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. Для реализации генератор псевдослучайной последовательности битов на основе регистра сдвига с линейной обратной связью (РСЛОС) в конфигурации Галуа был создан класс GaloisLFSR рисунок 1.

Этот класс содержит конструктор и 2 метода.

Конструктор GaloisLFSR - принимает два массива: register, который содержит начальную последовательность битов, и taps которые указывают на те биты регистра, которые используются для вычисления обратной связи. Создает объект класса GaloisLFSR.

Метод shift - сдвигает элементы регистра на один бит вправо и вычисляет значение обратной связи. Затем вставляет вычисленное значение обратной связи в первый бит регистра.

Метод generate - генерирует последовательность псевдослучайных битов заданной длины, используя метод shift(). Возвращает массив int[], содержащий сгенерированные биты.

```java
public class GaloisLFSR {
    8 usages
    private int[] register;     // Содержит значения битов регистра
    2 usages
    private int[] taps;         // Содержит номера разрядов, используемых для обратной связи

    1 usage
    public GaloisLFSR(int[] register, int[] taps) {
        this.register = register;
        this.taps = taps;
    }

    1 usage
    public void shift() {
        int feedback = 0;

        // Вычисляем обратную связь
        for (int tap : taps) {
            feedback ^= register[tap];
        }

        // Сдвигаем все биты регистра вправо на 1
        for (int i = register.length - 1; i > 0; i--) {
            register[i] = register[i - 1];
        }

        // Вставляем обратную связь в первый бит регистра
        register[0] = feedback;
    }

    2 usages
    public int[] generate(int length) {
        int[] bits = new int[length];

        for (int i = 0; i < length; i++) {
            // Генерируем следующий бит и записываем его в массив
            bits[i] = register[register.length - 1];

            // Сдвигаем регистр на один бит вправо
            shift();
        }
```

Рисунок 1 – Класс GaloisLFSR.

Далее в Main мы создаем 2 массива содержащих начальное значение регистра и образующий многочлен, создаем экземпляр класс и выводим последовательность в консоль.

```
int[] register = {1, 1, 0, 0, 1};  // Начальное значение регистра
int[] taps = {3, 2};           // Образующий многочлен для конфигурации Галуа x^3 + x^2 + 1
GaloisLFSR lfsr = new GaloisLFSR(register, taps);
int[] sequence = lfsr.generate( length: 15);
System.out.println(Arrays.toString(sequence));
```

Рисунок 2 – Вывод последовательности в консоль.

2. Результат был представлен в виде точечной диаграммы, где по горизонтали отложены порядковые номера генерируемых битов, а по вертикали – их значения.

Для отображения диаграммы был создан класс LFSRDiagram, который наследуется от класса Panel и предназначен для отображения диаграммы.

Этот класс содержит два метода:

- Конструктор LFSRDiagram - принимает массив sequence, который содержит последовательность битов регистра. Создает объект класса LFSRDiagram.

- Метод paint - отображает диаграмму на панели. Принимает объект Graphics g, который используется для рисования, определяет размеры и ширину полосы на основе размера панели и длины последовательности. Затем отрисовывает оси координат и точки, соответствующие элементам регистра. Радиус окружности каждой точки равен половине ширины полосы, а координата Y вычисляется на основе значения бита.

Класс представлен на рисунке 3.

```java
public class LFSRDiagram extends Panel {
    4 usages
    private int[] sequence;

    1 usage
    public LFSRDiagram(int[] sequence) { this.sequence = sequence; }

    public void paint(Graphics g) {
        int width = getSize().width;
        int height = getSize().height;
        int barWidth = Math.max(1, width / sequence.length);

        // Рисуем оси координат
        g.setColor(Color.BLACK);
        g.drawLine( x1: 0,  y1: height / 2, width,  y2: height / 2);
        g.drawLine( x1: width / 2,  y1: 0,  x2: width / 2, height);

        // Рисуем точки
        g.setColor(Color.RED);
        for (int i = 0; i < sequence.length; i++) {
            int x = i * barWidth;
            int y = height / 2 - 10 * sequence[i];
            g.fillOval(x, y,  width: barWidth / 2,  height: barWidth / 2);
        }
    }
}
```

Рисунок 3 - класс LFSRDiagram.

В методе Main создается объект LFSRDiagram с использованием последовательности sequence. Далее создается объект Frame, который представляет окно для отображения диаграммы. В этом объекте добавляется обработчик событий для закрытия окна, который завершает выполнение программы при закрытии окна. Затем объект диаграммы добавляется в объект frame, размер окна устанавливается на 500 x 300, и окно становится видимым с помощью метода setVisible.

Представлен на рисунке 4.

```
// Создаем окно и добавляем в него диаграмму
Frame frame = new Frame();
frame.addWindowListener((WindowAdapter) windowClosing(event) → { System.exit( status: 0); });
frame.add(diagram);
frame.setSize( width: 500,  height: 300);
frame.setVisible(true);
```

Рисунок 4 – Создание диаграммы.

3. С помощью критерия x^2 оценивается качество генерируемой последовательности рисунок 4.

Переменная numBins, которая хранит количество групп, на которые разбивается последовательность для проверки соответствия распределению равномерному. В данном случае у нас есть только 2 возможных значения (0 и 1), поэтому numBins установлено на 2.

С помощью элементов массива sequence вычисляются частоты появления символов и сохраняются в массив observedFrequencies. Затем вычисляется ожидаемая частота символов и для каждой группы, проводится x^2 тест для проверки соответствия полученных частот равномерному распределению. Результат теста сохраняется в переменную chiSquared.

```
int numBins = 2; // кол-во групп, на которые разбивается последовательность.
double[] observedFrequencies = new double[numBins];
for (int i = 0; i < sequence.length; i++) {
    observedFrequencies[sequence[i]]++;
}
double expectedFrequency = sequence.length / (double) numBins;
double chiSquared = 0;
for (int i = 0; i < numBins; i++) {
    chiSquared += Math.pow(observedFrequencies[i] - expectedFrequency, 2) / expectedFrequency;
}

System.out.println("Chi-squared: " + chiSquared);
```

Рисунок 4 – Проверка x^2.

4. Зашифровываем изображение с использованием гаммирования с помощью Линейного Регистра Сдвига (LFSR) рисунок 5.

Загружаем изображение, далее определяем количество блоков в изображении не учитывая заголовок.

Затем для каждого блока получаем следующий 8-битный блок данных из изображения и применяем гаммирование с использованием Линейного Регистра Сдвига (LFSR).

Для гаммирования создается временный массив keyStream, который содержит биты, сгенерированные Линейным Регистром Сдвига (LFSR), затем применяется операция XOR для обеспечения шифрования данных для каждого блока.

В конце зашифрованные данные сохраняются в новый файл с именем "tux_encrypted.bmp".

```
// Загрузка изображения в память
byte[] imageData = loadFile( filename: "tux.bmp");

// Гаммирование каждого блока изображения
int blocksCount = (imageData.length - HEADER_SIZE) / BLOCK_SIZE;
for (int i = 0; i < blocksCount; i++) {
    // Получение следующего 8-битного блока для шифрования
    byte[] block = new byte[BLOCK_SIZE];
    System.arraycopy(imageData, srcPos: HEADER_SIZE + i * BLOCK_SIZE, block, destPos: 0, BLOCK_SIZE);

    // Гаммирование блока и сохранение результата
    byte[] cipherBlock = new byte[BLOCK_SIZE];
    byte[] keyStream = toByteArray(lfsr.generate( length: BLOCK_SIZE * 8));
    for (int j = 0; j < BLOCK_SIZE; j++) {
        cipherBlock[j] = (byte) (block[j] ^ keyStream[j]);
    }
    System.arraycopy(cipherBlock, srcPos: 0, imageData, destPos: HEADER_SIZE + i * BLOCK_SIZE, BLOCK_SIZE);
}

// Сохранение зашифрованного изображения на диск
saveFile( filename: "tux_encrypted.bmp", imageData);
```

Рисунок 5 – Шифрование изображения.

Итог:

Сгенерированная последовательность битов, точечная диаграмма и оценка качества с помощью критерия x^2 представлено на рисунке 6.

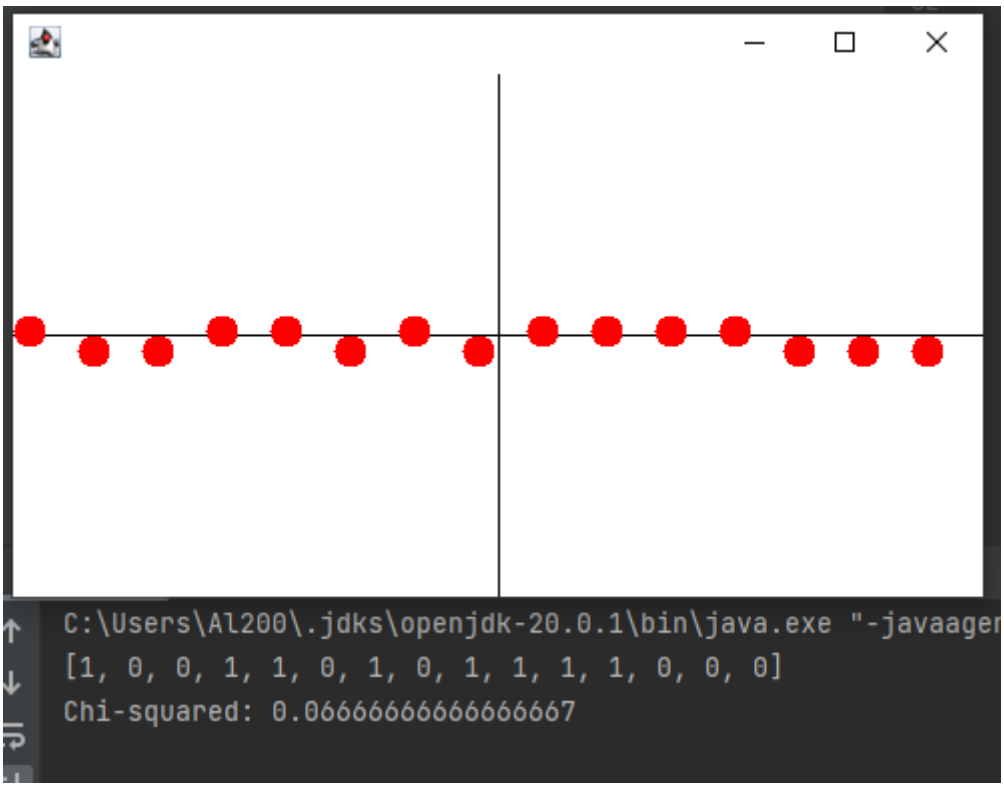

Рисунок 6 – Сгенерированная последовательность точечная диаграмма и оценка качества с помощью критерия x^2.

Зашифрованное изображение представлено на рисунке 7.

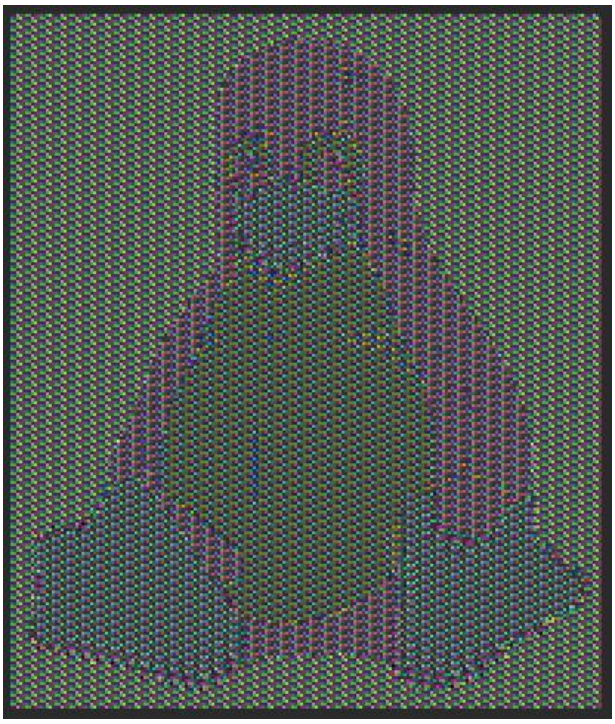

Рисунок 7 – Зашифрованное изображение.