**ЛАБОРАТОРНАЯ РАБОТА №3**

“Поточное симметричное шифрование”

по дисциплине

‘Информационная безопасность’

Вариант 5

***Выполнил:***

Соболев Иван Александрович

**Группа:** P34312

***Преподаватель:***

Маркина Татьяна Анатольевна

Санкт-Петербург, 2024

# Цель работы

# Изучение структуры и основных принципов работы современных алгоритмов поточного симметричного шифрования, приобретение навыков программной реализации поточных симметричных шифров.

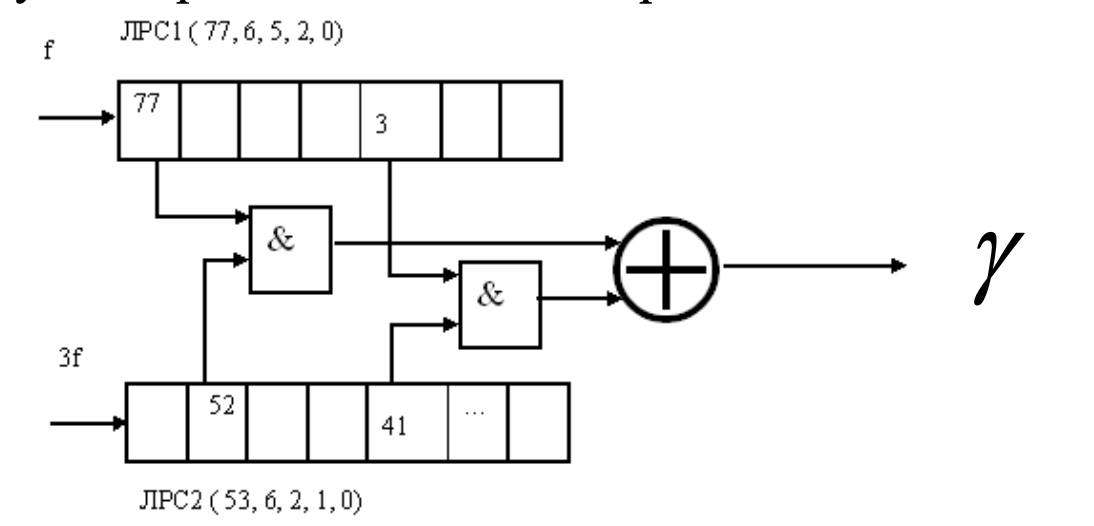
# Программные и аппаратные средства

Для выполнения лабораторной работы был использован компьютер со следующими характеристиками:

* Процессор: Apple M2
* Видеокарта: Apple M2
* Объем оперативной памяти: 8GB
* Использована операционная система: macOS 14.4.1
* Версия Python: 3.13

# Задание

Реализовать в программе поточное кодирование текста, вводимого с клавиатуры, с помощью заданной нелинейной схемы, использующей разные частоты тактирования ЛРС.



# Листинг разработанной программы

main.py

from encoder import encode  
from io\_utils import read\_from\_file, read\_filename, write\_to\_file, print\_red  
  
  
def main():  
 try:  
 input\_file = read\_filename(True)  
 output\_file = read\_filename(False)  
 if input\_file is not None and output\_file is not None:  
 data = read\_from\_file(input\_file)  
 encoded = encode(data)  
 write\_to\_file(output\_file, encoded)  
 except Exception as e:  
 print\_red(f'Ошибка чтения из файла: {str(e)}')  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

except Exception as e:  
 print\_red(f'Ошибка чтения из файла: {str(e)}')  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

encoder.py

from LFSR import LFSR  
  
INIT\_STATE\_LFSR1 = '100101111110010000110011001000111001110001111010101101101100101100000110111111'  
INIT\_STATE\_LFSR2 = '100000111111000111100001001010111001010001000110110011'  
  
seed1 = [int(bit) for bit in INIT\_STATE\_LFSR1]  
seed2 = [int(bit) for bit in INIT\_STATE\_LFSR2]  
lfsr1 = LFSR(seed1, [75, 6, 5, 2, 0])  
lfsr2 = LFSR(seed2, [53, 6, 2, 1, 0])  
  
  
def nonlinear\_schema(i):  
 and\_result = lfsr1.data[77] & lfsr2.data[52]  
 and\_result\_2 = lfsr1.data[3] & lfsr2.data[41]  
 lfsr1.shift\_right() # Обновляем LFSR1 каждый бит  
 if i % 3 == 0:  
 lfsr2.shift\_right() # Обновляем LFSR2 каждый третий бит  
 return and\_result ^ and\_result\_2  
  
  
def encode(text: bytes) -> bytes:  
 encoded = []  
 i = 0  
 for byte in text:  
 binary\_string = format(byte, '08b')  
 for bit in binary\_string:  
 gamma = nonlinear\_schema(i)  
 encoded.append(int(bit) ^ gamma)  
 i += 1  
 transformed\_text\_bytes = [  
 int(''.join(map(str, encoded[i:i + 8])), 2) for i  
 in range(0, len(encoded), 8)  
 ]  
 return bytes(transformed\_text\_bytes)

io\_utils.py

import os  
from typing import Optional  
  
  
def print\_red(message: str) -> None:  
 *"""Функция для вывода ошибок"""* print(f"\033[91m{message}\033[0m")  
  
  
def print\_green(message: str) -> None:  
 *"""Функция для вывода ответов сервиса"""* print(f"\033[92m{message}\033[0m")  
  
  
def read\_filename(is\_input) -> str:  
 *"""Функция чтения имени файла"""* if is\_input:  
 filename = input("Введите названия входного файла: ")  
 else:  
 filename = input("Введите названия выходного файла: ")  
 if not filename.strip():  
 print\_red("Ошибка: имя файла не должно быть пустым.")  
 return None  
 return filename  
  
  
def read\_from\_file(filename) -> Optional[bytes]:  
 *"""Функция чтения из файла"""* try:  
 if not os.path.isfile(filename):  
 print\_red(f"Ошибка: файл '{filename}' не найден.")  
 return None  
 with open(filename, "rb") as f:  
 buf = f.read()  
 if not buf.strip():  
 print\_red("Ошибка: файл пуст.")  
 return None  
 return buf  
 except Exception as e:  
 print\_red(f'Ошибка чтения из файла: {str(e)}')  
 return None  
  
  
def write\_to\_file(filename, content):  
 *"""Функция записи в файл"""* try:  
 with open(filename, 'wb') as file:  
 file.write(content)  
 print\_green(f"Данные успешно записаны в файл: {filename}")  
 except IOError as e:  
 print\_red(f"Ошибка записи в файл {filename}: {e}")  
 except Exception as e:  
 print\_red(f"Произошла ошибка: {e}")

print\_green(f"Данные успешно записаны в файл: {filename}")  
except IOError as e:  
 print\_red(f"Ошибка записи в файл {filename}: {e}")  
except Exception as e:  
 print\_red(f"Произошла ошибка: {e}")

LFSR.py

class LFSR:  
 def \_\_init\_\_(self, seed: list[int], taps: list[int]):  
 self.taps = taps  
 self.data = seed  
  
 def shift\_right(self):  
 new\_bit = 0  
 for tap in self.taps:  
 new\_bit ^= self.data[tap]  
 for i in range(len(self.data) - 1):  
 self.data[i] = self.data[i + 1]  
 self.data[-1] = new\_bit

# Результаты работы программы

Исходный текст:  
В одной из отдаленных улиц Москвы, в сером доме с белыми колоннами, антресолью и покривившимся балконом, жила некогда барыня, вдова, окруженная многочисленною дворней.

Шифрование текста:  


Результат шифрования:



Дешифрование текста:  


Дешифрованный текст:  
В одной из отдаленных улиц Москвы, в сером доме с белыми колоннами, антресолью и покривившимся балконом, жила некогда барыня, вдова, окруженная многочисленною дворней.