Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление 09.03.01 – «Информатика и вычислительная техника»

Дисциплина: «Безопасность и защита информации»

Профиль: «Автоматизированные системы интеллектуальной обработки данных и управления»

Семестр 6

ОТЧЕТ

по лабораторной работе №1

Тема: «Система омофонов»

Выполнил: студент группы АСУ-22-1б

Мельников Г. В. \_\_\_\_\_\_\_\_\_\_\_

Проверил: ст. преподаватель кафедры ИТАС

Шереметьев В. Г. \_\_\_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_

Пермь, 2025

**Теоретическая часть**

Система омофонов обеспечивает простейшую защиту от криптоаналитических атак, основанных на подсчете частот появления букв в шифртексте. Система омофонов является одноалфавитной, хотя при этом буквы исходного сообщения имеют несколько замен. Число замен берется пропорциональным вероятности появления буквы в открытом тексте.

Данные о распределениях вероятностей букв в русском и английском текстах приведены в таблицах. Буквы в таблицах указаны в порядке убывания вероятности их появления в тексте. Например, русская буква Е встречается в 36 раз чаще, чем буква Ф, а английская буква Е встречается в 123 раза чаще, чем буква Z.

Шифруя букву исходного сообщения, выбирают случайным образом одну из ее замен. Замены (часто называемые омофонами) могут быть представлены трехразрядными числами от 000 до 999. Например, в английском алфавите букве Е присваиваются 123 случайных номера, буквам В и G - по 16 номеров, а буквам J и Z - по 1 номеру. Если омофоны (замены) присваиваются случайным образом различным появлениям одной и той же буквы, тогда каждый омофон появляется в шифртексте равновероятно.

Распределение вероятностей букв в русских текстах

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Буква | Вероятность | Буква | Вероятность | Буква | Вероятность | Буква | Вероятность |
| Пробел | 0,175 | Р | 0,040 | Я | 0,018 | X | 0,009 |
| О | 0,090 | В | 0,038 | Ы | 0.016 | Ж | 0,007 |
| Е | 0,072 | Л | 0,035 | 3 | 0,016 | Ю | 0,006 |
| А | 0,062 | K | 0,028 | Ъ | 0,014 | Ш | 0,006 |
| И | 0,062 | M | 0,026 | Б | 0,014 | Ц | 0,004 |
| Н | 0,053 | Д | 0,025 | Г | 0,013 | Щ | 0,003 |
| Т | 0,053 | П | 0,023 | Ч | 0,012 | Э | 0,003 |
| C | 0,045 | У | 0,021 | Й | 0,010 | Ф | 0,002 |

Распределение вероятностей букв в английских текстах

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Буква | Вероятность | Буква | Вероятность | Буква | Вероятность |
| Е | 0,123 | L | 0,040 | В | 0,016 |
| Т | 0,096 | D | 0,036 | G | 0,016 |
| А | 0,081 | С | 0,032 | V | 0,009 |
| O | 0,079 | U | 0,031 | К | 0,005 |
| N | 0,072 | Р | 0,023 | Q | 0,002 |
| I | 0,071 | F | 0,023 | X | 0,002 |
| S | 0,066 | М | 0,022 | J | 0,001 |
| R | 0,060 | W | 0,020 | Z | 0,001 |
| Н | 0,051 | Y | 0,019 |  |  |

При таком подходе к формированию шифртекста простой подсчет частот уже ничего не дает криптоаналитику. Однако в принципе полезна также информация о распределении пар и троек букв в различных естественных языках. Если эту информацию использовать при криптоанализе, он будет проведен более успешно.

**Ход работы**

В Системе омофомов симметричный ключ шифрования, поэтому чтобы шифровать и расшифровывать сообщения нужно сформировать контейнер для хранения ключей. В качестве типа выбран Dictionary<char, List<string>>, где ключом является символ, а значением список возможных кодов. Коды формируются с помощью ДСЧ, а их количество для символа пропорционально его частоте появления в тексте. Было выбрано использовать 1000 кодов, от 0 до 999. Для однозначного декодирования коды дополняются незначащими нулями, таким образом, вместо 0 будет 000 и т д. Чтобы количество соответствовало заданному критерию было выбрано использовать следующую формулу:

Количество кодов = Округлить до ближайшего целого(

Количество возможных кодов / Количество символов \* Минимум(1, Частота символа \* 10));

Соотношение Количество возможных кодов / Количество символов задаёт среднее количество кодов для каждого символа, после чего данное среднее может поменяться в зависимости от частоты. Среднее нельзя превышать, иначе кодов может не хватить, поэтому коэффициент домножения ограничен сверху до 1. После одной итерации могут остаться неиспользованные коды, поэтому данная оперция выполняется в цикле, пока Количество возможных кодов > 0. Если округление до ближайшего целого выдало 0, то берётся 1, иначе возможен вечный цикл. Таким образом, после формирования контейнера сообщения можно начинать шифровать и расшифровывать. Для шифрования символа берём один из возможных его кодов с помощью ДСЧ, для расшифрования двигаемся по тройкам цифр, так как тройка – это код и по коду можно однозначно определить символ.

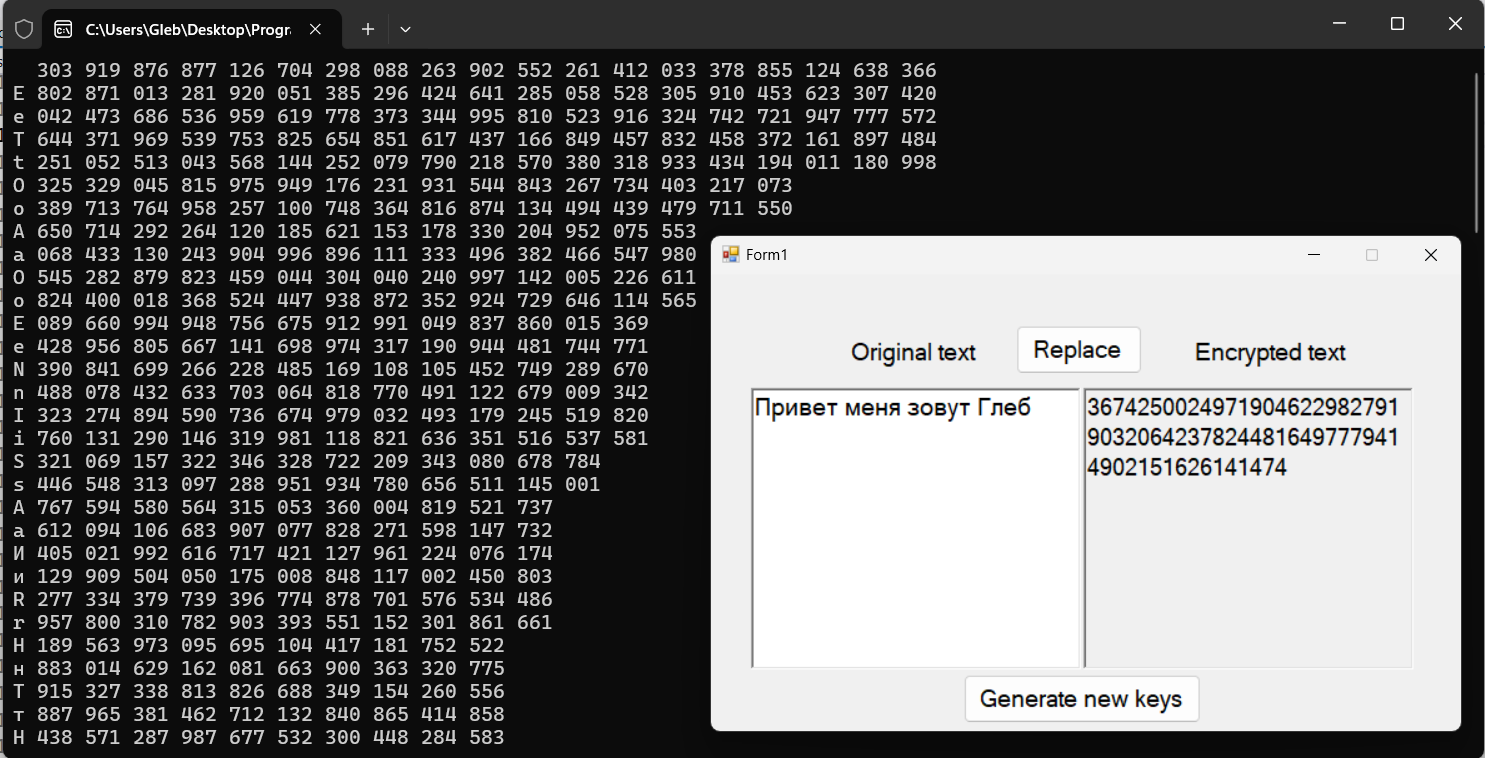
****

Рисунок 1 – пример работы 1

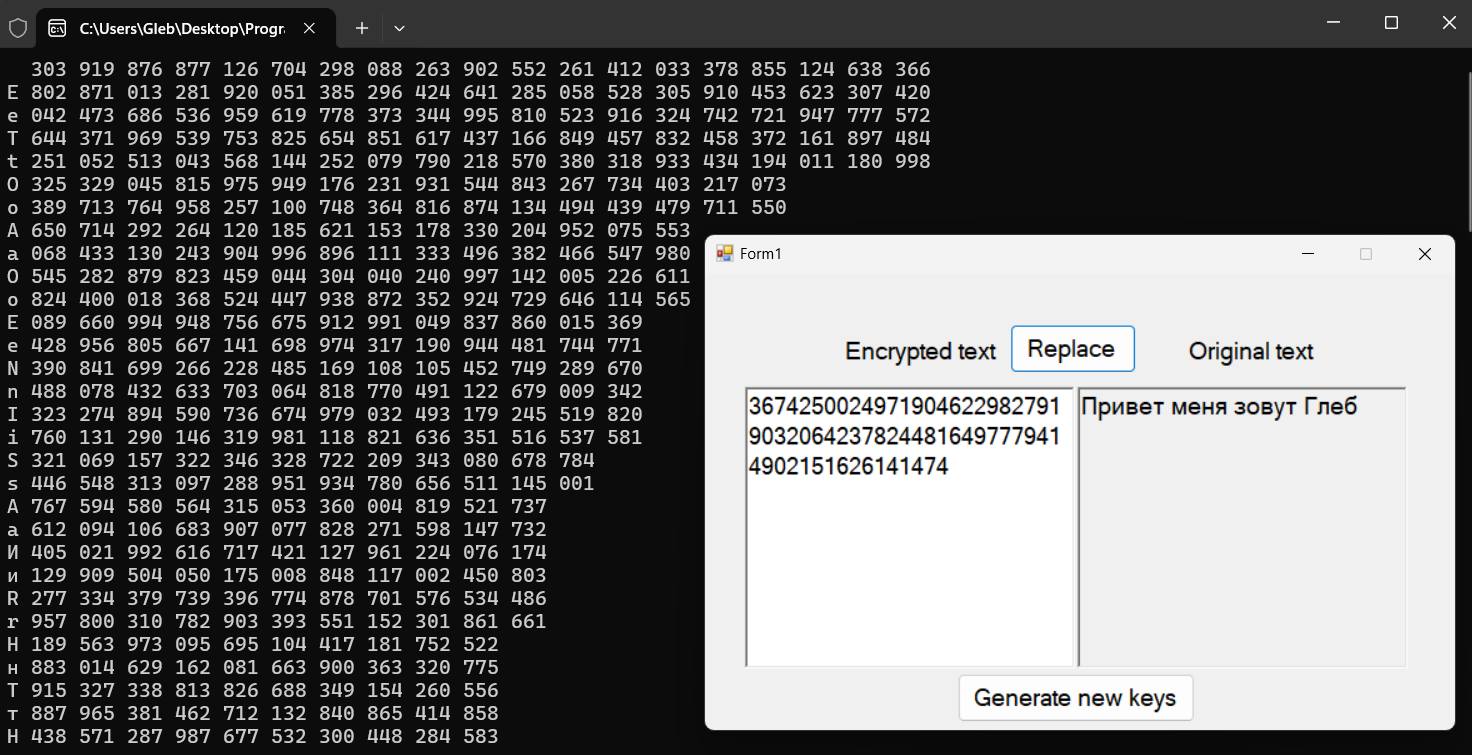


Рисунок 2 – пример использования функции “Replace”

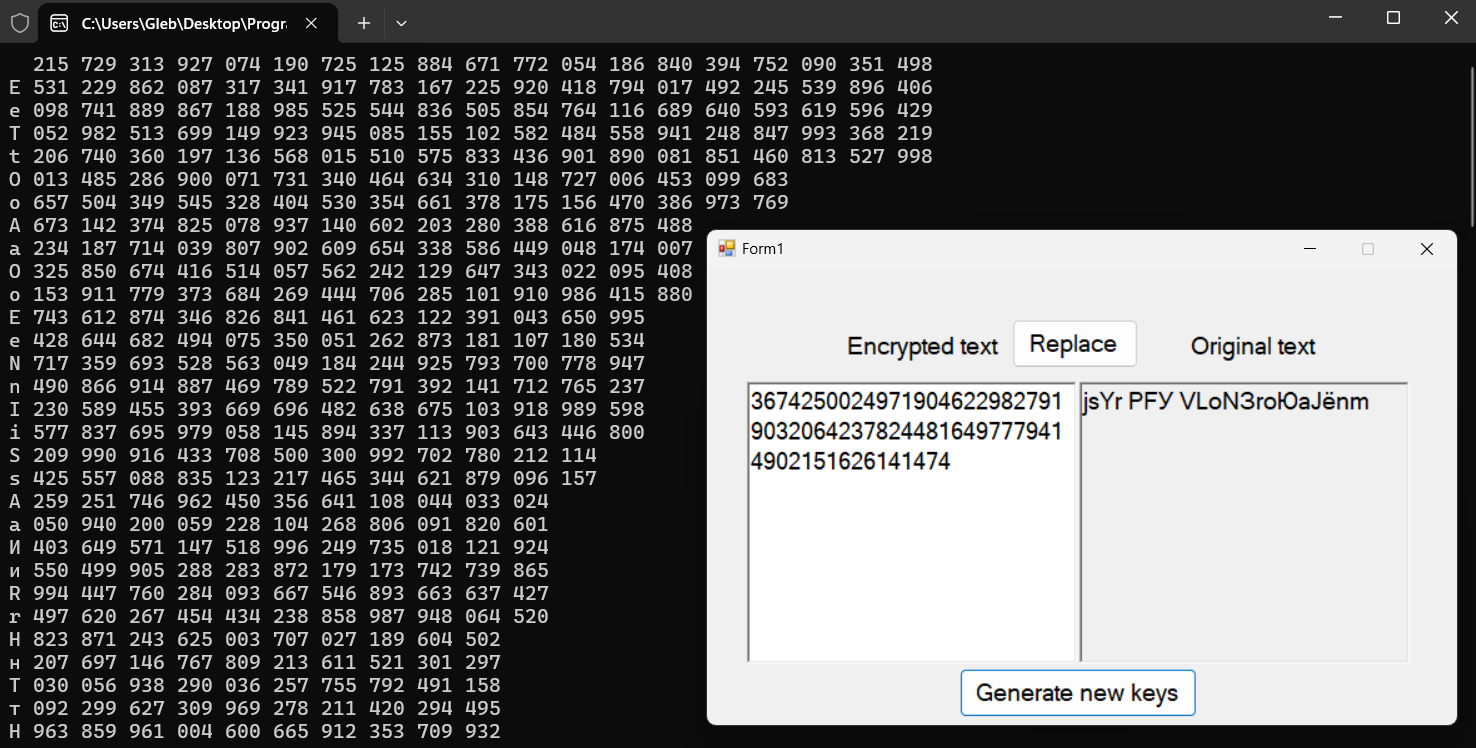


Рисунок 3 – пример использования функции “Generate new keys”

Листинг файла Form1.cs

using System;

using System.Drawing;

using System.Threading;

using System.Windows.Forms;

namespace WindowsFormsApp1

{

public partial class Form1 : Form

{

private bool isEncryptText = true;

private HomophoneSystem homophoneSystem = new HomophoneSystem();

private void FromText\_Changed(object sender, EventArgs e)

{

ConvertText();

}

private void ConvertText()

{

ToText.ForeColor = Color.Black;

ToText.Font = new Font(ToText.Font.FontFamily, 14);

try

{

ToText.Text = isEncryptText ?

EncryptText(FromText.Text) :

DecryptText(FromText.Text);

}

catch (Exception ex)

{

ToText.Text = ex.Message;

ToText.Font = new Font(ToText.Font.FontFamily, 25);

ToText.ForeColor = Color.Red;

}

}

private void ReplaceButton\_Click(object sender, EventArgs e)

{

isEncryptText = !isEncryptText;

FromText.Text = ToText.Text;

string c = FromTextLabel.Text;

FromTextLabel.Text = ToTextLabel.Text;

ToTextLabel.Text = c;

}

private string EncryptText(string text)

{

return homophoneSystem.Encrypt(text);

}

private string DecryptText(string text)

{

return homophoneSystem.Decrypt(text);

}

public Form1()

{

InitializeComponent();

}

private void GenerateNewKeysButton\_Click(object sender, EventArgs e)

{

homophoneSystem.GenerateNewKeys();

ConvertText();

}

}

}

Листинг файла HomophoneSystem.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace WindowsFormsApp1

{

internal class HomophoneSystem

{

private Random rnd = new Random();

private Dictionary<char, float> symbolFrequencyes;

public Dictionary<char, List<string>> symbolCodes;

public HomophoneSystem()

{

symbolFrequencyes =

new Dictionary<char, float>()

{

{' ', 0.175f},

{'О', 0.090f}, {'о', 0.090f},

{'Е', 0.072f}, {'е', 0.072f},

{'А', 0.062f}, {'а', 0.062f},

{'И', 0.062f}, {'и', 0.062f},

{'Н', 0.053f}, {'н', 0.053f},

{'Т', 0.053f}, {'т', 0.053f},

{'С', 0.045f}, {'с', 0.045f},

{'Р', 0.040f}, {'р', 0.040f},

{'В', 0.038f}, {'в', 0.038f},

{'Л', 0.035f}, {'л', 0.035f},

{'К', 0.028f}, {'к', 0.028f},

{'М', 0.026f}, {'м', 0.026f},

{'Д', 0.025f}, {'д', 0.025f},

{'П', 0.023f}, {'п', 0.023f},

{'У', 0.021f}, {'у', 0.021f},

{'Я', 0.018f}, {'я', 0.018f},

{'Ы', 0.016f}, {'ы', 0.016f},

{'Ь', 0.014f}, {'ь', 0.014f},

{'Б', 0.014f}, {'б', 0.014f},

{'Ч', 0.012f}, {'ч', 0.012f},

{'Й', 0.010f}, {'й', 0.010f},

{'Х', 0.009f}, {'х', 0.009f},

{'Ж', 0.007f}, {'ж', 0.007f},

{'Ю', 0.006f}, {'ю', 0.006f},

{'Ш', 0.006f}, {'ш', 0.006f},

{'Ц', 0.004f}, {'ц', 0.004f},

{'Щ', 0.003f}, {'щ', 0.003f},

{'Э', 0.003f}, {'э', 0.003f},

{'Ф', 0.002f}, {'ф', 0.002f},

{'Ё', 0.005f}, {'ё', 0.005f},

{'Г', 0.016f}, {'г', 0.016f},

{'З', 0.009f}, {'з', 0.009f},

{'Ъ', 0.014f}, {'ъ', 0.014f},

{'E', 0.123f}, {'e', 0.123f},

{'T', 0.096f}, {'t', 0.096f},

{'A', 0.081f}, {'a', 0.081f},

{'O', 0.079f}, {'o', 0.079f},

{'N', 0.072f}, {'n', 0.072f},

{'I', 0.071f}, {'i', 0.071f},

{'S', 0.066f}, {'s', 0.066f},

{'R', 0.060f}, {'r', 0.060f},

{'H', 0.051f}, {'h', 0.051f},

{'L', 0.040f}, {'l', 0.040f},

{'D', 0.036f}, {'d', 0.036f},

{'C', 0.032f}, {'c', 0.032f},

{'U', 0.031f}, {'u', 0.031f},

{'M', 0.022f}, {'m', 0.022f},

{'F', 0.022f}, {'f', 0.022f},

{'B', 0.016f}, {'b', 0.016f},

{'G', 0.016f}, {'g', 0.016f},

{'V', 0.009f}, {'v', 0.009f},

{'K', 0.005f}, {'k', 0.005f},

{'Q', 0.002f}, {'q', 0.002f},

{'X', 0.002f}, {'x', 0.002f},

{'J', 0.001f}, {'j', 0.001f},

{'Z', 0.001f}, {'z', 0.001f},

{'W', 0.019f}, {'w', 0.019f},

{'Y', 0.019f}, {'y', 0.019f},

{'P', 0.020f}, {'p', 0.020f}

};

symbolFrequencyes =

symbolFrequencyes.

OrderBy(t => -t.Value).

ToDictionary(t => t.Key, t => t.Value);

GenerateNewKeys();

}

public void GenerateNewKeys()

{

List<string> possibleCodes =

Enumerable.Range(0, 999).

Select(t => $"{new string('0', 3 - t.ToString().Length)}{t}").

ToList();

symbolCodes = new Dictionary<char, List<string>>();

while (possibleCodes.Count > 0)

{

var count = possibleCodes.Count;

foreach (var i in symbolFrequencyes)

{

var addedCodes = new List<string>();

var codesCount = Math.Round(

count / symbolFrequencyes.Count \* Math.Min(1, i.Value \* 10));

if (codesCount == 0)

{

codesCount = 1;

}

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

{

var index = rnd.Next(0, possibleCodes.Count - 1);

addedCodes.Add(possibleCodes[index]);

possibleCodes.RemoveAt(index);

}

symbolCodes[i.Key] =

!symbolCodes.ContainsKey(i.Key) ?

addedCodes :

symbolCodes[i.Key].Concat(addedCodes).ToList();

if (possibleCodes.Count == 0)

{

break;

}

}

}

#if DEBUG

Console.Clear();

Console.WriteLine("\x1b[3J");

Console.Clear();

foreach (var i in symbolCodes)

{

Console.WriteLine($"{i.Key} {string.Join(" ", i.Value)}");

}

#endif

}

public string Encrypt(string text)

{

if (text.Length == 0)

{

return string.Empty;

}

StringBuilder encryptedText = new StringBuilder();

foreach (var i in text)

{

if (!symbolCodes.ContainsKey(i))

{

throw new Exception($"An unknown '{i}' symbol has been encountered");

}

var codes = symbolCodes[i];

encryptedText.Append(codes[rnd.Next(0, codes.Count - 1)]);

}

return encryptedText.ToString();

}

public string Decrypt(string text)

{

if (text.Length % 3 != 0)

{

throw new Exception("Error. The text is not decipherable");

}

StringBuilder decryptedText = new StringBuilder();

for (int i = 0; i < text.Length; i += 3)

{

KeyValuePair<char, List<string>> pair;

try

{

pair = symbolCodes.

First(t => t.Value.Contains(text.Substring(i, 3)));

}

catch

{

throw new Exception("Error. The text is not decipherable");

}

decryptedText.Append(pair.Key);

}

return decryptedText.ToString();

}

}

}