МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5 НА ТЕМУ:**

**ИССЛЕДОВАНИЕ КРИПТОГРАФИЧЕСКИХ ШИФРОВ НА ОСНОВЕ ПЕРЕСТАНОВКИ СИМВОЛОВ**

                                                                 Выполнил студент 3 курса 5 группы

Дмитрук Илья Игоревич

Минск 2024

# Задание 1.

Сообщение шифровалось маршрутной перестановкой. C начала символы сообщения записывались по столбцам, затем считывались по строкам. Для реализации данного алгоритма шифрования, была разработана функция EncryptRoutePermutation. Её код представлен в листинге 1.1.

public static string EncryptRoutePermutation(string text)

{

char[,] table = new char[10, 50];

for (int i = 0, n = 0, t = 0; i < text.Length; i++, n++)

{

table[n, t] = text[i];

if (n == 9)

{

n = -1;

t++;

}

}

StringBuilder encryptText = new StringBuilder();

for (int i = 0; i < table.GetLength(0); i++)

{

for (int j = 0; j < table.GetLength(1); j++)

{

encryptText.Append(table[i, j]);

}

}

return encryptText.ToString();

}

Листинг 1.1 – Функции для шифрования маршрутной перестановкой

В качестве сообщения для шифрования использовался текст на польском языке. Текст, после шифрования, представлен на рисунке 1.1

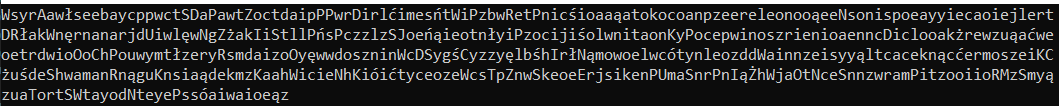


Рисунок 1.1 – Зашифрованный маршрутной перестановкой текст

Для расшифрования сообщения, необходимо было провести действия в обратном порядке. Код функции DecryptRoutePermutation, предназначенной для расшифрования сообщения, зашифрованного маршрутной перестановкой, представлен в листинге 1.2.

public static string DecryptRoutePermutation(string text)

{

char[,] table = new char[10, 50];

for (int i = 0, n = 0, t = 0; i < text.Length; i++, t++)

{

table[n, t] = text[i];

if (t == 49)

{

n++;

t = -1;

}

}

StringBuilder decrypttText = new StringBuilder();

for (int i = 0; i < table.GetLength(0); i++)

{

for (int j = 0; j < table.GetLength(1); j++)

{

decrypttText.Append(table[i, j]);

}

}

return decrypttText.ToString();

}

Листинг 1.2 – Функции для расшифрования маршрутной перестановки

В результате выполнения расшифрования, мы получили исходный текст. Он представлен на рисунке 1.2.

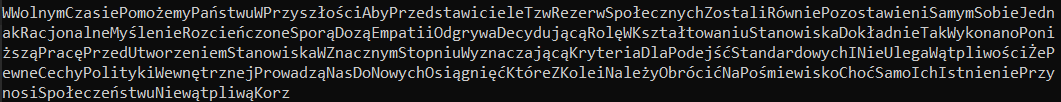


Рисунок 1.2 – Расшифрованный текст, зашифрованный маршрутной перестановкой

# Задание 2.

Для шифрования сообщения использовался шифр множественной перестановки. Суть состоит в том, что мы записываем символы текста сообщения в таблицу. Столбцы и строки таблицы именуем символами ключевых слов. Затем мы сортируем ячейки таблицы в алфавитном порядке символов, именующих строки и столбцы. Таким образом символы сообщения переместятся на новые позиции в таблице и у нас будет сформирован новый текст. Алгоритм данного шифрования реализован в функции EncryptMultiplePermutation. Её код представлен в листинге 2.1.

public static string EncryptMultiplePermutation(string text, string columnKey, string rowKey)

{

int columnLength = columnKey.Length;

int rowLength = rowKey.Length;

int index = 0;

int size = (int)Math.Ceiling((double)text.Length / (rowLength \* columnLength));

char[,] matrix = new char[rowLength, columnLength];

StringBuilder encryptedText = new StringBuilder();

for (int k = 0; k < size; k++)

{

for (int i = 0; i < rowLength; i++)

{

for (int j = 0; j < columnLength; j++)

{

if (index < text.Length) matrix[i, j] = text[index++];

else matrix[i, j] = '\0';

}

}

char[,] buff1 = new char[rowLength, columnLength];

int pr1 = 0;

HashSet<int> processedIndexes1 = new HashSet<int>();

foreach (char column in columnKey)

{

for (int columnIndex = 0; columnIndex < columnKey.Length; columnIndex++)

{

if (column == columnKey[columnIndex] && !processedIndexes1.Contains(columnIndex))

{

processedIndexes1.Add(columnIndex);

for (int i = 0; i < rowLength; i++)

buff1[i, pr1] = matrix[i, columnIndex];

pr1++;

}

}

}

char[,] buff2 = new char[rowLength, columnLength];

int pr2 = 0;

HashSet<int> processedIndexes2 = new HashSet<int>();

foreach (char row in rowKey)

{

for (int rowIndex = 0; rowIndex < rowKey.Length; rowIndex++)

{

if (row == rowKey[rowIndex] && !processedIndexes2.Contains(rowIndex))

{

processedIndexes2.Add(rowIndex);

for (int i = 0; i < columnLength; i++)

buff2[pr2, i] = buff1[rowIndex, i];

pr2++;

}

}

}

for (int i = 0; i < columnLength; i++)

{

for (int j = 0; j < rowLength; j++)

encryptedText.Append(buff2[j, i]);

}

}

return encryptedText.ToString();

}

Листинг 2.1 – Функции для шифрования шифром множественной перестановки

В качестве сообщения для шифрования использовался текст на польском языке. Текст, после шифрования, представлен на рисунке 2.1.

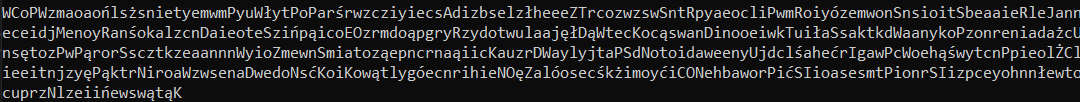


Рисунок 2.1 – Зашифрованный множественной перестановкой текст

Для расшифрования сообщения, необходимо было так же записать текст сообщения в таблицу и поменять ключи друг на друга. Реализация данного алгоритма расшифрования представлена в функции DecryptMultiplePermutation. Её код представлен в листинге 2.2.

public static string DecryptMultiplePermutation(string text, string rowKey, string columnKey)

{

int columnLength = columnKey.Length;

int rowLength = rowKey.Length;

int index = 0;

int size = (int)Math.Ceiling((double)text.Length / (rowLength \* columnLength));

char[,] matrix = new char[rowLength, columnLength];

StringBuilder decodeText = new StringBuilder();

for (int k = 0; k < size; k++)

{

for (int i = 0; i < rowLength; i++)

{

for (int j = 0; j < columnLength; j++)

{

if (index < text.Length) matrix[i, j] = text[index++];

else

{

matrix[i, j] = '\0';

}

}

}

char[,] buff1 = new char[rowLength, columnLength];

int pr1 = 0;

HashSet<int> processedIndexes1 = new HashSet<int>();

foreach (char row in rowKey)

{

for (int rowIndex = 0; rowIndex < rowKey.Length; rowIndex++)

{

if (row == rowKey[rowIndex] && !processedIndexes1.Contains(rowIndex))

{

processedIndexes1.Add(rowIndex);

for (int i = 0; i < columnLength; i++)

{

buff1[rowIndex, i] = matrix[pr1, i];

}

pr1++;

}

}

}

char[,] buff2 = new char[rowLength, columnLength];

int pr2 = 0;

HashSet<int> processedIndexes2 = new HashSet<int>();

foreach (char column in columnKey)

{

for (int columnIndex = 0; columnIndex < columnKey.Length; columnIndex++)

{

if (column == columnKey[columnIndex] && !processedIndexes2.Contains(columnIndex))

{

processedIndexes2.Add(columnIndex);

for (int i = 0; i < rowLength; i++)

{

buff2[i, columnIndex] = buff1[i, pr2];

}

pr2++;

}

}

}

for (int i = 0; i < columnLength; i++)

{

for (int j = 0; j < rowLength; j++)

{

decodeText.Append(buff2[j, i]);

}

}

}

return decodeText.ToString().Replace("\0", "");

}

Листинг 2.2 – Функции для расшифрования множественной перестановки

Результат расшифрования шифра множественной перестановки представлен на рисунке 2.2.

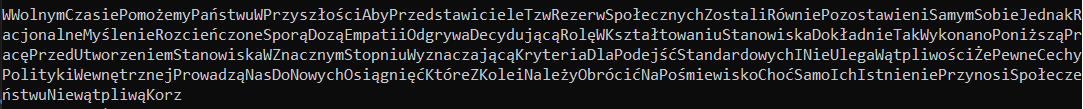


Рисунок 2.2 – Расшифрованный текс, зашифрованный множественной перестановкой

# Задание 3.

Для определения частоты появления символов в зашифрованных и расшифрованных сообщениях, были построены гистограммы. Они представлены на рисунках 3.1 – 3.3.

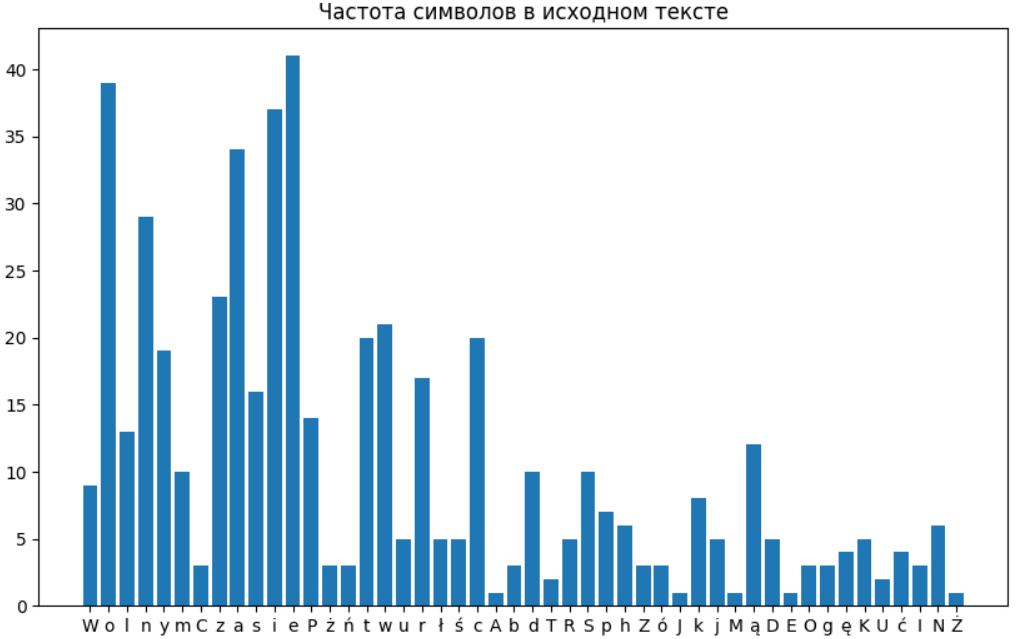


Рисунок 3.1 – Частота символов в исходном сообщении

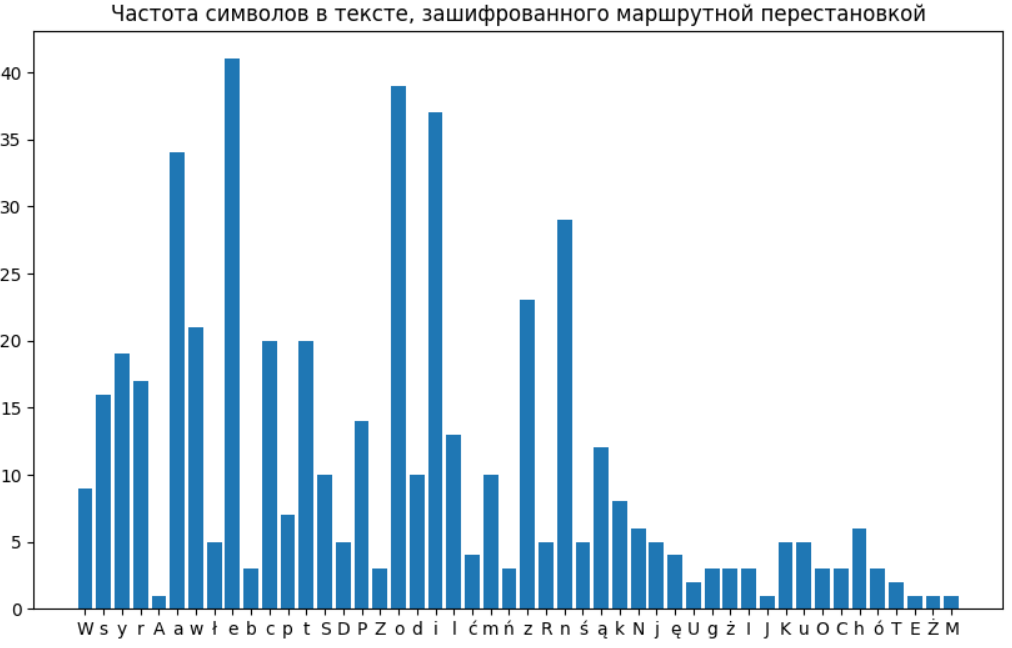


Рисунок 3.2 – Частота символов в тексте, зашифрованного маршрутной перестановкой



Рисунок 3.3 – Частота символов в тексте, зашифрованного множественной перестановкой

Так же был проведён анализ затраченного времени на шифрование и расшифрования сообщений. Результат измерений времён представлен на рисунке 3.4.

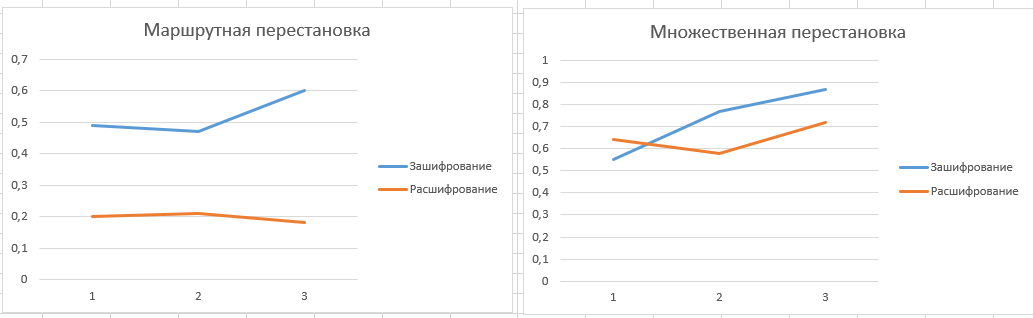


Рисунок 3.4 – Время шифрования и расшифрования

# Вывод

В ходе лабораторной работы были приобретены навыки шифрования и расшифрования сообщений разными способами перестановки символов. Данные способы были проанализированы на скорость выполнения работы. Так же они были реализованы программными средствами.