МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(МОСКОВСКИЙ ПОЛИТЕХ)

Факультет информационных технологий
Кафедра «Инфокогнитивные технологии»

ЛАБОРАТОРНАЯ РАБОТА №5

на тему: *«Изучение свойств мультипликативной группы расширенного поля Галуа»*

Направление подготовки 09.03.03 «Прикладная информатика»
Профиль «Корпоративные информационные системы»
Дисциплина «Защита информации»

Выполнил:

студентка группы 201-361

Саблина Анна Викторовна

Проверил:

Харченко Елена Алексеевна

Москва 2023

Теоретическая часть

РСЛОС (*регистр сдвига с линейной обратной связью*) – это линейный регистр сдвига, используемый в теории кодирования и криптографии для генерации псевдослучайных последовательностей. Регистр сдвига состоит из нескольких регистров, связанных линейной обратной связью, и использует операцию XOR для комбинирования значений регистров.

Изучение свойств мультипликативной группы расширенного поля Галуа:

• связано с алгеброй и теорией чисел. Расширенное поле Галуа является *конечным полем*, в котором операции сложения и умножения определены над элементами поля. Мультипликативная группа расширенного поля Галуа состоит из всех ненулевых элементов поля, и эта группа образует абелеву группу относительно операции умножения.

• включает в себя исследование порядка группы, генераторов группы, цикличности группы, степеней элементов группы и др. алгебраических свойств. Эти свойства могут быть применены в различных областях, таких как криптография, кодирование и теория ошибок, для разработки алгоритмов и протоколов с высокой степенью надежности и безопасности.

Предположим, что наши изначальные значения начального состояния регистра (seed) и положения обратной связи (taps): 10110 и $x^5 + x^2 + x$.

Регистр сдвига будет иметь 5 битов, которые мы можем обозначить как x_4, x_3, x_2, x_1 и x_0, где x_4 является самым старшим битом, а x_0 – самым младшим битом. Используя эту нотацию, начальное состояние регистра может быть представлено, например, как 10110.

Обратная связь определяется многочленом $x^5 + x^2 + x$. Это означает, что для вычисления следующего бита регистра (x0), мы будем использовать биты x_4, x_2 и x_1, выполняя операцию *XOR* (исключающее *ИЛИ*) над этими битами. Иными словами, мы выполняем следующую операцию: $x_0 = x_4 \oplus x_2 \oplus x_1$.

Для генерации псевдослучайной последовательности с помощью этого РСЛОС, мы будем последовательно сдвигать значения битов регистра, начиная с начального состояния. Каждый раз, когда мы сдвигаем значения, мы также вычисляем новое значение x_0 с помощью обратной связи.

Итерация	Состояние регистра
0	10110
1	01011
2	10101
3	11010
4	01101
5	00110
6	10011
7	11001
8	11100
9	01110
10	10111
11	11011
12	11101
13	11110
14	01111
15	00111
16	10011
17	11001
18	11100
19	01110
20	10111

Гаммирование – это процесс применения гаммы (псевдослучайной последовательности, называемой также гаммой) к сообщению путем применения операции побитового XOR между каждым битом сообщения и соответствующим битом гаммы.

Гаммирование широко используется в криптографии для защиты конфиденциальности данных. Оно основано на принципе одноразового блокнота, где гамма представляет собой случайную последовательность,

используемую только один раз для шифрования сообщения. Гамма должна быть длиной, достаточной для обеспечения безопасности шифрования.

Хи-квадрат (χ^2) является статистическим тестом, который используется для проверки независимости или соответствия между наблюдаемыми и ожидаемыми частотами в категориальных данных. Он используется для определения, насколько хорошо эмпирические данные соответствуют ожидаемым значениям или для определения наличия связи между двумя категориальными переменными.

Практическая часть

Для реализации генератора псевдослучайной последовательности битов на основе регистра сдвига с линейной обратной связью (РСЛОС) в конфигурации Галуа, был создан проект на языке Python.

Общий принцип работы программы:

1. Программа использует библиотеки `Tkinter` и `Matplotlib` для создания графического интерфейса и отображения графиков.
2. Она позволяет пользователю генерировать последовательности битов с использованием линейного регистра сдвига с обратной связью Галуа (`GaloisLFSR`) и выполнять тест хи-квадрат на случайность сгенерированных последовательностей.
3. Программа также предоставляет функциональность шифрования файлов BMP с использованием операции XOR.
4. Графический интерфейс позволяет пользователю вводить значения начального состояния регистра и положения обратной связи, генерировать графики на основе сгенерированных последовательностей и запускать процесс шифрования файлов BMP.
5. Основной цикл событий `tkinter` обрабатывает пользовательский ввод и отображает интерфейс программы.

```
# Создание окна tkinter
window = tk.Tk()
window.title("dp_Lab5_py")

# Создание фигуры matplotlib
figure = Figure(figsize=(6, 4), dpi=100)
subplot = figure.add_subplot(111)

# Создание холста для отображения фигуры matplotlib в tkinter
canvas = FigureCanvasTkAgg(figure, master=window)
canvas.draw()
canvas.get_tk_widget().pack(side=tk.LEFT)

# Создание фрейма для кнопки и полей ввода
controls_frame = tk.Frame(window)
controls_frame.pack(side=tk.RIGHT)

label1 = tk.Label(controls_frame, text="Enter seed:")
```

```
label1.pack(anchor=tk.W)  # Выравнивание метки слева

entrySeed = tk.Entry(controls_frame)
entrySeed.pack()

label2 = tk.Label(controls_frame, text="Enter taps:")
label2.pack(anchor=tk.W)  # Выравнивание метки слева

entryTaps = tk.Entry(controls_frame)
entryTaps.pack()

button = tk.Button(controls_frame, text="Generate graph", command=update_graph)
button.pack()

# Создание кнопки для обновления графика
button = tk.Button(controls_frame, text="Encrypt file", command=encrypt_file)
button.pack()

# Запуск цикла событий tkinter
tk.mainloop()
```

Листинг 1 – Создание кликабельного интерфейса программы

Класс `GaloisLFSR` описывает генерацию последовательности битов с использованием линейного регистра сдвига с обратной связью Галуа.

```
# Класс GaloisLFSR - линейный регистр сдвига с обратной связью Галуа
class GaloisLFSR:
    def __init__(self, taps, seed):
        self.taps = taps
        self.state = seed

    def shift(self):
        feedback = sum(self.state[tap] for tap in self.taps) % 2
        self.state = self.state[1:] + [feedback]

    def generate_sequence(self, length):
        sequence = []
        for _ in range(length):
            sequence.append(self.state[0])
            self.shift()
        return sequence
```

Листинг 2 – Класс GaloisLFSR

Функция `update_graph()` описывает обновление графика на основе введенных значений `seed` и `taps`.

```
# Обновляет график на основе введенного начального состояния
# регистра и положений обратной связи
def update_graph():
    subplot.cla()  # Clear the previous plot
    seed_array = [int(bit) for bit in entrySeed.get()]
    taps_array = list(map(int, entryTaps.get().split()))
    lfsr = GaloisLFSR(taps_array, seed_array)
    bit_sequence = lfsr.generate_sequence(150)
    x = range(len(bit_sequence))
    y = bit_sequence

    # subplot.figure()
    subplot.plot(x, y, marker='o', linestyle='', color='b')
```

```
    subplot.set_xlabel("X")
    subplot.set_ylabel("Y")
    subplot.set_title("Updated Plot")
    canvas.draw()
```

Листинг 3 – Обновление графика на основе введенных данных

Функция `chi_squared_test(bit_sequence)` описывает выполнение теста хи-квадрат на случайность сгенерированных последовательностей.

```
# Тестирование последовательности битов на случайность
# с использованием хи-квадратного теста и вывод результатов
def chi_squared_test(bit_sequence):
    # Шаг 1: Расчет наблюдаемых частот
    observed_zeros = sum(bit == 0 for bit in bit_sequence)
    observed_ones = len(bit_sequence) - observed_zeros

    # Шаг 2: Расчет ожидаемых частот при предположении случайности
    total_bits = len(bit_sequence)
    expected_zeros = total_bits / 2
    expected_ones = total_bits / 2

    # Шаг 3: Расчет значения статистики хи-квадрат
    chi_squared = ((observed_zeros - expected_zeros) ** 2) / expected_zeros
    chi_squared += ((observed_ones - expected_ones) ** 2) / expected_ones

    # Шаг 4: Степени свободы
    degrees_of_freedom = 1

    # Шаг 5: Сравнение со значением критического значения
    critical_value = 3.841  # For a significance level of 0.05 and 1 degree of freedom

    # Вывод результатов
    print("Observed Zeros:", observed_zeros)
    print("Observed Ones:", observed_ones)
    print("Expected Zeros:", expected_zeros)
    print("Expected Ones:", expected_ones)
    print("Chi-Squared:", chi_squared)
    print("Degrees of Freedom:", degrees_of_freedom)
    print("Critical Value:", critical_value)

    if chi_squared < critical_value:
        print("The bit sequence is likely random.")
    else:
        print("The bit sequence is not likely random.")
```

Листинг 4 – Оценка качества генерируемой последовательности битов

Функции `xor_cipher_bmp_file(filename, key_seed, taps)` и `encrypt_file()` описывают шифрование файлов в формате BMP с использованием операции XOR.

```
# Шифрует файл BMP с использованием операции XOR и
# сохраняет зашифрованный файл с указанным именем
def xor_cipher_bmp_file(filename, key_seed, taps):
    with open(filename, 'rb') as file:
        bmp_data = file.read()

    header = bmp_data[:110]  # BMP header is 110 bytes
    image_data = bytearray(bmp_data[110:])
```

```
    key_length = len(image_data)
    lfsr = GaloisLFSR(taps, key_seed)  # Пример положений обратной связи: 2, 3 и 5
    key_sequence_bit = lfsr.generate_sequence(key_length * 8)
    key_sequence = [sum([byte[b] << b for b in range(0, 8)])
                    for byte in zip(*(iter(key_sequence_bit),) * 8)]
    for i in range(key_length):
        image_data[i] ^= key_sequence[i]

    encrypted_bmp_data = header + bytes(image_data)

    with open('encrypted.bmp', 'wb') as file:
        file.write(encrypted_bmp_data)

    print("Encryption completed. Encrypted BMP saved as 'encrypted.bmp'.")


filename = 'tux.bmp'


# Зашифровывает файл BMP с использованием
# введенного начального состояния регистра
# и положений обратной связи
def encrypt_file():
    seed_array = [int(bit) for bit in entrySeed.get()]
    taps_array = list(map(int, entryTaps.get().split()))
    xor_cipher_bmp_file(filename, seed_array, taps_array)
```

Листинг 5 – Шифрование изображения путем однократного гаммирования

Результаты работы программы:

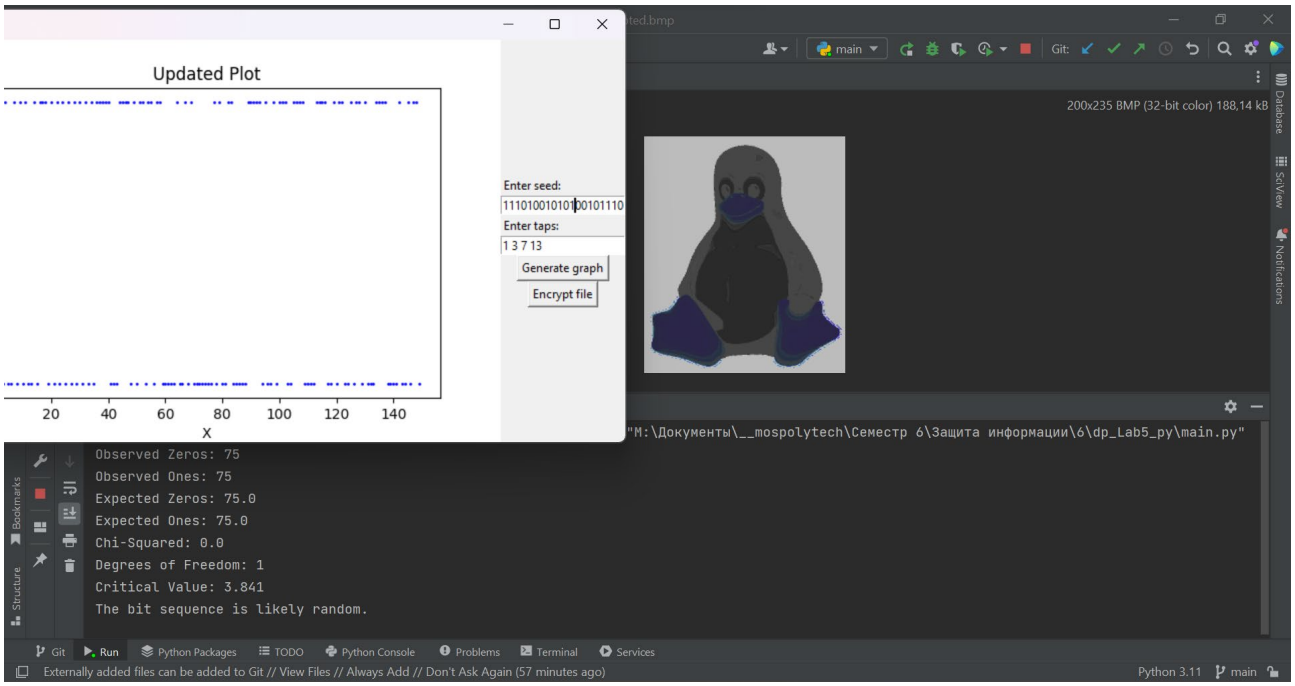

Рисунок 1 – Последовательность битов является идеально рандомной

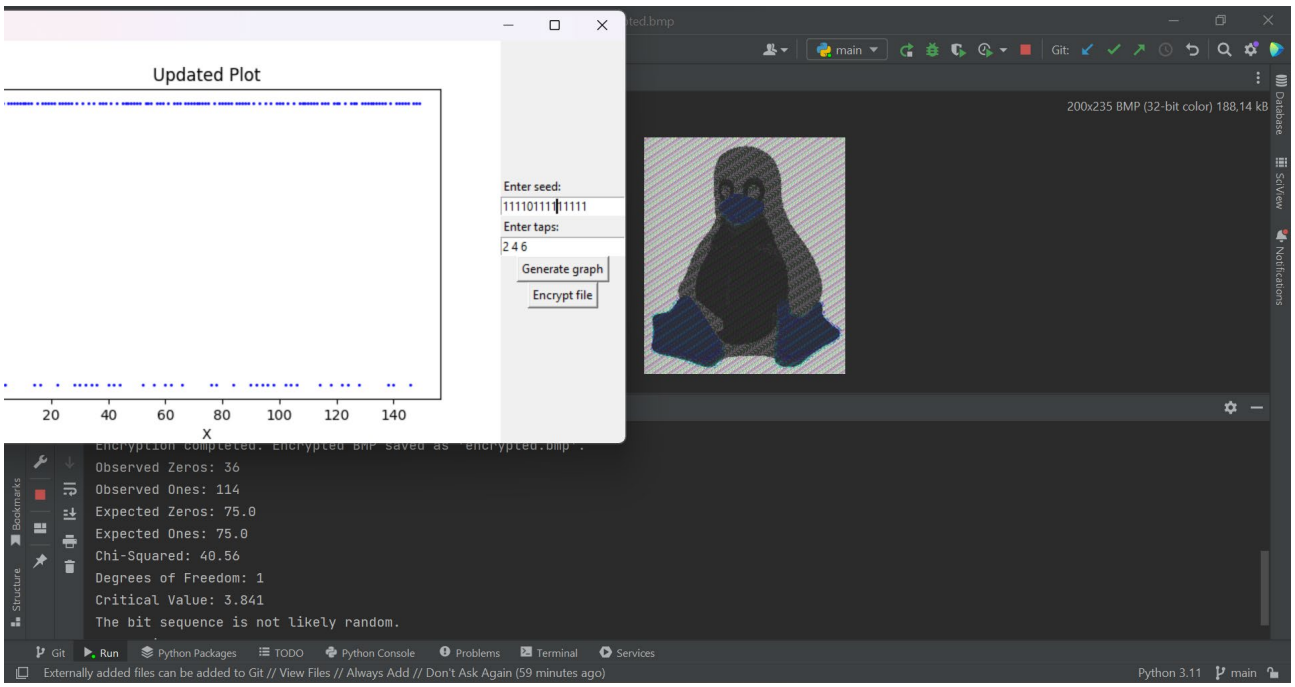

Рисунок 2 – Последовательность битов не является рандомной

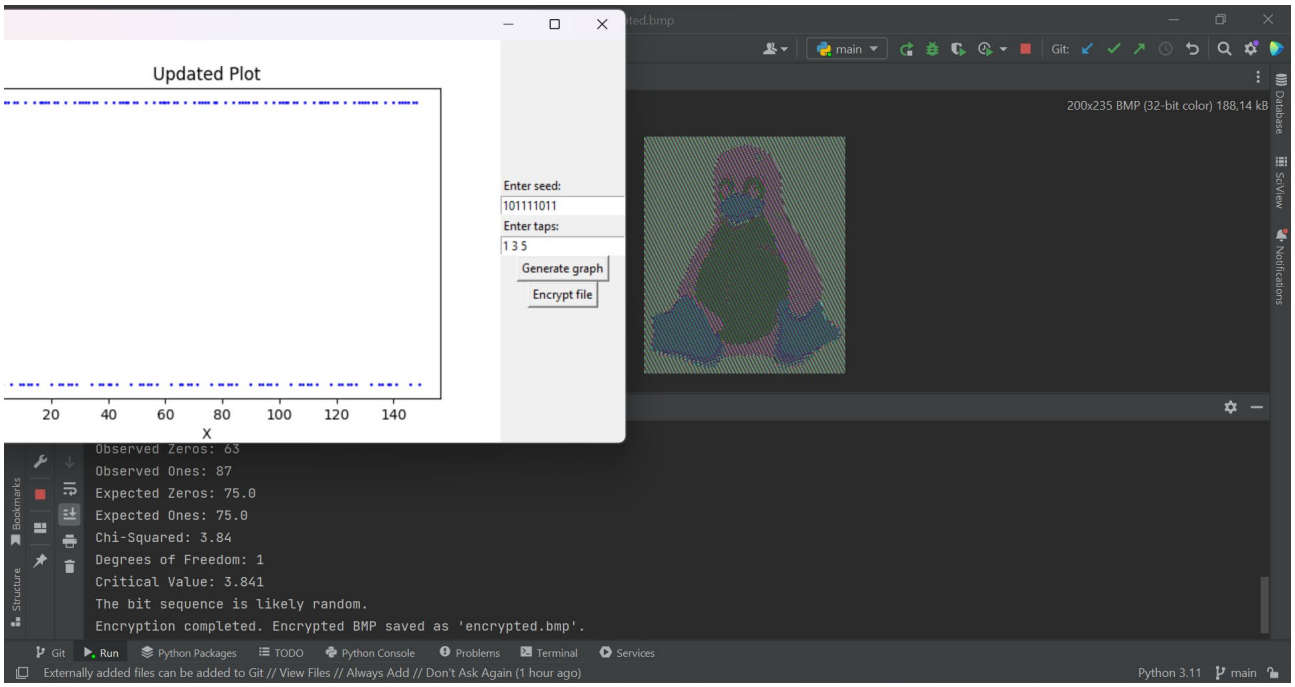

Рисунок 3 – Последовательность битов является рандомной

Ссылка на проект в репозитории GitHub:

– https://github.com/LazyShAman/dp/tree/main/5.