ГУАП

КАФЕДРА 33

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

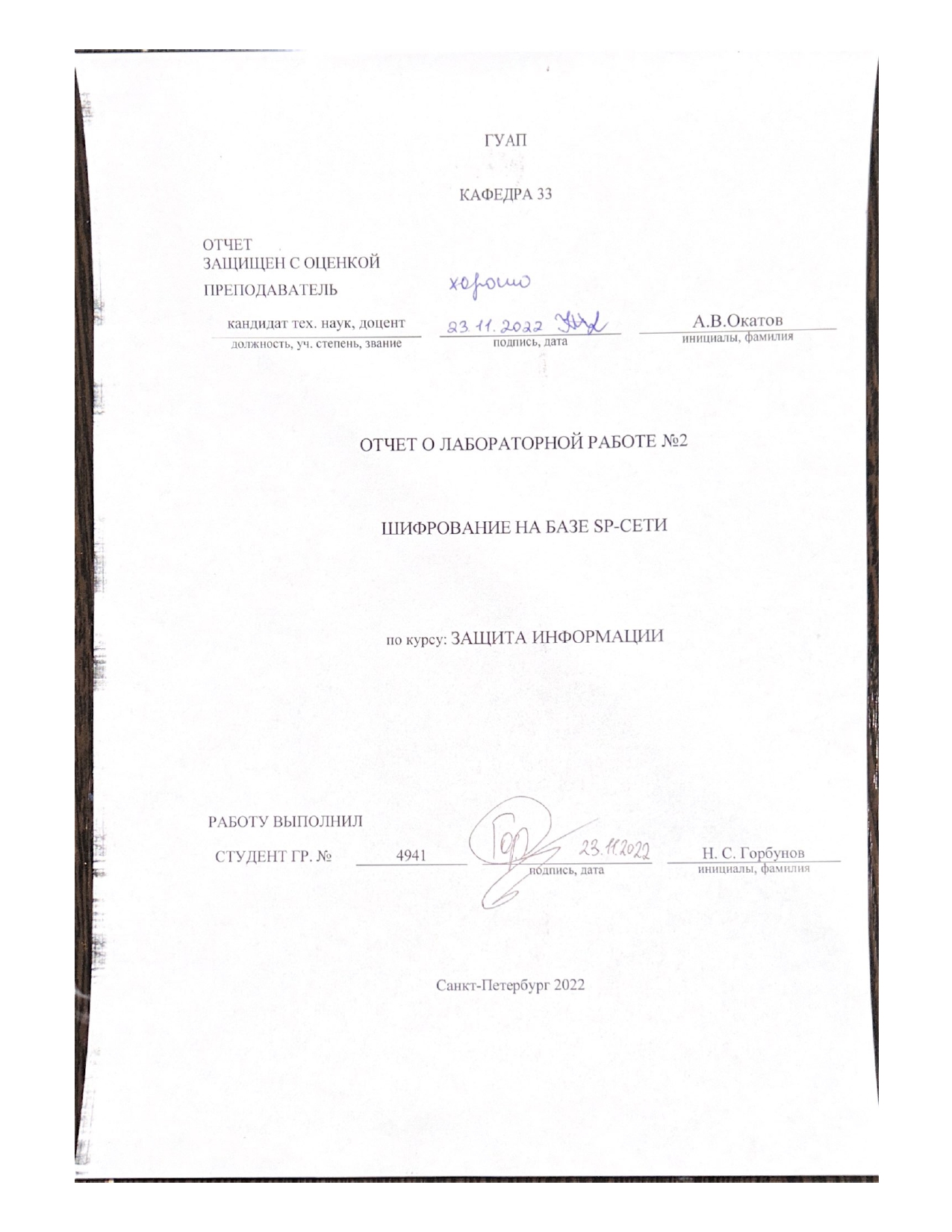
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доцент |  |  |  | А.В.Окатов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2  ШИФРОВАНИЕ НА БАЗЕ SP-СЕТИ |
|  |
| по курсу: ЗАЩИТА ИНФОРМАЦИИ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4941 |  |  |  | Н. С. Горбунов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

****

**Цель работы:** на практикеизучить итерационную шифрующую сеть.

**Ход работы**

Для работы с изображение выбрал формат Bitmap Picture (далее BMP) из-за того, что это по факту битовый массив. Каждый файл BMP содержит заголовок файла, заголовок изображения, растровые данные и карту цветов. Данные изображения начинаются с 54 позиции. Так как по заданию необходимо зашифровать монохромное изображение, то будем работать с интенсивностью, хранящейся в одном байте.

Эмпирическим путем выбрал размер блока данных равный 240 битам, то есть 30 байтам. Такой размер блока данных обеспечивает достаточное количество информации для хорошего качества шифрования изображения.

Размерность S-блока выбрал равной 12. Обоснование выбора:

* Такая размерность блока подходит под выбранный размер блока данных. То есть в одном блоке данных помещается 20 S-блоков.
* Так как в формате изображения который я выбрал , информация о пикселе кодируется 3 байтами, согласно стандарту BT-709, блок S размерностью 12 обеспечит «смешение» данных между байтами цветности пикселя, что обеспечит более надежное искажение изображения в зашифрованном виде.

Принцип работы моего S-блока заключается в выполнении над исходными битами операции логического НЕ (инверсии) и инвертирования порядка относительно центрального элемента. То есть 12 элемент на входе подается на первый выхода, 11 на входе на второй выхода и так далее.

Пример таблицы подстановки для 2х входных слов

|  |  |
| --- | --- |
| Входное слово | Выходное слово |
| 100101101010 | 101010010110 |
| 011011110001 | 011100001001 |
| 111100001100 | 110011110000 |

Частичная схема первого раунда SP-сети

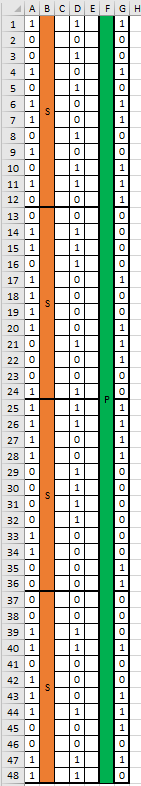
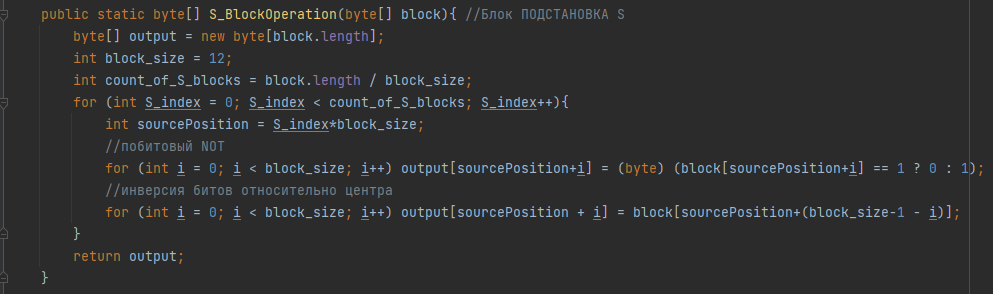


Рисунок 1. Пример первого раунда части SP-сети

Листинг S-блока на языке Java



Листинг P-блока на языке Java



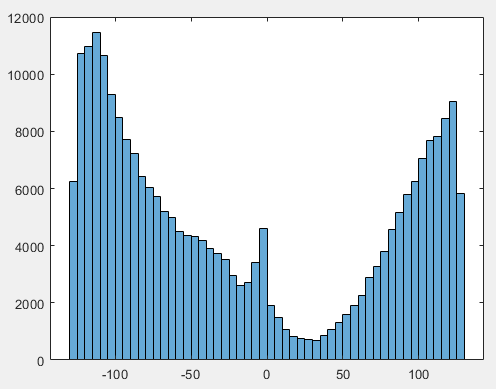
**Тестирование алгоритма**

На рисунке 2 представлено исходное изображение

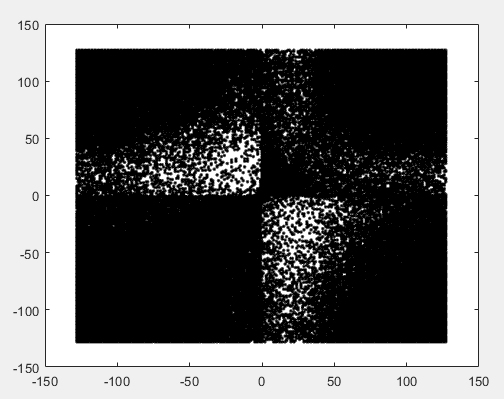


Рисунок 2. Исходное изображение

Гистограмма исходного изображения приведена на рисунке 3. Тест на решетчатость приведен на рисунке 4.



*Рисунок 3.*



*Рисунок 4.*

Исходя из визуального анализа графических тестов рисунков 3 и 4 можно сделать вывод, о том, что в приведенном массиве данных существуют закономерности. На графике теста на решетчатость видно неравномерность распределения значений пикселей.

**Шифрование изображения**

***После первого раунда шифрования изображение принимает вид:***

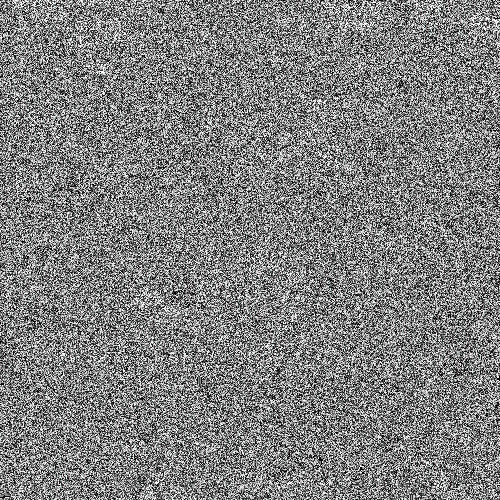
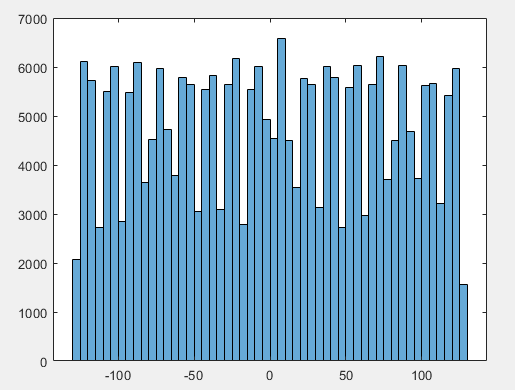
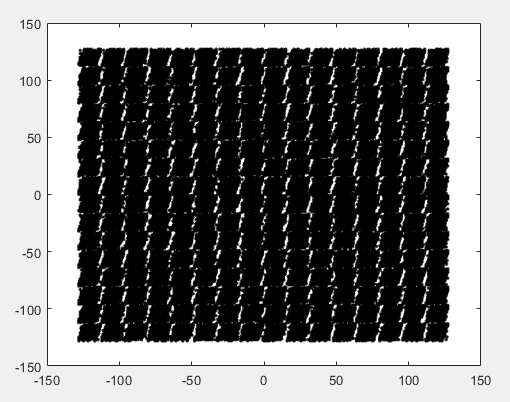


Рисунок 5. Первый раунд шифрования

Графические тесты:





Гистограмма и тест на решетчатость существенных изменений не приобрели после первого раунда шифрования. Хотя само зашифрованное изображение не имеет видимых на первый взгляд закономерностей.

***После второго раунда шифрования изображение принимает вид:***

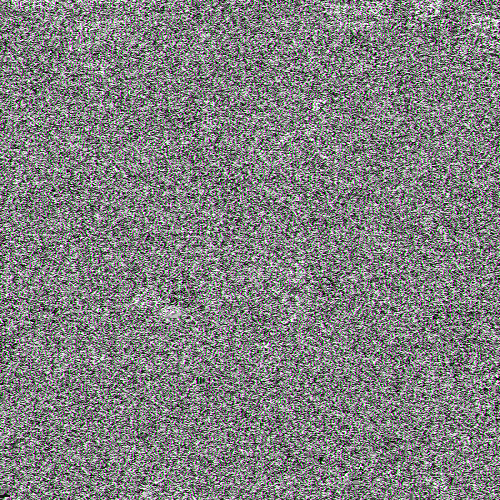
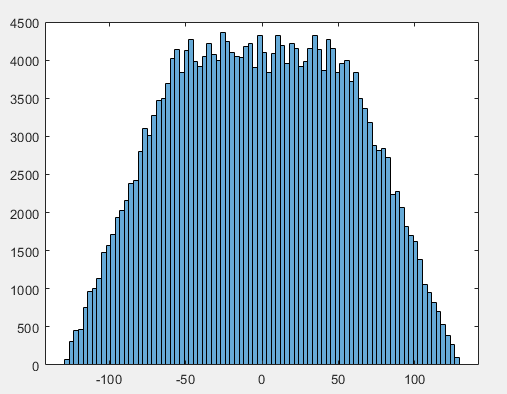
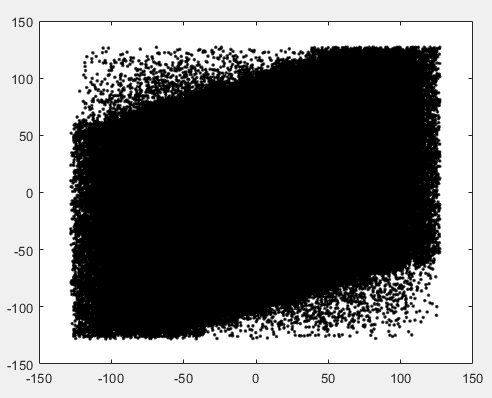


Рисунок 6. Второй раунд шифрования

Графические тесты:





Гистограмма приобрела очертания нормального распределения значений интенсивности. Но на тесте на решетчатость все еще видны полосы, которые говорят о существовании зависимости.

***После пятого раунда шифрования изображение принимает вид:***

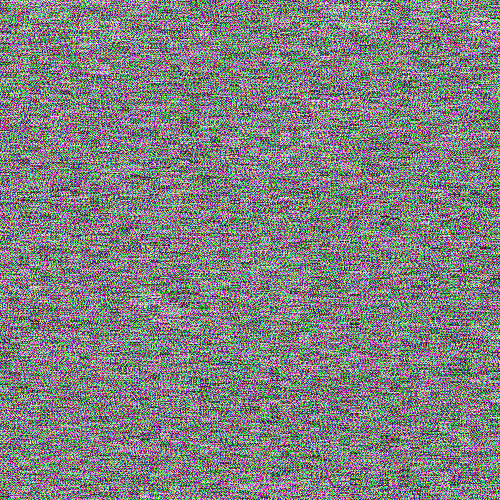
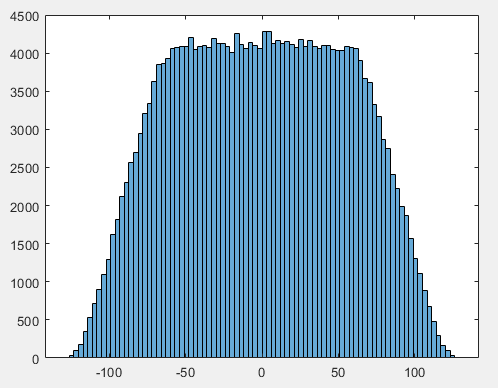
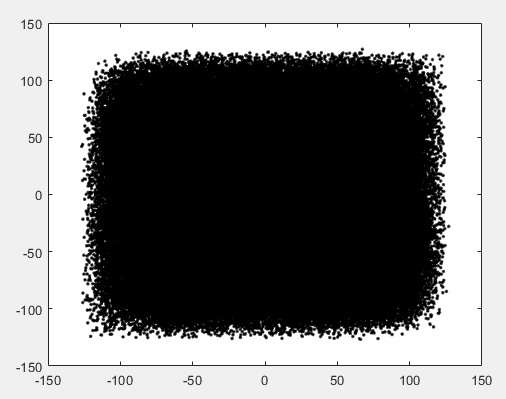


Рисунок 7. Пятый раунд шифрования.

Графические тесты:





После 5 раундов шифрования гистограмма указывает на низкий уровень зависимостей между элементами. А тест на решетчатость показывает случайное расположение элементов без видимых зависимостей. Визуальный тест также показывает абсолютное отсутствие контуров на изображении и его шумообразный вид. Элементы нормально распределены на плоскости.

Вывод: 5 раундов шифрования достаточно. При дальнейшем увеличении количества раундов наблюдается либо ухудшение качества шифрования, либо незначительный прирост наблюдаемых показателей. При тестах на различных типах изображения была обнаружена интересная особенность разработанной SP-сети: при больших однотонных участках на изначальном изображении, на зашифрованном, даже при большом количестве раундов и иных размерах S и P блоков прослеживаются силуэты, а тест на решетчатость показывает наличие зависимостей

**Дешифрование изображения:**

Для начала дешифруем при помощи P блока, а затем S-блоков. При этом должно быть так же 5 раундов дешифрования, как во время шифрования. При этом сдвиг в блоке Р при дешифровке должен уменьшаться с 5 до 1 в конечную итерацию. Таблица подстановки блока S взаимно-обратная, поэтому ее менять при дешифровании не стоит. Результат расшифровки приведет на рисунке 8.

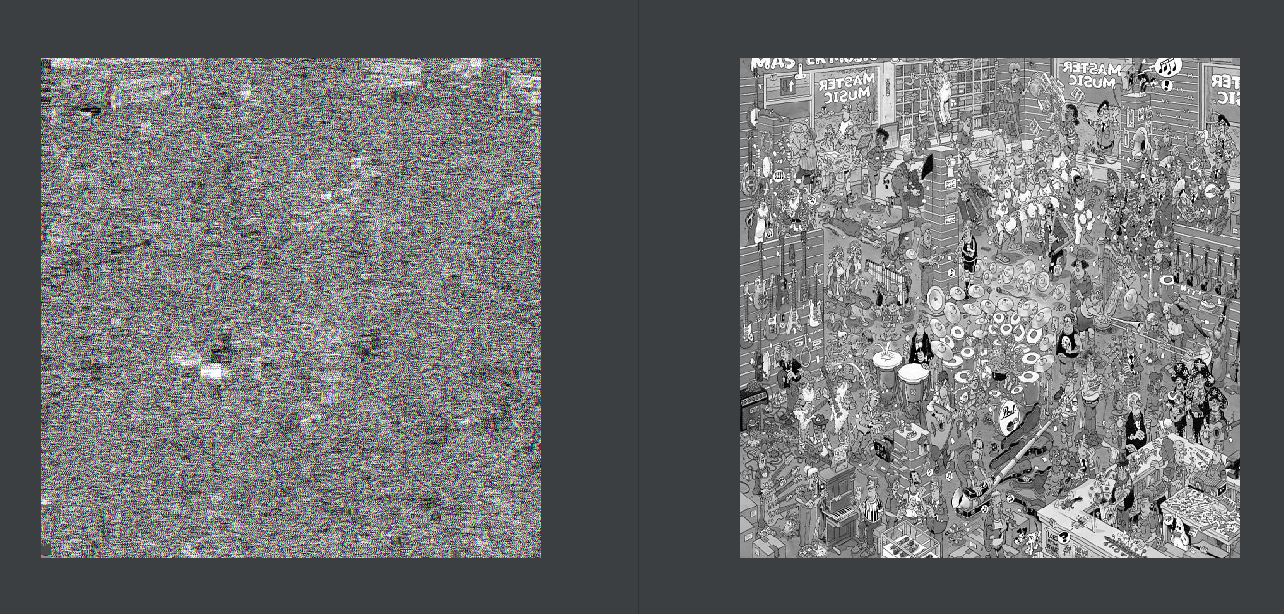


Рисунок 8. Результат дешифрования.

**Вывод:** Использование SP-сетей для кодирования информации позволяет достаточно быстро получить закодированный результат с равномерным распределением значений, визуально похожий на шум.

Для того, чтобы точно определить уровень достаточности шифрования необходимо использовать несколько разных видов тестов. Например, визуально оценить шумоподобность изображения, провести тест на решетчатость и построить и гистограмму распределения значений ряда яркостей.

Для получения хорошего результата шифрования изображения следует использовать несколько раундов шифрования.

Чтобы добиться хорошего шифрования стоит использовать алгоритмы псевдослучайной замены в S-блоках и изменяемый параметр перестановки в P блоках.

Приложение 1. Шифратор

import java.io.BufferedOutputStream;  
import java.io.File;  
import java.io.FileOutputStream;  
import java.io.IOException;  
import java.nio.file.Files;  
  
public class ImageEncryptor {  
 //BGR - порядок кодирования битов  
 //формула вычисления интенсивности пикселя по стандарту BT-709 Y = 0.2125·R + 0.7154·G + 0.0721·B  
 public static void main (String[] Args) throws IOException {  
 //EncryptImageBySPBlock("src/main/resources/picture.bmp", "src/main/resources/output.bmp");  
 *EncryptImageRoundByRound*("src/main/resources/picture.bmp","src/main/resources/output/", "output",30 );  
 }  
  
 public static void EncryptImageRoundByRound(String source, String destination, String outputFileName, int rounds)throws IOException{  
  
 byte[] bitmap = *ReadBytes*(source); //чтение  
 for(int i=0; i < rounds; i++)  
 {  
 bitmap = *encrypted\_bitmap\_by\_rounds*(bitmap, i); //шифрование  
 String dest = destination + outputFileName + Integer.*toString*(i) + ".bmp";  
 *WriteBytesToFile*(dest, bitmap); //запись  
 }  
 }  
  
 public static void EncryptImageBySPBlock (byte[] bitmap, String destination) throws IOException{  
 bitmap = *encrypted\_bitmap*(bitmap); //шифрование  
 *WriteBytesToFile*(destination, bitmap); //запись  
 }  
  
 public static byte[] encrypted\_bitmap\_by\_rounds (byte[] bitmap, int rounds){  
 int pixels\_count = bitmap.length;  
 byte[] output\_bitmap = new byte[pixels\_count];  
 //заполняем шапку файла  
 System.*arraycopy*(bitmap, 0, output\_bitmap, 0, 54);  
 //примем размерность нашего блока данных за 48 битов,  
 //то есть это 6 байтов, кодирующие 2 последовательных пикселя  
 int data\_block = 30; //ДЛИНА БЛОКА ДАННЫХ  
 for (int index = 54; index < pixels\_count; index += data\_block){  
 //здесь на каждой итерации обрабатываем 30 последовательных байтов в значимой массе байтов  
 //и заполняем их биты в общий битовый блок bits на 240 разрядов  
 byte[] bits = new byte[data\_block\*8]; //блок данных  
 for (int byte\_index = 0; byte\_index < data\_block; byte\_index++){ //читаю по порядку байты одного блока  
 for (int bit = byte\_index\*8+7; bit >= byte\_index\*8; bit--){ //читаю биты одного байта из блока  
 if ((bitmap[index+byte\_index]%2==1)||(bitmap[index+byte\_index]%2==-1)) bits[bit] = 1;  
 bitmap[index+byte\_index] = (byte) (bitmap[index+byte\_index] >> 1);  
 }  
 }  
 //обработка блока данных S и P блоками  
 bits = *BlockEncryptor*(bits, rounds);  
  
 //далее происходит запись в выходную карту байтов  
 for (int i = index; i < index+data\_block; i++){  
 int pre\_byte = 0;  
 for (int local\_bit = (i-index)\*8; local\_bit < (i-index)\*8 + 8; local\_bit++){  
 pre\_byte += bits[local\_bit]\*Math.*pow*(2,(7 - (local\_bit-((i-index)\*8))));  
 }  
 output\_bitmap[i] = (byte) pre\_byte;  
 }  
 }  
 return output\_bitmap;  
  
 }  
  
 public static byte[] BlockEncryptor(byte[] block, int rounds){  
 byte[] output = block;  
 for (int index = 0; index < rounds; index++){  
 output = *S\_BlockOperation*(output);  
 output = *P\_BlockOperation*(output, index+4);  
 }  
 return output;  
 }  
 public static void EncryptImageBySPBlock (String source, String destination) throws IOException{  
 byte[] bitmap = *ReadBytes*(source); //чтение  
 bitmap = *encrypted\_bitmap*(bitmap); //шифрование  
 *WriteBytesToFile*(destination, bitmap); //запись  
 }  
 public static void WriteBytesToFile (String source, byte[] bytes) throws IOException {  
 File output\_file = new File(source);  
 BufferedOutputStream output\_stream = new BufferedOutputStream(new FileOutputStream(output\_file));  
 output\_stream.write(bytes);  
 output\_stream.flush();  
 output\_stream.close();  
 }  
 public static byte[] ReadBytes (String source) throws IOException {  
 File input\_file = new File(source);  
 return Files.*readAllBytes*(input\_file.toPath());  
 }  
 public static byte[] encrypted\_bitmap (byte[] bitmap){  
 int pixels\_count = bitmap.length;  
 byte[] output\_bitmap = new byte[pixels\_count];  
 //заполняем шапку файла  
 System.*arraycopy*(bitmap, 0, output\_bitmap, 0, 54);  
 //примем размерность нашего блока данных за 48 битов,  
 //то есть это 6 байтов, кодирующие 2 последовательных пикселя  
 int data\_block = 24; //ДЛИНА БЛОКА ДАННЫХ  
 for (int index = 54; index < pixels\_count; index += data\_block){  
 //здесь на каждой итерации обрабатываем 30 последовательных байтов в значимой массе байтов  
 //и заполняем их биты в общий битовый блок bits на 240 разрядов  
 byte[] bits = new byte[data\_block\*8]; //блок данных  
 for (int byte\_index = 0; byte\_index < data\_block; byte\_index++){ //читаю по порядку байты одного блока  
 for (int bit = byte\_index\*8+7; bit >= byte\_index\*8; bit--){ //читаю биты одного байта из блока  
 if ((bitmap[index+byte\_index]%2==1)||(bitmap[index+byte\_index]%2==-1)) bits[bit] = 1;  
 bitmap[index+byte\_index] = (byte) (bitmap[index+byte\_index] >> 1);  
 }  
 }  
 //обработка блока данных S и P блоками  
 bits = *BlockEncryptor*(bits);  
  
 //далее происходит запись в выходную карту байтов  
 for (int i = index; i < index+data\_block; i++){  
 int pre\_byte = 0;  
 for (int local\_bit = (i-index)\*8; local\_bit < (i-index)\*8 + 8; local\_bit++){  
 pre\_byte += bits[local\_bit]\*Math.*pow*(2,(7 - (local\_bit-((i-index)\*8))));  
 }  
 output\_bitmap[i] = (byte) pre\_byte;  
 }  
 }  
 return output\_bitmap;  
  
 }  
 public static byte[] BlockEncryptor(byte[] block){  
 byte[] output = block;  
 int rounds =5;  
 for (int index = 0; index < rounds; index++){  
 output = *S\_BlockOperation*(output);  
 output = *P\_BlockOperation*(output, index+2);  
 }  
 return output;  
 }  
 public static byte[] S\_BlockOperation(byte[] block){ //Блок ПОДСТАНОВКА S  
 byte[] output = new byte[block.length];  
 int block\_size = 12;  
 int count\_of\_S\_blocks = block.length / block\_size;  
 for (int S\_index = 0; S\_index < count\_of\_S\_blocks; S\_index++){  
 int sourcePosition = S\_index\*block\_size;  
 //побитовый NOT  
 for (int i = 0; i < block\_size; i++) output[sourcePosition+i] = (byte) (block[sourcePosition+i] == 1 ? 0 : 1);  
 //инверсия битов относительно центра  
 for (int i = 0; i < block\_size; i++) output[sourcePosition + i] = block[sourcePosition+(block\_size-1 - i)];  
 }  
 return output;  
 }  
 public static byte[] P\_BlockOperation(byte[] block, int shift) { //Блок ПЕРЕСТАНОВКА P  
 if (block != null) {  
 int length = block.length;  
 byte[] out = new byte[length];  
 System.*arraycopy*(block, shift, out, 0, length - shift);  
 System.*arraycopy*(block, 0, out, length - shift, shift);  
 return out;  
 } else {  
 return null;  
 }  
 }  
}

Приложение 2. Дешифратор

import java.io.BufferedOutputStream;  
import java.io.File;  
import java.io.FileOutputStream;  
import java.io.IOException;  
import java.nio.file.Files;  
  
public class ImageDecrypt {  
 public static void main (String[] args) throws IOException {  
 *DecryptImageBySPBlock*("src/main/resources/output/output6.bmp", "src/main/resources/decrypted\_out.bmp");  
 }  
 public static void DecryptImageBySPBlock (String source, String destination) throws IOException {  
 byte[] bitmap = *ReadBytes*(source); //чтение  
 bitmap = *decrypted\_bitmap*(bitmap); //шифрование  
 *WriteBytesToFile*(destination, bitmap); //запись  
 }  
 public static void WriteBytesToFile (String source, byte[] bytes) throws IOException {  
 File output\_file = new File(source);  
 BufferedOutputStream output\_stream = new BufferedOutputStream(new FileOutputStream(output\_file));  
 output\_stream.write(bytes);  
 output\_stream.flush();  
 output\_stream.close();  
 }  
 public static byte[] ReadBytes (String source) throws IOException {  
 File input\_file = new File(source);  
 return Files.*readAllBytes*(input\_file.toPath());  
 }  
 public static byte[] decrypted\_bitmap (byte[] bitmap){  
 int pixels\_count = bitmap.length;  
 byte[] output\_bitmap = new byte[pixels\_count];  
 //заполняем шапку файла  
 System.*arraycopy*(bitmap, 0, output\_bitmap, 0, 54);  
 //примем размерность нашего блока данных за 48 битов,  
 //то есть это 6 байтов, кодирующие 2 последовательных пикселя  
 int data\_block = 6;  
 for (int index = 54; index < pixels\_count; index += data\_block){  
 //здесь на каждой итерации обрабатываем 6 последовательных байтов в значимой массе байтов  
 //и заполняем их биты в общий битовый блок bits на 48 разрядов  
 byte[] bits = new byte[data\_block\*8]; //блок данных  
 for (int byte\_index = 0; byte\_index < data\_block; byte\_index++){ //читаю по порядку байты одного блока  
 for (int bit = byte\_index\*8+7; bit >= byte\_index\*8; bit--){ //читаю биты одного байта из блока  
 if ((bitmap[index+byte\_index]%2==1)||(bitmap[index+byte\_index]%2==-1)) bits[bit] = 1;  
 bitmap[index+byte\_index] = (byte) (bitmap[index+byte\_index] >> 1);  
 }  
 }  
 //обработка блока данных S и P блоками  
 bits = *BlockDecrypt*(bits);  
 //далее происходит запись в выходную карту байтов  
 for (int i = index; i < index+data\_block; i++){  
 int pre\_byte = 0;  
 for (int local\_bit = (i-index)\*8; local\_bit < (i-index)\*8 + 8; local\_bit++){  
 pre\_byte += bits[local\_bit]\*Math.*pow*(2,(7 - (local\_bit-((i-index)\*8))));  
 }  
 output\_bitmap[i] = (byte) pre\_byte;  
 }  
 }  
 return output\_bitmap;  
  
 }  
 public static byte[] BlockDecrypt(byte[] block){  
 byte[] output = block;  
 int rounds = 5;  
 for (int index = rounds; index > 0; index--){  
 output = *P\_BlockOperation*(output, 44-index);  
 output = *S\_BlockOperation*(output);  
 }  
 return output;  
 }  
 public static byte[] S\_BlockOperation(byte[] block){ //Блок ПОДСТАНОВКА S  
 byte[] output = new byte[block.length];  
 int block\_size = 12;  
 int count\_of\_S\_blocks = block.length / block\_size;  
 for (int S\_index = 0; S\_index < count\_of\_S\_blocks; S\_index++){  
 int sourcePosition = S\_index\*block\_size;  
 for (int i = 0; i < block\_size; i++) output[sourcePosition+i] = (byte) (block[sourcePosition+i] == 1 ? 0 : 1);  
 output[sourcePosition] = block[sourcePosition+11];  
 output[sourcePosition+1] = block[sourcePosition+10];  
 output[sourcePosition+2] = block[sourcePosition+9];  
 output[sourcePosition+3] = block[sourcePosition+8];  
 output[sourcePosition+4] = block[sourcePosition+7];  
 output[sourcePosition+5] = block[sourcePosition+6];  
 output[sourcePosition+6] = block[sourcePosition+5];  
 output[sourcePosition+7] = block[sourcePosition+4];  
 output[sourcePosition+8] = block[sourcePosition+3];  
 output[sourcePosition+9] = block[sourcePosition+2];  
 output[sourcePosition+10] = block[sourcePosition+1];  
 output[sourcePosition+11] = block[sourcePosition];  
 }  
 return output;  
 }  
 public static byte[] P\_BlockOperation(byte[] block, int shift) { //Блок ПЕРЕСТАНОВКА P  
 if (block != null) {  
 int length = block.length;  
 byte[] out = new byte[length];  
 System.*arraycopy*(block, shift, out, 0, length - shift);  
 System.*arraycopy*(block, 0, out, length - shift, shift);  
 return out;  
 } else {  
 return null;  
 }  
 }  
}

Приложение 3. Подготовка данных для математического моделирования

import java.io.File;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.nio.file.Files;  
  
public class MathTesting {  
 public static void main (String[] args) throws IOException {  
 byte[] bitmap = *ReadBytes*("src/main/resources/output/output6.bmp");  
 byte[] targeted\_bitmap = new byte[bitmap.length-54];  
 System.*arraycopy*(bitmap, 54, targeted\_bitmap, 0, bitmap.length-54);  
 float[] ready\_vector = new float[targeted\_bitmap.length/3];  
 //вычисляем интенсивность  
 for (int pixel\_index = 0; pixel\_index < ready\_vector.length; pixel\_index++){  
 ready\_vector[pixel\_index] = ((float) (targeted\_bitmap[pixel\_index\*3] \* 0.0721))+ //BLUE  
 ((float) (targeted\_bitmap[pixel\_index\*3+1] \* 0.7154))+ //GREEN  
 ((float) (targeted\_bitmap[pixel\_index\*3+2] \* 0.2125)); //RED  
 }  
 try(FileWriter writer = new FileWriter("src/main/NumericData/source.txt", false))  
 {  
 for (int i = 0; i < ready\_vector.length; i++){  
 writer.write(String.*valueOf*(ready\_vector[i]));  
 writer.append('\n');  
 }  
 writer.flush();}  
 catch(IOException ex){  
 System.*out*.println(ex.getMessage());  
 }  
 try(FileWriter writer2 = new FileWriter("src/main/NumericData/source1.txt", false))  
 {  
 for (int i = 1; i < ready\_vector.length; i++){  
 writer2.write(String.*valueOf*(ready\_vector[i]));  
 writer2.append('\n');  
 }  
 writer2.write(String.*valueOf*(ready\_vector[ready\_vector.length-1]));  
 writer2.flush();  
 }  
 catch(IOException ex){  
 System.*out*.println(ex.getMessage());  
 }  
 }  
 public static byte[] ReadBytes (String source) throws IOException {  
 File input\_file = new File(source);  
 return Files.*readAllBytes*(input\_file.toPath());  
 }  
  
}

Приложение 4. M-файл для графических тестов

%читаем первый файл

fileID = fopen('source.txt', 'r');

formatSpec = '%f';

sizeA = [1 Inf];

A = fscanf(fileID, formatSpec, sizeA);

A = A';

fclose(fileID);

%читаем второй файл

fileID1 = fopen('source1.txt', 'r');

formatSpec1 = '%f';

sizeB = [1 Inf];

B = fscanf(fileID1, formatSpec1, sizeB);

B = B';

fclose(fileID1);

histogram(A);

figure();

plot (A, B, 'k.');