МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА 51

КУРСОВАЯ РАБОТА (ПРОЕКТ)
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

ассистент		М.Н. Исаева
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ (ПРОЕКТУ)

СИММЕТРИЧНЫЙ БЛОКОВЫЙ ШИФР

по дисциплине: КРИПТОГРАФИЧЕСКИЕ МЕТТОДЫ ЗАЩИТЫ ИНФОРМАЦИИ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 5911		Е.А. Белов
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2022

Цель работы:

Разработайте и реализуйте симметричный блоковый шифр с ключом и размером блока 16 битов, представить схему шифра для согласования. Произвести атаку методом «грубой силы». Оценить сложность проведения данной атаки для разработанного алгоритма и в общем случае.

Вычислить коэффициент корреляции для входного и выходного потоков алгоритма шифрования, оценить распределение «0» и «1» в выходном потоке. По результатам проведенных тестов сделать выводы о качестве и стойкости реализованных систем шифрования. На каждой итерации выводить результат шифрования для визуальной оценки работы системы шифрования.

Для шифрования необходимо выбрать изображение и показывать его после каждого раунда шифрования. Для исходного изображения и изображения после шифрования - построить график автокорреляции и тест на решетчатость.

Ход выполнения работы:

1. **Описание алгоритма**

На вход алгоритма подается поток данных в виде байтов. После чего данные разбиваются на блоки по 16 бит. Задается случайный ключ размером 128 бита, который так же делится на 8 подключей по 16 бит.

Структура реализованного алгоритма показана на схеме (Рис.1). Процесс шифрования состоит из восьми раундов шифрования и одного выходного преобразования. Операции в раунде зависят от четности раунда.

Четный раунд. В таких раундах происходит операция умножения по модулю $2^{16} + 1$. Каждый блок размером 16 бит умножается на подключ такого же размера.

Нечетный раунд. Исходный или полученный после предыдущего раунда блок делится на две части по 8 бит. Также делится i-ый подключ на части по 8 бит.

- Левая часть блока складывается по модулю двух(XOR) с левой частью ключа.
- Правая часть складывается по модулю 2^8 с правой частью ключа.
- Части меняются местами и склеиваются в блок 16 бит.

Финальный раунд. Результат выполнения восьми раундов умножается по модулю $2^{16} + 1$ на полученную сумму минимальной левой части и максимальной правой части подключа.

Схема представленного алгоритма:

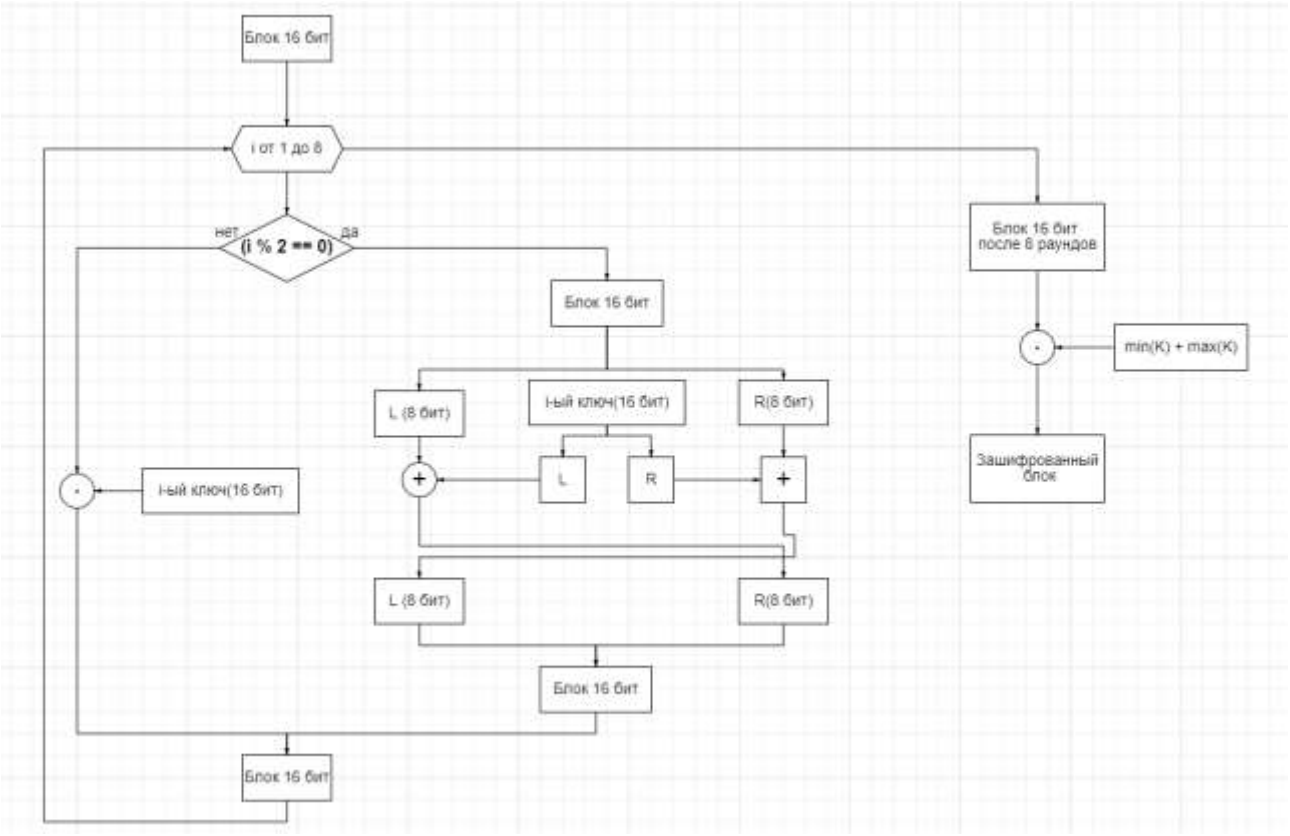

Рис.1 Схема раундов шифрования

2. Результат программной реализации шифрования

Как исходное изображение задается lena.bmp 512x512 пикселей. Представлены результаты после каждого раунда шифрования, а также тест на решетчатость и автокорреляцию.

Исходное изображение:

Рис.2 Исходное изображение lena.bmp

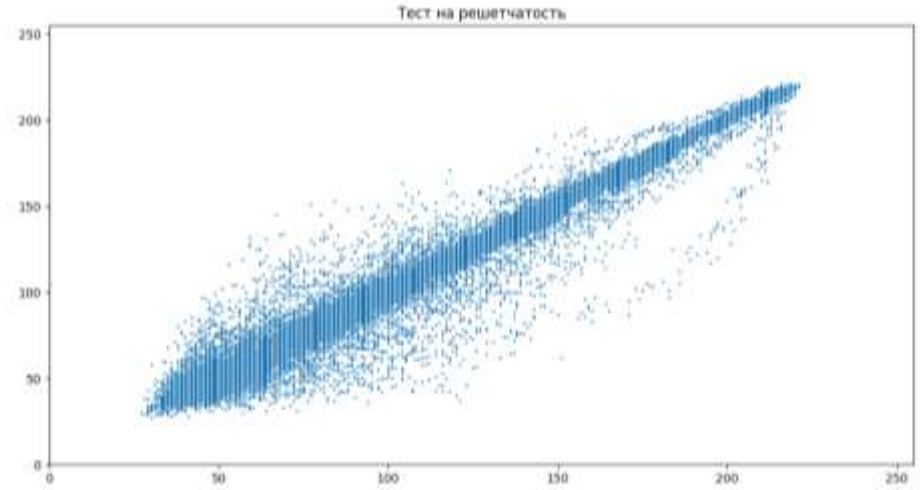

Рис.3 Тест на решетчатость исходного изображения

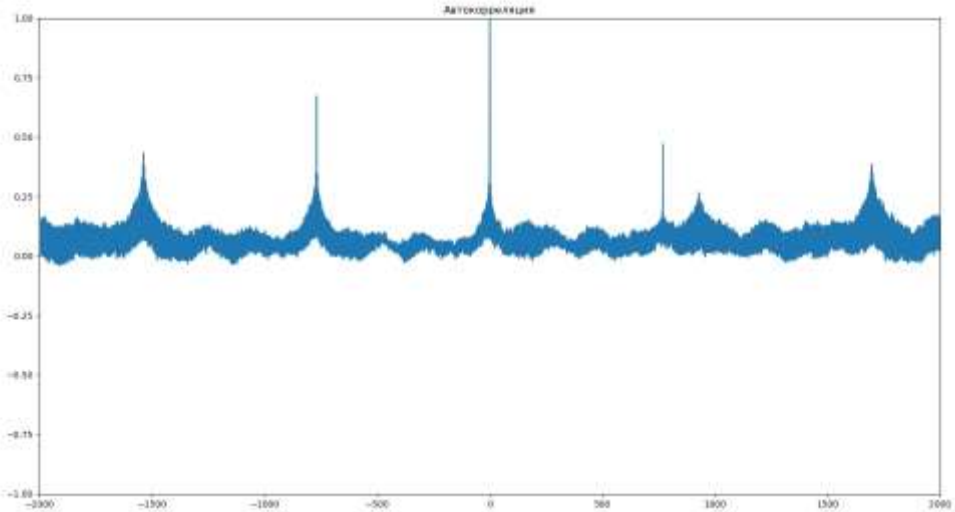

Рис.4 Автокорреляционный тест исходного изображения

Изображение после первого раунда шифрования:

Рис.5 Изображение после одного раунда шифрования

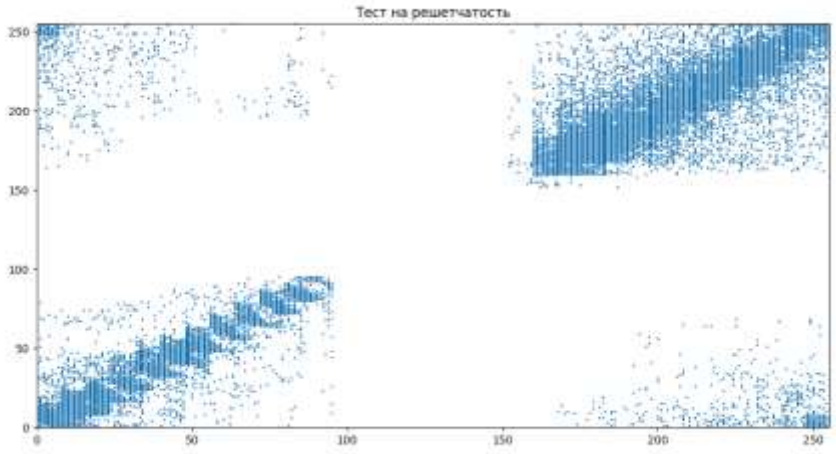

Рис.6 Тест на решетчатость

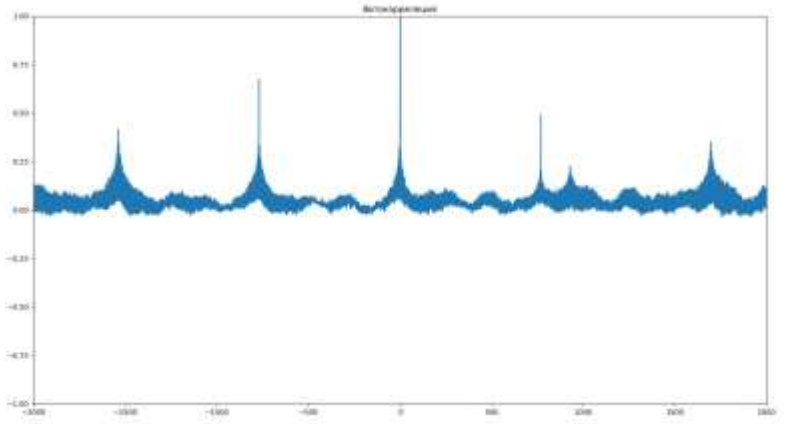

Рис.7 Автокорреляционный тест

Изображение после второго раунда шифрования:

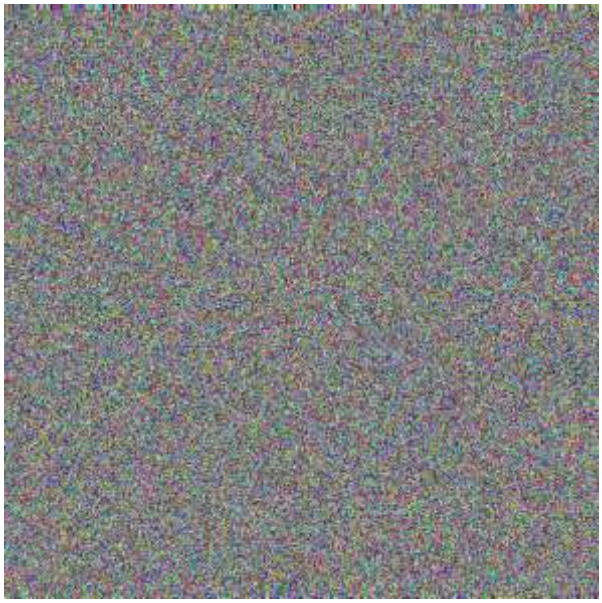

Рис.8 Изображение после двух раундов шифрования

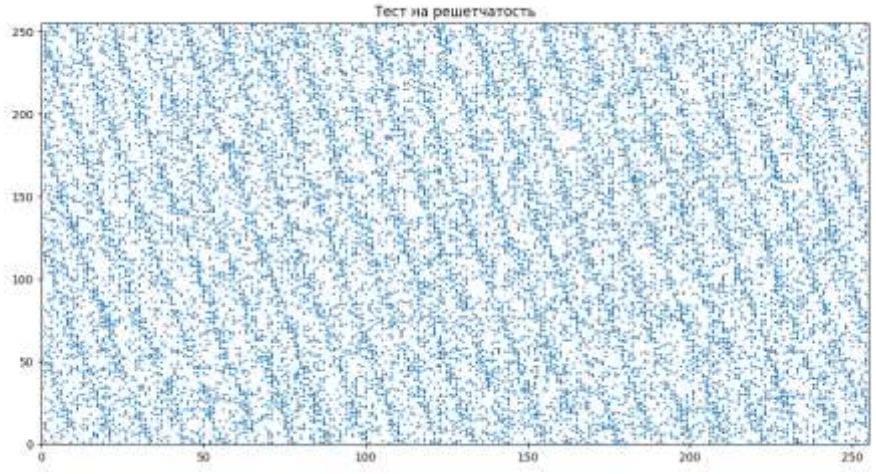

Рис.9 Тест на решетчатость

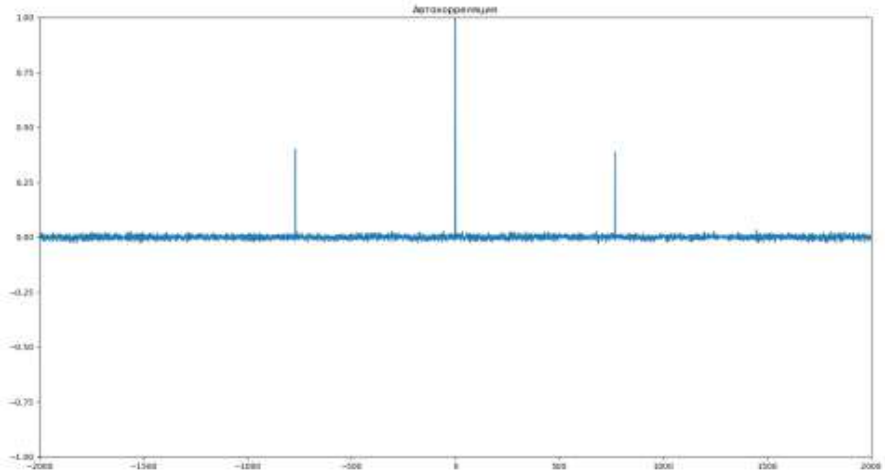

Рис.10 Автокорреляционный тест

Изображение после третьего раунда шифрования:

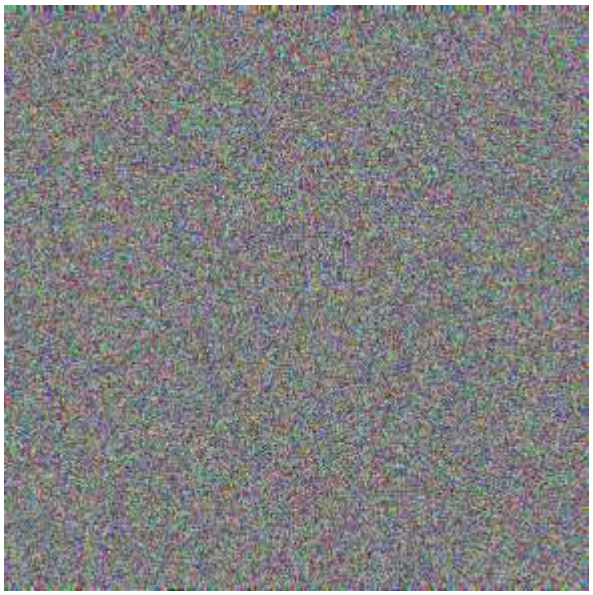

Рис.11 Изображение после трех раундов шифрования

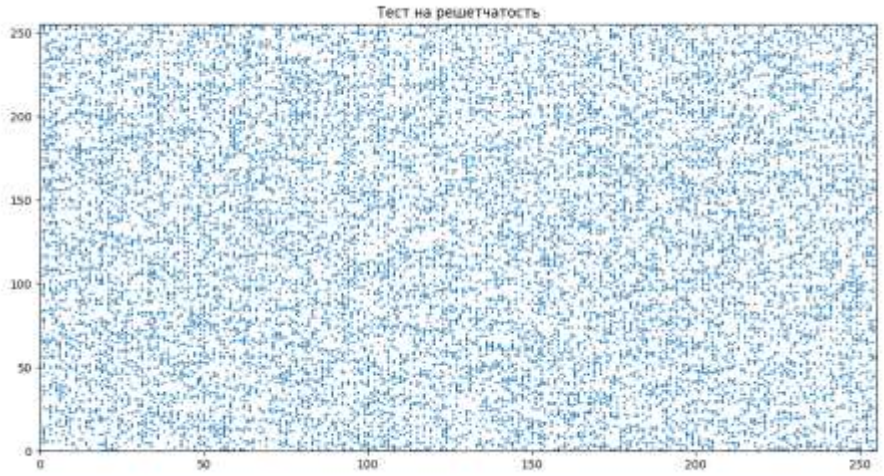

Рис.12 Тест на решетчатость

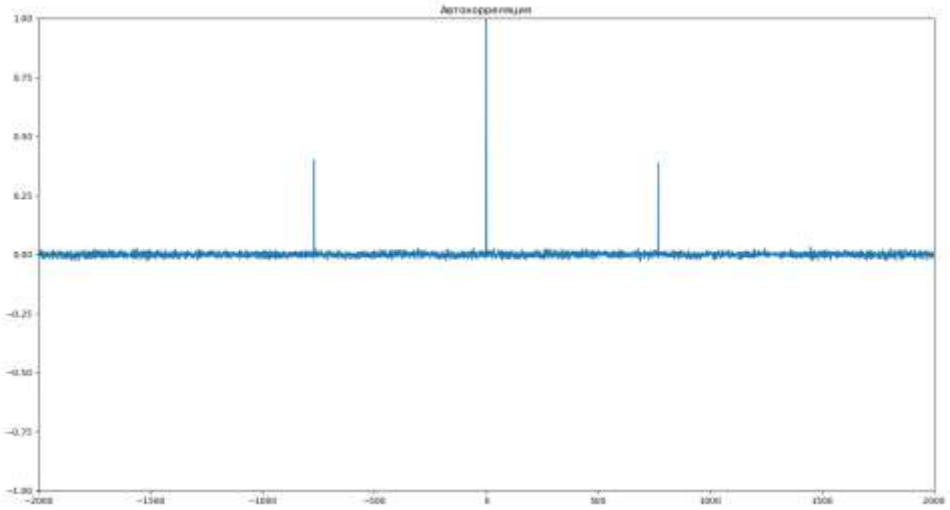

Рис.13 Автокорреляционный тест

Изображение после четвертого раунда шифрования:

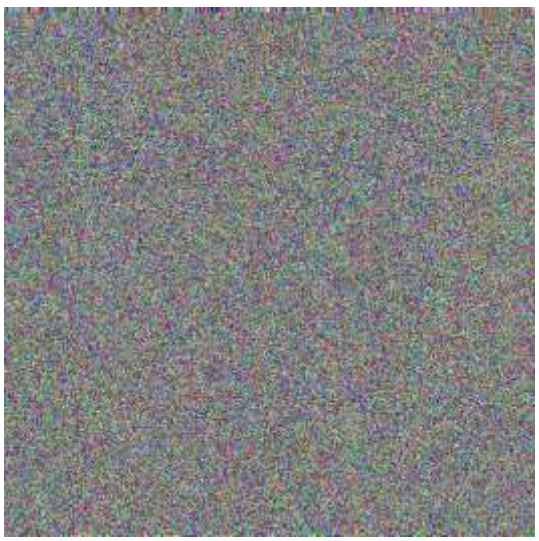

Рис.14 Изображение после четырех раундов шифрования

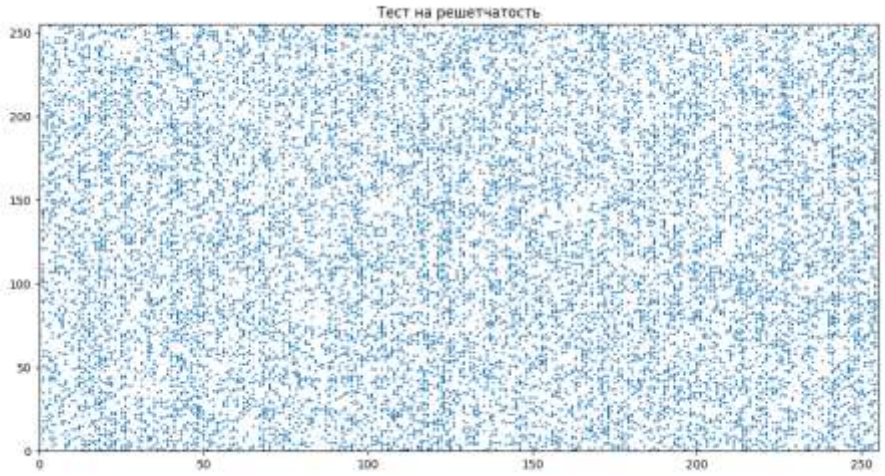

Рис.15 Тест на решетчатость

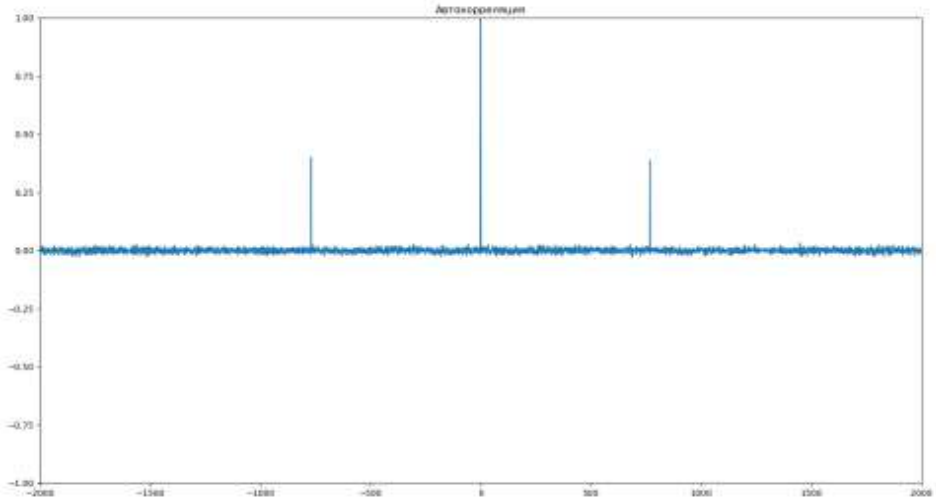

Рис.16 Автокорреляционный тест

Изображение после пятого раунда шифрования:

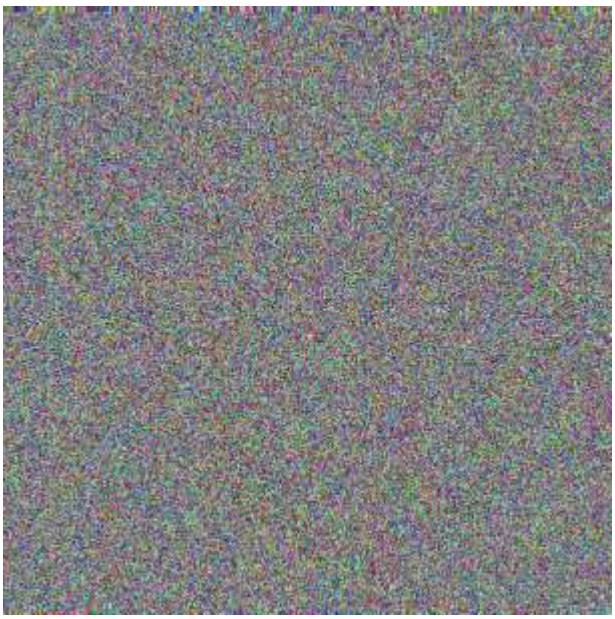

Рис.17 Изображение после пяти раундов шифрования

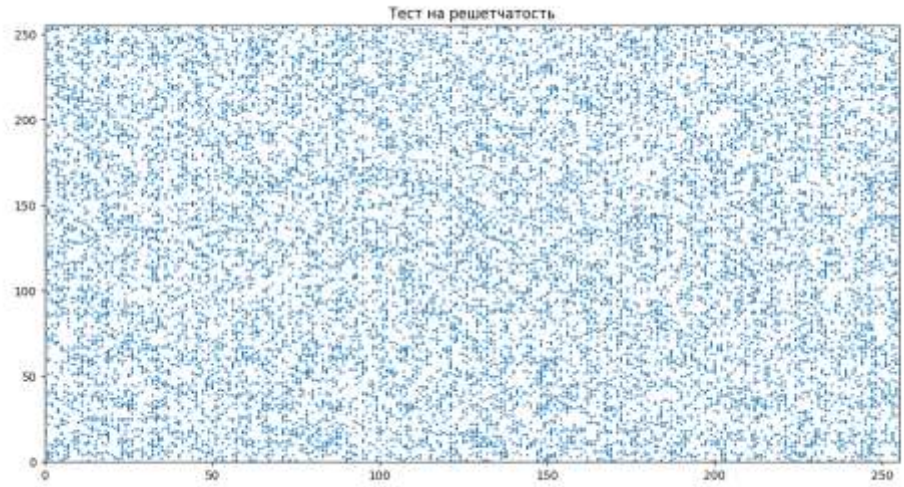

Рис.18 Тест на решетчатость

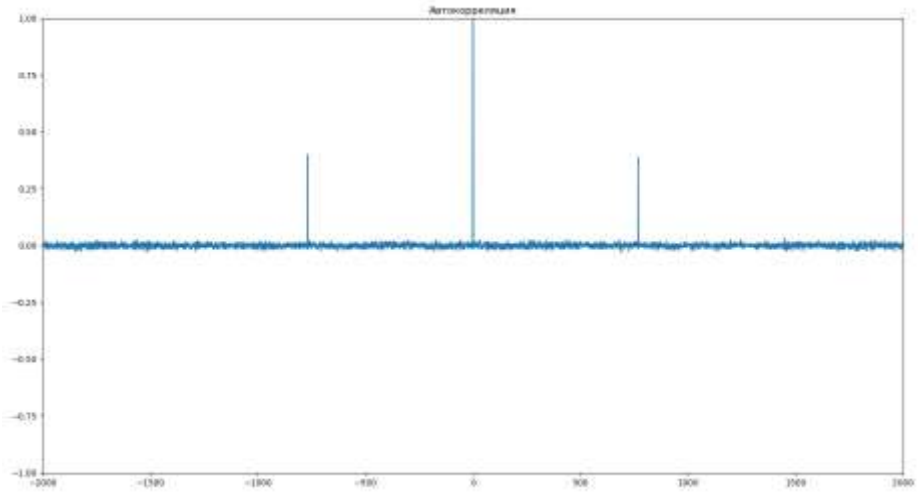

Рис.19 Автокорреляционный тест

Изображение после шестого раунда шифрования:

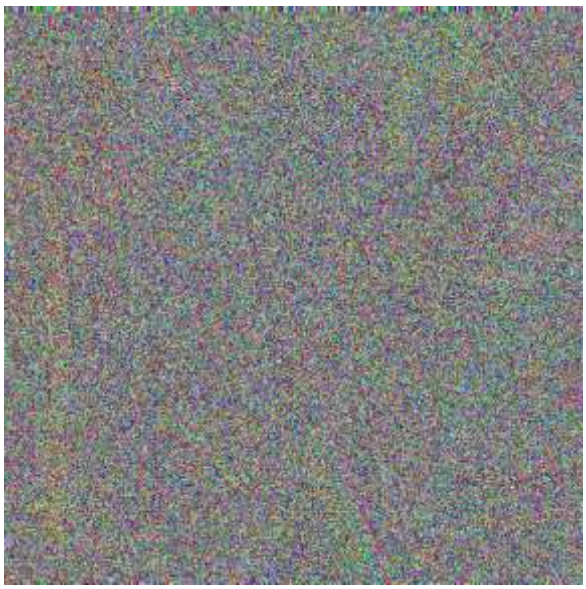

Рис.20 Изображение после шести раундов шифрования

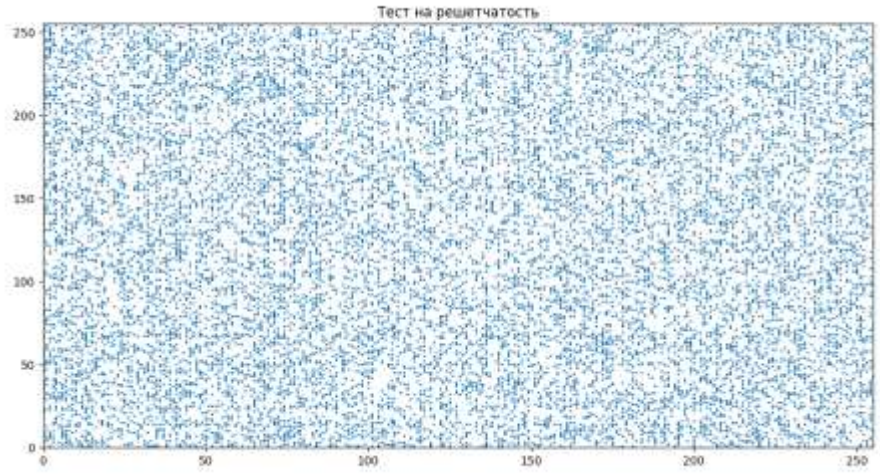

Рис.21 Тест на решетчатость

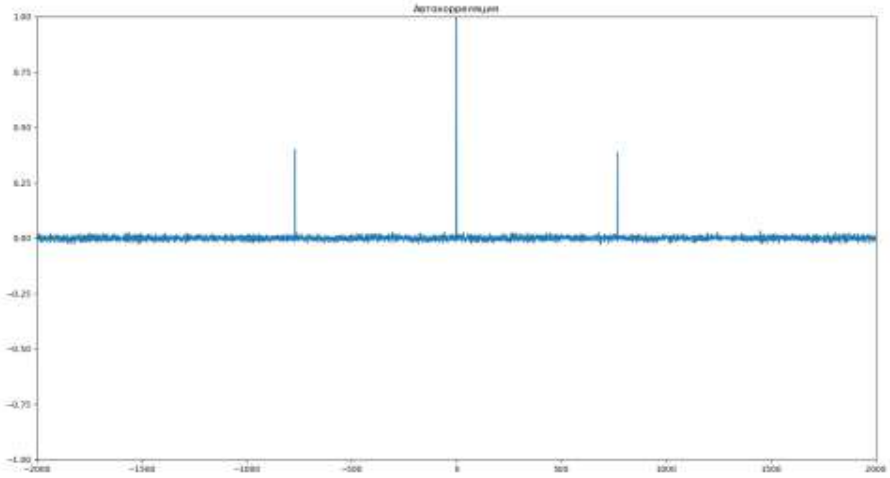

Рис.22 Автокорреляционный тест

Изображение после седьмого раунда шифрования:

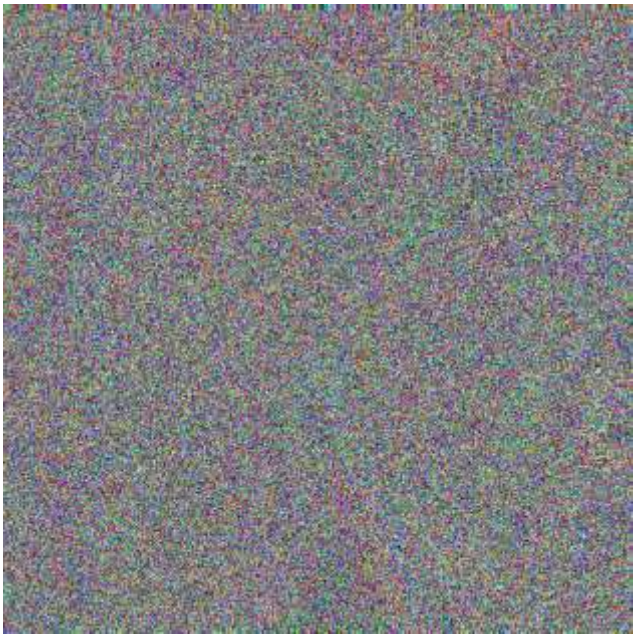

Рис.23 Изображение после семи раундов шифрования

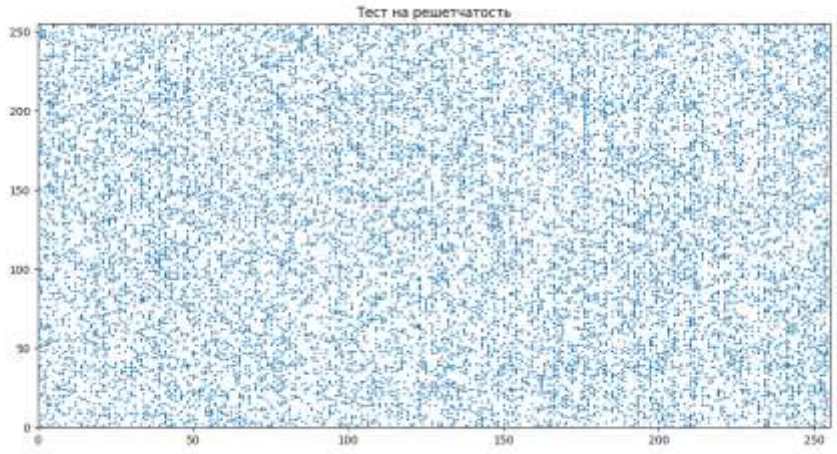

Рис.24 Тест на решетчатость

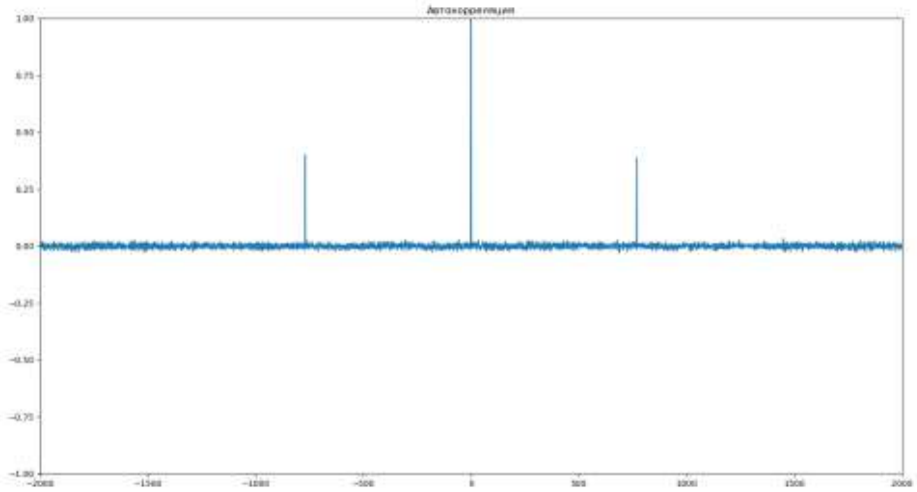

Рис.25 Автокорреляционный тест

Изображение после восьмого раунда шифрования:

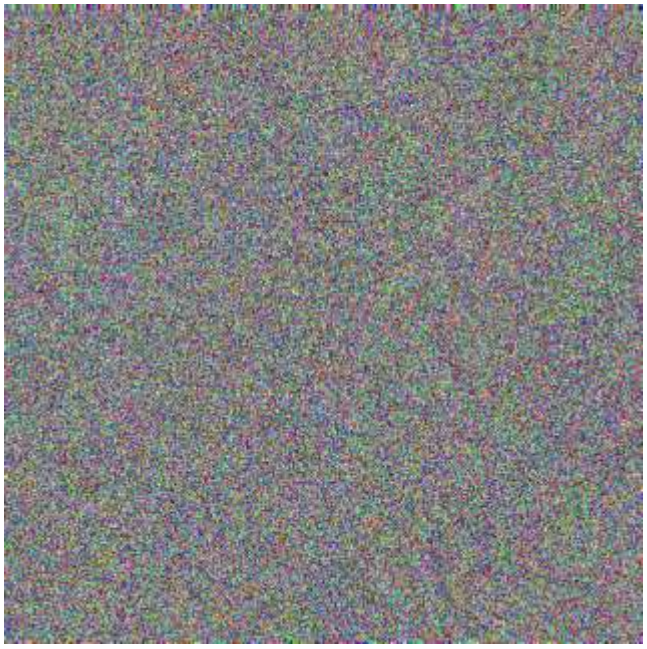

Рис.26 Изображение после восьми раундов шифрования

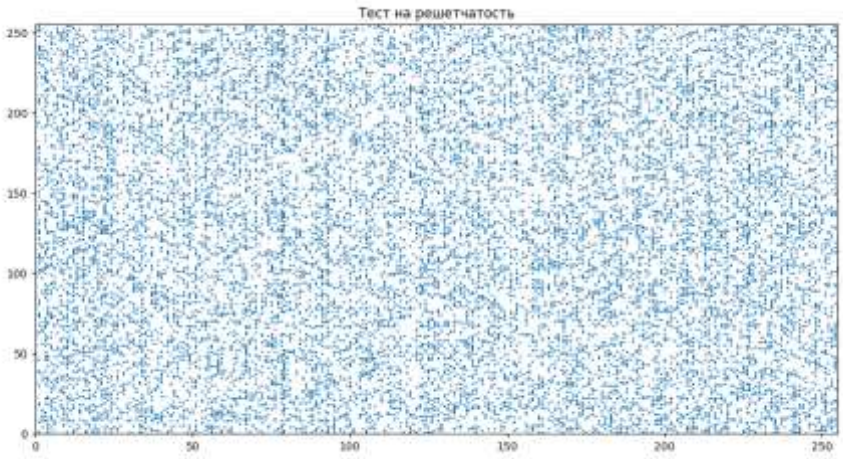

Рис.27 Тест на решетчатость

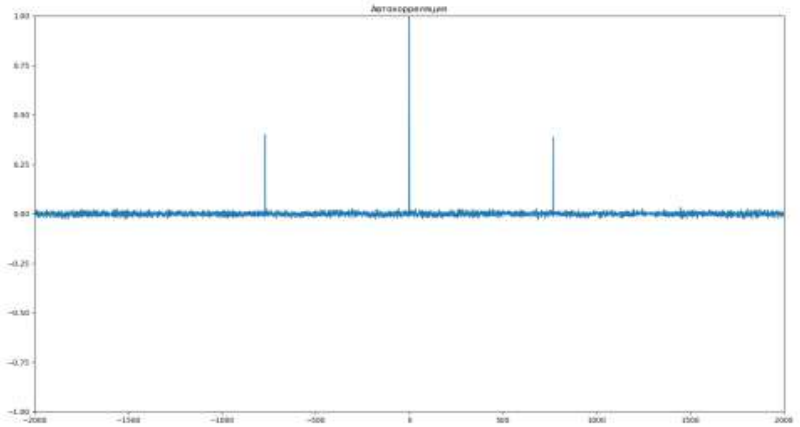

Рис.28 Автокорреляционный тест

Изображение после финального раунда шифрования:

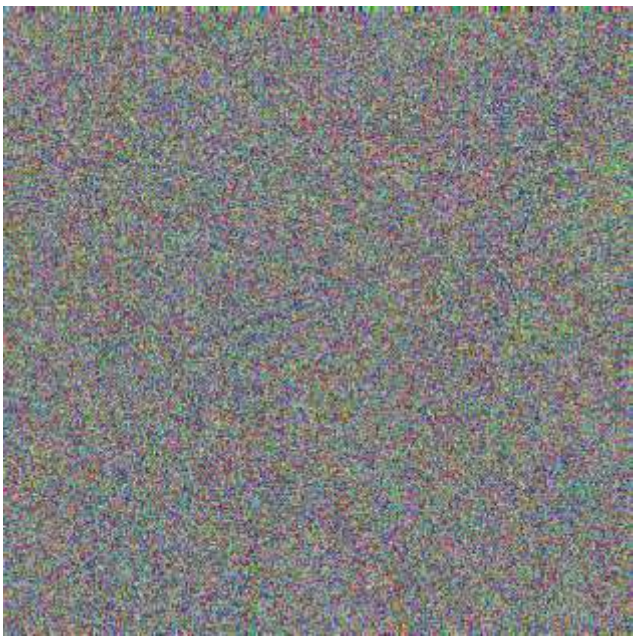

Рис.29 Изображение после шифрования

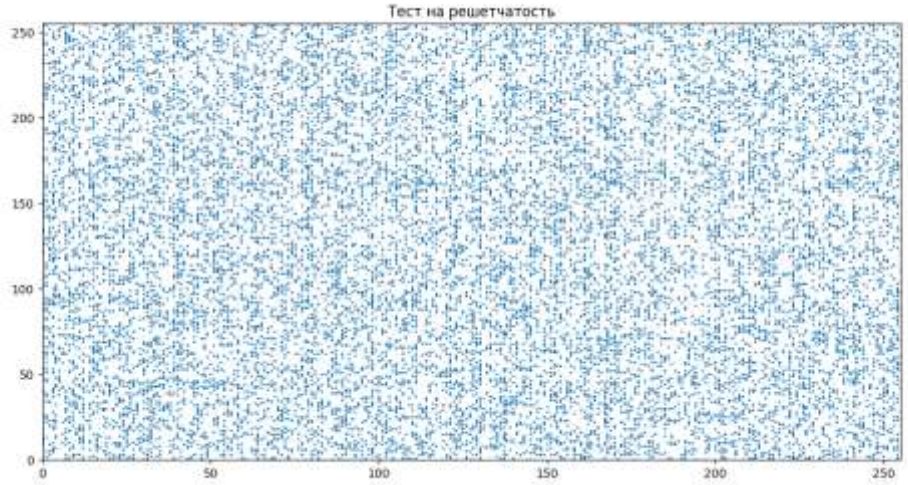

Рис.30 Тест на решетчатость

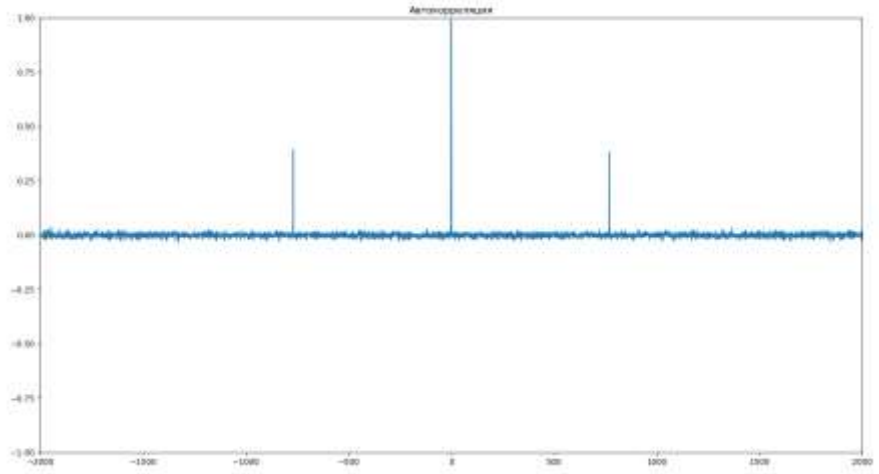

Рис.31 Автокорреляционный тест

По результатам программной реализации алгоритма видно, что после второго раунда невозможно визуально определить даже очертания исходного изображения. Тест на решетчатость показывает, что с каждым раундом значения пикселей распределяются равномерно. Значения на графике автокорреляции стремятся ближе к нулевому.

Проведение атаки методом «грубой силы» бессмысленно из-за большой сложности полного перебора, так как ключ в данном алгоритме имеет размер 128 бит.

Вывод

В ходе выполнения курсовой работы разработан и реализован алгоритм симметричного блокового шифра с операциями умножения по модулю $2^{16} + 1$, сложения по модулю 2(xor) и сложения по модулю 2^8. Проведен анализ результатов путем выполнения автокорреляционного теста и теста на решетчатость. Сделан вывод о невозможности визуального получения информации после шифрования.

Листинг программы, реализованной на языке Java

BlockCipher.java

```
package src;

import java.util.*;
import java.nio.ByteBuffer;
import java.security.SecureRandom;

public class BlockCipher {
	private Vector<byte[]> key;
	private Vector<byte[]> blockOfMessage;
	private static int sizeOfBlocks = 2;
	private static int numRounds = 8;

	public BlockCipher(byte[] message, byte[] key) {
		this.key = splitOfBlocks(key, this.sizeOfBlocks);
		this.blockOfMessage = splitOfBlocks(message, this.sizeOfBlocks);
		// Tools.printBlocks(this.key);
	}

	public BlockCipher(byte[] message) {
		this.key =
splitOfBlocks(generatingKey(this.sizeOfBlocks*this.numRounds),
this.sizeOfBlocks);
		this.blockOfMessage = splitOfBlocks(message, this.sizeOfBlocks);
		// Tools.printBlocks(this.blockOfMessage);
	}
	public Vector<byte[]> getBlocks() {
		return this.blockOfMessage;
	}
	public byte[] getMessage() {
		ByteBuffer res = ByteBuff-
er.allocate(this.blockOfMessage.size()*this.sizeOfBlocks);
		for(byte[] block: this.blockOfMessage) {
			res.put(block);
		}
		return res.array();
	}

	public byte[] getKey() {
		ByteBuffer res = ByteBuff-
er.allocate(this.key.size()*this.sizeOfBlocks);
		for(byte[] block: this.key) {
			res.put(block);
		}
		return res.array();
	}

	private Vector<byte[]> splitOfBlocks(byte[] data, int size) {
		Vector<byte[]> result = new Vector<byte[]>();
		int num = data.length / size;
		// System.out.println("Size data: " + data.length);
		if(data.length % size != 0) {
			int addToSize = (num + 1) * size - data.length;
			ByteBuffer buf = ByteBuffer.allocate(data.length +
addToSize);
			buf.put(data);
			for(int i = 0; i < addToSize; i++) {
```

```
                        byte nu = 0;
                        buf.put(nu);
                }
                data = buf.array();
                num = data.length / size;
        }

        for(int i = 0; i < num; i++) {
                byte[] tmp = new byte[size];
                for(int j = 0; j < size; j++) {
                        tmp[j] = data[i*size + j];
                }
                result.add(tmp);
        }
        return result;
}

public static byte[] generatingKey(int size) {
        SecureRandom random = new SecureRandom();
        byte[] bytes = new byte[size];
        random.nextBytes(bytes);
        return bytes;
}

public byte[] searchForShift() {
        byte min = (byte)255;
        byte max = (byte)0;
        for(byte[] b: this.key) {
                byte tmp1 = b[0];
                byte tmp2 = b[1];

                if(tmp1 < min){
                        min = tmp1;
                }
                if(tmp2 > max) {
                        max = tmp2;
                }
        }
        int res = Operations.concat2Bytes(max, min);
        if(res <= 0) {
                res = -res;
        }
        byte[] resByte = new byte[2];
        resByte[0] = (byte)(res >> 8);
        resByte[1] = (byte)(res);

        return resByte;

}

public void encrypt() {

        for(byte[] block: this.blockOfMessage) {

                for(int i = 0; i < this.numRounds; i++) {
                        if((i % 2) == 0) {
                                roundAddByte(block, this.key.get(i));
                        } else {
                                roundMul(block, this.key.get(i));
```

```
                        }
                    }
                    roundMul(block, searchForShift());
                }
            }

            public void decrypt() {

                for(byte[] block: this.blockOfMessage) {

                    roundMul(block, roundInvMul(searchForShift()));
                    for(int i = (this.numRounds - 1); i > -1; --i) {
                        if((i % 2) == 0) {
                            roundAddInvByte(block, this.key.get(i));
                        } else {
                            roundMul(block,
roundInvMul(this.key.get(i)));
                        }
                    }

                }
            }

            public byte[] roundXor(byte[] block, byte[] key) {
                byte l = block[0];
                byte r = block[1];

                byte tmpl = (byte)(r ^ key[0]);
                byte tmpr = (byte)(l);
                block[0] = tmpl;
                block[1] = tmpr;

                return block;
            }
            public byte[] roundXorInv(byte[] block, byte[] key) {
                byte l = block[0];
                byte r = block[1];

                byte tmpr = (byte)(l ^ key[0]);
                byte tmpl = (byte)(r);
                block[0] = tmpl;
                block[1] = tmpr;

                return block;
            }
            private byte[] roundMul(byte[] block, byte[] key) {
                int x = Operations.concat2Bytes(block[0], block[1]);
                x = Operations.mul(x, Operations.concat2Bytes(key[0], key[1]));
                block[0] = (byte)(x >> 8);
                block[1] = (byte) x;
                return block;
            }

            private byte[] roundAdd(byte[] block, byte[] key) {
                int x = Operations.concat2Bytes(block[0], block[1]);
                x = Operations.add(x, Operations.concat2Bytes(key[0], key[1]));
                block[0] = (byte)(x >> 8);
                block[1] = (byte) x;
                return block;
```

```
        }

        private byte[] roundAddByte(byte[] block, byte[] key) {
                byte l = block[0];
                byte r = block[1];

                byte tmpl = Operations.addByte(r, key[0]);
                byte tmpr = (byte)(l ^ key[1]);

                block[0] = tmpl;
                block[1] = tmpr;

                return block;
        }
        private byte[] roundAddInvByte(byte[] block, byte[] key) {
                byte l = block[0];
                byte r = block[1];

                byte tmpr =  Operations.addByte(l , Opera-
tions.addInvByte(key[0]));
                byte tmpl = (byte)(r ^ key[1]);

                block[0] = tmpl;
                block[1] = tmpr;

                return block;
        }
        private byte[] roundInvMul(byte[] key) {
                byte[] tmp = new byte[this.sizeOfBlocks];
                int tmpInt = Operations.mulInv(key);
                tmp[0] = (byte)(tmpInt >> 8);
                tmp[1] = (byte)(tmpInt);
                return tmp;
        }

        private byte[] roundInvAdd(byte[] key) {
                byte[] tmp = new byte[this.sizeOfBlocks];
                int tmpInt = Operations.addInv(key);
                tmp[0] = (byte)(tmpInt >> 8);
                tmp[1] = (byte)(tmpInt);
                return tmp;
        }
}
```

Main.java

```
package src;

import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.util.*;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.io.UnsupportedEncodingException;
import java.nio.ByteBuffer;


public class Main {
        public static void main(String[] args)        {
```

```
                byte[] key = BlockCipher.generatingKey(16);
                System.out.println(Arrays.toString(key));

                Main.testImage("lena.bmp", key);
                // Main.testString("abcdefgh", key);
                // Main.test(key);
        }
        public static void testImage(String name, byte[] key) {
                Image image = new Image();
                byte[] message = new byte[1];
                try {
                        message = image.readImage(name);
                } catch(IOException e) {
                        System.out.println("Image reading error.");
                }

                BlockCipher enc =  new BlockCipher(message, key);
                enc.encrypt();

                try {
                        image.writeImage("encrypt/encrypt.bmp",
enc.getMessage());
                } catch(IOException e) {
                        System.out.println("Image reading error.");
                }

                BlockCipher dec =  new BlockCipher(enc.getMessage(), key);
                dec.decrypt();
                try {
                        image.writeImage("encrypt/decrypt.bmp",
dec.getMessage());
                } catch(IOException e) {
                        System.out.println("Image reading error.");
                }
//------------- Resheto -------------------------------------------------
                // try {
                //      Main.resheto(message, "test/resheto.txt");
                // } catch(FileNotFoundException fileNot) {
                //      System.out.println("File not found");
                // } catch(UnsupportedEncodingException e) {
                //      System.out.println("UnsupportedEncodingException");
                // }
//------------- AutoCorrelation -----------------------------------------------
-\
                try {
                        BlockCipher orig = new BlockCipher(message, key);
                        Main.autocorrelationTest(enc.getBlocks(),
"test/autoRoundFinal.txt");
                } catch(FileNotFoundException fileNot) {
                        System.out.println("File not found");
                } catch(UnsupportedEncodingException e) {
                        System.out.println("UnsupportedEncodingException");
                }
//-------------BruteForce-----------------------------------------------
                // int resForce = Main.bruteForce(enc.getMessage(),
dec.getMessage());
        }
        public static int bruteForce(byte[] enMessage, byte[] decMessage) {
                int n = 0;
```

```
                // Vector<byte[]> keys = Operations.force(16);
                for(byte i = 0; i < 256; i++) {
                        for(byte j = 0; j < 256; j++) {
                                byte[] ar = {i,j};
                                BlockCipher dec =  new BlockCipher(enMessage,
ar);
                                dec.decrypt();
                                if(Arrays.equals(decMessage, dec.getMessage())) {
                                        return n;
                                }
                                System.out.println(Arrays.toString(ar));

                                System.out.println(n);
                                n++;
                        }
                }
                return 0;
        }
        private static void resheto(byte[] data, String filename) throws File-
NotFoundException, UnsupportedEncodingException {
        PrintWriter writer = new PrintWriter(filename, "UTF-8");
        for(int i = 0; i < data.length/10; i++) {
            int a = (int)data[i];
            int b = (int)data[i+1];
            a &=  0b00000000000000000000000011111111;
            b &=  0b00000000000000000000000011111111;
            writer.println(a + " " + b);
        }
        writer.close();
    }

        private static byte[] getCor(Vector<byte[]> vec, int index) {
                if(index < 0) {
                        index = vec.size() + index;
                }
                return vec.get(index);
        }
        private static void autocorrelationTest(Vector<byte[]> blocks, String
name) throws FileNotFoundException, UnsupportedEncodingException {
        PrintWriter writer = new PrintWriter(name, "UTF-8");
                int size = 4000;
                int matches = 0;
                int noMatches = 0;
                Vector<byte[]> shiftBlock = new Vector<byte[]>();
                for(int i = -(size/2); i < size/2; i++) {
                        shiftBlock.add(Main.getCor(blocks, i));
                }
                for (int i = -(size/2); i < (size/2); i++) {

                        if(i != 0) {
                                byte[] tmp = shiftBlock.get(0);

                                for(int j = 0; j < shiftBlock.size() - 1; j++) {
                                        shiftBlock.set(j, shiftBlock.get(j+1));
                                }

                                shiftBlock.set(shiftBlock.size() - 1, tmp);
                        }
```

```
                        for(int j = 0; j < shiftBlock.size(); j++) {
                                int tmp1 = Opera-
tions.concat2Bytes(shiftBlock.get(j)[0], shiftBlock.get(j)[1]);
                                int tmp2 = Opera-
tions.concat2Bytes(blocks.get(j)[0], blocks.get(j)[1]);

                                // System.out.println(tmp1);
                                // System.out.println(tmp2);

                                String str11 = Integer.toBinaryString(tmp1 |
0b10000000000000000000000000000000);
                                String str22 = Integer.toBinaryString(tmp2 |
0b10000000000000000000000000000000);

                                char[] str1 = str11.toCharArray();
                                char[] str2 = str22.toCharArray();


                                for(int k = 16; k < 32; k++) {
                                        if(str1[k] == str2[k]) {
                                                matches++;
                                        } else {
                                                noMatches++;
                                        }
                                }

                        }
                        double diff = (matches - noMatches) / (16.0 * size);
                        // System.out.println("Посчитал " + i + "/" +
blocks.size() + " = " + diff);
                        writer.println(i + " " + diff);
                        matches = 0;
                        noMatches = 0;
                }
                writer.close();
        }
        public static void testString(String str, byte[] key) {
                System.out.println(str);
                byte[] message = str.getBytes(StandardCharsets.UTF_8);

                BlockCipher enc =  new BlockCipher(message, key);
                enc.encrypt();
                System.out.println(new String(enc.getMessage(), Standard-
Charsets.UTF_8));


                BlockCipher dec =  new BlockCipher(enc.getMessage(), key);
                dec.decrypt();
                System.out.println(new String(dec.getMessage(), Standard-
Charsets.UTF_8));
        }


}
```

Operations.java

```
package src;
```

```
import java.util.*;

public class Operations {

        public static int add(int x, int y) {
                return (x + y) & 0xFFFF;
        }

        public static int addInv(byte[] bx) {
                int x = Operations.concat2Bytes(bx[0], bx[1]);
                return (0x10000 - x) & 0xFFFF;
        }

        public static byte addByte(byte x, byte y) {
                int b = (int)x;
                int s = (int)y;
                int res = (b + s) & 0xFF;
                return (byte)res;
        }

        public static byte addInvByte(byte x) {
                int b = (int)x;
                int res = (0x100 - b) & 0xFF;
                return (byte)res;
        }

        public static int mul(int x, int y) {
                long m = (long) x * y;
                if(m != 0) {
                        return (int)(m % 0x10001) & 0xFFFF;
                } else {
                        if(x != 0 || y != 0) {
                                return (1 - x - y) & 0xFFFF;
                        }
                        return 1;
                }
        }

        public static int mulInv(byte[] bx) {
                int x = Operations.concat2Bytes(bx[0], bx[1]);
         if (x <= 1) {
             // 0 and 1 are their own inverses
             return x;
         }
         try {
             int y = 0x10001;
             int t0 = 1;
             int t1 = 0;
             while (true) {
                 t1 += y / x * t0;
                 y %= x;
                 if (y == 1) {
                     return (1 - t1) & 0xffff;
                 }
                 t0 += x / y * t1;
                 x %= y;
                 if (x == 1) {
                     return t0;
                 }
```

```
            }
        } catch (ArithmeticException e) {
            return 0;
        }
    }

        public static byte[] concat2Bytes(byte[] x, byte[] y) {
                byte[] out  = new byte[x.length + y.length];
                int i = 0;
                for(byte a: x) {
                        out[i++] = a;
                }
                for(byte a: y) {
                        out[i++] = a;
                }
                return out;

        }
        public static int concat2Bytes(int x, int y) {
                x = (x & 0xFF) << 8;
                y = y & 0xFF;
                return (x | y);
        }

        public static Vector<byte[]> force(int size) {
                Vector<byte[]> result = new Vector<byte[]>((int)Math.pow(2,
size));

                for(byte i = 0; i < 256; i++) {
                        for(byte j = 0; j < 256; j++) {
                                byte[] ar = {i,j};
                                result.add(ar);
                        }
                }
                return result;
        }
}
```

Image.java

```
package src;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;

public class Image {

    public byte[] image;
    public int headLength;
    private int width;
    private int height;
    private int bpp;
```

```
    public Image() {
        headLength = 0;
        image = new byte[1];
    }

    public byte[] readImage(String filename) throws IOException {
        File file = new File(filename);
        BufferedImage img = ImageIO.read(file);

        this.width = img.getWidth();
        this.height = img.getHeight();
        this.bpp = getBitsPerPixel(img);

        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        ImageIO.write(img, "bmp", baos);
        baos.flush();
        byte[] imageInByte = baos.toByteArray();
        baos.close();

        this.image = imageInByte;
        this.headLength = imageInByte.length - (width * height * bpp);
        // System.out.println("Image headlength: " + this.headLength);
        byte[] noHeadImage = this.cutHead();
        if(noHeadImage.length%8 != 0) {
            System.out.println("\n[x] Bad image size!!!!!!!!!!\n");
            return null;
        } else {
            return noHeadImage;
        }
    }

    private byte[] cutHead() {
        byte[] imageNoHead = new byte[image.length - headLength];

        for(int i = headLength, j = 0; i < imageNoHead.length; i++, j++) {
            imageNoHead[j] = image[i];
        }
        return imageNoHead;
    }

    public void writeImage(String filename, byte[] imageInByte) throws IOExcep-
tion {
        this.addHead(imageInByte);
        BufferedImage newimg = ImageIO.read(new ByteArrayIn-
putStream(this.image));
        ImageIO.write(newimg, "bmp", new File(filename));
    }

    private void addHead(byte[] img) {
        for(int i = headLength, j = 0; i < img.length; i++, j++) {
            this.image[i] = img[j];
        }
    }

    public int getBitsPerPixel(BufferedImage img) {
        return img.getColorModel().getPixelSize() / Byte.SIZE;
    }
    public void infoOfImage() {
```

```
        System.out.println("Input image size: " + this.width + " x " +
this.height + " (bpp = " + this.bpp + ")");
        System.out.println("Image size = " + this.width * this.height * this.bpp
+ "\n");
    }
}
```

Tools.java

```
package src;

import java.nio.ByteBuffer;
import java.util.*;


public class Tools {
    public static void printBlocks(Vector<byte[]> vec) {
                for(byte[] a: vec) {
                        Tools.printBits(a);
                        System.out.println();
                }
        }
        public static void printBits(byte[] data) {
                for(int i = 0; i < data.length; i++) {
                        System.out.print(Integer.toString((int)data[i] )+ " ");
                }
                System.out.println();
        }
        public static String toBinaryString(byte i) {
            char digits[] = {'0', '1'};
            char[] buf = new char[8];
            int charPos = 8;
            byte radix = (byte)2;
            byte mask = (byte)(radix - 1);
            do {
                        buf[--charPos] = digits[i & mask];
                i >>>= 1;
            } while (i != 0);
            for (int j = 0; j < charPos; j++) {
                buf[j] = '0';
            }
            return new String(buf/*, charPos, (8 - charPos)*/);
        }
        public static String byteArrayToHex(byte[] a) {
        StringBuilder sb = new StringBuilder(a.length * 2);
        for(byte b: a)
                        sb.append(Integer.toHexString(b & 0xFF));
        return sb.toString();
    }

}
```