УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

**Лабораторная работа №1.1**

Дисциплина «Информационная безопасность»

Вариант 3

**Выполнил: студент группы P34131**

*Кузнецов Максим Александрович*

**Преподаватель:**

*Маркина Татьяна Анатольевна*

Санкт-Петербург, 2023 г.

Цель работы

Изучение основных принципов шифрования информации, знакомство с широко известными алгоритмами шифрования, приобретение навыков их программной реализации.

Задание

Реализовать шифрование и дешифрацию файла с использованием метода биграмм. Ключевое слово вводится.

Листинг разработанной программы

encode.py

from tools import \*  
  
  
  
def encode(keyword, origin\_phrase, should\_extend=False, print\_steps=False, print\_table=False) -> str:  
 *"""  
 @:param keyword -- ключевое слово, по которому будет строиться матрица и шифроваться слово  
  
 @:param origin\_phrase -- оригинальное слово, которое будет зашифровано  
  
 @:param should\_extend -- False по-умолчанию, флаг, который отвечает за разделение биграммы из двух одинаковых букв  
  
 @:param print\_steps -- False по-умолчанию, флаг, который отвечает за показ пошагового конвертирования биграмм  
  
 @:param print\_table -- False по-умолчанию, флаг, который отвечает за отображение матрицы по ключевому слову  
 """* origin\_phrase = ''.join(re.findall('[А-ЯЁ .,]', origin\_phrase))  
 if should\_extend:  
 if len(origin\_phrase) > 1:  
 for i in range(0, len(origin\_phrase), 2):  
 if origin\_phrase[i] == origin\_phrase[i + 1]:  
 origin\_phrase = origin\_phrase[:i + 1] + 'Я' + origin\_phrase[i + 1:]  
  
 if len(origin\_phrase) % 2 == 1:  
 origin\_phrase = origin\_phrase + " "  
  
 print(f'Итоговая фраза: "{origin\_phrase}"')  
  
 enc\_phrase = ""  
 matrix = build\_table(keyword, print\_table)  
 for i in range(0, len(origin\_phrase), 2):  
 first\_y, first\_x = find\_position(matrix, origin\_phrase[i])  
 second\_y, second\_x = find\_position(matrix, origin\_phrase[i + 1])  
  
 if first\_x == second\_x:  
 enc\_phrase = enc\_phrase + matrix[0 if first\_y == 5 else first\_y + 1][first\_x]  
 enc\_phrase = enc\_phrase + matrix[0 if second\_y == 5 else second\_y + 1][second\_x]  
  
 elif first\_y == second\_y:  
 enc\_phrase = enc\_phrase + matrix[first\_y][0 if first\_x == 5 else first\_x + 1]  
 enc\_phrase = enc\_phrase + matrix[second\_y][0 if second\_x == 5 else second\_x + 1]  
  
 else:  
 enc\_phrase = enc\_phrase + matrix[first\_y][second\_x]  
 enc\_phrase = enc\_phrase + matrix[second\_y][first\_x]  
 if print\_steps:  
 print(f'{origin\_phrase[i]}{origin\_phrase[i + 1]}-->{enc\_phrase[i]}{enc\_phrase[i + 1]}')  
  
 print(f'Зашифрованная итоговая фраза: {enc\_phrase}')  
 return enc\_phrase

decode.py

from tools import \*  
  
  
def decode(keyword, origin\_phrase, print\_steps=False, print\_table=False) -> str:  
 *"""  
 @:param keyword -- ключевое слово, по которому будет строиться матрица и дешифроваться слово  
  
 @:param origin\_phrase -- оригинальное слово, которое будет дешифровано  
  
 @:param print\_steps -- False по-умолчанию, флаг, который отвечает за показ пошагового конвертирования биграмм  
  
 @:param print\_table -- False по-умолчанию, флаг, который отвечает за отображение матрицы по ключевому слову  
 """* temp = len(origin\_phrase)  
  
 origin\_phrase = ''.join(re.findall('[А-ЯЁ ,.]', origin\_phrase))  
 if temp != len(origin\_phrase) or temp % 2 == 1:  
 raise Exception(f'ОШИБКА: нечетное количество символом, либо недопустимые элементы')  
  
 print(f'Итоговая фраза: "{origin\_phrase}"')  
  
 dec\_phrase = ""  
 matrix = build\_table(keyword, print\_table)  
 for i in range(0, len(origin\_phrase), 2):  
 first\_y, first\_x = find\_position(matrix, origin\_phrase[i])  
 second\_y, second\_x = find\_position(matrix, origin\_phrase[i + 1])  
  
 if first\_x == second\_x:  
 dec\_phrase = dec\_phrase + matrix[5 if first\_y == 0 else first\_y - 1][first\_x]  
 dec\_phrase = dec\_phrase + matrix[5 if second\_y == 0 else second\_y - 1][second\_x]  
  
 elif first\_y == second\_y:  
 dec\_phrase = dec\_phrase + matrix[first\_y][5 if first\_x == 0 else first\_x - 1]  
 dec\_phrase = dec\_phrase + matrix[second\_y][5 if second\_x == 0 else second\_x - 1]  
  
 else:  
 dec\_phrase = dec\_phrase + matrix[first\_y][second\_x]  
 dec\_phrase = dec\_phrase + matrix[second\_y][first\_x]  
 if print\_steps:  
 print(f'{origin\_phrase[i]}{origin\_phrase[i + 1]}-->{dec\_phrase[i]}{dec\_phrase[i + 1]}')  
  
 print(f'Дешифрованная итоговая фраза: {dec\_phrase}')  
 return dec\_phrase

tools.py

import re  
  
  
def find\_position(square, letter) -> (int, int):  
 for y in range(0, 6):  
 for x in range(0, 6):  
 if square[y][x] == letter:  
 return y, x  
  
  
def build\_table(keyword, print\_table=False) -> [[], [], [], [], [], []]:  
 keyword = ''.join(re.findall('[А-ЯЁ ,.]', keyword))  
 print(f'Итоговое ключевое слово: "{keyword}"')  
  
 encryption\_square = [[], [], [], [], [], []]  
  
 keyword = ''.join(sorted(set(keyword), key=keyword.index))  
  
 total = keyword + "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ .,"  
  
 encryption\_word = ''.join(sorted(set(total), key=total.index))  
 if print\_table:  
 print("\nМатрица 6х6:")  
 for i in range(0, 6):  
 for j in range(0, 6):  
 encryption\_square[i].append(encryption\_word[j + i \* 6])  
 if print\_table:  
 print(encryption\_square[i])  
  
 return encryption\_square

main.py

from encode import \*  
from decode import \*  
  
while 1:  
 print("Введите ключевое слово (допуст. символы: [А-Я ,.]):")  
 keyword = input().upper()  
  
 print("Введите 1 -- для кодирования, 2 -- для декодирования:")  
 mode = input()  
 if mode == '1':  
 print("Фраза для шифрования:")  
 phrase = input().upper()  
 encode(keyword, phrase, print\_steps=True, print\_table=True)  
 elif mode == '2':  
 print("Фраза для дешифрования:")  
 phrase = input().upper()  
 decode(keyword, phrase, print\_steps=True, print\_table=True)  
 else:  
 print("Неверный ввод. Отмена операции.")

Вывод

В результате выполнения лабораторной работы я:

* Узнал о методе шифрования как «биграммный» (шифр Плейфера)
* Разработал программную реализацию алгоритма шифрования на языке Python