Учреждение образования

«Белорусский государственный технологический университет»

**Отчет по лабораторной работе №3**

по дисциплине: «Защита информации и надежность информационных систем»

Исполнитель:

Воликов Д. А., ФИТ 4-7

Руководитель:

Асс. Савельева М. Г.

Минск 2024

# Лабораторная работа №3. Исследование криптографических шифров на основе перестановки символов

**Цель**: изучение и приобретение практических навыков разработки и использования приложений для реализации перестановочных шифров (работа рассчитана на 4 часа аудиторных занятий).

**Задачи**:

1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости перестановочных шифров.
2. Разработать приложение для реализации указанных преподавателем методов перестановочного зашифрования/расшифрования.
3. Выполнить исследование криптостойкости шифров на основе статистических данных о частотах появления символов в исходном и зашифрованном сообщениях.
4. Оценить скорость зашифрования/расшифрования реализованных способов шифров.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

## Теоретические сведения

Сущность перестановочного шифрования состоит в том, что исходный текст (*М*) и зашифрованный текст (*С*) основаны на использовании одного и того же алфавита, а тайной или ключевой информацией является алгоритм перестановки.

Шифры перестановки относятся к классу симметричных. Элементами текста могут быть отдельные символы (самый распространенный случай), пары, тройки букв и т. д.

Классическими примерами перестановочных шифров являются анаграммы. Анаграмма – литературный прием, состоящий в перестановке букв (или звуков), что в результате дает другое слово или словосочетание, например: проездной – подрезной, листовка – вокалист, апельсин – спаниель.

В классической криптографии шифры перестановки делятся на два подкласса:

* шифры простой, или одинарной, перестановки – при зашифровании символы открытого текста *М*i перемещаются с исходных позиций в новые (в шифртексте *С*i) один раз;
* шифры сложной, или множественной, перестановки – при зашифровании символы открытого текста *М*i перемещаются с исходных позиций в новые (в шифртексте *С*i) несколько раз.

Среди шифров одинарной перестановки иногда выделяют шифры простой перестановки (или перестановки без ключа). Символы открытого текста *М*i перемешиваются по каким-либо правилам. Формально каждое из таких правил может рассматриваться в качестве ключа.

Простейшим примером является запись открытого текста в обратной последовательности. Так, если *М*i = «шифр перестановки», то *С*i = «иквонатсереп рфиш». Если переставляются в соответствующем порядке пары букв, то *С*i = «киованстрепе фрши». При более длинных сообщениях можно таким же образом перемещать целые слова или блоки слов.

Подобную перестановку можно трактовать как транспозицию.

В общем случае для использования шифров одинарной перестановки используется таблица, состоящая из двух строк: в первой строке записываются буквы, во второй – цифры *J*. Строки состоят из *n* столбцов. Буквы составляют шифруемое сообщение. Цифры *J* = *j*1, *j*2, …, *j*n, где *j*1 – номер позиции в зашифрованном сообщении первого символа открытого текста, *j*2 – номер позиции в зашифрованном сообщении второго символа открытого текста и т. д. Таким образом, порядок следования цифр определяется используемым правилом (ключом) перестановки символов открытого текста для получения шифрограммы.

Основой современных шифров типа маршрутной перестановки является геометрическая фигура, обычно прямоугольник или прямоугольная матрица. В ячейки этой фигуры по определенному маршруту (слева направо, сверху вниз или каким-либо иным образом) записывается открытый текст.

Для зашифрования и расшифрования необходимо было иметь абсолютно одинаковые жезлы. На такой предмет наматывалась пергаментная лента. Далее на эту ленту построчно наносился текст. Для расшифрования ленту с передаваемым сообщением нужно было намотать так же, как и при нанесении открытого текста.

Уже упоминавшуюся маршрутную перестановку можно усложнить и считывать не по столбцам, а по спирали (рис. 3.1, а), зигзагом (рис. 3.1, б), змейкой (рис. 3.1, в) или каким-то другим способом. Такие способы шифрования несколько усложняют процесс, однако усиливают криптостойкость шифра.

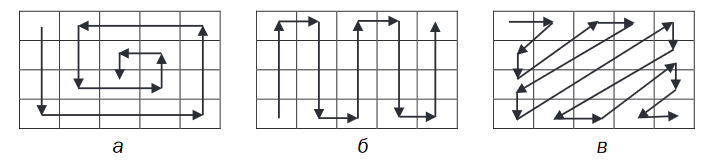


Рисунок 3.1 – Графическое представление методов маршрутной перестановки:  
а – по спирали; б – зигзагом; в – змейкой

Не менее занимательным и не менее сложным является организация маршрутов на основе «магических квадратов» – квадратных матриц со вписанными в каждую клетку неповторяющимися последовательными числами от 1, сумма которых по каждому столбцу, каждой строке и каждой диагонали дает одно и то же число.

Особенностью шифров множественной перестановки является минимум двукратная перестановка символов шифруемого сообщения. В простейшем случае это может задаваться перемешиванием не только столбцов, но и строк. Таким образом, этот случай соответствует использованию двух основных ключей: длина одного из них равна числу столбцов, другого – числу строк. К ключевой информацию мы можем относить также способы вписывания сообщения и считывания отдельных символов из текущего столбца матрицы.

Возьмём для примера слова «крипто» и «слово» как ключи. Необходимо зашифровать сообщение *М*i = «шифр вертикальной перестановки» (*n* = 30).

Строим основную таблицу 5×6 (рис. 3.2), в которую по строкам будет записано исходное сообщение.

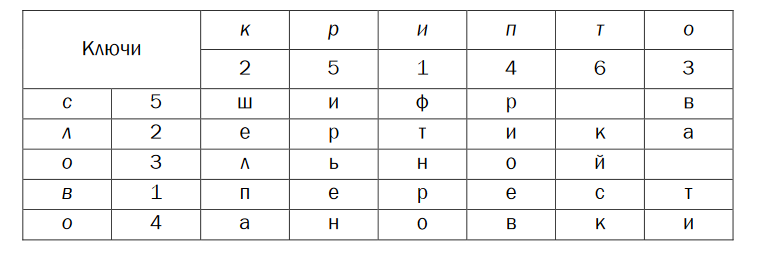


Рисунок 3.2 – Таблица с сообщением и ключами

Отсортируем последовательно строки в соответствии с ключом «слово» (рис. 3.3).

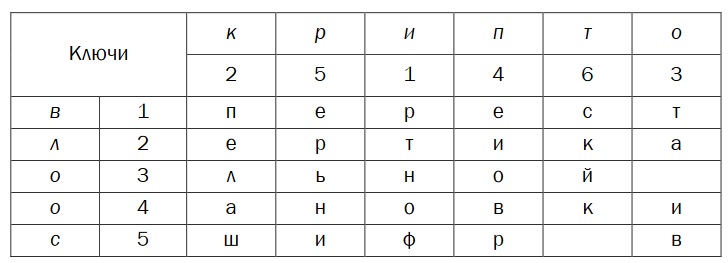


Рисунок 3.3 – Отсортированная таблица по строкам в соответствии с ключевым словом «слово»

И столбцы – в соответствии с ключевым словом «крипто» (рис. 3.4).

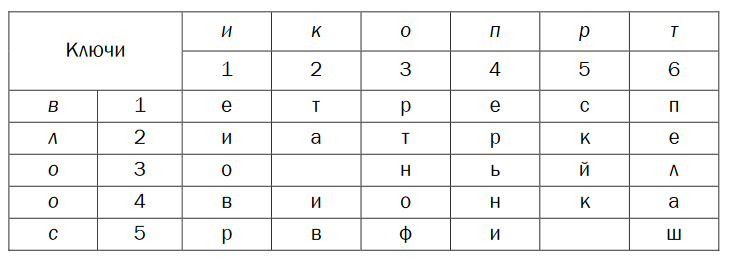


Рисунок 3.4 – Отсортированная таблица по столбцам в соответствии с ключевым словом «крипто»

Получим итоговую шифрограмму *С*i = «еиоврта ивртноферьнискйк пелаш».

Шифры гаммирования рассматриваются как самостоятельный класс. Такие шифры схожи с перестановочными тем, что в обоих случаях можно использовать табличное представление выполняемых операций на основе ключей. Вместе с тем шифры гаммирования имеют много общего с подстановочными шифрами, поскольку на самом деле при зашифровании происходит подмена одних символов другими.

## 1.2 Практическое задание

**Вариант 2**. По условию язык алфавита – русский. Использовать следующие шифры:

* Маршрутная перестановка (маршрут – по спирали; параметры таблицы – по указанию преподавателя).
* Множественная перестановка, ключевые слова – собственные имя и фамилия.

**Задание 1**. Разработать авторское приложение в соответствии с целью лабораторной работы. Приложение должно выполнять зашифрование/расшифрование текстовых документов (объемом не менее 500 знаков), созданных на основе алфавита языка в соответствии с вариантом.

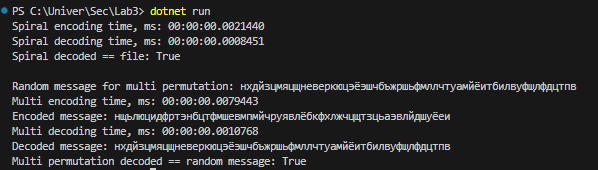


Рисунок 3.5 – Результат работы программы

Программа выводит время зашифрования/расшифрования текстового файла, размером в 642 символа. А также можно увидеть результат успешного декодирования зашифрованного текста.

**Задание 2**. Сформировать гистограммы частот появления символов для исходного и зашифрованных сообщений.

Гистограмма частот появления символов для исходного текста предоставлена на рисунке 3.6.

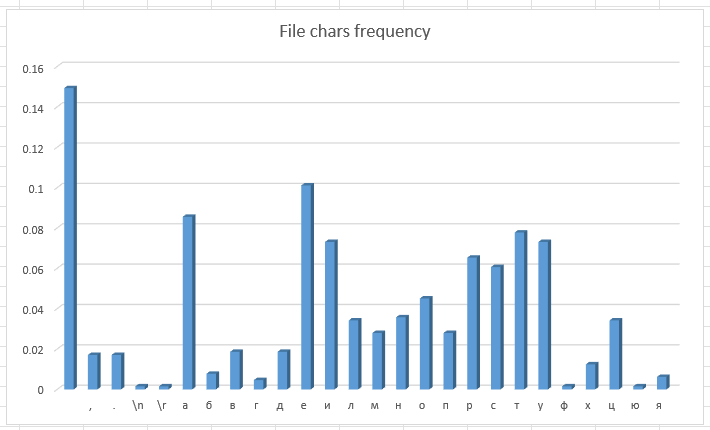


Рисунок 3.6 – Гистограмма частот появления символов для исходного сообщения

Гистограмма частот появления символов для зашифрованного текста методом маршрутной перестановки (по спирали) предоставлена на рисунке 3.7.

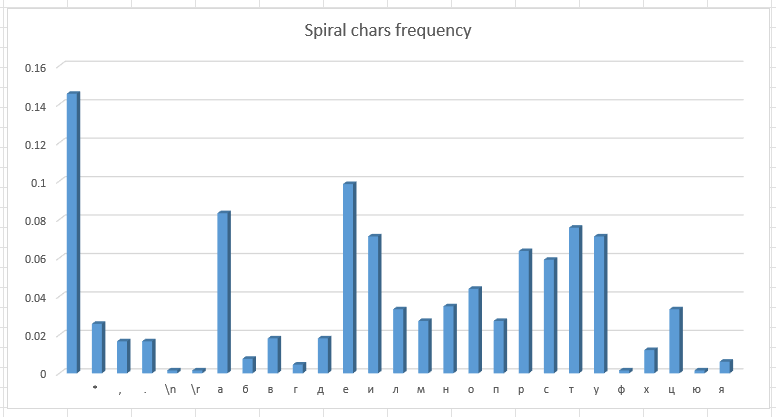


Рисунок 3.7 – Гистограмма частот появления символов для зашифрованного текста методом маршрутной перестановки (по спирали)

Можно заметить некую разницу, ибо алгоритм дополняет сообщение символом «\*», если при заданных параметрах таблицы остаются пустые ячейки.

Так, как файл хранит в себе 500+ символов, а мои имя и фамилия имеют длину по 7 символов (т.е. таблица зашифрования методом множественной перестановки будет иметь только 49 ячеек), то для этого задания будем генерировать случайное сообщение на русском языке. Гистограмма частот появления символов для случайно сгенерированного текста предоставлена на рисунке 3.8.

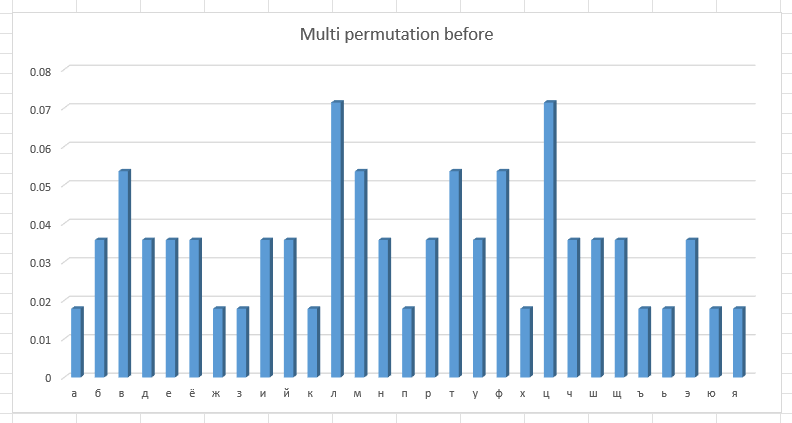


Рисунок 3.8 – Гистограмма частот появления символов для случайно сгенерированного текста

Гистограмма частот появления символов для зашифрованного случайно сгенерированного текста методом множественной перестановки предоставлена на рисунке 3.9.

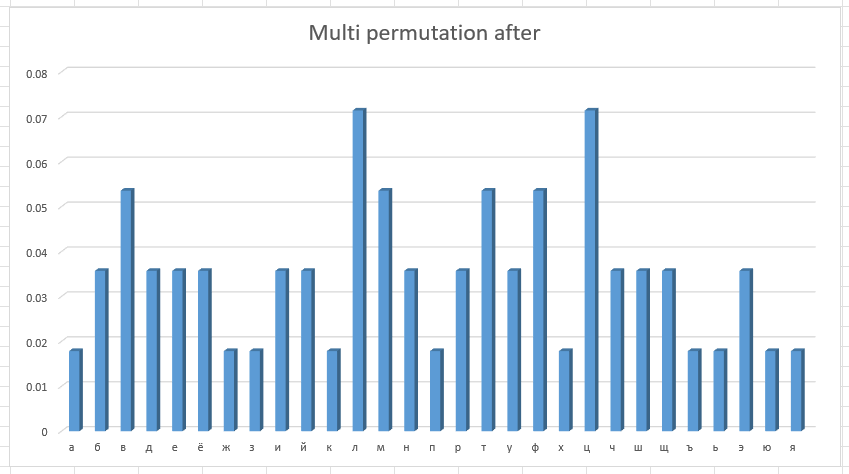


Рисунок 3.9 – Гистограмма частот появления символов для зашифрованного сообщения методом множественной перестановки

Как можно заметить, графики частот не изменились, ибо переставлялись те же символы, что и были в сообщении. Никаким образом не добавлялись новые, как это было в прошлой лабораторной работе, где символы брались напрямую из алфавита и менялось их количество, а следовательно, и частота. Здесь же частоты не изменяются.

**Задание №3**. Оценить время выполнения операций зашифрования/расшифрования авторского приложения.

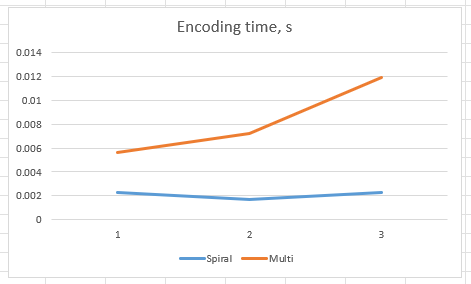


Рисунок 3.10 – Время зашифрования текста в секундах

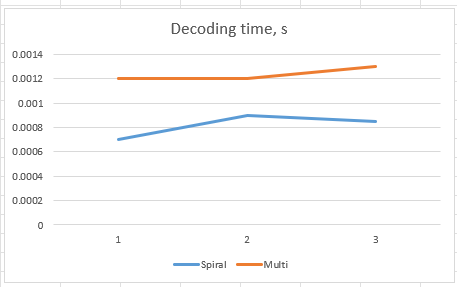


Рисунок 3.11 – Время расшифрования текста в секундах

**Вывод**: в ходе этой лабораторной работы я изучил различные криптографические методы шифрования на основе перестановки символов. Написал программу, которая зашифровывает/расшифровывает исходный текст. Рассчитал время исполнения каждого процесса, а также построил графики частот появления символов на основе исходного текста, а также на основе зашифрованных текстов.

# Листинг «Лабораторная работа №3»

using System.Text;

static void CalculateCharsFrequency(ref Dictionary<char, int> alphabet, string message)

{

foreach (var ch in message) {

if (!alphabet.TryAdd(ch, 1)) {

alphabet[ch] += 1;

}

}

}

static void CharsFrequencyToCSV(ref readonly Dictionary<char, int> alphabet, string fileName)

{

int length = alphabet.Sum(item => item.Value);

File.WriteAllText(fileName, "Char\tFrequency\n");

foreach (var item in alphabet) {

string ch = item.Key == '\n' ? "\\n" :

item.Key == '\r' ? "\\r" :

item.Key == '\"' ? "\\\"" :

item.Key == '\'' ? "\\'" :

item.Key.ToString();

File.AppendAllText(fileName, ch + "\t" + (double)item.Value / length + '\n');

}

}

static int CalculateMatrixRows(int columns, int messageLength) {

return (int)Math.Ceiling((double)messageLength / columns);

}

static string SpiralPermutationEncode(string message, int columns)

{

if (columns < 5) {

throw new Exception("Spiral permutation encoding: Number of columns must be >= 5");

}

int rows = CalculateMatrixRows(columns, message.Length);

char[,] matrix = new char[rows, columns];

string formatMessage = message + new string('\*', columns \* rows - message.Length);

for (int i = 0; i < rows; i++) {

for (int j = 0; j < columns; j++) {

matrix[i, j] = formatMessage[i \* columns + j];

}

}

string encodedMessage = "";

// Вообще не понимаю как читать по спирали, пытался вывести закономерность, но ничего не понял

// Не знаю что здесь вообще происходит, взял готовый алгоритм с какого-то задрыпанного сайта по С

// https://www.haikson.com/programming/zapolnenie-dvumernoj-matritsyi-po-spirali/

int Ibeg = 0;

int Ifin = 0;

int Jbeg = 0;

int Jfin = 0;

int counter = 0;

int I = 0;

int J = 0;

while (counter < rows \* columns){

encodedMessage += formatMessage[I \* columns + J];

if (I == Ibeg && J < columns - Jfin - 1)

++J;

else if (J == columns - Jfin - 1 && I < rows - Ifin - 1)

++I;

else if (I == rows - Ifin - 1 && J > Jbeg)

--J;

else

--I;

if ((I == Ibeg + 1) && (J == Jbeg) && (Jbeg != columns - Jfin - 1)){

++Ibeg;

++Ifin;

++Jbeg;

++Jfin;

}

counter++;

}

// -------------------------------------------

return encodedMessage;

}

static string SpiralPermutationDecode(string message, int columns)

{

if (columns < 5) {

throw new Exception("Spiral permutation encoding: Number of columns must be >= 5");

}

int rows = CalculateMatrixRows(columns, message.Length);

char[,] matrix = new char[rows, columns];

int Ibeg = 0;

int Ifin = 0;

int Jbeg = 0;

int Jfin = 0;

int counter = 0;

int I = 0;

int J = 0;

while (counter < rows \* columns){

matrix[I, J] = message[counter];

if (I == Ibeg && J < columns - Jfin - 1)

++J;

else if (J == columns - Jfin - 1 && I < rows - Ifin - 1)

++I;

else if (I == rows - Ifin - 1 && J > Jbeg)

--J;

else

--I;

if ((I == Ibeg + 1) && (J == Jbeg) && (Jbeg != columns - Jfin - 1)){

++Ibeg;

++Ifin;

++Jbeg;

++Jfin;

}

counter++;

}

string decodedMessage = "";

for (int i = 0; i < rows; i++) {

for (int j = 0; j < columns; j++) {

decodedMessage += matrix[i, j].ToString();

}

}

return decodedMessage[..decodedMessage.IndexOf('\*')];

}

static string GenerateRandomMessage(int length)

{

string characters = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя";

string message = "";

for (int i = 0; i < length; i++) {

int rand = new Random().Next() % characters.Length;

message += characters[rand];

}

return message;

}

static string MultiPermutationEncode(string message, string key1, string key2)

{

string encodedMessage = "";

var key1copy = new StringBuilder(key1);

var key2copy = new StringBuilder(key2);

char[][] matrix = new char[key1.Length][];

string sortedKey1 = string.Concat(key1.OrderBy(c => c));

string sortedKey2 = string.Concat(key2.OrderBy(c => c));

for (int i = 0; i < key1.Length; i++) {

matrix[i] = new char[key2.Length];

for (int j = 0; j < key2.Length; j++) {

matrix[i][j] = message[i \* key2.Length + j];

}

}

char[][] matrixBuf = (char[][])matrix.Clone();

for (int i = 0; i < sortedKey1.Length; i++) {

int rowIndex = key1copy.ToString().IndexOf(sortedKey1[i]);

key1copy[rowIndex] = '\_';

matrix[i] = (char[])matrixBuf[rowIndex].Clone();

}

for (int i = 0; i < key1.Length; i++) {

for (int j = 0; j < key2.Length; j++) {

matrixBuf[i][j] = matrix[i][j];

}

}

for (int i = 0; i < key1.Length; i++) {

for (int j = 0; j < key2.Length; j++) {

int columnIndex = key2copy.ToString().IndexOf(sortedKey2[j]);

key2copy[columnIndex] = '\_';

matrix[i][j] = matrixBuf[i][columnIndex];

}

key2copy = new StringBuilder(key2);

}

for (int i = 0; i < key2.Length; i++) {

for (int j = 0; j < key1.Length; j++) {

encodedMessage += matrix[j][i];

}

}

return encodedMessage;

}

static string MultiPermutationDecode(string message, string key1, string key2)

{

string decodedMessage = "";

char[][] matrix = new char[key1.Length][];

char[][] matrixBuf = new char[key1.Length][];

var sortedKey1 = new StringBuilder(string.Concat(key1.OrderBy(c => c)));

var sortedKey2 = new StringBuilder(string.Concat(key2.OrderBy(c => c)));

for (int i = 0; i < key1.Length; i++) {

matrix[i] = new char[key2.Length];

matrixBuf[i] = new char[key2.Length];

for (int j = 0; j < key2.Length; j++) {

matrix[i][j] = message[j \* key1.Length + i];

matrixBuf[i][j] = message[j \* key1.Length + i];

}

}

for (int i = 0; i < key1.Length; i++) {

for (int j = 0; j < key2.Length; j++) {

int columnIndex = sortedKey2.ToString().IndexOf(key2[j]);

sortedKey2[columnIndex] = '\_';

matrix[i][j] = matrixBuf[i][columnIndex];

}

sortedKey2 = new StringBuilder(string.Concat(key2.OrderBy(c => c)));

}

for (int i = 0; i < key1.Length; i++) {

for (int j = 0; j < key2.Length; j++) {

matrixBuf[i][j] = matrix[i][j];

}

}

for (int i = 0; i < sortedKey1.Length; i++) {

int rowIndex = sortedKey1.ToString().IndexOf(key1[i]);

sortedKey1[rowIndex] = '\_';

matrix[i] = (char[])matrixBuf[rowIndex].Clone();

}

for (int i = 0; i < key1.Length; i++) {

for (int j = 0; j < key2.Length; j++) {

decodedMessage += matrix[i][j];

}

}

return decodedMessage;

}

var fileContent = File.ReadAllText("file.txt").ToLower();

var alphabet = new Dictionary<char, int>();

var spiralAlphabet = new Dictionary<char, int>();

var multiAlphabetBefore = new Dictionary<char, int>();

var multiAlphabetAfter = new Dictionary<char, int>();

string csvFileContent = @"./file.csv";

string csvSpiral = @"./spiral.csv";

string csvMultiBefore = @"./multiBefore.csv";

string csvMultiAfter = @"./multiAfter.csv";

CalculateCharsFrequency(ref alphabet, fileContent);

CharsFrequencyToCSV(ref alphabet, csvFileContent);

int spiralColumns = 20;

var watch = System.Diagnostics.Stopwatch.StartNew();

var spiralEncodedMessage = SpiralPermutationEncode(fileContent, spiralColumns);

watch.Stop();

Console.WriteLine("Spiral encoding time, ms: " + watch.Elapsed);

watch = System.Diagnostics.Stopwatch.StartNew();

var spiralDecodedMessage = SpiralPermutationDecode(spiralEncodedMessage, spiralColumns);

watch.Stop();

Console.WriteLine("Spiral decoding time, ms: " + watch.Elapsed);

CalculateCharsFrequency(ref spiralAlphabet, spiralEncodedMessage);

CharsFrequencyToCSV(ref spiralAlphabet, csvSpiral);

Console.WriteLine($"Spiral decoded == file: {spiralDecodedMessage == fileContent}");

const string key1 = "воликов";

const string key2 = "дмитрийй";

string randomMessage = GenerateRandomMessage(key1.Length \* key2.Length);

CalculateCharsFrequency(ref multiAlphabetBefore, randomMessage);

CharsFrequencyToCSV(ref multiAlphabetBefore, csvMultiBefore);

Console.WriteLine();

Console.WriteLine("Random message for multi permutation: " + randomMessage);

watch = System.Diagnostics.Stopwatch.StartNew();

var multiEncodedMessage = MultiPermutationEncode(randomMessage, key1, key2);

watch.Stop();

Console.WriteLine("Multi encoding time, ms: " + watch.Elapsed);

Console.WriteLine("Encoded message: " + multiEncodedMessage);

CalculateCharsFrequency(ref multiAlphabetAfter, multiEncodedMessage);

CharsFrequencyToCSV(ref multiAlphabetAfter, csvMultiAfter);

watch = System.Diagnostics.Stopwatch.StartNew();

var multiDecodedMessage = MultiPermutationDecode(multiEncodedMessage, key1, key2);

watch.Stop();

Console.WriteLine("Multi decoding time, ms: " + watch.Elapsed);

Console.WriteLine("Decoded message: " + multiDecodedMessage);

Console.WriteLine($"Multi permutation decoded == random message: {multiDecodedMessage == randomMessage}");

Листинг 1 – Код программы на языке C#