Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

**Лабораторная работа № 3**

« ИССЛЕДОВАНИЕ КРИПТОГРАФИЧЕСКИХ ШИФРОВ НА ОСНОВЕ ПЕРЕСТАНОВКИ СИМВОЛОВ »

Выполнил:

Студент: Ковалев А.А.

ФИТ 3 курс 4 группа

Преподаватель: Копыток Д.В.

Минск 2022

1. **Описание приложения**

Приложение написано на языке программирования C# и позволяет провести шифрование сообщения на Немецком языке с применением алгоритмов маршрутной перестановки змейкой и множественной перестановки.

1. **Листинги и описание реализации приложения**

Описание класса, содержащий совокупность методов, полей и свойств, используемых для реализации алгоритма множественной перестановки представлен в листинге 2.1.

class MultipleReplacement

{

private List<KeyValuePair<int,char>> keyHorizontal;

private List<KeyValuePair<int, char>> keyVertical;

private string text;

public string Text

{

get { return text; }

set { text = value; }

}

public List<KeyValuePair<int, char>> KeyVertical

{

get { return keyVertical; }

set { keyVertical = value; }

}

public List<KeyValuePair<int, char>> KeyHorizontal

{

get { return keyHorizontal; }

set { keyHorizontal = value; }

}

public MultipleReplacement()

{

}

public void printKeys()

{

Console.Write($" ");

foreach (KeyValuePair<int,char> x in keyHorizontal)

{

Console.Write($"{x.Value,5}");

}

Console.WriteLine($"");

Console.Write($" ");

foreach (KeyValuePair<int, char> x in keyHorizontal)

{

Console.Write($"{x.Key,5}");

}

Console.WriteLine($"");

foreach (KeyValuePair<int, char> x in keyVertical)

{

Console.Write($"{x.Value,2}");

Console.WriteLine($"{x.Key,2}");

}

}

public MultipleReplacement(string text, List<KeyValuePair<int, char>> keyVertical, List<KeyValuePair<int, char>> keyHorizontal)

{

if(keyVertical.Count \* keyHorizontal.Count >= text.Length)

{

Text = text;

}

else

{

throw new Exception("Сообщение слишком длинное для данных ключей!");

}

KeyVertical = keyVertical;

KeyHorizontal = keyHorizontal;

}

public void printMatrix(char[,] input)

{

int tableWidth = keyVertical.Count;

int tableHeight = KeyHorizontal.Count;

for (int w = 0; w < tableWidth; w++)

{

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

{

Console.Write($"{(input[w, i] == ' ' || input[w, i] == '\0'? '\*' : input[w, i]),5}");

}

Console.Write($"\n");

}

Console.Write($"\n");

}

public char[,] createMatrix(string input)

{

int tableWidth = keyVertical.Count;

int tableHeight = KeyHorizontal.Count;

char[,] table = new char[tableWidth, tableHeight];

int l = 0;

for (int w = 0; w < tableWidth; w++)

{

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

{

if (l == input.Length) break;

table[w, i] = input[l++];

}

if (l == input.Length) break;

}

return table;

}

public string createString(char[] input)

{

StringBuilder stringBuilder = new StringBuilder();

for (int w = 0; w < input.Length; w++)

{

stringBuilder.Append(input[w]);

}

return stringBuilder.ToString();

}

public string createString(char[,] input)

{

int tableWidth = keyVertical.Count;

int tableHeight = KeyHorizontal.Count;

StringBuilder stringBuilder = new StringBuilder();

for (int w = 0; w < tableWidth; w++)

{

for (int l = 0; l < tableHeight; l++)

stringBuilder.Append(input[w, l]);

}

return stringBuilder.ToString();

}

public string Encrypt()

{

char[,] table = createMatrix(Text);

int tableWidth = keyVertical.Count;

int tableHeight = KeyHorizontal.Count;

char[,] result = new char[tableWidth, tableHeight];

int iteration = 0;

while(iteration++ < tableHeight )

{

int k = 0;

for (int y = 0; y < tableWidth; y++)

{

k = keyHorizontal.IndexOf(keyHorizontal.Where(l => l.Key == iteration).First());

result[y,iteration - 1] = table[y,k];

}

}

Console.WriteLine("После горизонтальной перестановки:\n");

printMatrix(result);

char[,] resultV = new char[tableWidth, tableHeight];

iteration = 0;

while (iteration++ < tableWidth)

{

int k = 0;

for (int y = 0; y < tableHeight ; y++)

{

k = keyVertical.IndexOf(keyVertical.Where(l => l.Key == iteration).First());

resultV[iteration - 1,y] = result[k,y];

}

}

Console.WriteLine("После вертикальной перестановки:\n");

printMatrix(resultV);

return createString(resultV);

}

}

**Листинг 2.1 – Класс для шифрования сообщения алгоритмом множественной перестановки**

Метод createMatrix используется для преобразования входной последовательности символов в двумерную матрицу для дальнейшего вывода на консоль и визуализации процесса перестановки.

Метод createString используется для преобразования двумерного массива символов строку для последующего вывода в консоль результата шифрования и дешифрования сообщения.

Методы printMatrix и printKeys используются для вывода на консоль соответственно матрицы символов и ключей, используемых для шифрования сообщений.

Методы encrypt и decrypt используются соответственно для реализации процесса шифрования и дешифрования сообщения.

Описание класса, содержащий совокупность методов, полей и свойств, используемых для реализации алгоритма маршрутной перестановки змейкой, представлен в листинге 2.2.

class SnakeEncrypter

{

private int tableHeight;

private int tableWidth;

private string text;

public int TableHeight

{

get { return tableHeight; }

set { tableHeight = value; }

}

public int TableWidth

{

get { return tableWidth; }

set { tableWidth = value; }

}

public string Text

{

get { return text; }

set { text = value; }

}

public SnakeEncrypter(int tableHeight, int tableWidth, string text)

{

TableHeight = tableHeight;

TableWidth = tableWidth;

Text = text;

}

public char[,] createMatrix(string input)

{

char[,] table = new char[tableWidth, tableHeight];

int l = 0;

for(int w = 0; w< tableWidth; w++)

{

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

{

if (l == input.Length) break;

table[w, i] = input[l++];

}

if (l == input.Length) break;

}

return table;

}

public string createString(char[] input)

{

StringBuilder stringBuilder = new StringBuilder();

for (int w = 0; w < input.Length; w++)

{

stringBuilder.Append(input[w]);

}

return stringBuilder.ToString();

}

public string createString(char[,] input)

{

StringBuilder stringBuilder = new StringBuilder();

for (int w = 0; w < tableWidth; w++)

{

for(int l = 0; l < tableHeight ; l++)

stringBuilder.Append(input[w,l]);

}

return stringBuilder.ToString();

}

public void printMatrix(char[,] input)

{

for (int w = 0; w < tableWidth; w++)

{

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

{

Console.Write($"{(input[w, i] == ' ' ? '\*' : input[w, i])} ");

}

Console.Write($"\n");

}

Console.Write($"\n");

}

public string Encrypt()

{

char[,] table = createMatrix(text);

char[] result = new char[text.Length];

int x = 0, y = 0, l =0;

while (true)

{

if (x < tableHeight)

{

if (table[y, x] != '\0')

result[l++] = table[y, x++];

else

x++;

}

else

x++;

while(x != 0)

{

if (y < tableWidth && x < tableHeight)

{

if (table[y, x] != '\0')

result[l++] = table[y++, x--];

else

{

y++; x--;

}

}

else

{

y++; x--;

}

}

if(y < tableWidth)

{

if (table[y, x] != '\0')

result[l++] = table[y++, x];

else

y++;

}

else

y++;

while (y != 0)

{

if (y < tableWidth && x < tableHeight)

{

if (table[y, x] != '\0')

result[l++] = table[y--, x++];

else

{

y--; x++;

}

}

else

{

y--; x++;

}

}

if (l == text.Length) break;

}

return createString(result);

}

public char[,] Decrypt(string input)

{

char[,] table = new char[tableWidth,tableHeight];

int x = 0, y = 0, l = 0;

int last = tableWidth \* tableHeight - input.Length;

int downY = (last % tableHeight == last ? tableHeight : (tableHeight - last / tableHeight) - 1);

int downX = (last % tableWidth == last ? tableWidth : (tableWidth - last % tableWidth) - 1);

List<KeyValuePair<int,int>> abadonCells = new List<KeyValuePair<int, int>>();

for(int xa = tableWidth - 1, iter = last; ;xa--)

{

for(int ya = tableHeight - 1; ya >= 0; ya--)

{

abadonCells.Add(new KeyValuePair<int,int>(ya,xa));

last--;

if (last == 0) break;

}

if (last == 0) break;

}

while (true)

{

if (x < tableHeight)

{

if (abadonCells.Where(z => z.Key == x && z.Value == y).Count() == 0)

table[y, x++] = input[l++];

else x++;

}

else х++;

while (x != 0)

{

if (l == text.Length) break;

if (y < tableWidth && x < tableHeight)

{

if (abadonCells.Where(z => z.Key == x && z.Value == y).Count() == 0)

table[y++, x--] = input[l++];

else

{

y++; x--;

}

}

else

{y++; x--;}

}

if (y < tableWidth)

{

if (abadonCells.Where(z => z.Key == x && z.Value == y).Count() == 0)

table[y++, x] = input[l++];

else

y++;

}

else

y++;

while (y != 0)

{

if (l == text.Length) break;

if (y < tableWidth && x < tableHeight)

{

if (abadonCells.Where(z => z.Key == x && z.Value == y).Count() == 0)

table[y--, x++] = input[l++];

else

{

y--; x++;

}

}

else

{y--; x++;}

}

if (l == text.Length) break;

}

return table;

}

}

**Листинг 2.2 – Класс для шифрования сообщения алгоритмом маршрутной перестановки змейкой**

Метод createMatrix используется для преобразования входной последовательности символов в двумерную матрицу для дальнейшего вывода на консоль и визуализации процесса перестановки.

Метод createString используется для преобразования двумерного массива символов строку для последующего вывода в консоль результата шифрования и дешифрования сообщения.

Методы printMatrix и printKeys используются для вывода на консоль соответственно матрицы символов и ключей, используемых для шифрования сообщений.

Методы encrypt и decrypt используются соответственно для реализации процесса шифрования и дешифрования сообщения.

Данные классы используются в методе main, посредством вызова методов сервисов, реализующих процесс подсчёта частоты вхождения каждого из символов в сообщение и шифрования сообщений двумя методами. Описание главного метода представлен в листинге 2.3.

class Program

{

static void Main(string[] args)

{

List<char> deuthAlphabet = new List<char>()

{

'a','b','c','d','e','f','g','h','i','j','k',

'l','m','n','o','p','r','s','t','u','v',

'w'

};

int[] keyH = new int[] { 5, 3, 6, 4, 2, 7, 1, 9, 8 };

char[] keyHword = new char[] { 'а', 'л', 'е', 'к', 'с', 'а', 'н', 'д', 'р' };

int[] keyV = new int[] { 6, 2, 5, 4, 3, 1 ,7};

char[] keyVword = new char[] { 'к', 'о', 'в', 'а', 'л', 'е', 'в' };

List<KeyValuePair<int,char>>keyVertical = new List<KeyValuePair<int, char>>();

List<KeyValuePair<int, char>> keyHorizontal = new List<KeyValuePair<int, char>>();

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

{

keyVertical.Add(new KeyValuePair<int,char>(keyV[i],keyVword[i]));

}

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

{

keyHorizontal.Add(new KeyValuePair<int, char>(keyH[i],keyHword[i]));

}

try

{

SnakeService.MakeSnake(deuthAlphabet);

ReplacementService.MakeReplace(deuthAlphabet,keyVertical,keyHorizontal);

}

catch (Exception ex)

{

Console.WriteLine($"\n{ex.Message}");

}

}

}

**Листинг 2.3 – Класс c главным методом**

Описание сервисов, использующих методы классов для шифрования представлены в листингах 2.4 и 2.5.

static class ReplacementService

{

public static void MakeReplace(List<char> deuthAlphabet,List<KeyValuePair<int,char>> keyVertical, List<KeyValuePair<int, char>> keyHorizontal)

{

EntropyChecker deuthChecker = new EntropyChecker(deuthAlphabet, 0, "Немецкий");

string deuthText = deuthChecker.OpenDocument("deuth.txt").ReadToEnd().ToLower();

List<char> deuthTextTrimmedList = deuthText.TakeLast(keyVertical.Count \* keyHorizontal.Count).ToList();

StringBuilder deuthTextTrimmedBuilder = new();

foreach(char x in deuthTextTrimmedList)

{

deuthTextTrimmedBuilder.Append(x);

}

string deuthTextTrimmed = deuthTextTrimmedBuilder.ToString();

Regex regex = new Regex(@"\W");

deuthTextTrimmed = regex.Replace(deuthTextTrimmed, "");

Dictionary<char, int> deuthDict = deuthChecker.alphabetListToDictionary();

MultipleReplacement replaceEncrypter = new MultipleReplacement(deuthTextTrimmed,keyVertical,keyHorizontal);

deuthChecker.getSymbolsCounts(deuthTextTrimmed, deuthDict);

Dictionary<char, double> deuthChances = deuthChecker.getSymbolsChances(deuthTextTrimmed, deuthDict);

deuthChecker.printAlphabet();

deuthChecker.printChances(deuthChances);

replaceEncrypter.printKeys();

replaceEncrypter.printMatrix(replaceEncrypter.createMatrix(replaceEncrypter.Text));

Stopwatch first = new Stopwatch();

first.Start();

string resultEnc = replaceEncrypter.Encrypt();

first.Stop();

Console.WriteLine($"Время шифрования: {first.ElapsedMilliseconds} мс \n"); ;

replaceEncrypter.printMatrix(replaceEncrypter.createMatrix(resultEnc));

Console.WriteLine(resultEnc);

ExcelDocumentCreator<char, double> excel = new ExcelDocumentCreator<char, double>(new System.IO.FileInfo("Lab3.xlsx"));

excel.createWorksheet("first");

excel.addValuesFromDict(deuthChances, "first", 3);

excel.pack.Save();

Console.ReadKey();

}

}

**Листинг 2.4 – Класс сервиса шифрования методом множественной перестановки**

static class SnakeService

{

public static void MakeSnake(List<char> deuthAlphabet)

{

EntropyChecker deuthChecker = new EntropyChecker(deuthAlphabet, 0, "Немецкий");

string deuthText = deuthChecker.OpenDocument("deuth.txt").ReadToEnd().ToLower();

Regex regex = new Regex(@"\W");

deuthText = regex.Replace(deuthText, "");

Dictionary<char, int> deuthDict = deuthChecker.alphabetListToDictionary();

SnakeEncrypter snakeEncrypter = new SnakeEncrypter(Convert.ToInt32(Math.Sqrt(deuthText.Length)), Convert.ToInt32(Math.Sqrt(deuthText.Length)), deuthText);

deuthChecker.getSymbolsCounts(deuthText, deuthDict);

Dictionary<char, double> deuthChances = deuthChecker.getSymbolsChances(deuthText, deuthDict);

deuthChecker.printAlphabet();

deuthChecker.printChances(deuthChances);

snakeEncrypter.printMatrix(snakeEncrypter.createMatrix(snakeEncrypter.Text));

Stopwatch first = new Stopwatch();

first.Start();

string resultEnc = snakeEncrypter.Encrypt();

first.Stop();

Console.WriteLine($"Время шифрования: {first.ElapsedMilliseconds} мс \n"); ;

Console.WriteLine(resultEnc);

char[,] resultDecr = snakeEncrypter.Decrypt(resultEnc);

snakeEncrypter.printMatrix(resultDecr);

string resultDecrStr = snakeEncrypter.createString(resultDecr);

ExcelDocumentCreator<char, double> excel = new ExcelDocumentCreator<char, double>(new System.IO.FileInfo("Lab3.xlsx"));

excel.createWorksheet("first");

excel.addValuesFromDict(deuthChances, "first", 0);

excel.pack.Save();

Console.ReadKey();

Console.WriteLine(resultDecrStr);

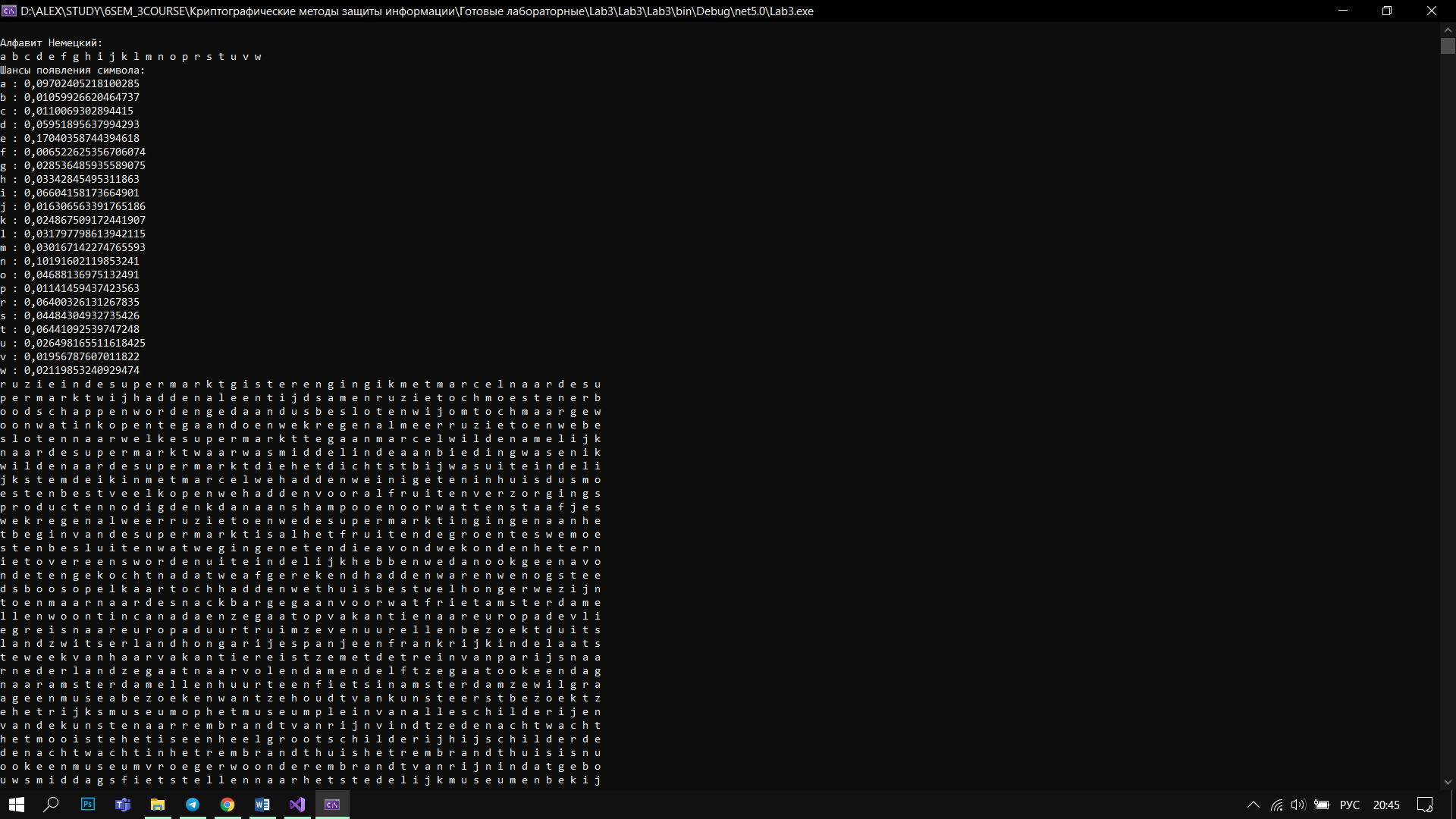
}

}

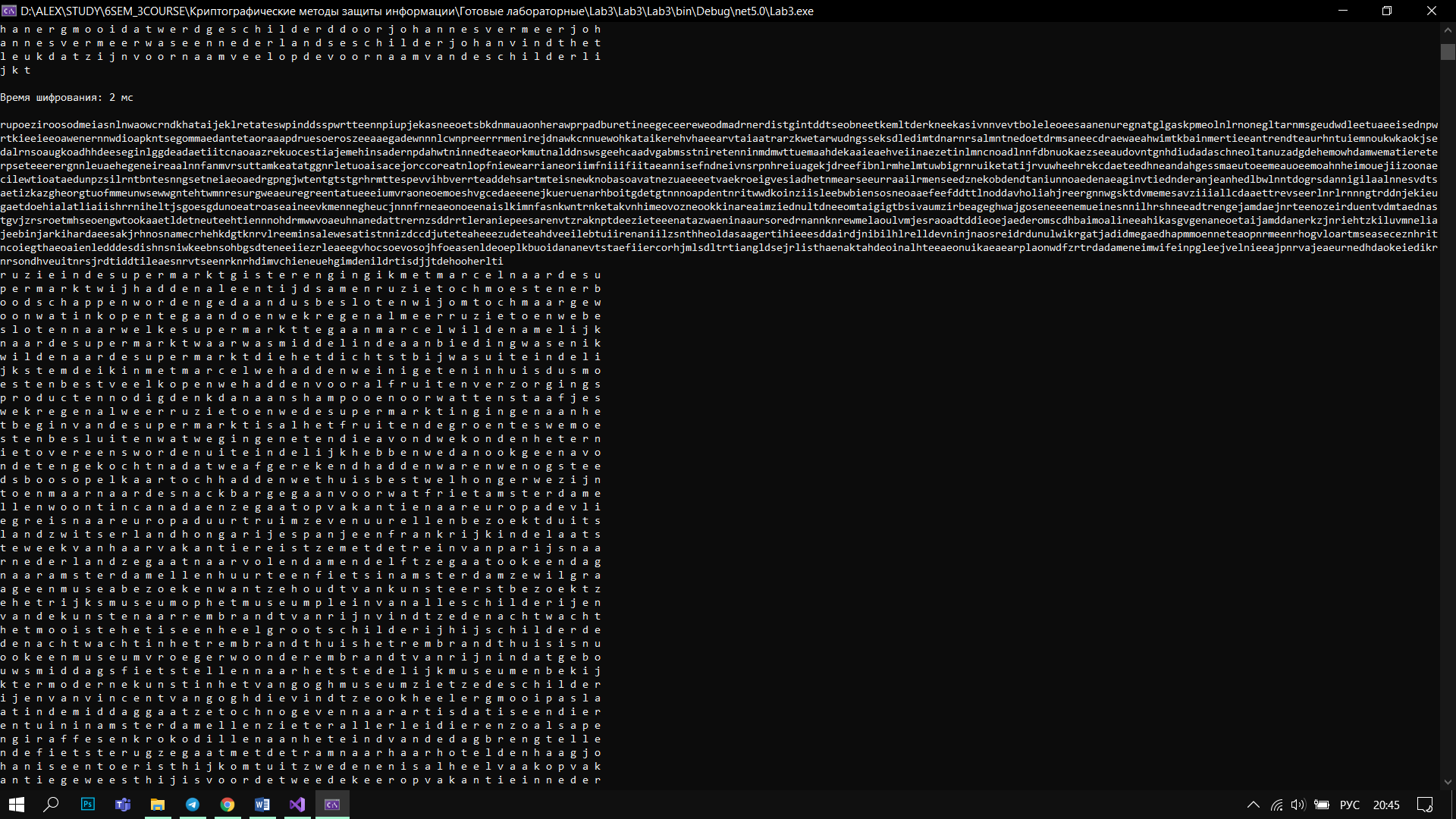
**Листинг 2.5 – Класс сервиса шифрования методом маршрутной перестановки**

**3.Результат выполнения программы**

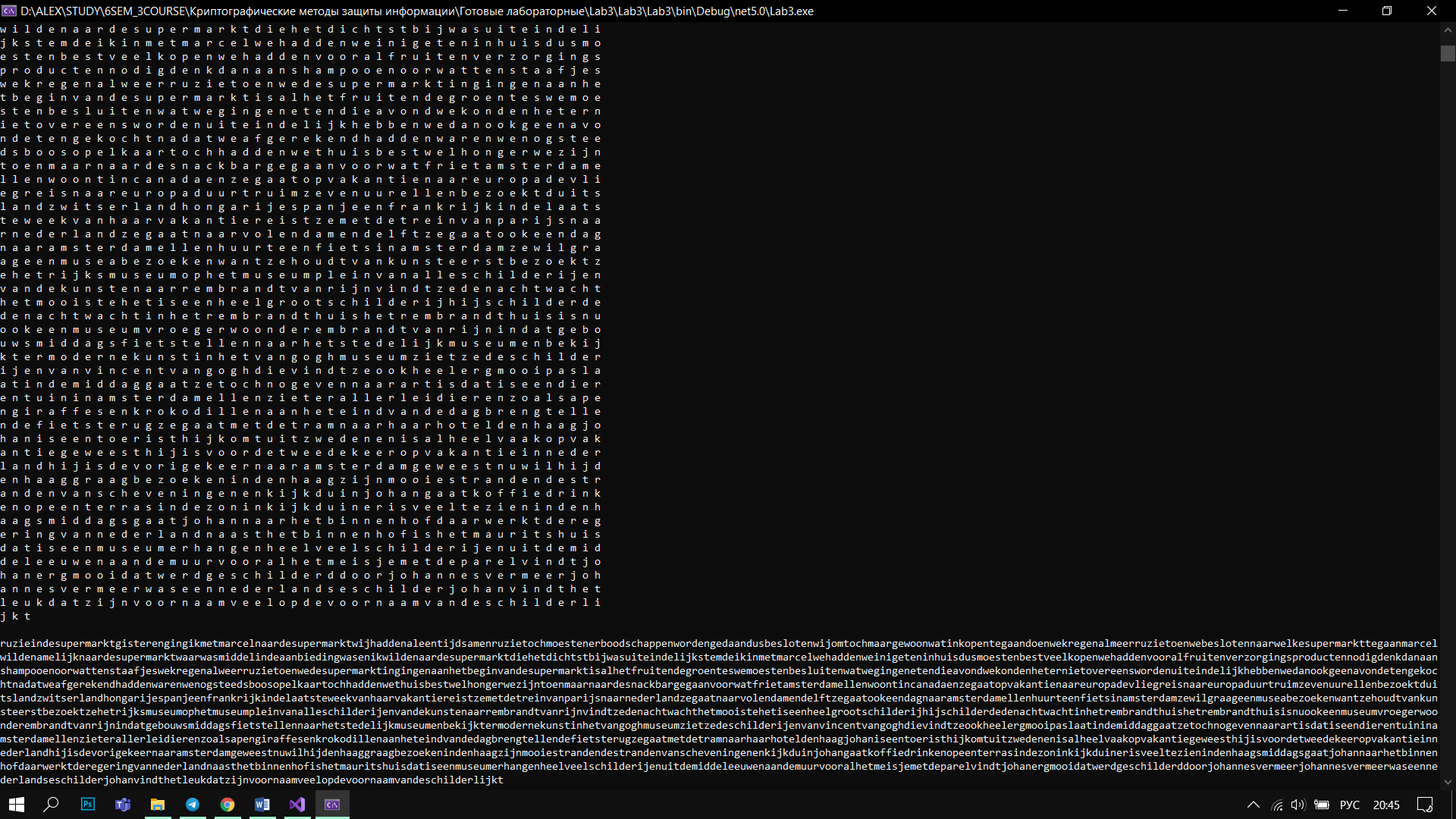
Результат выполнения шифрования алгоритмом маршрутной перестановки изображён на рисунках 3.1-3.3.



**Рисунок 3.1 – результат выполнения шифрования алгоритмом маршрутной перестановки**

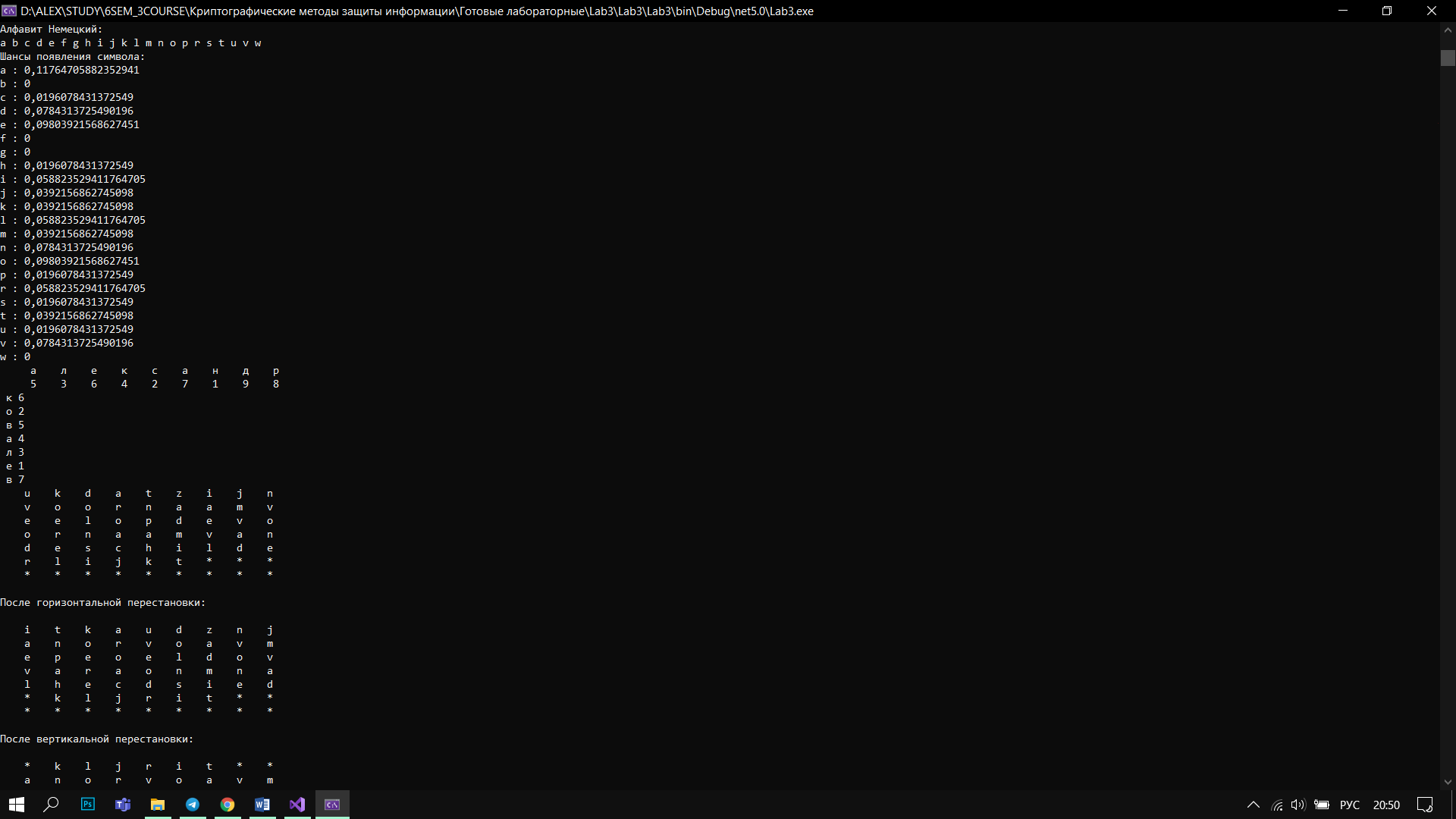


**Рисунок 3.2 – результат выполнения шифрования алгоритмом маршрутной перестановки**

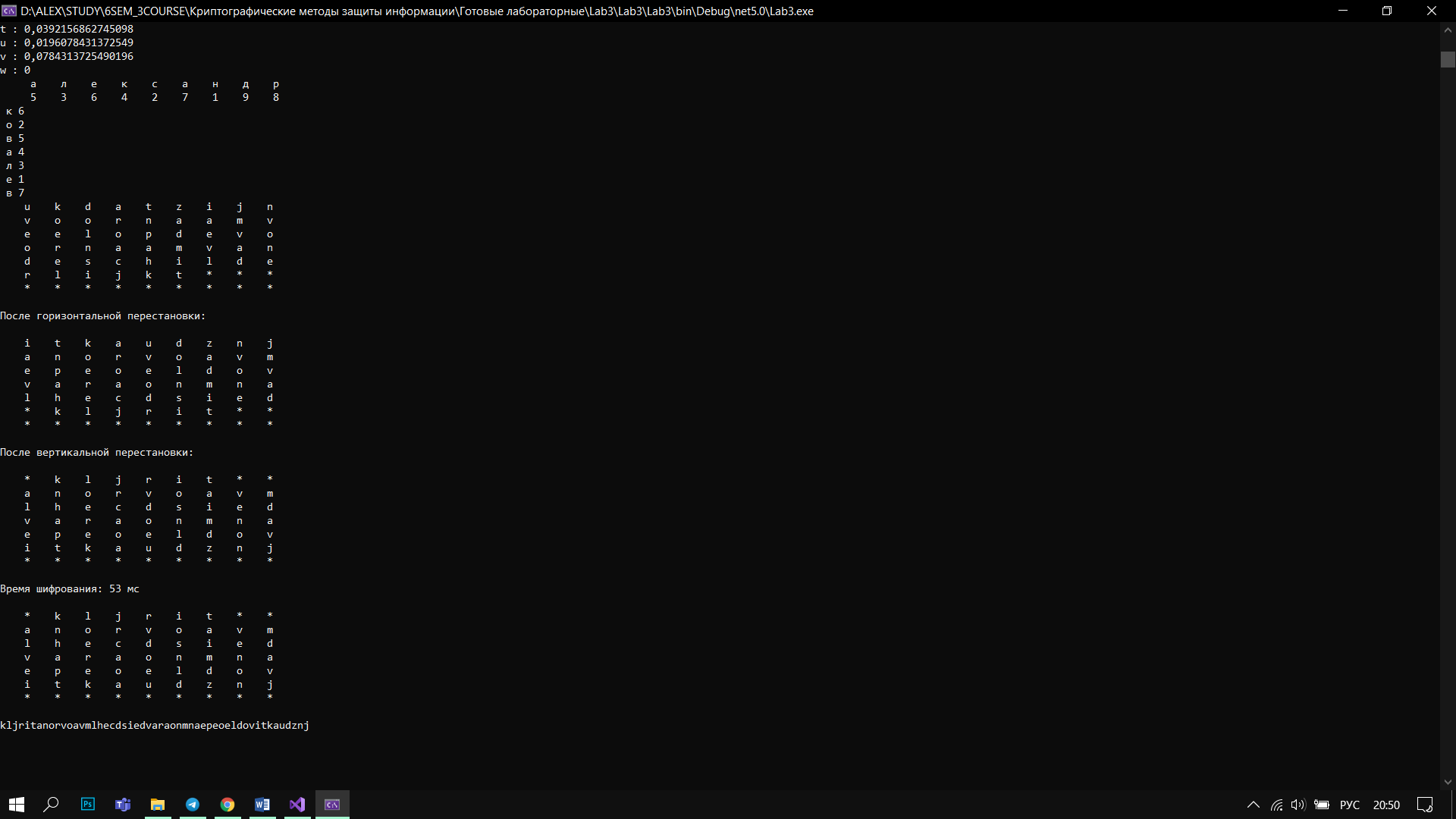


**Рисунок 3.3 – результат выполнения шифрования алгоритмом маршрутной перестановки**

Результат выполнения шифрования алгоритмом множественной перестановки изображён на рисунках 3.4-3.5.

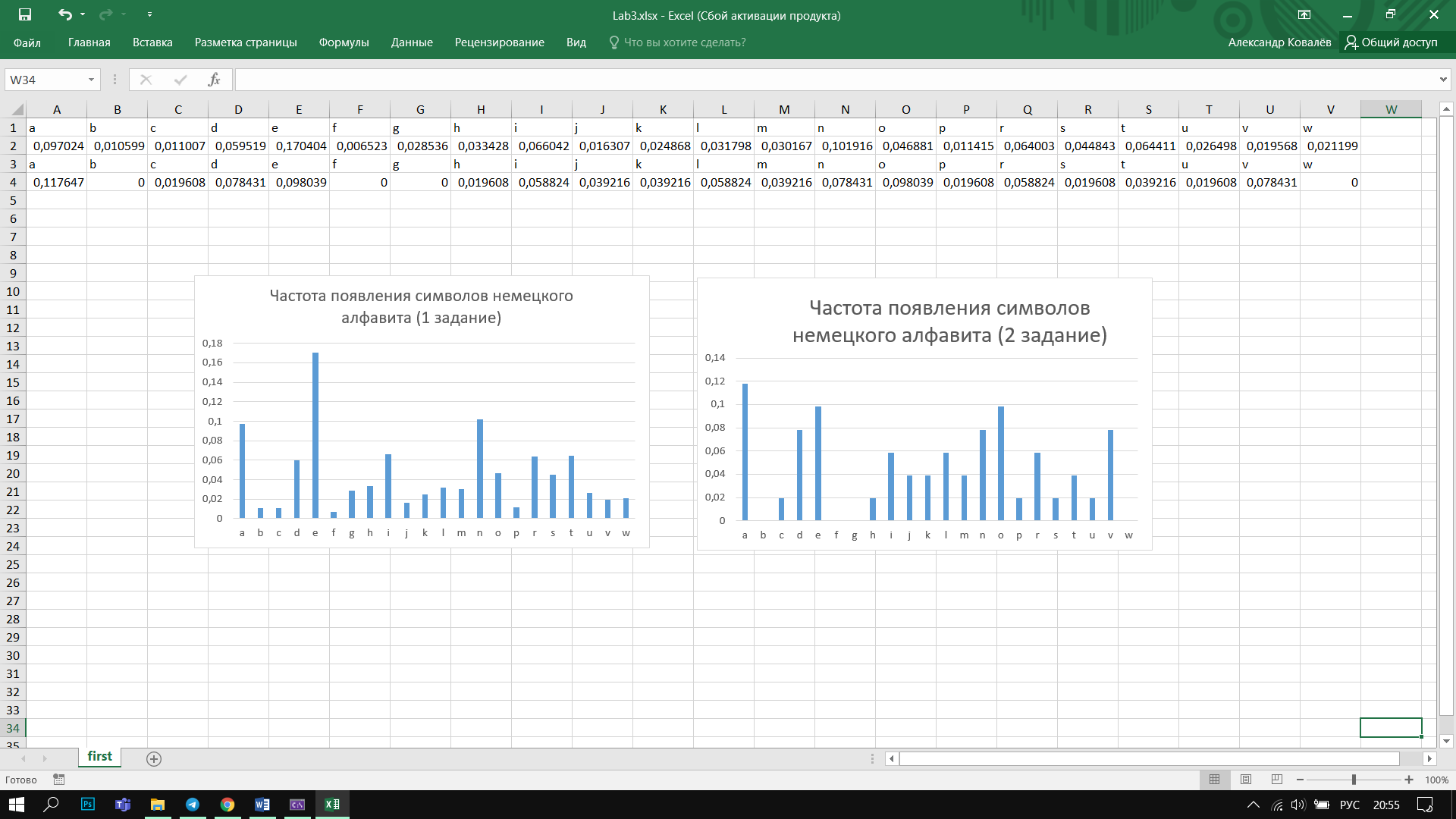


**Рисунок 3.4 – результат выполнения шифрования алгоритмом множественной перестановки**



**Рисунок 3.5 – результат выполнения шифрования алгоритмом множественной перестановки**

В результате выполнения программы формируется файл расширения «.xslx», содержащий таблицу с данными о частоте появления символов в зашифрованном сообщении. На основании представленных таблиц формируются две гистограммы, визуализирующие представленные данные. Гистограммы изображены на рисунке 3.6.



**Рисунок 3.6 – гистограммы частот появления символов в сообщениях**

**Вывод:**

В результате выполнения лабораторное работы были изучены алгоритмы шифров на основе перестановки символов и реализованы алгоритмы шифрования на основе маршрутной перестановки змейкой и множественной перестановки.