МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Отчет по лабораторной работе №2**

**ИССЛЕДОВАНИЕ КРИПТОГРАФИЧЕСКИХ**

**ШИФРОВ НА ОСНОВЕ ПОДСТАНОВКИ**

**(ЗАМЕНЫ) СИМВОЛОВ**

Выполнил:

Cтудент 3 курса 1 группы

Парибок И. А.

Вариант 5

Минск 2023

**Цель**: изучение и приобретение практических навыков разработки и использования приложений для реализации подстановочных шифров.

**Практическое задание:**

1. Разработать авторское приложение в соответствии с целью лабораторной работы. Приложение должно реализовывать следующие операции:

• выполнять зашифрование/расшифрование текстовых документов (объемом не менее 5 тысяч знаков), созданных на основе алфавита языка в соответствии с нижеследующей таблицей вариантов задания; при этом следует использовать шифры подстановки из третьего столбца данной таблицы (варианты задания представлены Рис. 1);

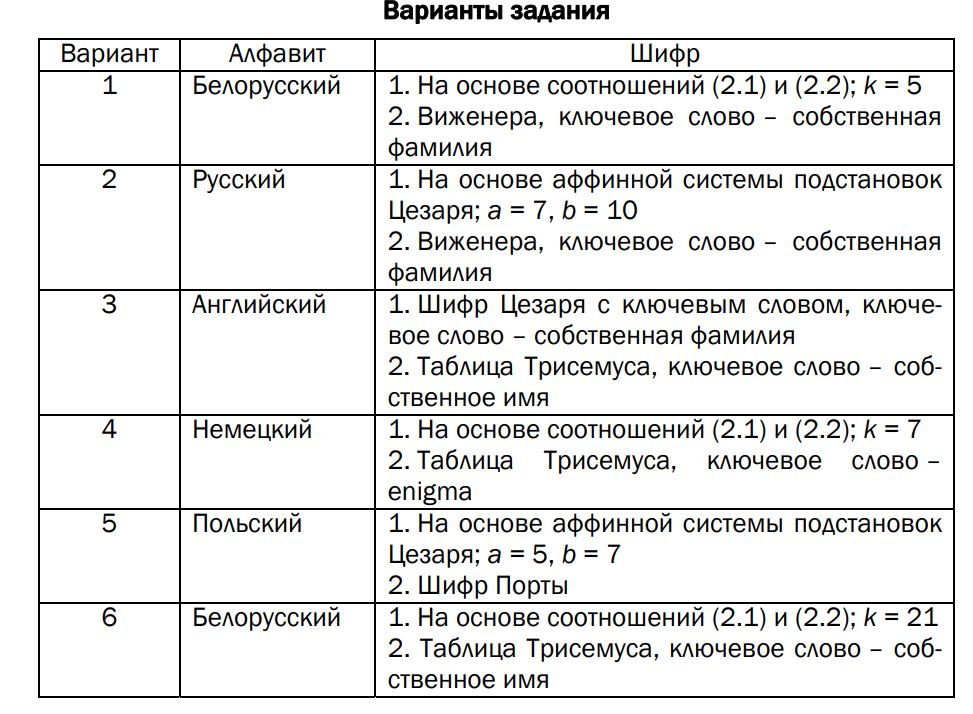


Рисунок 1 – Варианты заданий

• сформировать гистограммы частот появления символов для исходного и зашифрованного сообщений;

• оценить время выполнения операций зашифрования/расшифрования;

2. Результаты оформить в виде отчета по установленным правилам.

**Выполнение работы:**

Были реализованы шифры на основе аффинной системы подстановок Цезаря и шифра Порты. На рисунках 2-3 представлены результаты выполнения приложения, а именно - шифрования и дешифрования текста на основе аффинной системы подстановок Цезаря. На рисунке 4 показана скорость выполнения программного кода шифрования и дешифрования. Исходя из полученных данных, можно сделать вывод, что в данной реализации дешифрование полученных данных более затратно по времени, чем шифрование.

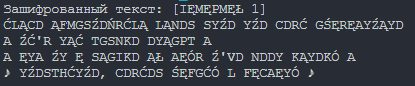


Рисунок 2 – Результат шифрования текста на основе аффинной системы подстановок Цезаря

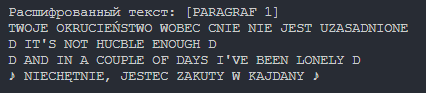


Рисунок 3 – Результат дешифрования текста на основе аффинной системы подстановок Цезаря



Рисунок 4 – Скорость выполнения шифрования и дешифрования текста основе аффинной системы подстановок Цезаря

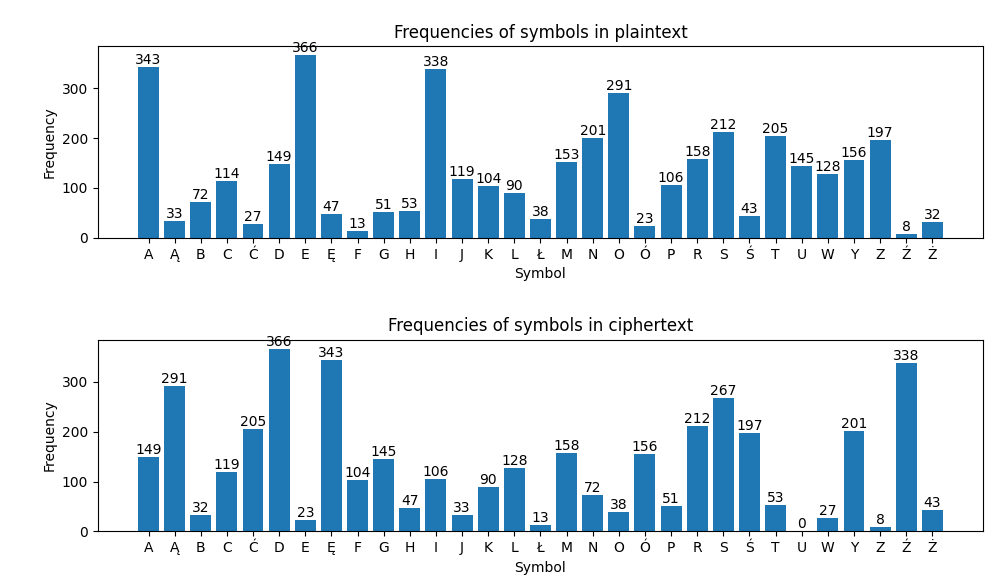


Рисунок 5 – Гистограмма использования символов до и после шифрования на основе аффинной системы подстановок Цезаря

Алгоритмы шифрования и дешифрования текста основе аффинной системы подстановок Цезаря представлены в листинге 1. В качестве значений функции принимают ключи а, б сообщения для шифрования/дешифрования и используемый алфавит. Полный листинг программы с использованием шифрования и дешифрования текста основе аффинной системы подстановок Цезаря представлен в приложение А.

*#Алгоритм шифрования*

def affine\_cipher\_encrypt(plaintext, a, b, alphabet):

    ciphertext = ""

    for char in plaintext:

        if char in alphabet:

            char\_index = alphabet.index(char)

            new\_index = (a \* char\_index + b) % len(alphabet)

            ciphertext += alphabet[new\_index]

        else:

            ciphertext += char

    return ciphertext

*#Алгоритм дешифрования*

def affine\_cipher\_decrypt(ciphertext, a, b, alphabet):

    plaintext = ""

    for char in ciphertext:

        if char in alphabet:

            char\_index = alphabet.index(char)

*# Вычисляем обратный элемент a в кольце вычетов по модулю len(alphabet)*

            inverse\_a = pow(a, -1, len(alphabet))

            new\_index = (inverse\_a \* (char\_index - b)) % len(alphabet)

            plaintext += alphabet[new\_index]

        else:

            plaintext += char

    return plaintext

Листинг 1 – Алгоритмы шифрования и дешифрования текста основе аффинной системы подстановок Цезаря

Аффинная система шифрования является одним из видов шифрования подстановки, которая использует линейную функцию для перестановки символов в сообщении. Шифр Цезаря является частным случаем аффинной системы, где каждый символ заменяется другим символом, который находится на фиксированном расстоянии в алфавите.

Для создания аффинной системы шифрования Цезаря с ключами a=5 и b=7 для польского языка, необходимо сначала создать алфавит из всех символов, которые будут использоваться в сообщении. Алфавит польского языка состоит из 32 символов, поэтому алфавит для шифрования и дешифрования будет содержать 32 символа.

Символы алфавита можно представить в виде чисел от 0 до 31. Для шифрования и дешифрования текста используется следующая формула:

* шифрование: c = (a \* p + b) % m
* дешифрование: p = (a\_inv \* (c - b)) % m

где:

* p - оригинальный символ (plaintext)
* c - зашифрованный символ (ciphertext)
* m - количество символов в алфавите (32)
* a и b - ключи шифрования (a=5, b=7)
* a\_inv - обратное значение a по модулю m

На рисунках 6-7 представлены результаты выполнения приложения, а именно - шифрование и дешифрование с использованием шифра Порты. В качестве ключа использовалось слово "ilua". На рисунке 8 показана скорость выполнения программного кода шифрования и дешифрования. Исходя из полученных данных, можно сделать вывод, что в данной реализации дешифрование полученных данных занимает немного больше времени, чем шифрование.

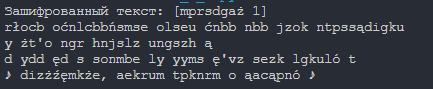


Рисунок 6 – Результат дешифрования текста на основе шифра Порты

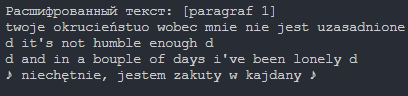


Рисунок 7 – Результат дешифрования текста на основе шифра Порты



Рисунок 8 – Скорость выполнения шифрования и дешифрования текста на основе шифра Порты

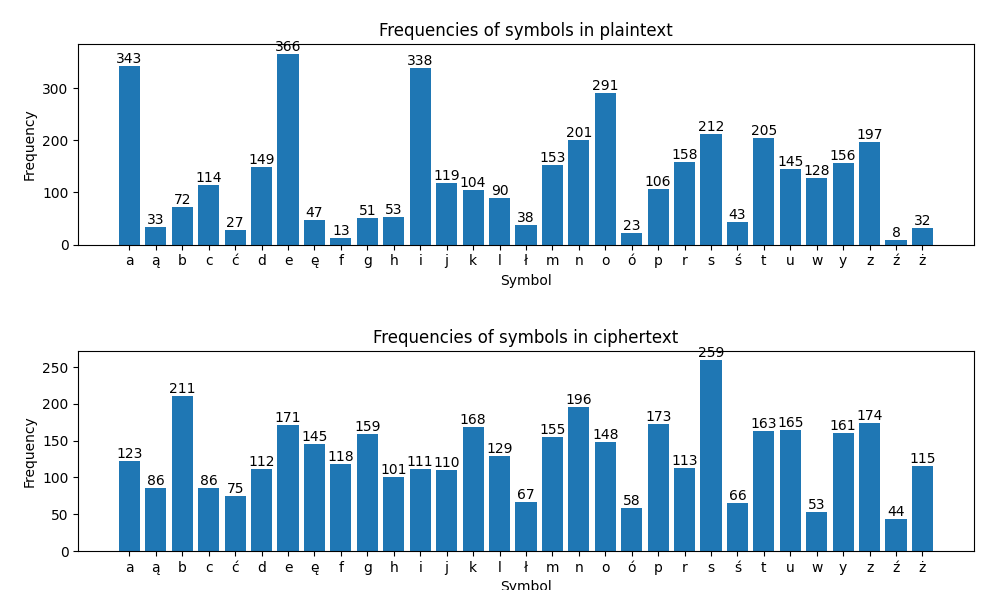


Рисунок 9 – Гистограмма использования символов до и после шифрования на основе шифра Порты

Алгоритмы шифрования и дешифрования текста на основе шифра Порты представлены в листинге 2. В качестве значений функции принимают ключ, сообщения для шифрования/дешифрования и используемый алфавит. Полный листинг программы с использованием шифрования и дешифрования текста основе шифра Порты изложен в приложение Б.

def porta\_cipher\_encrypt(plaintext, key, alphabet):

    key = key.lower()

    key\_length = len(key)

    key\_pos = [alphabet.index(char) for char in key]

    vigenere\_table = generate\_vigenere\_table(alphabet)

    ciphertext = ''

    for i, char in enumerate(plaintext):

        if char in alphabet:

            col = alphabet.index(char)

            row = vigenere\_table[key[i % key\_length]][key\_pos[i % key\_length]]

            ciphertext += vigenere\_table[row][col]

        else:

            ciphertext += char

    return ciphertext

def porta\_cipher\_decrypt(ciphertext, key, alphabet):

    key = key.lower()

    key\_length = len(key)

    key\_pos = [alphabet.index(char) for char in key]

    vigenere\_table = generate\_vigenere\_table(alphabet)

    plaintext = ''

    for i, char in enumerate(ciphertext):

        if char in alphabet:

            row = vigenere\_table[key[i % key\_length]][key\_pos[i % key\_length]]

            col = vigenere\_table[row].index(char)

            plaintext += alphabet[col]

        else:

            plaintext += char

    return plaintext

Листинг 2 – Алгоритмы шифрования и дешифрования текста основе шифра Порты

В ходе работы использовался Шифр Порты с ключом Ilua. Этот шифр основан на шифре Порты, который использует табличную перестановку символов.

Для шифрования сообщения с помощью шифра Порты с ключом Ilua, нужно выполнить следующие шаги:

* Записать сообщение, которое необходимо зашифровать, в виде строки символов на польском языке.
* Определить ключ Ilua - это случайная последовательность чисел от 1 до длины сообщения. Например, если сообщение состоит из 10 символов, то ключ Ilua может выглядеть как "4 1 9 7 2 10 8 6 5 3".
* Разбить сообщение на блоки по длине ключа Ilua. Например, если ключ Ilua имеет длину 10, то сообщение будет разделено на 10 блоков по одному символу.
* Расположить символы в каждом блоке в соответствии с порядком, определенным ключом Ilua.
* Объединить блоки, чтобы получить зашифрованное сообщение.
* Для дешифрования зашифрованного сообщения, нужно выполнить обратные действия:
* Записать зашифрованное сообщение в виде строки символов на польском языке.
* Определить ключ Ilua - это тот же самый ключ, который использовался для шифрования сообщения.
* Разбить зашифрованное сообщение на блоки по длине ключа Ilua.
* Расположить символы в каждом блоке в соответствии с порядком, определенным ключом Ilua.
* Объединить блоки, чтобы получить исходное сообщение.

Шифр Порты с ключом Ilua является достаточно простым методом шифрования, который может быть использован для защиты конфиденциальной информации в случаях, когда не требуется высокая степень безопасности. Однако, его использование не рекомендуется в критических ситуациях, так как он может быть легко взломан с помощью методов криптоанализа.

**Вывод:** в результате выполненной работы были реализованы два метода шифрования: на основе аффинной системы подстановок Цезаря и шифра Порты. В качестве шифруемых данных использовался файл с более чем 5000 символами. Программная реализация выводит гистограмму появления символов до и после шифрования сообщения. Также были получены данные о скорости работы шифрования.

**Приложение А**

import time

from graph import prrint\_freg,count\_frequencies

*#Алгоритм шифрования*

def affine\_cipher\_encrypt(plaintext, a, b, alphabet):

    ciphertext = ""

    for char in plaintext:

        if char in alphabet:

            char\_index = alphabet.index(char)

            new\_index = (a \* char\_index + b) % len(alphabet)

            ciphertext += alphabet[new\_index]

        else:

            ciphertext += char

    return ciphertext

*#Алгоритм дешифрования*

def affine\_cipher\_decrypt(ciphertext, a, b, alphabet):

    plaintext = ""

    for char in ciphertext:

        if char in alphabet:

            char\_index = alphabet.index(char)

*# Вычисляем обратный элемент a в кольце вычетов по модулю len(alphabet)*

            inverse\_a = pow(a, -1, len(alphabet))

            new\_index = (inverse\_a \* (char\_index - b)) % len(alphabet)

            plaintext += alphabet[new\_index]

        else:

            plaintext += char

    return plaintext

alphabet = "AĄBCĆDEĘFGHIJKLŁMNOÓPRSSŚTUWYZŹŻ"

a = 5

b = 7

with open('Text.txt', 'r', encoding='utf-8') as f:

    text = f.read().upper()

start\_encrypt = time.time()

ciphertext = affine\_cipher\_encrypt(text, a, b, alphabet)

end\_encrypt = time.time()

print("Зашифрованный текст:", ciphertext)

start\_decrypt = time.time()

decrypted\_plaintext = affine\_cipher\_decrypt(ciphertext, a, b, alphabet)

end\_decrypt = time.time()

print('-'\*55)

print("Расшифрованный текст:",decrypted\_plaintext)

print("Encryption time:", end\_encrypt - start\_encrypt, "seconds")

print("Decryption time:", end\_decrypt - start\_decrypt, "seconds")

*# подсчет частот символов*

plain\_freq = count\_frequencies(text, alphabet)

cipher\_freq = count\_frequencies(ciphertext, alphabet)

prrint\_freg(plain\_freq,cipher\_freq)

**Приложение Б**

import time

from graph import prrint\_freg,count\_frequencies

def generate\_vigenere\_table(alphabet):

*# функция для генерации таблицы Виженера на основе заданного алфавита*

    table = {}

    n = len(alphabet)

    for i, char in enumerate(alphabet):

        row = []

        for j in range(n):

            row.append(alphabet[(i+j) % n])

        table[char] = row

    return table

def porta\_cipher\_encrypt(plaintext, key, alphabet):

*# функция для шифрования текста шифром Порты*

    key = key.lower()

    key\_length = len(key)

    key\_pos = [alphabet.index(char) for char in key]

    vigenere\_table = generate\_vigenere\_table(alphabet)

    ciphertext = ''

    for i, char in enumerate(plaintext):

        if char in alphabet:

            col = alphabet.index(char)

            row = vigenere\_table[key[i % key\_length]][key\_pos[i % key\_length]]

            ciphertext += vigenere\_table[row][col]

        else:

            ciphertext += char

    return ciphertext

def porta\_cipher\_decrypt(ciphertext, key, alphabet):

*# функция для дешифрования текста, зашифрованного шифром Порты*

    key = key.lower()

    key\_length = len(key)

    key\_pos = [alphabet.index(char) for char in key]

    vigenere\_table = generate\_vigenere\_table(alphabet)

    plaintext = ''

    for i, char in enumerate(ciphertext):

        if char in alphabet:

            row = vigenere\_table[key[i % key\_length]][key\_pos[i % key\_length]]

            col = vigenere\_table[row].index(char)

            plaintext += alphabet[col]

        else:

            plaintext += char

    return plaintext

alphabet = "aąbcćdeęfghijklłmnoóprssśtuwyzźż"

key = "ilua"

with open('Text.txt', 'r', encoding='utf-8') as f:

    text = f.read().lower()

*# Зашифровка*

start\_encrypt = time.time()

ciphertext = porta\_cipher\_encrypt(text, key, alphabet)

end\_encrypt = time.time()

print("Зашифрованный текст:", ciphertext)

*# Расшифровка*

start\_decrypt = time.time()

decrypted\_text = porta\_cipher\_decrypt(ciphertext, key, alphabet)

end\_decrypt = time.time()

print('-'\*55)

print("Расшифрованный текст:", decrypted\_text)

print("Encryption time:", end\_encrypt - start\_encrypt, "seconds")

print("Decryption time:", end\_decrypt - start\_decrypt, "seconds")

*# подсчет частот символов*

plain\_freq = count\_frequencies(text, alphabet)

cipher\_freq = count\_frequencies(ciphertext, alphabet)

prrint\_freg(plain\_freq,cipher\_freq)