Министерство науки и высшего образования Российской Федерации

Новосибирский Государственный технический университет

Кафедра автоматизированных систем управления



ОТЧЁТ

по лабораторной работе №1

по дисциплине «Защита информации»

«Методы шифрования и основные понятия криптографии»

Выполнили: Проверил:

Студенты гр. «АВТ-812», «АВТФ» «к.т.н., доцент»

Березин Дмитрий Качальский В.Г.

Бородина Алина

Глинин Евгений

«\_\_\_» \_\_\_\_\_\_ 20\_\_г. «\_\_\_» \_\_\_\_\_\_ 20\_\_ г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (подпись)

г. Новосибирск

2021 г.

Содержание

[1 Цель работы 3](#_Toc85048237)

[2 Постановка задачи 3](#_Toc85048238)

[4 Преобразования исходного текста в криптограмму и обратно 5](#_Toc85048239)

[5 Примеры шифрования-дешифрования текста 6](#_Toc85048240)

[6 Выводы 8](#_Toc85048241)

[7 Листинг 9](#_Toc85048242)

# 1 Цель работы

Изучить методы шифрования и основные понятия криптографии. Реализовать программно шифр Виженера.

# 2 Постановка задачи

* Построить блок схему шифрующего и дешифрующего устройства.
* Записать математические уравнения прямого и обратного преобразования исходного текста в криптограмму и обратно, используя выражения и уравнения (1) - (6).
* Включить в алфавит буквы кириллицы и вспомогательные знаки. Провести 3-5 примеров шифрования-дешифрования текста.
* Написать программу шифрования-дешифрования на любом доступном Вам языке программирования c формированием загрузочного .exe файла
  + с клавиатуры произвольного текста;
  + с чтением и записью в текстовый файл.

**3 Блок схема шифрующего и дешифрующего устройства**

На рисунках 1-2 представлены алгоритмы шифрования и дешифрования.

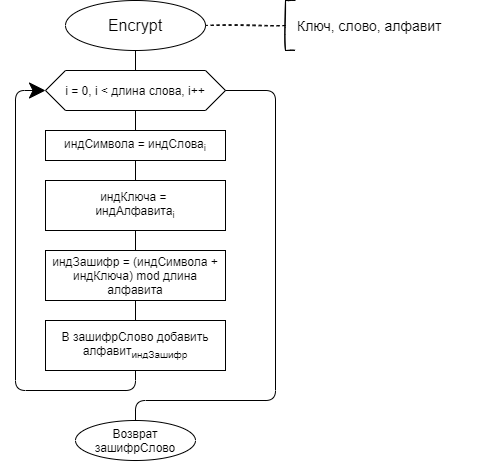


Рисунок 1 – Алгоритм шифрования Виженера

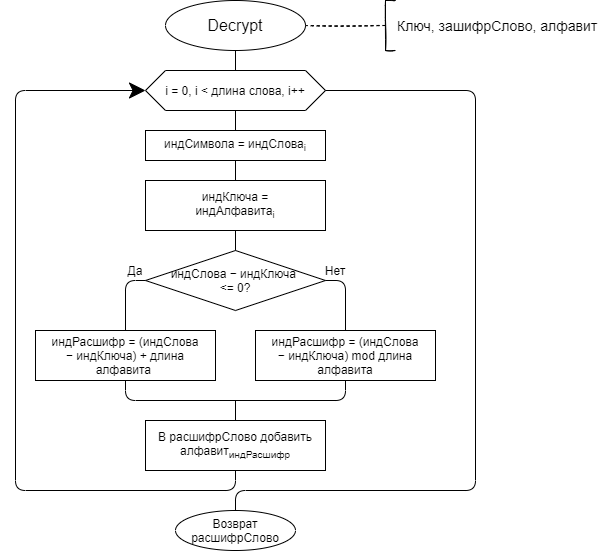


Рисунок 2 – Алгоритм дешифрования Виженера

# 4 Преобразования исходного текста в криптограмму и обратно

На рисунке 3 представлена модель криптографической системы.

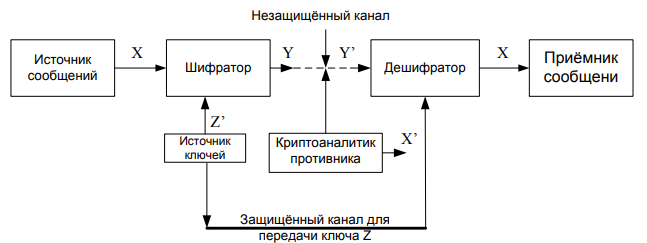


Рисунок 3 – Модель криптографической системы

Источник сообщений порождает открытый текст

X={x1,x2,… xi}, i – число символов открытого текста. (1)

Источник ключей генерирует k знаков ключа - символов некоторого конечного алфавита A. Шифратор преобразует открытый текст X в шифр текст:

Y={y1,y2, … yj}, i – число символов закрытого текста. (2)

Последнее преобразование записывается в виде:

Y=E(X;Z), Е – прямое преобразование(шифрование). (3)

Z={z1,z2,… zk} - секретный ключ, k – размерность ключа. (4)

В нашем случае суммирование по модулю A общего числа допустимых символов в шифруемом алфавите.

Y = (X + Z) mod A

Дешифратор, получив шифр текст, выполняет обратное преобразование:

X=D(Y;Z), где (5)

D=E-1, - обратное преобразование(дешифрование). (6)

# 5 Примеры шифрования-дешифрования текста

Алфавит: а б в г д е ж з и к л м н о п р с т у ф х ц ч ш щ э ю я ъ ь

На рисунках 4-5 представлены результаты работы программы.

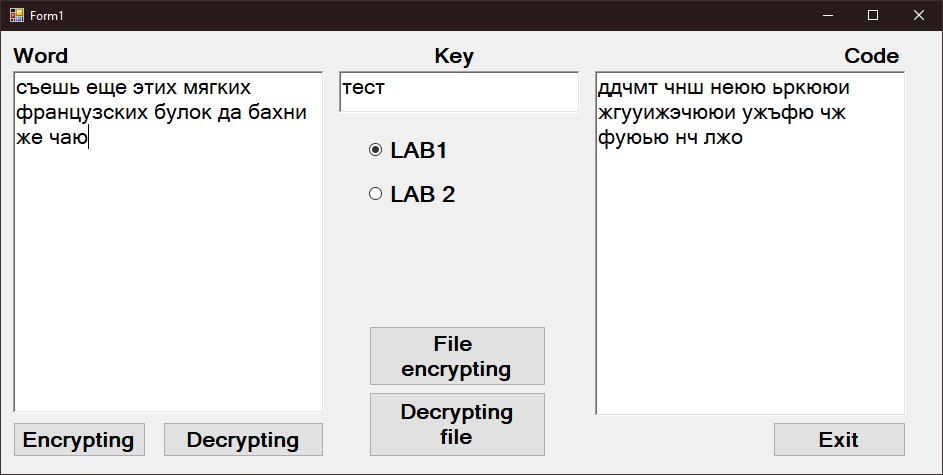


Рисунок 4 – Модель криптографической системы

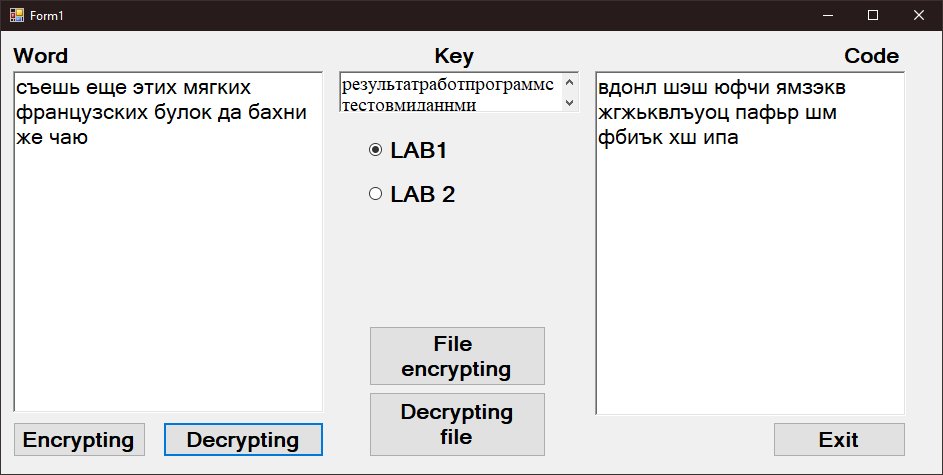


Рисунок 5 – Модель криптографической системы

На рисунке 6 представлен текстовый файл с исходной информацией.

На рисунке 7 представлен зашифрованный файл со следующим ключом:

результатработпрограммстестовмиданнми .

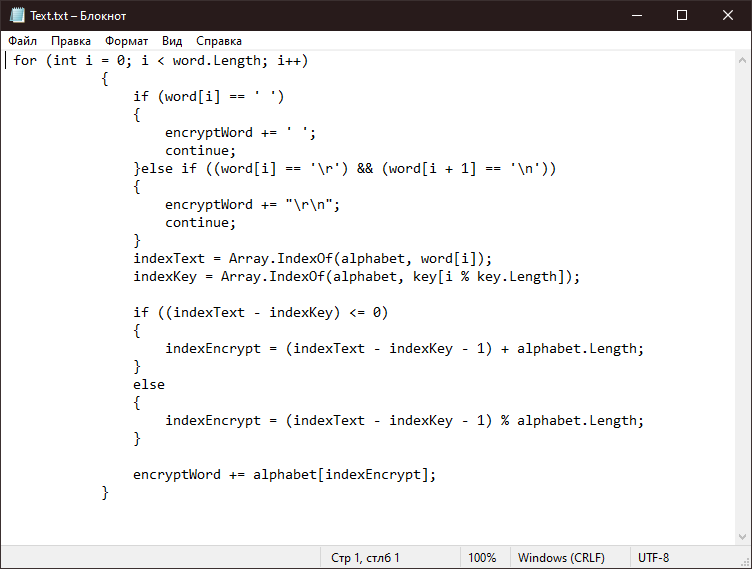


Рисунок 6 – Текстовый файл с исходной информацией

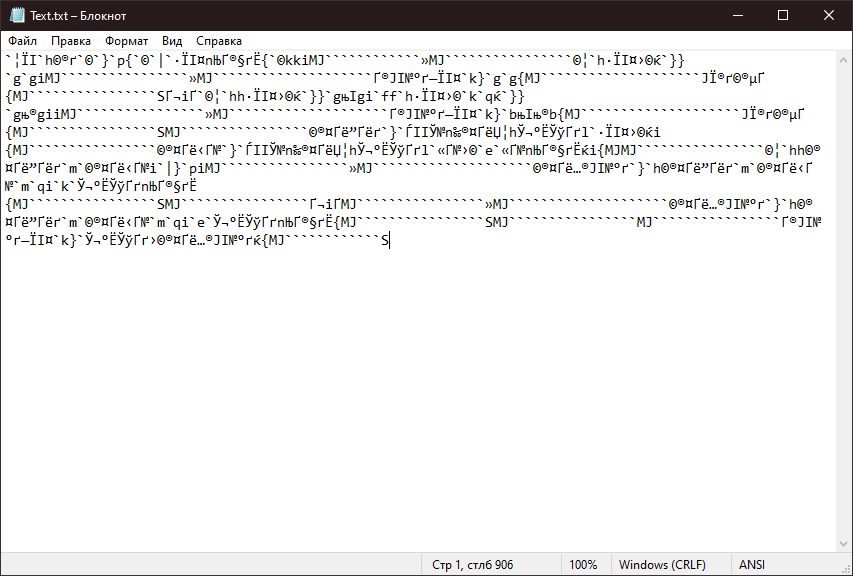


Рисунок 7 – Текстовый файл с зашифрованной информацией

# 6 Выводы

Изучили методы шифрования и основные понятия криптографии. Реализовали программно шифр Виженера на языке программирования C#.

# 7 Листинг

Encryption.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp

{

public class Encryption

{

protected char[] alphabet = { 'а','б', 'в', 'г',

'д','е', 'ж', 'з',

'и', 'к', 'л', 'м',

'н','о', 'п', 'р',

'с', 'т', 'у', 'ф',

'х', 'ц', 'ч', 'ш',

'щ', 'э', 'ю', 'я',

'ъ', 'ь' };

protected string word;

protected string key;

protected byte[] bt;

public Encryption()

{

this.word = "EMPTY";

this.key = "EMPTY";

}

public Encryption(string word, string key)

{

this.word = word;

this.key = key;

}

public Encryption(byte[] bt, string key)

{

this.bt = bt;

this.key = key;

}

public string Encrypt()

{

int indexEncrypt = 0;

int indexWord;

int indexKey;

string encryptWord = "";

for (int i = 0; i < word.Length; i++)

{

if(word[i] == ' ')

{

encryptWord += ' ';

continue;

} else if ((word[i] == '\r') && (word[i+1] == '\n'))

{

encryptWord += "\r\n";

continue;

}

indexWord = Array.IndexOf(alphabet, word[i]);

indexKey = Array.IndexOf(alphabet, key[i % key.Length]);

indexEncrypt = (indexWord + indexKey + 1) % alphabet.Length ;

encryptWord += alphabet[indexEncrypt];

}

return encryptWord;

}

public string EncryptTwo()

{

int indexEncrypt = 0;

int indexWord;

int indexKey;

string encryptWord = "";

for (int i = 0; i < word.Length; i++)

{

if (word[i] == ' ')

{

encryptWord += ' ';

continue;

}

else if ((word[i] == '\r') && (word[i + 1] == '\n'))

{

encryptWord += "\r\n";

continue;

}

indexWord = Array.IndexOf(alphabet, word[i]);

indexKey = Array.IndexOf(alphabet, key[i % key.Length]);

indexEncrypt = (indexWord + indexKey) % alphabet.Length;

encryptWord += alphabet[indexEncrypt];

}

return encryptWord;

}

public string Decrypt()

{

int indexDecrypt = 0;

int indexText;

int indexKey;

string decryptWord = "";

for (int i = 0; i < word.Length; i++)

{

if (word[i] == ' ')

{

decryptWord += ' ';

continue;

}else if ((word[i] == '\r') && (word[i + 1] == '\n'))

{

decryptWord += "\r\n";

continue;

}

indexText = Array.IndexOf(alphabet, word[i]);

indexKey = Array.IndexOf(alphabet, key[i % key.Length]);

if ((indexText - indexKey) <= 0)

{

indexDecrypt = (indexText - indexKey - 1) + alphabet.Length;

}

else

{

indexDecrypt = (indexText - indexKey - 1) % alphabet.Length;

}

decryptWord += alphabet[indexDecrypt];

}

return decryptWord;

}

public string DecryptTwo()

{

int indexDecrypt = 0;

int indexText;

int indexKey;

string decryptWord = "";

for (int i = 0; i < word.Length; i++)

{

if (word[i] == ' ')

{

decryptWord += ' ';

continue;

}

else if ((word[i] == '\r') && (word[i + 1] == '\n'))

{

decryptWord += "\r\n";

continue;

}

indexText = Array.IndexOf(alphabet, word[i]);

indexKey = Array.IndexOf(alphabet, key[i % key.Length]);

if ((indexText - indexKey) <= 0)

{

indexDecrypt = (indexText - indexKey) + alphabet.Length;

}

else

{

indexDecrypt = (indexText - indexKey) % alphabet.Length;

}

decryptWord += alphabet[indexDecrypt];

}

return decryptWord;

}

public byte[] fileEncrypt()

{

byte[] keyInASCII = Encoding.ASCII.GetBytes(key);

byte[] indexEncrypt = new byte[bt.Length];

for (int i = 0; i < bt.Length; i++)

indexEncrypt[i] = Convert.ToByte((bt[i] + keyInASCII[i % keyInASCII.Length] + 1) % (Byte.MaxValue + 1));

return indexEncrypt;

}

public byte[] fileDecrypt()

{

byte[] keyInASCII = Encoding.ASCII.GetBytes(key);

byte[] indexDecrypt = new byte[bt.Length];

int indexByte;

for (int i = 0; i < bt.Length; i++)

{

if ((bt[i] - keyInASCII[i % keyInASCII.Length] - 1) < 0)

{

indexByte = bt[i] - keyInASCII[i % keyInASCII.Length] - 1;

indexDecrypt[i] = Convert.ToByte(indexByte + (Byte.MaxValue + 1));

}

else

{

indexByte = bt[i] - keyInASCII[i % keyInASCII.Length] - 1;

indexDecrypt[i] = Convert.ToByte(indexByte % (Byte.MaxValue + 1));

}

}

return indexDecrypt;

}

public byte[] fileEncryptTwo()

{

byte[] keyInASCII = Encoding.ASCII.GetBytes(key);

byte[] indexEncrypt = new byte[bt.Length];

for (int i = 0; i < bt.Length; i++)

indexEncrypt[i] = Convert.ToByte((bt[i] + keyInASCII[i % keyInASCII.Length]) % (Byte.MaxValue + 1));

return indexEncrypt;

}

public byte[] fileDecryptTwo()

{

byte[] keyInASCII = Encoding.ASCII.GetBytes(key);

byte[] indexDecrypt = new byte[bt.Length];

int indexByte;

for (int i = 0; i < bt.Length; i++)

{

if ((bt[i] - keyInASCII[i % keyInASCII.Length]) < 0)

{

indexByte = bt[i] - keyInASCII[i % keyInASCII.Length];

indexDecrypt[i] = Convert.ToByte(indexByte + (Byte.MaxValue + 1));

}

else

{

indexByte = bt[i] - keyInASCII[i % keyInASCII.Length];

indexDecrypt[i] = Convert.ToByte(indexByte % (Byte.MaxValue + 1));

}

}

return indexDecrypt;

}

}

}