Лабораторная работа №2

Дисциплина: Математические основы защиты информации и информационной безопасности

Аветисян Давид Артурович

Содержание

1 Цель работы **5**

2 Задание **6**

3 Выполнение лабораторной работы **7**

4 Выводы **19**

List of Tables

List of Figures

3.1 Начало реализации маршрутного шифрования 7
3.2 Реализация маршрутного шифрования 8
3.3 Реализация формирования криптограммы 9
3.4 Проверка реализации маршрутного шифрования 10
3.5 Начало реализации шифрования с помощью решёток 11
3.6 Реализация шифрования с помощью решёток 12
3.7 Функция вращения матрицы на 90 градусов по часовой стрелк . . 12
3.8 Первый и второй цикл заполнения матрицы текстом пользователя 13
3.9 Третий и четвёртый цикл заполнения матрицы текстом пользователя 14
3.10 Тест реализации маршрутного шифрования из лабораторной работы №2 15
3.11 Тест реализации маршрутного шифрования для дополнительной проверки 15
3.12 Реализация таблицы с русским алфавитом 16
3.13 Начало реализации шифра Виженера 16
3.14 Реализация шифра Виженера 17
3.15 Проверка реализации шифра Виженера 18

1 Цель работы

Познакомиться с шифрами перестановки.

2 Задание

1. Программно реализовать маршрутное шифрование.
2. Программно реализовать шифрование с помощью решёток.
3. Программно реализовать шифр Виженера.

3 Выполнение лабораторной работы

1) Все шифрования я реализовывал на языке python. Сначала я реализовал возможность выбора одного из трёх шифров пользователем. Далее при помощи match-case я реализовал выполнение конкретных шифрований. Реализацию я начал с маршрутного шифрования. Я сделал запрос текста и пароля у пользователя, их фильтрацию на наличие пробелов, цифр или знаков пунктуации, а также возвёл всё в верхний регистр. Также я добавил проверки текста и пароля на соответствие требованиям для шифрования.

```
print('Выберите один из шифров перестановки:')
print('1. Маршрутное шифрование')
print('2. Шифрование с помощью решёток')
print("3. Таблица Виженера")
choice = input('Введите номер выбранного шифра: ')
match choice:
    case '1':
        text = input('Введите текст: ')
        filter_text = ''.join(filter(str.isalnum, text)).upper()
        password = input('Введите пароль: ')
        filter_password = ''.join(filter(str.isalnum, password)).upper()
        if len(filter_text) > len(filter_password):
            if len(set(filter_password)) == len(filter_password):
                encrypter_route = route_cipher(filter_text, filter_password)
                print("Криптограмма:", encrypter_route)
            else:
                print('Буквы пароля не должны повторяться!')
        else:
            print('Текст должен быть длиннее пароля!')
    case '2':
        print('2. Шифрование с помощью решёток')
    case '3':
        print("3. Таблица Виженера")
    case _:
        print('Некорректный выбор!')
```

Figure 3.1: Начало реализации маршрутного шифрования

Я реализовал русский алфавит для удобного заполнения массивов. Далее я

реализовал функцию route_cipher, в которой создал матрицу размером с текст пользователя и шириной с его пароль. Я заполнил её побуквенно текстом пользователя, а пустые места заполнил случайными русскими буквами.

```python
import math
import random

russian_letter = [chr(i) for i in range(ord('А'), ord('Я') + 1)]

def cryptogram(matrix, password):
    return

def route_cipher(text, password):
    rows = math.ceil(len(text)/len(password))
    cols = len(password)
    matrix = []
    for _ in range(rows):
        row = [random.choice(russian_letter) for _ in range(cols)]
        matrix.append(row)

    i, j = 0, 0
    for char in text:
        matrix[i][j] = char
        j += 1
        if j >= cols:
            i += 1
            j = 0

    print('Маршрутное шифрование:')
    return cryptogram(matrix, password)
```

Figure 3.2: Реализация маршрутного шифрования

После я реализовал общую для первого и второго шифрования функцию cryptogram, которая из заданного массива и введённого пользователем пароля создаёт таблицу, которую выводит, а затем создаётся криптограмму. Данная функция сначала добавляет к одному массиву 2 строчки (с паролем и ASCII номерами букв этого пароля). Далее она берёт последнюю строчку матрицы, сортирует по возрастанию, а затем формирует новую матрицу из старой, учитывая отсортированную последнюю строку. Этот функционал схож как для маршрутного шифрования, так и для шифрования с помощью решёток.

```python
def cryptogram(matrix, password):
    matrix.append[''] * len(password)
    matrix.append[''] * len(password)
    j = 0
    for char in password:
        matrix[-2][j] = char
        matrix[-1][j] = ord(char)
        j += 1

    for i in range(len(matrix[:-1])):
        for j in range(len(matrix[i])):
            value = matrix[i][j]
            if i == len(matrix[:-2]):
                print(Fore.GREEN + str(value) + Style.RESET_ALL, end=' ')
            else:
                print(value, end=' ')
        print()

    last_row = matrix[-1]
    sorted_indices = sorted(range(len(last_row)), key=lambda x: last_row[x])
    sorted_matrix = []
    for row in matrix[:-2]:
        sorted_row = [row[i] for i in sorted_indices]
        sorted_matrix.append(sorted_row)
    transposed_matrix = [[row[i] for row in sorted_matrix] for i in range(len(sorted_matrix[0]))]
    result = ''.join(''.join(row) for row in transposed_matrix)
    return result
```

Figure 3.3: Реализация формирования криптограммы

Далее я запустил два теста через командную строку. Один тест как в теории к лабораторной работе №2. Второй тест для дополнительной проверки. Шифрование совпало с тестом в лабораторной работе №2, и реализовано верно.

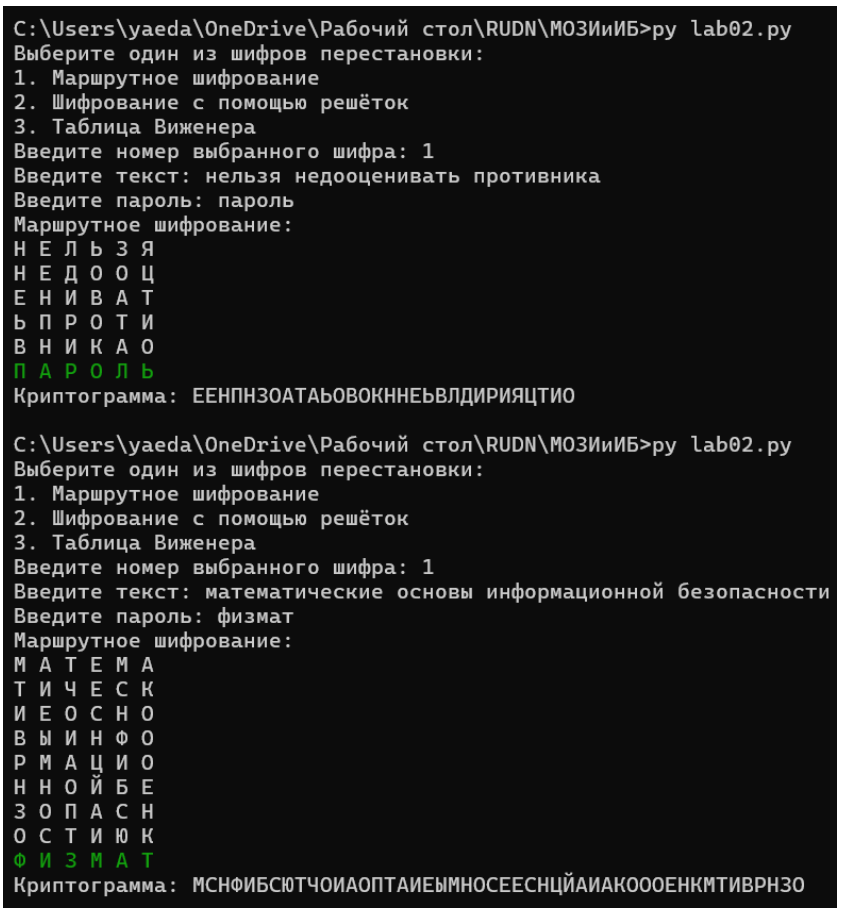

Figure 3.4: Проверка реализации маршрутного шифрования

2) Затем я перешёл к реализации шифрования с помощью решёток. Я аналогично предыдущему шифрованию запросил текст у пользователя, но в данном случае я запрашиваю пароль необходимой длины 2k, как сказано в теории к лабораторной работы. При этом длина текста N должна быть равна k**2. Также я аналогично проверяю текст и пароль на соответствие требованиям для шифрования.

```python
case '2':
    text = input('Введите текст: ')
    filter_text = ''.join(filter(str.isalnum, text)).upper()
    sqrt_text = math.ceil(math.sqrt(len(filter_text)))
    if sqrt_text % 2 == 1:
        sqrt_text += 1
    while len(filter_text) < (sqrt_text**2):
        filter_text += random.choice(russian_letters)
    password = input(f'Введите пароль длинной {sqrt_text} букв(ы): ')
    filter_password = ''.join(filter(str.isalnum, password)).upper()
    if len(set(filter_password)) == len(filter_password):
        if sqrt_text == len(filter_password):
            encrypted_lattice = lattice_cipher(filter_text, filter_password)
            print("Криптограмма:", encrypted_lattice)
        else:
            print(f'В пароле должно быть {sqrt_text} букв(ы)!')
    else:
        print('Буквы пароля не должны повторяться!')
```

Figure 3.5: Начало реализации шифрования с помощью решёток

Далее я реализовал функцию lattice_cipher, в которой я создал матрицу размера 2k и заполнил её нулями. Далее я заполнял каждую четверть значениями от 1 до k**2 в соответствии с инструкцией учебника. Далее я определил случайным образом по одному уникальному значению из получившейся матрицы и записал их индексы. Далее я вывел данную матрицу, закрашивая выбранные значения красным цветом.

```
def lattice_cipher(text, password):
    rows, cols = len(password), len(password)
    matrix = [[0 for _ in range(cols)] for _ in range(rows)]
    k = int(len(password)/2)

    for _ in range(4):
        i, j, number = 0, 0, 0
        for i in range(k):
            for j in range(k):
                number += 1
                matrix[i][j] = number
            matrix = rotate_matrix_clocwise(matrix)

    unique_values = set()
    for row in matrix:
        unique_values.update(row)
    coordinates = {value: [] for value in unique_values}
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            value = matrix[i][j]
            if value in coordinates:
                coordinates[value].append((i, j))
        selected_coordinates = {}
        for number in range(1, k**2+1):
            if coordinates[number]:
                selected_coordinates[number] = random.choice(coordinates[number])

    print("Решето:")
    for i in range(len(matrix)):
        for j in range (len(matrix[i])):
            value = matrix[i][j]
            if (i,j) in selected_coordinates.values():
                print(Fore.RED + str(value) + Style.RESET_ALL, end=' ')
            else:
                print(value, end=' ')
        print()
```

Figure 3.6: Реализация шифрования с помощью решёток

Для того чтобы заполнить матрицу значениями от 1 до k**2, я реализовал функцию вращения матрицы на 90 градусов по часовой стрелке.

```
def rotate_matrix_clocwise(matrix):
    transposed_matrix = []
    for j in range(len(matrix[0]) - 1, -1, -1):
        new_row = []
        for i in range(len(matrix)):
            new_row.append(matrix[i][j])
        transposed_matrix.append(new_row)
    return transposed_matrix
```

Figure 3.7: Функция вращения матрицы на 90 градусов по часовой стрелк

Далее для правильного заполнения матрицы текстом пользователя, я реализо-

вал 4 похожих, но разных цикла. Сначала я заполнял все уникальные значения побуквенно текстом пользователя. Затем я поворачивал матрицу на 90 градусов по часовой стрелке, и менял направление заполнения матрицы в соответствии с описанием в лабораторной работе №2. И обязательно в конце использовал ранее реализованную функцию формирования криптограммы.

```
letter, i, j, count = 0, 0, 0, 0
while count < k**2:
    if (i,j) in selected_coordinates.values():
        matrix[i][j] = text[letter]
        letter += 1
        count += 1
    j += 1
    if j >= cols:
        i += 1
        j = 0
    matrix = rotate_matrix_clocwise(matrix)

i, j, count = rows - 1, 0, 0
while count < k**2:
    if (i,j) in selected_coordinates.values():
        matrix[i][j] = text[letter]
        letter += 1
        count += 1
    i -= 1
    if i < 0:
        j += 1
        i = rows - 1
    matrix = rotate_matrix_clocwise(matrix)
```

Figure 3.8: Первый и второй цикл заполнения матрицы текстом пользователя

```
i, j, count = rows - 1, cols - 1, 0
while count < k**2:
    if (i,j) in selected_coordinates.values():
        matrix[i][j] = text[letter]
        letter += 1
        count += 1
    j -= 1
    if j < 0:
        i -= 1
        j = cols - 1
    matrix = rotate_matrix_clocwise(matrix)

i, j, count = 0, cols - 1, 0
while count < k**2:
    if (i,j) in selected_coordinates.values():
        matrix[i][j] = text[letter]
        letter += 1
        count += 1
    i += 1
    if i >= rows:
        j -= 1
        i = 0
    matrix = rotate_matrix_clocwise(matrix)

print('Шифрование с помощью решёток:')
return cryptogram(matrix, password)
```

Figure 3.9: Третий и четвёртый цикл заполнения матрицы текстом пользователя

Далее я запустил два теста через командную строку. Один тест как в теории к лабораторной работе №2. Второй тест для дополнительной проверки. Шифрование совпало с тестом в лабораторной работе №2, и реализовано верно.

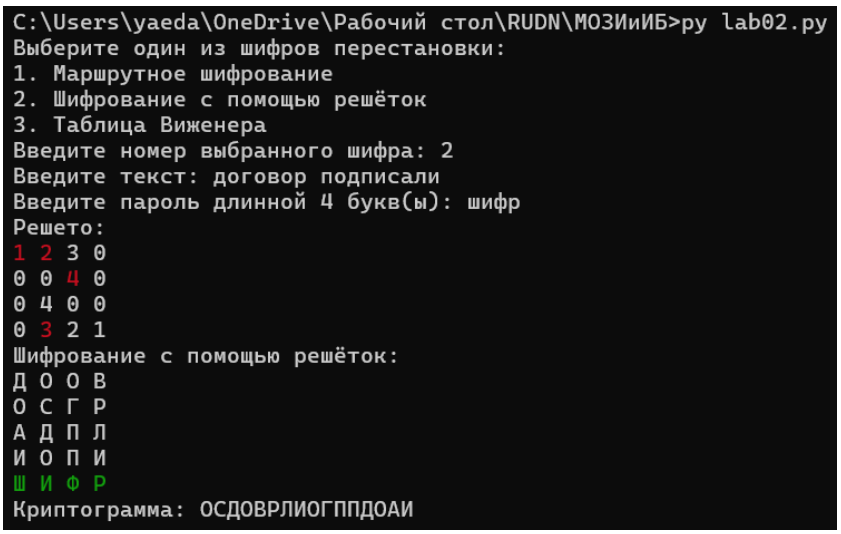

Figure 3.10: Тест реализации маршрутного шифрования из лабораторной работы №2

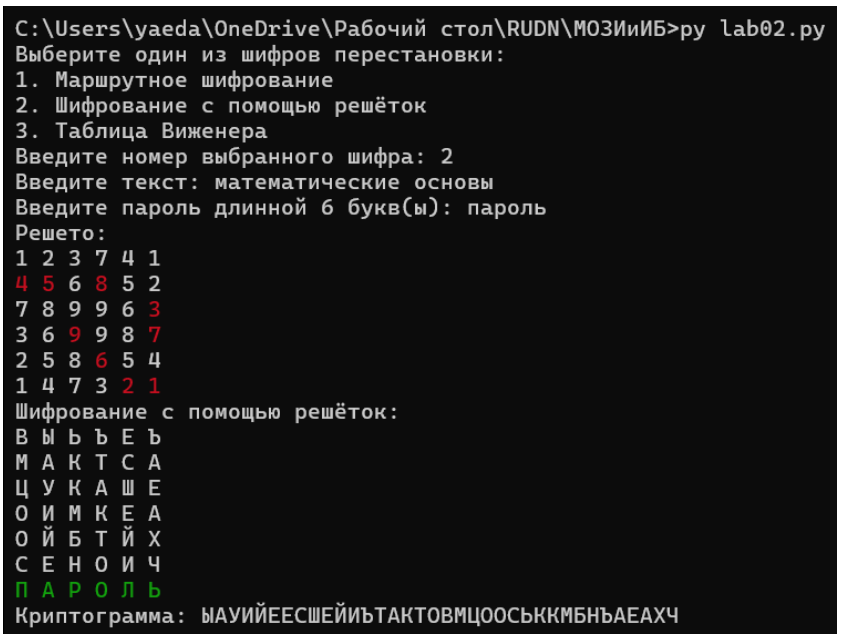

Figure 3.11: Тест реализации маршрутного шифрования для дополнительной проверки

3) И в конце я перешёл к реализации шифра Виженера. В первую очередь для его реализации нам потребуется таблица с русским алфавитом, где каждая следующая строка сдвигается на одну букву. Данную таблицу я реализовал в виде матрицы, используя ранее составленную матрицу с русским

алфавитом.

```
import math
import random
import numpy as np
from colorama import Fore, Style, init

init(autoreset=True)

russian_letters = [chr(i) for i in range(ord('А'), ord('Я') + 1)]
russian_matrix = np.empty(len(russian_letters). len(russian_letters), dtype='<U1')
for i in range(len(russian_letters)):
    for j in range(len(russian_letters)):
        letter_index = (i + j) % len(russian_letters)
        russian_matrix[i][j] = russian_letters[letter_index]
```

Figure 3.12: Реализация таблицы с русским алфавитом

Затем я перешёл к реализации шифра Виженера. Я аналогично предыдущим шифрованиям запросил текст и пароль у пользователя, отфильтровал их, а также проверил на соответствие требованиям для шифрования. Так как для шифрования используется таблица с русским алфавитом, то и текст и пароль должны содержать только русские буквы.

```
case '3':
    text = input('Введите текст: ')
    filter_text = ''.join(filter(str.isalnum, text)).upper()
    password = input('Введите пароль: ')
    filter_password = ''.join(filter(str.isalnum, password)).upper()
    if len(filter_text) > len(filter_password):
        if all('А' <= char <= 'Я' for char in filter_text):
            if all('А' <= char <= 'Я' for char in filter_password):
                vigener_cipher(filter_text, filter_password)
            else:
                print('Пароль должен быть только из русских букв (и без ё)!')
        else:
            print('Текст должен быть только из русских букв (и без ё)!')
    else:
        print('Текст должен быть длиннее пароля!')
case _:
    print('Некорректный выбор!')
```

Figure 3.13: Начало реализации шифра Виженера

Далее я реализовал функцию vigener_cipher, в которой создаётся матрица, где первая строка - текст пользователя, вторая строка - повторяющийся пароль пользователя, а третья строка пуста (для будущей криптограммы). Далее я брал первую букву первой строки (текста пользователя), находил её в первой строке

таблицы с русским алфавитом и записывал index j. Затем я брал первую букву второй строки (повторяющийся пароль), находил её в первом столбце таблицы с русским алфавитом и записывал index i. И наконец я находил в таблице с русским алфавитом букву с индексами (i, j), и записывал её в первый слот третьей строки (будущей криптограммы). Так для каждой буквы текста пользователя.

```
def vigener_cipher(text, password):
    row1 = list(text)
    repeated_password = (password * (len(text) // len(password) + 1))[:len(text)]
    row2 = list(repeated_password)
    row3 = ['' for _ in range(len(text))]
    matrix = [row1, row2, row3]

    n, index_i, index_j = 0, -1, -1
    for n in range(len(text)):
        for j in range(len(russian_matrix[0])):
            if russian_matrix[0][j] == matrix [0][n]:
                index_j = j
                break
        for i in range(len(russian_matrix)):
            if russian_matrix[i][0] == matrix [1][n]:
                index_i = i
                break
        matrix[2][n] = russian_matrix[index_i][index_j]

    print("Шифр Виженера:")
    for i in range(len(matrix)):
        for j in range (len(matrix[i])):
            value = matrix[i][j]
            if i == len(matrix[:-1]):
                print(Fore.GREEN + str(value) + Style.RESET_ALL, end=' ')
            else:
                print(value, end=' ')
        print()
    return
```

Figure 3.14: Реализация шифра Виженера

Далее я запустил два теста через командную строку. Один тест как в теории к лабораторной работе №2. Второй тест для дополнительной проверки. Шифрование совпало с тестом в лабораторной работе №2, и реализовано верно.

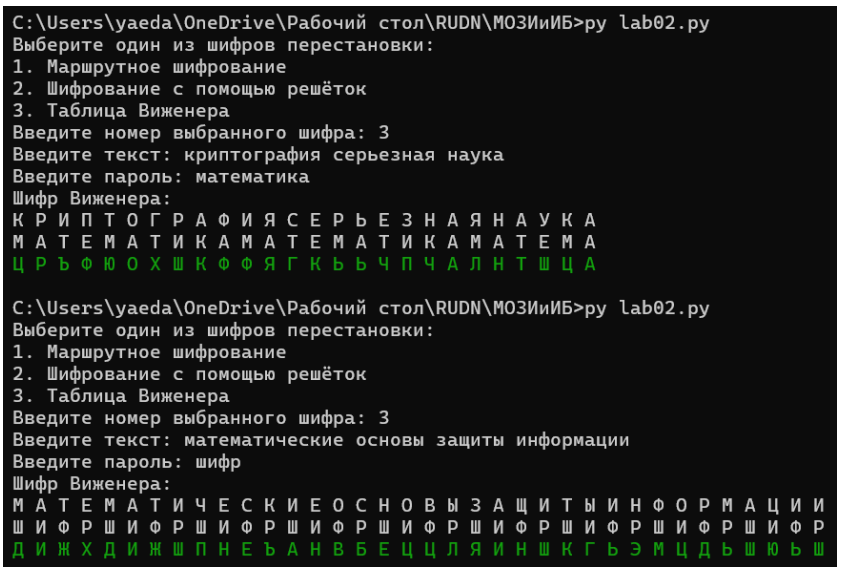

Figure 3.15: Проверка реализации шифра Виженера

4 Выводы

Я программно реализовал шифры перестановки.