МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

**Московский авиационный институт**

**(национальный исследовательский университет)**

Институт № 8

Компьютерные науки и прикладная математика

**Кафедра 806 «Вычислительная математика и программирование»**

# Курсовой проект

по дисциплине «Основы криптографии»

Тема:

«XTR + RC6»

**Выполнил:** студент группы М8О-311Б-20

Комиссаров Антон Сергеевич

(Фамилия, имя, отчество)

**Преподаватель:**

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 и должны генерироваться вероятностными тестами простоты (Соловея-Штрассена, Миллера-Рабина, Ферма));
   (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 распределенным между сторонами сеансовым ключом (с использованием IV при режиме шифрования, отличном от ECB) на одной стороне с последующей передачей ею зашифрованного файла (вместе с вектором инициализации) другой стороне;
   (e) Асинхронно и многопоточно (по возможности) дешифровать переданный зашифрованный файл распределенным между сторонами сеансовым ключом (с использованием IV при режиме шифрования, отличном от ECB), с избавлением от набивки (padding);
   (f) Отображать прогресс операций шифрования и дешифрования при помощи элемента управления ProgressBar;
   (g) 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 на экране появляется система авторизации.


Рис. 1: Вход в систему

Новый пользователь может зарегистрироваться в системе с помощью уникального приглашения от другого пользователя.

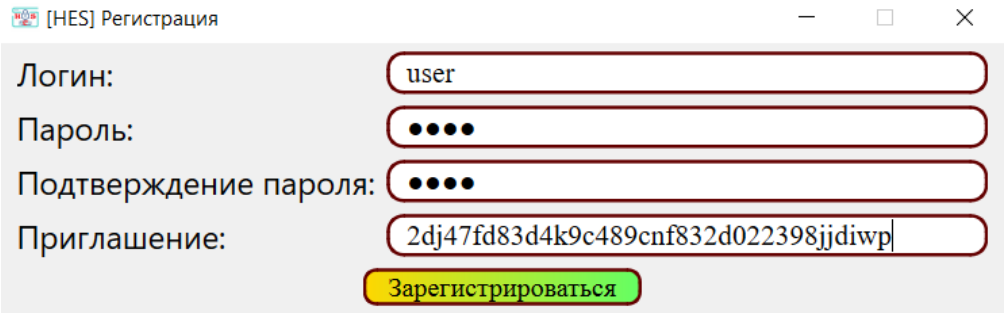


Рис. 2: Регистрация в системе

После успешной авторизации пользователь получает доступ к главному окну системы со всем функционалом. На боковой панели располагается приветствие пользователя с указанием его логина, а также две кнопки: выход из аккаунта и генерация приглашения.

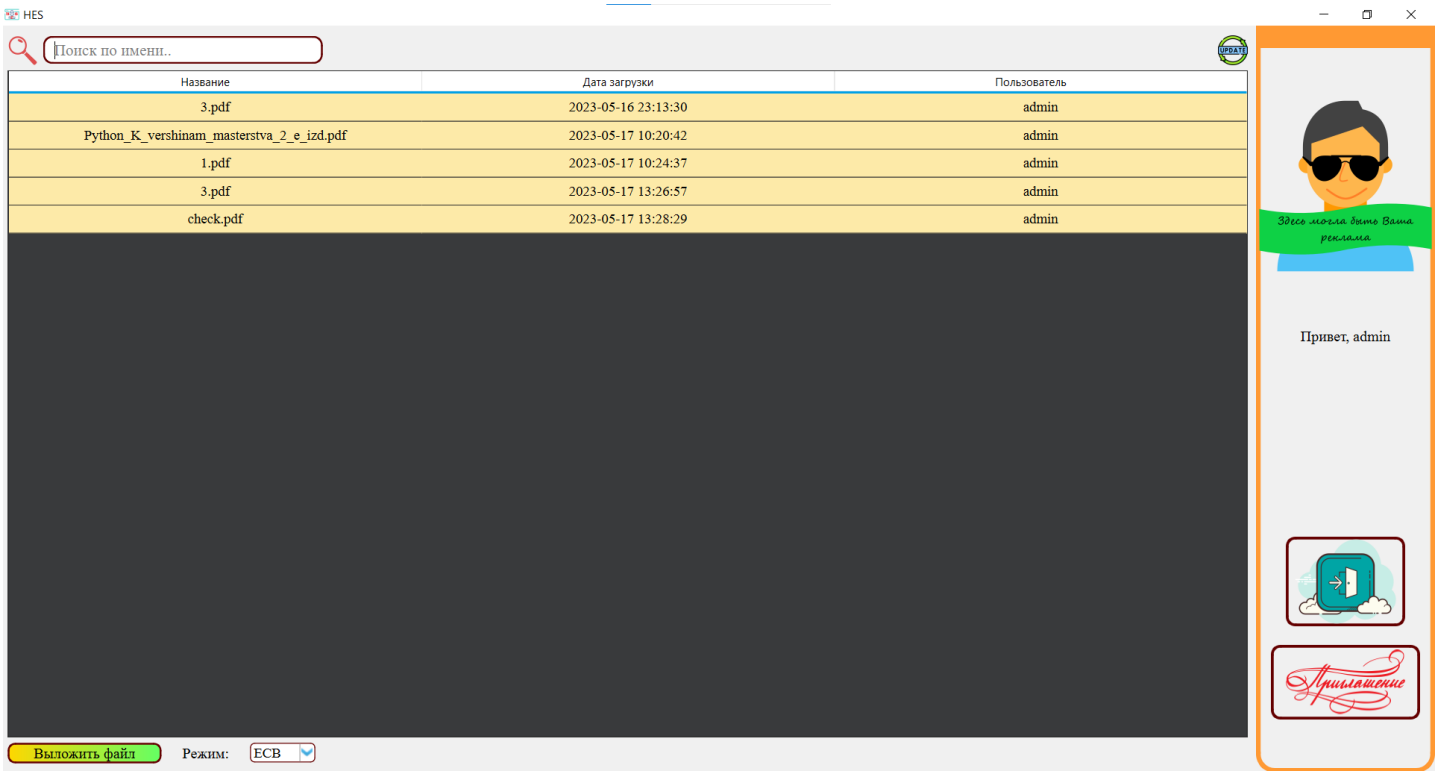


Рис. 3: Успешная авторизация

С помощью поисковой строки пользователь может отфильтровать файлы по названию, используя регулярные выражения. Также есть возможность выполнить сортировку файлов по столбцам.

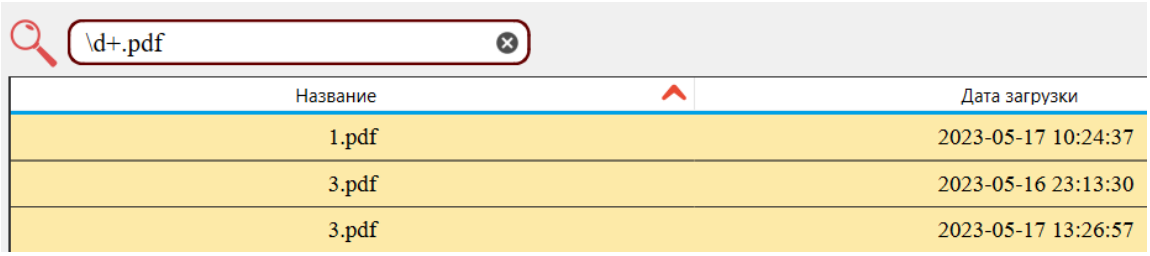


Рис. 4: Поиск и сортировка

Для загрузки файла на сервер пользователю необходимо выбрать из выпадающего списка режим шифрования и нажать кнопку “Выложить файл”.

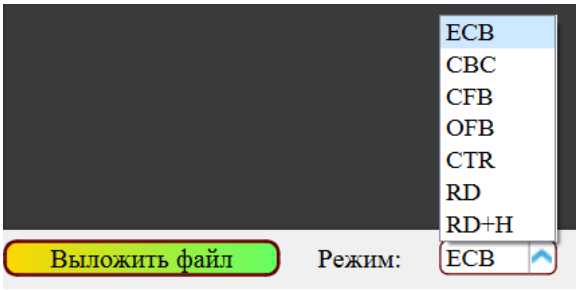


Рис. 5: Выбор режима шифрования

После этого необходимо выбрать файл для отправки.

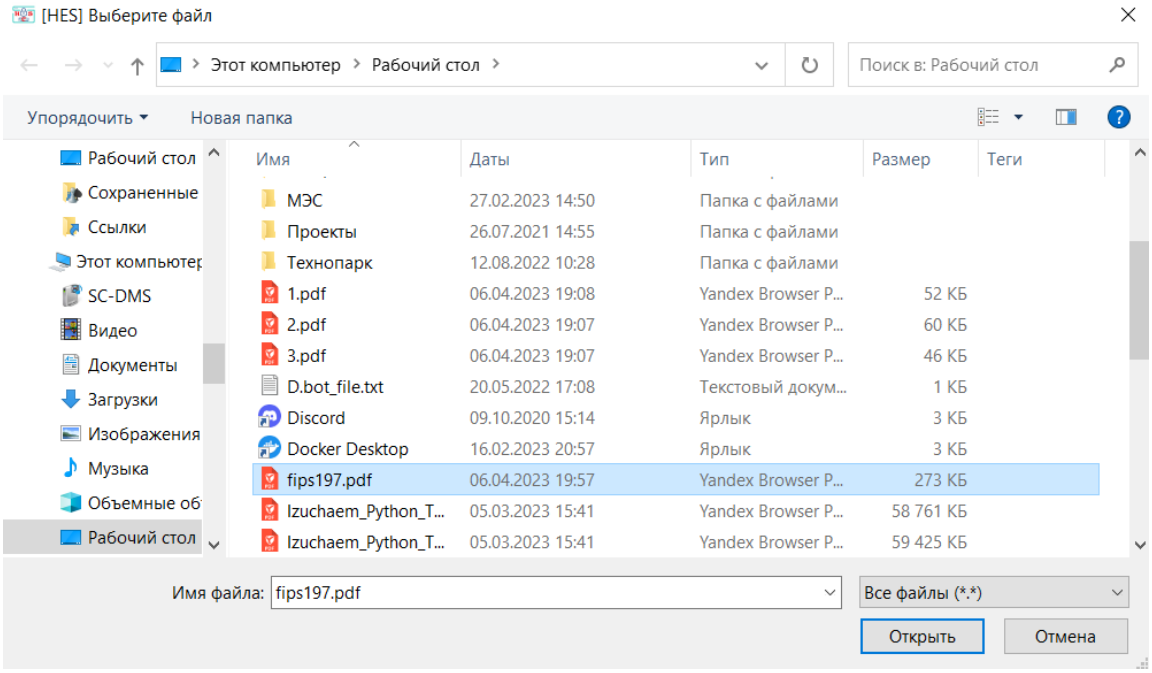


Рис. 6: Выбор файла

Затем появляется окно с указанием статуса обработки файла и прогрессом (для шифрования / дешифрования). Для оптимальной нагрузки на систему одновременно может шифроваться и дешифроваться не более 3-х файлов. Пользователь может отменить отправку файла на сервер, закрыв соответствующее окно. После завершения обработки происходит автоматическое обновление таблицы с файлами, которые находятся на сервере. Пользователь может самостоятельно вызвать операцию обновления, нажав на кнопку с надписью “Update”.

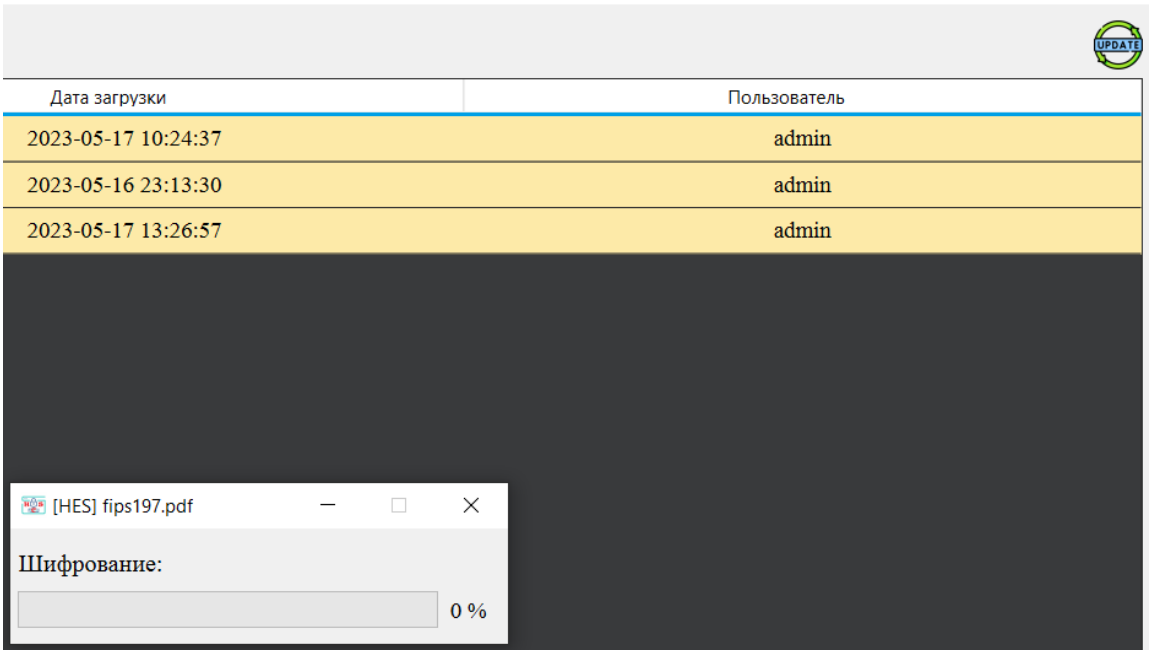


Рис. 7: Загрузка файла

При нажатии правой кнопки мыши по файлу в таблице появляется контекстное меню с возможностью скачивания и удаления файла с сервера. Возможностью удаления файла обладает только тот пользователь, который его загрузил.

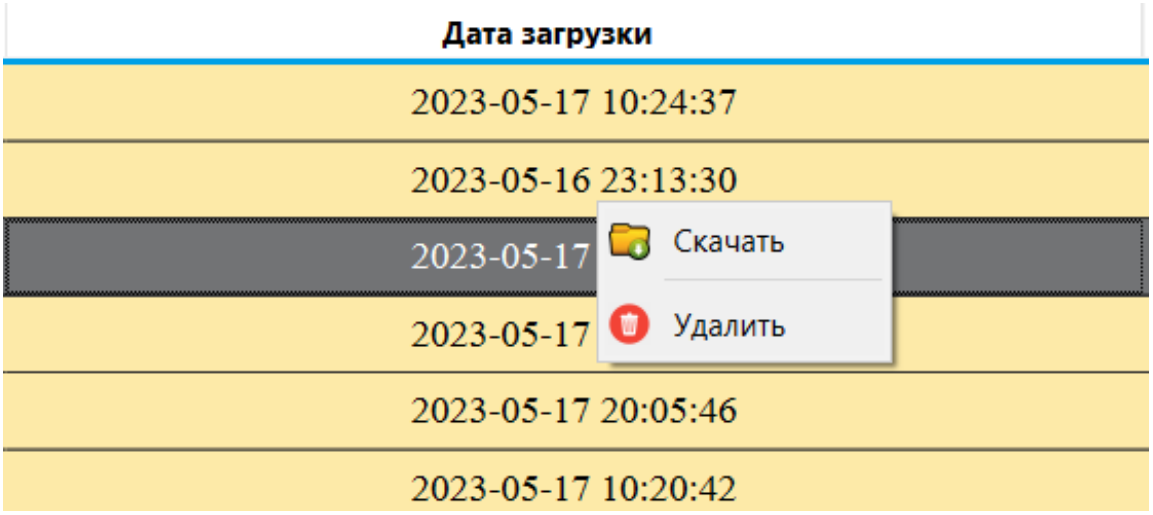


Рис. 8: Меню действий с файлом

При скачивании файла необходимо указать для файла конечную директорию и его имя. После этого начинается скачивание файла с сервера и его последующее дешифрование.

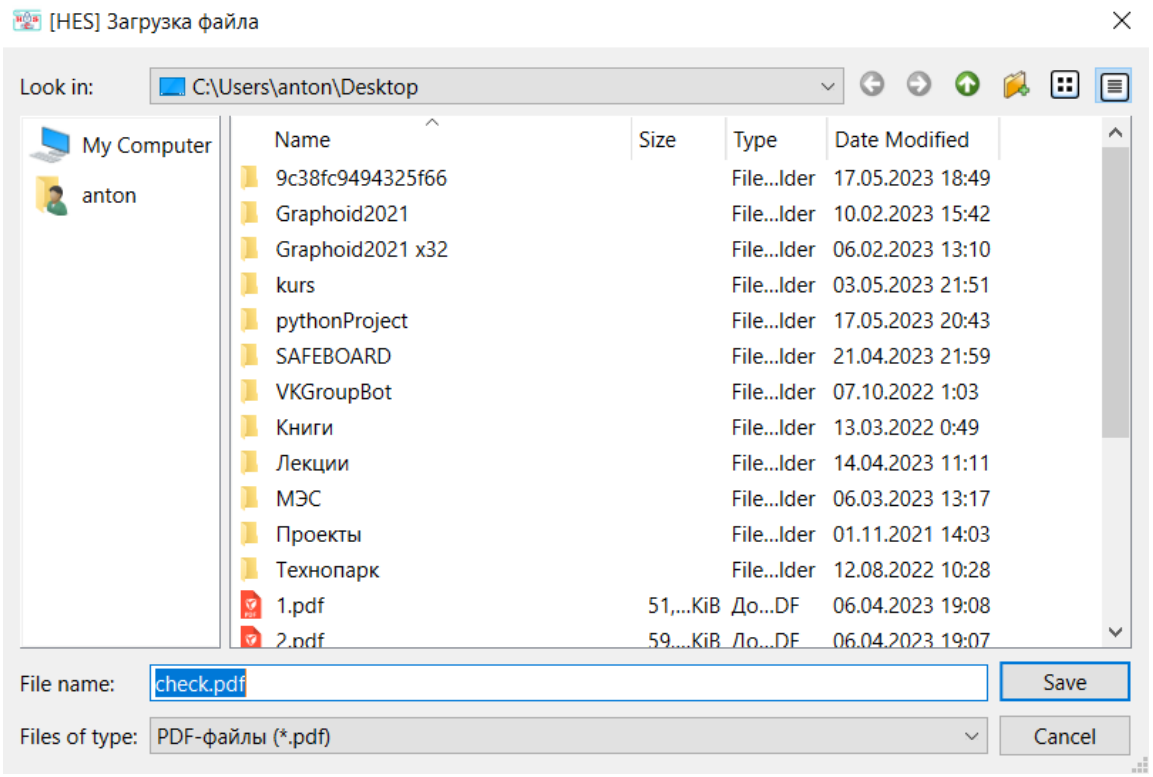


Рис. 9: Выбор пути сохранения

# 3 Структура приложения

Приложение обладает трёхуровневой архитектурой: все запросы от клиента поступают на сервер, где они обрабатываются, при необходимости получают информацию из базы данных или добавляют в неё, формируя ответ.

## 3.1 Сервер

Реализован на языке программирования Python с использованием фреймворка Flask. В качестве базы данных используется компактная встраиваемая СУБД SQLite.

1. server.py – Содержит в себе обработку запросов клиента и всю логику сервера в целом.

2. db.py – Реализует класс *DBAggregator*, предназначенный для непосредственного взаимодействия с базой данных.

3. init.sql – SQL-скрипт для создания необходимых таблиц при первичном (“чистом”) запуске сервера.

## 3.2 Клиент

Реализован на языке программирования Python с использованием библиотеки PyQt6. Для связи с сервером используется библиотека requests.

### 3.2.1 Интерфейс (папка *compiled_ui*)

1. main_page.py – Шаблон главного окна.

2. login.py – Шаблон окна входа.

3. registration.py – Шаблон окна регистрации.

4. file_manager.py – Шаблон окна обработки файла.

### 3.2.2 Взаимодействие с пользователем

1. main.py – Логика работы главного окна.
2. login.py – Реализация входа пользователя в систему.
3. registration.py – Реализация регистрации пользователя.
4. file_manager.py – Реализация обработки файла с его последующей отправкой.
5. updater.py – Реализация обновления таблицы списка файлов на сервере.
6. file_name_item.py – Реализация элемента таблицы файлов для хранения названия файла и его номера.

## 3.3 Алгоритмы шифрования

1. simplicity_tests.py – Реализация вероятностных тестов простоты.
2. cryption_algorithms.py – Реализация алгоритмов шифрования RC6 и XTR.
3. file_encryption.py – Реализация режимов шифрования и шифрования файла в целом (с набивкой PKCS7).

## 3.4 Прочее

1. variables.py – Установочные данные для всей системы в целом.

# 4 Основные алгоритмы

## 4.1 Сервер

Общее взаимодействие клиента и сервера представлено на схеме ниже:

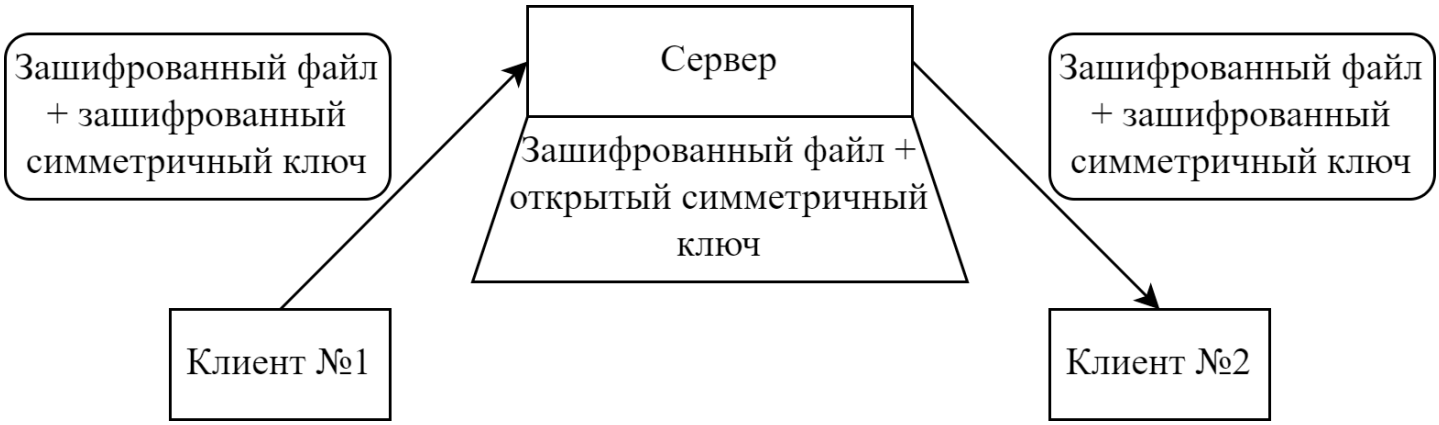


Рис. 10: Общая схема

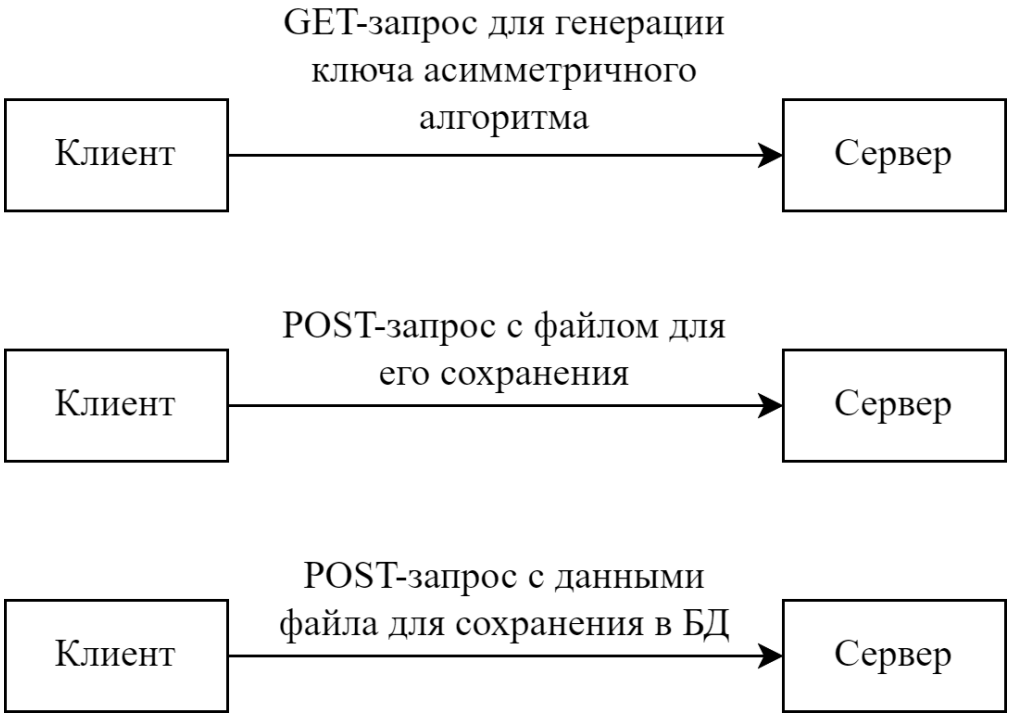


Рис. 11: Добавление файла на сервер

Добавление файла:

- GET-запрос для генерации ключа асимметричного алгоритма: генерация открытого ключа XTR и половины закрытого ключа по схеме Эль-Гамаля (выбор случайного $k$ из промежутка $[2; q-3]$ и вычисление его следа).

```
@api.route('/key/asymmetric/', methods=['GET'])
def get_asymmetric_key():
  xtr = ca.XTR(variables.TEST, variables.TEST_PRECISION, variables.XTR_KEY_BIT_SIZE)
  open_key = xtr.generate_key()
  el_gamal_key = xtr.get_el_gamal_key()
  key_index = base.reg_el_gamal_key(open_key[0], el_gamal_key[0])
  return json.dumps(dict(p=open_key[0], q=open_key[1], tr=open_key[2], tr_k=el_gamal_key[1],
  ↪  key_index=key_index))
```

- POST-запрос для сохранения файла: получение файла в двоичном виде из запроса и его сохранение под переданным серверным названием.

```
@api.route('/file', methods=['POST'])
def upload_file():
  try:
    FileStorage(request.files.get('file')).save(os.path.join(variables.DATA_DIR,
    ↪  request.form['name']))
    return 'OK'
  except KeyError:
    abort(400, 'Некорректный запрос')
```

- POST-запрос для сохранения информации о файле в базу данных: запись в БД системного названия, оригинального названия, владельца файла, режима шифрования, а также вектора инициализации при наличии.
  Происходит вычисление второй половины закрытого ключа XTR (на основе переданного в запросе значения следа и ранее выбранного $k$), которая используется при дешифровке ключа RC6 для файла. Полученный симметричный ключ записывается в открытом виде в базу данных. Для повышения безопасности хранения ключа используется индексация по значению хэш-функции $hashlib.sha256$ от системного названия файла.

```
@api.route('/file/info', methods=['POST'])
def upload_file_info():
  args = json.loads(request.json)
  try:
    base.upload_file(args['new_name'], args['old_name'], args['owner'], args['mode'], "" if
    ↪  args['vector'] is None else args['vector'])
    xtr_value = base.get_xtr_values(args['key_index'])
    sym_key = bytes(pair[0] ^ pair[1] for pair in zip(convert_str(args['sym_key']),
    ca.XTR.get_symmetric_key_back(xtr_value[0], xtr_value[1],
    args['tr_g_b'])))
    base.add_sym_key(sha256(args['new_name'].encode('utf-8')).hexdigest(),
    ↪  convert_bytes(sym_key))
    return 'OK'
```

```
    except KeyError:
      abort(400, 'Некорректный запрос')
```

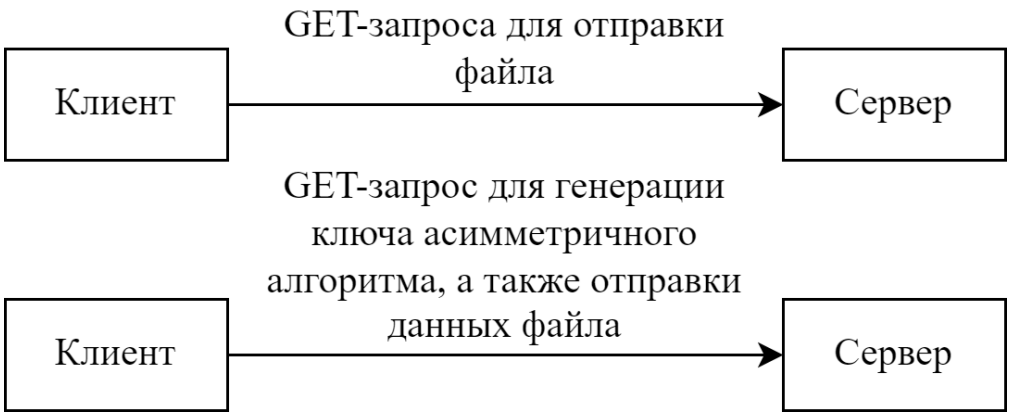


Рис. 12: Скачивание файла с сервера

Скачивание файла:

- GET-запрос для отправки файла: получение системного названия файла из БД по переданному индеку и его отправка пользователю.

```
@api.route('/file/<file_id>', methods=['GET'])
def download_file(file_id: int):
  try:
    return send_file(os.path.join(variables.DATA_DIR, base.get_file_name(file_id)),
    ↪  as_attachment=True)
  except FileNotFoundError:
    abort(400, 'Некорректный запрос')
```

- GET-запрос для генерации ключа асимметричного алгоритма, а также отправки данных файла: генерация половины закрытого ключа по схеме Эль-Гамаля (выбор случайного $b$ из промежутка $[2; q-3]$ и вычисление его следа) на основе переданных значений $(p, q, Tr_k)$ и отправка зашифрованного ключа RC6, следа $Tr_b$, режима шифрования и вектора инициализации при наличии.

```
@api.route('/file/info/', methods=['GET'])
def download_file_info():
  args = json.loads(request.json)
  try:
    sym_key = base.get_sym_key(args['file_id'])
    trace, key = ca.XTR.get_symmetric_key(args['p'], args['q'], args['tr'], args['tr_k'])
    key = bytes(pair[0] ^ pair[1] for pair in zip(key, convert_str(sym_key)))
    file_data = base.get_file_data(args['file_id'])
```

```
9         return json.dumps(dict(key=convert_bytes(key), tr=trace, mode=file_data[0],
          ↪  init_vector=file_data[1]))
10      except KeyError:
11        abort(400, 'Некорректный запрос')
```

## 4.2 Клиент

Основная сложность клиента заключалась в реализации взаимодействия между графическим интерфейсом и потоками обработки файлов. Для решения этой проблемы используются потоки Qt *QThread*: для каждого процесса шифрования / дешифрования запускается отдельный поток-обработчик, который привязан к окну отображения статуса и прогресса *[Приложение 6.2.4]*. Для связи обработчиков с основным потоком приложения используются слоты и сигналы Qt.

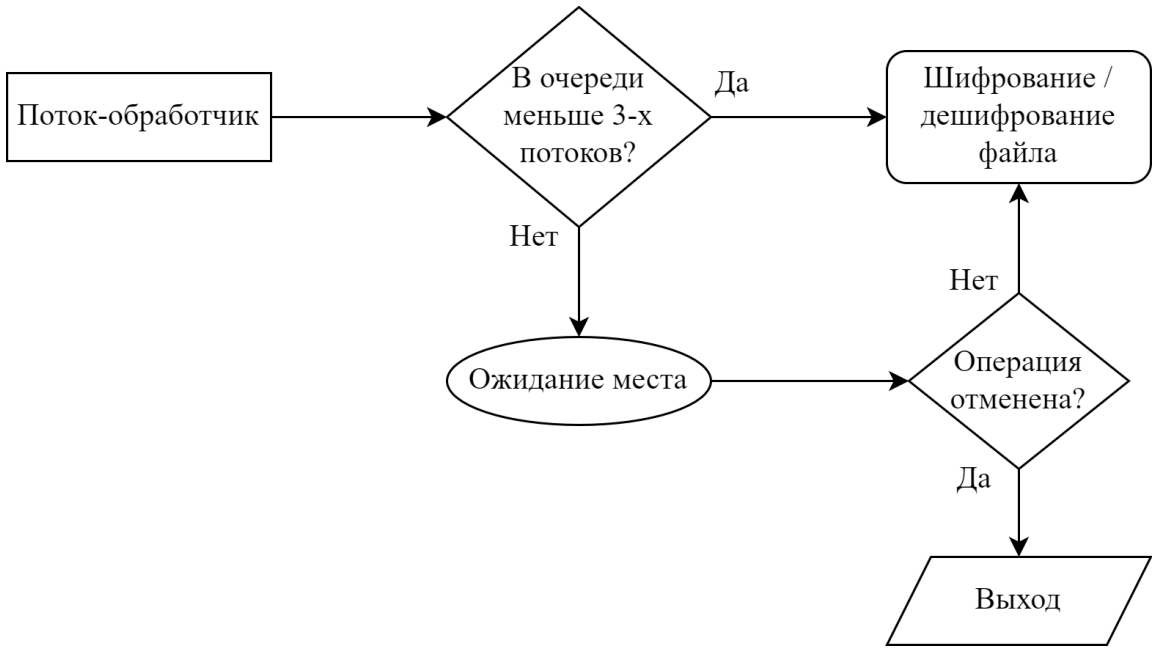


Рис. 13: Ограничение потоков шифрования / дешифрования

Для получения максимальной эффективности при обработке файла одновременное количество шифруемых / дешифруемых файлов ограничено с помощью потокобезопасной очереди: если количество обработчиков больше трёх, то новый поток ждёт до тех пор, пока не появится свободное место.

Пример кода для шифрования (дешифрование происходит аналогично):

```
1 def encrypt(self, in_file: str, out_file: str):
2   self.thread_queue.put(True)
```

```
  if self.__exit:
    self.thread_queue.get()
    return
  self.__progress = 0
  with (open(in_file, 'rb') as f_in, open(out_file, 'wb') as f_out):
    match self.__mode:
    case AggregatorMode.ECB:
      mode_aggregator = ModeECB(self.__block_size, self.__algorithm)
    case AggregatorMode.CBC:
      mode_aggregator = ModeCBC(self.__block_size, self.__algorithm, self.__init_vector)
    case AggregatorMode.CFB:
      mode_aggregator = ModeCFB(self.__block_size, self.__algorithm, self.__init_vector)
    case AggregatorMode.OFB:
      mode_aggregator = ModeOFB(self.__block_size, self.__algorithm, self.__init_vector)
    case AggregatorMode.CTR:
      mode_aggregator = ModeCTR(self.__block_size, self.__algorithm)
    case AggregatorMode.RD:
      mode_aggregator = ModeRD(self.__block_size, self.__algorithm, self.__init_vector)
    case AggregatorMode.RDH:
      mode_aggregator = ModeRDH(self.__block_size, self.__algorithm, self.__init_vector)
    while (block := f_in.read(self.__block_size * variables.BLOCK_ENCRYPT_SIZE)) and not self.__exit:
      count = self.__block_size - len(block) % self.__block_size
      f_out.write(bytes(mode_aggregator.encrypt(block + count.to_bytes(1, 'big') * count)))
      self.__progress += len(block)
      self.progress.emit(self.__progress)
  self.thread_queue.get()
```

## 4.3 Алгоритмы шифрования

Симметричный алгоритм RC6 не вызвал в своей реализации трудностей. Для организации процесса шифрования и дешифрования файла использовались наработки из лабораторной работы. Многопоточность была реализована с помощью класса ThreadPool из модуля multiprocessing.

При работе с асимметричный алгоритмом XTR возникли небольшие трудности. Для этого алгоритма необходимо было реализовать класс вычисления следа по заданным начальным параметрам и класс элемента поля $GF(p^2)$: элемент состоит из двух значений поля $GF(p)$, а операции между ними реализованы особым образом.

```
def get_swapped(self) -> 'GFP2Element':
  return type(self)(self.__p, params=(self.__b, self.__a))

def get_square(self) -> 'GFP2Element':
```

```
    return type(self)(self.__p, params=(self.__b * (self.__b - 2 * self.__a), self.__a * (self.__a - 2 *
    ↪  self.__b)))

def __sub__(self, other) -> 'GFP2Element':
    return type(self)(self.__p, params=(self.__a - other.__a, self.__b - other.__b))

def __add__(self, other) -> 'GFP2Element':
    return type(self)(self.__p, params=(self.__a + other.__a, self.__b + other.__b))

@staticmethod
def special_operation(x: 'GFP2Element', y: 'GFP2Element', z: 'GFP2Element') -> 'GFP2Element':
    return type(x)(x.__p, params=(z.__a * (y.__a - x.__b - y.__b) + z.__b * (x.__b - x.__a + y.__b),
    ↪  z.__a * (x.__a - x.__b + y.__a) + z.__b * (y.__b - x.__a - y.__a)))
```

# 5 Список использованных источников


1. ru.wikipedia.org. RC6.
   URL: https://ru.wikipedia.org/wiki/RC6 (дата обращения 06.05.2023).

2. ru.wikipedia.org. XTR (алгоритм).
   URL: https://ru.wikipedia.org/wiki/XTR_(алгоритм) (дата обращения 07.05.2023).

3. Arjen K. Lenstra, Eric R. Verheul. The XTR public key system.
   URL: https://www.iacr.org/archive/crypto2000/18800001/18800001.pdf (дата обращения 07.05.2023).

4. Arjen K. Lenstra, Eric R. Verheul. An overview of the XTR public key system.
   URL: https://web.archive.org/web/20060415185309/http://www.win.tue.nl:80/klenstra/xtrsurvey.pdf (дата обращения 07.05.2023).

# 6 Приложение

## 6.1 Сервер

### 6.1.1 server.py

```
import os
from hashlib import sha256

from flask import Flask, json, request, abort, Response, send_file
from werkzeug.datastructures import FileStorage

import cryption_algorithms as ca
import variables
from client.file_manager import convert_bytes, convert_str
from db import DBAggregator

api = Flask(__name__)
api.config.from_object(__name__)
api.config.update(dict(
    DATABASE=os.path.join(api.root_path, 'base.db'),
))
api.config.from_envvar('FLASKR_SETTINGS', silent=True)
base = DBAggregator(api.config['DATABASE'])


@api.route('/user/login/', methods=['GET'])
def login():
    args = json.loads(request.json)
    try:
        return json.dumps(dict(index=base.check_user(args['login'], args['password'])))
    except KeyError:
        abort(400, 'Некорректный запрос')


@api.route('/user/invite/', methods=['GET'])
def invite_user():
    try:
        return json.dumps(base.invite_user(json.loads(request.json).get('user_id')))
    except KeyError:
        abort(400, 'Некорректный запрос')


@api.route('/user/reg', methods=['POST'])
def add_user():
    args = json.loads(request.json)
```

```
41      try:
42          return Response(status=base.add_user(args['login'], args['password'], args['invitation']))
43      except KeyError:
44          abort(400, 'Некорректный запрос')
45
46
47  @api.route('/key/asymmetric/', methods=['GET'])
48  def get_asymmetric_key():
49      xtr = ca.XTR(variables.TEST, variables.TEST_PRECISION, variables.XTR_KEY_BIT_SIZE)
50      open_key = xtr.generate_key()
51      el_gamal_key = xtr.get_el_gamal_key()
52      key_index = base.reg_el_gamal_key(open_key[0], el_gamal_key[0])
53      return json.dumps(dict(p=open_key[0], q=open_key[1], tr=open_key[2], tr_k=el_gamal_key[1],
        ↪  key_index=key_index))
54
55
56  @api.route('/file', methods=['POST'])
57  def upload_file():
58      try:
59          FileStorage(request.files.get('file')).save(os.path.join(variables.DATA_DIR,
            ↪  request.form['name']))
60          return 'OK'
61      except KeyError:
62          abort(400, 'Некорректный запрос')
63
64
65  @api.route('/file/info', methods=['POST'])
66  def upload_file_info():
67      args = json.loads(request.json)
68      try:
69          base.upload_file(args['new_name'], args['old_name'], args['owner'], args['mode'],
70                           "" if args['vector'] is None else args['vector'])
71          xtr_value = base.get_xtr_values(args['key_index'])
72          sym_key = bytes(pair[0] ^ pair[1] for pair in zip(convert_str(args['sym_key']),
73                                                            ca.XTR.get_symmetric_key_back(xtr_value[0],
                                                              ↪  xtr_value[1],
74
                                                                                            ↪  args['tr_g_b'])))
75          base.add_sym_key(sha256(args['new_name'].encode('utf-8')).hexdigest(), convert_bytes(sym_key))
76          return 'OK'
77      except KeyError:
78          abort(400, 'Некорректный запрос')
79
80
81  @api.route('/files/', methods=['GET'])
82  def get_server_files():
83      return json.dumps(base.get_server_files())
```

```
84
85
86  @api.route('/file/<file_id>', methods=['GET'])
87  def download_file(file_id: int):
88      try:
89          return send_file(os.path.join(variables.DATA_DIR, base.get_file_name(file_id)),
            ↪  as_attachment=True)
90      except FileNotFoundError:
91          abort(400, 'Некорректный запрос')
92
93
94  @api.route('/file/info/', methods=['GET'])
95  def download_file_info():
96      args = json.loads(request.json)
97      try:
98          sym_key = base.get_sym_key(args['file_id'])
99          trace, key = ca.XTR.get_symmetric_key(args['p'], args['q'], args['tr'], args['tr_k'])
100         key = bytes(pair[0] ^ pair[1] for pair in zip(key, convert_str(sym_key)))
101         file_data = base.get_file_data(args['file_id'])
102         return json.dumps(dict(key=convert_bytes(key), tr=trace, mode=file_data[0],
            ↪  init_vector=file_data[1]))
103     except KeyError:
104         abort(400, 'Некорректный запрос')
105
106
107 @api.route('/file/delete/<file_id>', methods=['DELETE'])
108 def delete_file(file_id: int):
109     try:
110         file_name = base.delete_file(file_id)
111         os.remove(os.path.join(variables.DATA_DIR, file_name))
112         return 'OK'
113     except FileNotFoundError:
114         abort(400, 'Некорректный запрос')
115
116
117 if __name__ == '__main__':
118     api.run(debug=True, threaded=True)
```

### 6.1.2 db.py

```
1  import os
2  import sqlite3
3  from hashlib import sha256
4  from typing import Optional, Dict, Tuple, List
5  from uuid import uuid4
6
```

```python

def conn(function):
    def f(self, *args, **kwargs):
        connection = sqlite3.connect(self.path)
        connection.row_factory = sqlite3.Row
        self.cursor = connection.cursor()
        result = function(self, *args, **kwargs)
        connection.close()
        return result

    return f


class DBAggregator:
    def __init__(self, path: bytes):
        self.path = path
        if not os.path.exists(path):
            with open(path, 'w'):
                pass
            connection = sqlite3.connect(self.path)
            connection.row_factory = sqlite3.Row
            with open('init.sql', mode='r') as f:
                connection.cursor().executescript(f.read())
            connection.commit()
            connection.close()

        self.cursor: Optional[sqlite3.Cursor] = None

    @conn
    def check_user(self, login: str, password: str) -> Optional[int]:
        self.cursor.execute(f'select user_id from SystemUser where login = "{login}" and password =
        ↪  "{password}"')
        answer = self.cursor.fetchone()
        if answer is not None:
            return answer[0]
        return None

    @conn
    def add_user(self, login: str, password: str, invitation: str) -> int:
        self.cursor.execute(f'select user_id from SystemUser where login = "{login}"')
        if self.cursor.fetchone() is not None:
            return 201
        self.cursor.execute(f'select user_id from SystemUser where invitation = "{invitation}"')
        if self.cursor.fetchone() is None:
            return 202
        self.cursor.execute(f'insert into SystemUser(login, password) values("{login}",
        ↪  "{password}")')
```

```
52          self.cursor.connection.commit()
53          return 200
54
55      @conn
56      def invite_user(self, user_id: int) -> Dict:
57          self.cursor.execute(f'select invitation from SystemUser where user_id = {user_id}')
58          if (invitation := self.cursor.fetchone()[0]) is not None:
59              return dict(invitation=invitation, generated=False)
60          invitation = uuid4()
61          self.cursor.execute(f'update SystemUser set invitation = "{invitation}" where user_id =
            ↪ {user_id}')
62          self.cursor.connection.commit()
63          return dict(invitation=invitation, generated=True)
64
65      @conn
66      def reg_el_gamal_key(self, p: int, k: int) -> int:
67          self.cursor.execute(f'insert into AsymmetricKey(p_value, k_value) values("{p}", "{k}")')
68          self.cursor.connection.commit()
69          return self.cursor.execute(f'select max(key_id) from AsymmetricKey').fetchone()[0]
70
71      @conn
72      def upload_file(self, system_name: str, file_name: str, owner: int, crypt_mode: int, init_vector:
        ↪ str) -> None:
73          self.cursor.execute(f'insert into UploadFile(system_file_name, file_name, owner, crypt_mode,
            ↪ init_vector)'
74                              f'values("{system_name}", "{file_name}", {owner}, {crypt_mode},
                              ↪ "{init_vector}")')
75          self.cursor.connection.commit()
76
77      @conn
78      def add_sym_key(self, name: str, key: str) -> None:
79          self.cursor.execute(f'insert into SymmetricKey(file_hash, file_key) values("{name}",
            ↪ "{key}")')
80          self.cursor.connection.commit()
81
82      @conn
83      def get_xtr_values(self, index: int) -> Tuple[int, int]:
84          result = self.cursor.execute(f'select p_value, k_value from AsymmetricKey where key_id =
            ↪ {index}').fetchone()
85          self.cursor.execute(f'delete from AsymmetricKey where key_id = {index}')
86          self.cursor.connection.commit()
87          return int(result[0]), int(result[1])
88
89      @conn
90      def get_server_files(self) -> List[Dict]:
91          return [dict(row) for row in self.cursor.execute(f'select file_id, file_name, (select login
            ↪ from SystemUser '
```

```
                                                    f'where user_id = owner) as user, upload_time from '
                                                    f'UploadFile').fetchall()]

    @conn
    def delete_file(self, file_id: int) -> str:
        file_name = self.cursor.execute(f'select system_file_name from UploadFile where '
                                        f'file_id = {file_id}').fetchone()[0]
        self.cursor.execute(f'delete from UploadFile where file_id = {file_id}')
        self.cursor.connection.commit()
        file_hash = sha256(file_name.encode('utf-8')).hexdigest()
        self.cursor.execute(f'delete from SymmetricKey where file_hash = "{file_hash}"')
        self.cursor.connection.commit()
        return file_name

    @conn
    def get_file_name(self, file_id: int) -> str:
        return self.cursor.execute(f'select system_file_name from UploadFile where file_id = {file_id}').fetchone()[0]

    @conn
    def get_sym_key(self, file_id: int) -> str:
        file_hash = sha256(self.cursor.execute(f'select system_file_name from UploadFile where file_id = {file_id}')
                           .fetchone()[0].encode('utf-8')).hexdigest()
        return self.cursor.execute(f'select file_key from SymmetricKey where file_hash = "{file_hash}"').fetchone()[0]

    @conn
    def get_file_data(self, file_id: int) -> Tuple[int, str]:
        return tuple(self.cursor.execute(f'select crypt_mode, init_vector from UploadFile where '
                                         f'file_id = {file_id}').fetchone())
```

### 6.1.3 init.sql

```
drop table if exists SystemUser;
create table SystemUser (
    user_id integer primary key autoincrement,
    login varchar(40) not null,
    password varchar(64) not null,
    invitation varchar(36) DEFAULT(NULL)
);

drop table if exists SymmetricKey;
create table SymmetricKey (
    file_hash varchar(64) primary key,
```

```
    file_key varchar(256) not null
);

drop table if exists AsymmetricKey;
create table AsymmetricKey (
    key_id integer primary key autoincrement,
    p_value varchar(128) not null,
    k_value varchar(128) not null
);

drop table if exists UploadFile;
create table UploadFile (
    file_id integer primary key autoincrement,
    system_file_name varchar(40) not null,
    file_name varchar(256) not null,
    owner integer not null,
    upload_time timestamp DEFAULT(datetime('now', '+3 hours')),
    crypt_mode integer not null,
    init_vector varchar(256) DEFAULT(NULL),
    FOREIGN KEY(owner) REFERENCES SystemUser(user_id)
);
```

## 6.2 Клиент

### 6.2.1 main.py

```
import json
import os
import re
import sys
from typing import Optional

import requests
from PyQt6 import QtGui, QtCore
from PyQt6.QtCore import QThread, QPoint
from PyQt6.QtGui import QAction, QIcon, QPixmap
from PyQt6.QtWidgets import *

import updater
import variables
from client.compiled_ui.main_page import Ui_MainWindow
from client.file_name_item import FileNameItem
from file_manager import FileManager
from login import LogWindow
from variables import ICON_PATH, SERVER_ADDRESS

```

```
21
22 class HESApp(QMainWindow):
23     def __init__(self):
24         super(HESApp, self).__init__()
25         self.ui = Ui_MainWindow()
26         self.ui.setupUi(self)
27         icon = QtGui.QIcon()
28         icon.addPixmap(QtGui.QPixmap(ICON_PATH), QtGui.QIcon.Mode.Normal, QtGui.QIcon.State.Off)
29         self.setWindowIcon(icon)
30
31         self.user_id: Optional[int] = None
32         self.user_login: Optional[str] = None
33         self.hide()
34         login = LogWindow(self.windowIcon())
35         login.user_index.connect(self.set_user)
36         login.exec()
37         if self.user_id is None:
38             quit(0)
39         for f in os.listdir(variables.TEMP_DIR):
40             os.remove(os.path.join(variables.TEMP_DIR, f))
41         self.ui.dockWidget.setFixedWidth(self.ui.label.width())
42         self.ui.dockWidget.setFeatures(QDockWidget.DockWidgetFeature.DockWidgetMovable)
43         self.ui.dockWidget.setAllowedAreas(QtCore.Qt.DockWidgetArea.LeftDockWidgetArea |
44                                            QtCore.Qt.DockWidgetArea.RightDockWidgetArea)
45         self.ui.dockWidgetContents.setLayout(self.ui.verticalLayout)
46         self.settings = QtCore.QSettings("T-Corp.", "HES")
47         self.addDockWidget(self.settings.value("DockSide",
        ↪  QtCore.Qt.DockWidgetArea.RightDockWidgetArea),
48                            self.ui.dockWidget)
49         self.ui.dockWidget.dockLocationChanged.connect(self.changed_panel_side)
50         self.ui.label.setText(f"Пользователь: {self.user_id}")
51         self.ui.verticalLayout.setAlignment(QtCore.Qt.AlignmentFlag.AlignCenter)
52         self.ui.label.setPixmap(QtGui.QPixmap("images/person.png"))
53         self.ui.pushButton.setIcon(QtGui.QIcon(QtGui.QPixmap("images/quit.png")))
54         self.ui.pushButton.clicked.connect(self.logout)
55         self.ui.pushButton_2.setIcon(QtGui.QIcon(QtGui.QPixmap("images/invite.png")))
56         self.ui.pushButton_2.clicked.connect(self.invite)
57
58         self.ui.centralwidget.setLayout(self.ui.verticalLayout_2)
59         self.ui.label_3.setPixmap(QtGui.QPixmap("images/search.png"))
60         self.update_btn = updater.UpdateBtn()
61         self.update_btn.setAlignment(QtCore.Qt.AlignmentFlag.AlignRight)
62         self.ui.horizontalLayout.addWidget(self.update_btn)
63         self.update_btn.clicked.connect(self.update_files)
64         self.ui.horizontalLayout_2.setAlignment(QtCore.Qt.AlignmentFlag.AlignLeft)
65         self.ui.comboBox.setStyleSheet(self.ui.comboBox.styleSheet() +
66                                        "QComboBox::down-arrow { image: url(images/arrow.png); }"
```

```
67                                      "QComboBox::down-arrow:on { image: url(images/arrow_up.png);
                                        ↪  }")
68          self.ui.comboBox.setCurrentIndex(self.settings.value("CryptMode", 0))
69          self.ui.comboBox.currentIndexChanged.connect(lambda: self.settings.setValue("CryptMode",
70
                                                                                         ↪  self.ui.comboBox.curr
71          self.ui.pushButton_3.clicked.connect(self.upload_file)
72
73          self.thread: Optional[QThread] = None
74          self.updater: Optional[updater.Updater] = None
75          self.file_managers = {}
76          self.file_managers_count = 0
77
78          self.ui.lineEdit.textEdited.connect(self.search_by_name)
79          self.ui.lineEdit.inputRejected.connect(self.search_by_name)
80          self.ui.tableWidget.setHorizontalHeaderLabels(["Название", "Дата загрузки", "Пользователь"])
81          self.ui.tableWidget.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMode.Stretch)
82          self.ui.tableWidget.setContextMenuPolicy(QtCore.Qt.ContextMenuPolicy.CustomContextMenu)
83          self.ui.tableWidget.customContextMenuRequested.connect(self.table_menu_show)
84          self.table_menu = QMenu(self)
85          download = QAction("Скачать", self.table_menu)
86          download.setIcon(QIcon(QPixmap("images/download.ico")))
87          download.triggered.connect(self.download_file)
88          delete = QAction("Удалить", self.table_menu)
89          delete.setIcon(QIcon(QPixmap("images/delete.ico")))
90          delete.triggered.connect(self.delete_file)
91          self.table_menu.addAction(download)
92          self.table_menu.addSeparator()
93          self.table_menu.addAction(delete)
94          self.aim_row: int = -1
95
96          self.showMaximized()
97          self.update_btn.clicked.emit()
98
99      def download_file(self):
100         aim_file = self.ui.tableWidget.item(self.aim_row, 0)
101         aim_text = aim_file.text()
102         file_type = aim_text[aim_text.rfind(".") + 1:]
103         file_path = QFileDialog.getSaveFileName(self, "[HES] Загрузка файла",
104                                                 os.path.join(self.settings.value("DownloadFolder",
                                                    ↪  ""),
105                                                              aim_text), f"{file_type.upper()}-файлы
                                                              ↪  (*.{file_type});;"
106                                                                      f"Все файлы (*.*)",
107                                                 options=QFileDialog.Option.DontUseNativeDialog)[0]
108         if not file_path:
109             return
```

```
110         if file_path.rfind(".") == -1:
111             file_path += f".{file_type}"
112         self.settings.setValue("DownloadFolder", file_path[:file_path.rfind("/")])
113         filer = FileManager(self.windowIcon(), self.user_id, file_path,
            ↪  self.ui.comboBox.currentIndex(),
114                             self.file_managers_count, aim_file.file_id)
115         self.file_managers[self.file_managers_count] = filer
116         filer.delete.connect(self.delete_filer)
117         filer.show()
118         self.file_managers_count += 1
119 
120     def delete_file(self):
121         answer = QMessageBox.question(self, "[HES] Удаление файла", "Вы точно хотите удалить этот
            ↪  файл?",
122                                       QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No)
123         if answer == QMessageBox.StandardButton.No:
124             return
125         try:
126             response = requests.delete(f"{SERVER_ADDRESS}/file/delete/"
127                                        f"{self.ui.tableWidget.item(self.aim_row, 0).file_id}")
128             if response.status_code != 200:
129                 QMessageBox.warning(self, "[HES] Удаление файла", "Не удалось получить корректный
                    ↪  ответ от сервера!")
130                 return
131             self.update_btn.clicked.emit()
132         except requests.ConnectionError:
133             QMessageBox.critical(self, "[HES] Удаление файла", "Не удалось подключиться к серверу!")
134 
135     def table_menu_show(self, point: QPoint):
136         self.aim_row = self.ui.tableWidget.rowAt(point.y())
137         if self.aim_row == -1:
138             return
139         if self.ui.tableWidget.item(self.aim_row, 2).text() != self.user_login:
140             self.table_menu.actions()[2].setEnabled(False)
141         else:
142             self.table_menu.actions()[2].setEnabled(True)
143         self.table_menu.exec(self.ui.tableWidget.mapToGlobal(point))
144 
145     def search_by_name(self, text: str):
146         if text == "":
147             for i in range(self.ui.tableWidget.rowCount()):
148                 self.ui.tableWidget.showRow(i)
149             return
150         text = text.lower()
151         try:
152             for i in range(self.ui.tableWidget.rowCount()):
153                 if re.search(text, self.ui.tableWidget.item(i, 0).text().lower()) is None:
```

```
154                     self.ui.tableWidget.hideRow(i)
155                 else:
156                     self.ui.tableWidget.showRow(i)
157         except re.error:
158             pass
159 
160     def add_file_to_table(self, file_id: int, name: str, upload_time: str, user: str):
161         self.ui.tableWidget.setRowCount(self.ui.tableWidget.rowCount() + 1)
162         self.ui.tableWidget.setItem(self.ui.tableWidget.rowCount() - 1, 0, FileNameItem(name,
        ↪  file_id))
163         self.ui.tableWidget.setItem(self.ui.tableWidget.rowCount() - 1, 1,
        ↪  self.prepare_item(upload_time))
164         self.ui.tableWidget.setItem(self.ui.tableWidget.rowCount() - 1, 2, self.prepare_item(user))
165 
166     @staticmethod
167     def prepare_item(text: str):
168         item = QTableWidgetItem(text)
169         item.setTextAlignment(QtCore.Qt.AlignmentFlag.AlignCenter)
170         item.setFont(QtGui.QFont("Times New Roman", 11))
171         return item
172 
173     def upload_file(self):
174         folder = self.settings.value("FileFolder", "")
175         if (result := QFileDialog.getOpenFileName(self, "[HES] Выберите файл", folder, "Все файлы
        ↪  (*.*)")[0]) == '':
176             return
177         self.settings.setValue("FileFolder", result[:result.rfind('')])
178         if os.stat(result).st_size > variables.MAX_BYTE_FILE_SIZE:
179             QMessageBox.warning(self, "[HES] Ошибка", "Файл превышает установленный максимальный
            ↪  размер!")
180             return
181         filer = FileManager(self.windowIcon(), self.user_id, result, self.ui.comboBox.currentIndex(),
182                             self.file_managers_count)
183         self.file_managers[self.file_managers_count] = filer
184         filer.delete.connect(self.delete_filer)
185         filer.show()
186         self.file_managers_count += 1
187 
188     def delete_filer(self, index: int):
189         del self.file_managers[index]
190         self.update_btn.clicked.emit()
191 
192     def update_files(self):
193         self.ui.tableWidget.setSortingEnabled(False)
194         self.ui.lineEdit.setText("")
195         self.ui.tableWidget.setRowCount(0)
196         self.thread = QThread()
```

```
197         self.updater = updater.Updater()
198         self.updater.moveToThread(self.thread)
199         self.updater.error.connect(lambda value: QMessageBox.critical(self, "[HES] Ошибка",
200                                                                       f'\tОшибка при
                                                                      ↪  обновлении:\n{value}'))
201         self.updater.add_file.connect(self.add_file_to_table)
202         self.thread.started.connect(self.updater.run)
203         self.updater.finished.connect(lambda: self.update_btn.stop_updating())
204         self.updater.finished.connect(self.thread.quit)
205         self.updater.finished.connect(lambda: self.ui.tableWidget.setSortingEnabled(True))
206         self.updater.finished.connect(self.updater.deleteLater)
207         self.thread.finished.connect(self.thread.deleteLater)
208         self.thread.start()
209 
210     def set_user(self, user: int, name: str):
211         self.user_id = user
212         self.user_login = name
213         self.ui.label_2.setText(f"Привет, {name if len(name) <= 15 else name[:15] + '...'}")
214 
215     def changed_panel_side(self):
216         self.settings.setValue("DockSide", self.dockWidgetArea(self.ui.dockWidget))
217 
218     def logout(self):
219         self.hide()
220         self.user_id = None
221         login = LogWindow(self.windowIcon())
222         login.user_index.connect(self.set_user)
223         login.exec()
224         if self.user_id is None:
225             quit(0)
226         self.showMaximized()
227 
228     def invite(self):
229         try:
230             response = requests.get(f"{SERVER_ADDRESS}/user/invite/",
231                                     json=json.dumps(dict(user_id=self.user_id)))
232             if response.status_code != 200:
233                 QMessageBox.warning(self, "[HES] Приглашение", "Не удалось получить корректный ответ
                    ↪  от сервера!")
234                 return
235             result = json.loads(response.text)
236             QApplication.clipboard().setText(result['invitation'])
237             QMessageBox.information(self, "[HES] Приглашение",
238                                     f"{'Сгенерировано новое приглашение' if result['generated'] else
                                        ↪  'Приглашение'}:\n"
239                                     f"{result['invitation']}\n\n(Помещено в буфер обмена)")
240         except requests.ConnectionError:
```

```
            QMessageBox.critical(self, "[HES] Приглашение", "Не удалось подключиться к серверу!")

    def closeEvent(self, a0: QtGui.QCloseEvent) -> None:
        for manager in self.file_managers.values():
            manager.index = None
            manager.close()
        a0.accept()


if __name__ == '__main__':
    app = QApplication([])
    application = HESApp()
    application.show()
    sys.exit(app.exec())
```

### 6.2.2 login.py

```
import hashlib
import json

import requests
from PyQt6 import QtWidgets, QtCore, QtGui
from PyQt6.QtGui import QPixmap

from client.compiled_ui.login import Ui_Login
from registration import RegWindow
from variables import SERVER_ADDRESS


class LogWindow(QtWidgets.QDialog):
    user_index = QtCore.pyqtSignal(int, str)

    def __init__(self, icon: QtGui.QIcon):
        super(LogWindow, self).__init__()
        self.ui = Ui_Login()
        self.ui.setupUi(self)
        self.setLayout(self.ui.verticalLayout)
        self.setWindowIcon(icon)
        self.setWindowFlags(QtCore.Qt.WindowType.WindowMinimizeButtonHint |
        ↪  QtCore.Qt.WindowType.WindowCloseButtonHint)
        self.ui.pushButton.clicked.connect(self.login)
        self.ui.pushButton_2.clicked.connect(self.registration_window)
        self.ui.label_3.setPixmap(QPixmap('images/login.png'))
        self.__settings = QtCore.QSettings("T-Corp.", "HES")
        if self.__settings.contains("UserLogin"):
            self.ui.lineEdit.setText(self.__settings.value("UserLogin"))
```

```
            self.ui.lineEdit_2.setFocus()

    def registration_window(self):
        registration = RegWindow(self.windowIcon())
        registration.exec()

    def login(self):
        if not len(user_login := self.ui.lineEdit.text().strip()) or not
        ↪  len(self.ui.lineEdit_2.text()):
            QtWidgets.QMessageBox.warning(self, "[HES] Ошибка входа", "Введите логин и пароль!")
            return
        try:
            response = requests.get(f"{SERVER_ADDRESS}/user/login/",
                                    json=json.dumps(dict(login=(user_login := user_login.lower()),

                                                         ↪  password=hashlib.sha256(str.encode(self.ui.lineE
                                                         .hexdigest()))))
            if response.status_code != 200:
                QtWidgets.QMessageBox.warning(self, "[HES] Ошибка входа",
                                              "Не удалось получить корректный ответ от сервера!")
                return
            if (index := json.loads(response.text).get('index')) is None:
                QtWidgets.QMessageBox.warning(self, "[HES] Ошибка входа",
                                              "Некорректный логин или пароль!")
                return
            self.user_index.emit(index, user_login)
            self.__settings.setValue("UserLogin", user_login)
            self.close()
        except requests.ConnectionError:
            QtWidgets.QMessageBox.critical(self, "[HES] Ошибка входа", "Не удалось подключиться к
            ↪  серверу!")
```

### 6.2.3 registration.py

```
import hashlib
import json

import requests
from PyQt6 import QtWidgets, QtCore, QtGui

from client.compiled_ui.registration import Ui_Registration
from variables import SERVER_ADDRESS


class RegWindow(QtWidgets.QDialog):
    def __init__(self, icon: QtGui.QIcon):
        super(RegWindow, self).__init__()
```

```
14          self.ui = Ui_Registration()
15          self.ui.setupUi(self)
16          self.setLayout(self.ui.verticalLayout)
17          self.setWindowIcon(icon)
18          self.setWindowFlags(QtCore.Qt.WindowType.WindowMinimizeButtonHint |
            ↪  QtCore.Qt.WindowType.WindowCloseButtonHint)
19          self.ui.pushButton.clicked.connect(self.registration)
20
21      def registration(self):
22          try:
23              if not len(self.ui.lineEdit.text().strip()):
24                  raise ValueError('Введите логин!')
25              if not len(self.ui.lineEdit_2.text()):
26                  raise ValueError('Введите пароль!')
27              if self.ui.lineEdit_2.text() != self.ui.lineEdit_3.text():
28                  raise ValueError('Пароли не совпадают!')
29              if len(self.ui.lineEdit_4.text()) != 36:
30                  raise ValueError('Приглашение должно содержать 36 символов!')
31              response = requests.post(f"{SERVER_ADDRESS}/user/reg",
32
                                         ↪  json=json.dumps(dict(login=self.ui.lineEdit.text().strip().lower(),
33
                                                                 ↪  password=hashlib.sha256(str.encode(self.ui.line
34                                                                 .hexdigest(),
35                                                                 invitation=self.ui.lineEdit_4.text())))
36              if response.status_code == 201:
37                  raise ValueError('Пользователь с таким логином уже существует!')
38              if response.status_code == 202:
39                  raise ValueError('Данного приглашения не существует!')
40              self.close()
41          except ValueError as error:
42              QtWidgets.QMessageBox.warning(self, "[HES] Ошибка регистрации", error.args[0])
43          except requests.ConnectionError:
44              QtWidgets.QMessageBox.critical(self, "[HES] Ошибка регистрации", "Не удалось подключиться
                ↪  к серверу!")
```

### 6.2.4 file_manager.py

```
1  import base64
2  import json
3  import os
4  import time
5  from typing import Optional, Tuple
6  from uuid import uuid4
7
8  import requests
9  from PyQt6 import QtWidgets, QtCore, QtGui
```

```
10  from PyQt6.QtCore import QObject, pyqtSignal, QThread
11
12  import cryption_algorithms as ca
13  import file_encryption as fe
14  import variables
15  from client.compiled_ui.file_manager import Ui_FileManager
16  from cryption_algorithms import RC6
17  from variables import SERVER_ADDRESS
18
19
20  def convert_bytes(bytes_to_convert: bytes) -> str:
21      return base64.b64encode(bytes_to_convert).decode()
22
23
24  def convert_str(str_to_convert: str) -> bytes:
25      return base64.b64decode(str_to_convert)
26
27
28  class Runner(QObject):
29      finished = pyqtSignal()
30      byte_size = pyqtSignal(int)
31      progress = pyqtSignal(int)
32      status = pyqtSignal(str)
33      result = pyqtSignal(str)
34
35      def __init__(self, user_id: int, file_path: str, mode: int, new_path: str, file_id: Optional[int])
        ↪  -> None:
36          super(Runner, self).__init__()
37          self.__user = user_id
38          self.__file_path = file_path
39          self.__mode = fe.AggregatorMode(mode)
40          self.__new_path = new_path
41          self.done = False
42          self.encrypter = None
43          self.__exit = False
44          self.__file_id = file_id
45
46      def close_file_worker(self):
47          self.__exit = True
48          if self.encrypter is not None:
49              self.encrypter.shutdown()
50
51      @staticmethod
52      def get_symmetric_key() -> Tuple[int, Tuple[int, int], bytes]:
53          try:
54              response = requests.get(f"{SERVER_ADDRESS}/key/asymmetric/")
55              if response.status_code != 200:
```

```
56                 raise ValueError("Не удалось получить корректный ответ от сервера!")
57             result = json.loads(response.text)
58             key = ca.XTR.get_symmetric_key(result['p'], result['q'], result['tr'], result['tr_k'])
59             return result['key_index'], key[0], key[1]
60         except requests.ConnectionError:
61             raise ValueError("Не удалось подключиться к серверу!")
62 
63     def send_to_server(self, key_index: int, tr_g_b: Tuple[int, int], sym_key: bytes, old_file_name:
       ↪ str,
64                        new_file_path: str, init_vector: Optional[bytes]) -> None:
65         try:
66             new_name = new_file_path[new_file_path.rfind('\\') + 1:]
67             response = requests.post(f"{SERVER_ADDRESS}/file", files={'file': open(new_file_path,
               ↪ 'rb')},
68                                      data=dict(name=new_name))
69             if response.status_code != 200:
70                 raise ValueError("Не удалось получить корректный ответ от сервера!")
71             response = requests.post(f"{SERVER_ADDRESS}/file/info", json=json.dumps(
72                 dict(key_index=key_index, tr_g_b=tr_g_b, sym_key=convert_bytes(bytes(sym_key)),
73                      old_name=old_file_name, new_name=new_name, mode=self.__mode.value,
                       ↪ owner=self.__user,
74                      vector=None if init_vector is None else convert_bytes(bytes(init_vector)))))
75             if response.status_code != 200:
76                 raise ValueError("Не удалось получить корректный ответ от сервера!")
77         except requests.ConnectionError:
78             raise ValueError("Не удалось подключиться к серверу!")
79 
80     def encrypt(self):
81         algorithm = RC6(variables.RC6_WORD_BIT_SIZE, variables.RC6_ROUND_COUNT,
           ↪ variables.RC6_KEY_BYTE_SIZE)
82         key = os.urandom(variables.RC6_KEY_BYTE_SIZE)
83         vector = None if self.__mode == fe.AggregatorMode.ECB or self.__mode == fe.AggregatorMode.CTR
           ↪ else \
84             os.urandom(variables.RC6_WORD_BIT_SIZE >> 1)
85         self.encrypter = fe.Encrypter(algorithm, key, self.__mode, vector,
           ↪ block_size=variables.RC6_WORD_BIT_SIZE >> 1)
86         self.encrypter.progress.connect(self.progress)
87         try:
88             data_for_send = self.get_symmetric_key()
89             if len(key) > len(data_for_send[2]):
90                 raise ValueError("Выявлено несоответствие установочных данных!\n"
91                                  "Ключ не должен быть больше двойной длины элемента следа!")
92             self.status.emit('Шифрование:')
93             self.encrypter.encrypt(self.__file_path, self.__new_path)
94             if not self.__exit:
95                 self.status.emit('Отправка файла...')
96                 encrypted_key = bytes(pair[0] ^ pair[1] for pair in zip(key, data_for_send[2]))
```

```
97                  self.send_to_server(data_for_send[0], data_for_send[1], encrypted_key,
98                                      self.__file_path[self.__file_path.rfind("/") + 1:],
                                        ↪  self.__new_path, vector)
99                  self.status.emit('Файл отправлен')
100                 os.remove(self.__new_path)
101                 time.sleep(0.5)
102                 self.result.emit('')
103             except FileNotFoundError:
104                 self.result.emit('Файл не найден!')
105             except Exception as e:
106                 self.result.emit(str(e.args[0]))
107             self.done = True
108             self.finished.emit()
109
110     def download_from_server(self, open_key, el_gamal_key) -> Tuple[Optional[bytes], bytes, Tuple[int,
        ↪  int]]:
111         try:
112             response = requests.get(f"{SERVER_ADDRESS}/file/{self.__file_id}")
113             if response.status_code != 200:
114                 raise ValueError("Не удалось получить корректный ответ от сервера!")
115             with open(self.__new_path, 'wb') as f:
116                 f.write(response.content)
117             response = requests.get(f"{SERVER_ADDRESS}/file/info/",
118                                     json=json.dumps(dict(p=open_key[0], q=open_key[1], tr=open_key[2],
119                                                          tr_k=el_gamal_key[1],
                                                            ↪  file_id=self.__file_id)))
120             if response.status_code != 200:
121                 raise ValueError("Не удалось получить корректный ответ от сервера!")
122             result = json.loads(response.text)
123             self.__mode = fe.AggregatorMode(result['mode'])
124             return (None if result['init_vector'] == '' else convert_str(result['init_vector'])), \
125                 convert_str(result['key']), result['tr']
126         except requests.ConnectionError:
127             raise ValueError("Не удалось подключиться к серверу!")
128
129     def decrypt(self):
130         self.status.emit('Скачивание файла...')
131         xtr = ca.XTR(variables.TEST, variables.TEST_PRECISION, variables.XTR_KEY_BIT_SIZE)
132         try:
133             open_key = xtr.generate_key()
134             el_gamal_key = xtr.get_el_gamal_key()
135             data_from_download = self.download_from_server(open_key, el_gamal_key)
136             key = ca.XTR.get_symmetric_key_back(open_key[0], el_gamal_key[0], data_from_download[2])
137             sym_key = bytes(pair[0] ^ pair[1] for pair in zip(key, data_from_download[1]))
138             self.status.emit('Дешифрование:')
139             self.byte_size.emit(os.stat(self.__new_path).st_size)
```

```
            algorithm = RC6(variables.RC6_WORD_BIT_SIZE, variables.RC6_ROUND_COUNT,
            ↪  variables.RC6_KEY_BYTE_SIZE)
            self.encrypter = fe.Encrypter(algorithm, sym_key, self.__mode, data_from_download[0],
                                          block_size=variables.RC6_WORD_BIT_SIZE >> 1)
            self.encrypter.progress.connect(self.progress)
            self.encrypter.decrypt(self.__new_path, self.__file_path)
            if not self.__exit:
                self.status.emit('Файл дешифрован')
                os.remove(self.__new_path)
                time.sleep(0.5)
                self.result.emit('')
        except FileNotFoundError:
            self.result.emit('Файл не найден!')
        except Exception as e:
            self.result.emit(str(e.args[0]))
        self.done = True
        self.finished.emit()


class FileManager(QtWidgets.QDialog):
    delete = QtCore.pyqtSignal(int)

    def __init__(self, icon: QtGui.QIcon, user_id: int, file_path: str, mode: int, index: int,
                 file_id: Optional[int] = None):
        super(FileManager, self).__init__()
        self.ui = Ui_FileManager()
        self.ui.setupUi(self)
        self.__name = file_path[file_path.rfind("/") + 1:]
        self.index = index
        self.setWindowTitle(f'[HES] {self.__name}')
        self.setLayout(self.ui.verticalLayout)
        self.setWindowIcon(icon)
        self.setWindowFlags(QtCore.Qt.WindowType.WindowMinimizeButtonHint |
        ↪  QtCore.Qt.WindowType.WindowCloseButtonHint)
        self.ui.progressBar.setMaximum(os.stat(file_path).st_size if file_id is None else 100)
        self.__new_path = os.path.join(variables.TEMP_DIR, str(uuid4()))

        self.thread = QThread()
        self.worker = Runner(user_id, file_path, mode, self.__new_path, file_id)
        self.worker.moveToThread(self.thread)
        self.thread.started.connect(self.worker.encrypt if file_id is None else self.worker.decrypt)
        self.worker.finished.connect(self.thread.quit)
        self.worker.finished.connect(self.worker.deleteLater)
        self.thread.finished.connect(self.thread.deleteLater)
        self.worker.status.connect(lambda value: self.ui.label.setText(value))
        self.worker.progress.connect(lambda value: self.ui.progressBar.setValue(value))
        self.worker.byte_size.connect(lambda value: self.ui.progressBar.setMaximum(value))
```

```
185         self.worker.result.connect(self.file_result)
186         self.thread.start()
187 
188     def file_result(self, message: str):
189         if message != '':
190             QtWidgets.QMessageBox.critical(self, "[HES] Ошибка файла", f'\t{self.__name}:\n{message}')
191         else:
192             self.ui.progressBar.setValue(self.ui.progressBar.maximum())
193         self.close()
194 
195     def closeEvent(self, a0: QtGui.QCloseEvent) -> None:
196         if not self.worker.done:
197             self.worker.close_file_worker()
198             self.worker.finished.emit()
199         if self.index is not None:
200             self.delete.emit(self.index)
201         a0.accept()
```

### 6.2.5 updater.py

```
1  import json
2  
3  import requests
4  from PyQt6.QtCore import pyqtSignal, Qt, QObject, QTimer
5  from PyQt6.QtGui import QMovie, QPixmap
6  from PyQt6.QtWidgets import QLabel
7  
8  from variables import SERVER_ADDRESS
9  
10 
11 class Updater(QObject):
12     finished = pyqtSignal()
13     error = pyqtSignal(str)
14     add_file = pyqtSignal(int, str, str, str)
15 
16     def __init__(self):
17         super(Updater, self).__init__()
18 
19     def run(self):
20         try:
21             response = requests.get(f"{SERVER_ADDRESS}/files/")
22             if response.status_code != 200:
23                 self.error.emit("Не удалось получить корректный ответ от сервера!")
24                 self.finished.emit()
25                 return
26             for file in json.loads(response.text):
```

```
                self.add_file.emit(file['file_id'], file['file_name'], file['upload_time'],
                ↪  file['user'])
        except requests.ConnectionError:
            self.error.emit("Не удалось подключиться к серверу!")
        self.finished.emit()


class UpdateBtn(QLabel):
    clicked = pyqtSignal()

    def __init__(self):
        super(UpdateBtn, self).__init__()
        self.setFixedSize(32, 32)
        self.setScaledContents(True)
        self.__image = QPixmap("images/update.png")
        self.setPixmap(self.__image)
        self.__finish_image = QPixmap("images/done.png")
        self.__movie = QMovie("images/updating.gif")
        self.__timer = QTimer()
        self.__timer.timeout.connect(self.open_to_update)
        self.__block = False

    def mousePressEvent(self, event):
        if event.button() == Qt.MouseButton.LeftButton and not self.__block:
            self.__block = True
            self.setMovie(self.__movie)
            self.__movie.start()
            self.clicked.emit()

    def stop_updating(self):
        self.__movie.stop()
        self.setPixmap(self.__finish_image)
        self.__timer.singleShot(1000, self.open_to_update)

    def open_to_update(self):
        self.setPixmap(self.__image)
        self.__block = False
```

### 6.2.6 file_name_item.py

```
from PyQt6.QtCore import Qt
from PyQt6.QtGui import QFont
from PyQt6.QtWidgets import QTableWidgetItem


class FileNameItem(QTableWidgetItem):
```

```
    def __init__(self, name: str, index: int):
        super(QTableWidgetItem, self).__init__(name)
        self.file_id = index
        self.setTextAlignment(Qt.AlignmentFlag.AlignCenter)
        self.setFont(QFont("Times New Roman", 11))
```

## 6.3 Алгоритмы шифрования

### 6.3.1 simplicity_tests.py

```
import math
import random
from abc import ABC, abstractmethod
from enum import Enum


class SimplicityTest(ABC):
    @abstractmethod
    def check(self, number: int, precision: float) -> bool:
        pass


class TestMode(Enum):
    FERMAT = 0
    SOLOVEY_STRASSEN = 1
    MILLER_RABIN = 2


class LegendreSymbol:
    def calculate(self, first: int, second: int) -> int:
        if first <= 0:
            raise ValueError("Числитель не является целым числом!")
        if second <= 2:
            raise ValueError("Знаменатель должен быть больше 2-х!")
        if not second % 2:
            raise ValueError("Знаменатель является чётным числом!")
        if not (comp := first % second):
            return 0
        if comp == 1:
            return 1
        value = 1 if not ((comp - 1) * (second - 1) >> 2 & 1) else -1
        if not comp % 2:
            return self.calculate(second % comp, comp) * value
        value = 2 if not ((second * second - 1) >> 3 & 1) else 1
        return self.calculate(comp >> 1, second) * value

```

```python

class JacobiSymbol:
    def calculate(self, first: int, second: int) -> int:
        if first <= 0:
            raise ValueError("Числитель не является целым числом!")
        if second <= 1:
            raise ValueError("Знаменатель должен быть больше единицы!")
        if not second % 2:
            raise ValueError("Знаменатель является чётным числом!")
        if first == 1:
            return 1
        value = 1 if not (second >> 1 & 1) else -1
        if first < 0:
            return self.calculate(-first, second) * value
        value = 1 if not ((second * second - 1) >> 3 & 1) else -1
        if not first % 2:
            return self.calculate(first >> 1, second) * value
        value = 1 if not (((first - 1) * (second - 1)) >> 2 & 1) else -1
        return self.calculate(second % first, first) * value


class FermatTest(SimplicityTest):
    def check(self, number: int, probability: float) -> bool:
        if number < 2:
            raise ValueError("Число должно быть больше 1!")
        if number == 2:
            return True
        random_set = set()
        for i in range(math.ceil(-math.log2(1 - probability))):
            random_value = random.randint(2, number - 1)
            while random_value in random_set:
                random_value = random.randint(2, number - 1)
            if math.gcd(random_value, number) != 1 or pow(random_value, number - 1, number) != 1:
                return False
            random_set.add(random_value)
            if len(random_set) == number - 2:
                return True
        return True


class SoloveyStrassenTest(SimplicityTest):
    def check(self, number: int, probability: float) -> bool:
        if number < 2:
            raise ValueError("Число должно быть больше 1!")
        if number == 2:
            return True
        jacobi_object = JacobiSymbol()
```

```
84          random_set = set()
85          for i in range(math.ceil(-math.log2(1 - probability))):
86              random_value = random.randint(2, number - 1)
87              while random_value in random_set:
88                  random_value = random.randint(2, number - 1)
89              if math.gcd(random_value, number) != 1 or pow(random_value, number >> 1, number) != \
90                      jacobi_object.calculate(random_value, number):
91                  return False
92              random_set.add(random_value)
93              if len(random_set) == number - 2:
94                  return True
95          return True
96
97
98  class MillerRabinTest(SimplicityTest):
99      def check(self, number: int, probability: float) -> bool:
100         if number < 2:
101             raise ValueError("Число должно быть больше 1!")
102         if number == 2:
103             return True
104         t = number - 1
105         difference = 0
106         while not 1 & t:
107             difference += 1
108             t >>= 1
109         out = False
110         random_set = set()
111         for i in range(math.ceil(-math.log(1 - probability, 4))):
112             if len(random_set) == number - 2:
113                 return True
114             random_value = random.randint(2, number - 1)
115             while random_value in random_set:
116                 random_value = random.randint(2, number - 1)
117             x = pow(random_value, t, number)
118             random_set.add(random_value)
119             if x == 1 or x == number - 1:
120                 continue
121             for j in range(difference - 1):
122                 x = (x * x) % number
123                 if x == 1:
124                     return False
125                 if x == number - 1:
126                     out = True
127                     break
128             if out:
129                 out = False
130                 continue
```

```
131             return False
132         return True
```

### 6.3.2 cryption_algorithms.py

```
1  import copy
2  import math
3  import random
4  from typing import Optional, Tuple
5
6  import simplicity_tests as st
7  import variables
8
9
10 class RC6:
11     def __init__(self, w: int = 32, r: int = 20, b: int = 16):
12         """
13         :param w: длина одного слова из 4-х [16, 32, 64]
14         :param r: число раундов [0..255]
15         :param b: длина ключа в байтах [0..255]
16         """
17         self.__block_length = w if w in {16, 32, 64} else 32
18         self.__block_count = self.__block_length >> 1
19         self.__block_size = self.__block_count >> 2
20         self.__mod_value = 2 ** self.__block_length - 1
21         self.__rounds = max(min(255, r), 0)
22         self.__key_length = max(min(255, b - b % 8), 0)
23         self.__keys = None
24
25     def key_extension(self, key: bytes) -> None:
26         if len(key) != self.__key_length:
27             raise ValueError("Длина ключа не соответствует заявленной!")
28
29         p = {16: 0xb7e1, 32: 0xb7e15163, 64: 0xb7e151628aed2a6b}.get(self.__block_length)
30         q = {16: 0x9e37, 32: 0x9e3779b9, 64: 0x9e3779b97f4a7c15}.get(self.__block_length)
31
32         word_byte_length = self.__block_length // 8
33         if self.__key_length % word_byte_length:
34             key = b'\x00' * (word_byte_length - (self.__key_length % word_byte_length)) + key
35         words = [int.from_bytes(key[i: i + word_byte_length], byteorder='big')
36                  for i in range(0, len(key), word_byte_length)] if self.__key_length else [0]
37
38         self.__keys = [p]
39         double_rounds = 2 * (self.__rounds + 2)
40         for i in range(1, double_rounds):
41             self.__keys.append(self.__keys[-1] + q)
```

```
42
43          g = h = i = j = 0
44          for runner in range(3 * max(len(words), double_rounds)):
45              g = self.__keys[i] = self.left_shift((self.__keys[i] + g + h), 3)
46              h = words[j] = self.left_shift(words[j] + g + h, g + h)
47              i = (j + 1) % double_rounds
48              j = (j + 1) % len(words)
49
50      def encrypt(self, data: bytes) -> bytes:
51          if len(data) != self.__block_count:
52              raise ValueError("Некорректная длина данных!")
53          a, b, c, d = [int.from_bytes(data[i:i + self.__block_size], byteorder='big')
54                        for i in range(0, self.__block_count, self.__block_size)]
55          b = (b + self.__keys[0]) & self.__mod_value
56          d = (d + self.__keys[1]) & self.__mod_value
57          logarithm = int(math.log10(self.__block_length))
58          for i in range(1, self.__rounds + 1):
59              t = self.left_shift((b * (2 * b + 1)) & self.__mod_value, logarithm)
60              u = self.left_shift((d * (2 * d + 1)) & self.__mod_value, logarithm)
61              a = (self.left_shift(a ^ t, u) + self.__keys[2 * i]) & self.__mod_value
62              c = (self.left_shift(c ^ u, t) + self.__keys[2 * i + 1]) & self.__mod_value
63              a, b, c, d = b, c, d, a
64          a = (a + self.__keys[-2]) & self.__mod_value
65          c = (c + self.__keys[-1]) & self.__mod_value
66          return a.to_bytes(self.__block_size, 'big') + b.to_bytes(self.__block_size, 'big') + \
67              c.to_bytes(self.__block_size, 'big') + d.to_bytes(self.__block_size, 'big')
68
69      def decrypt(self, data) -> bytes:
70          if len(data) != self.__block_count:
71              raise ValueError("Некорректная длина данных!")
72          a, b, c, d = [int.from_bytes(data[i:i + self.__block_size], byteorder='big')
73                        for i in range(0, self.__block_count, self.__block_size)]
74          a = self.minus_modulo(a, self.__keys[-2])
75          c = self.minus_modulo(c, self.__keys[-1])
76          logarithm = int(math.log10(self.__block_length))
77          for i in range(self.__rounds, 0, -1):
78              a, b, c, d = d, a, b, c
79              t = self.left_shift((b * (2 * b + 1)) & self.__mod_value, logarithm)
80              u = self.left_shift((d * (2 * d + 1)) & self.__mod_value, logarithm)
81              a = (self.right_shift(self.minus_modulo(a, self.__keys[2 * i]), u) ^ t)
82              c = (self.right_shift(self.minus_modulo(c, self.__keys[2 * i + 1]), t) ^ u)
83          b = self.minus_modulo(b, self.__keys[0])
84          d = self.minus_modulo(d, self.__keys[1])
85          return a.to_bytes(self.__block_size, 'big') + b.to_bytes(self.__block_size, 'big') + \
86              c.to_bytes(self.__block_size, 'big') + d.to_bytes(self.__block_size, 'big')
87
88      def minus_modulo(self, a: int, b: int) -> int:
```

```
        return a - b & self.__mod_value

    def left_shift(self, x: int, n: int) -> int:
        n = n % self.__block_length
        return (x << n & (2 ** self.__block_length - 1)) | (x >> (self.__block_length - n))

    def right_shift(self, x: int, n: int) -> int:
        n = n % self.__block_length
        return (x >> n) | ((x & (2 ** n - 1)) << (self.__block_length - n))


class GFP2Element:
    def __init__(self, p: int, value: Optional[int] = None, params: Optional[Tuple[int, int]] = None):
        self.__p = p
        self.__a, self.__b = None, None
        if value is None:
            if params is not None:
                self.__a, self.__b = params[0] % p, params[1] % p
            else:
                self.randomize()
        else:
            self.__a = self.__b = (-value) % p

    def randomize(self) -> None:
        while True:
            self.__a = random.randint(0, self.__p - 1)
            self.__b = random.randint(0, self.__p - 1)
            if self.__a != self.__b:
                return

    def get_swapped(self) -> 'GFP2Element':
        return type(self)(self.__p, params=(self.__b, self.__a))

    def get_square(self) -> 'GFP2Element':
        return type(self)(self.__p, params=(self.__b * (self.__b - 2 * self.__a), self.__a * (self.__a
        ↪  - 2 * self.__b)))

    def __eq__(self, other: Optional['GFP2Element']) -> bool:
        if other is None:
            return False
        return self.__a == other.__a and self.__b == other.__b

    def is_GFP(self) -> bool:
        return self.__a == self.__b

    def __sub__(self, other) -> 'GFP2Element':
        return type(self)(self.__p, params=(self.__a - other.__a, self.__b - other.__b))
```

```
135
136     def __add__(self, other) -> 'GFP2Element':
137         return type(self)(self.__p, params=(self.__a + other.__a, self.__b + other.__b))
138
139     @staticmethod
140     def special_operation(x: 'GFP2Element', y: 'GFP2Element', z: 'GFP2Element') -> 'GFP2Element':
141         return type(x)(x.__p, params=(z.__a * (y.__a - x.__b - y.__b) + z.__b * (x.__b - x.__a +
        ↪  y.__b),
142                                       z.__a * (x.__a - x.__b + y.__a) + z.__b * (y.__b - x.__a -
                                      ↪  y.__a)))
143
144     def __str__(self) -> str:
145         return f"GPP2Element({self.__a}, {self.__b})"
146
147     def __copy__(self) -> 'GFP2Element':
148         return type(self)(self.__p, params=(self.__a, self.__b))
149
150     def get_bytes(self, byte_length: int) -> bytes:
151         """
152         Получение байтового представление
153         :param byte_length: Длина байтового представления одного числа
154         [Лучше использовать байты для: XTR_bit_length * 3]
155         :return: Байтовое представление длиной 2 * byte_length
156         """
157         return self.__a.to_bytes(byte_length, 'big') + self.__b.to_bytes(byte_length, 'big')
158
159     def get_values(self) -> Tuple[int, int]:
160         return self.__a, self.__b
161
162
163 class XTR:
164     def __init__(self, test_mode: st.TestMode, probability: float = 0.9, bit_length: int = 128):
165         match test_mode:
166             case st.TestMode.FERMAT:
167                 self.__test: st.SimplicityTest = st.FermatTest()
168             case st.TestMode.SOLOVEY_STRASSEN:
169                 self.__test: st.SimplicityTest = st.SoloveyStrassenTest()
170             case st.TestMode.MILLER_RABIN:
171                 self.__test: st.SimplicityTest = st.MillerRabinTest()
172         self.__probability = probability
173         self.__bit_length = bit_length
174         self.public_key = None
175
176     def generate_key(self) -> Tuple[int, int, Tuple[int, int]]:
177         while True:
178             r = random.getrandbits(self.__bit_length)
179             q = r ** 2 - r + 1
```

```
            if q % 12 == 7 and self.__test.check(q, self.__probability):
                break
        while True:
            k = random.getrandbits(self.__bit_length)
            p = r + k * q
            if p % 3 == 2 and self.__test.check(p, self.__probability):
                break

        quotient = (p ** 2 - p + 1) // q
        c = GFP2Element(p)
        three = GFP2Element(p, 3)
        tracer = self.Tracer(p)

        while True:
            if not tracer.calculate_tr(p + 1, c).is_GFP():
                if (tr := tracer.calculate_tr(quotient)) != three:
                    break
            c.randomize()

        self.public_key = (p, q, tr.get_values())
        return self.public_key

    def get_el_gamal_key(self) -> Tuple[int, Tuple[int, int]]:
        tracer = self.Tracer(self.public_key[0])
        k = random.randint(2, self.public_key[1] - 3)
        trace_g_k = tracer.calculate_tr(k, GFP2Element(self.public_key[0], params=self.public_key[2]))
        return k, trace_g_k.get_values()

    @staticmethod
    def get_symmetric_key_back(p: int, k: int, tr: Tuple[int, int]) -> bytes:
        tracer = XTR.Tracer(p)
        return tracer.calculate_tr(k, GFP2Element(p, params=tr)).get_bytes(3 *
        ↪  (variables.XTR_KEY_BIT_SIZE >> 3))

    @staticmethod
    def get_symmetric_key(p: int, q: int, tr: Tuple[int, int], tr_k: Tuple[int, int]) \
            -> Tuple[Tuple[int, int], bytes]:
        b = random.randint(2, q - 3)
        tracer = XTR.Tracer(p)
        trace_g_b = tracer.calculate_tr(b, GFP2Element(p, params=tr))
        trace_g_bk = tracer.calculate_tr(b, GFP2Element(p, params=tr_k))
        return trace_g_b.get_values(), trace_g_bk.get_bytes(3 * (variables.XTR_KEY_BIT_SIZE >> 3))

    class Tracer:
        def __init__(self, p: int):
            self.__p = p
            self.__c: Optional[GFP2Element] = None
```

```
226             self.__c_dict: Optional[dict] = None
227
228         def calculate_tr(self, n: int, c: Optional[GFP2Element] = None) -> GFP2Element:
229             if c is not None and c != self.__c:
230                 self.__c = copy.copy(c)
231                 self.__c_dict = {0: GFP2Element(self.__p, 3), 1: self.__c}
232             if self.__c is None:
233                 raise ValueError('Отсутствует значение "c"!')
234             return self.__calculate_c(n)
235
236         def __calculate_c(self, n: int) -> GFP2Element:
237             if n in self.__c_dict:
238                 return self.__c_dict[n]
239             current_n = 1
240             for bit in map(int, bin(n)[3:]):
241                 new_n = (current_n << 1) | bit
242                 if new_n not in self.__c_dict:
243                     current_c = self.__c_dict[current_n]
244                     self.__c_dict[new_n] = (GFP2Element.special_operation(self.__calculate_c(current_n
                        ↪  + 1), self.__c,
245                                                                           current_c) +
246                                             self.__calculate_c(current_n - 1).get_swapped()) if bit
                                                ↪  else \
247                         (current_c.get_square() - (current_c + current_c).get_swapped())
248                 current_n = new_n
249             return self.__c_dict[n]
```

### 6.3.3 file_encryption.py

```
1  import enum
2  import math
3  from multiprocessing import cpu_count
4  from multiprocessing.pool import ThreadPool
5  from queue import Queue
6
7  from PyQt6.QtCore import QObject, pyqtSignal
8
9  import variables
10 from cryption_algorithms import RC6
11
12
13 class AggregatorMode(enum.Enum):
14     ECB = 0
15     CBC = 1
16     CFB = 2
17     OFB = 3
```

```
18      CTR = 4
19      RD = 5
20      RDH = 6
21
22
23  class ModeECB:
24      def __init__(self, block_size, algorithm: RC6):
25          self.__block_size = block_size
26          self.__algorithm = algorithm
27
28      def encrypt(self, data):
29          with ThreadPool(processes=cpu_count()) as pool:
30              return [byte for block in pool.map(self.__algorithm.encrypt,
31                                                 [data[i: i + self.__block_size]
32                                                  for i in range(0, len(data), self.__block_size)]) for
                                                  ↪  byte in block]
33
34      def decrypt(self, data):
35          with ThreadPool(processes=cpu_count()) as pool:
36              return [byte for block in pool.map(self.__algorithm.decrypt,
37                                                 [data[i: i + self.__block_size]
38                                                  for i in range(0, len(data), self.__block_size)]) for
                                                  ↪  byte in block]
39
40
41  class ModeCBC:
42      def __init__(self, block_size, algorithm: RC6, init):
43          self.__block_size = block_size
44          self.__algorithm = algorithm
45          self.__previous_block = init
46
47      def encrypt(self, data):
48          result = []
49          for i in range(0, len(data), self.__block_size):
50              self.__previous_block = self.__algorithm.encrypt(bytes(f ^ s for f, s in
51                                                                     zip(self.__previous_block,
52                                                                         data[i: i +
                                                                          ↪  self.__block_size])))
53              result.extend(self.__previous_block)
54          return result
55
56      def decrypt(self, data):
57          result = []
58          for i in range(0, len(data), self.__block_size):
59              result.extend(list(f ^ s for f, s in zip(self.__previous_block,
60                                                      self.__algorithm.decrypt(data[i: i +
                                                       ↪  self.__block_size]))))
```

```
61              self.__previous_block = data[i: i + self.__block_size]
62          return result
63
64
65  class ModeCFB:
66      def __init__(self, block_size, algorithm: RC6, init):
67          self.__block_size = block_size
68          self.__algorithm = algorithm
69          self.__previous_block = init
70
71      def encrypt(self, data):
72          result = []
73          for i in range(0, len(data), self.__block_size):
74              self.__previous_block = bytes(f ^ s for f, s in
                ↪  zip(self.__algorithm.encrypt(self.__previous_block),
75                                                                  data[i: i + self.__block_size]))
76              result.extend(self.__previous_block)
77          return result
78
79      def decrypt(self, data):
80          result = []
81          for i in range(0, len(data), self.__block_size):
82              result.extend(list(f ^ s for f, s in zip(self.__algorithm.encrypt(self.__previous_block),
83                                                      data[i: i + self.__block_size])))
84              self.__previous_block = data[i: i + self.__block_size]
85          return result
86
87
88  class ModeOFB:
89      def __init__(self, block_size, algorithm: RC6, init):
90          self.__block_size = block_size
91          self.__algorithm = algorithm
92          self.__previous_block = init
93
94      def encrypt(self, data):
95          result = []
96          for i in range(0, len(data), self.__block_size):
97              self.__previous_block = self.__algorithm.encrypt(self.__previous_block)
98              result.extend(list(f ^ s for f, s in zip(self.__previous_block, data[i: i +
                ↪  self.__block_size])))
99          return result
100
101     def decrypt(self, data):
102         result = []
103         for i in range(0, len(data), self.__block_size):
104             self.__previous_block = self.__algorithm.encrypt(self.__previous_block)
```

```
105             result.extend(list(f ^ s for f, s in zip(self.__previous_block, data[i: i +
                ↪  self.__block_size])))
106         return result
107
108
109 class ModeCTR:
110     def __init__(self, block_size, algorithm: RC6):
111         self.__block_size = block_size
112         self.__algorithm = algorithm
113         self.__counter = 1
114
115     def encrypt(self, data):
116         result = []
117         with ThreadPool(processes=cpu_count()) as pool:
118             for (index, block) in enumerate(pool.map(self.__algorithm.encrypt,
119                                                      (i.to_bytes(self.__block_size, byteorder="big")
120                                                       for i in range(self.__counter,
121                                                                      self.__counter +
122                                                                      math.ceil(len(data) /
                                                                      ↪  self.__block_size))))):
123                 pos = index * self.__block_size
124                 result.extend(list(f ^ s for f, s in zip(block, data[pos: pos + self.__block_size])))
125         self.__counter += math.ceil(len(data) / self.__block_size)
126         return result
127
128     def decrypt(self, data):
129         result = []
130         with ThreadPool(processes=cpu_count()) as pool:
131             for (index, block) in enumerate(pool.map(self.__algorithm.encrypt,
132                                                      (i.to_bytes(self.__block_size, byteorder="big")
133                                                       for i in range(self.__counter,
134                                                                      self.__counter +
135                                                                      math.ceil(len(data) /
                                                                      ↪  self.__block_size))))):
136                 pos = index * self.__block_size
137                 result.extend(list(f ^ s for f, s in zip(block, data[pos: pos + self.__block_size])))
138         self.__counter += math.ceil(len(data) / self.__block_size)
139         return result
140
141
142 class ModeRD:
143     def __init__(self, block_size, algorithm: RC6, init=None):
144         self.__block_size = block_size
145         self.__algorithm = algorithm
146         if init is not None:
147             self.__init_vector = init
148             self.__delta = int.from_bytes(init[len(init) // 2:], byteorder='big')
```

```
149         else:
150             self.__init_vector = None
151             self.__delta = None
152         self.__block_value = None
153 
154     def encrypt(self, data):
155         result = []
156         blocks = []
157         if self.__block_value is None:
158             self.__block_value = int.from_bytes(self.__init_vector, byteorder='big')
159             blocks.append(self.__init_vector)
160         blocks.extend([[a ^ b for a, b in zip(f, s)] for f, s in
161                        zip((i.to_bytes(self.__block_size, byteorder='big')
162                             for i in range(self.__block_value,
163                                            self.__block_value + math.ceil(len(data) /
                                               ↪  self.__block_size) * self.__delta,
164                                            self.__delta)),
165                            (data[i: i + self.__block_size] for i in range(0, len(data),
                               ↪  self.__block_size)))])
166         with ThreadPool(processes=cpu_count()) as pool:
167             for block in pool.map(self.__algorithm.encrypt, blocks):
168                 result.extend(block)
169         self.__block_value += math.ceil(len(data) / self.__block_size) * self.__delta
170         return result
171 
172     def decrypt(self, data):
173         result = []
174         if self.__block_value is None:
175             self.__delta =
                ↪  int.from_bytes(self.__algorithm.decrypt(data[:self.__block_size])[self.__block_size //
                ↪  2:],
176                                       byteorder='big')
177             self.__block_value = int.from_bytes(self.__algorithm.decrypt(data[:self.__block_size]),
                ↪  byteorder='big')
178             data = data[self.__block_size:]
179         with ThreadPool(processes=cpu_count()) as pool:
180             for block in pool.map(self.__algorithm.decrypt, [data[i: i + self.__block_size]
181                                                              for i in range(0, len(data),
                                                                 ↪  self.__block_size)]):
182                 result.extend(f ^ s for f, s in zip(block,
183                                                    self.__block_value.to_bytes(self.__block_size,
                                                       ↪  byteorder='big')))
184                 self.__block_value += self.__delta
185         return result
186 
187 
188 class ModeRDH:
```

```
189     def __init__(self, block_size, algorithm: RC6, init=None):
190         self.__block_size = block_size
191         self.__algorithm = algorithm
192         if init is not None:
193             self.__init_vector = init
194             self.__delta = int.from_bytes(init[len(init) // 2:], byteorder='big')
195         else:
196             self.__init_vector = None
197             self.__delta = None
198         self.__block_value = None
199 
200     def encrypt(self, data):
201         result = []
202         blocks = []
203         if self.__block_value is None:
204             self.__block_value = int.from_bytes(self.__init_vector, byteorder='big')
205             blocks.append(self.__init_vector)
206             blocks.append(hash(tuple(data)).to_bytes(self.__block_size, byteorder='big', signed=True))
207         blocks.extend([[a ^ b for a, b in zip(f, s)] for f, s in
208                        zip((i.to_bytes(self.__block_size, byteorder='big')
209                             for i in range(self.__block_value,
210                                            self.__block_value + math.ceil(len(data) /
                                               ↪  self.__block_size) * self.__delta,
211                                            self.__delta)),
212                            (data[i: i + self.__block_size] for i in range(0, len(data),
                            ↪  self.__block_size)))])
213         with ThreadPool(processes=cpu_count()) as pool:
214             for block in pool.map(self.__algorithm.encrypt, blocks):
215                 result.extend(block)
216         self.__block_value += math.ceil(len(data) / self.__block_size) * self.__delta
217         return result
218 
219     def decrypt(self, data):
220         result = []
221         hash_value = None
222         if self.__block_value is None:
223             self.__delta =
                ↪  int.from_bytes(self.__algorithm.decrypt(data[:self.__block_size])[self.__block_size //
                ↪  2:],
224                                               byteorder='big')
225             self.__block_value = int.from_bytes(self.__algorithm.decrypt(data[:self.__block_size]),
                ↪  byteorder='big')
226             hash_value = int.from_bytes(self.__algorithm.decrypt(data[self.__block_size:2 *
                ↪  self.__block_size]),
227                                         byteorder='big', signed=True)
228             data = data[2 * self.__block_size:]
229         with ThreadPool(processes=cpu_count()) as pool:
```

```
            for block in pool.map(self.__algorithm.decrypt, [data[i: i + self.__block_size]
                                                             for i in range(0, len(data),
                                                             ↪ self.__block_size)]):
                result.extend(f ^ s for f, s in zip(block,
                ↪ self.__block_value.to_bytes(self.__block_size,

                                                                                ↪ byteorder='big')))
                self.__block_value += self.__delta
        if hash_value is not None and hash_value != hash(tuple(result)):
            raise ValueError('[RDH] Подмена данных!')
        return result


class Encrypter(QObject):
    progress = pyqtSignal(int)
    thread_queue = Queue(variables.THREAD_QUEUE_SIZE)

    def shutdown(self):
        self.__exit = True
        self.deleteLater()

    def __init__(self, algorithm: RC6, key: bytes, mode: AggregatorMode, init_vector=None, **kwargs):
        super(Encrypter, self).__init__()
        self.__algorithm = algorithm
        self.__algorithm.key_extension(key)
        self.__mode = mode
        self.__init_vector = init_vector
        self.__block_size = kwargs['block_size'] if 'block_size' in kwargs else 8
        self.__progress = 0
        self.__exit = False

    def encrypt(self, in_file: str, out_file: str):
        self.thread_queue.put(True)
        if self.__exit:
            self.thread_queue.get()
            return
        self.__progress = 0
        with (open(in_file, 'rb') as f_in,
              open(out_file, 'wb') as f_out):
            match self.__mode:
                case AggregatorMode.ECB:
                    mode_aggregator = ModeECB(self.__block_size, self.__algorithm)
                case AggregatorMode.CBC:
                    mode_aggregator = ModeCBC(self.__block_size, self.__algorithm, self.__init_vector)
                case AggregatorMode.CFB:
                    mode_aggregator = ModeCFB(self.__block_size, self.__algorithm, self.__init_vector)
                case AggregatorMode.OFB:
```

```
                    mode_aggregator = ModeOFB(self.__block_size, self.__algorithm, self.__init_vector)
                case AggregatorMode.CTR:
                    mode_aggregator = ModeCTR(self.__block_size, self.__algorithm)
                case AggregatorMode.RD:
                    mode_aggregator = ModeRD(self.__block_size, self.__algorithm, self.__init_vector)
                case AggregatorMode.RDH:
                    mode_aggregator = ModeRDH(self.__block_size, self.__algorithm, self.__init_vector)
            while (block := f_in.read(self.__block_size * variables.BLOCK_ENCRYPT_SIZE)) and not
            ↪  self.__exit:
                count = self.__block_size - len(block) % self.__block_size
                f_out.write(bytes(mode_aggregator.encrypt(block + count.to_bytes(1, 'big') * count)))
                self.__progress += len(block)
                self.progress.emit(self.__progress)
        self.thread_queue.get()

    def decrypt(self, in_file: str, out_file: str):
        self.thread_queue.put(True)
        if self.__exit:
            self.thread_queue.get()
            return
        self.__progress = 0
        with (open(in_file, 'rb') as f_in,
              open(out_file, 'wb') as f_out):
            match self.__mode:
                case AggregatorMode.ECB:
                    mode_aggregator = ModeECB(self.__block_size, self.__algorithm)
                case AggregatorMode.CBC:
                    mode_aggregator = ModeCBC(self.__block_size, self.__algorithm, self.__init_vector)
                case AggregatorMode.CFB:
                    mode_aggregator = ModeCFB(self.__block_size, self.__algorithm, self.__init_vector)
                case AggregatorMode.OFB:
                    mode_aggregator = ModeOFB(self.__block_size, self.__algorithm, self.__init_vector)
                case AggregatorMode.CTR:
                    mode_aggregator = ModeCTR(self.__block_size, self.__algorithm)
                case AggregatorMode.RD:
                    mode_aggregator = ModeRD(self.__block_size, self.__algorithm, self.__init_vector)
                case AggregatorMode.RDH:
                    mode_aggregator = ModeRDH(self.__block_size, self.__algorithm, self.__init_vector)
            while (block := f_in.read(self.__block_size * (variables.BLOCK_ENCRYPT_SIZE + 3)
                                      if self.__mode == AggregatorMode.RDH else
                                      self.__block_size * (variables.BLOCK_ENCRYPT_SIZE + 1))) and not
                                      ↪  self.__exit:
                result = mode_aggregator.decrypt(block)
                del result[-result[-1]:]
                f_out.write(bytes(result))
                self.__progress += len(block)
                self.progress.emit(self.__progress)
```

```
319                 self.thread_queue.get()
```

## 6.4 Прочее

### 6.4.1 variables.py

```
1  from typing import Final
2
3  import simplicity_tests as st
4
5  ICON_PATH: Final[str] =
   ↪  r'C:\Users\anton\PycharmProjects\HybridEncryptionSystem\client\images\icon.ico'
6  TEMP_DIR: Final[str] = r'C:\Users\anton\PycharmProjects\HybridEncryptionSystem\client\temp'
7  DATA_DIR: Final[str] = r'C:\Users\anton\PycharmProjects\HybridEncryptionSystem\server\data_directory'
8  SERVER_ADDRESS: Final[str] = 'http://127.0.0.1:5000'
9  BLOCK_ENCRYPT_SIZE: Final[int] = 10000
10 RC6_WORD_BIT_SIZE: Final[int] = 64
11 RC6_ROUND_COUNT: Final[int] = 20
12 RC6_KEY_BYTE_SIZE: Final[int] = 16
13 THREAD_QUEUE_SIZE: Final[int] = 3
14 TEST: Final[st.TestMode] = st.TestMode.MILLER_RABIN
15 TEST_PRECISION: Final[float] = 0.999
16 XTR_KEY_BIT_SIZE: Final[int] = 128
17 MAX_BYTE_FILE_SIZE: Final[int] = 50 * 1024 * 1024
```