ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ
ПО ДИСЦИПЛИНЕ
«Защита информации»

Руководители _____ Е.А. Харченко

Москва 2023

СПИСОК ИСПОЛНИТЕЛЕЙ

Студ. группы 201-361 _____ А.Е. Сильченко

Ход работы:

1) Генерируем ключ используя OpenSSSL и записываем его в файл рисунок 1.

```
// генерация ключа
String keyPath = "key.txt";

String command = "openssl rand -hex 16";

Process process = Runtime.getRuntime().exec(command);

InputStream inputStream = process.getInputStream();
BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));

String key = reader.readLine();

File file = new File(keyPath);
FileWriter fw = new FileWriter(file);
fw.write(key);
fw.close();
```

Рисунок 1 – Генерация ключа.

2) Создаем 2 временных файла, для хранения тела изображения (без заголовка) и для хранения зашифрованного тела изображения, эти же файлы будут использоваться для расшифровки файла только наоборот, в первом будет хранится зашифрованное тело изображения, во втором расшифрованное. Также создаем два выходных файла для зашифрованного файла (создан ниже) и для расшифрованного. В метод для шифрования передаем следующие параметры: режим шифрования, исходное изображение, ключ, зашифрованное изображение и 2 временных файла рисунок 2.

```
// зашифровка и расшифровка изображения в режиме ECB
String fileECB1 = new File( pathname: "temp\\ECB1.txt").getPath();
String fileECB2 = new File( pathname: "temp\\ECB2.txt").getPath();
String de_ECB = new File( pathname: "de_ECB.bmp").getPath();
encryptImage( mode: "ECB",  inputFilename: "tux.bmp", keyPath,  outputFilename: "tux_ecb.bmp",fileECB1, fileECB2 );
decryptImage( mode: "ECB",  inputFilename: "tux_ecb.bmp", keyPath, de_ECB,fileECB1, fileECB2 );

// зашифровка и расшифровка изображения в режиме CBC
String fileCBC1 = new File( pathname: "temp\\CBC1.txt").getPath();
String fileCBC2 = new File( pathname: "temp\\CBC2.txt").getPath();
String de_CBC = new File( pathname: "de_CBC.bmp").getPath();
encryptImage( mode: "CBC",  inputFilename: "tux.bmp", keyPath,  outputFilename: "tux_cbc.bmp", fileCBC1, fileCBC2);
decryptImage( mode: "CBC",  inputFilename: "tux_cbc.bmp", keyPath, de_CBC, fileCBC1, fileCBC2);

// зашифровка и расшифровка изображения в режиме CFB
String fileCFB1 = new File( pathname: "temp\\CFB1.txt").getPath();
String fileCFB2 = new File( pathname: "temp\\CFB2.txt").getPath();
String de_CFB = new File( pathname: "de_CFB.bmp").getPath();
encryptImage( mode: "CFB",  inputFilename: "tux.bmp", keyPath,  outputFilename: "tux_cfb.bmp", fileCFB1, fileCFB2);
decryptImage( mode: "CFB",  inputFilename: "tux_cfb.bmp", keyPath, de_CFB, fileCFB1, fileCFB2);

// зашифровка изображения в режиме OFB
String fileOFB1 = new File( pathname: "temp\\OFB1.txt").getPath();
String fileOFB2 = new File( pathname: "temp\\OFB2.txt").getPath();
String de_OFB = new File( pathname: "de_OFB.bmp").getPath();
encryptImage( mode: "OFB",  inputFilename: "tux.bmp", keyPath,  outputFilename: "tux_ofb.bmp", fileOFB1, fileOFB2);
decryptImage( mode: "OFB",  inputFilename: "tux_ofb.bmp", keyPath, de_OFB, fileOFB1, fileOFB2);

// вывод информации о файлах
File tuxFile = new File( pathname: "tux.bmp");
File tuxEcbFile = new File( pathname: "tux_ecb.bmp");
File tuxCbcFile = new File( pathname: "tux_cbc.bmp");
File tuxCfbFile = new File( pathname: "tux_cfb.bmp");
File tuxOfbFile = new File( pathname: "tux_ofb.bmp");
```

Рисунок 2 – Основная часть программы.

3) В методе для шифрования изображения мы получаем байты заголовка (в данном случае заголовок равен 110 байт) и тело изображения. Тело изображения записываем в первый временный файл. Далее в OpenSSL мы передаем режим шифрования который передается этому методу, первый временный файл в котором хранится тело изображения и файл в который будет помещено зашифрованное тело изображения и ключ. Далее мы считываем полученный файл и записываем зашифрованное тело в переменную «shBytes». Далее мы записываем в файл который мы получим на выходе сначала нетронутый заголовок, потом зашифрованное тело рисунок 3.

```
private static void encryptImage(String mode, String inputFilename, String keyFilename, String outputFilename, String file1, String file2)

    //читаем исходный файл
    FileInputStream inputStream = new FileInputStream(inputFilename);
    byte[] header = new byte[110];
    inputStream.read(header); //получаем заголовок
    byte[] imageBytes = new byte[inputStream.available()];
    inputStream.read(imageBytes); //получаем тело
    inputStream.close();

    //записываем в отдельный файл тело изображения
    FileOutputStream outputStream = new FileOutputStream(file1);
    outputStream.write(imageBytes);
    outputStream.close();

    // зашифровка тела файла
    String encryptCommand = ("openssl enc -aes-256-" + mode + " -in " + file1 + " -out " + file2 + " -pass file:" + keyFilename);
    Runtime rt = Runtime.getRuntime();
    Process process = rt.exec(encryptCommand);
    int exitCode = process.waitFor();
    if (exitCode != 0) {
        throw new Exception("Command execution failed with exit code " + exitCode);
    }

    //получения массива байтов зашифрованного тела
    FileInputStream inputStream1 = new FileInputStream(file2);
    byte[] shBytes = new byte[inputStream1.available()];
    inputStream1.read(shBytes);
    inputStream1.close();

    //собираем изображение вместе
    FileOutputStream outputStream1 = new FileOutputStream(outputFilename);
    outputStream1.write(header);
    outputStream1.write(shBytes);
    outputStream1.close();
```

Рисунок 3 – encryptImage.

4) Метод для расшифровки аналогичен методу шифрования, только в OpenSSL передается дополнительная опция «-d» для расшифровки файла. И в этом методе в первом файле будет хранится зашифрованное тело, а во втором расшифрованное тело изображения. Метод представлен на рисунке 5.

```java
private static void decryptImage(String mode, String inputFilename, String keyFilename, String outputFilename, String file1, String file2)

    FileInputStream inputStream = new FileInputStream(inputFilename);
    byte[] header = new byte[110];
    inputStream.read(header); //получаем заголовок
    byte[] imageBytes = new byte[inputStream.available()];
    inputStream.read(imageBytes); //получаем тело
    inputStream.close();

    //записываем в отдельный файл тело цикла
    FileOutputStream outputStream = new FileOutputStream(file1);
    outputStream.write(imageBytes);
    outputStream.close();

    // зашифровка тела файла
    String encryptCommand = ("openssl enc -d -aes-256-" + mode + " -in " + file1 + " -out " + file2 + " -pass file:" + keyFilename);
    Runtime rt = Runtime.getRuntime();
    Process process = rt.exec(encryptCommand);
    int exitCode = process.waitFor();
    if (exitCode != 0) {
        throw new Exception("Command execution failed with exit code " + exitCode);
    }

    //получения массива байтов зашифрованного тела
    FileInputStream inputStream1 = new FileInputStream(file2);
    byte[] shBytes = new byte[inputStream1.available()];
    inputStream1.read(shBytes);
    inputStream1.close();

    //собираем изображение вместе
    FileOutputStream outputStream1 = new FileOutputStream(outputFilename);
    outputStream1.write(header);
    outputStream1.write(shBytes);
    outputStream1.close();
```

Рисунок 4 – decryptImage.

5) В результате мы получаем 4 зашифрованных изображения (представлены ниже) и 4 расшифрованных изображений и дополнительно для сравнения изображений я вывожу их размер в байтах рисунок 5.

```
ux_cfb.bmp
ux_ecb.bmp
ux_ofb.bmp
rnal Libraries
openjdk-20 > C:\Users\Al200\.jdks\oper
ches and Consoles
```

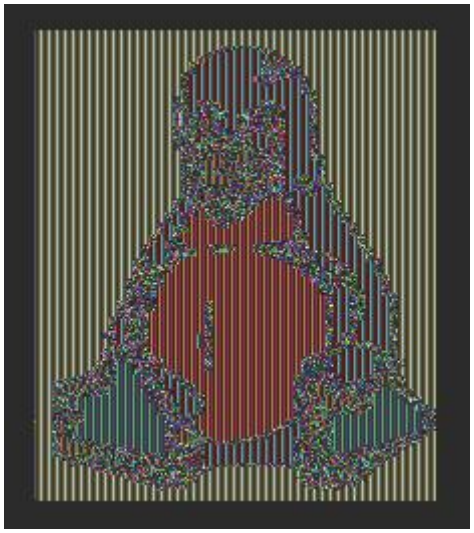
```java
        System.out.println("Оригинальное изображение: " + tuxFile.length() + " bytes");
        System.out.println("ECB image: " + tuxEcbFile.length() + " bytes");
        System.out.println("CBC image: " + tuxCbcFile.length() + " bytes");
        System.out.println("CFB image: " + tuxCfbFile.length() + " bytes");
        System.out.println("OFB image: " + tuxOfbFile.length() + " bytes");

        System.out.println("Шифрование и расшифрование завершено успешно.");
    }
```

```
Main
C:\Users\Al200\.jdks\openjdk-20.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 202
Оригинальное изображение: 188138 bytes
ECB image: 188158 bytes
CBC image: 188158 bytes
CFB image: 188154 bytes
OFB image: 188154 bytes
Шифрование и расшифрование завершено успешно.
```

Рисунок 5 – Размер полученных файлов

Зашифрованное изображение в режиме ECB



Зашифрованное изображение в режиме CBC

Зашифрованное изображение в режиме CFB

Зашифрованное изображение в режиме OFB