Министерство науки и высшего образования

Российской Федерации

Федеральное государственное бюджетное образовательное   
учреждение высшего образования

«Новосибирский государственный технический университет»



Кафедра теоретической и прикладной информатики

### Лабораторная работа № 1 по дисциплине «Программные Средства Защиты Информации»

### Гаммирование. Моделирование работы скремблера

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Факультет: | ПМИ |  |  |
| Группа: | ПМИМ-01 |  |  |
| Студенты: | Ершов П. К.  Малышкина Е. Д.  Слободчикова А. Э. |  |  |
| Бригада: | 2 |  |  |
| Преподаватель: | Авдеенко Т. В. |  |  |

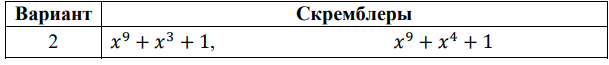
Новосибирск

2021

1. **Цель работы**

Цель работы Освоить на практике применение режима однократного гаммирования. Исследовать побитное непрерывное шифрование данных. Ознакомиться с шифрованием информации при помощи скремблера.

1. **Задание**
2. Реализовать приложение для шифрования, позволяющее выполнять следующие действия:
   1. Шифровать данные в режиме однократного гаммирования:
   2. шифруемый текст должен храниться в файле;
   3. ключ шифрования должен задаваться случайным образом;
   4. зашифрованный текст должен сохраняться в один файл, а использовавшийся при шифровании ключ – в другой;
   5. в процессе шифрования предусмотреть возможность просмотра и изменения ключа, шифруемого и зашифрованного текстов в двоичном, шестнадцатеричном и символьном виде.
   6. Шифровать данные при помощи каждого заданного в варианте скремблера:
3. шифруемый текст должен храниться в файле;
4. ключ шифрования должен задаваться случайным образом;
5. зашифрованный текст должен сохраняться в один файл, а использовавшийся при шифровании ключ – в другой;
6. в процессе шифрования предусмотреть возможность просмотра и изменения ключа, шифруемого и зашифрованного текстов в двоичном, шестнадцатеричном и символьном виде.
   1. Проводить исследование генерируемой каждым скремблером последовательности псевдослучайных чисел при заданном начальном ключе:
      1. получать период скремблера;
      2. проверять равномерность последовательности по критерию
      3. исследовать последовательность на свойства сбалансированности, цикличности, корреляции.
7. Реализовать приложение для дешифрования, позволяющее выполнять следующие действия:
   1. Дешифровать данные в режиме однократного гаммирования:
   2. шифруемый текст должен храниться в файле;
   3. ключ шифрования должен задаваться случайным образом;
   4. зашифрованный текст должен сохраняться в один файл, а использовавшийся при шифровании ключ – в другой;
   5. в процессе шифрования предусмотреть возможность просмотра и изменения ключа, шифруемого и зашифрованного текстов в двоичном, шестнадцатеричном и символьном виде.
   6. Дешифровать данные при помощи каждого заданного в варианте скремблера:
8. шифруемый текст должен храниться в файле;
9. ключ шифрования должен задаваться случайным образом;
10. зашифрованный текст должен сохраняться в один файл, а использовавшийся при шифровании ключ – в другой;
11. в процессе шифрования предусмотреть возможность просмотра и изменения ключа, шифруемого и зашифрованного текстов в двоичном, шестнадцатеричном и символьном виде.
12. С помощью реализованных приложений выполнить следующие задания:
    1. Протестировать правильность работы разработанных приложений.
    2. Определить ключ, с помощью которого зашифрованный текст может быть преобразован в некоторый осмысленный фрагмент текста, представляющий собой один из возможных вариантов прочтения открытого текста.
    3. Определить и выразить аналитически, каким образом, имея зашифрованные тексты двух телеграмм, злоумышленник может получить обе телеграммы, не зная ключа и не стремясь его определить. Привести пример.
    4. Исследовать генерируемые каждым скремблером последовательности псевдослучайных чисел при различных начальных значениях скремблера.
    5. Сделать выводы о проделанной работе
13. **Вариант**

****

1. **Исследования**
   1. Демонстрация работы программы

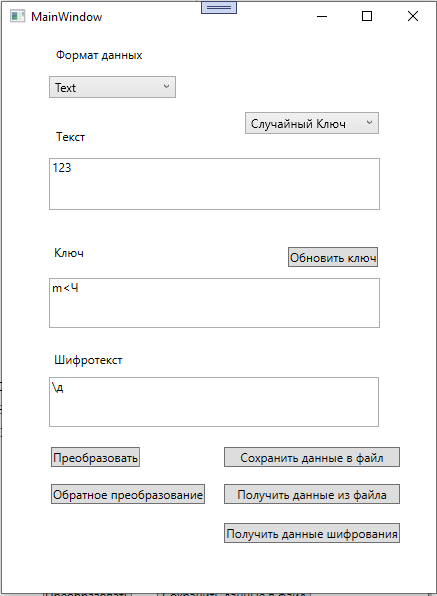


Рисунок 1. Текстовый формат данных

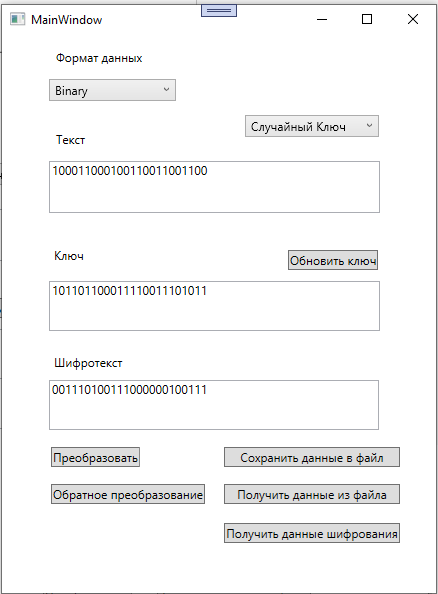


Рисунок 2. Двоичный формат данных

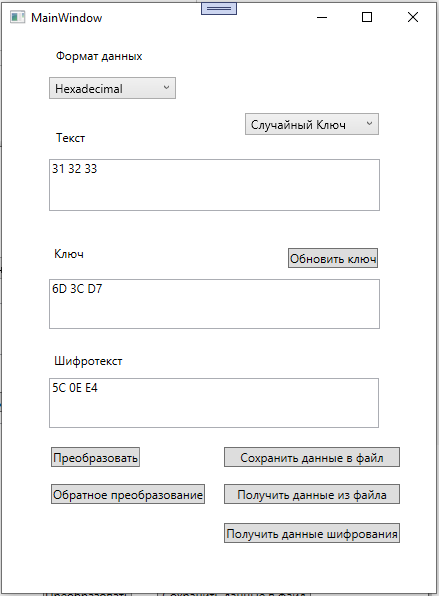


Рисунок 3. Шестнадцатеричный формат данных

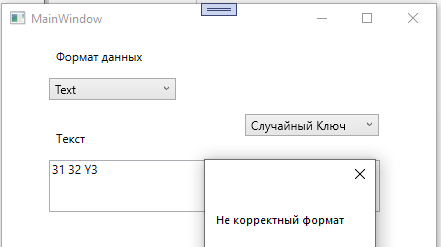


Рисунок 4. Контроль формата данных при преобразовании

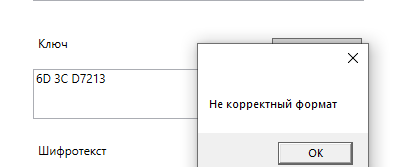


Рисунок 5. Контроль длины данных при преобразовании форматов

* 1. Гаммирование случайным ключом

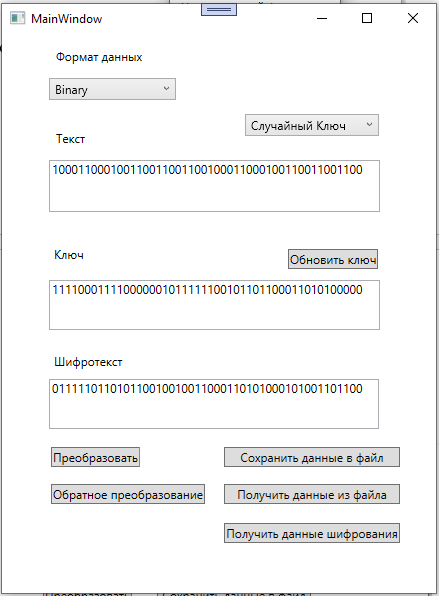


Рисунок 6. Гаммирование случайным ключом

* 1. Гаммирование скремблером
     1. Скремблер

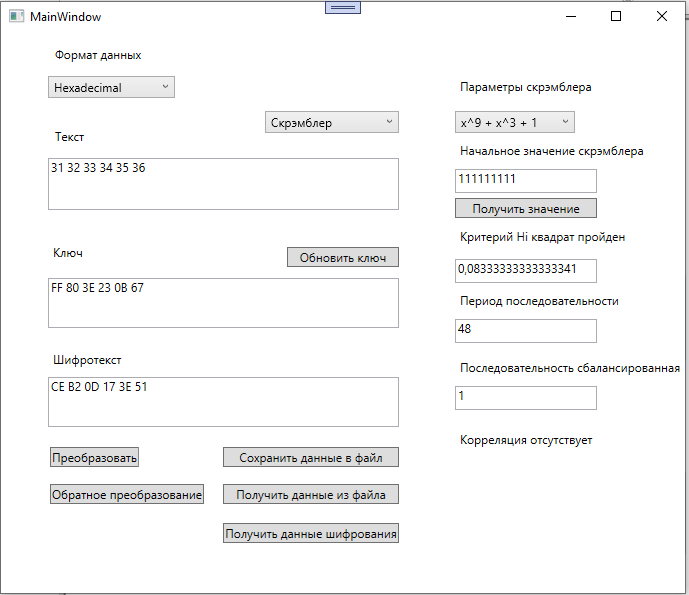


Рисунок 7. Гаммирование первым вариантом скремблера

* + 1. Скремблер

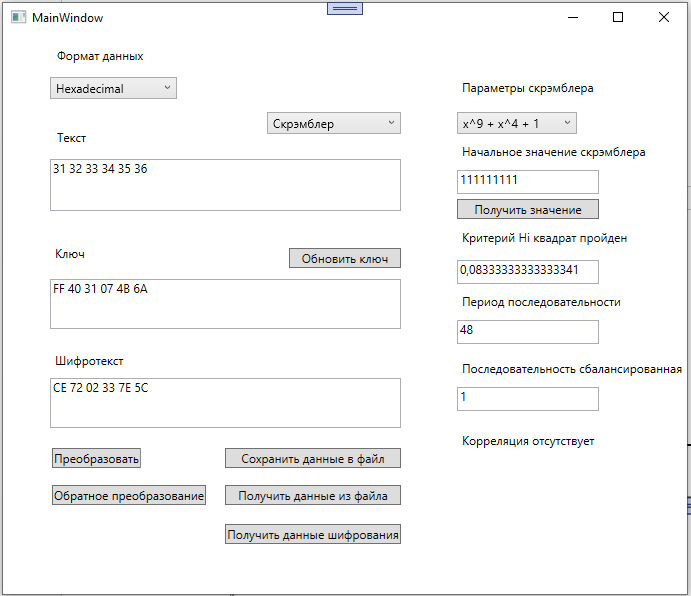


Рисунок 8. Гаммирование вторым вариантом скремблера

* 1. Определить ключ, с помощью которого зашифрованный текст может быть преобразован в некоторый осмысленный фрагмент текста, представляющий собой один из возможных вариантов прочтения открытого текста.

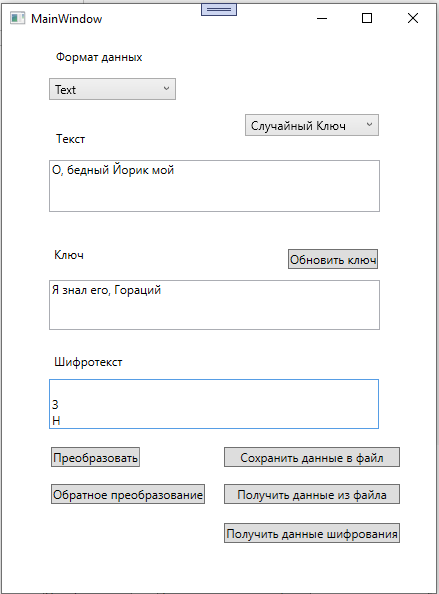


Рисунок 9. Шифруем текст осмысленной фразой

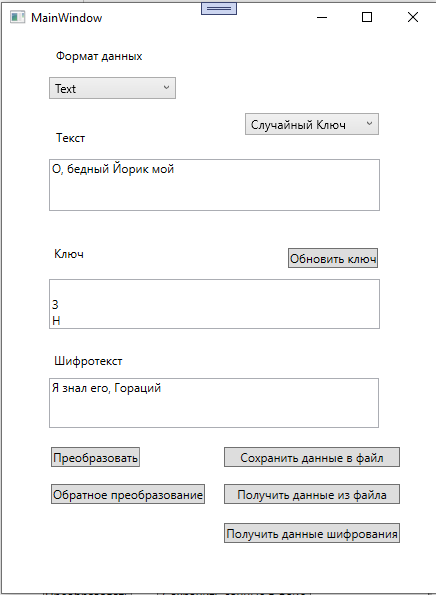


Рисунок 10. Используем полученный шифротекст как ключ и получаем из первичного текста осмысленный шифротекст

* 1. Определить и выразить аналитически, каким образом, имея зашифрованные тексты двух телеграмм, злоумышленник может получить обе телеграммы, не зная ключа и не стремясь его определить. Привести пример

Пусть имеются две телеграммы одинаковой длины, зашифрованные используя один ключ:

Тогда, исходя из свойств операции XOR можно получить:

Откуда следует, что если применить операцию XOR на зашифрованные телеграммы, результатом будет информация, в каких позициях у телеграмм совпадают символы.

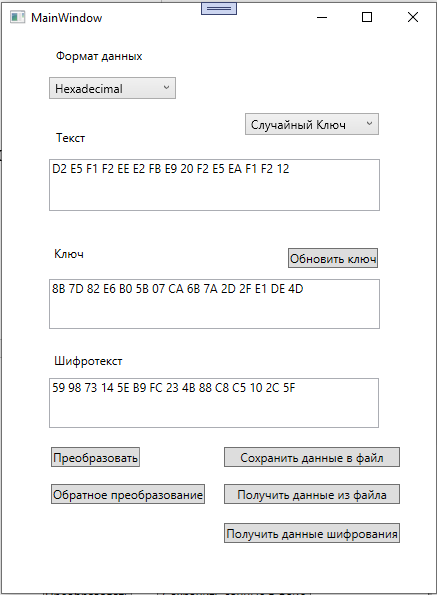


Рисунок 11. Первый шифротекст

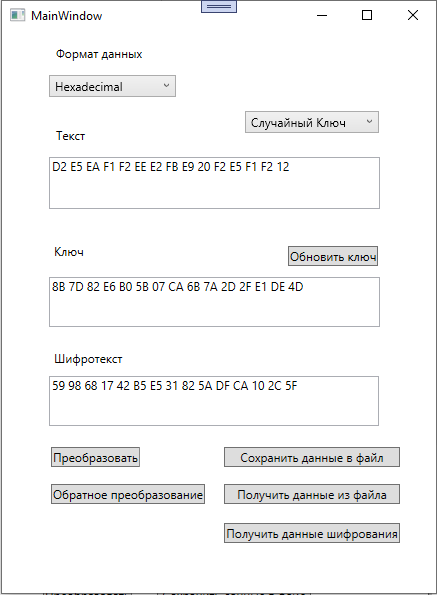


Рисунок 12. Второй шифротекст

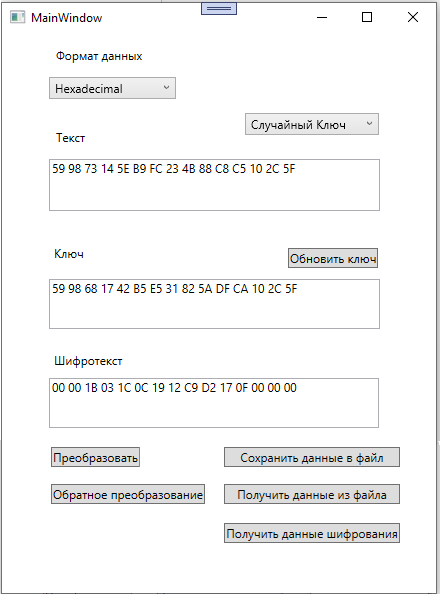


Рисунок 13. Результат операции XOR для двух шифротекстов, образованных одним ключом

Например, пусть нам известно, что первое сообщение имеет в себе приветствие из 5 букв. Таких слов не много, поэтому возьмем то, что чаще встречается - слово ”hello”. Тогда, исходя из результата операции XOR, можно заметить, что второе сообщение начинается с тех же 5 символов, что и первое сообщение. Также, допустим нам известно, что получатель обоих сообщений один человек и он любит творчество Mobb Deep, написавших композицию ”Hell on Earth”. Исходя из этой информации можно предположить, что первое сообщение содержало приветствие адресованное получателю, в виде ”Hello name а второе сообщение содержало название этой композиции без использования отступа между словами.

* 1. Результаты работы скремблера при разных начальных значениях

|  |  |  |  |
| --- | --- | --- | --- |
| Скремблер  Начальное значение регистра |  | |  |
| 000000001 | Значение ключа | 80 BF 60 6E 7A 4C | 80 5F 67 7C DA 4A |
|  | |  |
| 000000010 | Значение ключа | C0 5F 30 37 3D A6 | C0 AF 33 3E 6D A5 |
|  | |  |
| 000001100 | Значение ключа | F0 17 CC 4D 8F A9 | F0 EB 8C 4F 5B E9 |
|  | |  |
| 000101100 | Значение ключа | FC 05 73 D3 63 6A | FC 3A E3 D3 56 BA |
|  | |  |
| 111111111 | Значение ключа | FF 80 3E 23 0B 67 | FF 40 31 07 4B 6A |
|  | |  |

1. **Выводы**

В ходе выполнения данной лабораторной работы было освоено на практике применение режима однократного гаммирования; исследовано побитное непрерывное шифрование данных; а также изучено шифрование информации при помощи скремблера.

Алгоритм гаммирования, в основе которого лежит идея, что не зная ключа шифрования, невозможно получить исходное сообщение, действительно работает, если для каждого следующего сообщения генерировать новый ключ необходимой длинны, где для каждого бита вероятность равна 0.5, из-за чего даже при полном переборе ключей оставит большое количество вариантов текста.

Повторное использование ключа наоборот поможет злоумышленнику сократить количество вариантов текста при переборе, а при наличии нескольких сообщений, зашифрованных одинаковым ключом, сделает возможным расшифровку обоих сообщений без знания этого ключа.

1. **Код программы**

**GammaCrypt.cs**

public static class GammaCrypt

{

public static byte[] RandKey(int leng) //Случайный ключ

{

Random rnd = new Random((int)DateTime.Now.Ticks);

var bt = new Byte[leng];

rnd.NextBytes(bt);

return bt;

}

public static byte[] Encryptor(byte[] text, byte[] key) //Шифровщик/дешифровщик

{

Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);

var cipherText = new byte[text.Length];

new BitArray(key).Xor(new BitArray(text)).CopyTo(cipherText, 0);

return cipherText;

}

public static (string output, int code) Gamming(string Text, string Key, string flag = "Text") //Универсальный преобразователь

{

string chiphrtext = "";

int code = 0;

if (flag == "Text")

{

if (Text.Length == Key.Length)

{

var text = ConverteUtility.ConvertStringToByteArray(Text);

var key = ConverteUtility.ConvertStringToByteArray(Key);

var tt = Encryptor(text, key);

chiphrtext = ConverteUtility.ConvertByteArrayToString(tt);

}

else

code = 3;

}

if (flag == "Binary")

{

if (ConverteUtility.CheckIncorrectFormat(Text, "Bin") && ConverteUtility.CheckIncorrectFormat(Key, "Bin"))

{

if (ConverteUtility.CheckIncorrectLength(Text) && ConverteUtility.CheckIncorrectLength(Key))

{

if (Text.Length == Key.Length)

{

var text = ConverteUtility.ConvertBinaryStrToByte(Text);

var key = ConverteUtility.ConvertBinaryStrToByte(Key);

chiphrtext = ConverteUtility.ConvertByteArraToBinaryStr(Encryptor(text, key));

}

else

code = 3;

}

else

code = 2;

}

else

code = 1;

}

if (flag == "Hexadecimal")

{

if (ConverteUtility.CheckIncorrectFormat(Text, "Hex") && ConverteUtility.CheckIncorrectFormat(Key, "Hex"))

{

if (ConverteUtility.CheckIncorrectLength(Text) && ConverteUtility.CheckIncorrectLength(Key))

{

if (Text.Length == Key.Length)

{

var text = ConverteUtility.HexStringToByteArray(Text);

var key = ConverteUtility.HexStringToByteArray(Key);

chiphrtext = ConverteUtility.ByteArrayToHexString(Encryptor(text, key));

}

else

code = 3;

}

else

code = 2;

}

else

code = 1;

}

return (chiphrtext, code);

}

public static int Peroid(string seq) //Вычисление периода

{

int per = 1;

int step = 0;

while (step + per != seq.Length)

{

if (seq[step] != seq[per + step])

{

++per;

step = 0;

}

else

{

++step;

}

}

return per;

}

}

**ScammblerClass.cs**

public static class ScammblerClass

{

public static uint[] LFSR\_one(int Length, uint start, int format = 1) //Скремблер

{

uint[] output = new uint[Length];

var ShiftRegister = start;

if(format == 0)

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

{

ShiftRegister = ((((ShiftRegister >> 9) ^ (ShiftRegister >> 3) ^ ShiftRegister) & 0x001) << 9) | (ShiftRegister >> 1);

output[i] = ShiftRegister | 0x01;

}

else

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

{

ShiftRegister = ((((ShiftRegister >> 9) ^ (ShiftRegister >> 4) ^ ShiftRegister) & 0x001) << 9) | (ShiftRegister >> 1);

output[i] = ShiftRegister | 0x01;

}

return output;

}

public static double Hi2(string seq) //Критерий Хи квадрат

{

int n = seq.Length;

int z = 0, o = 0;

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

if (seq[i] == '0')

z++;

else

o++;

double s = 2.0D \* ((double)(n));

s \*= Math.Pow(((double)z) / ((double)n) - 0.5, 2.0) + Math.Pow(((double)o) / ((double)n) - 0.5, 2.0);

return s;

}

public static (bool flag, double bal) Balance(string seq) //Тест на сбалансированность

{

bool flag = true;

int interval = 1000;

int index = 0;

int n = seq.Length;

var bal = 0.0;

for (int j = 0; flag && index < n; j++)

{

int z = 0, o = 0;

for (int i = j \* interval; flag && i < interval \* (j + 1); i++)

{

index++;

if (seq[j] == '0')

z++;

else

o++;

}

bal = (double)Math.Abs(z - o) / interval;

if (bal > 0.05)

flag = false;

}

return (flag, bal);

}

public static (bool flag, double cor) Correlation(string seq) //Тест на корреляции

{

bool flag = true;

int pl = 0;

int mi = 0;

int n = seq.Length;

int sdvig = 5;

for (int i = sdvig; i < n - sdvig; i++)

{

if (seq[i] == seq[i + sdvig])

pl++;

else

mi++;

}

if ((double)Math.Abs(pl - mi) / (n - sdvig - sdvig) > 0.05)

flag = false;

return (flag, (double)Math.Abs(pl - mi) / (n - sdvig - sdvig));

}

}

**ConverteUtility.cs**

public static class ConverteUtility

{

private const string AllowedСharHex = "0123456789ABCDEF";

private const string AllowedСharBin = "01";

private static Encoding enc = Encoding.GetEncoding(1251);

public static bool CheckIncorrectFormat(string text, string format = "Bin")

{

bool flag = false;

string sample = "";

string buff = text.Replace(" ", "");

switch (format)

{

case "Bin":

{

sample = AllowedСharBin;

break;

}

case "Hex":

{

sample = AllowedСharHex;

break;

}

}

foreach (var i in buff)

{

flag = false;

foreach (var j in sample)

{

if (i == j)

{

flag = true;

break;

}

}

if (!flag)

break;

}

return flag;

}

public static bool CheckIncorrectLength(string text)

{

var buff = text.Replace(" ", "");

if (buff.Length % 2 != 0)

return false;

else

return true;

}

public static string GenStartVal(int leng)

{

var output = "";

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

output += "0";

return output;

}

public static string ByteArrayToHexString(byte[] Bytes)

{

return BitConverter.ToString(Bytes).Replace("-", " ");

}

public static byte[] HexStringToByteArray(string HexStr)

{

var Hex = HexStr.Replace(" ", "");

byte[] Bytes = new byte[Hex.Length / 2];

int[] HexValue = new int[] { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,

0x08, 0x09, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F };

for (int x = 0, i = 0; i < Hex.Length; i += 2, x += 1)

{

Bytes[x] = (byte)(HexValue[Char.ToUpper(Hex[i + 0]) - '0'] << 4 |

HexValue[Char.ToUpper(Hex[i + 1]) - '0']);

}

return Bytes;

}

public static byte[] ConvertStringToByteArray(string text)

{

Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);

return enc.GetBytes(text);

}

public static string ConvertByteArrayToString(byte[] text)

{

Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);

return enc.GetString(text);

}

public static string ConvertStringToBinaryStr(string text)

{

Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);

var s = new StringBuilder();

foreach (bool bb in new BitArray(enc.GetBytes(text)))

s.Append(bb ? '1' : '0');

return s.ToString();

}

public static string ConvertByteArraToBinaryStr(byte[] text)

{

Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);

var s = new StringBuilder();

foreach (bool bb in new BitArray(text))

s.Append(bb ? '1' : '0');

return s.ToString();

}

public static string UniConvert(string text, string inflag = "Text", string outflag = "Text")

{

string output = "";

byte[] buff = new byte[text.Length];

if (inflag == outflag)

return text;

switch (inflag)

{

case "Text":

{

buff = ConvertStringToByteArray(text);

break;

}

case "Binary":

{

buff = ConvertBinaryStrToByte(text);

break;

}

case "Hexadecimal":

{

buff = HexStringToByteArray(text);

break;

}

}

switch (outflag)

{

case "Text":

{

output = ConvertByteArrayToString(buff);

break;

}

case "Binary":

{

output = ConvertByteArraToBinaryStr(buff);

break;

}

case "Hexadecimal":

{

output = ByteArrayToHexString(buff);

break;

}

}

return output;

}

public static byte[] ConvertBinaryStrToByte(string binary)

{

Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);

int numOfBytes = binary.Length / 8;

byte[] bytes = new byte[numOfBytes];

var buf = new BitArray(bytes);

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

{

if (binary[i] == '1')

buf[i] = true;

else

buf[i] = false;

}

buf.CopyTo(bytes, 0);

return bytes;

}

public static string PadToByte(string binary)

{

string output = "";

if (binary.Length < 8)

{

for (int i = 0; i < 8 - binary.Length; i++)

output += "0";

output += binary;

return output;

}

if (binary.Length % 8 != 0)

{

for (int i = 0; i < (binary.Length / 8 + 1) \* 8 - binary.Length; i++)

output += "0";

output += binary;

return output;

}

return binary;

}

public static string GetScramKey(uint[] scrambler)

{

var s = new StringBuilder();

foreach (var i in scrambler)

{

var ttt = Convert.ToInt32(i);

var b = BitConverter.GetBytes(Convert.ToInt16(ttt));

byte[] b1 = new byte[1] {b[0]};

var buf = new BitArray(b1);

var t = buf.Length - 1;

s.Append(buf.Get(t) ? '1' : '0');

}

return s.ToString();

}

}

**FileUtility.cs**

public static class FileUtility

{

public static string RootDirectory = Directory.GetCurrentDirectory();

public static DataModel DeserializeString(string filename) => JsonSerializer.Deserialize<DataModel>(filename);

public static string Serialize(DataModel Data) => JsonSerializer.Serialize(Data);

public static void JSONSave(string filename, string text) => File.WriteAllText($"{Directory.GetCurrentDirectory()}\\Resources\\{filename}", text);

public static string JSONSrt(string filename) => File.ReadAllText($"{Directory.GetCurrentDirectory()}\\Resources\\{filename}");

}

**MainWindow.xaml.cs**

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private string TextFormarFlag = "Text";

private string KeyFormarFlag = "Rand";

private string ResourceFile = "Data.json";

private string ScramblerKey = "";

private double HiCrit = 3.842;

private void Grid\_Loaded(object sender, RoutedEventArgs e)

{

Width = 450;

}

private void TextFormat\_DropDownClosed(object sender, EventArgs e)

{

if (TextFormarFlag == "Text")

{

Text.Text = ConverteUtility.UniConvert(Text.Text, TextFormarFlag, TextFormat.Text);

Key.Text = ConverteUtility.UniConvert(Key.Text, TextFormarFlag, TextFormat.Text);

Chiphrtext.Text = ConverteUtility.UniConvert(Chiphrtext.Text, TextFormarFlag, TextFormat.Text);

}

if (TextFormarFlag == "Binary")

if (ConverteUtility.CheckIncorrectFormat(Text.Text, "Bin") && ConverteUtility.CheckIncorrectLength(Text.Text))

{

Text.Text = ConverteUtility.UniConvert(Text.Text, TextFormarFlag, TextFormat.Text);

Key.Text = ConverteUtility.UniConvert(Key.Text, TextFormarFlag, TextFormat.Text);

Chiphrtext.Text = ConverteUtility.UniConvert(Chiphrtext.Text, TextFormarFlag, TextFormat.Text);

}

else

{

MessageBox.Show("Не корректный формат");

TextFormat.SelectedIndex = 1;

}

if (TextFormarFlag == "Hexadecimal")

if (ConverteUtility.CheckIncorrectFormat(Text.Text, "Hex") && ConverteUtility.CheckIncorrectLength(Text.Text))

{

Text.Text = ConverteUtility.UniConvert(Text.Text, TextFormarFlag, TextFormat.Text);

Key.Text = ConverteUtility.UniConvert(Key.Text, TextFormarFlag, TextFormat.Text);

Chiphrtext.Text = ConverteUtility.UniConvert(Chiphrtext.Text, TextFormarFlag, TextFormat.Text);

}

else

{

MessageBox.Show("Не корректный формат");

TextFormat.SelectedIndex = 2;

}

}

private void TextFormat\_DropDownOpened(object sender, EventArgs e)

{

TextFormarFlag = TextFormat.Text;

}

private void CiphButton\_Click(object sender, RoutedEventArgs e)

{

var encryptResults = GammaCrypt.Gamming(Text.Text, Key.Text, TextFormat.Text);

if (encryptResults.code == 0)

Chiphrtext.Text = encryptResults.output;

if (encryptResults.code == 3)

MessageBox.Show($"Длины текста и ключа не совпадают: Text = {Text.Text.Length}, Key = {Key.Text.Length}");

if (encryptResults.code == 2)

MessageBox.Show("Не корректная длина текста или ключа");

if (encryptResults.code == 1)

MessageBox.Show("Не корректный формат текста или ключа");

}

private void UpdateKey\_Click(object sender, RoutedEventArgs e)

{

if (KeyFormarFlag == "Rand")

{

if (TextFormat.Text == "Text")

Key.Text = ConverteUtility.ConvertByteArrayToString(GammaCrypt.RandKey(Text.Text.Length));

if (TextFormat.Text == "Binary")

Key.Text = ConverteUtility.ConvertByteArraToBinaryStr(GammaCrypt.RandKey(Text.Text.Length));

if (TextFormat.Text == "Hexadecimal")

Key.Text = ConverteUtility.ByteArrayToHexString(GammaCrypt.RandKey(Text.Text.Length));

}

if (KeyFormarFlag == "Scrambler")

{

if (TextFormat.Text == "Text")

{

var start = ConverteUtility.PadToByte(ScramStart.Text);

var startbin = ConverteUtility.ConvertBinaryStrToByte(start);

var bb = BitConverter.ToUInt16(startbin, 0);

var rt = new BitArray(ConverteUtility.ConvertStringToByteArray(Text.Text));

var b = ScammblerClass.LFSR\_one(rt.Length, Convert.ToUInt32(bb), ScramFormat.SelectedIndex);

ScramblerKey = ConverteUtility.GetScramKey(b);

Key.Text = ConverteUtility.ConvertByteArrayToString(ConverteUtility.ConvertBinaryStrToByte(ScramblerKey));

var hi = ScammblerClass.Hi2(ScramblerKey);

var baltest = ScammblerClass.Balance(ScramblerKey);

var corrtest = ScammblerClass.Correlation(ScramblerKey);

Period.Text = GammaCrypt.Peroid(ScramblerKey).ToString();

Hi2Test.Text = hi.ToString();

BalansTest.Text = baltest.bal.ToString();

if (hi <= HiCrit)

HiLabel.Content = "Критерий Hi квадрат пройден";

else

HiLabel.Content = "Критерий Hi квадрат не пройден";

if (baltest.flag)

BalansTestLabel.Content = "Последовательность сбалансированная";

else

BalansTestLabel.Content = "Последовательность сбалансированная";

if (corrtest.flag)

CorrelTestLabel.Content = "Корреляция присутствует";

else

CorrelTestLabel.Content = "Корреляция отсутствует";

}

if (TextFormat.Text == "Binary")

{

var start = ConverteUtility.PadToByte(ScramStart.Text);

var startbin = ConverteUtility.ConvertBinaryStrToByte(start);

var bb = BitConverter.ToUInt16(startbin, 0);

var rt = new BitArray(ConverteUtility.ConvertStringToByteArray(Text.Text));

var b = ScammblerClass.LFSR\_one(rt.Length, Convert.ToUInt32(bb), ScramFormat.SelectedIndex);

ScramblerKey = ConverteUtility.GetScramKey(b);

Key.Text = ScramblerKey;

var hi = ScammblerClass.Hi2(ScramblerKey);

var baltest = ScammblerClass.Balance(ScramblerKey);

var corrtest = ScammblerClass.Correlation(ScramblerKey);

Period.Text = GammaCrypt.Peroid(ScramblerKey).ToString();

Hi2Test.Text = hi.ToString();

BalansTest.Text = baltest.bal.ToString();

if (hi <= HiCrit)

HiLabel.Content = "Критерий Hi квадрат пройден";

else

HiLabel.Content = "Критерий Hi квадрат не пройден";

if (baltest.flag)

BalansTestLabel.Content = "Последовательность сбалансированная";

else

BalansTestLabel.Content = "Последовательность сбалансированная";

if (corrtest.flag)

CorrelTestLabel.Content = "Корреляция присутствует";

else

CorrelTestLabel.Content = "Корреляция отсутствует";

}

if (TextFormat.Text == "Hexadecimal")

{

var start = ConverteUtility.PadToByte(ScramStart.Text);

var startbin = ConverteUtility.ConvertBinaryStrToByte(start);

var bb = BitConverter.ToUInt16(startbin, 0);

var rt = new BitArray(ConverteUtility.ConvertStringToByteArray(Text.Text));

var b = ScammblerClass.LFSR\_one(rt.Length, Convert.ToUInt32(bb), ScramFormat.SelectedIndex);

ScramblerKey = ConverteUtility.GetScramKey(b);

Key.Text = ConverteUtility.ByteArrayToHexString(ConverteUtility.ConvertBinaryStrToByte(ScramblerKey));

var hi = ScammblerClass.Hi2(ScramblerKey);

var baltest = ScammblerClass.Balance(ScramblerKey);

var corrtest = ScammblerClass.Correlation(ScramblerKey);

Period.Text = GammaCrypt.Peroid(ScramblerKey).ToString();

Hi2Test.Text = hi.ToString();

BalansTest.Text = baltest.bal.ToString();

if (hi <= HiCrit)

HiLabel.Content = "Критерий Hi квадрат пройден";

else

HiLabel.Content = "Критерий Hi квадрат не пройден";

if (baltest.flag)

BalansTestLabel.Content = "Последовательность сбалансированная";

else

BalansTestLabel.Content = "Последовательность сбалансированная";

if (corrtest.flag)

CorrelTestLabel.Content = "Корреляция присутствует";

else

CorrelTestLabel.Content = "Корреляция отсутствует";

}

}

}

private void KeyType\_DropDownClosed(object sender, EventArgs e)

{

if (KeyType.Text == "Случайный Ключ")

{

Width = 450;

KeyFormarFlag = "Rand";

}

else

{

Width = 700;

KeyFormarFlag = "Scrambler";

ScramStart.Text += ConverteUtility.GenStartVal(9);

}

}

private void SaveFile\_Click(object sender, RoutedEventArgs e)

{

var data = new DataModel();

data.text = Text.Text;

data.key = Key.Text;

data.chiphr = Chiphrtext.Text;

var str = FileUtility.Serialize(data);

FileUtility.JSONSave(ResourceFile, str);

}

private void FileLoad\_Click(object sender, RoutedEventArgs e)

{

Text.Clear();

Key.Clear();

Chiphrtext.Clear();

var data = FileUtility.DeserializeString(FileUtility.JSONSrt(ResourceFile));

Text.Text = data.text;

Key.Text = data.key;

Chiphrtext.Text = data.chiphr;

}

private void ScramStart\_TextChanged(object sender, System.Windows.Controls.TextChangedEventArgs e)

{

if(ScramStart.Text.Length != 9 || !ConverteUtility.CheckIncorrectFormat(ScramStart.Text, "Bin"))

ScramStart.Text = ConverteUtility.GenStartVal(9);

}

}