МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Московский авиационный институт

(национальный исследовательский университет)

Институт № 8

Компьютерные науки и прикладная математика

Кафедра 806 «Компьютерная математика»

КУРСОВАЯ РАБОТА

по дисциплине «Криптография»

на тему: Разработка криптографической системы «LUC и MAGENTA»

Выполнил: студент группы М8О-311Б-20

Иваненков Лев Михайлович

(Фамилия, имя, отчество)

(подпись)

Принял: доцент кафедры 806

Романенков Александр Михайлович

(Фамилия, имя, отчество)

(подпись)

Оценка: **Дата**:

Москва, 2023

Содержание

1 Техническое задание

1. Реализовать асимметричный алгоритм шифрования.
2. Реализовать симметричный алгоритм шифрования.
3. Реализовать приложение (оконное или web), позволяющее:
   (a) Генерировать сеансовый ключ симметричного алгоритма;
   (b) Генерировать ключи асимметричного алгоритма в целях распределения между сторонами, участвующими в обмене данными, сеансового ключа (простые числа, требуемые при генерации ключей, должны иметь в битовом представлении размер не менее 64 бит;
   (c) Генерировать вектор инициализации (IV) для его применения в режимах шифрования: CBC, CFB, OFB, CTR, RD, RD+H;
   (d) Асинхронно и многопоточно (по возможности) шифровать файл распределённым между сторонами сеансовым ключом (с использованием IV при режиме шифрования, отличном от ECB)
   (e) Асинхронно и многопоточно (по возможности) дешифровать переданный зашифрованный файл распределённым между сторонами сеансовым ключом (с использованием IV при режиме шифрования, отличном от ECB), с избавлением от набивки (padding);
   (f) Опционально: отменить операцию [де]шифрования/скачивания/загрузки по запросу пользователя.

Передача файлов должна быть организована при помощи сервера, на который можно отправить зашифрованный файл и скачать его. На/С сервер(а) одновременно можно отправлять/скачивать произвольное количество файлов. Структура файлов произвольна (текст, изображения, видео, аудио, etc.). Количество клиентских приложений, подключаемых к серверному, произвольно. Для симметричного алгоритма используйте тип набивки (padding) PKCS7.

2 Руководство пользователя

При запуске программы выводится окно с двумя кнопками для создания новой сессии или подключения к существующей.

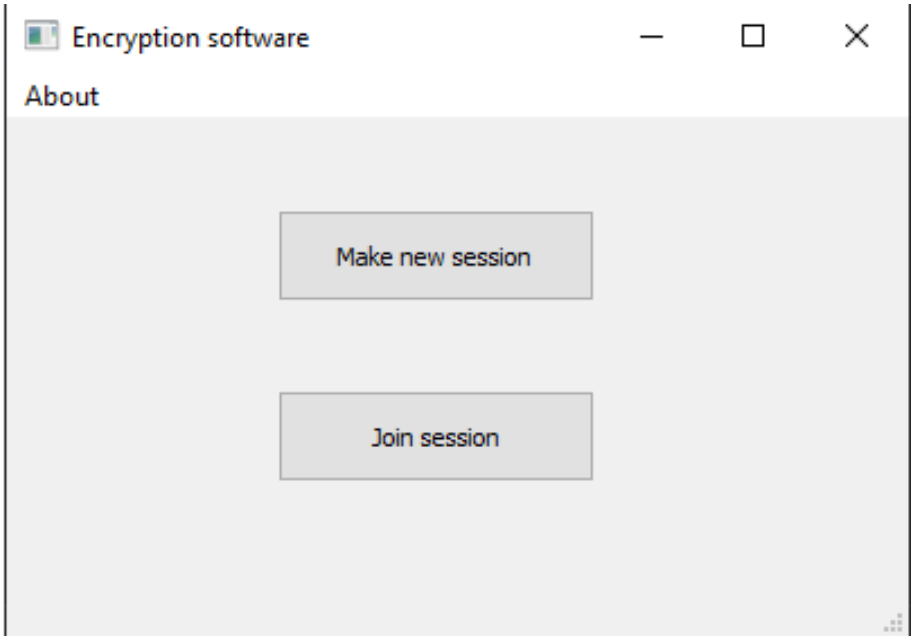

Рис. 1: Окно подключения/создания сессии

При нажатие на кнопки открывается окно для ввода логина и пароля сессии:

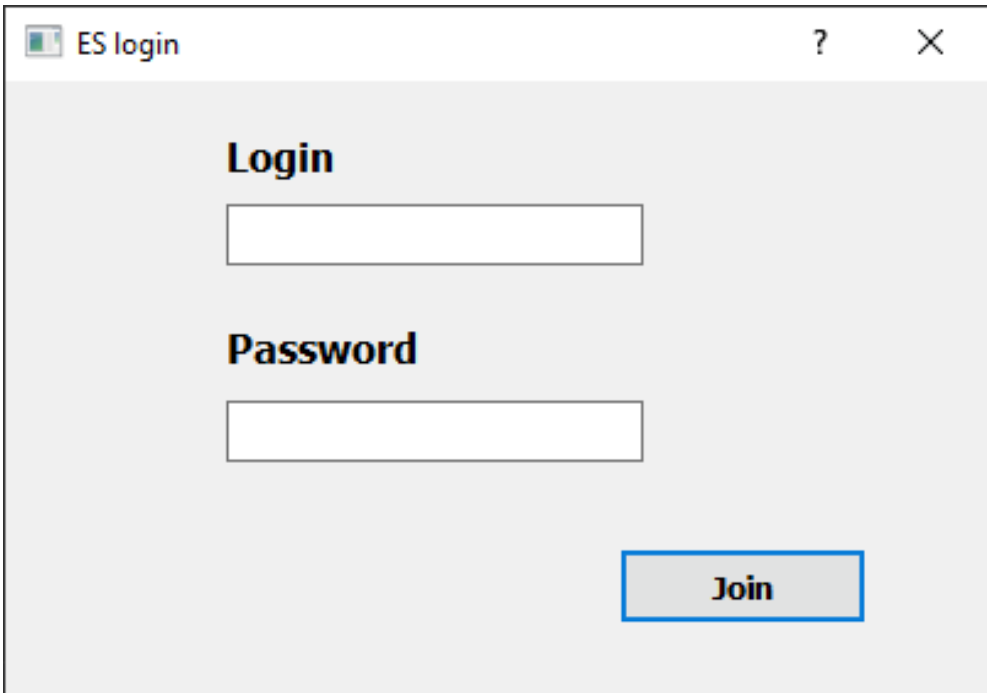

Рис. 2: Окно входа в сессию

После положительной авторизации окно входа в сессию закрывается, но также можно создавать новые сессии.

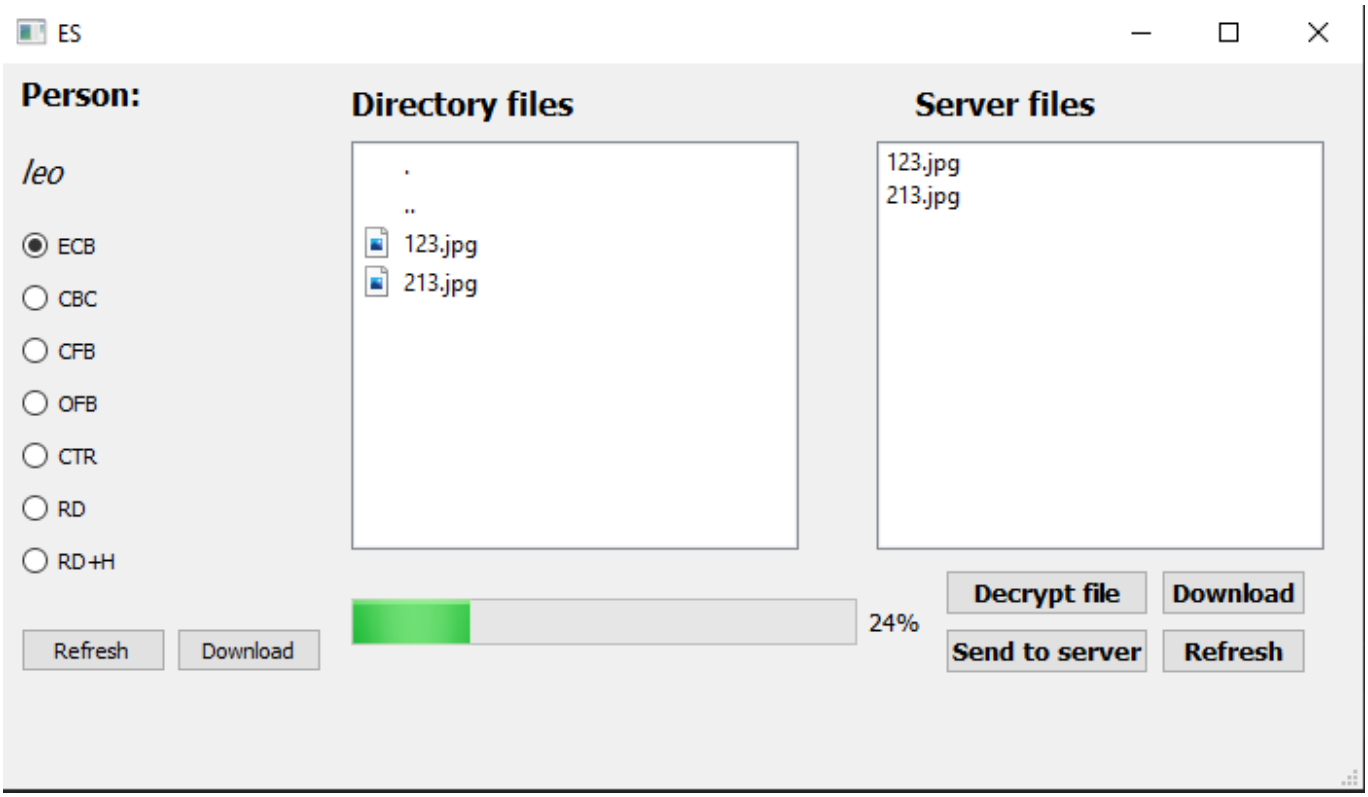

Рис. 3: Окно входа в сессию

После авторизации открывается окно приложения с основным функционалом приложения:

1. Отправление файлов на сервер по открытому каналу в папку текущего пользователя

2. Скачивание файлов с сервера текущего пользователя

Также можно выбрать режим шифрования, представленные в колонке в левой части окна.

Для выбора файла для отправки на сервер, представлен список файлов 'Directory files' с возможностью перемещениями по директориям.

Для отпраки файла на сервер нужно выделить файл и нажать кнопку 'Send to server'. После этого произойдет генерация сеансового ключа симметричного алгоритма и асимметричного в целях распределения между клиентом и сервером, участвующими в обмене данными. Далее файл зашифруется при помощи алгоритма MAGENTA и

отправится по открытому каналу на сервер. После получения сервером файла, он расшифруется и появится в списке файлов 'Server files'.

Для скачивания файла с сервера нужно выделить файл в списке 'Server files' и нажать кнопку 'Download'.

3 Структура сервиса

Сервис разделен на клиентскую и серверную части, также разработан модуль для шифрования.

Клиентская часть (EncryptionSoftware) отвечает за взаимодействие с пользователем: для отправки/скачивания файлов, выборов режимов шифрования, а также для генерации ключей и шифровании на стороне пользователя.

Серверная часть (EncryptionServer) отвечает за хранение файлов пользователя на сервере, а также для генерации ключей и шифровании на стороне сервера.

В модуле для шифрования реализованы алгоритмы LUC - для генерации ключей асимметричного алгоритма в целях распределения между сторонами, участвующими в обмене данными, сеансового ключа и MAGENTA - для шифрования файлов для передачи их по открытому каналу связи в зашифрованном виде.

3.1 EncryptionSoftware

Для создания приложения клиента используется python и QT.

1. **class Sessions**

   Данный класс наследуется от QMainWindow и используется для создания подключений к серверу.

2. **class Account**

   Данный класс наследуется от QDialog и используется для генерации диалогового окна для регистрации нового пользователя и создания подключения с сервером.

3. **class LogNew**

   Данный класс наследуется от QMainWindow и используется для отправления файлов на сервер по открытому каналу в папку текущего пользователя и скачивание файлов с сервера текущего пользователя.

3.2 EncryptionServer

Для создания сервера используется веб-фреймворк FastAPI и база данных SQLite.

1. **app.py**

   В данном файле описана основная логика работы брейкпоинтов.

2. **methods.py**

   В данном файле реализованы вспомогательные функции для работы с входными и выходными данными.

3. **db_connect.py**

   В данном вспомогательном файле реализовано подключение к базе данных.

4. **db_models.py**

   В данном вспомогательном файле описаны модели для десериализация данных получаемых из запросов в базу данных.

5. **settings.py**

   В данном вспомогательном файле сохранены основные константы.

6. **requirments.txt**

   В данном файле описан список внешних зависимостей.

7. **encsoft.db**

   Файл базы данных SQLite.

3.3 Алгоритмы шифрования

3.3.1 LUC

LUC (LUCifer) – это алгоритм шифрования с открытым ключом, который был разработан в 1990 году Чарльзом Ли, Джеймсом Масоном и Робертом Ньюманом.

Шифрование LUC основано на трудности вычисления дискретного логарифма в конечном поле. Алгоритм использует два простых числа p и q таких, что q делит p-1, а также генератор g, являющийся элементом поля Fp. Публичный ключ состоит из трех параметров – p, q и g. Для шифрования сообщения M длиной не более q-1 бита.

Алгоритм LUC обладает надежностью, эквивалентной надежности других алгоритмов с открытым ключом, но требует более длительного времени для зашифровки и расшифровки сообщений.

encryption.py

В данном файле определен класс LUCKey - отвечающий за генерацию ключей для алгоритма шифрования LUC.

Также опрделен класс LUC, в котором реализован алгоритмы шифрования и дешифрования данных.

3.3.2 MAGENTA

MAGENTA - это алгоритм блочного шифрования с переменной длиной ключа, который использует 64-битные блоки и имеет общую длину ключа от 128 до 256 бит.

Алгоритм MAGENTA состоит из нескольких раундов, каждый из которых включает в себя следующие шаги:

1. Замена - каждый байт в блоке заменяется на другой байт из некоторой таблицы подстановки.

2. Перестановка - байты блока переставляются в соответствии с некоторой таблицей перестановки.

3. XOR - блок ксорится с частью ключа.

4. Шаг смешивания - результат ксора проходит через сложную процедуру смешивания, включающую в себя умножение, сложение и использование таблиц подстановки.

encryption.py

В данном файле определен интерфейс Magenta в котором определены дефолтные методы для операций с блоками и абстрактные методы для шифрования и дешифрования.

Также определен класс-контекст EncryptMode, предоставляющий объектный функционал по выполнению шифрования и дешифрования симметричным алгоритмом с поддержкой одного из режимов шифрования (задаётся перечислением): ECB, CBC, CFB, OFB, CTR, RD, RD+H.

4 Архитектура системы

Данная система построена в архитектурном стиле RESTful сервисов. Приложение разделено на серверную и клиентскую части.

4.1 Сервер

Основная работа сервера - обработка Http запросов, а также взаимодействие с базой данных.

4.1.1 Основные эндпоинты

```
@app.get("/filenames/{user_id}", status_code=status.HTTP_200_OK)
async def get_file_names(user_id: str, db: Session =
 ↪  Depends(get_db)):
```

GET-запрос для выдачи списка файлов принадлежащих пользователя на сервере.

```
@app.get("/key_asymmetric/{username}",
 ↪  status_code=status.HTTP_200_OK)
def get_asymmetric_key(
        response: Response,
        username: str,
        db: Session = Depends(get_db)
):
```

GET-запрос для отправки открытой части ассиметричного ключа.

```
@app.post("/key/{username}", status_code=status.HTTP_200_OK)
async def get_keys(
        response: Response,
        username: str,
        key: Optional[str] = None,
        c_0: Optional[str] = None,
        db: Session = Depends(get_db)
):
```

POST-запрос для отправки зашифрованного симметричного ключа на сервер.

```
@app.post("/login/{username}", status_code=status.HTTP_200_OK)
async def login_user(
        response: Response,
        username: str,
        password: Optional[str] = None,
        db: Session = Depends(get_db)):
```

POST-запрос для входа или регистрации нового пользователя в системе.

```
@app.post("/file/upload/{user_id}", status_code=status.HTTP_200_OK)
async def upload_file(
        response: Response,
        user_id: Optional[str] = None,
        cypher_type: Optional[str] = None,
        file: UploadFile = File(...),
        db: Session = Depends(get_db)
):
```

POST-запрос для отправки зашифрованного файла на сервер.

```
@app.get("/file/download/{user_id}", status_code=status.HTTP_200_OK)
async def download_file(
        response: Response,
        file_name: str,
        user_id: str,
        db: Session = Depends(get_db)
):
```

GET-запрос для скачивания зашифрованного файла с сервера.

4.1.2 Взаимодействие с базой данных

Для взаимодействия с базой данных используется фреймворк SQLAlchemy при помощи котрого были определены ORM модели для всез таблиц: User, Files, данные

таблицы имеют связь один ко многим соответственно и отвечают за хранение данных о пользователе и файла на сервере.

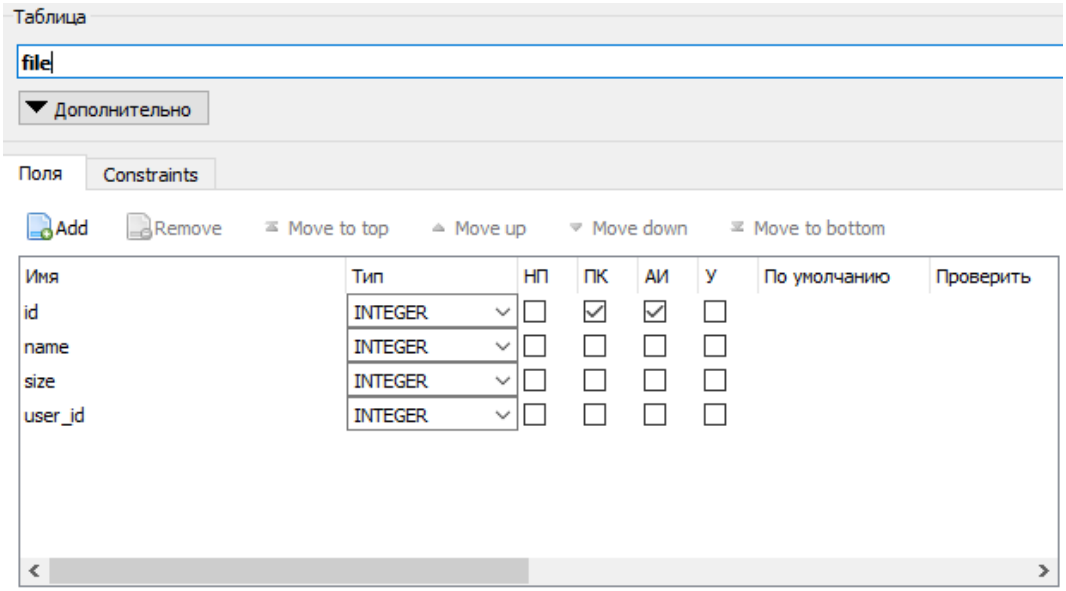

Рис. 4: Таблица File

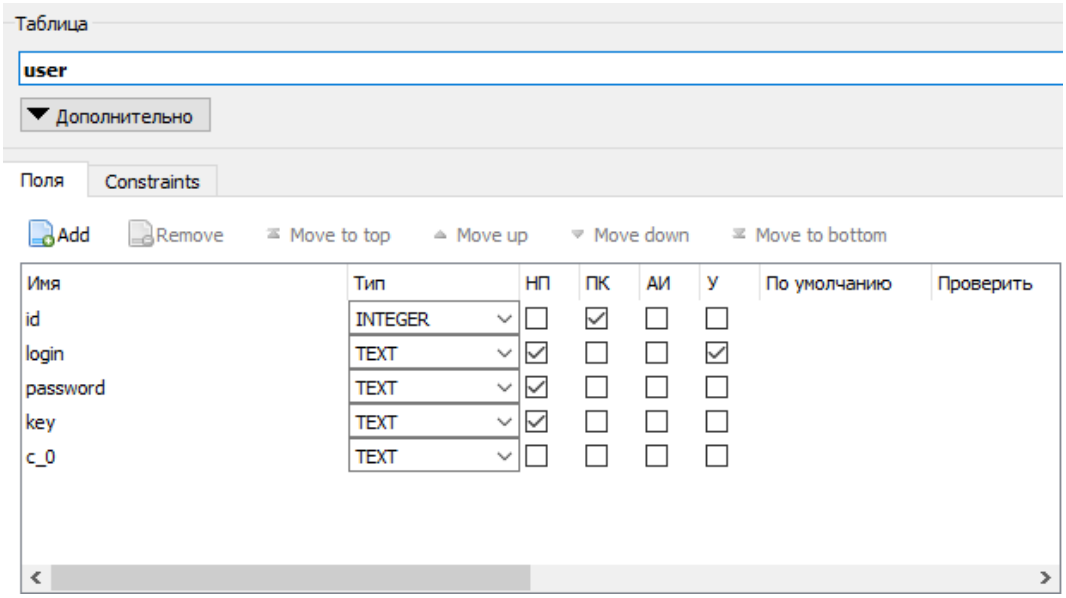

Рис. 5: Таблица User

4.2 Клиент

В приложение клиента всего 3 окна: создание сессий, регистрации или входа пользователя и окно управление файлами.

При открытии приложения открывается окно создания сессий – оно является родительским ко всем остальным, поэтому будет открыто всегда, пока работает приложение. Далее при регистрации новой сессии окрывается диалоговое окно с вводом логина и пароля, которое закроется при успешной авторизации. Праллельно может быть открытно несколько окон регистрации и окон управления файлами.

Для удобства папки пользователей, которые открываются в программе клиента находятся в папке с проектом программы: каждый пользователь имее свой уникальный id и его папка названа этим числом.

5 Список использованных источников

1. Лутц М. Изучаем Python, 4-е издание. – Пер. с англ. – СПб.:Символ-Плюс, 2011. -– 1280 с.

2. Златопольский Д.М. Основы программирования на языке Python. – М.: ДМК Пресс, 2017. — 284 с.

3. Земор, Ж. Курс криптографии / Ж. Земор. – М.: Регулярная и хаотическая динамика, Институт компьютерных исследований, 2006. - 256 с.

6 Приложение

account:

```
1
2  import os
3
4  from PyQt5 import QtCore, QtGui, QtWidgets
5  # from PyQt5.QtCore import *
6  from PyQt5.QtGui import *
7  # from PyQt5.QtWidgets import *
8  from PyQt5.QtWidgets import QApplication, QMainWindow, QFileSystemModel, QTableView, QMessageBox
9  from PyQt5.QtCore import QDir
10 from encryptmode import ECB, OFB, CBC, CFB, EncryptMode, LUC
11 import sys
12 import requests
13 from variables import SERVER_ADDRESS
14 import json
15 import random
16 import string
17
18
19 def randomword(length):
20     letters = string.ascii_lowercase
21     return ''.join(random.choice(letters) for i in range(length))
22
23
24 class Account(QMainWindow):
25     def __init__(self, login: str = '', user_id: str = '1', parent=None):
26         super(Account, self).__init__(parent)
27         self.encryption_type = None
28         self.chosen_file = None
29         self.chosen_serv_file = None
30         self.user_id = user_id
31         self.login = login
32         self.key = None
33         self.key_crypt = ''
34         self.c_0 = None
35         self.generate_keys()
36
37         self.user_dir = f'file_to_send/{self.user_id}/'
38         self.file_model = QFileSystemModel(self)
39         self.serv_files_model = QStandardItemModel(self)
40
41         self.statusbar = QtWidgets.QStatusBar(self)
42         self.menubar = QtWidgets.QMenuBar(self)
```

```
43
44          self.centralwidget = QtWidgets.QWidget(self)
45          self.widget = QtWidgets.QWidget(self.centralwidget)
46          self.widget1 = QtWidgets.QWidget(self.centralwidget)
47
48          self.gridLayout = QtWidgets.QGridLayout(self.widget)
49          self.gridLayout_2 = QtWidgets.QGridLayout(self.widget1)
50
51          self.pb_refresh = QtWidgets.QPushButton(self.widget1)
52          self.pb_send = QtWidgets.QPushButton(self.widget1)
53          self.pb_download = QtWidgets.QPushButton(self.widget1)
54          self.pb_decrypt = QtWidgets.QPushButton(self.widget1)
55          self.progressBar = QtWidgets.QProgressBar(self.widget1)
56
57          self.rb_rdh = QtWidgets.QRadioButton(self.widget)
58          self.rb_rd = QtWidgets.QRadioButton(self.widget)
59          self.rb_ctr = QtWidgets.QRadioButton(self.widget)
60          self.rb_ofb = QtWidgets.QRadioButton(self.widget)
61          self.rb_cfb = QtWidgets.QRadioButton(self.widget)
62          self.rb_cbc = QtWidgets.QRadioButton(self.widget)
63          self.rb_ecb = QtWidgets.QRadioButton(self.widget)
64          self.enc_mode = None
65          self.encrypter = None
66
67          self.label_3 = QtWidgets.QLabel(self.centralwidget)
68          self.label_2 = QtWidgets.QLabel(self.centralwidget)
69
70          self.lv_files_server = QtWidgets.QListView(self.centralwidget)
71          self.lv_files = QtWidgets.QListView(self.centralwidget)
72
73          self.lb_login = QtWidgets.QLabel(self.centralwidget)
74          self.lb_login.setText(login)
75          self.label = QtWidgets.QLabel(self.centralwidget)
76
77          self.setup_ui()
78          self.add_functions()
79          self.add_directories()
80
81      def setup_ui(self):
82          self.setObjectName("MainWindow")
83          self.resize(700, 373)
84          sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Fixed)
85          sizePolicy.setHorizontalStretch(0)
86          sizePolicy.setVerticalStretch(0)
87          sizePolicy.setHeightForWidth(self.sizePolicy().hasHeightForWidth())
88          self.setSizePolicy(sizePolicy)
89          self.centralwidget.setObjectName("centralwidget")
```

```
90          self.lv_files.setGeometry(QtCore.QRect(180, 40, 230, 210))
91          self.lv_files.setObjectName("lv_files")
92          self.label.setGeometry(QtCore.QRect(10, 0, 71, 31))
93          font = QtGui.QFont()
94          font.setPointSize(12)
95          font.setBold(True)
96          font.setWeight(75)
97          self.label.setFont(font)
98          self.label.setObjectName("label")
99          self.lb_login.setGeometry(QtCore.QRect(10, 30, 81, 51))
100         font = QtGui.QFont()
101         font.setPointSize(12)
102         font.setItalic(True)
103         self.lb_login.setFont(font)
104         self.lb_login.setObjectName("lb_login")
105         self.lv_files_server.setGeometry(QtCore.QRect(450, 40, 230, 210))
106         self.lv_files_server.setObjectName("lv_files_server")
107         self.pushButton = QtWidgets.QPushButton(self.centralwidget)
108         self.pushButton.setGeometry(QtCore.QRect(10, 290, 75, 23))
109         self.pushButton.setObjectName("pushButton")
110         self.pushButton_2 = QtWidgets.QPushButton(self.centralwidget)
111         self.pushButton_2.setGeometry(QtCore.QRect(90, 290, 75, 23))
112         self.pushButton_2.setObjectName("pushButton_2")
113         self.label_2.setGeometry(QtCore.QRect(470, 10, 111, 21))
114         font = QtGui.QFont()
115         font.setPointSize(12)
116         font.setBold(True)
117         font.setWeight(75)
118         self.label_2.setFont(font)
119         self.label_2.setObjectName("label_2")
120
121         self.label_3.setGeometry(QtCore.QRect(180, 10, 161, 21))
122         font = QtGui.QFont()
123         font.setPointSize(12)
124         font.setBold(True)
125         font.setWeight(75)
126         self.label_3.setFont(font)
127         self.label_3.setObjectName("label_3")
128         self.widget.setGeometry(QtCore.QRect(11, 81, 121, 191))
129         self.widget.setObjectName("widget")
130         self.gridLayout.setContentsMargins(0, 0, 0, 0)
131         self.gridLayout.setObjectName("gridLayout")
132         self.rb_ecb.setEnabled(True)
133         self.rb_ecb.setChecked(True)
134         self.rb_ecb.setObjectName("rb_ecb")
135         self.gridLayout.addWidget(self.rb_ecb, 0, 0, 1, 1)
136         self.rb_cbc.setObjectName("rb_cbc")
```

```
137         self.gridLayout.addWidget(self.rb_cbc, 1, 0, 1, 1)
138         self.rb_cfb.setObjectName("rb_cfb")
139         self.gridLayout.addWidget(self.rb_cfb, 2, 0, 1, 1)
140         self.rb_ofb.setObjectName("rb_ofb")
141         self.gridLayout.addWidget(self.rb_ofb, 3, 0, 1, 1)
142         self.rb_ctr.setObjectName("rb_ctr")
143         self.gridLayout.addWidget(self.rb_ctr, 4, 0, 1, 1)
144         self.rb_rd.setObjectName("rb_rd")
145         self.gridLayout.addWidget(self.rb_rd, 5, 0, 1, 1)
146         self.rb_rdh.setObjectName("rb_rdh")
147         self.gridLayout.addWidget(self.rb_rdh, 6, 0, 1, 1)
148         self.widget1.setGeometry(QtCore.QRect(180, 260, 491, 56))
149         self.widget1.setObjectName("widget1")
150         self.gridLayout_2.setContentsMargins(0, 0, 0, 0)
151         self.gridLayout_2.setObjectName("gridLayout_2")
152         font = QtGui.QFont()
153         font.setPointSize(10)
154         self.progressBar.setFont(font)
155         self.progressBar.setProperty("value", 24)
156         self.progressBar.setObjectName("progressBar")
157         self.gridLayout_2.addWidget(self.progressBar, 0, 0, 2, 1)
158         font = QtGui.QFont()
159         font.setPointSize(10)
160         font.setBold(True)
161         font.setWeight(75)
162         self.pb_decrypt.setFont(font)
163         self.pb_decrypt.setObjectName("pb_decrypt")
164         self.gridLayout_2.addWidget(self.pb_decrypt, 0, 1, 1, 1)
165         font = QtGui.QFont()
166         font.setPointSize(10)
167         font.setBold(True)
168         font.setWeight(75)
169         self.pb_download.setFont(font)
170         self.pb_download.setObjectName("pb_download")
171         self.gridLayout_2.addWidget(self.pb_download, 0, 2, 1, 1)
172         font = QtGui.QFont()
173         font.setPointSize(10)
174         font.setBold(True)
175         font.setWeight(75)
176         self.pb_send.setFont(font)
177         self.pb_send.setObjectName("pb_encrypt")
178         self.gridLayout_2.addWidget(self.pb_send, 1, 1, 1, 1)
179         font = QtGui.QFont()
180         font.setPointSize(10)
181         font.setBold(True)
182         font.setWeight(75)
183         self.pb_refresh.setFont(font)
```

```
184         self.pb_refresh.setObjectName("pb_refresh")
185         self.gridLayout_2.addWidget(self.pb_refresh, 1, 2, 1, 1)
186         self.setCentralWidget(self.centralwidget)
187         self.menubar.setGeometry(QtCore.QRect(0, 0, 700, 21))
188         self.menubar.setObjectName("menubar")
189         self.setMenuBar(self.menubar)
190         self.statusbar.setObjectName("statusbar")
191         self.setStatusBar(self.statusbar)
192 
193         self.translate_ui()
194         QtCore.QMetaObject.connectSlotsByName(self)
195 
196     def translate_ui(self):
197         _translate = QtCore.QCoreApplication.translate
198         self.setWindowTitle(_translate("MainWindow", "ES"))
199         self.label.setText(_translate("MainWindow", "Person:"))
200         # self.lb_login.setText(_translate("MainWindow", "Leo"))
201         self.label_2.setText(_translate("MainWindow", "Server files"))
202         self.label_3.setText(_translate("MainWindow", "Directory files"))
203         self.pushButton.setText(_translate("MainWindow", "Refresh"))
204         self.pushButton_2.setText(_translate("MainWindow", "Download"))
205         self.rb_ecb.setText(_translate("MainWindow", "ECB"))
206         self.rb_cbc.setText(_translate("MainWindow", "CBC"))
207         self.rb_cfb.setText(_translate("MainWindow", "CFB"))
208         self.rb_ofb.setText(_translate("MainWindow", "OFB"))
209         self.rb_ctr.setText(_translate("MainWindow", "CTR"))
210         self.rb_rd.setText(_translate("MainWindow", "RD"))
211         self.rb_rdh.setText(_translate("MainWindow", "RD+H"))
212         self.pb_decrypt.setText(_translate("MainWindow", "Decrypt file"))
213         self.pb_download.setText(_translate("MainWindow", "Download"))
214         self.pb_send.setText(_translate("MainWindow", "Send to server"))
215         self.pb_refresh.setText(_translate("MainWindow", "Refresh"))
216 
217     def generate_keys(self):
218         self.key = Account.generate_key()
219         print(self.key)
220         self.c_0 = Account.generate_key()
221         print(self.key)
222         self.encrypt_symmetric_key()
223         try:
224             resp = requests.post(f"{SERVER_ADDRESS}/key/{self.login}?key={self.key}&c_0={self.c_0}")
225         except requests.ConnectionError:
226             message = QMessageBox()
227             message.setWindowTitle("Error connection")
228             message.setText("Can not connect to server")
229             message.setIcon(QMessageBox.Warning)
230             message.exec_()
```

```
231
232     def encrypt_symmetric_key(self):
233         try:
234             resp = requests.get(f"{SERVER_ADDRESS}/key_asymmetric/{self.login}")
235             print(resp.json())
236
237             for ent in self.key:
238                 # print(LUC.encrypt_num(ord(ent), resp.json()['e'], resp.json()['n']))
239                 self.key_crypt += f"{LUC.encrypt_num(ord(ent), resp.json()['e'], resp.json()['n'])}"
240             print(self.key_crypt)
241
242         except requests.ConnectionError:
243             message = QMessageBox()
244             message.setWindowTitle("Error connection")
245             message.setText("Can not connect to server")
246             message.setIcon(QMessageBox.Warning)
247             message.exec_()
248
249     def add_functions(self):
250         self.pb_refresh.clicked.connect(self.refresh_tables)
251         self.pb_decrypt.clicked.connect(self.decrypt)
252         self.pb_send.clicked.connect(self.send_to_server)
253         self.pb_download.clicked.connect(self.download)
254         self.lv_files.doubleClicked[QtCore.QModelIndex].connect(self.double_clicked_table)
255         self.lv_files.clicked[QtCore.QModelIndex].connect(self.clicked_table)
256         self.lv_files_server.clicked[QtCore.QModelIndex].connect(self.clicked_serv_files)
257
258     def add_directories(self):
259         self.file_model.setFilter(QDir.AllEntries)
260         self.file_model.setRootPath("C:/Users/leo/PycharmProjects/EncryptionSoftware")
261         self.lv_files.setModel(self.file_model)
262         self.lv_files.setRootIndex(
263
              ↪  self.file_model.index(f"C:/Users/leo/PycharmProjects/EncryptionSoftware/file_to_send/{self.us
264         try:
265             resp = requests.get(f"{SERVER_ADDRESS}/filenames/{self.user_id}")
266             print(json.dumps(resp.json()))
267             for ent in resp.json():
268                 item = QStandardItem(ent['name'])
269                 self.serv_files_model.appendRow(item)
270             self.lv_files_server.setModel(self.serv_files_model)
271         except requests.ConnectionError:
272             message = QMessageBox()
273             message.setWindowTitle("Error connection")
274             message.setText("Can not connect to server")
275             message.setIcon(QMessageBox.Warning)
276             message.exec_()
```

```
277
278     def double_clicked_table(self, index):
279         item = self.file_model.fileInfo(index)
280         if item.fileName() == "..":
281             directory = item.dir()
282             directory.cdUp()
283             self.lv_files.setRootIndex(self.file_model.index(directory.absolutePath()))
284         elif item.fileName() == ".":
285             self.lv_files.setRootIndex(self.file_model.index(""))
286         elif item.isDir():
287             self.lv_files.setRootIndex(index)
288
289     def clicked_serv_files(self, index):
290         self.chosen_serv_file = self.lv_files_server.currentIndex().data()
291         print(self.chosen_serv_file)
292
293     def clicked_table(self, index):
294         self.chosen_file = self.lv_files.currentIndex().data()
295         print(self.chosen_file)
296
297     @staticmethod
298     def generate_key():
299         letters = string.ascii_lowercase
300         return ''.join(random.choice(letters) for i in range(16))
301
302     def refresh_tables(self):
303         try:
304             resp = requests.get(f"{SERVER_ADDRESS}/filenames/{self.user_id}")
305             self.serv_files_model.clear()
306             print(json.dumps(resp.json()))
307             for ent in resp.json():
308                 item = QStandardItem(ent['name'])
309                 self.serv_files_model.appendRow(item)
310         except requests.ConnectionError:
311             message = QMessageBox()
312             message.setWindowTitle("Error connection")
313             message.setText("Can not connect to server")
314             message.setIcon(QMessageBox.Warning)
315             message.exec_()
316
317     def decrypt(self):
318         pass
319         # TODO: Расшифровка файла
320
321     def rb_checked(self):
322         if self.rb_rdh.isChecked():
323             self.encrypter = ECB(self.key.encode())
```

```
            self.encryption_type = 'ECB'
        elif self.rb_rd.isChecked():
            self.encrypter = ECB(self.key.encode())
            self.encryption_type = 'ECB'
        elif self.rb_ecb.isChecked():
            self.encrypter = ECB(self.key.encode())
            self.encryption_type = 'ECB'
        elif self.rb_cbc.isChecked():
            self.encrypter = CBC(self.key.encode(), self.c_0.encode())
            self.encryption_type = 'CBC'
        elif self.rb_cfb.isChecked():
            self.encrypter = CFB(self.key.encode(), self.c_0.encode())
            self.encryption_type = 'CFB'
        elif self.rb_ctr.isChecked():
            self.encrypter = ECB(self.key.encode())
            self.encryption_type = 'ECB'
        elif self.rb_ofb.isChecked():
            self.encrypter = OFB(self.key.encode(), self.c_0.encode())
            self.encryption_type = 'OFB'

    def send_to_server(self):
        self.rb_checked()
        if not self.chosen_file:
            message = QMessageBox()
            message.setWindowTitle("Error file click")
            message.setText("Choose the file!")
            message.setIcon(QMessageBox.Warning)
            message.exec_()
        else:
            if os.path.exists(self.user_dir + self.chosen_file):
                try:
                    with open(self.user_dir + self.chosen_file, 'rb') as f:
                        open_text = f.read()

                    enc = self.encrypter.encode(open_text)
                    with open(f"temp/{self.chosen_file}", 'wb') as f:
                        f.write(enc)
                    resp =
                    ↪  requests.post(f"{SERVER_ADDRESS}/file/upload/{self.user_id}?cypher_type={self.encrypt
                                     files={'file': open(f"temp/{self.chosen_file}", 'rb')})
                    os.remove(f"temp/{self.chosen_file}")
                except requests.ConnectionError:
                    message = QMessageBox()
                    message.setWindowTitle("Error connection")
                    message.setText("Can not connect to server")
                    message.setIcon(QMessageBox.Warning)
                    message.exec_()
```

```
370
371             else:
372                 message = QMessageBox()
373                 message.setWindowTitle("Error file")
374                 message.setText("File is not exists")
375                 message.setIcon(QMessageBox.Warning)
376                 message.exec_()
377         self.chosen_file = None
378         self.refresh_tables()
379
380     def download(self):
381         if not self.chosen_serv_file:
382             message = QMessageBox()
383             message.setWindowTitle("Error file click")
384             message.setText("Choose the file!")
385             message.setIcon(QMessageBox.Warning)
386             message.exec_()
387         else:
388             try:
389                 resp =
                    ↪  requests.get(f"{SERVER_ADDRESS}/file/download/{self.user_id}?file_name={self.chosen_serv_
390                 if resp.status_code == 200 or resp.status_code == 201:
391                     if not os.path.exists(f'{self.user_dir}'):
392                         os.makedirs(f'{self.user_dir}')
393                     with open(self.user_dir + self.chosen_serv_file, 'wb') as f:
394                         f.write(resp.content)
395                 else:
396                     message = QMessageBox()
397                     message.setWindowTitle("Error file")
398                     message.setText("File does not exist on server!")
399                     message.setIcon(QMessageBox.Warning)
400                     message.exec_()
401             except requests.ConnectionError:
402                 message = QMessageBox()
403                 message.setWindowTitle("Error connection")
404                 message.setText("Can not connect to server")
405                 message.setIcon(QMessageBox.Warning)
406                 message.exec_()
407
408         self.chosen_serv_file = None
409         self.refresh_tables()
410
411
412 if __name__ == '__main__':
413     app = QApplication(sys.argv)
414     window = Account()
415     window.show()
```

```
416     sys.exit(app.exec_())
417
```

sessions.py:

```
1  from PyQt5 import QtCore, QtGui, QtWidgets
2 from PyQt5.QtWidgets import QDialog, QApplication, QMainWindow
3 from log_new_session import LogNew
4 import sys
5
6
7 class Sessions(QMainWindow):
8     def __init__(self, parent=None):
9         super(Sessions, self).__init__(parent)
10        self.setObjectName("MainWindow")
11        self.resize(400, 250)
12        self.centralwidget = QtWidgets.QWidget(self)
13        self.centralwidget.setObjectName("centralwidget")
14        self.pb_new_session = QtWidgets.QPushButton(self.centralwidget)
15        self.pb_new_session.setGeometry(QtCore.QRect(120, 40, 141, 41))
16        self.pb_new_session.setObjectName("pb_new_session")
17        self.pb_join_session = QtWidgets.QPushButton(self.centralwidget)
18        self.pb_join_session.setGeometry(QtCore.QRect(120, 120, 141, 41))
19        self.pb_join_session.setObjectName("pb_join_session")
20        self.setCentralWidget(self.centralwidget)
21        self.menubar = QtWidgets.QMenuBar(self)
22        self.menubar.setGeometry(QtCore.QRect(0, 0, 400, 21))
23        self.menubar.setObjectName("menubar")
24        self.menuAbout = QtWidgets.QMenu(self.menubar)
25        self.menuAbout.setObjectName("menuAbout")
26        self.setMenuBar(self.menubar)
27        self.statusbar = QtWidgets.QStatusBar(self)
28        self.statusbar.setObjectName("statusbar")
29        self.setStatusBar(self.statusbar)
30        self.menubar.addAction(self.menuAbout.menuAction())
31
32        self.retranslateUi()
33        QtCore.QMetaObject.connectSlotsByName(self)
34        self.add_functions()
35
36    def retranslateUi(self):
37        _translate = QtCore.QCoreApplication.translate
38        self.setWindowTitle(_translate("MainWindow", "Encryption software"))
39        self.pb_new_session.setText(_translate("MainWindow", "Make new session"))
40        self.pb_join_session.setText(_translate("MainWindow", "Join session"))
```

```
41          self.menuAbout.setTitle(_translate("MainWindow", "About"))
42
43      def add_functions(self):
44          self.pb_new_session.clicked.connect(lambda: self.join_new_session())
45          self.pb_join_session.clicked.connect(lambda: self.join_new_session())
46
47      def join_new_session(self):
48          session = LogNew(self)
49          session.show()
50
51
52  if __name__ == '__main__':
53      app = QApplication(sys.argv)
54      window = Sessions()
55      window.show()
56      sys.exit(app.exec_())
57
```

---

log_new_session:

```
1  import os
2
3  from PyQt5 import QtCore, QtGui, QtWidgets
4  from PyQt5.QtWidgets import QDialog, QApplication, QMessageBox
5  from account import Account
6  import sys
7  import requests
8  from variables import SERVER_ADDRESS
9
10
11 class LogNew(QDialog):
12     def __init__(self, parent=None):
13         super(LogNew, self).__init__(parent)
14         self.setObjectName("Dialog")
15         self.resize(400, 250)
16         self.pb_join = QtWidgets.QPushButton(self)
17         self.pb_join.setGeometry(QtCore.QRect(250, 190, 101, 31))
18         font = QtGui.QFont()
19         font.setPointSize(10)
20         font.setBold(True)
21         font.setItalic(False)
22         font.setWeight(75)
23         self.pb_join.setFont(font)
24         self.pb_join.setObjectName("pb_join")
25         self.label = QtWidgets.QLabel(self)
```

```
26          self.label.setGeometry(QtCore.QRect(90, 100, 111, 16))
27          font = QtGui.QFont()
28          font.setPointSize(12)
29          font.setBold(True)
30          font.setWeight(75)
31          self.label.setFont(font)
32          self.label.setObjectName("label")
33          self.le_pass = QtWidgets.QLineEdit(self)
34          self.le_pass.setGeometry(QtCore.QRect(90, 130, 170, 25))
35          self.le_pass.setObjectName("le_pass")
36          self.label_2 = QtWidgets.QLabel(self)
37          self.label_2.setGeometry(QtCore.QRect(90, 20, 71, 21))
38          font = QtGui.QFont()
39          font.setPointSize(12)
40          font.setBold(True)
41          font.setWeight(75)
42          self.label_2.setFont(font)
43          self.label_2.setObjectName("label_2")
44          self.le_login = QtWidgets.QLineEdit(self)
45          self.le_login.setGeometry(QtCore.QRect(90, 50, 170, 25))
46          self.le_login.setObjectName("le_login")
47
48          self.retranslateUi()
49          QtCore.QMetaObject.connectSlotsByName(self)
50          self.add_functions()
51
52      def retranslateUi(self):
53          _translate = QtCore.QCoreApplication.translate
54          self.setWindowTitle(_translate("Dialog", "ES login"))
55          self.pb_join.setText(_translate("Dialog", "Join"))
56          self.label.setText(_translate("Dialog", "Password"))
57          self.label_2.setText(_translate("Dialog", "Login"))
58
59      def add_functions(self):
60          self.pb_join.clicked.connect(lambda: self.join_new_session(self.le_login.text(),
            ↪  self.le_pass.text()))
61
62      def join_new_session(self, login: str, password: str):
63          if login == '' or password == '':
64              message = QMessageBox()
65              message.setWindowTitle("Error connection")
66              message.setText("You forgot to enter your username or password")
67              message.setIcon(QMessageBox.Warning)
68              message.exec_()
69          else:
70              try:
71                  resp = requests.post(f"{SERVER_ADDRESS}/login/{login}?password={password}")
```

```
                if resp.status_code == 200:
                    print(f"User logined: {resp.text}")
                    account = Account(login, resp.text, parent=self)
                    account.show()
                    self.close()
                elif resp.status_code == 201:
                    print(f"User created: {resp.text}")
                    if not os.path.exists(f'file_to_send/{resp.text}'):
                        os.makedirs(f'file_to_send/{resp.text}')
                    account = Account(login, resp.text, parent=self)
                    account.show()
                    self.close()
                else:
                    message = QMessageBox()
                    message.setWindowTitle("Error connection")
                    message.setText("Password is incorrect")
                    message.setIcon(QMessageBox.Warning)
                    message.exec_()
            except requests.ConnectionError:
                message = QMessageBox()
                message.setWindowTitle("Error connection")
                message.setText("Can not connect to server")
                message.setIcon(QMessageBox.Warning)
                message.exec_()


if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = LogNew()
    window.show()
    sys.exit(app.exec_())
```

PrimeNumber.py:

```
import math
import random
from enum import Enum
from Symbols import Symbols


class PrimeType(Enum):
    MillerRabin = 0
```

```
    Fermat = 1
    SoloveyShtrasen = 2


class PrimeGenerator:
    def __init__(self, accuracy):
        self.accuracy = accuracy
        self.count = self.num_passe()
        print(self.count)

    @staticmethod
    def generate_prime(n):
        pass

    @staticmethod
    def generate_prime_range(start, stop):
        pass

    def num_passe(self):
        pass


class MillerRabin(PrimeGenerator):
    def __init__(self, accuracy):
        super().__init__(accuracy)

    def num_passe(self):
        n = 1
        count = 0
        while self.accuracy > 1 - n:
            n /= 4
            count += 1
        return count

    @staticmethod
    def miller_rabin_pass(a, s, d, n):
        a_to_power = pow(a, d, n)
        i = 0

        if a_to_power == 1:
            return True

        while i < s - 1:
            if a_to_power == n - 1:
                return True
            a_to_power = (a_to_power * a_to_power) % n
            i += 1
```

```
56
57          return a_to_power == n - 1
58
59      def miller_rabin(self, n):
60          d = n - 1
61          s = 0
62          while d % 2 == 0:
63              d >>= 1
64              s += 1
65          i = 1
66          while i <= self.count:
67              a = random.randrange(2, n - 1)
68              if not self.miller_rabin_pass(a, s, d, n):
69                  return False
70              i += 1
71
72          return True
73
74      def generate_prime(self, n):
75          while True:
76              p = random.getrandbits(n)
77              # force p to have nbits and be odd
78              p |= 2 ** n | 1
79              if self.miller_rabin(p):
80                  return p
81
82      def generate_prime_range(self, start, stop):
83          while True:
84              p = random.randrange(start, stop - 1)
85              p |= 1
86              if self.miller_rabin(p):
87                  return p
88
89
90  class SoloveyShtrasen(PrimeGenerator):
91      def __init__(self, accuracy):
92          super().__init__(accuracy)
93
94      def num_passe(self):
95          n = 1
96          count = 0
97          while self.accuracy > 1 - n:
98              n /= 2
99              count += 1
100         return count
101
102     def generate_prime(self, n):
```

```
        while True:
            p = random.getrandbits(n)
            p |= 2 ** n | 1
            if self.solovey_shtrasen(p):
                return p

    def generate_prime_range(self, start, stop):
        while True:
            p = random.randrange(start, stop - 1)
            p |= 1
            if self.solovey_shtrasen(p):
                return p

    @staticmethod
    def solovey_shtrasen(n):
        a = random.randint(2, n - 1)
        g = ((n - 1) // 2)
        r = pow(a, g, n)

        if math.gcd(a, n) > 1:
            return False

        if r != 1 and r != n - 1:
            return False

        if r % n != Symbols.jacobi(a, n) % n:
            return False
        return True


class Fermat(PrimeGenerator):
    def __init__(self, accuracy):
        super().__init__(accuracy)

    def num_passe(self):
        n = 1
        count = 0
        while self.accuracy > 1 - n:
            n /= 2
            count += 1
        return count

    def generate_prime(self, n):
        while True:
            p = random.getrandbits(n)
            p |= 2 ** n | 1
            if self.fermat(p):
```

```
150                 return p
151
152     def generate_prime_range(self, start, stop):
153         while True:
154             p = random.randrange(start, stop - 1)
155             p |= 1
156             if self.fermat(p):
157                 return p
158
159     def fermat(self, n):
160         for i in range(self.count):
161             g = random.randint(2, n - 1)
162             if pow(g, n - 1, n) != 1:
163                 return False
164         return True
165
```

Symbols.py:

```
1  class Symbols:
2      @staticmethod
3      def jacobi(a, n):
4          assert(n > a > 0 and n % 2 == 1)
5          t = 1
6          while a != 0:
7              while a % 2 == 0:
8                  a /= 2
9                  r = n % 8
10                 if r == 3 or r == 5:
11                     t = -t
12             a, n = n, a
13             if a % 4 == n % 4 == 3:
14                 t = -t
15             a %= n
16         if n == 1:
17             return t
18         else:
19             return 0
20
21     @staticmethod
22     def legendre_symbol(a, p):
23         a = a % p
24         if not a:
25             return 0
26         if pow(a, (p - 1) // 2, p) == 1:
```

```
            return 1
        return -1

```

encryptionmode.py:

```
from abc import ABCMeta, abstractmethod
from PrimeNumber import MillerRabin, Fermat, SoloveyShtrasen
from PrimeNumber import PrimeType as mode
import random
import math
from typing import Optional
from sympy import legendre_symbol
import numpy as np


def egcd(a, b):
    u, u1 = 1, 0
    v, v1 = 0, 1
    while b:
        q = a // b
        u, u1 = u1, u - q * u1
        v, v1 = v1, v - q * v1
        a, b = b, a - q * b
    return u, v, a


def mod_inverse(e, n):
    return egcd(e, n)[0] % n


def legendre(q, a, p):
    if a == 0:
        return 0
    if a == p:
        return 0
    if a == -p:
        return 0
    if a < 0:
        return 1
    if a != 1:
        t, j2_p, q1 = 1, 1, 0
        if a > p:
            a = a % p
        # Частные случаи Якоби
```

```
        if a == 1:
            return q
        if a == 2:
            return q * pow(-1, (p * p - 1) // 8)
        if a % 2 == 0:
            while (a // pow(2, t)) % 2 == 0:
                t += 1
            a = a // pow(2, t)
            if t % 2 != 0:
                j2_p = pow(-1, (p * p - 1) // 8)
        q1 = (pow(-1, ((p - 1) // 2) * ((a - 1) // 2))) // j2_p
        return legendre(q * q1, p, a)
    else:
        return q


def lcm(a, b):
    m = a * b
    while a != 0 and b != 0:
        if a > b:
            a %= b
        else:
            b %= a
    return m // (a + b)


class Magenta:
    def __init__(self, key: bytes):
        '''
        Constructor takes key with length 16 or 24 or 32 bytes.
        '''
        self._s = self._generate_S()
        self._key = key
        self._get_key_order(key)

    def _get_key_order(self, key: bytes):
        '''
        Takes key 16 or 24 or 32 bytes.
        Return key order array for encryption.
        '''
        key_len = len(key)
        if key_len == 16:
            k1, k2 = self._key[:8], self._key[8:]
            self._key_order = (k1, k1, k2, k2, k1, k1)

        elif key_len == 24:
            k1, k2, k3 = key[:8], key[8:16], key[16:24]
```

```
87              self._key_order = (k1, k2, k3, k3, k2, k1)
88
89          else:  # key_len == 32
90              k1, k2 = key[:8], key[8:16]
91              k3, k4 = key[16:24], key[24:32]
92              self._key_order = (k1, k2, k3, k4, k4, k3, k2, k1)
93
94      def _encode_block(self, block: bytes):
95          '''
96          Takes block 16 bytes.
97          Return encrypted block 16 bytes.
98          '''
99          imd = block
100         for k in self._key_order:
101             imd = self._FK(k, imd)
102
103         return imd
104
105     def _decode_block(self, block: bytes):
106         """
107         Takes block 16 bytes.
108         Return decrypted block 16 bytes.
109         """
110         return self._V(self._encode_block(self._V(block)))
111
112     def _FK(self, key: bytes, block: bytes):
113         """
114         Round function.
115         Takes `block` 16 bytes and round `key` 8 bytes.
116         """
117         assert len(key) == 8 and len(block) == 16
118
119         # split block 16 bytes into two blocks 8 bytes
120         x1, x2 = block[:8], block[8:]
121
122         # (X(2),X(1) xor F(X(2),SK(n)))
123         imd = self._F(x2 + key)
124         r = bytearray()
125         for i in range(8):
126             r.append(imd[i] ^ x1[i])
127
128         return x2 + r
129
130     def _F(self, block: bytes):
131         """
132         Takes 16 bytes, return first 8 bytes of _S(_C(3, block))
133         """
```

```
        assert len(block) == 16
        res = self._S(self._C(3, block))

        return res[:8]

    @staticmethod
    def _V(arr: bytes):
        """
        Permute arr
        """
        assert len(arr) == 16

        return arr[8:] + arr[:8]

    @staticmethod
    def _generate_S():
        """
        Generate s-block.
        """
        el = 1
        s_arr = [1]
        for _ in range(255):
            el <<= 1
            if el > 255:
                el = (0xFF & el) ^ 101
            s_arr.append(el)
        s_arr[255] = 0

        return s_arr

    def _f(self, x: int):
        """
        Takes 1 byte, return 1 byte. Byte takes as int.
        Return element by index `x` in s-block.
        """
        assert 0 <= x <= 255

        return self._s[x]

    def _A(self, x: int, y: int):
        """
        Takes and return 1 byte.
        Byte takes as int.
        """
        assert 0 <= x <= 255 and 0 <= y <= 255

        return self._f(x ^ self._f(y))
```

```
181
182     def _PE(self, x: int, y: int):
183         """
184         Takes `x`, `y`: 1 byte, return tuple with 2 bytes.
185         Concat results of A(x, y) and A(y, x).
186         """
187         assert 0 <= x <= 255 and 0 <= y <= 255
188
189         return self._A(x, y), self._A(y, x)
190
191     def _P(self, arr_x: bytes):
192         """
193         Takes and return 16 bytes.
194         X=X0X1...X14X15
195         (PE(X0,X8)PE(X1,X9)...PE(X6,X14)PE(X7,X15)) - concat results PE(Xi,Xi+8) i=0...7, Xi - 1 byte.
196         """
197         assert len(arr_x) == 16
198
199         res = bytearray()
200         for i in range(8):
201             res.extend(self._PE(arr_x[i], arr_x[i + 8]))
202
203         return res
204
205     def _T(self, arr_x: bytes):
206         """
207         Use _P(arr_x) function 4 time.
208         """
209         assert len(arr_x) == 16
210
211         res = arr_x
212         for _ in range(4):
213             res = self._P(res)
214
215         return res
216
217     @staticmethod
218     def _S(arr_x: bytes):
219         """
220         Permute bytes `arr_x`: first write bytes with even sequence number, then other.
221         """
222         assert len(arr_x) == 16
223
224         permut = [0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15]
225         res = bytearray()
226         for index in permut:
227             res.append(arr_x[index])
```

```
228
229         return res
230
231     def _C(self, k: int, arr_x: bytes):
232         """
233         Recursive function:
234         C(1,X) = T(X)
235         C(k,X) = T(X ^ S(C(k-1,X)))
236         Takes and return 16 bytes.
237         """
238         assert k >= 1 and len(arr_x) == 16
239
240         if k == 1:
241             return self._T(arr_x)
242
243         # intermediate array
244         imd = self._S(self._C(k - 1, arr_x))
245
246         res = self._xor_bytes(arr_x, imd)
247
248         return self._T(res)
249
250     @staticmethod
251     def _xor_bytes(b1: bytes, b2: bytes):
252         """
253         Return b1 ^ b2
254         """
255         assert len(b1) == 16 and len(b2) == 16
256
257         res = bytearray()
258         for i in range(16):
259             res.append(b1[i] ^ b2[i])
260
261         return res
262
263
264 class LUCifer:
265     def __init__(self, key_length, key_magenta, test_type: mode, accuracy=0.9995):
266         self.s_n = None
267         self.leg_dq = None
268         self.leg_dp = None
269         self.q = None
270         self.p = None
271         self.__e = None
272         self.__d = None
273         self.__n = None
274         self.key_length = key_length
```

```
275         self.test_type = test_type
276         self.__key = key_magenta
277 
278         if self.test_type == mode.MillerRabin:
279             self.test_prime = MillerRabin(accuracy)
280         elif self.test_type == mode.Fermat:
281             self.test_prime = Fermat(accuracy)
282         else:
283             self.test_prime = SoloveyShtrasen(accuracy)
284 
285     def create_keys(self):
286         self.p = self.test_prime.generate_prime(self.key_length)
287         self.q = self.test_prime.generate_prime_range(self.p + 1, 2 * self.p)
288         self.__n = self.p * self.q
289         self.__e = self.generate_e()
290         self.__d = self.calculate_d()
291         self.leg_dp = legendre_symbol(self.__d, self.p)
292         self.leg_dq = legendre_symbol(self.__d, self.q)
293         self.s_n = lcm(self.p - self.leg_dp, self.q - self.leg_dq)
294         print(f"Open key: [{self.__e}, {self.__n}]")
295 
296     def generate_e(self):
297         temp = (self.p - 1) * (self.q - 1) * (self.p + 1) * (self.q + 1)
298         while True:
299             e = random.getrandbits(self.key_length // 4)
300             if math.gcd(e, temp) == 1:
301                 return e
302 
303     def calculate_d(self):
304         return pow(self.__key, 2) - 4
305 
306     def calculate_dd(self):
307         return mod_inverse(self.__e, self.s_n)
308 
309     @staticmethod
310     def encrypt(magenta_key, e, n):
311         e += 1
312         Vn = np.zeros(e)
313         Vn[0] = 2
314         Vn[1] = magenta_key
315         i = 2
316         while i < e:
317             Vn[i] = (magenta_key * Vn[i - 1] - Vn[i - 2]) % n
318             i += 1
319         return int(Vn[e - 1])
320 
321     @staticmethod
```

```
322     def decrypt(magenta_key, e, n):
323         e += 1
324         Vn = np.zeros(e)
325         Vn[0] = 2
326         Vn[1] = magenta_key
327         i = 2
328         while i < e:
329             Vn[i] = (magenta_key * Vn[i - 1] - Vn[i - 2]) % n
330             i += 1
331         return int(Vn[e - 1])
332 
333 
334 class EncryptMode(Magenta, metaclass=ABCMeta):
335 
336     def __init__(self, key, c0):
337         #
338         key = self._check_key(key)
339         super().__init__(key)
340         self._check_c0(c0)
341 
342     @abstractmethod
343     def encode(self, text: bytes):
344         '''
345         Encode text.
346         '''
347 
348     @abstractmethod
349     def decode(self, text: bytes):
350         '''
351         Decode text.
352         '''
353 
354     def _check_length(self, text: bytes):
355         '''
356         Check length, if len(text) % 16 != 0, complite length.
357         '''
358         if len(text) % 16 != 0:
359             return text + bytes(16 - len(text) % 16)
360         return text
361 
362     def _check_c0(self, c0):
363         '''
364         Check c0 length. If length < 16 complite if > 16 cut.
365         '''
366         if len(c0) < 16:
367             self._c0 = c0 + bytes(16 - len(c0))
368 
```

```
369         elif len(c0) > 16:
370             self._c0 = c0[:16]
371 
372         else:
373             self._c0 = c0
374 
375     def _check_key(self, key: bytes):
376         '''
377         Check key length. If length not equals 16, 24, 32 complite.
378         If great then 32 cut.
379         '''
380         key_length = len(key)
381 
382         if key_length == 16 or key_length == 24 or key_length == 32:
383             return key
384 
385         if key_length < 16:
386             return key + bytes(16 - key_length)
387 
388         elif key_length < 24:
389             return key + bytes(24 - key_length)
390 
391         elif key_length < 32:
392             return key + bytes(32 - key_length)
393 
394         else:
395             return key[:32]
396 
397 
398 class ECB(EncryptMode):
399 
400     def __init__(self, key):
401         super().__init__(key, bytes())
402 
403     def encode(self, text: bytes):
404         '''
405         Encode text.
406         '''
407         text = self._check_length(text)
408         res = bytearray()
409         for i in range(0, len(text) - 15, 16):
410             res.extend(self._encode_block(text[i:i + 16]))
411 
412         return res
413 
414     def decode(self, text: bytes):
415         '''
```

```
416         Decode text
417         '''
418         text = self._check_length(text)
419 
420         res = bytearray()
421         for i in range(0, len(text) - 15, 16):
422             res.extend(self._decode_block(text[i:i + 16]))
423 
424         while res[-1] == 0:
425             del res[-1]
426 
427         return res
428 
429 
430 class CFB(EncryptMode):
431 
432     def encode(self, text: bytes):
433         '''
434         Encode text.
435         '''
436         text = self._check_length(text)
437 
438         res = bytearray()
439         prev = self._c0
440         for i in range(0, len(text) - 15, 16):
441             prev = self._xor_bytes(self._encode_block(prev), text[i:i + 16])
442             res.extend(prev)
443 
444         return res
445 
446     def decode(self, text: bytes):
447         '''
448         Decode text
449         '''
450         res = bytearray()
451         prev = self._c0
452         for i in range(0, len(text) - 15, 16):
453             res.extend(self._xor_bytes(self._encode_block(prev), text[i:i + 16]))
454             prev = text[i:i + 16]
455 
456         while res[-1] == 0:
457             del res[-1]
458 
459         return res
460 
461 
462 class CBC(EncryptMode):
```

```
463
464     def encode(self, text: bytes):
465         '''
466         Encode text.
467         '''
468         text = self._check_length(text)
469
470         prev = self._c0
471         res = bytearray()
472         for i in range(0, len(text) - 15, 16):
473             prev = self._encode_block(self._xor_bytes(text[i:i + 16], prev))
474             res.extend(prev)
475
476         return res
477
478     def decode(self, text: bytes):
479         '''
480         Decode text
481         '''
482         text = self._check_length(text)
483
484         prev = self._c0
485         res = bytearray()
486         for i in range(0, len(text) - 15, 16):
487             res.extend(self._xor_bytes(self._decode_block(text[i:i + 16]), prev))
488             prev = text[i:i + 16]
489
490         while res[-1] == 0:
491             del res[-1]
492
493         return res
494
495
496
497 class OFB(EncryptMode):
498
499     def encode(self, text: bytes):
500         '''
501         Encode text.
502         '''
503         text = self._check_length(text)
504         res = bytearray()
505         prev = self._c0
506         for i in range(0, len(text) - 15, 16):
507             prev = self._encode_block(prev)
508             res.extend(self._xor_bytes(prev, text[i:i + 16]))
509
```

```
510         return res
511 
512     def decode(self, text: bytes):
513         '''
514         Decode text
515         '''
516         text = self._check_length(text)
517 
518         res = self.encode(text)
519 
520         while res[-1] == 0:
521             del res[-1]
522 
523         return res
524 
525 
```

---

app.py:

```
1  import uvicorn
2  from methods import *
3  
4  from fastapi import FastAPI, Response, status, Depends, Query, File, UploadFile
5  from typing import Optional, List
6  from starlette.responses import FileResponse
7  
8  import db_models
9  from db_connect import engine, SessionLocal
10 from sqlalchemy.orm import Session
11 from db_models import Files
12 from encryptmode import LUCKey
13 from PrimeNumber import PrimeType as mode
14 
15 # DB
16 db_models.Base.metadata.create_all(engine)
17 
18 keys = LUCKey(256, mode.MillerRabin, 0.9999999)
19 
20 
21 def get_db():
22     with SessionLocal() as db:
23         return db
24 
25 
26 # END DB
```

```python


app = FastAPI()


@app.get("/filenames/{user_id}", status_code=status.HTTP_200_OK)  # , response_model=List[str]
async def get_file_names(user_id: str, db: Session = Depends(get_db)):
    res = db.query(Files).filter(Files.user_id == user_id).all()
    print(res)
    return res


@app.get("/key_asymmetric/{username}", status_code=status.HTTP_200_OK)
def get_asymmetric_key(
        response: Response,
        username: str,
        db: Session = Depends(get_db)
):
    return keys.get_open_key()


@app.post("/key/{username}", status_code=status.HTTP_200_OK)
async def get_keys(
        response: Response,
        username: str,
        key: Optional[str] = None,
        c_0: Optional[str] = None,
        db: Session = Depends(get_db)
):
    user = get_user_from_db(db, username)
    if user:
        user_update_keys(db, key=key, user_name=username, c_0=c_0)
    else:
        response.status_code = status.HTTP_404_NOT_FOUND


@app.post("/login/{username}", status_code=status.HTTP_200_OK)
async def login_user(
        response: Response,
        username: str,
        password: Optional[str] = None,
        db: Session = Depends(get_db)):
    user = get_user_from_db(db, username)
    if user:
        if user.password == password:
            response.status_code = status.HTTP_200_OK
            return get_user_id_from_db(db, username)
```

```
        else:
            response.status_code = status.HTTP_401_UNAUTHORIZED
            return {'msg': 'Wrong password'}
    else:
        response.status_code = status.HTTP_201_CREATED
        return add_user_to_db(
            db,
            login=username,
            password=password,
        )


@app.post("/file/upload/{user_id}", status_code=status.HTTP_200_OK)
async def upload_file(
        response: Response,
        user_id: Optional[str] = None,
        cypher_type: Optional[str] = None,
        file: UploadFile = File(...),
        db: Session = Depends(get_db)
):
    key = get_key(db, user_id)[0]
    c_0 = get_c_0(db, user_id)[0]
    # Format new filename
    full_name = format_filename(file)

    # Save file
    await save_file_to_uploads(file, full_name, user_id)

    with open(f'{UPLOADED_FILES_PATH}{user_id}/{full_name}', 'rb') as f:
        open_text = f.read()
    print(key)

    enc = get_encoder(cypher_type, key, c_0)
    decoded = enc.decode(open_text)
    os.remove(f'{UPLOADED_FILES_PATH}{user_id}/{full_name}')
    with open(f'{UPLOADED_FILES_PATH}{user_id}/{full_name}', 'wb') as f:
        f.write(decoded)

    # Get file size
    file_size = get_file_size(full_name, user_id)

    # Get info from DB
    file_info_from_db = get_file_from_db(db, full_name, user_id)

    # Add to DB
    if not file_info_from_db:
        response.status_code = status.HTTP_201_CREATED
```

```
121         return add_file_to_db(
122             db,
123             full_name=full_name,
124             file_size=file_size,
125             user_id=user_id,
126             # file=file
127         )
128 
129 
130 @app.get("/file/download/{user_id}", status_code=status.HTTP_200_OK)
131 async def download_file(
132         response: Response,
133         file_name: str,
134         user_id: str,
135         db: Session = Depends(get_db)
136 ):
137     file_info_from_db = get_file_from_db(db, file_name, user_id)
138 
139     if file_info_from_db:
140         if
        ↪  os.path.exists(f"{UPLOADED_FILES_PATH}{file_info_from_db.user_id}/{file_info_from_db.name}"):
141             file_resp =
            ↪  FileResponse(f"{UPLOADED_FILES_PATH}{file_info_from_db.user_id}/{file_info_from_db.name}")
142             response.status_code = status.HTTP_200_OK
143             return file_resp
144         else:
145             response.status_code = status.HTTP_404_NOT_FOUND
146             return {'msg': 'File not found'}
147     else:
148         response.status_code = status.HTTP_404_NOT_FOUND
149         return {'msg': 'File not found'}
150 
151 
152 @app.delete("/api/delete", tags=["Delete"])
153 async def delete_file(
154         response: Response,
155         file_id: int,
156         db: Session = Depends(get_db)
157 ):
158     file_info_from_db = get_file_from_db(db, file_id)
159 
160     if file_info_from_db:
161         # Delete file from DB
162         delete_file_from_db(db, file_info_from_db)
163 
164         # Delete file from uploads
165         delete_file_from_uploads(file_info_from_db.name)
```

```
166
167         response.status_code = status.HTTP_200_OK
168         return {'msg': f'File {file_info_from_db.name} successfully deleted'}
169     else:
170         response.status_code = status.HTTP_404_NOT_FOUND
171         return {'msg': f'File does not exist'}
172
173
174 @app.get("/api/get", tags=["Get files"], status_code=status.HTTP_200_OK)
175 async def root(
176         # *,
177         response: Response,
178         id: Optional[List[int]] = Query(None),
179         name: Optional[List[str]] = Query(None),
180         tag: Optional[List[str]] = Query(None),
181         limit: Optional[int] = None,
182         offset: Optional[int] = None,
183         db: Session = Depends(get_db)
184 ):
185     # All records by default
186     query = db.query(db_models.Files).all()
187     files_in_db = get_files_from_db_limit_offset(db, query, limit, offset)
188
189     if id and not name and not tag:
190         query = db.query(db_models.Files).filter(db_models.Files.file_id.in_(id)).all()
191         files_in_db = get_files_from_db_limit_offset(db, query, limit, offset)
192
193     elif id and name and not tag:
194         query = db.query(db_models.Files).filter(db_models.Files.file_id.in_(id)) \
195             .filter(db_models.Files.name.in_(name)) \
196             .all()
197         files_in_db = get_files_from_db_limit_offset(db, query, limit, offset)
198
199     elif id and name and tag:
200         query = db.query(db_models.Files).filter(db_models.Files.file_id.in_(id)) \
201             .filter(db_models.Files.name.in_(name)) \
202             .filter(db_models.Files.tag.in_(tag)) \
203             .all()
204         files_in_db = get_files_from_db_limit_offset(db, query, limit, offset)
205
206     elif id and not name and tag:
207         query = db.query(db_models.Files).filter(db_models.Files.file_id.in_(id)) \
208             .filter(db_models.Files.tag.in_(tag)) \
209             .all()
210         files_in_db = get_files_from_db_limit_offset(db, query, limit, offset)
211
212     elif not id and name and tag:
```

```
        query = db.query(db_models.Files).filter(db_models.Files.name.in_(name)) \
            .filter(db_models.Files.tag.in_(tag)) \
            .all()
        files_in_db = get_files_from_db_limit_offset(db, query, limit, offset)

    elif not id and not name and tag:
        query = db.query(db_models.Files).filter(db_models.Files.tag.in_(tag)).all()
        files_in_db = get_files_from_db_limit_offset(db, query, limit, offset)

    elif not id and name and not tag:
        query = db.query(db_models.Files).filter(db_models.Files.name.in_(name)).all()
        files_in_db = get_files_from_db_limit_offset(db, query, limit, offset)

    if len(files_in_db) == 0:
        response.status_code = status.HTTP_404_NOT_FOUND
        return {'message': 'No results =('}

    response.status_code = status.HTTP_200_OK
    return files_in_db


if __name__ == "__main__":
    uvicorn.run("app:app", host="127.0.0.1", reload=True)

```

db_connect.py:

```
 from sqlalchemy import Column, Integer, String, ForeignKey
from db_connect import Base
from sqlalchemy.orm import relationship


class Files(Base):
    __tablename__ = 'file'
    id = Column(Integer, primary_key=True, index=True, autoincrement=True)
    name = Column(String)
    size = Column(Integer)
    user_id = Column(Integer, ForeignKey("user.id"))
    user = relationship("User")


class User(Base):
    __tablename__ = 'user'
    id = Column(Integer, primary_key=True, index=True, autoincrement=True)
    login = Column(String)
```

```
    password = Column(String)
    key = Column(String)
    c_0 = Column(String)
```

db_models.py:

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker


SQLALCHEMY_DATABASE_URL = 'sqlite:///./encsoft.db'

engine = create_engine(SQLALCHEMY_DATABASE_URL, connect_args={'check_same_thread': False})

SessionLocal = sessionmaker(bind=engine, autocommit=False, autoflush=False)

Base = declarative_base()
```

methods.py:

```
import os
from datetime import datetime

import db_models
from settings import *
from encryptmode import ECB, OFB, CBC, CFB, EncryptMode


# Get file info from DB
def get_file_from_db(db, file_name, user_id):
    return db.query(db_models.Files) \
        .filter(db_models.Files.name == file_name
                and db_models.Files.user_id == user_id).first()


def get_user_from_db(db, user_name):
    return db.query(db_models.User).filter(db_models.User.login == user_name).first()


def get_user_id_from_db(db, user_name):
    ans = db.query(db_models.User).filter(db_models.User.login == user_name).one()
    return ans.id
```

```
23
24
25 def get_encoder(cypher_type, key, c_0: str = None):
26
27     if cypher_type == 'ECB':
28         return ECB(key.encode())
29     elif cypher_type == 'RDH':
30         return CBC(key.encode(), c_0.encode())
31     elif cypher_type == 'RD':
32         return CBC(key.encode(), c_0.encode())
33     elif cypher_type == 'CBC':
34         return CBC(key.encode(), c_0.encode())
35     elif cypher_type == 'CTR':
36         return CBC(key.encode(), c_0.encode())
37     elif cypher_type == 'OFB':
38         return OFB(key.encode(), c_0.encode())
39     elif cypher_type == 'CFB':
40         return CFB(key.encode(), c_0.encode())
41
42
43 def user_update_keys(db, key, c_0, user_name):
44     db.query(db_models.User). \
45         filter(db_models.User.login == user_name). \
46         update({db_models.User.key: key, db_models.User.c_0: c_0})
47     db.commit()
48
49
50 def get_key(db, user_id):
51     return db.query(db_models.User.key).filter(db_models.User.id == user_id).first()
52
53
54 def get_c_0(db, user_id):
55     return db.query(db_models.User.c_0).filter(db_models.User.id == user_id).first()
56
57
58 def add_user_to_db(db, **kwargs):
59     new_user = db_models.User(
60         login=kwargs['login'],
61         password=kwargs['password'],
62         key=kwargs['key'],
63         c_0=kwargs['c_0']
64     )
65     db.add(new_user)
66     db.commit()
67     db.refresh(new_user)
68     return db.query(db_models.User.id).filter(db_models.User.login == kwargs['login']).one().id
69
```

```python

# Offset\limit
def get_files_from_db_limit_offset(db, query, limit: int = None, offset: int = None):
    if limit and not offset:
        query = query[:limit]
    elif limit and offset:
        limit += offset
        query = query[offset:limit]
    elif not limit and offset:
        query = query[offset:]
    return query


# Delete file from uploads folder
def delete_file_from_uploads(file_name):
    try:
        os.remove(UPLOADED_FILES_PATH + file_name)
    except Exception as e:
        print(e)


# Save file to uploads folder
async def save_file_to_uploads(file, filename, user_id):
    if not os.path.exists(f'{UPLOADED_FILES_PATH}{user_id}'):
        os.makedirs(f'{UPLOADED_FILES_PATH}{user_id}')
    with open(f'{UPLOADED_FILES_PATH}{user_id}/{filename}', "wb") as uploaded_file:
        file_content = await file.read()
        uploaded_file.write(file_content)
        uploaded_file.close()


# Format filename
def format_filename(file):
    # Split filename and extention
    filename, ext = os.path.splitext(file.filename)

    return filename + ext


# Get file size
def get_file_size(filename, user_id):
    file_path = f'{UPLOADED_FILES_PATH}{user_id}/{filename}'
    return os.path.getsize(file_path)


# Add File to DB
def add_file_to_db(db, **kwargs):
```

```
117     new_file = db_models.Files(
118         name=kwargs['full_name'],
119         size=kwargs['file_size'],
120         user_id=kwargs['user_id']
121     )
122     db.add(new_file)
123     db.commit()
124     db.refresh(new_file)
125     return new_file
126 
127 
```