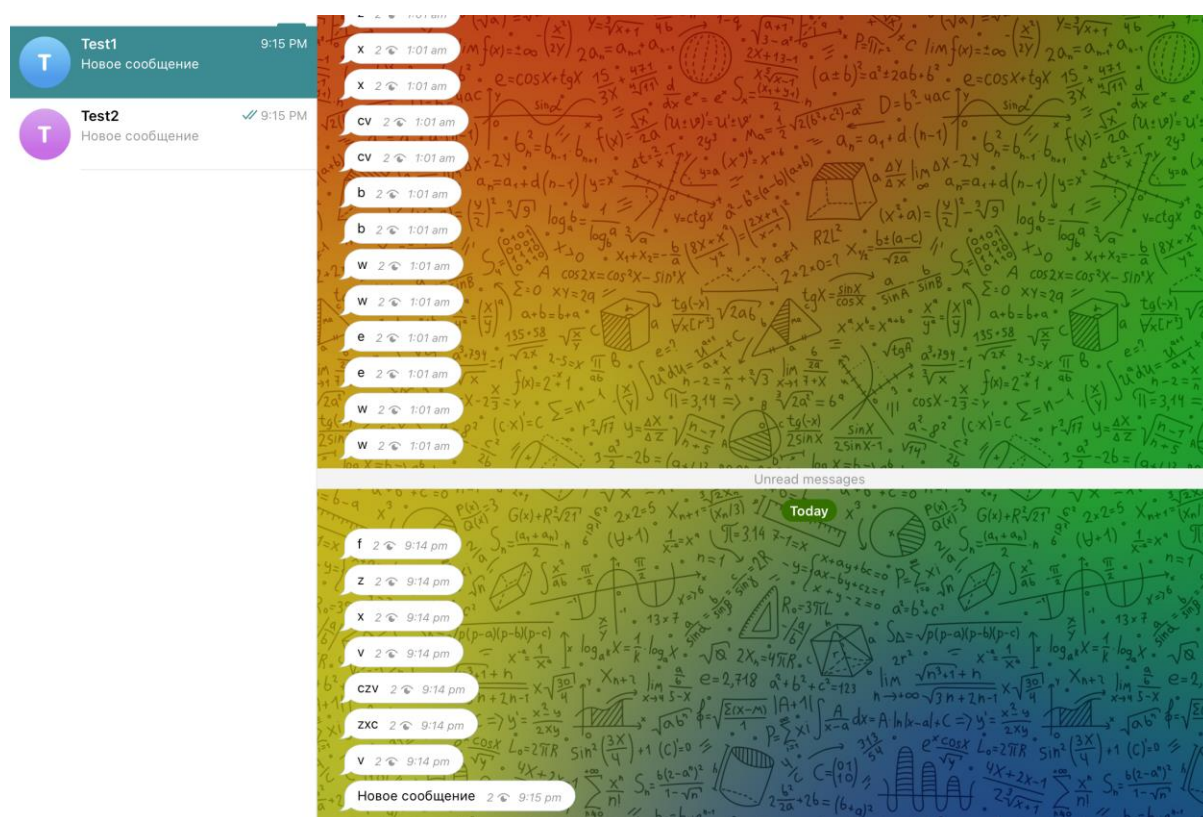


Рисунок Б.9 – Агрегация сообщения в канал-приёмник

## ПРИЛОЖЕНИЕ В. АНТИПЛАГИАТ

# ПРИЛОЖЕНИЕ Г. ЛИСТИНГ КОДА

```

from .Socket import Socket
from .Crypto import *

class AESSocket:
    def __init__(self, socket):
        self.socket = socket
        aes_key = get_key_with_dh(self.socket)
        self.aes = AES(aes_key)

    def exchange_aes_key(self):
        aes_key = get_key_with_dh(self)
        self.aes = AES(aes_key)

    def write_bytes(self, data):
        encrypted_data = self.aes.encrypt(data)
        self.socket.write_bytes(encrypted_data)

    def read_bytes(self):
        received_data = self.socket.read_bytes()
        data = self.aes.decrypt(received_data)
        return data

    def write_with_rsa(self, data, rsa_public_key):
        encrypted_data = RSA.encrypt(data, rsa_public_key)
        self.write_bytes(encrypted_data)

    def read_with_rsa(self, rsa_private_key):
        received_data = self.read_bytes()
        data = RSA.decrypt(received_data, rsa_private_key)
        return data

    def read_big_number(self):
        # from bytes string to int
        # b'12341234' -> 12341234
        return int(self.read_bytes().decode('utf-8'))

def get_key_with_dh(socket):
    prime = socket.read_big_number()
    g = socket.read_big_number()
    server_public_key = socket.read_big_number()
    dh = DH() # Diffie Hellman
    client_private_key = dh.generate_private_key(prime)
    client_public_key = dh.generate_public_key(prime, g, client_private_key)
    socket.write_bytes(str(client_public_key.encode()))
    key = dh.get_secret_key(server_public_key, client_private_key, prime)
    byte_key = key.to_bytes(16, 'big')
    return byte_key

from .Socket import Socket
from .AESocket import AESocket, get_key_with_dh
import json
from .RSASocketWriter import RSASocketWriter
from .Crypto import DH, RSA

    self.send_secret_string(password, rsa_socket_writer)
    return self.receive_answer()

def sign_in_password(self, login: str, password: str):
    data = {
        'command': 'sign in',
        'method': 'password'
    }
    self.send_command_and_check_answer(data)
    rsa_socket_writer = RSASocketWriter(self.aes_socket)
    self.send_secret_string(login, rsa_socket_writer)
    self.send_secret_string(password, rsa_socket_writer)
    result = self.receive_answer()
    if result['status'] == 'ok':
        self.aes_socket.exchange_aes_key()
    return result

def sign_in_token(self, login: str, hex_token: str):
    data = {
        'command': 'sign in',
        'method': 'token',
        'sign': hex_token
    }
    self.send_command_and_check_answer(data)
    rsa_socket_writer = RSASocketWriter(self.aes_socket)
    self.send_secret_string(login, rsa_socket_writer)
    result = self.receive_answer()
    if result['status'] == 'ok':
        self.aes_socket.exchange_aes_key()
    return result

def sign_out(self):
    data = {
        'command': 'sign out'
    }
    self.send_command_and_check_answer(data)
    result = self.receive_answer()
    self.aes_socket = None
    return result

    def create_connector(self, social_network_type='telegram', create_receiver=True,
create_sender=True):
        data = {
            'command': 'create connector',
            'social network type': social_network_type,
            'create receiver': create_receiver,
            'create sender': create_sender
        }
        self.send_command_and_check_answer(data)
        json_answer = self.receive_answer()
        while json_answer['status'] == 'need value':
            user_value = input(json_answer['value name'] + ': ')
            rsa_socket_writer = RSASocketWriter(self.aes_socket)
            rsa_socket_writer.write_bytes(user_value.encode())
            # self.aes_socket.write_bytes(user_value.encode())
            json_answer = self.receive_answer()
        return json_answer

    def create_connector_command(self, social_network_type='telegram', create_receiver=True,
create_sender=True):
        data = {
            'command': 'create connector',
            'social network type': social_network_type,
            'create receiver': create_receiver,
            'create sender': create_sender
        }
        self.send_command_and_check_answer(data)
        json_answer = self.receive_answer()
        return json_answer

def send_user_data(self, string):
    rsa_socket_writer = RSASocketWriter(self.aes_socket)
    rsa_socket_writer.write_bytes(string.encode())
    json_answer = self.receive_answer()

```

```

    return json_answer

def create_source(self, connector_id, channel_name: str):
    data = {
        'command': 'create source',
        'connector id': connector_id,
        'channel name': channel_name
    }
    self.send_command_and_check_answer(data)
    return self.receive_answer()

def create_destination(self, connector_id, channel_name: str):
    data = {
        'command': 'create destination',
        'connector id': connector_id,
        'channel name': channel_name
    }
    self.send_command_and_check_answer(data)
    return self.receive_answer()

def create_relation(self, source_id, destination_id):
    data = {
        'command': 'create relation',
        'source id': source_id,
        'destination id': destination_id
    }
    self.send_command_and_check_answer(data)
    return self.receive_answer()

def get_connectors(self):
    data = {
        'command': 'get connectors'
    }
    self.send_command_and_check_answer(data)
    return self.receive_answer()

def get_senders(self):
    data = {
        'command': 'get senders'
    }
    self.send_command_and_check_answer(data)
    return self.receive_answer()

def get_receivers(self):
    data = {
        'command': 'get receivers'
    }
    self.send_command_and_check_answer(data)
    return self.receive_answer()

def get_sources(self, receiver_id):
    data = {
        'command': 'get sources',
        'receiver id': receiver_id
    }
    self.send_command_and_check_answer(data)
    return self.receive_answer()

def get_destinations(self, sender_id):
    data = {
        'command': 'get destinations',
        'sender id': sender_id
    }
    self.send_command_and_check_answer(data)
    return self.receive_answer()

def get_relations(self, destination_id):
    data = {
        'command': 'get relations',
        'destination id': destination_id
    }
    self.send_command_and_check_answer(data)
    return self.receive_answer()

def create_token(self):
    data = {
        'command': 'create token'
    }
    self.send_command_and_check_answer(data)

    token = get_key_with_dh(self.aes_socket)
    hex_token = token.hex()
    hex_upper_token = hex_token.upper()
    bytes_hex_token = hex_upper_token.encode()
    print(hex_upper_token)

    private_key = RSA.generate_private_key()
    public_key = private_key.public_key()
    public_key_pem = RSA.serialize_public_key(public_key)
    self.aes_socket.write_bytes(public_key_pem)

    signed_token = RSA.sign(bytes_hex_token, private_key)
    hex_signed_token = signed_token.hex()
    # self.aes_socket.write_bytes(signed_token)
    return hex_signed_token

    @staticmethod
    def send_secret_string(string: str, rsa_socket_writer):
        byte_data = string.encode()
        rsa_socket_writer.write_bytes(byte_data)

    @staticmethod
    def send_secret_bytes(data, rsa_socket_writer):
        rsa_socket_writer.write_bytes(data)

    def send_command_and_check_answer(self, command):
        json_dump = json.dumps(command)
        byte_json_dump = json_dump.encode()
        self.aes_socket.write_bytes(byte_json_dump)
        result = self.receive_answer()
        # print(f'received answer: {result}')
        if result['status'] == 'error':
            raise RuntimeError(result['description'])
        return result

    def receive_answer(self):
        answer_bytes = self.aes_socket.read_bytes()
        answer = answer_bytes.decode()
        json_answer = json.loads(answer)
        return json_answer

    from random import randint
    import os
    from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
    from cryptography.hazmat.primitives.asymmetric import rsa
    from cryptography.hazmat.primitives import serialization
    from cryptography.hazmat.primitives import hashes
    from cryptography.hazmat.primitives.asymmetric import padding

    class DH:
        @staticmethod
        def generate_private_key(prime: int):
            return randint(1, prime-1)

        @staticmethod
        def generate_public_key(prime: int, g: int, private_key: int):
            return pow(g, private_key, prime)

        @staticmethod
        def get_secret_key(server_public_key: int, client_private_key: int, prime: int):
            return pow(server_public_key, client_private_key, prime)

    class AES:
        def __init__(self, key='0'):
            if key != '0':
                self.key = key
            else:
                self.key = self.generate_key()
            self.cipher = Cipher(algorithms.AES(self.key), modes.ECB())

        @staticmethod
        def generate_key():
            return os.urandom(16)

        def encrypt(self, data):
            encryptor = self.cipher.encryptor()
            aligned_data = align_to_size(data)

```

```

encrypted_data = encryptor.update(aligned_data)
encryptor.finalize()
return encrypted_data

def decrypt(self, encrypted_data):
    decryptor = self.cipher.decryptor()
    decrypted_data = decryptor.update(encrypted_data)
    decryptor.finalize()
    decrypted_data_without_alignment = release_align(decrypted_data)
    return decrypted_data_without_alignment

```

```

class RSA:
    @staticmethod
    def encrypt(data, public_key):
        return public_key.encrypt(
            data,
            padding.OAEP(
                mgf=padding.MGF1(algorithm=hashes.SHA256()),
                algorithm=hashes.SHA256(),
                label=None
            )
        )

    @staticmethod
    def decrypt(encrypted_data, private_key):
        return private_key.decrypt(
            encrypted_data,
            padding.OAEP(
                mgf=padding.MGF1(algorithm=hashes.SHA256()),
                algorithm=hashes.SHA256(),
                label=None
            )
        )

    @staticmethod
    def create_public_key_from_bytes(data):
        return serialization.load_pem_public_key(data)

    @staticmethod
    def create_private_key_from_bytes(data):
        return serialization.load_pem_private_key(data, password=None)

    @staticmethod
    def serialize_public_key(public_key):
        return public_key.public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo
        )

    @staticmethod
    def serialize_private_key(private_key):
        return private_key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.TraditionalOpenSSL,
            encryption_algorithm=serialization.NoEncryption()
        )

    @staticmethod
    def generate_private_key():
        return rsa.generate_private_key(
            public_exponent=65537,
            key_size=2048,
        )

    @staticmethod
    def sign(data, private_key):
        return private_key.sign(
            data,
            padding.PKCS1v15(),
            hashes.SHA256()
        )

    @staticmethod
    def verify(signature, data, public_key):
        try:
            public_key.verify(
                signature,
                data,

```

```

padding.PKCS1v15(),
hashes.SHA256()
)
return True
except Exception:
    return False

```

```

def print_variable_and_size(variable_name: str, variable):
    print(f"{variable_name}: {variable}")
    print(f"{variable_name} length: {len(variable) * 8} bits ({len(variable)} bytes)\n")

```

```

def align_to_size(data, size=16):
    aligned_data = data + chr(size - len(data) % size).encode('utf-8') * (size - len(data) % size)
    return aligned_data

```

```

def release_align(aligned_data):
    data = aligned_data
    last_byte = aligned_data[-1]
    while data[-1] == last_byte:
        data = data[:-1]
    return data
from .Crypto import RSA

```

```

class RSASocketReader:
    def __init__(self, socket, rsa_private_key=None):
        self.socket = socket
        if rsa_private_key is None:
            received_private_key = self.socket.read_bytes()
            self.rsa_private_key = RSA.create_private_key_from_bytes(received_private_key)
        else:
            self.rsa_private_key = rsa_private_key

    def read_bytes(self):
        received_data = self.socket.read_bytes()
        data = RSA.decrypt(received_data, self.rsa_private_key)
        return data

```

```

from .Crypto import RSA

```

```

class RSASocketWriter:
    def __init__(self, socket, rsa_public_key=None):
        self.socket = socket
        if rsa_public_key is None:
            received_public_key = self.socket.read_bytes()
            self.rsa_public_key = RSA.create_public_key_from_bytes(received_public_key)
        else:
            self.rsa_public_key = rsa_public_key

    def write_bytes(self, data):
        encrypted_data = RSA.encrypt(data, self.rsa_public_key)
        self.socket.write_bytes(encrypted_data)

```

```

import socket

```

```

class Socket:

    PACKAGE_SIZE = 4096 * 1024

    def __init__(self):
        self.address = "78.31.180.46"
        self.port = 50000
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket.settimeout(10)
        try:
            self.socket.connect((self.address, self.port))
            print(self.address)
            print(self.port)
            # print("Connected successfully")
        except Exception:
            print("Connection error")

    def read_big_number(self):
        # from bytes string to int

```

```

# b'12341234' -> 12341234
return int(self.read_bytes().decode('utf-8'))

def send_int(self, value: int) -> None:
    self.socket.send(value.to_bytes(length=4, byteorder='big'))

def get_int(self) -> int:
    int_size = 4
    result = self.socket.recv(int_size)
    result = int.from_bytes(result, byteorder='big')
    return result

def read_bytes(self) -> bytes:
    size = self.get_int()
    storage = bytearray()
    while len(storage) != size:
        storage += self.socket.recv(min(self.PACKAGE_SIZE, size - len(storage)))
    # print("Read")
    return bytes(storage)

def write_bytes(self, message_bytes) -> None:
    self.send_int(len(message_bytes))
    size = len(message_bytes)
    sent_bytes = 0
    while size != sent_bytes:
        sent_bytes += self.socket.send(message_bytes[sent_bytes:])
    # print(f"Sent: {sent_bytes}")

from django.urls import path, re_path

from . import views

urlpatterns = [
    path('silur/', views.homepage, name='homepage'),
    path('silur/connectors/', views.connectors, name='connectors'),
    path('silur/myprofile/', views.myprofile, name='myprofile'),
    path('silur/subscriptions/', views.subscriptions, name='subscriptions'),
    path('silur/registration/', views.registration, name='registration'),
    path('silur/sign_in/', views.sign_in, name='sign_in'),
    path('silur/create_source/', views.create_source, name='create_source'),
    path('silur/create_destination/', views.create_destination, name='create_destination'),
    # path('silur/relation/', views.relation, name='relation'),
    path('silur/create_connector/', views.create_connector, name='create_connector'),
    path('silur/create_relation/', views.create_relation, name='create_relation'),
    path("", views.homepage),
    path('silur/<str:data_info>', views.user_data_input, name='data_input'),
]

import random

from django.shortcuts import render, redirect
from forms import *
from .SilurCryptoSockets.ClienSession import ClientSession
import json
from .models import *
from .helper import find_in_dictionary
from django.http import HttpResponseRedirect, HttpResponse
from .SilurCryptoSockets.AESSocket import AESSocket

# session = ClientSession()
# session.sign_in_password('Login', 'Password')
sessions = {}

def is_authorized(request):
    if request.session.session_key is not None:
        return True
    else:
        return False

def create_session(request, login, password):
    new_session = ClientSession()
    response = new_session.sign_in_password(login, password)
    if response['status'] == 'ok':
        token = new_session.create_token()
        request.session['token'] = token
        request.session['login'] = login
    else:
        raise RuntimeError('Incorrect login or password')

return new_session

def restore_session(request):
    restored_session = ClientSession()
    login = request.session['login']
    token = request.session['token']
    restored_session.sign_in_token(login, token)
    return restored_session

def homepage(request):
    my_key = request.session.session_key
    # request.session.create()
    data = {
        'session_key': my_key,
    }
    response = render(request, 'silur/homepage.html', data)
    return response

def connectors(request):
    # if not is_authorized(request):
    #     return HttpResponseRedirect('/silur/')
    # To homepage</a>

    # if request.session['login'] not in sessions:
    #     sessions[request.session['login']] = restore_session(request)
    session = restore_session(request)
    if request.method == 'POST':
        print(request.POST)
        method = request.POST['method']
        method, obj = method.split(' ')
        if method == 'Add':
            session.sign_out()
            if obj == 'connector':
                return redirect('create_connector')
            elif obj == 'destination':
                return redirect('create_destination')
            elif obj == 'source':
                return redirect('create_source')
        # print(method)
    server_response = session.get_connectors()
    connectors = []
    # server_response = {'connectors': [{'id': 8, 'meta_info': {'state': 'activated'}, 'type': 'telegram'}],
    #     'description': 'The server sent your connectors', 'status': 'ok'}
    server_connectors = server_response['connectors']
    for connector_info in server_connectors:
        connector = Connector(connector_info['id'], type=connector_info['type'],
                               state=connector_info['meta_info']['state'])
        connectors.append(connector)
    # print(connectors)
    data = {
        'connectors': connectors
    }
    session.sign_out()
    return render(request, 'silur/connectors.html', data)

def myprofile(request):
    if request.method == 'POST':
        print(request.POST)
        method = request.POST['method']
        if method == 'Sign in':
            return HttpResponseRedirect('/silur/sign_in')
        elif method == 'Register':
            return HttpResponseRedirect('/silur/registration')
        elif method == 'Sign out':
            pass
        # if request.session['login'] in sessions:
        #     sessions[request.session['login']].sign_out()
        #     del sessions[request.session['login']]
        # request.session.delete(request.session.session_key)
        # return HttpResponseRedirect('/silur/')

    ##### TOKEN #####
    # sessions[request.session['login']].sign_out()
    # sessions[request.session['login']] = ClientSession()
    # print(sessions[request.session['login']].sign_in_token(request.session['login'],

```



```
request.session['token'])
```

```
login = request.session.get('login', 'Guest')
# if login == 'Guest':
buttons = ['Sign in', 'Register']
# else:
#     buttons = ['Sign out']
data = {
    'login': login,
    'token': request.session.get('token', 0),
    'buttons': buttons
}
return render(request, 'silur/myprofile.html', data)
```

```
def subscriptions(request):
    # if not is_authorized(request):
    #     return HttpResponseRedirect("/h1>You must be authorized!</h1><a href=\"/silur/\">To
homepage</a>")

    # if request.session['login'] not in sessions:
    #     sessions[request.session['login']] = restore_session(request)
    session = restore_session(request)
    if request.method == 'POST':
        print(request.POST)
        method = request.POST['method']
        method, obj = method.split(' ')
        if method == 'Add':
            session.sign_out()
            return redirect('create_relation')
    senders = session.get_senders()
    server_destinations = []
    for sender in senders['senders']:
        server_response_destinations = session.get_destinations(sender['id'])
        server_destinations.append(server_response_destinations['destinations'])
    # server_response_destinations = {'description': 'Sent destinations', 'destinations': [
    #     {'constrains': None, 'id': 14, 'meta_info': {'chat_id': -1001610866332}, 'sender_id': 7},
    #     {'constrains': None, 'id': 15, 'meta_info': {'chat_id': -1001578380296}, 'sender_id': 7}], 'status':
    'ok'}
    # server_destinations = server_response_destinations['destinations']

    # Hardcoded values of relations
    # relations_for_destination_14 = {'description': 'Sent relations', 'relations': [
    #     {'constrains': {}, 'destination_id': 14, 'id': 13, 'is_frozen': True, 'source_id': 15}], 'status': 'ok'}
    # relations_for_destination_15 = {'description': 'Sent relations', 'relations': [
    #     {'constrains': {}, 'destination_id': 15, 'id': 14, 'is_frozen': True, 'source_id': 16},
    #     {'constrains': {}, 'destination_id': 15, 'id': 15, 'is_frozen': True, 'source_id': 15}], 'status': 'ok'}
    # server_response_relations = []
    # server_response_relations.append(relations_for_destination_15)
    # server_response_relations.append(relations_for_destination_14)
```

```
destinations_with_sources = []
for destination_list in server_destinations:
    for destination in destination_list:
        destination_obj = Destination(destination['id'],
chat_id=destination['meta_info']['chat_id'],
            sender_id=destination['sender_id'])
        # print(f"Destination ID: {destination['id']}")
        server_response = session.get_relations(destination['id'])
        # server_response = server_response_relations.pop()
        relations = server_response['relations']
        # print(relations)
        server_receivers = session.get_receivers()
        for receiver in server_receivers['receivers']:
            server_sources = session.get_sources(receiver['id'])
            # server_sources = {'description': 'Sent sources',
            #     'sources': [{'id': 15, 'meta_info': {'chat_id': -1001615953184}, 'receiver_id': 7},
            #     {'id': 16, 'meta_info': {'chat_id': -1001610866332}, 'receiver_id': 7}],
            #     'status': 'ok'}
            sources = []
            for relation in relations:
                server_source = find_in_dictionary_set(server_sources['sources'], 'id',
relation['source_id'])
                # print(f"Server source: {server_source}")
                source = Source(server_source['id'], chat_id=server_source['meta_info']['chat_id'],
                    receiver_id=server_source['receiver_id'])
                sources.append(source)
            destinations_with_sources.append((destination_obj, sources))
    # print(destinations_with_sources)
```

```
data = {
    'destinations': destinations_with_sources,
}
session.sign_out()
return render(request, 'silur/subscriptions.html', data)
```

```
def registration(request):
    error = ""
    if request.method == 'POST':
        form = RegistrationForm(request.POST)
        if form.is_valid():
            data = form.cleaned_data
            session = ClientSession()
            result = session.register(data['login'], data['password'],
                data['first_name'], data['second_name'])
            # session.sign_out()
            if result['status'] == 'ok':
                try:
                    session = create_session(request, data['login'], data['password'])
                    return redirect('myprofile')
                except RuntimeError as re:
                    error = re.args[0]
                    session.sign_out()
            else:
                error = "Cannot create user"
                session.sign_out()
        else:
            form = RegistrationForm()
    data = {
        'form': form,
        'error': error
    }
    return render(request, 'silur/registration.html', data)
```

```
def sign_in(request):
    error = ""
    if request.method == 'POST':
        form = SignInForm(request.POST)
        if form.is_valid():
            data = form.cleaned_data
            try:
                session = create_session(request, data['login'], data['password'])
                session.sign_out()
                # sessions[data['login']] = session
                return redirect('myprofile')
            except RuntimeError as re:
                error = re.args[0]
                print(error)
            else:
                form = SignInForm()
    data = {
        'form': form,
        'button_name': 'Sign in',
        'error': error
    }
    return render(request, 'silur/form_template.html', data)
```

```
def user_data_input(request, data_info):
    if request.method == 'POST':
        form = DataFromUser(request.POST)
        # print(form.data)
        # print(request.POST)
        if form.is_valid():
            data = form.cleaned_data
            session = sessions[request.session.session_key]
            response = session.send_user_data(data['data'])
            if response['status'] == 'need value':
                return HttpResponseRedirect("/silur/" + response['value name'] + "/")
            else:
                session.sign_out()
                del sessions[request.session.session_key]
                return HttpResponseRedirect("/silur/connectors/")
        else:
            form = DataFromUser()
    data = {
        'form': form,
```

```

        'data_info': data_info
    }
    return render(request, 'silur/user_data_input.html', data)

def create_destination(request):
    session = restore_session(request)
    if request.method == 'POST':
        form = DestinationForm(request.POST)
        # print(form.data)
        # print(request.POST)
        if form.is_valid():
            data = form.cleaned_data
            session.create_destination(data['connector_id'], data['channel_name'])
            session.sign_out()
            return HttpResponseRedirect('/silur/connectors/')
        else:
            form = DestinationForm()
    server_connectors = session.get_connectors()
    # server_connectors = {'connectors': [{'id': 8, 'meta_info': {'state': 'activated'}, 'type': 'telegram'}],
    #                      'description': 'The server sent your connectors', 'status': 'ok'}
    connectors_id = []
    for connector in server_connectors['connectors']:
        connectors_id.append((connector['id'], "Connector " + str(connector['id'])))

    data = {
        'form': form,
        'button_name': 'Save destination',
        'connectors': connectors_id
    }

    return render(request, 'silur/create_destination.html', data)

def create_relation(request):
    session = restore_session(request)
    if request.method == 'POST':
        form = RelationForm(request.POST)
        print(form.data)
        # print(request.POST)
        if form.is_valid():
            data = form.cleaned_data
            session.create_relation(data['source'], data['destination'])
            session.sign_out()
            return redirect('subscriptions')
        else:
            form = RelationForm()

    receivers = session.get_receivers()
    sources = []
    for receiver in receivers['receivers']:
        server_response_sources = session.get_sources(receiver['id'])
        sources.append(server_response_sources['sources'])

    # sources = session.get_sources()
    # sources = {'description': 'Sent sources',
    #            'sources': [{'id': 15, 'meta_info': {'chat_id': -1001615953184}, 'receiver_id': 7},
    #                       {'id': 16, 'meta_info': {'chat_id': -1001610866332}, 'receiver_id': 7}], 'status': 'ok'}
    sources_id = []
    for source_list in sources:
        for source in source_list:
            sources_id.append((source['id'], "Source " + str(source['id'])))

    senders = session.get_senders()
    destinations = []
    for sender in senders['senders']:
        server_response_destinations = session.get_destinations(sender['id'])
        destinations.append(server_response_destinations['destinations'])
    # destinations = session.get_destinations()
    # destinations = {'description': 'Sent destinations', 'destinations': [
    #                 {'constraints': None, 'id': 14, 'meta_info': {'chat_id': -1001610866332}, 'sender_id': 7},
    #                 {'constraints': None, 'id': 15, 'meta_info': {'chat_id': -1001578380296}, 'sender_id': 7}], 'status':
    #                 'ok'}
    destinations_id = []
    for destination_list in destinations:
        for destination in destination_list:
            destinations_id.append((destination['id'], "Destination " + str(destination['id'])))

    data = {

```

```

        'form': form,
        'sources': sources_id,
        'destinations': destinations_id,
        'button_name': 'Save relation',
    }
    session.sign_out()
    return render(request, 'silur/create_relation.html', data)

```

```

def create_source(request):
    session = restore_session(request)
    if request.method == 'POST':
        form = SourceForm(request.POST)
        # print(form.data)
        # print(request.POST)
        if form.is_valid():
            data = form.cleaned_data
            session.create_source(data['connector_id'], data['channel_name'])
            session.sign_out()
            return HttpResponseRedirect('/silur/connectors/')
        else:
            form = SourceForm()
    server_connectors = session.get_connectors()
    # server_connectors = {'connectors': [{'id': 8, 'meta_info': {'state': 'activated'}, 'type': 'telegram'}],
    #                      'description': 'The server sent your connectors', 'status': 'ok'}
    connectors_id = []
    for connector in server_connectors['connectors']:
        connectors_id.append((connector['id'], "Connector " + str(connector['id'])))

    data = {
        'form': form,
        'button_name': 'Save source',
        'connectors': connectors_id
    }

    return render(request, 'silur/create_source.html', data)

```

```
def create_connector(request):
```

```

    if request.method == 'POST':
        form = ConnectorForm(request.POST)
        print(form.data)
        print(request.POST)
        if form.is_valid():
            session = restore_session(request)
            sessions[request.session.session_key] = session
            response = session.create_connector_command()
            if response['status'] == 'need value':
                return HttpResponseRedirect('/silur/' + response['value name'] + '/')
            else:
                session.sign_out()
                del sessions[request.session.session_key]
        else:
            form = ConnectorForm()

    data = {
        'form': form,
        'button_name': 'Create connector'
    }

    return render(request, 'silur/form_template.html', data)

```

"""djangoProject URL Configuration

The 'urlpatterns' list routes URLs to views. For more information please see:

<https://docs.djangoproject.com/en/3.2/topics/http/urls/>

Examples:

Function views

1. Add an import: from my\_app import views
2. Add a URL to urlpatterns: path("", views.home, name='home')

Class-based views

1. Add an import: from other\_app.views import Home
2. Add a URL to urlpatterns: path("", Home.as\_view(), name='home')

Including another URLconf