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

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

**Выполнил:**

студент группы 201-361

Погудин Александр

Москва 2023

Цель работы: реализовать генератор псевдослучайной последовательности битов на основе регистра сдвига с линейной обратной связью (РСЛОС) в конфигурации Галуа. Результат представьте в виде точечной диаграммы. С помощью критерия $\chi 2$ оцените качество любой генерируемой последовательности. Путем однократного гаммирования, не затрагивая заголовочную часть, зашифруйте изображение tux.png.

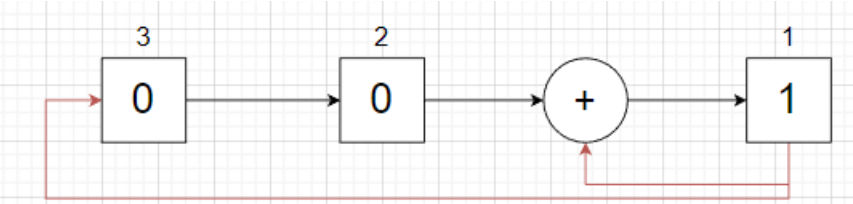
## Введение

Регистр сдвига с линейной обратной связью (РСЛОС) в конфигурации Галуа - это тип схемы генерации псевдослучайных последовательностей, который используется в шифровании и защите конфиденциальных данных. Он состоит из нескольких последовательных регистров, которые сдвигают свои значения на один бит с каждым тактом. Регистры сдвига преобразуют позиции двоичных чисел в значении регистра.

Одной из ключевых особенностей РСЛОС является его свойство линейной обратной связи, где значения в регистре обратно связываются с другими значениями в регистре, чтобы создать циклическую последовательность. Конфигурация Галуа относится к тому, какие значения используются в операциях обратной связи.

В РСЛОС в конфигурации Галуа каждый регистр имеет свой собственный многочлен Галуа, который используется для линейной обратной связи. Многочлен Галуа задает правила линейной обратной связи между регистрами и определяет, какие биты регистра можно использовать в обратной связи. Это позволяет линейно преобразовывать состояние регистра и создавать псевдослучайную последовательность данных.

РСЛОС в конфигурации Галуа часто используется в криптографии в качестве генератора псевдослучайных чисел. Он является легким в реализации и может быть быстро выполнен на аппаратном устройстве. Однако, его использование сейчас ограничено из-за недостаточной стойкости в отношении атак по времени и пространству.




p(x) = x^3 + x +1

| № | 3 | 2 | 1 | Бит |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 2 | 1 | 1 | 1 | 1 |
| 3 | 1 | 1 | 0 | 0 |
| 4 | 0 | 1 | 1 | 1 |
| 5 | 1 | 0 | 0 | 0 |
| 6 | 0 | 1 | 0 | 0 |
| 7 | 0 | 0 | 1 | 1 |

Мультипликативная группа расширенного поля Галуа - это математический объект, который имеет много применений в криптографии, а также в других областях математики и инженерии. Расширенное поле Галуа является расширением простого поля Галуа, то есть его размерность больше, чем у простого поля.

Мультипликативная группа расширенного поля Галуа состоит из всех ненулевых элементов поля, которые могут быть перемножены друг с другом в рамках поля, что делает его абелевой группой.

Мультипликативная группа расширенного поля Галуа обычно обозначается как GF(p^n)*, где p - простое число, а n - степень расширения поля. Примером такого поля является GF(2^8)*, которое состоит из 2^8-1 = 255 ненулевых элементов.

В криптографии мультипликативная группа расширенного поля Галуа используется для шифрования и дешифрования данных с помощью различных алгоритмов, таких как алгоритм AES (Advanced Encryption Standard). В AES мультипликативная группа используется для

преобразования открытого текста и ключа в шифрованный текст или для обратного преобразования зашифрованного текста в открытый текст и ключ. Мультипликативная группа расширенного поля Галуа также имеет применение в других областях математики и инженерии, таких как теория чисел, телекоммуникации, теория кодирования и т.д.

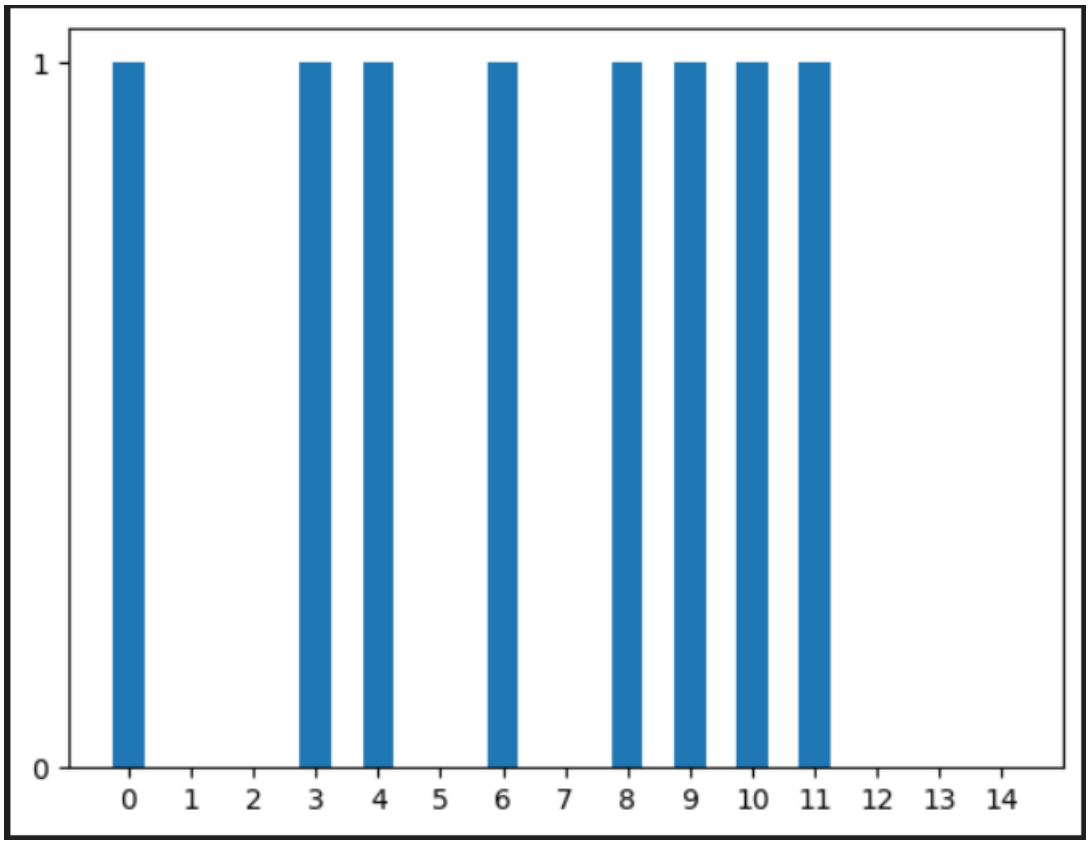
## Программы

Генератор псевдослучайной последовательности битов на основе регистра сдвига с линейной обратной связью (РСЛОС) в конфигурации Галуа, результат представлен в виде точечной диаграммы.

```python
import matplotlib.pyplot as plt

# Ввод данных
w = list(map(int, list(input("Введите начальное значение сдвигового регистра: ")))) # [0,0,0,1]
p = list(map(int, input("Введите образующий многочлен: ").split())) # [4, 3]
bit = p[0]

l = []

# конфигация Галуа
while not(w in l):
    l.append(w.copy())
    feedback = w[-1]
    for j in range(bit-1, 0, -1):
        if bit-j in p[1:]:
            w[j] = int(feedback != w[j-1])
        else:
            w[j] = w[j-1]
    w[0] = feedback

# Вывод результата
for value in l:
    print(value, value[-1])

s = ""
for bit in l:
    s += str(bit[-1]) + ", "

print(s[:-2])

# Построение диаграммы
plt.bar(list(range(len(l))), [i[-1] for i in l], width = 0.50)
plt.xticks(list(range(len(l))))
plt.yticks([0, 1])
```

Результат работы программы:

```
[0, 0, 0, 1] 1
[1, 1, 0, 0] 0
[0, 1, 1, 0] 0
[0, 0, 1, 1] 1
[1, 1, 0, 1] 1
[1, 0, 1, 0] 0
[0, 1, 0, 1] 1
[1, 1, 1, 0] 0
[0, 1, 1, 1] 1
[1, 1, 1, 1] 1
[1, 0, 1, 1] 1
[1, 0, 0, 1] 1
[1, 0, 0, 0] 0
[0, 1, 0, 0] 0
[0, 0, 1, 0] 0
1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0
```




Описание программы

1. Пользователю предлагается ввести начальное значение сдвигового регистра и образующий многочлен (параметры могут быть введены с клавиатуры).

2. Затем программа создает пустой список (названный "l"), который будет содержать все состояния регистра, используемые для генерации псевдослучайных чисел.

3. Она также создает переменную "bit", которая будет хранить количество бит в сдвиговом регистре (равное степени образующего многочлена).

4. Далее идет основной цикл программы, который работает до тех пор, пока текущее состояние сдвигового регистра еще не было использовано для генерации псевдослучайного числа.

5. В каждой итерации цикла текущее состояние регистра добавляется в список состояний "l".

6. Затем программа определяет обратную связь (последний бит регистра) и обновляет состояние регистра, используя образующий многочлен. В частности, каждый бит регистра, кроме первого, заменяется на значение, которое равно последнему биту регистра XOR с другим битом, который не содержится в многочлене обратной связи.

7. Последний бит регистра берется за обратную связь.

8. Производится вывод все состояния и последний бит в каждом состоянии.

9. Теперь строим график сгенерированной последовательности чисел с помощью библиотеки Matplotlib.

С помощью критерия χ2 оцениваем качество генерируемой последовательности.

```python
from scipy import stats

pars = lambda k, m, n: len([i for i in range(n-1) if seq[i] == k and seq[i+1] == m])

def chi_squared_test(seq):
    n = len(seq)
    exp_freq = (n-1) / 4

    # Посчет количество битовых пар 00, 01, 10 и 11 в последовательности.
    freq_00 = pars(0, 0, n)
    freq_01 = pars(0, 1, n)
    freq_10 = pars(1, 0, n)
    freq_11 = pars(1, 1, n)

    # Ожидаемое количество битовых пар для равномерно
    # распределенной последовательности той же длины.
    chi_square_stat = ((freq_00 - exp_freq)**2 + (freq_01 - exp_freq)**2
                       + (freq_10 - exp_freq)**2 + (freq_11 - exp_freq)**2) / exp_freq

    # число степеней свободы
    # В данном случае имеются 4 категории битовых пар,
    # поэтому число степеней свободы равно k = 4 - 1 = 3,
    # где k - число степеней свободы.
    k = 3

    # уровень значимости alpha (0.05)
    alpha = 0.05

    # табличное значение критерия chi_squared
    chi_squared_table_value = 7.815

    # расчет p-value (вероятность получить для данной вероятностной модели
    # распределения значений случайной величины такое же или более экстремальное
    #  значение статистики (среднего арифметического, медианы и др.),
    # по сравнению с ранее наблюдаемым, при условии, что нулевая гипотеза верна.)
    p_value = 1 - stats.chi2.cdf(chi_square_stat, k)

    return p_value >= alpha, chi_square_stat, chi_squared_table_value

seq = [1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0]  # ваша генерируемая последовательность бит
is_random, chi_sq_stat, chi_sq_table_value = chi_squared_test(seq)

print(f'Генерируемая последовательность: {seq}')
print(f'Статистика chi-squared: {chi_sq_stat}')

if is_random:
    print('Генерируемая последовательность можно признать случайной')
else:
    print('Генерируемая последовательность скорее всего не случайная')
```

Результат работы программы:

```
Генерируемая последовательность: [1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0]
Статистика chi-squared: 0.2857142857142857
Генерируемая последовательность можно признать случайной
```

Описание программы:

1. В первой строке импортируется модуль stats из библиотеки scipy.
2. Затем определяется лямбда-функция pars, которая принимает три аргумента k, m и n. Эта функция возвращает количество пар битов в последовательности seq, которые равны k и m. Функция используется при вычислении ожидаемых частот.
3. Функция chi_squared_test(seq) вызывается с аргументом seq, который является массивом из битовой последовательности, и выполняет следующие действия:
   1. Вычисляется длина последовательности и сохраняется в переменной n.
   2. Вычисляется ожидаемая частота exp_freq на каждую пару битов последовательности seq.
   3. Для каждой пары битов (i, j) (где i и j могут быть равными 0 или 1) в последовательности seq находится количество пар, соответствующих этой паре. Для пары (0, 0) количество пар сохраняется в переменной freq_00, для пары (0, 1) - в переменной freq_01, аналогично для пар (1,0) и (1,1).
   4. Вычисляется значение статистики критерия хи-квадрат chi_square_stat, используя формулу критерия для этой задачи.
   5. Задаются значения уровня значимости alpha, числа степеней свободы k и критического значения статистики критерия chi_squared_table_value.
   6. Находится P-value - вероятность получить значение статистики хи-квадрат большее, чем вычисленное значение chi_square_stat.
   7. Возвращаются значения:
      - p_value >= alpha - бинарный результат теста (равен True если последовательность seq случайна и False в противном случае).
      - chi_square_stat - значение статистики критерия хи-квадрат.

- chi_squared_table_value - критическое значение статистики критерия по таблице стандартных значений.

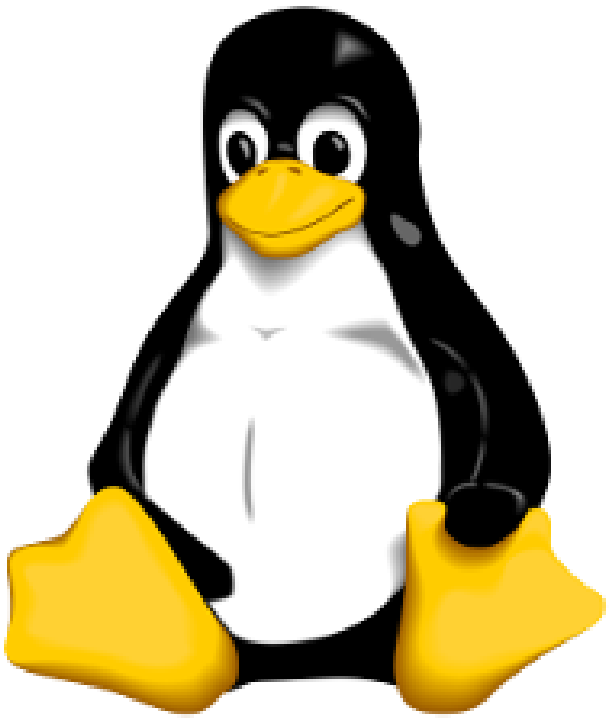
4. Наконец, функция вызывается для тестирования последовательности seq, и результаты сохраняются в переменные is_random, chi_sq_stat, chi_sq_table_value. Значение is_random равно True, так как предоставленная последовательность seq является случайной.

Путем однократного гаммирования, не затрагивая заголовочную часть, зашифруется изображение tux.png.

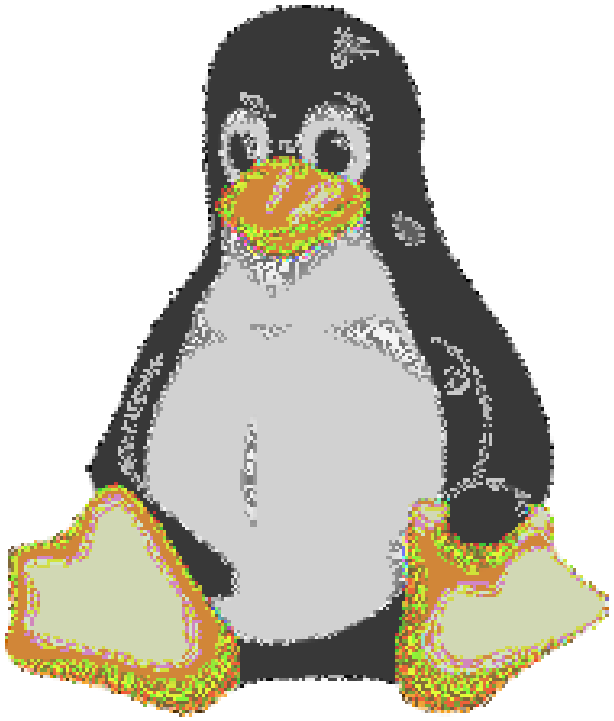
```
from PIL import Image

p = [8, 6, 5, 4]
bit = p[0]

image = Image.open("tux.png")
image_bytes = image.convert("RGB").tobytes()

pixels = list(image.getdata())

def chifr(color):
    w = list(map(int, list(bin(color)[2:].zfill(8))))
    feedback = w[-1]
    for j in range(bit-1, 0, -1):
        if bit-j in p[1:]:
            w[j] = int(feedback != w[j-1])
        else:
            w[j] = w[j-1]
    w[0] = feedback
    return int("".join(list(map(str, w))), 2)

for i in range(len(pixels)):
    r, g, b, a = pixels[i]
    pixels[i] = (chifr(r), chifr(g), chifr(b), chifr(a))

image_ecb = Image.new(image.mode, image.size)
image_ecb.putdata(pixels)
image_ecb.save(f"tuxNEW.png", "PNG")
```

Результат работы программы:

Исходное изображение



Итоговое изображение



Описание программы:

1. В первой строке программы импортируется модуль Image из библиотеки PIL (Python Imaging Library), который позволяет открывать и сохранять изображения различных форматов.

2. Создается список p, который содержит четыре элемента. Этот список определяет, какие биты используются для шифрования, каким образом происходит обратная связь и каков порядок отбора битов.

3. Берется первый элемент из списка p и сохраняется в переменной bit. Этот бит будет использоваться при шифровании.

4. Используется метод Image.open, чтобы открыть изображение "tux.png". Содержимое изображения конвертируется в формат RGB и конвертируется в байты, которые сохраняются в переменной image_bytes.

5. Создается список pixels, который содержит значения каждого пикселя изображения.

6. Определяется функция chifr, которая принимает одно значение color. Данная функция получает двоичную запись значения цвета, затем выполняет шифрование на основе значений из списка p. Каждому пикселю изображения присваивается новое значение, которое было зашифровано(преобразовано) с помощью данной функции.

7. Выполняется цикл для каждого пикселя в pixels. Затем извлекается значение каждого цветового канала (R, G, B, A) и передается функции chifr, чтобы зашифровать значение каждого канала.

8. Создается новый экземпляр объекта image_ecb с теми же размерами, что и исходное изображение. Затем метод putdata используется, чтобы поместить пиксели с новыми зашифрованными значениями. Новое изображение сохраняется в файл "tuxNEW.png".