

Министерство науки и высшего образования
Российской Федерации

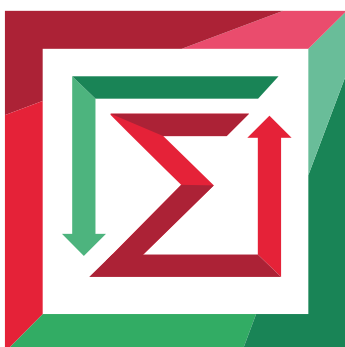
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ»



**НГТУ
НЭТИ**

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:	Авдеев 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.

2. Задание

1. Реализовать приложение для шифрования, позволяющее выполнять следующие действия:

1.1. Шифровать данные в режиме однократного гаммирования:

- 1) шифруемый текст должен храниться в файле;
- 2) ключ шифрования должен задаваться случайным образом;
- 3) зашифрованный текст должен сохраняться в один файл, а использовавшийся при шифровании ключ – в другой;
- 4) в процессе шифрования предусмотреть возможность просмотра и изменения ключа, шифруемого и зашифрованного текстов в двоичном, шестнадцатеричном и символьном виде.

1.2. Шифровать данные при помощи каждого заданного в варианте скремблера:

- 1) шифруемый текст должен храниться в файле;
- 2) ключ шифрования должен задаваться случайным образом;
- 3) зашифрованный текст должен сохраняться в один файл, а использовавшийся при шифровании ключ – в другой;
- 4) в процессе шифрования предусмотреть возможность просмотра и изменения ключа, шифруемого и зашифрованного текстов в двоичном, шестнадцатеричном и символьном виде.

1.3. Проводить исследование генерируемой каждым скремблером последовательности псевдослучайных чисел при заданном начальном ключе:

- 1) получать период скремблера;
- 2) проверять равномерность последовательности по критерию χ^2
- 3) исследовать последовательность на свойства сбалансированности, цикличности, корреляции.

2. Реализовать приложение для дешифрования, позволяющее выполнять следующие действия:
 - 2.1. Дешифровать данные в режиме однократного гаммирования:
 - 5) шифруемый текст должен храниться в файле;
 - 6) ключ шифрования должен задаваться случайным образом;
 - 7) зашифрованный текст должен сохраняться в один файл, а использовавшийся при шифровании ключ – в другой;
 - 8) в процессе шифрования предусмотреть возможность просмотра и изменения ключа, шифруемого и зашифрованного текстов в двоичном, шестнадцатеричном и символьном виде.
 - 2.2. Дешифровать данные при помощи каждого заданного в варианте скремблера:
 - 5) шифруемый текст должен храниться в файле;
 - 6) ключ шифрования должен задаваться случайным образом;
 - 7) зашифрованный текст должен сохраняться в один файл, а использовавшийся при шифровании ключ – в другой;
 - 8) в процессе шифрования предусмотреть возможность просмотра и изменения ключа, шифруемого и зашифрованного текстов в двоичном, шестнадцатеричном и символьном виде.
3. С помощью реализованных приложений выполнить следующие задания:
 - 3.1. Протестировать правильность работы разработанных приложений.
 - 3.2. Определить ключ, с помощью которого зашифрованный текст может быть преобразован в некоторый осмысленный фрагмент текста, представляющий собой один из возможных вариантов прочтения открытого текста.
 - 3.3. 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.
 - 3.4. Исследовать генерируемые каждым скремблером последовательности псевдослучайных чисел при различных начальных значениях скремблера.
 - 3.5. Сделать выводы о проделанной работе

3. Вариант

Вариант	Скремблеры	
2	$x^9 + x^3 + 1,$	$x^9 + x^4 + 1$

4. Исследования

4.1. Демонстрация работы программы

The screenshot shows a window titled 'MainWindow' with standard Windows controls (minimize, maximize, close). The interface is divided into several sections:

- Формат данных (Data Format):** A dropdown menu currently set to 'Text'.
- Текст (Text):** A text input field containing the value '123'.
- Ключ (Key):** A text input field containing the value 'm<4'. To its right is a button labeled 'Обновить ключ' (Update key).
- Шифротекст (Ciphertext):** A text input field containing the value '\d'.
- Buttons:** At the bottom, there are five buttons arranged in three rows:
 - Row 1: 'Преобразовать' (Transform) and 'Сохранить данные в файл' (Save data to file).
 - Row 2: 'Обратное преобразование' (Inverse transformation) and 'Получить данные из файла' (Get data from file).
 - Row 3: 'Получить данные шифрования' (Get encryption data).

Рисунок 1. Текстовый формат данных

MainWindow

Формат данных

Binary

Текст

Случайный Ключ

100011000100110011001100

Ключ

Обновить ключ

101101100011110011101011

Шифротекст

001110100111000000100111

Преобразовать

Сохранить данные в файл

Обратное преобразование

Получить данные из файла

Получить данные шифрования

Рисунок 2. Двоичный формат данных

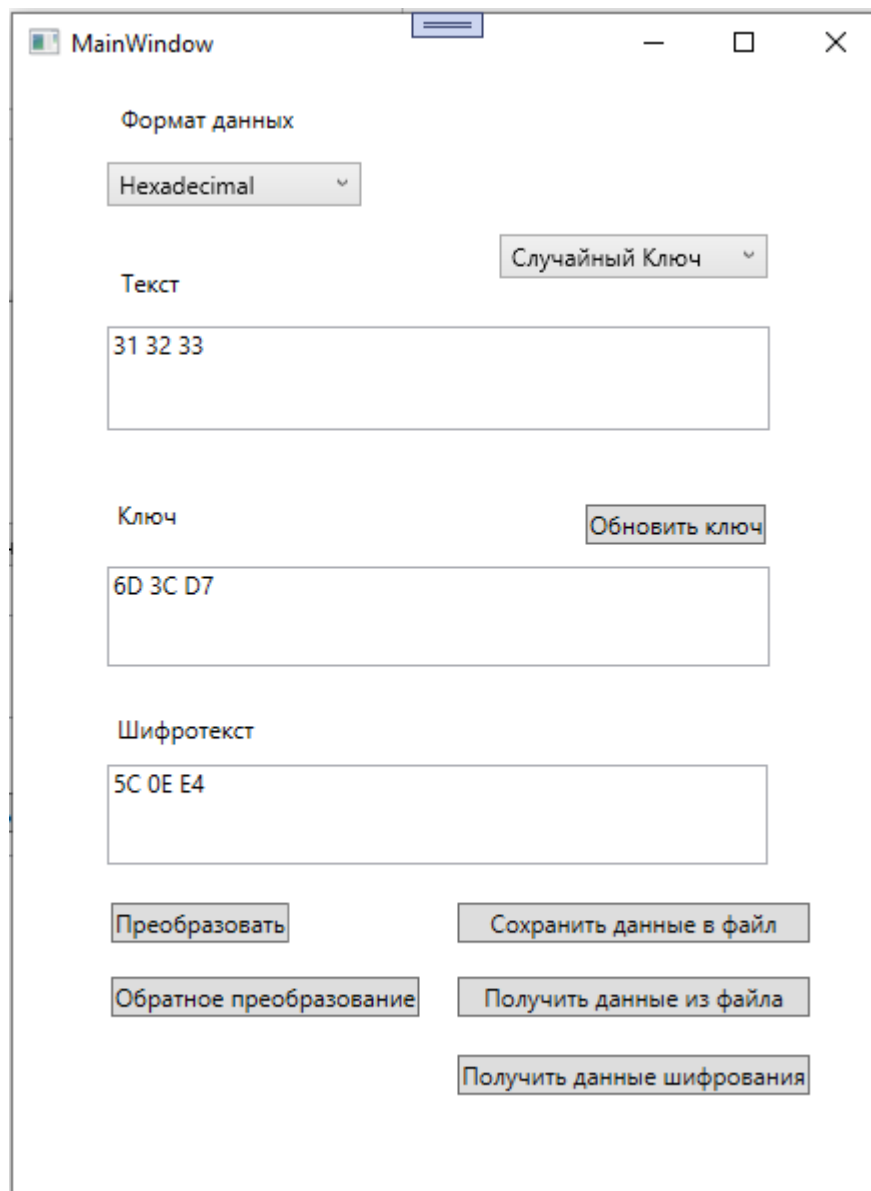


Рисунок 3. Шестнадцатеричный формат данных

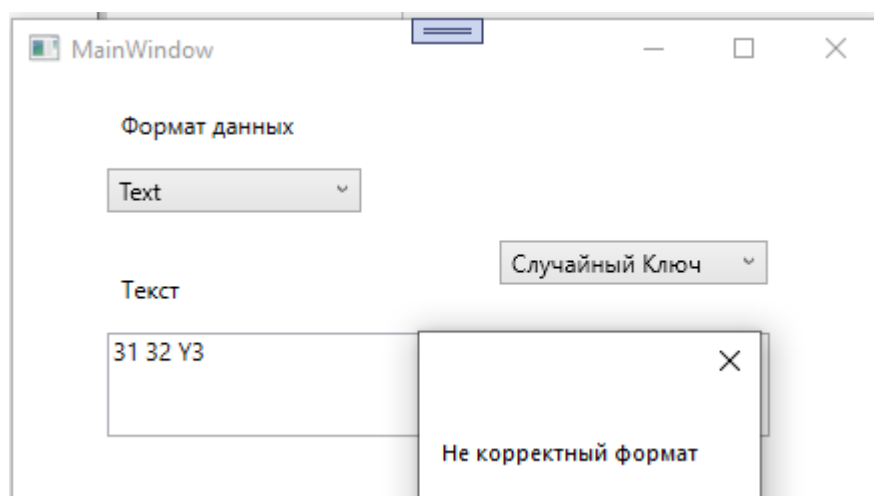


Рисунок 4. Контроль формата данных при преобразовании

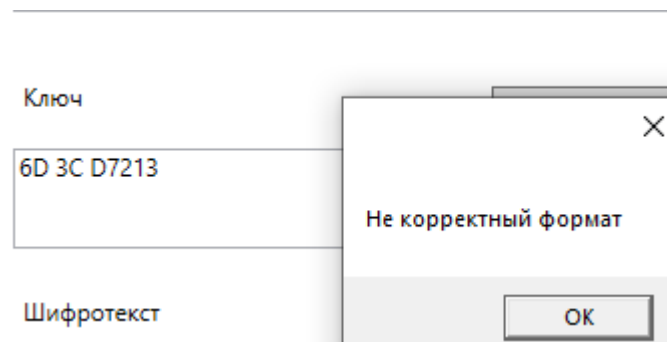


Рисунок 5. Контроль длины данных при преобразовании форматов

4.2. Гаммирование случайным ключом

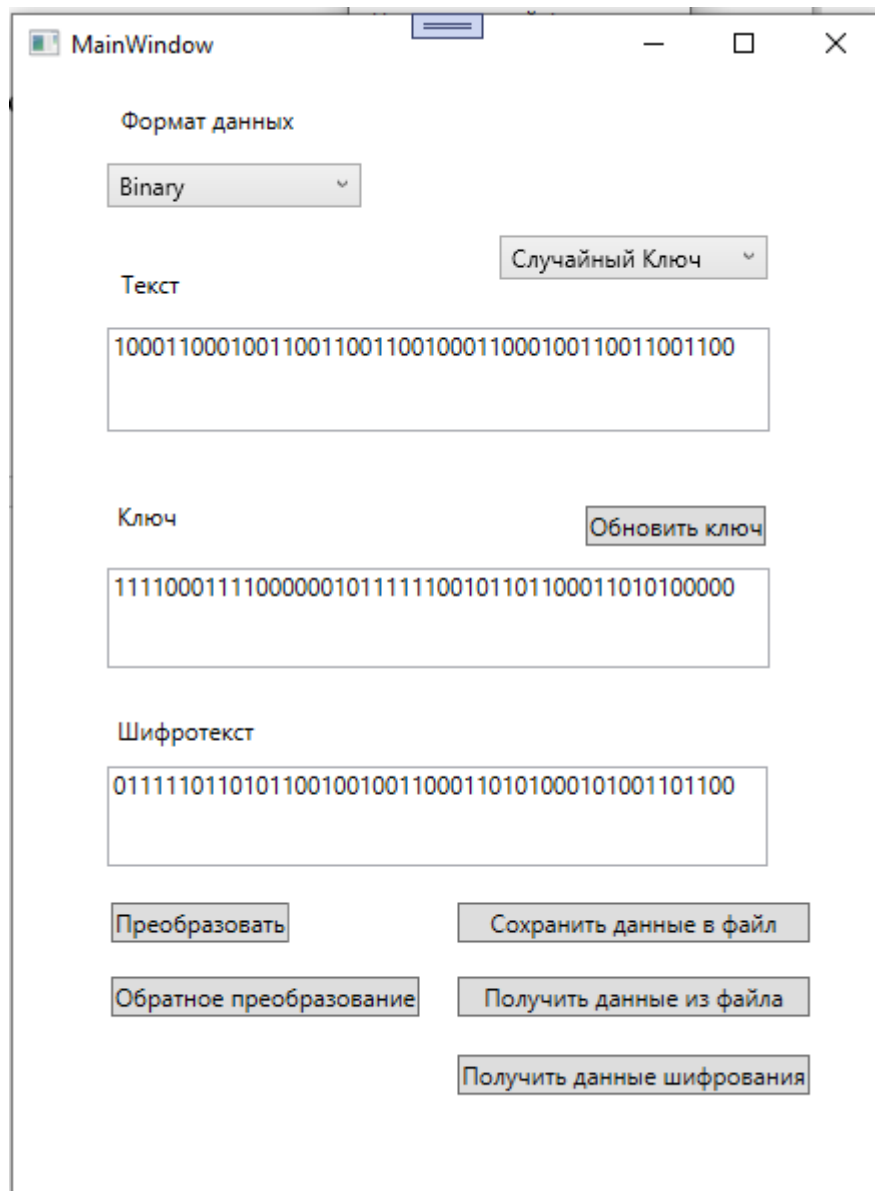


Рисунок 6. Гаммирование случайным ключом

4.3. Гаммирование скремблером

4.3.1. Скремблер $x^9 + x^3 + 1$

The screenshot shows a software window titled 'MainWindow' with a standard Windows title bar (minimize, maximize, close buttons). The interface is divided into two main sections: data input/output on the left and scrambling parameters on the right.

Left Section (Data):

- Формат данных:** A dropdown menu set to 'Hexadecimal'.
- Текст:** A text input field containing '31 32 33 34 35 36'. Above it is a 'Скремблер' dropdown menu.
- Ключ:** A text input field containing 'FF 80 3E 23 0B 67'. To its right is an 'Обновить ключ' button.
- Шифротекст:** A text input field containing 'CE B2 0D 17 3E 51'.
- Buttons:** At the bottom, there are five buttons: 'Преобразовать', 'Сохранить данные в файл', 'Обратное преобразование', 'Получить данные из файла', and 'Получить данные шифрования'.

Right Section (Parameters):

- Параметры скремблера:** A dropdown menu set to ' $x^9 + x^3 + 1$ '.
- Начальное значение скремблера:** A text input field containing '11111111'. Below it is a 'Получить значение' button.
- Критерий Нй квадрат пройден:** A text input field containing '0,08333333333333341'.
- Период последовательности:** A text input field containing '48'.
- Последовательность сбалансированная:** A text input field containing '1'.
- Корреляция отсутствует:** A label indicating the correlation status.

Рисунок 7. Гаммирование первым вариантом скремблера

4.3.2. Скремблер $x^9 + x^4 + 1$

The application window, titled 'MainWindow', contains the following elements:

- Формат данных:** A dropdown menu set to 'Hexadecimal'.
- Текст:** A text input field containing '31 32 33 34 35 36'.
- Ключ:** A text input field containing 'FF 40 31 07 4B 6A'.
- Шифротекст:** A text input field containing 'CE 72 02 33 7E 5C'.
- Buttons:** 'Преобразовать', 'Обратное преобразование', 'Сохранить данные в файл', 'Получить данные из файла', and 'Получить данные шифрования'.
- Параметры скремблера:**
 - A dropdown menu set to $x^9 + x^4 + 1$.
 - Начальное значение скремблера:** A text input field containing '111111111' and a 'Получить значение' button.
 - Критерий Нй квадрат пройден:** A text input field containing '0,08333333333333341'.
 - Период последовательности:** A text input field containing '48'.
 - Последовательность сбалансированная:** A text input field containing '1'.
 - Корреляция отсутствует** (text label).

Рисунок 8. Гаммирование вторым вариантом скремблера

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

The screenshot shows a Windows application window titled "MainWindow". The interface is organized into several sections:

- Формат данных**: A dropdown menu currently set to "Text".
- Текст**: A text input field containing the phrase "О, бедный Йорик мой". To its right is a button labeled "Случайный Ключ" with a dropdown arrow.
- Ключ**: A text input field containing "Я знал его, Гораций". To its right is a button labeled "Обновить ключ".
- Шифротекст**: A text input field containing the encrypted text "З" and "Н" on two separate lines.
- Buttons**: At the bottom, there are five buttons arranged in three rows:
 - Row 1: "Преобразовать" and "Сохранить данные в файл".
 - Row 2: "Обратное преобразование" and "Получить данные из файла".
 - Row 3: "Получить данные шифрования".

Рисунок 9. Шифруем текст осмысленной фразой

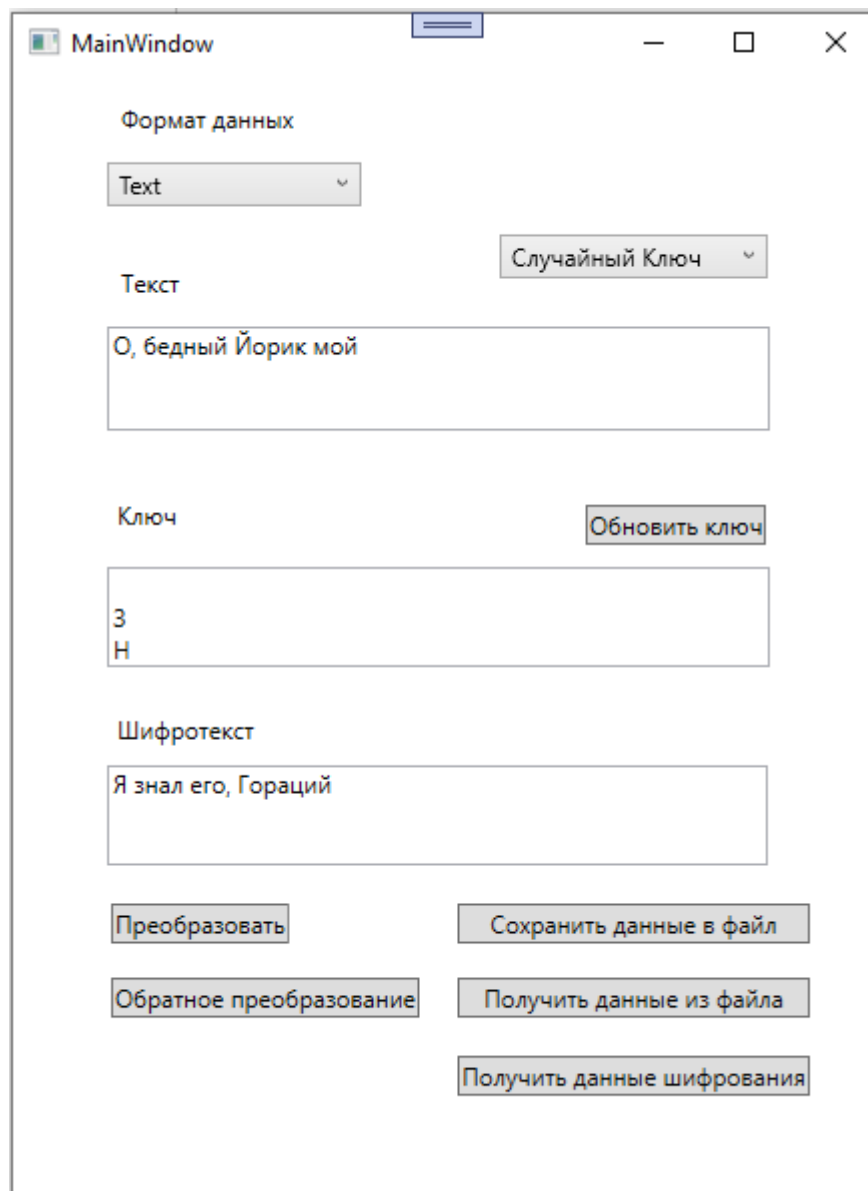


Рисунок 10. Используем полученный шифротекст как ключ и получаем из первичного текста осмысленный шифротекст

- 4.5. 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:

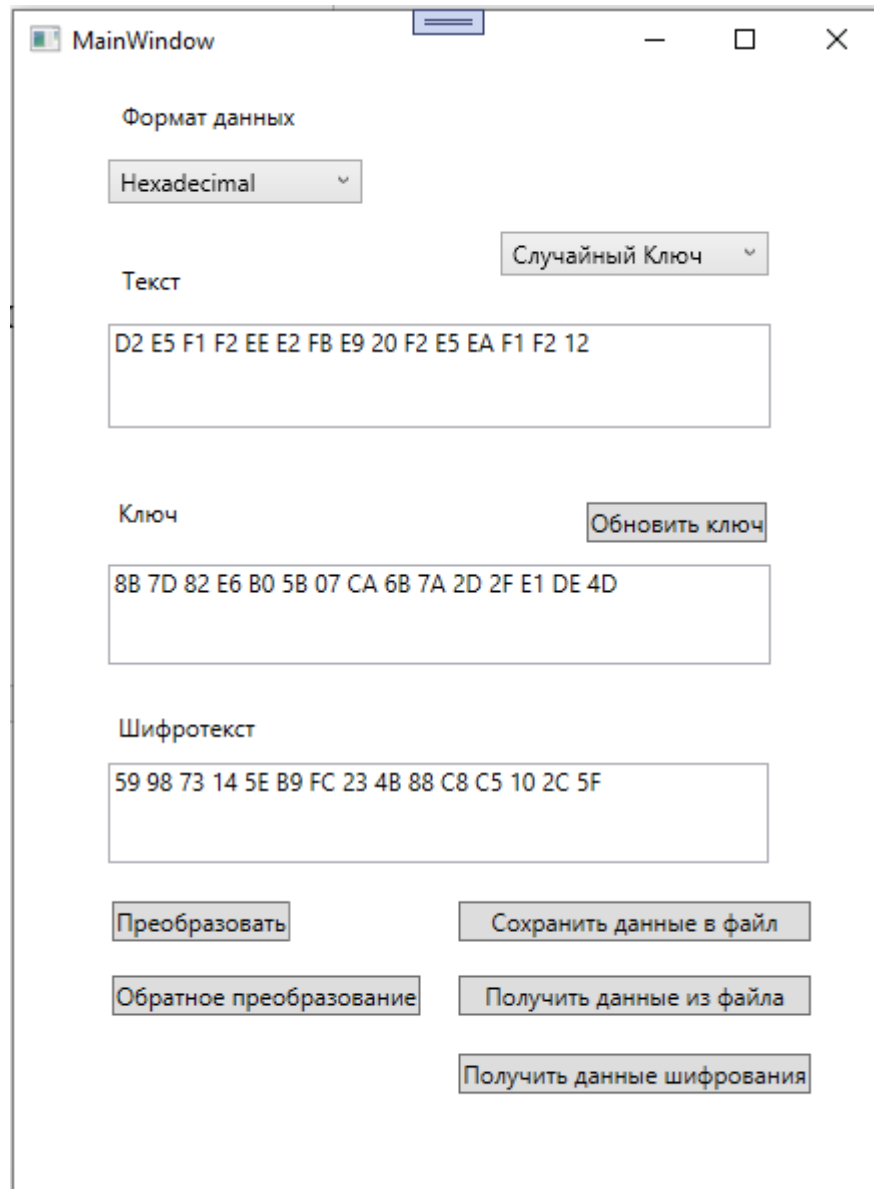
$$Y_1 = P_1 \oplus K$$

$$Y_2 = P_2 \oplus K$$

Тогда, исходя из свойств операции XOR можно получить:

$$Y_1 \oplus Y_2 = P_1 \oplus K \oplus P_2 \oplus K = P_1 \oplus P_2$$

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



MainWindow

Формат данных
Hexadecimal

Текст
Случайный Ключ

D2 E5 F1 F2 EE E2 FB E9 20 F2 E5 EA F1 F2 12

Ключ
Обновить ключ

8B 7D 82 E6 B0 5B 07 CA 6B 7A 2D 2F E1 DE 4D

Шифротекст

59 98 73 14 5E B9 FC 23 4B 88 C8 C5 10 2C 5F

Преобразовать Сохранить данные в файл

Обратное преобразование Получить данные из файла

Получить данные шифрования

Рисунок 11. Первый шифротекст

MainWindow

Формат данных

Hexadecimal

Текст

Случайный Ключ

D2 E5 EA F1 F2 EE E2 FB E9 20 F2 E5 F1 F2 12

Ключ

Обновить ключ

8B 7D 82 E6 B0 5B 07 CA 6B 7A 2D 2F E1 DE 4D

Шифротекст

59 98 68 17 42 B5 E5 31 82 5A DF CA 10 2C 5F

Преобразовать

Сохранить данные в файл

Обратное преобразование

Получить данные из файла

Получить данные шифрования

Рисунок 12. Второй шифротекст

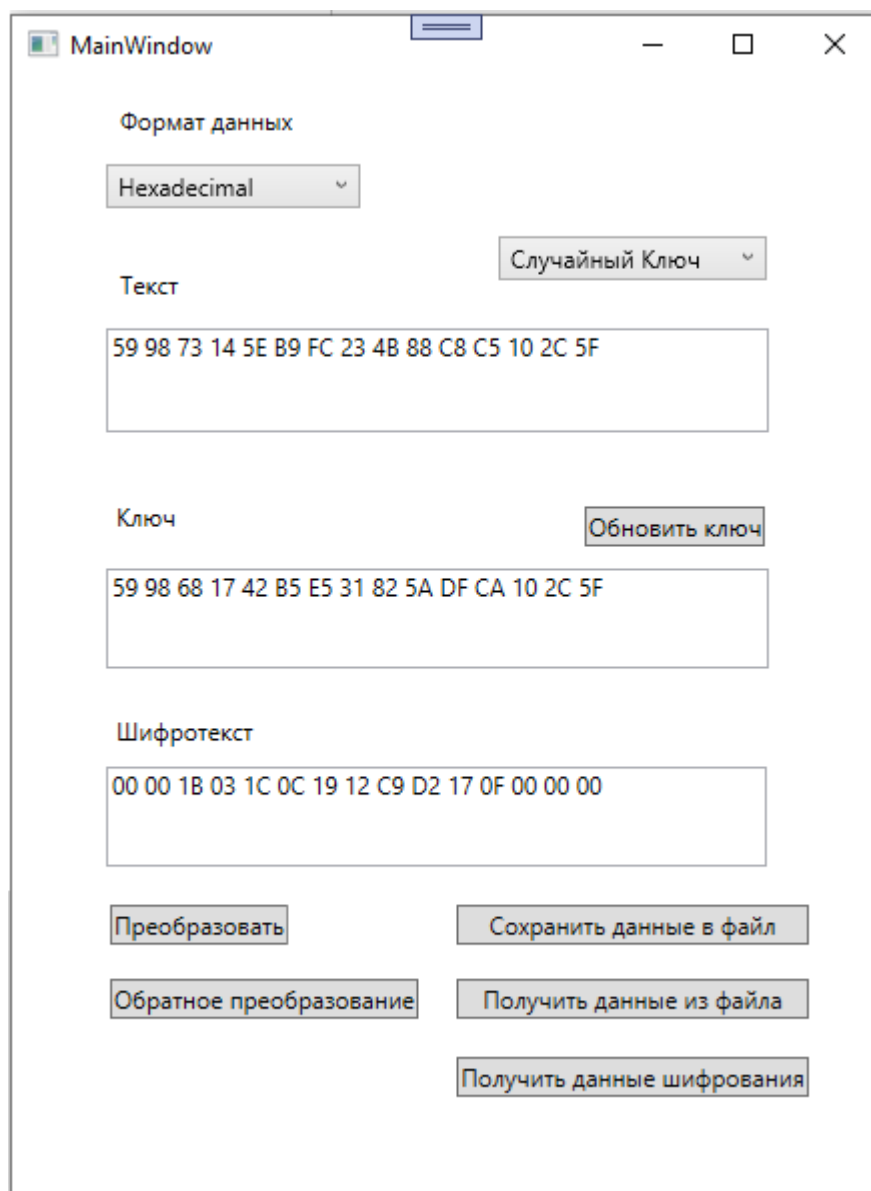


Рисунок 13. Результат операции XOR для двух шифротекстов, образованных одним ключом

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 Deer, 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.

4.6. Результаты работы скремблера при разных начальных значениях

Скремблер Начальное значение регистра	$x^9 + x^3 + 1$		$x^9 + x^4 + 1$
	Значение ключа	80 BF 60 6E 7A 4C	80 5F 67 7C DA 4A
000000001	Значение ключа	80 BF 60 6E 7A 4C	80 5F 67 7C DA 4A
	Критерий Ни квадрат пройден	0,0833333333333341	0,0833333333333341
	Период последовательности	47	47
	Последовательность сбалансированная	1	1
	Корреляция отсутствует		
000000010	Значение ключа	C0 5F 30 37 3D A6	C0 AF 33 3E 6D A5
	Критерий Ни квадрат пройден	0	0,3333333333333332
	Период последовательности	48	48
	Последовательность сбалансированная	1	1
	Корреляция отсутствует		
000001100	Значение ключа	F0 17 CC 4D 8F A9	F0 EB 8C 4F 5B E9
	Критерий Ни квадрат пройден	0,0833333333333341	1,3333333333333335
	Период последовательности	48	48
	Последовательность сбалансированная	1	1
	Корреляция отсутствует		

000101100	Значение ключа	FC 05 73 D3 63 6A	FC 3A E3 D3 56 BA
	Критерий Нй квадрат пройден	Критерий Нй квадрат пройден	
	0,333333333333332	2,08333333333333	
	Период последовательности	Период последовательности	
	47	48	
	Последовательность сбалансированная	Последовательность сбалансированная	
	1	1	
	Корреляция присутствует	Корреляция отсутствует	
111111111	Значение ключа	FF 80 3E 23 0B 67	FF 40 31 07 4B 6A
	Критерий Нй квадрат пройден	Критерий Нй квадрат пройден	
	0,0833333333333341	0,0833333333333341	
	Период последовательности	Период последовательности	
	48	48	
	Последовательность сбалансированная	Последовательность сбалансированная	
	1	1	
	Корреляция отсутствует	Корреляция отсутствует	

5. 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 длины, 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.

6. 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 = ConvertUtility.ConvertStringToByteArray(Text);
                var key = ConvertUtility.ConvertStringToByteArray(Key);
                var tt = Encryptor(text, key);
                chiphrtext = ConvertUtility.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 = 1;
            }
        }
    }
}

```

```

        code = 2;

    }

    else

        code = 1;

    }

    return (chiphrttext, 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;
}

}

```

ScamblerClass.cs

```
public static class ScamblerClass
{
    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 = 2.00 * ((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 AllowedCharHex = "0123456789ABCDEF";

    private const string AllowedCharBin = "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 = AllowedCharBin;
                break;
            }
            case "Hex":
            {
                sample = AllowedCharHex;
                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, 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 = ConvertUtility.UniConvert(Text.Text, TextFormarFlag,
                TextFormat.Text);
                Key.Text = ConvertUtility.UniConvert(Key.Text, TextFormarFlag,
                TextFormat.Text);
                Chiphrttext.Text = ConvertUtility.UniConvert(Chiphrttext.Text, TextFormarFlag,
                TextFormat.Text);
            }

            if (TextFormarFlag == "Binary")
            {
                if (ConvertUtility.CheckIncorrectFormat(Text.Text, "Bin") &&
                ConvertUtility.CheckIncorrectLength(Text.Text))
                {
                    Text.Text = ConvertUtility.UniConvert(Text.Text, TextFormarFlag,
                    TextFormat.Text);
                }
            }
        }
    }
}
```

```

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

        Chiphrttext.Text = ConvertUtility.UniConvert(Chiphrttext.Text,
TextFormarFlag, TextFormat.Text);
    }
    else
    {
        MessageBox.Show("Не корректный формат");
        TextFormat.SelectedIndex = 1;
    }

    if (TextFormarFlag == "Hexadecimal")
        if (ConvertUtility.CheckIncorrectFormat(Text.Text, "Hex") &&
ConvertUtility.CheckIncorrectLength(Text.Text))
        {
            Text.Text = ConvertUtility.UniConvert(Text.Text, TextFormarFlag,
TextFormat.Text);

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

            Chiphrttext.Text = ConvertUtility.UniConvert(Chiphrttext.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)

```



```

        ChiphrtText.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 =
                ConvertUtility.ConvertByteArrayToString(GammaCrypt.RandKey(Text.Text.Length));

            if (TextFormat.Text == "Binary")
                Key.Text =
                ConvertUtility.ConvertByteArrayToBinaryStr(GammaCrypt.RandKey(Text.Text.Length));

            if (TextFormat.Text == "Hexadecimal")
                Key.Text =
                ConvertUtility.ByteArrayToHexString(GammaCrypt.RandKey(Text.Text.Length));
        }

        if (KeyFormarFlag == "Scrambler")
        {
            if (TextFormat.Text == "Text")
            {
                var start = ConvertUtility.PadToByte(ScramStart.Text);
                var startbin = ConvertUtility.ConvertBinaryStrToByte(start);
                var bb = BitConverter.ToInt16(startbin, 0);

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

                var b = ScamblerClass.LFSR_one(rt.Length, Convert.ToInt32(bb),
                ScramFormat.SelectedIndex);

                ScramblerKey = ConvertUtility.GetScramKey(b);

                Key.Text =
                ConvertUtility.ConvertByteArrayToString(ConvertUtility.ConvertBinaryStrToByte(ScramblerKey))
                ;
            }
        }
    }

```

```

var hi = ScamblerClass.Hi2(ScramblerKey);
var baltest = ScamblerClass.Balance(ScramblerKey);
var corrttest = ScamblerClass.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 (corrttest.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.ToInt16(startbin, 0);
    var rt = new
    BitArray(ConverteUtility.ConvertStringToByteArray(Text.Text));

    var b = ScamblerClass.LFSR_one(rt.Length, Convert.ToInt32(bb),
    ScramFormat.SelectedIndex);

    ScramblerKey = ConverteUtility.GetScramKey(b);
    Key.Text = ScramblerKey;

    var hi = ScamblerClass.Hi2(ScramblerKey);

```

```

var baltest = ScamblerClass.Balance(ScramblerKey);
var corrttest = ScamblerClass.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 (corrttest.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.ToInt16(startbin, 0);
    var rt = new
BitArray(ConverteUtility.ConvertStringToByteArray(Text.Text));
    var b = ScamblerClass.LFSR_one(rt.Length, Convert.ToInt32(bb),
ScramFormat.SelectedIndex);
    ScramblerKey = ConverteUtility.GetScramKey(b);
    Key.Text =
ConverteUtility.ByteArrayToHexString(ConverteUtility.ConvertBinaryStrToByte(ScramblerKey));

    var hi = ScamblerClass.Hi2(ScramblerKey);
    var baltest = ScamblerClass.Balance(ScramblerKey);

```

```

var corrtest = ScamblerClass.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);  
}
```

```
}
```