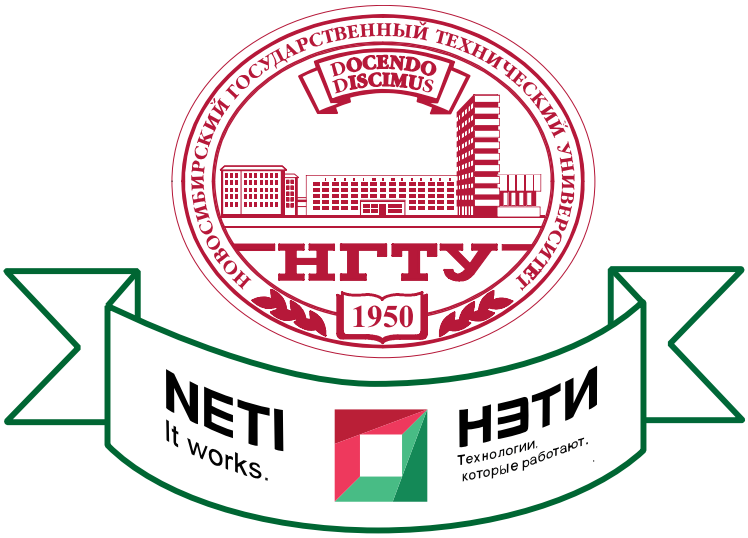
Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

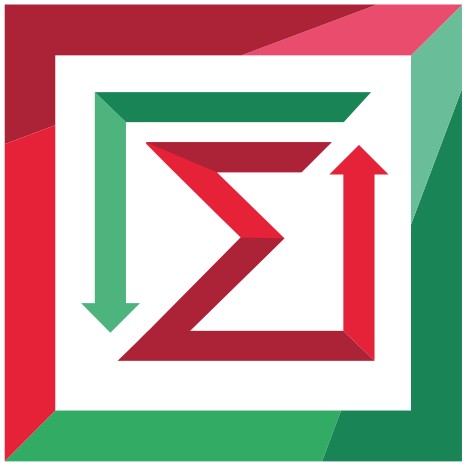


Кафедра теоретической и прикладной информатики

Лабораторная работа № 1

по дисциплине «ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ»

ГАММИРОВАНИЕ.
МОДЕЛИРОВАНИЕ РАБОТЫ СКРЕМБЛЕРА



Факультет:	ПМИ
Группа:	ПМИ-02
Вариант:	6
Студенты:	Сидоров Даниил,
	Дюков Богдан
Преподаватели:	Авдеенко Татьяна Владимировна,
	Кутузова Ирина Александровна.

Новосибирск

2026

1. Цель работы

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

2. Задача

I. Реализовать приложение для шифрования, позволяющее выполнять перечисленные действия.

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.
2. Шифровать данные при помощи каждого заданного в варианте скремблера:
   1) шифруемый текст должен храниться в одном файле, начальное значение скремблера – в другом;
   2) зашифрованный текст должен сохраняться в файл;
   3) в процессе шифрования предусмотреть возможность просмотра и изменения начального значения скремблера, шифруемого и зашифрованного текстов в двоичном, шестнадцатеричном и символьном виде.
3. Проводить исследование генерируемой каждым скремблером последовательности псевдослучайных чисел при заданном начальном ключе:
   1) получать период скремблера;
   2) проверять равномерность последовательности по критерию $\chi 2$;
   3) исследовать последовательность на свойства сбалансированности, цикличности, корреляции.

II. Реализовать приложение для дешифрования, позволяющее выполнять перечисленные действия.

1. Дешифровать данные в режиме однократного гаммирования:
   1) зашифрованный текст должен храниться в одном файле, ключ – в другом;
   2) расшифрованный текст должен сохраняться в файл;
   3) в процессе дешифрования предусмотреть возможность просмотра и изменения ключа, зашифрованного и расшифрованного текстов в двоичном, шестнадцатеричном и символьном виде.
2. Расшифровать данные при помощи каждого заданного в варианте скремблера:
   1) зашифрованный текст должен храниться в одном файле, начальное значение скремблера – в другом;
   2) зашифрованный текст должен сохраняться в файл;

3) в процессе дешифрования предусмотреть возможность просмотра и изменения начального значения скремблера, зашифрованного и расшифрованного текстов в двоичном, шестнадцатеричном и символьном виде.

III. 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.

Вариант	Скремблеры
6	$f(x) = x^7 + x^5 + x^2 + 1,$ $f(x) = x^7 + x + 1$

3. Метод решения задачи

1) Однократное гаммирование

Гаммирование – это наложение (снятие) на открытые (зашифрованные) данные криптографической гаммы, т. е. последовательности элементов данных, вырабатываемых с помощью некоторого криптографического алгоритма, для получения зашифрованных (открытых) данных.

Алгоритм шифрования в гаммировании включает следующие шаги:

1) Подготовка открытого текста и ключа. Чтобы обеспечить абсолютную стойкость шифра, необходимо выполнить следующие условия: случайность и однократное использование ключа, а также его равенство длине открытого текста.
2) Шифрование. Для этого применяется операция XOR к каждому символу открытого текста и соответствующему символу ключа. Это дает нам зашифрованный текст.

Гаммирование является симметричным алгоритмом. Поскольку двойное прибавление одной и той же величины по модулю 2 восстанавливает исходное значение, шифрование и дешифрование выполняется одной и той же программой.

2) Скремблер

Основой данного алгоритма является регистр сдвига с линейной обратной связью (LFSR или РСЛОС). Благодаря нему генерируется псевдослучайную равномерно распределенную последовательность. LFSR задаётся характеристическим многочленом степени **L**.

В регистре сдвига с линейной обратной связью выделяют две части:

- собственно регистр сдвига;
- схему обратной связи, вычисляющую значение вдвигаемого бита.

Регистр состоит из функциональных ячеек памяти, в каждой из которых хранится текущее состояние (значение) одного бита. Количество **L** называют длиной регистра. Биты (ячейки) обычно нумеруются числами i = 0, 1, … , **L**-1.

Функцией обратной связи для РСЛОС является линейная булева функция от значений всех или некоторых битов регистра. Сложение по модулю 2 является линейной булевой функцией и применяется в регистрах. При этом биты, являющиеся переменными функции обратной связи, называются **отводами**, а сам регистр называется **конфигурацией Фибоначчи**.

Формирование последовательности выглядит следующим образом:

1) Задается начальное состояние регистра и из многочлена находятся номера битов отвода;
2) Читается бит, расположенный в ячейке **L**-1; этот бит является очередным битом выходной последовательности;
3) С помощью функции обратной связи (операция XOR для битов отвода) вычисляется новое значение для ячейки 0, используя текущие значения ячеек;
4) Содержимое каждой ячейки регистра перемещается в следующую ячейку;
5) В ячейку 0 записывается бит, вычисленный в 3 пункте функцией обратной связи.

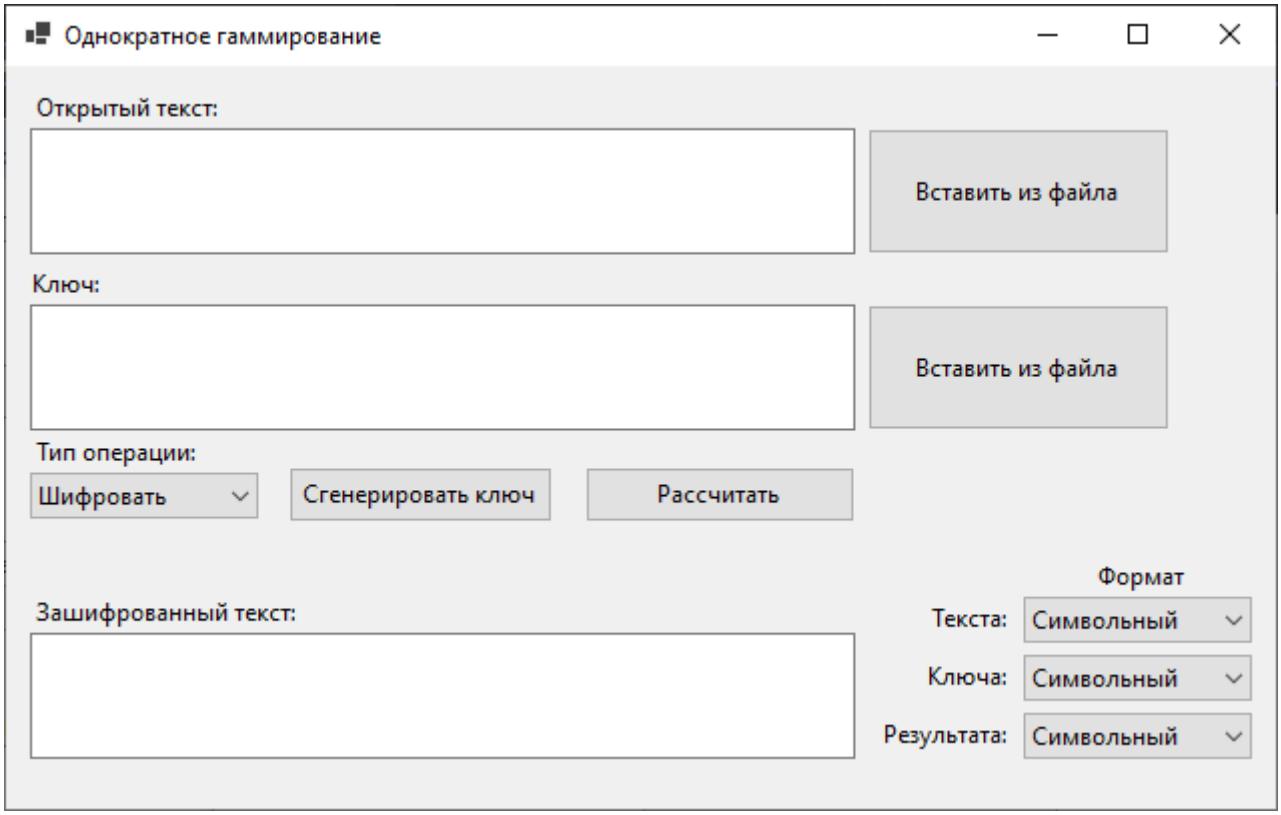
Выходная последовательность формируется в порядке генерации в РСЛОС.

Повторяя пункты 2-5, генерируется такая последовательность, которая используется в качестве гаммы при шифровании/дешифровании текста.

4. Разработанное программное средство

Разработанное программное средство представляет собой приложение Windows Forms.

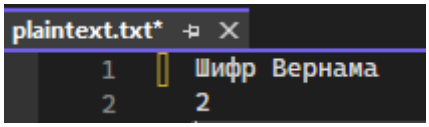
Интерфейс окна с однократным гаммированием:



Он содержит текстовые поля для открытого текста, ключа или шифротекста. Переключатель типа операции позволяет выполнить шифрование и дешифрование в одном окне. Предусмотрена возможность вставки из файлов открытого текста (или шифротекста) и ключа.

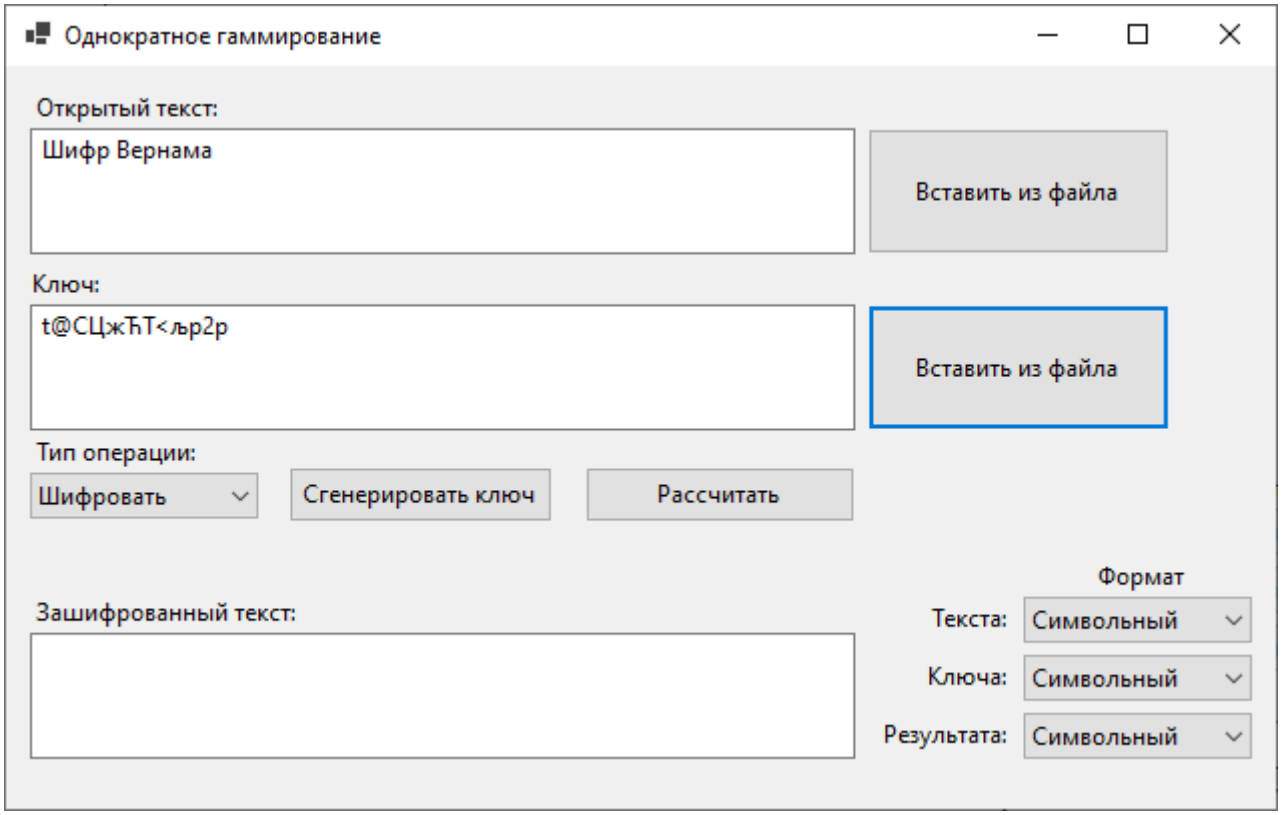
Также имеется возможность сгенерировать ключ той же длины, что и открытый текст (шифротекст). Переключатели под заголовком формата предоставляют просмотра и изменения ключа, открытого текста, зашифрованного и расшифрованного текстов в двоичном, шестнадцатеричном и символьном виде.

Рассмотрим пример шифрования. Пусть содержимое файла “plaintext.txt” следующее:



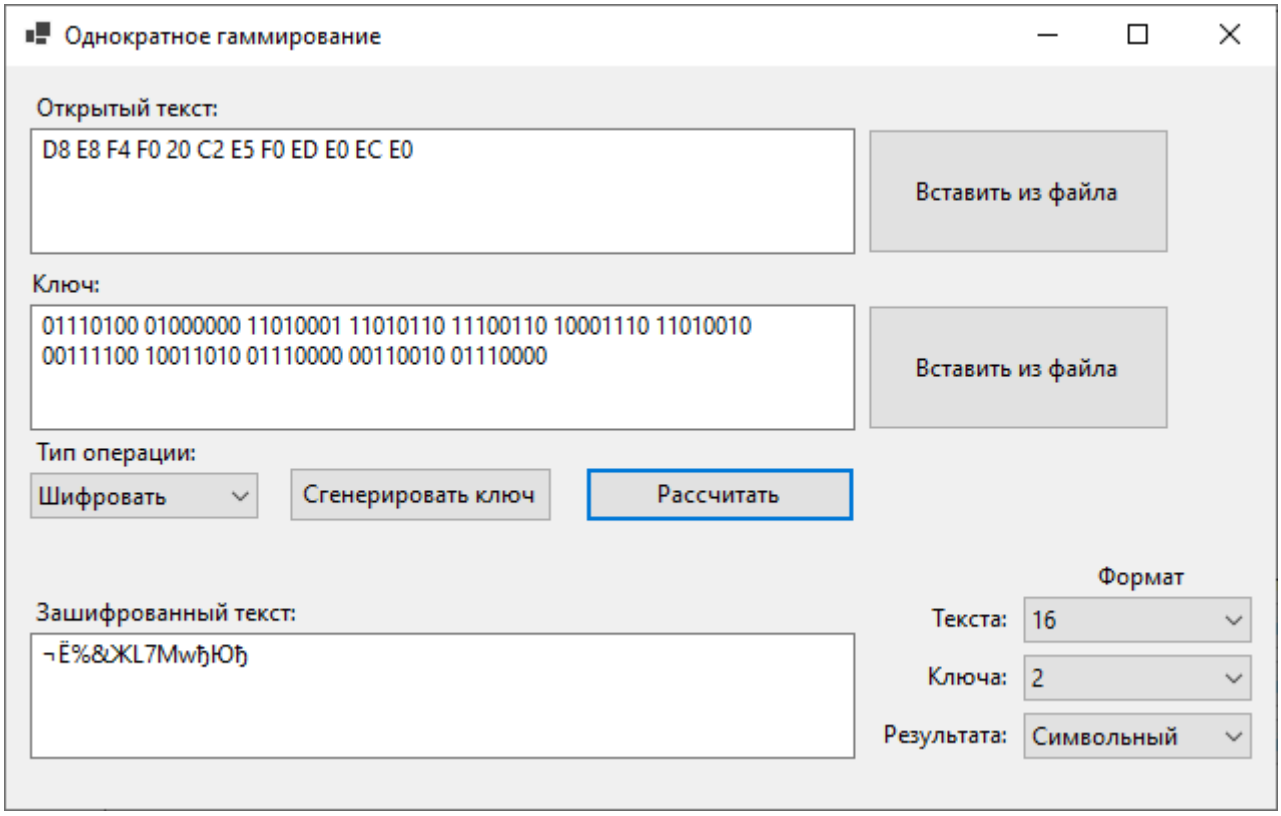
Цифра 2 под открытым текстом позволяет определить, что формат открытого текста – символьный (цифра 0 – двоичный формат, цифра 1 – шестнадцатеричный).

Запустим приложение, вставим в текстовое поле данные из файла (используя кнопку “Вставить из файла”) и нажмем кнопку “Сгенерировать ключ”:



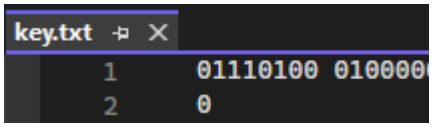
При генерации ключа учитывается длина открытого текста, а сами символы ключа выбираются случайно по таблице ASCII Windows-1251 (в десятичном формате от 32 до 255 включительно, не включая специальные символы).

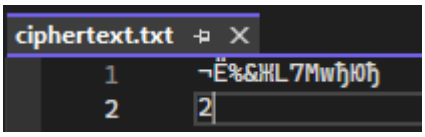
Таблица ASCII Windows-1251 также обеспечивает возможность работы с данными в различных форматах. Например, переведем открытый текст в шестнадцатеричный формат, а ключ – в двоичный формат. Сразу нажмем кнопку “Рассчитать”, чтобы получить шифротекст:



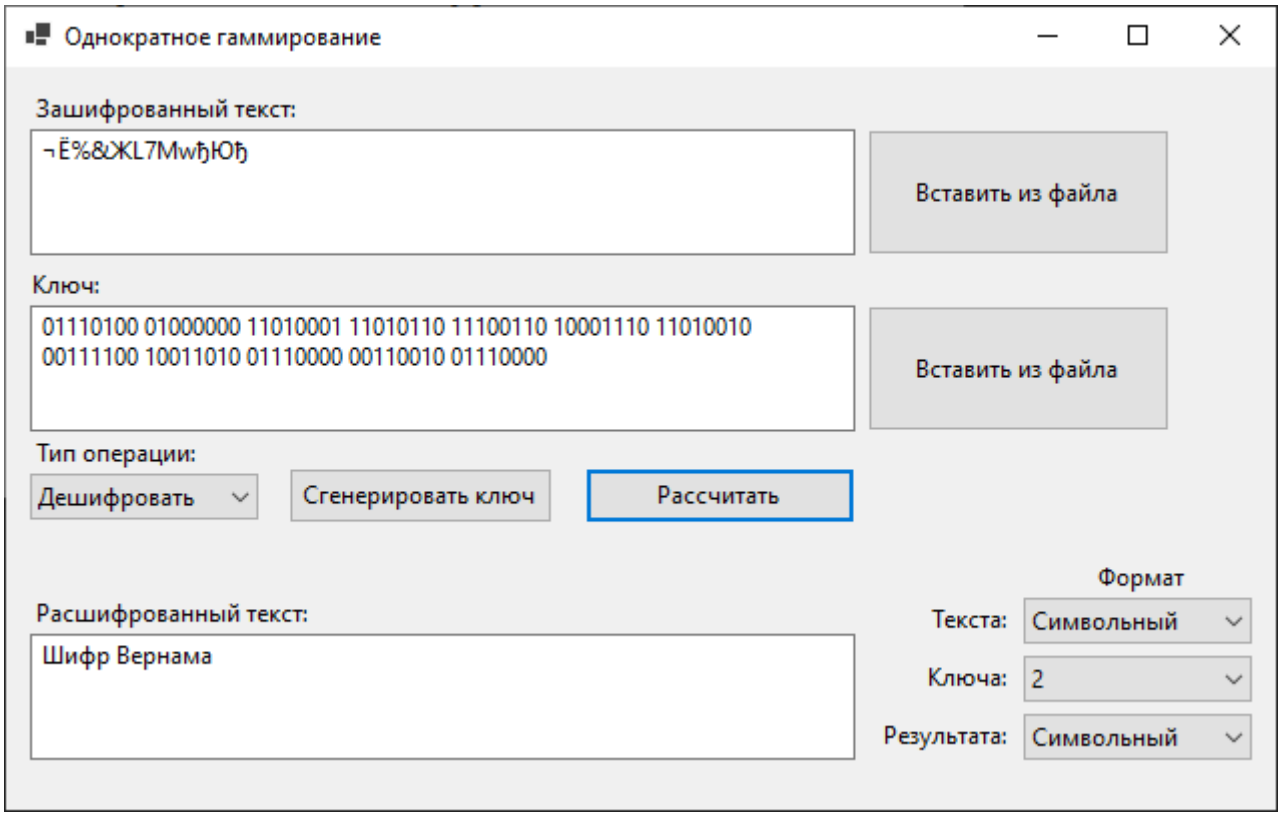
Нам неважно, в каком формате пользователь работает с данными, ведь при шифровании/дешифровании открытый текст/шифротекст и ключ переводятся в двоичный формат, после чего по результатам XOR мы получаем шифротекст/расшифрованный текст также в двоичном формате, который можно перевести в любой другой формат, определяемый пользователем.

Сразу после шифрования все данные (ключ, шифротекст) сохраняются в соответствующие файлы:

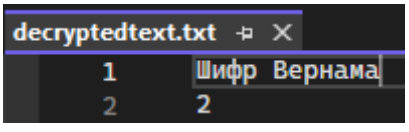




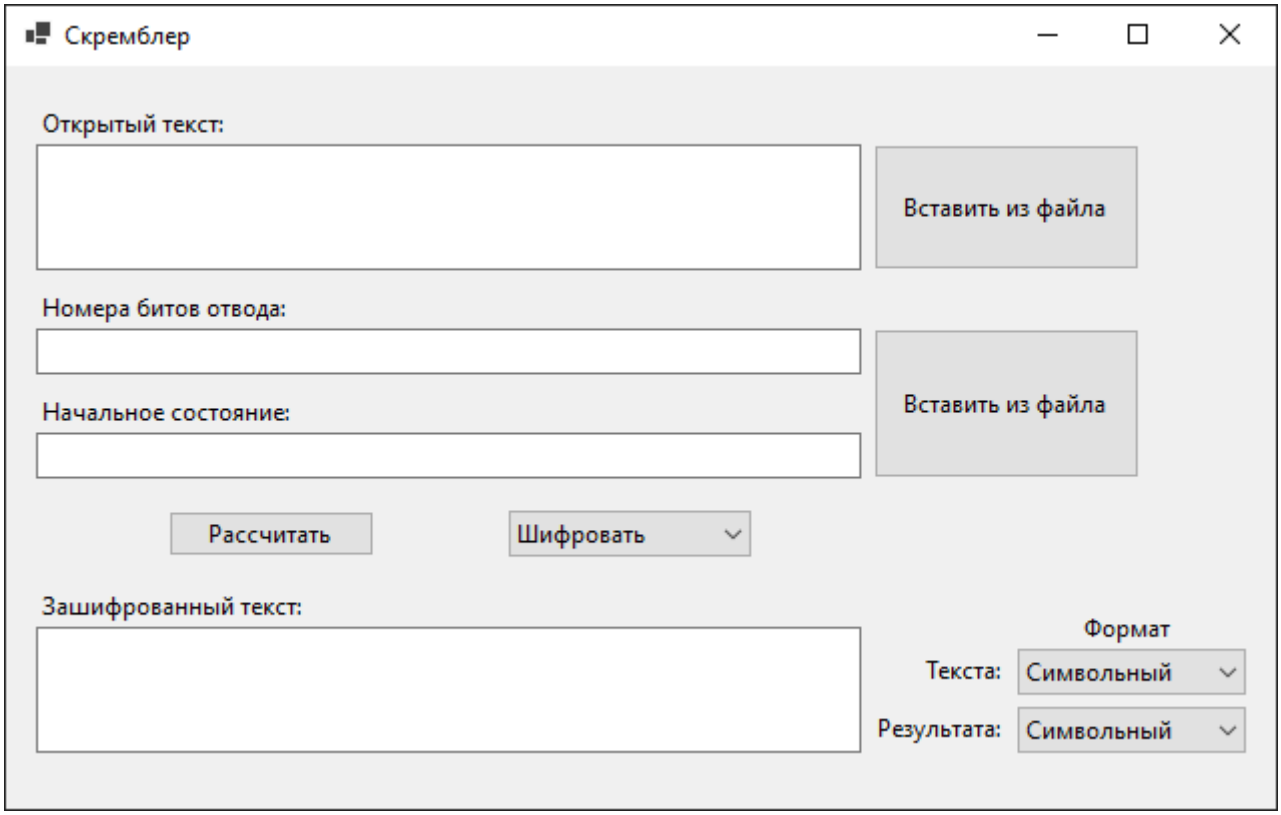
Теперь расшифруем наш шифротекст. Выбираем тип операции – дешифровать. Вставляем из файлов шифротекст и ключ, нажимаем кнопку “Рассчитать” и убеждаемся в правильности работы алгоритма:



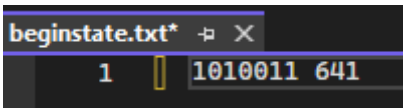
Расшифрованный текст сохраняется в соответствующий файл:



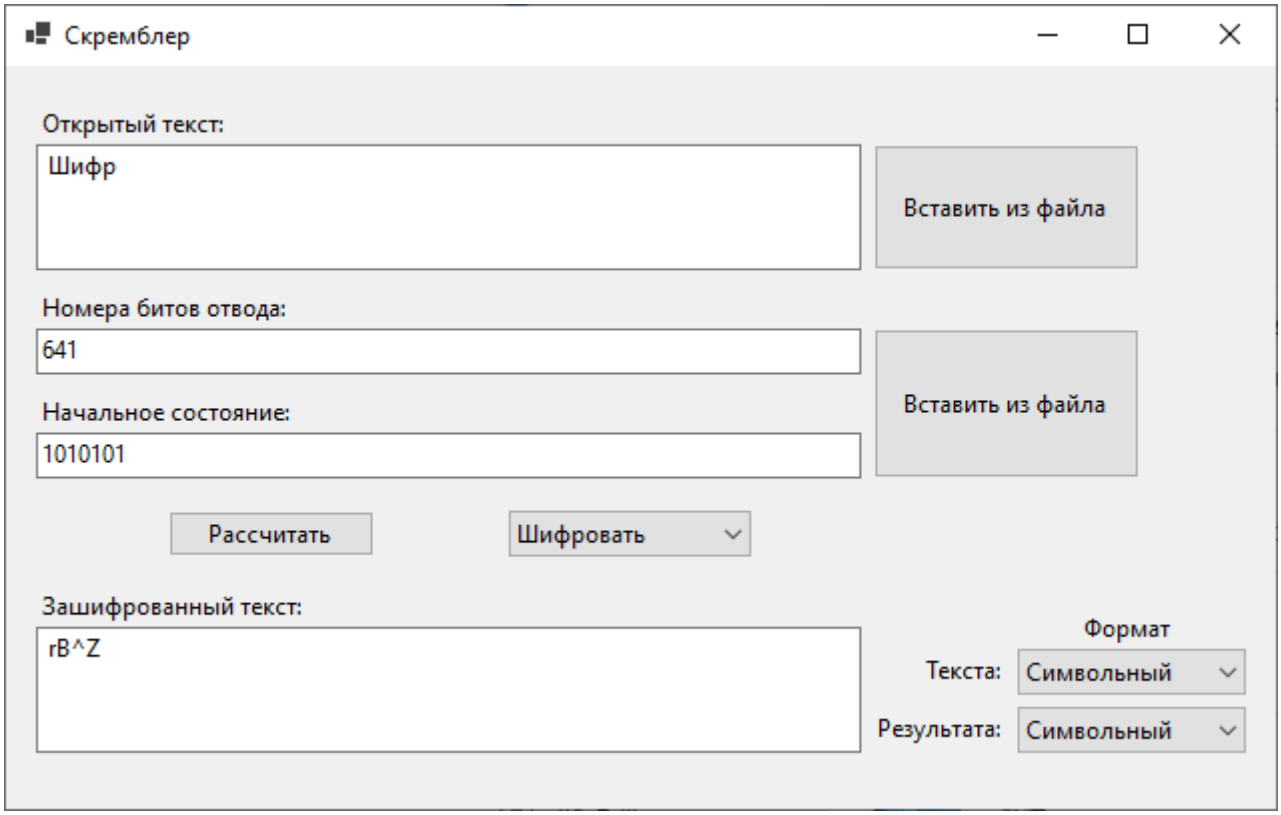
Перейдем к шифрованию при помощи скремблера. Заместо ключа в данном алгоритме задаются два параметра: номера битов отвода, которые определяются из соответствующего полинома, а также начальное состояние скремблера. Интерфейс соответствующего окна:

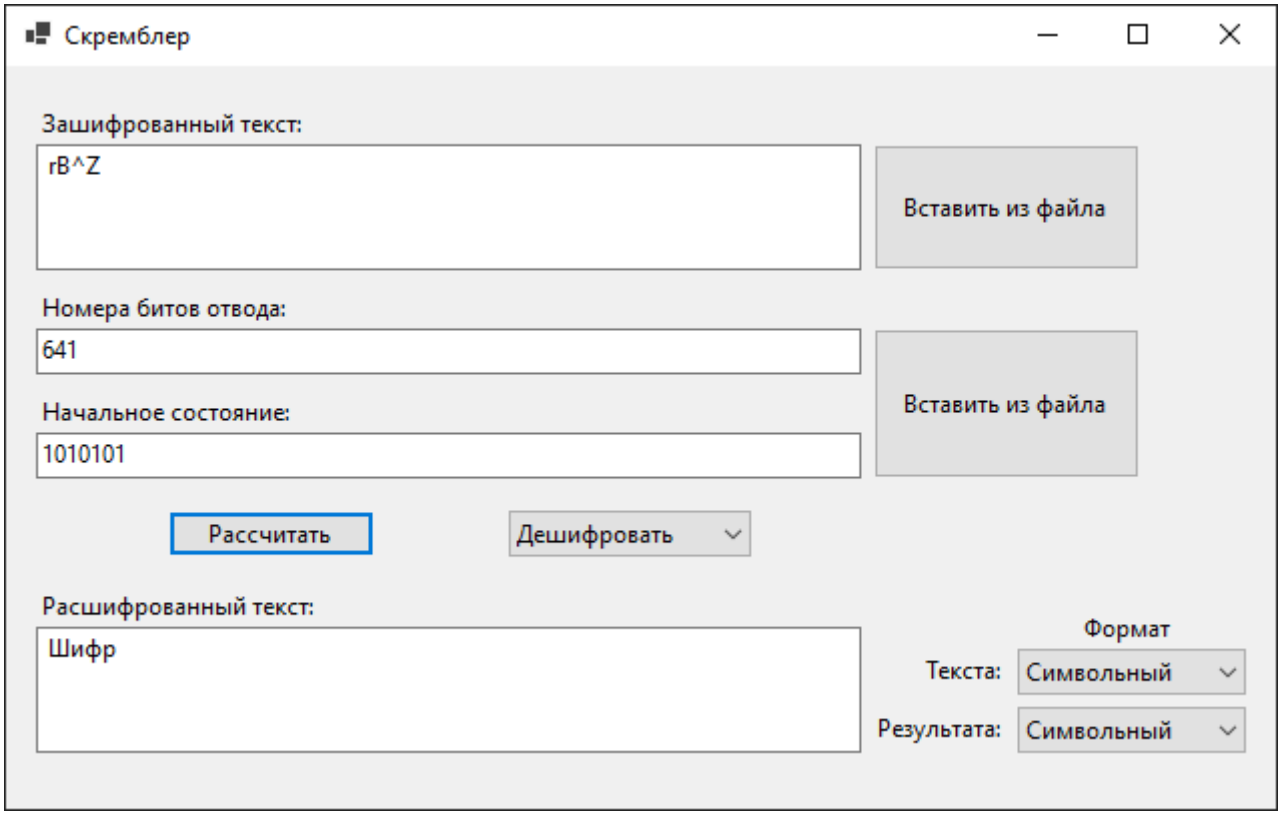


Рассмотрим файл с параметрами скремблера:



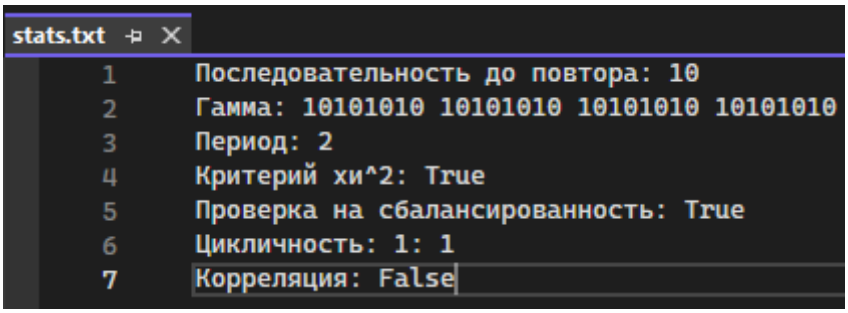
Сначала идет начальное состояние скремблера, после чего идут номера битов отвода (соответствуют полиному $f(x) = x^7 + x^5 + x^2 +$), операция XOR с которыми даст нам новый крайний левый бит при очередном сдвиге регистра.





5. Исследование скремблерной последовательности

В приложении скремлера, когда пользователь шифрует открытый текст, генерируемая последовательность анализируется, а результаты записываются в файл stats.txt. Содержимое этого файла для примера выше:



1) **Последовательность до повтора** определяет выходную последовательность, которая сгенерирована до первого повтора состояния скрамблера (она определяет период и не обязательно начинается с первого выданного скремблером бита).
2) **Гамма** есть не что иное, как псевдослучайная последовательность (генерируемая тем же самым скремблером) такой длины, которой будет достаточно для полного покрытия открытого текста.
3) **Период** – минимальная длина получаемой последовательности до начала её повторения. РСЛОС длины L имеет 2^L начальных состояний, задающих значения бит в ячейках. Из них $2^L - 1$ состояний - ненулевые, так что генерируемая последовательность имеет период $T \leq 2^L - 1$.

4) **Критерий согласия χ2-Пирсона** - используется для проверки гипотезы о том, что сгенерированная последовательность подчиняется равномерному распределению. Для этого вычисляют статистику критерия по формуле:

$$\chi^2 = \sum_{i=1}^{k} \frac{(n_i - p_i N)^2}{p_i N},$$

где k – число интервалов (у нас 2 интервала: один для двоичных нулей, другой для двоичных единиц);

n_i – количество вхождений нулей/единиц в зависимости от индекса;

$p_i = 1/k$ – вероятность попадание чисел в i-й интервал (эталонный случай);

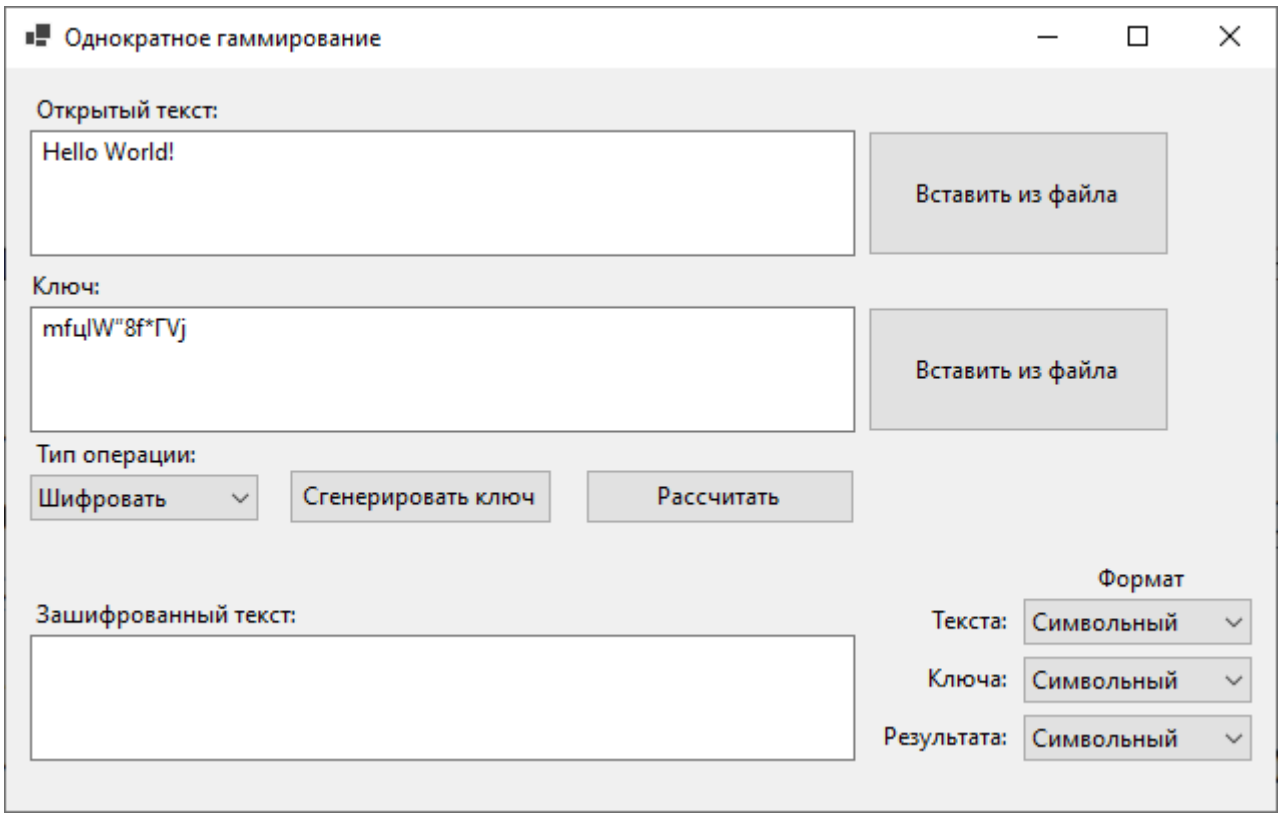
N – общее количество сгенерированных чисел.

Данная статистика сравнивается с табличным значением при уровне значимости = 0.05 и числом степеней свободы, определяемым длиной последовательности. Если статистика меньше табличного значения, то генератор удовлетворяет требованию равномерного распределения.

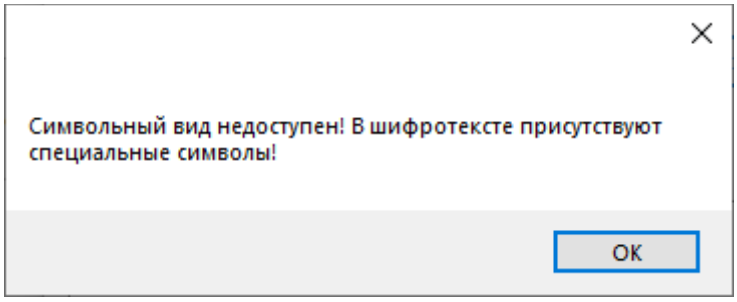
5) **Сбалансированность** - для каждого интервала последовательности количество двоичных единиц должно отличаться от числа двоичных нулей не больше чем на несколько процентов от их общего количества на интервале. В нашем случае разница между количеством нулей и единиц не должна превышать 2% от общего количества символов.
6) **Цикличность** - непрерывную последовательность одинаковых двоичных чисел называют **циклом**. Появление иной двоичной цифры автоматически начинает новый цикл. Длина цикла равна количеству одинаковых цифр в нем. Необходимо, чтобы половина всех «полосок» имела длину 1, одна четвертая – длину 2, одна восьмая – длину 3 и т. д. В нашем случае мы выводим статистику в формате <длина полоски = 1>: <доля от общего числа полосок> | <длина полоски = 2>: <доля от общего числа полосок>…
7) **Корреляция** - создаем циклически сдвинутую копию последовательности и побитно сравниваем исходную последовательность и сдвинутую копию. Для удовлетворения условия корреляции число совпадений должно отличаться от числа несовпадений не более чем на 2% от общей длины последовательности.

6. Тестирование

1) Рассмотрим приложение с однократным гаммированием. Простейший тест уже был выполнен при рассмотрении программного средства. Теперь рассмотрим сложный случай:

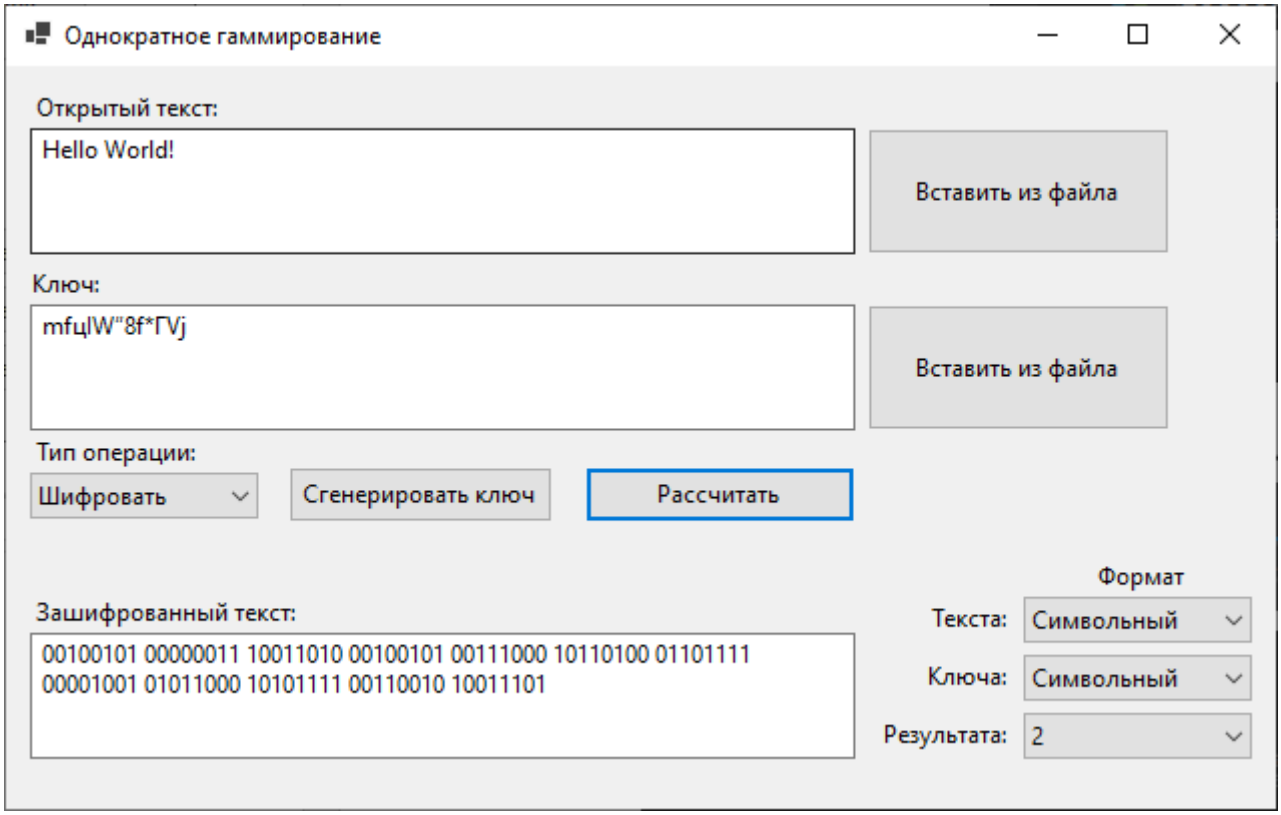


Шифрование пройдет успешно, однако в процессе преобразования результата в символьный вид пользователю будет отправлено следующее сообщение:

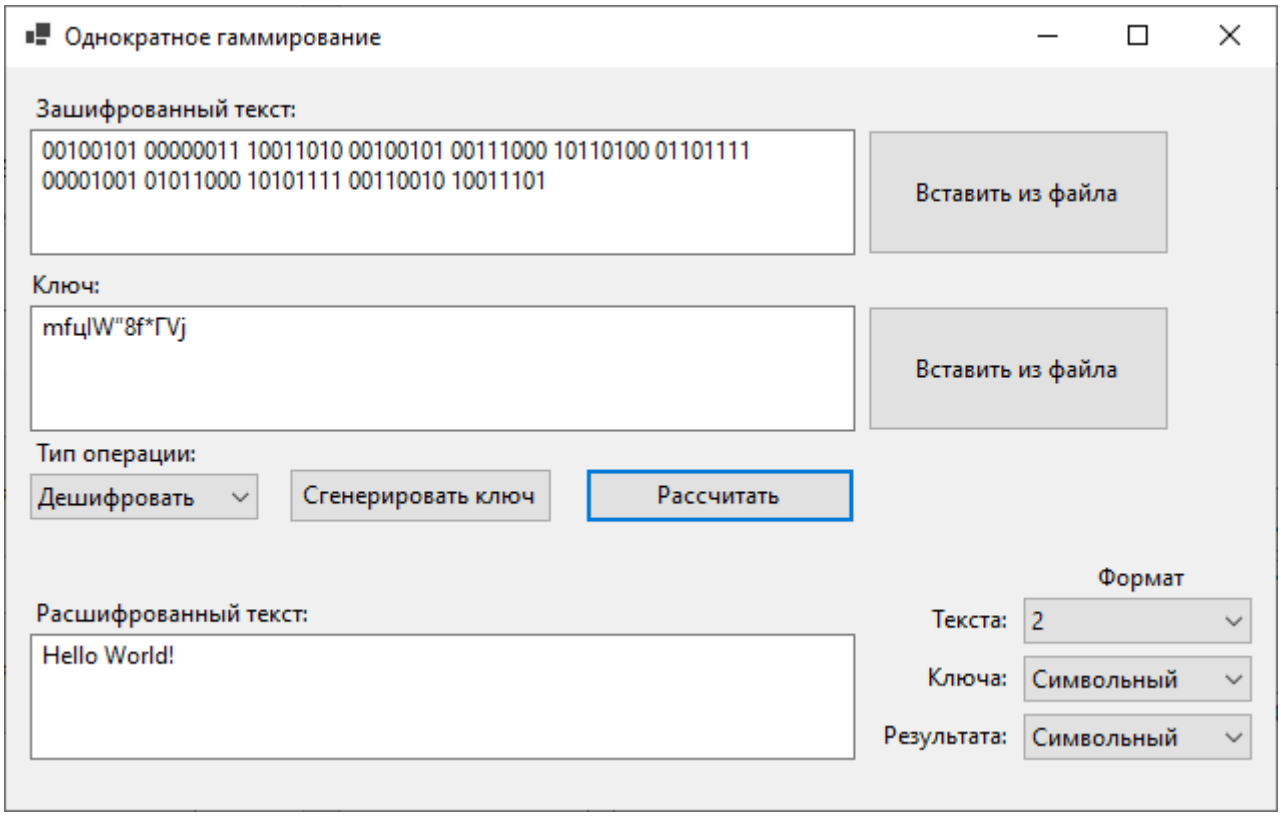


Это означает, что в шифротексте появился как минимум один так называемый специальный символ (в десятичном виде – от 0 до 31), что обязательно негативно скажется при дешифровании. Именно поэтому наше приложение запрещает показ и редактирование такого шифротекста в символьном виде, однако дает возможность это делать в других форматах.

После показа предупреждающего сообщения, шифротекст вставляется в соответствующее текстовое поле в двоичном формате:

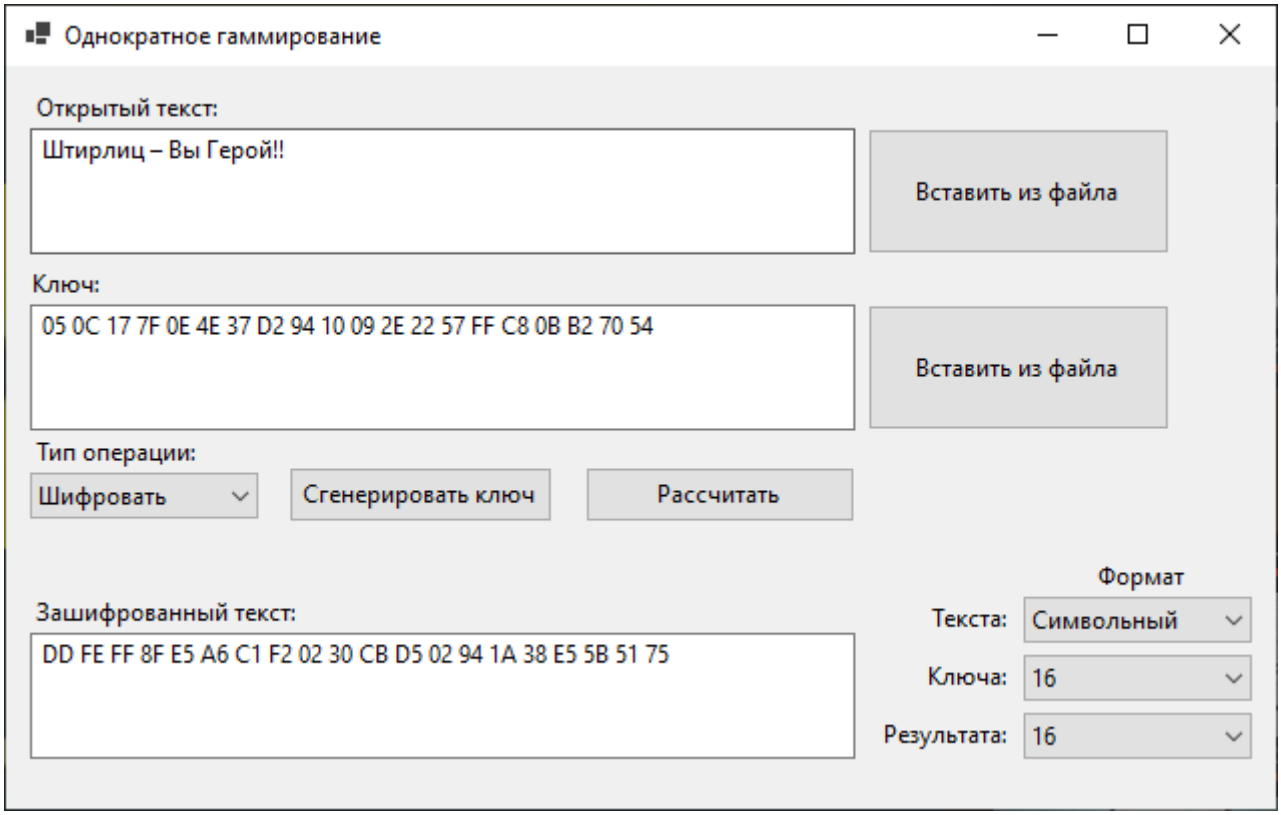


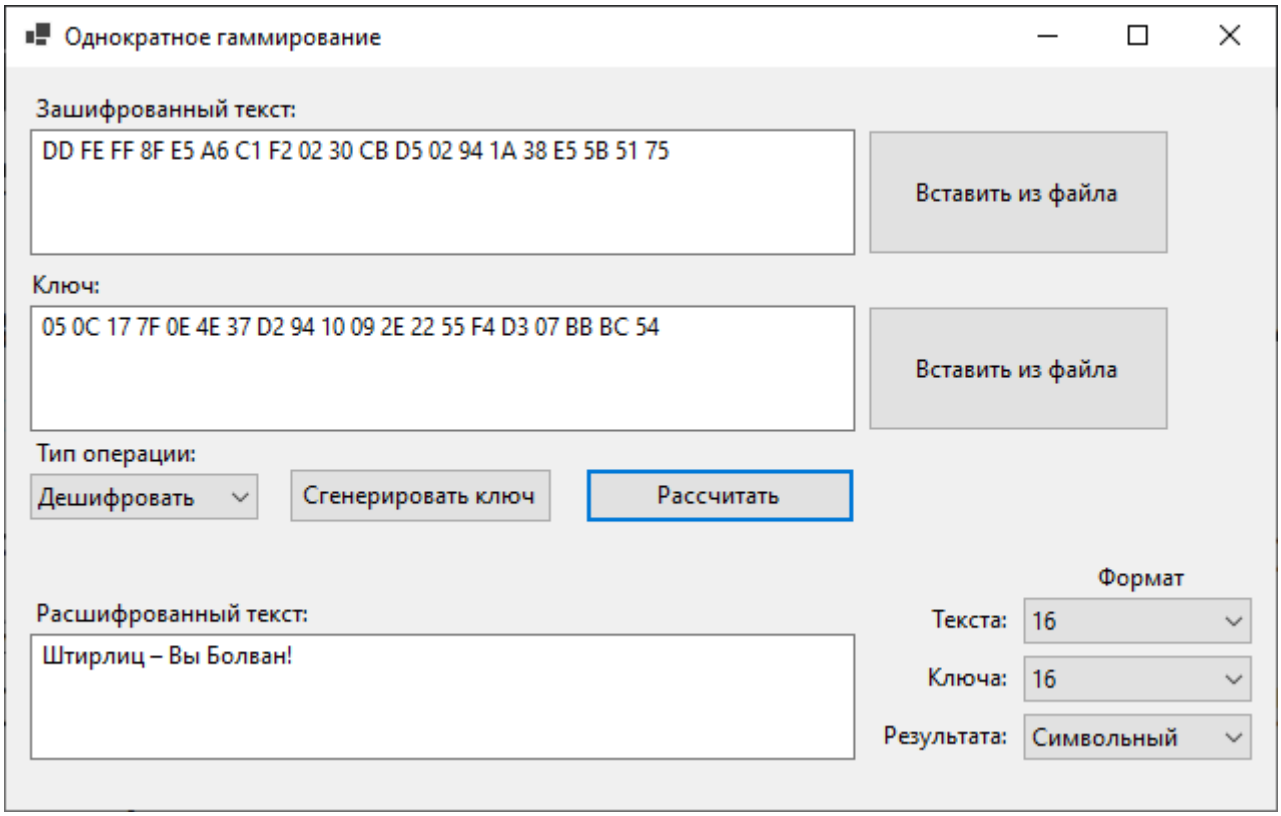
Ну и процесс дешифрования:



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

   Для этого рассмотрим пример:





В этом примере мы определили ключ (при дешифровании), который выдает осмысленный (но неверный) вариант прочтения открытого текста. Это подчеркивает абсолютную надежность шифра. Даже если криптоаналитик имеет неограниченные временные и вычислительные ресурсы, он не сможет получить информацию об открытом тексте, если ему известно только зашифрованное сообщение. Все возможные ключевые последовательности

равновероятны, и, следовательно, любые сообщения также возможны. Криптоалгоритм не раскрывает никакой информации об открытом тексте.

3) Определили и выразили аналитически, каким образом, имея зашифрованные тексты двух телеграмм, злоумышленник может получить обе телеграммы, не зная ключа и не стремясь его определить. Привели пример.

Эта концепция известна как атака двух шифротекстов. Это метод криптоанализа, который используется, когда злоумышленник имеет доступ к двум шифротекстам, которые были зашифрованы с использованием одного и того же ключа. В контексте гаммирования, это может быть особенно полезно, если ключ используется более одного раза, что является нарушением основного правила безопасного использования гаммирования.

Аналитически это можно выразить следующим образом. Пусть у нас есть два шифротекста: C_1и C_2, которые были получены путем гаммирования двух открытых текстов: P_1и P_2, с использованием одного и того же ключа K. Тогда, если мы применим операцию XOR к C_1и C_2, мы получим:

$$C_1 \oplus C_2 = (P_1 \oplus K) \oplus (P_2 \oplus K) = P_1 \oplus P_2.$$

Это происходит потому что $K \oplus K$ = 0, и любое число при XOR с 0 равно самому числу. Таким образом, злоумышленник, имея доступ к C_1и C_2, может вычислить $P_1 \oplus P_2$ без знания K.

В качестве примера, предположим, что у злоумышленника следующие два шифротекста, зашифрованные с использованием одного и того же ключа:

C_1 = 0110010
C_2 = 1100101

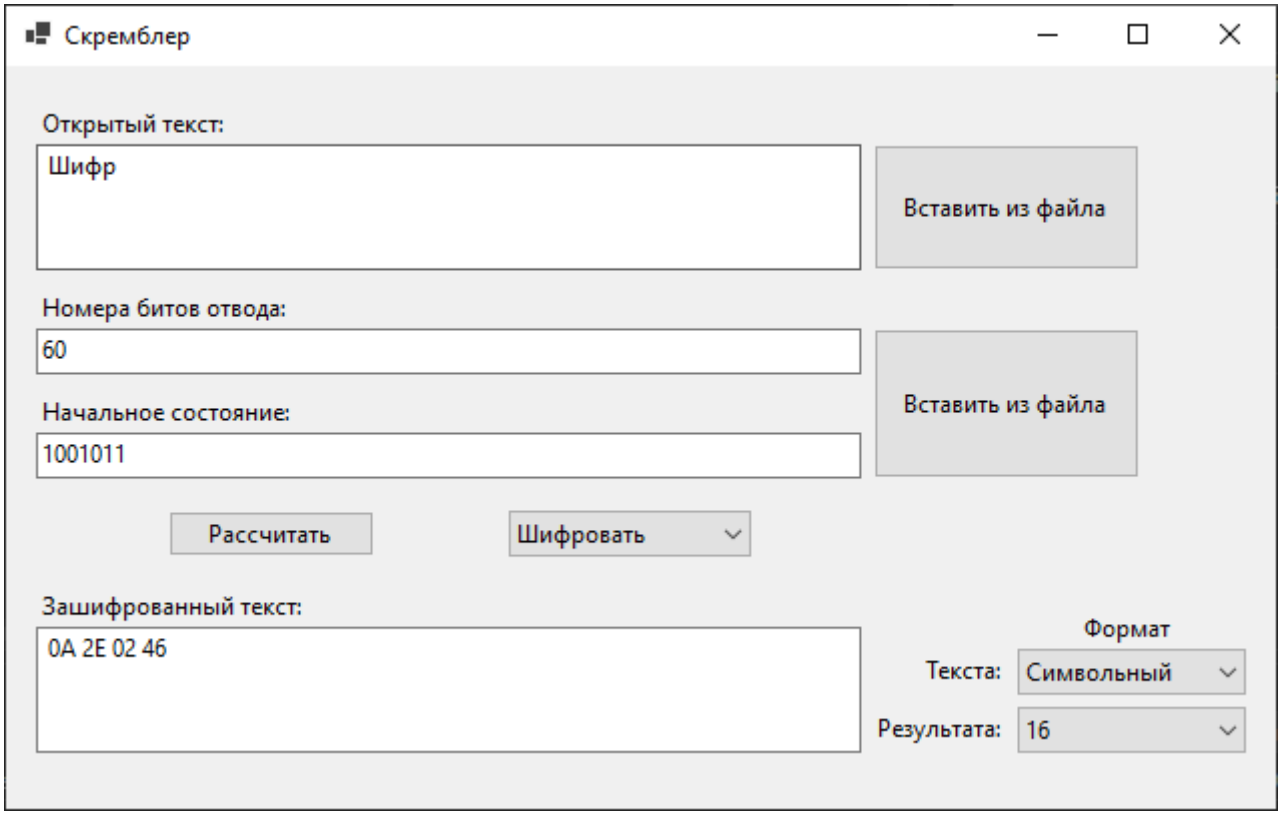
Тогда он может получить: $P_1 \oplus P_2 = C_1 \oplus C_2 = 1010111$;

Далее предположим, что злоумышленник знает первые три бита P_1. Пусть они равны 111. Тогда он с легкостью может определить первые три бита P_2 (101 XOR 111 = 010).

Таким образом, злоумышленник может получить часть открытого текста, не зная ключа. Продолжая этот процесс и используя дополнительные знания или предположения о структуре открытого текста, злоумышленник может восстановить все сообщение.

4) Исследовать генерируемые каждым скремблером последовательности псевдослучайных чисел при различных начальных значениях скремблера.

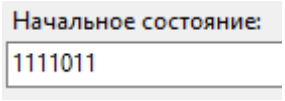
Рассмотрим теперь пример, где биты отвода определяются многочленом $f(x) = x^7 + x + 1$:




```
Последовательность до повтора: 11010010111000110
Гамма: 11010010 11000110 11110110 10110110
Период: 127
Критерий хи^2: True
Проверка на сбалансированность: True
Цикличность: 1: 0,5076923076923077 |
             2: 0,26153846153846155 |
             3: 0,1076923076923077 |
             4: 0,06153846153846154 |
             5: 0,03076923076923077 |
             6: 0,015384615384615385 |
             7: 0,015384615384615385
Корреляция: True
```

Отметим тот факт, что данный многочлен, в отличие от рассмотренного ранее, является примитивным, и последовательность, основанная на нем, независимо от начального состояния, всегда будет иметь максимальный период, равный $T = 2^7 - 1 = 127$.

Немного изменим начальное состояние скремблера:




```
Последовательность до повтора: 11011110111010110
Гамма: 11011110 11010110 11001001 00011100
Период: 127
Критерий хи^2: True
Проверка на сбалансированность: True
Цикличность: 1: 0,5 |
             2: 0,25 |
             3: 0,125 |
             4: 0,0625 |
             5: 0,03125 |
             6: 0,015625 |
             7: 0,015625
Корреляция: True
```

Получили удовлетворяющую всем свойствам последовательность.

7. Вывод

В ходе проведения лабораторной работы мы освоили два важных метода шифрования: однократное гаммирование и скремблирование с использованием LFSR.

Однократное гаммирование является абсолютно криптостойким методом шифрования, при условии соблюдения трех основных правил: полная случайность ключа, равенство длин ключа и открытого текста, а также однократное использование ключа. Это делает его идеальным выбором для обеспечения конфиденциальности в критически важных приложениях. Однако, несмотря на его криптостойкость, однократное гаммирование имеет свои недостатки. Главным из них является необходимость безопасного обмена ключами одинаковой длины с открытым текстом, что может быть сложно реализовать на практике.

С другой стороны, скремблирование с использованием LFSR представляет собой эффективный метод генерации псевдослучайных последовательностей, которые могут быть использованы в качестве ключей для гаммирования. Однако линейный характер генерируемых последовательностей делает этот метод уязвимым для атак. Для повышения уровня безопасности можно увеличить размер LFSR или использовать более сложные схемы генерации ключа.

8. Код программы

Код однократного гаммирования:

```
public partial class Form1 : Form
{
    private int previousIndex;
    Random random;

    public Form1()
    {
        InitializeComponent();

        comboBox1.Tag = textBox1;
        comboBox2.Tag = textBox2;
        comboBox3.Tag = textBox3;

        textBox1.Tag = comboBox1;
        textBox2.Tag = comboBox2;
        textBox3.Tag = comboBox3;

        comboBox1.SelectedIndex = 2;
        comboBox2.SelectedIndex = 2;
        comboBox3.SelectedIndex = 2;
        comboBox4.SelectedIndex = 0;
```

```
        random = new Random();
    }

    // Перевод символьной строки в двоичную
    public static string StringToBinary(string data)
    {

System.Text.Encoding.RegisterProvider(System.Text.CodePagesEncodingProvider.Instance);

        // Получаем байты строки в кодировке Windows-1251
        var bytes = Encoding.GetEncoding("windows-1251").GetBytes(data);

        var binary = new StringBuilder();

        foreach (byte b in bytes)
        {
            binary.AppendFormat("{0} ", Convert.ToString(b, 2).PadLeft(8, '0'));
        }

        if (binary.Length > 0)
        {
            binary.Remove(binary.Length - 1, 1);
        }

        return binary.ToString();
    }

    // Перевод двоичной строки в символьную
    public static string BinaryToString(string data)
    {

System.Text.Encoding.RegisterProvider(System.Text.CodePagesEncodingProvider.Instance);

        var text = new StringBuilder();

        var numbers = data.Split(' ');

        foreach (string number in numbers)
        {
            if (number.Length > 0)
            {
                var byteValue = Convert.ToByte(number, 2);

                // Запрещаем перевод в символьный вид, потому что в строке есть спец.
символы
                if (byteValue < 32)
                {
                    throw new Exception("В шифротексте присутствуют специальные
символы!");
                }

                text.Append(Encoding.GetEncoding("windows-1251").GetString(new byte[] {
byteValue }));
            }
        }

        return text.ToString();
    }

    // Перевод двоичной строки в шестнадцатеричную
```

```
    public static string BinaryToHex(string binary)
    {
        var hex = new StringBuilder();

        var numbers = binary.Split(' ');

        foreach (string number in numbers)
        {
            if (number.Length > 0)
            {
                var byteValue = Convert.ToByte(number, 2);
                hex.AppendFormat("{0} ", Convert.ToString(byteValue, 16).PadLeft(2,
'0').ToUpper());

            }
        }

        // Удаляем лишний пробел в конце строки
        if (hex.Length > 0)
        {
            hex.Remove(hex.Length - 1, 1);
        }

        return hex.ToString();
    }

    // Перевод шестнадцатеричной строки в двоичную
    public static string HexToBinary(string hex)
    {
        var binary = new StringBuilder();

        // Разделяем входную строку на отдельные числа с помощью пробела
        var numbers = hex.Split(' ');

        foreach (string number in numbers)
        {
            if (number.Length > 0)
            {
                var byteValue = Convert.ToByte(number, 16);
                binary.AppendFormat("{0} ", Convert.ToString(byteValue, 2).PadLeft(8,
'0'));
            }
        }

        // Удаляем лишний пробел в конце строки
        if (binary.Length > 0)
        {
            binary.Remove(binary.Length - 1, 1);
        }

        return binary.ToString();
    }

    // Код выполняется после нажатия на кнопку "Сгенерировать ключ"
    // Сначала узнаем размер будущего ключа
    // После генерируем рандомно в двоичном виде ключ
    // В зависимости от формата текстового поля, переводим ключ соответствующим образом
    // В конце концов вставляем ключ в текстовое поле
    private void button1_Click(object sender, EventArgs e)
    {
        int keySize;
```

```
        if (comboBox1.SelectedIndex == 2)
        {
            keySize = textBox1.Text.Length;
        }
        else
        {
            keySize = textBox1.Text.Split(' ').Length;
        }

        StringBuilder key = new StringBuilder();

        for (int i = 0; i < keySize; i++)
        {
            key.AppendFormat("{0} ", Convert.ToString(random.Next(32, 256),
2).PadLeft(8, '0'));
        }

        if (key.Length > 0)
        {
            key.Remove(key.Length - 1, 1);
        }

        if (comboBox2.SelectedIndex == 1)
        {
            textBox2.Text = BinaryToHex(key.ToString());
        }
        else if (comboBox2.SelectedIndex == 2)
        {
            textBox2.Text = BinaryToString(key.ToString());
        }
        else
        {
            textBox2.Text = key.ToString();
        }

    }

    // Обеспечиваем сохранение предыдущего значения переключателя для формата текста
    // Это необходимо, чтобы при переводе из одного формата в другой знать из какого
формата переводим
    private void comboBox_DropDown(object sender, EventArgs e)
    {
        previousIndex = (sender as ComboBox).SelectedIndex;
    }

    // Когда пользователь переключает формат определенного текстового поля
    // То необходимо в этом текстовом поле перевести текст соответствующим образом
    // Тут же обрабатываем невозможность перевода в символьный вид, если в тексте
присутствуют спец. символы
    private void comboBox_SelectedIndexChanged(object sender, EventArgs e)
    {
        ComboBox comboBox = sender as ComboBox;
        TextBox textBox = comboBox.Tag as TextBox;

        try
        {
            if (comboBox.SelectedIndex == 0 && previousIndex == 2)
            {
                textBox.Text = StringToBinary(textBox.Text);
            }
```

```
            else if (comboBox.SelectedIndex == 2 && previousIndex == 0)
            {
                textBox.Text = BinaryToString(textBox.Text);
            }
            else if (comboBox.SelectedIndex == 1 && previousIndex == 0)
            {
                textBox.Text = BinaryToHex(textBox.Text);
            }
            else if (comboBox.SelectedIndex == 0 && previousIndex == 1)
            {
                textBox.Text = HexToBinary(textBox.Text);
            }
            else if (comboBox.SelectedIndex == 1 && previousIndex == 2)
            {
                textBox.Text = BinaryToHex(StringToBinary(textBox.Text));
            }
            else if (comboBox.SelectedIndex == 2 && previousIndex == 1)
            {
                textBox.Text = BinaryToString(HexToBinary(textBox.Text));
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Символьный вид недоступен! " + ex.Message);
            comboBox.SelectedIndex = previousIndex;
        }

        previousIndex = comboBox.SelectedIndex;
    }

    // Событие при вводе текста во всех текстовых полях
    // В зависимости от формата разрешаем или запрещаем определенный пользовательский
ввод
    // Например, если формат двоичный, то разрешаем пользователю вводить только 1, 0 или
BackSpace и т.д.
    private void textBox_KeyPress(object sender, KeyPressEventArgs e)
    {
        TextBox textBox = sender as TextBox;
        ComboBox comboBox = textBox.Tag as ComboBox;

        if (comboBox.SelectedIndex == 0)
        {
            // Проверяем, что введены только 0 и 1
            if (e.KeyChar != '0' && e.KeyChar != '1' && !Char.IsControl(e.KeyChar))
            {
                e.Handled = true;
            }

            var numbers = textBox.Text.Split(' ');

            if (numbers.Length > 0 && numbers[numbers.Length - 1].Length == 8 &&
!Char.IsControl(e.KeyChar))
            {
                textBox.AppendText(" ");
            }

        }
        else if (comboBox.SelectedIndex == 1)
        {
            // Проверяем, что введены только 0-9, A-F, a-f или Backspace
            if (!char.IsDigit(e.KeyChar) && !char.IsControl(e.KeyChar) &&
```

```
                !(e.KeyChar >= 'A' && e.KeyChar <= 'F') &&
                !(e.KeyChar >= 'a' && e.KeyChar <= 'f'))
            {
                e.Handled = true;
            }

            var numbers = textBox.Text.Split(' ');

            if (numbers.Length > 0 && numbers[numbers.Length - 1].Length == 2 &&
!Char.IsControl(e.KeyChar))
            {
                textBox.AppendText(" ");
            }

            e.KeyChar = char.ToUpper(e.KeyChar);
        }
    }

    // Событие при нажатии на кнопку "Рассчитать"
    private void button2_Click(object sender, EventArgs e)
    {
        if (textBox1.Text.Length == 0)
        {
            MessageBox.Show("Текстовое поле пустое!");
            return;
        }

        // Приводим строки в текстбоксах 1 и 2 к двоичному виду
        string text1 = textBox1.Text;
        string text2 = textBox2.Text;

        text1 = comboBox1.SelectedIndex == 1 ? HexToBinary(text1) :
comboBox1.SelectedIndex == 2 ? StringToBinary(text1) : text1;
        text2 = comboBox2.SelectedIndex == 1 ? HexToBinary(text2) :
comboBox2.SelectedIndex == 2 ? StringToBinary(text2) : text2;

        // Выполняем операцию XOR между строками
        string[] text1Bits = text1.Split(' ');
        string[] text2Bits = text2.Split(' ');

        if (text1Bits.Length != text2Bits.Length)
        {
            MessageBox.Show("Неравенство длины ключа и текста");
            return;
        }

        StringBuilder result = new StringBuilder();

        for (int i = 0; i < text1Bits.Length; i++)
        {
            int bit1 = Convert.ToInt32(text1Bits[i], 2);
            int bit2 = Convert.ToInt32(text2Bits[i], 2);

            int xor = bit1 ^ bit2;

            result.AppendFormat("{0} ", Convert.ToString(xor, 2).PadLeft(8, '0'));
        }

        // Удаляем лишний пробел в конце строки
        if (result.Length > 0)
        {
```

```
            result.Remove(result.Length - 1, 1);
        }

        // Преобразуем результат в выбранный вид и вставляем его в результирующее
текстовое поле
        if (comboBox3.SelectedIndex == 1)
        {
            textBox3.Text = BinaryToHex(result.ToString());
        }
        else if (comboBox3.SelectedIndex == 2)
        {
            try
            {
                textBox3.Text = BinaryToString(result.ToString());
            }
            catch (Exception ex)
            {
                MessageBox.Show("Символьный вид недоступен! " + ex.Message);
                comboBox3.SelectedIndex = 0;
                textBox3.Text = result.ToString();
            }
        }
        else
        {
            textBox3.Text = result.ToString();
        }

        // Сохраняем данные в файлы
        // Если шифруем - открытый текст, ключ и шифротекст
        // Если дешифруем - результат дешифрования
        if (comboBox4.SelectedIndex == 0)
        {
            SaveToFile(textBox1.Text, comboBox1.SelectedIndex, "plaintext.txt");
            SaveToFile(textBox2.Text, comboBox2.SelectedIndex, "key.txt");
            SaveToFile(textBox3.Text, comboBox3.SelectedIndex, "ciphertext.txt");
        }
        else
        {
            SaveToFile(textBox3.Text, comboBox3.SelectedIndex, "decryptedtext.txt");
        }

    }

    // Когда меняем тип операции (шифрование на дешифрование и наоборот)
    // Очищаем все окно и меняем несколько label соответствующим образом
    private void comboBox4_SelectedIndexChanged(object sender, EventArgs e)
    {
        textBox1.Text = "";
        textBox2.Text = "";
        textBox3.Text = "";
        comboBox1.SelectedIndex = 2;
        comboBox2.SelectedIndex = 2;
        comboBox3.SelectedIndex = 2;

        if (comboBox4.SelectedIndex == 0)
        {
            label1.Text = "Открытый текст:";
            label3.Text = "Зашифрованный текст:";
        }
        else
        {
```

```
            label1.Text = "Зашифрованный текст:";
            label3.Text = "Расшифрованный текст:";
        }

    }

    // Если пытаемся вставить из файла в текстовое поле открытый текст или шифротекст
    // Зависит от типа операции
    private void button3_Click(object sender, EventArgs e)
    {
        try
        {
            var fileData = LoadFromFile(comboBox4.SelectedIndex == 0 ? "plaintext.txt" :
"ciphertext.txt");
            comboBox1.SelectedIndex = fileData.Item2;
            textBox1.Text = fileData.Item1;
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    // Вставка ключа из файла
    private void button4_Click(object sender, EventArgs e)
    {
        try
        {
            var fileData = LoadFromFile("key.txt");
            comboBox2.SelectedIndex = fileData.Item2;
            textBox2.Text = fileData.Item1;
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    // Загрузка данных из файла
    // (string, int) означает, что возвращаем данные и номер, означающий их формат
    // номер 2 - символьный вид, 1 - шестнадцатеричный вид, 0 - двоичный
    public static (string, int) LoadFromFile(string fileName)
    {
        string[] lines = System.IO.File.ReadAllLines(fileName);

        if (lines.Length == 0)
        {
            throw new Exception("Файл пуст!");
        }

        return (string.Join("\n", lines.Take(lines.Length - 1)),
int.Parse(lines[lines.Length - 1]));
    }

    // Сохранение в файл данных (вместе с индексом формата)
    public static void SaveToFile(string text, int index, string fileName)
    {
        // Создаем список строк для записи в файл
        List<string> lines = new List<string>();
        lines.Add(text);
        lines.Add(index.ToString());
```

```
        // Записываем строки в файл
        System.IO.File.WriteAllLines(fileName, lines);
    }

}
```

Код скремблера:

```
public partial class Form2 : Form
{
    private int previousIndex;

    public Form2()
    {
        InitializeComponent();

        comboBox1.Tag = textBox4;
        comboBox3.Tag = textBox5;

        textBox4.Tag = comboBox1;
        textBox5.Tag = comboBox3;

        comboBox1.SelectedIndex = 2;
        comboBox3.SelectedIndex = 2;
        comboBox4.SelectedIndex = 0;
    }

    // Получаем бит из числа Value, номер бита number (нумерация справа налево)
    private static int GetBit(uint Value, int number)
    {
        return Convert.ToInt32((Value & 1 << number) != 0);
    }

    // Получаем последовательность с помощью генератора в формате строки
    // Просто генерация битов gammaLength раз
    private string LFSRForGamma(uint startState, int[] feedbackPoints, int
startStateLength, int gammaLength)
    {
        string bitsResult = ""; // Строка с результирующей последовательностью битов

        uint nextState = startState; // Текущее состояние регистра

        int lastBit = startStateLength - 1; // Запоминаем номер первого бита

        while (bitsResult.Length != gammaLength)
        {
            // Получаем крайний бит (он справа): 1001 -> 1 и сохраняем его в результат
            bitsResult += GetBit(nextState, 0);

            // Начало операции XOR, берем один из битов отвода
            int xor = GetBit(nextState, feedbackPoints[0]);

            // Продолжаем XOR с остальными отводами
            for (int i = 1; i < feedbackPoints.Length; i++)
            {
                xor ^= GetBit(nextState, feedbackPoints[i]);
            }
```

```
            // Сдвигаем все биты вправо на 1 позицию
            nextState >>= 1;

            // В ячейку 0 записываем бит, ранее вычисленный функцией обратной связи XOR
            // Если xor равен 0, то нету смысла что-либо делать, там и так 0 записан
            if (xor == 1)
            {
                nextState |= 1u << lastBit;
            }

        }

        return bitsResult;
    }

    // Тот же самый генератор, что и выше, но тут мы получаем последовательность до
зацикливания
    // Позволяет определить период = длине этой последовательности
    private string LFSRWithoutRepeat(uint startState, int[] feedbackPoints, int
startStateLength)
    {
        StringBuilder bitsResult = new StringBuilder(); // Строка с результирующей
последовательностью битов

        uint nextState = startState; // Текущее состояние регистра

        int lastBit = startStateLength - 1; // Запоминаем номер первого бита

        // Создаем словарь для хранения всех состояний, которые когда-либо были, и их
позиций
        Dictionary<uint, int> previousStates = new Dictionary<uint, int>();
        previousStates.Add(startState, 0);

        while (true)
        {
            //textBox3.AppendText(Environment.NewLine + Convert.ToString(nextState, 2));
            // Получаем крайний бит (он справа): 1001 -> 1 и сохраняем его в результат
            bitsResult.Append(GetBit(nextState, 0));

            // Начало операции XOR, берем один из битов отвода
            int xor = GetBit(nextState, feedbackPoints[0]);

            // Продолжаем XOR с остальными отводами
            for (int i = 1; i < feedbackPoints.Length; i++)
            {
                xor ^= GetBit(nextState, feedbackPoints[i]);
            }

            // Сдвигаем все биты вправо на 1 позицию
            nextState >>= 1;

            // В ячейку 0 записываем бит, ранее вычисленный функцией обратной связи XOR
            // Если xor равен 0, то нету смысла что-либо делать, там и так 0 записан
            if (xor == 1)
            {
                nextState |= 1u << lastBit;
            }

            // Сверяем текущее состояние с словарем всех предыдущих состояний
            if (previousStates.ContainsKey(nextState))
            {
```

```
                // Возвращаем подстроку, начиная с позиции, соответствующей повторному
состоянию
                return bitsResult.ToString().Substring(previousStates[nextState]);
            }
            else
            {
                // Если текущего состояния еще не было, добавляем его в словарь
                previousStates.Add(nextState, bitsResult.Length);
            }
        }
    }

    // Вызываем генератор для получения последовательности до зацикливания
    // Но перед этим приводим биты отвода в правильный вид
    public string GetLFSRSequence(string beginState, string tabBitNumbers)
    {
        int[] array = new int[tabBitNumbers.Length];

        for (int i = 0; i < array.Length; i++)
        {
            array[i] = Math.Abs(Convert.ToInt32(tabBitNumbers[i].ToString()) -
beginState.Length + 1);
        }

        return LFSRWithoutRepeat(Convert.ToUInt32(beginState, 2), array,
beginState.Length);
    }

    // Вызываем генератор для получения гаммы
    // Перед этим приводим биты отвода в правильный вид
    // Также добавляем гамме пробелы после каждого 8-го бита
    public string GetGamma(string binaryOpenText, string tabBitNumbers, string
beginState)
    {
        int[] array = new int[tabBitNumbers.Length];

        for (int i = 0; i < array.Length; i++)
        {
            array[i] = Math.Abs(Convert.ToInt32(tabBitNumbers[i].ToString()) -
beginState.Length + 1);
        }

        binaryOpenText = binaryOpenText.Replace(" ", "");

        StringBuilder gamma = new
StringBuilder(LFSRForGamma(Convert.ToUInt32(beginState, 2), array, beginState.Length,
binaryOpenText.Length));


        for (int i = 7; i < gamma.Length; i += 9)
        {
            gamma.Insert(i + 1, " ");
        }

        return gamma.ToString();
    }

    // Критическое значение для хи-квадрат
    public double CalculateCriticalValue(double nu)
    {
```

```
        // Критические значения статистики Skp при заданных степени свободы, уровень
значимости - 0.05
        double[] xi2 = { 3.841, 5.991, 7.815, 9.488, 11.070, 12.592,
            14.067, 15.507, 16.919, 18.307, 19.675, 21.026, 22.362,
            23.685, 24.996, 26.296, 27.587, 28.869, 30.144, 31.410,
            32.671, 33.924, 35.172, 36.415, 37.652, 38.885, 40.113,
            41.337, 42.557, 43.773 };

        // Для ню меньше 30 берем значение из массива, всё что выше вычисляется по
формуле
        if (nu <= 30)
        {
            return xi2[(int)nu - 1];
        }
        else
        {
            double xp = 1.64;
            double criticalValue = nu + Math.Sqrt(2 * nu) * xp + 2.0 / 3.0 *
Math.Pow(xp, 2) - 2.0 / 3.0 + 1.0 / Math.Sqrt(nu);
            return criticalValue;
        }

    }

    // Получение статистики хи-квадрат
    public double ChiSquareTest(string sequence)
    {
        int numberOfZeros = sequence.Count(c => c == '0');
        int numberOfOnes = sequence.Count(c => c == '1');

        double expectedCount = sequence.Length / 2.0;

        double chiSquare = Math.Pow(numberOfZeros - expectedCount, 2) / expectedCount +
                           Math.Pow(numberOfOnes - expectedCount, 2) / expectedCount;

        return chiSquare;
    }

    // Проверка последовательности на сбалансированность
    public bool IsBalanced(string sequence)
    {
        int countOfOnes = sequence.Count(c => c == '1');
        int countOfZeros = sequence.Count(c => c == '0');

        // Последовательность считается сбалансированной, если количество нулей и единиц
примерно одинаково.
        // Разница между количеством нулей и единиц не должна превышать 2% от общего
количества символов.
        double differenceThreshold = sequence.Length * 0.02;

        return Math.Abs(countOfOnes - countOfZeros) <= differenceThreshold;
    }

    // Проверка на цикличность
    public Dictionary<int, double> GetCycleProportions(string sequence)
    {
        var cycles = new Dictionary<int, int>();
        int currentCycleLength = 1;
        char currentChar = sequence[0];

        for (int i = 1; i < sequence.Length; i++)
```

```
        {

            if (sequence[i] == currentChar)
            {
                currentCycleLength++;
            }
            else
            {
                if (cycles.ContainsKey(currentCycleLength))
                {
                    cycles[currentCycleLength]++;
                }
                else
                {
                    cycles[currentCycleLength] = 1;
                }

                currentChar = sequence[i];
                currentCycleLength = 1;
            }

        }

        // Добавляем последний цикл
        if (cycles.ContainsKey(currentCycleLength))
        {
            cycles[currentCycleLength]++;
        }
        else
        {
            cycles[currentCycleLength] = 1;
        }

        // Подсчитываем общее количество циклов
        int totalCycles = cycles.Sum(c => c.Value);

        // Создаем словарь для хранения долей каждого цикла
        var cycleProportions = new Dictionary<int, double>();

        foreach (var cycle in cycles)
        {
            cycleProportions[cycle.Key] = (double)cycle.Value / totalCycles;
        }

        return cycleProportions;
    }

    // Форматируем строку для отображения цикличности в файле stats.txt
    public string GetCycleProportionsString(Dictionary<int, double> cycleProportions)
    {
        // Сортируем словарь по возрастанию ключа (длины цикла)
        var sortedProportions = cycleProportions.OrderBy(c => c.Key);

        // Преобразуем каждую пару ключ-значение в строку формата "<длина полосы>:
<доля>"
        var proportionStrings = sortedProportions.Select(c => $"{c.Key}: {c.Value}");

        // Объединяем все строки в одну, разделяя их запятыми
        string result = string.Join(" | ", proportionStrings);

        return result;
```

```
    }

    // Проверка на корреляцию
    public bool IsCorrelated(string sequence)
    {
        // Создаем циклически сдвинутую копию последовательности
        string shiftedSequence = sequence[sequence.Length - 1] + sequence.Substring(0,
sequence.Length - 1);

        int matches = 0;
        int mismatches = 0;

        // Сравниваем исходную последовательность и сдвинутую копию
        for (int i = 0; i < sequence.Length; i++)
        {
            if (sequence[i] == shiftedSequence[i])
            {
                matches++;
            }
            else
            {
                mismatches++;
            }
        }

        // Для удовлетворения условия корреляции число совпадений должно отличаться
        // От числа несовпадений не более чем на 2% от общей длины последовательности
        return Math.Abs(matches - mismatches) <= 0.02 * sequence.Length;
    }

    public void SaveToFile(string text, string fileName)
    {
        System.IO.File.WriteAllText(fileName, text);
    }

    // При нажатии на кнопку "Рассчитать"
    private void button4_Click(object sender, EventArgs e)
    {
        // Получаем гамму с помощью генератора, учитывая длину открытого
текста/шифротекста в 2-м виде
        var binaryOpenText = comboBox1.SelectedIndex == 1 ?
Form1.HexToBinary(textBox4.Text) : comboBox1.SelectedIndex == 2 ?
Form1.StringToBinary(textBox4.Text) : textBox4.Text;
        var gamma = GetGamma(binaryOpenText, textBox2.Text, textBox1.Text);

        // Выполняем операцию XOR между строками
        var openTextBits = binaryOpenText.Split(' ');
        var gammaBits = gamma.Split(' ');

        StringBuilder xorResult = new StringBuilder();

        for (int i = 0; i < openTextBits.Length; i++)
        {
            int bit1 = Convert.ToInt32(openTextBits[i], 2);
            int bit2 = Convert.ToInt32(gammaBits[i], 2);

            int xor = bit1 ^ bit2;

            xorResult.AppendFormat("{0} ", Convert.ToString(xor, 2).PadLeft(8, '0'));
        }
```

```
        // Удаляем лишний пробел в конце строки
        if (xorResult.Length > 0)
        {
            xorResult.Remove(xorResult.Length - 1, 1);
        }

        // Преобразуем результат в выбранный вид и вставляем его в текстбокс
        if (comboBox3.SelectedIndex == 1)
        {
            textBox5.Text = Form1.BinaryToHex(xorResult.ToString());
        }
        else if (comboBox3.SelectedIndex == 2)
        {
            try
            {
                textBox5.Text = Form1.BinaryToString(xorResult.ToString());
            }
            catch (Exception ex)
            {
                MessageBox.Show("Символьный вид недоступен! " + ex.Message);
                comboBox3.SelectedIndex = 0;
                textBox5.Text = xorResult.ToString();
            }
        }
        else
        {
            textBox5.Text = xorResult.ToString();
        }

        // Сохраняем открытый текст, параметры генератора и шифротекст, если происходит
шифрование
        // При дешифровании - сохраняем только расшифрованный текст
        if (comboBox4.SelectedIndex == 0)
        {
            SaveToFile(textBox4.Text + "\n" + comboBox1.SelectedIndex, "plaintext.txt");
            SaveToFile(textBox5.Text + "\n" + comboBox3.SelectedIndex,
"ciphertext.txt");
            SaveToFile(textBox1.Text + " " + textBox2.Text, "beginstate.txt");
        }
        else
        {
            SaveToFile(textBox5.Text + "\n" + comboBox3.SelectedIndex,
"decryptedtext.txt");
        }

        // Анализ последовательности, сгенерированной генератором и запись в файл
stats.txt
        var lfseSequence = GetLFSRSequence(textBox1.Text, textBox2.Text);

        SaveToFile("Последовательность до повтора: " + lfseSequence
            + "\nГамма: " + gamma
            + "\nПериод: " + lfseSequence.Length
            + "\nКритерий хи^2: " + (ChiSquareTest(lfseSequence) <
CalculateCriticalValue(lfseSequence.Length))
            + "\nПроверка на сбалансированность: " + IsBalanced(lfseSequence)
            + "\nЦикличность: " +
GetCycleProportionsString(GetCycleProportions(lfseSequence))
            + "\nКорреляция: " + IsCorrelated(lfseSequence), "stats.txt");
    }

    // Событие при вводе текста во всех текстовых полях
```

```
    // В зависимости от формата разрешаем или запрещаем определенный пользовательский
ввод
    // Например, если формат двоичный, то разрешаем пользователю вводить только 1, 0 или
BackSpace и т.д.
    private void textBox_KeyPress(object sender, KeyPressEventArgs e)
    {
        System.Windows.Forms.TextBox textBox = sender as System.Windows.Forms.TextBox;
        System.Windows.Forms.ComboBox comboBox = textBox.Tag as
System.Windows.Forms.ComboBox;

        if (comboBox.SelectedIndex == 0)
        {
            // Проверяем, что введены только 0 и 1
            if (e.KeyChar != '0' && e.KeyChar != '1' && !Char.IsControl(e.KeyChar))
            {
                e.Handled = true;
            }

            var numbers = textBox.Text.Split(' ');

            if (numbers.Length > 0 && numbers[numbers.Length - 1].Length == 8 &&
!Char.IsControl(e.KeyChar))
            {
                textBox.AppendText(" ");
            }

        }
        else if (comboBox.SelectedIndex == 1)
        {
            // Проверяем, что введены только 0-9, A-F, a-f или Backspace
            if (!char.IsDigit(e.KeyChar) && !char.IsControl(e.KeyChar) &&
                !(e.KeyChar >= 'A' && e.KeyChar <= 'F') &&
                !(e.KeyChar >= 'a' && e.KeyChar <= 'f'))
            {
                e.Handled = true;
            }

            var numbers = textBox.Text.Split(' ');

            if (numbers.Length > 0 && numbers[numbers.Length - 1].Length == 2 &&
!Char.IsControl(e.KeyChar))
            {
                textBox.AppendText(" ");
            }

            e.KeyChar = char.ToUpper(e.KeyChar);
        }
    }

    // Обеспечиваем сохранение предыдущего значения переключателя для формата текста
    // Это необходимо, чтобы при переводе из одного формата в другой знать из какого
формата переводим
    private void comboBox_DropDown(object sender, EventArgs e)
    {
        previousIndex = (sender as System.Windows.Forms.ComboBox).SelectedIndex;
    }

    // Когда пользователь переключает формат определенного текстового поля
    // То необходимо в этом текстовом поле перевести текст соответствующим образом
    // Тут же обрабатываем невозможность перевода в символьный вид, если в тексте
присутствуют спец. символы
```

```
    private void comboBox_SelectedIndexChanged(object sender, EventArgs e)
    {
        System.Windows.Forms.ComboBox comboBox = sender as
System.Windows.Forms.ComboBox;
        System.Windows.Forms.TextBox textBox = comboBox.Tag as
System.Windows.Forms.TextBox;

        try
        {

            if (comboBox.SelectedIndex == 0 && previousIndex == 2)
            {
                textBox.Text = Form1.StringToBinary(textBox.Text);
            }
            else if (comboBox.SelectedIndex == 2 && previousIndex == 0)
            {
                textBox.Text = Form1.BinaryToString(textBox.Text);
            }
            else if (comboBox.SelectedIndex == 1 && previousIndex == 0)
            {
                textBox.Text = Form1.BinaryToHex(textBox.Text);
            }
            else if (comboBox.SelectedIndex == 0 && previousIndex == 1)
            {
                textBox.Text = Form1.HexToBinary(textBox.Text);
            }
            else if (comboBox.SelectedIndex == 1 && previousIndex == 2)
            {
                textBox.Text = Form1.BinaryToHex(Form1.StringToBinary(textBox.Text));
            }
            else if (comboBox.SelectedIndex == 2 && previousIndex == 1)
            {
                textBox.Text = Form1.BinaryToString(Form1.HexToBinary(textBox.Text));
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Символьный вид недоступен! " + ex.Message);
            comboBox.SelectedIndex = previousIndex;
        }

        previousIndex = comboBox.SelectedIndex;
    }

    // Получаем из файла данные скремблера (начальное состояние и номера битов отвода)
    public (string, string) LoadBeginState(string fileName)
    {
        string[] lines = System.IO.File.ReadAllLines(fileName);

        if (lines.Length == 0)
        {
            throw new Exception("Файл пуст!");
        }

        var fileData = lines[0].Split(" ");

        return (fileData[0], fileData[1]);
    }

    // Вставка данных скремблера в текстовые поля
    private void button2_Click(object sender, EventArgs e)
```

```
    {
        var fileData = LoadBeginState("beginstate.txt");
        textBox1.Text = fileData.Item1;
        textBox2.Text = fileData.Item2;
    }

    // Если пытаемся вставить из файла в текстовое поле открытый текст или шифротекст
    // Зависит от типа операции
    private void button3_Click(object sender, EventArgs e)
    {
        try
        {
            var fileData = Form1.LoadFromFile(comboBox4.SelectedIndex == 0 ?
"plaintext.txt" : "ciphertext.txt");
            comboBox1.SelectedIndex = fileData.Item2;
            textBox4.Text = fileData.Item1;
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    // Когда меняем тип операции (шифрование на дешифрование и наоборот)
    // Очищаем все окно и меняем несколько label соответствующим образом
    private void comboBox4_SelectedIndexChanged(object sender, EventArgs e)
    {
        textBox1.Text = "";
        textBox2.Text = "";
        textBox4.Text = "";
        textBox5.Text = "";
        comboBox1.SelectedIndex = 2;
        comboBox3.SelectedIndex = 2;

        if (comboBox4.SelectedIndex == 0)
        {
            label4.Text = "Открытый текст:";
            label9.Text = "Зашифрованный текст:";
        }
        else
        {
            label4.Text = "Зашифрованный текст:";
            label9.Text = "Расшифрованный текст:";
        }
    }

    // Разрешаем в текстовом поле для начального состояния ввод только 1 или 0 или
BackSpace
    private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
    {
        if (e.KeyChar != '0' && e.KeyChar != '1' && !char.IsControl(e.KeyChar))
        {
            e.Handled = true;
        }
    }

    // Разрешаем в текстовом поле для битов отвода ввод только цифр и BackSpace
    private void textBox2_KeyPress(object sender, KeyPressEventArgs e)
    {
        if (!char.IsDigit(e.KeyChar) && !char.IsControl(e.KeyChar))
        {
```

```
            e.Handled = true;
        }
    }
}
```