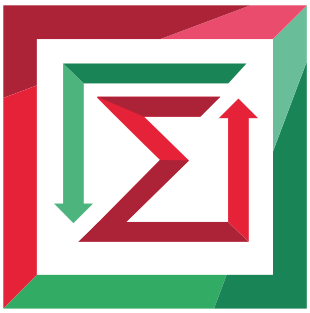
Министерство образования и науки Российской Федерации

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

Кафедра теоретической и прикладной информатики

Лабораторная работа № 2
по дисциплине «Информационная безопасность»



ФАКУЛЬТЕТ:	ПМИ
ГРУППА:	ПМИ-82
СТУДЕНТЫ:	Хайдаев К.Е Зяблицева У.П.
ВАРИАНТ:	2
ПРЕПОДАВАТЕЛЬ:	Авдеенко Т.В.

Новосибирск

2022

1. Цель работы:

Изучить существующие алгоритмы вычисления дайджестов сообщений и написать программу, реализующую заданный алгоритм хэширования.

2. Ход работы:

1) **Задание**:

Реализовать приложение с графическим интерфейсом, позволяющее выполнять следующие действия.

1. Генерировать псевдослучайную последовательность с помощью заданного в варианте алгоритма:

1) все входные параметры генератора должны задаваться из файла или вводиться в приложении;
2) сгенерированная последовательность, состоящая из 0 и 1, должна сохраняться в файл;

2. Проверять полученную псевдослучайную последовательность на равномерность и случайность с помощью трех рассмотренных тестов:

1. результат проверки каждого теста должен отображаться в приложении;
2. все вычисляемые промежуточные значения (все шаги алгоритма теста) могут отображаться в приложении или сохраняться в файл.

С помощью реализованного приложения выполнить следующие задания.

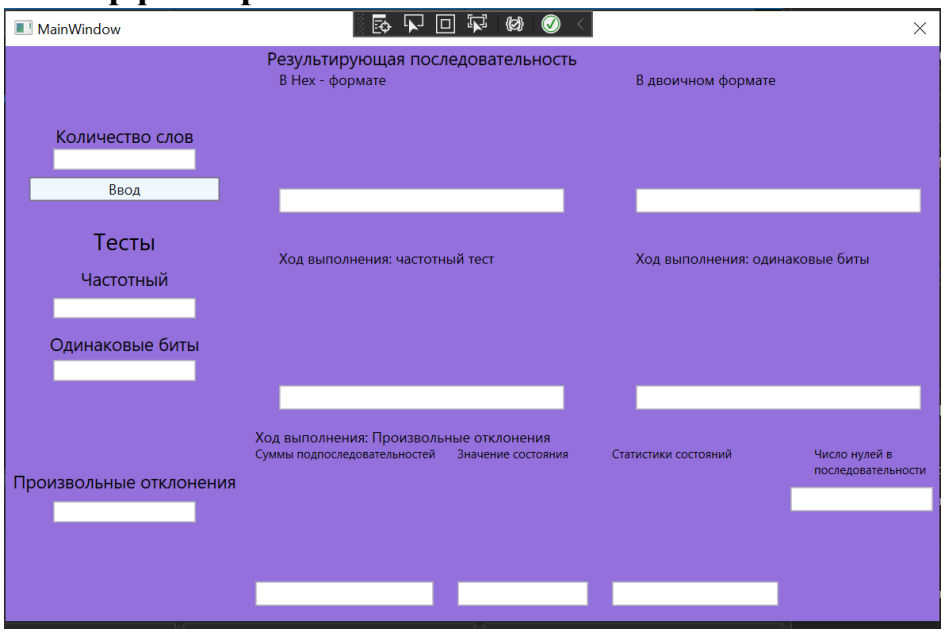
3. Протестировать правильность работы разработанного приложения.
4. Сгенерировать последовательность из не менее 10 000 бит и исследовать ее на равномерность и случайность.
5. Сделать вывод о случайности сгенерированной последовательности и о возможности ее использования в качестве криптографически безопасной псевдослучайной последовательности.

3. Вариант: Алгоритм ANSI X9.17

4. Описание разработанного программного средства

Разработанная программа способна генерировать псевдослучайную последовательность чисел заданной длины. Программа выводит сгенерированную последовательность в 16-ричном (для наглядности отличия 64-битных чисел друг от друга) и в двоичном формате, так же занося двоичный формат в выходной файл. Программа тестирует последовательность на случайность и равномерность применением трёх тестов и выводит как результаты тестов, так и их промежуточные результаты.

5. Интерфейс приложения



6. Исследования:

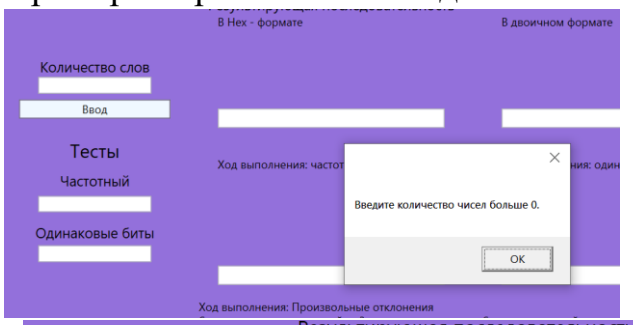
6.1) Демонстрация работоспособности на примере хэширования нескольких файлов.

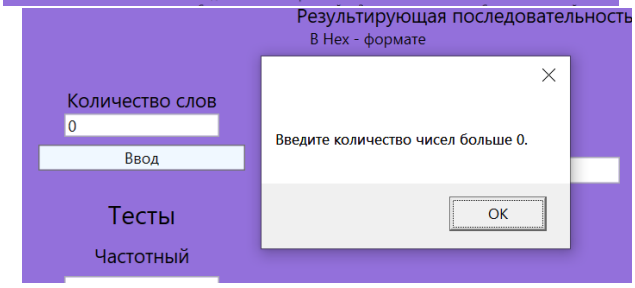
№ теста	Длина сообщения	Результаты		Частотный тест	Тест на последовательность одинаковых бит	Расширенный тест на отклонения
		16-ричная форма	2-чная форма			
1	2	bccc5ed626046afe9cceee895d70c197	10111100 1100110 0010111 1011010 1100010 0110000 0010001 1010101 1111110 1001110 0110011 1011101 1101000 1001010 1110101 1100001 1000001 1001011 1	Пройден	Пройден	Пройден
2	10	444987a9e5de38d9b2c6abafbee7c88ca4d74487093661c61e5e853f93c3bbfcd033a88c925bbea753ef4ff0562	100010001 0010011 0000111 1010100 1111001 0111011 1100011 1000110 1100110 1100101 1000110 1010101	Пройден	Пройден	Пройден

		bcede 799424a098 1fb9cb b4db18496f 08ddff d931f2a960 5dd948 c46d75c372 20f71b	1101011 1110111 1101110 0111110 0100010 0011001 0100100 1101011 1010001 0010000 1110000 1001001 1011001 1000011 1000110 1111001 0111101 0000101 0011111 1100100 1111000 0111011 1011111 1110011 0100000 0110011 1010100 0100011 0010010 0100101 1011101 1111010 1001111 0100111 1101111 0100111 1111100 0001010 1100010 1011110 0111011 0111101 1110011 0010100 0010010 0101000 0010011 0000001 1111101 1100111 0010111 0110100 1101101 1000110 0001001 0010110 1111000 0100011 0111011 1111111 1101100 1001100 0111110 0101010 1001011 0000001 0111011 1011001 0100100 0110001 0001101 1010111 0101110 0001101 1100100 0100000 1111011 1000110 11			

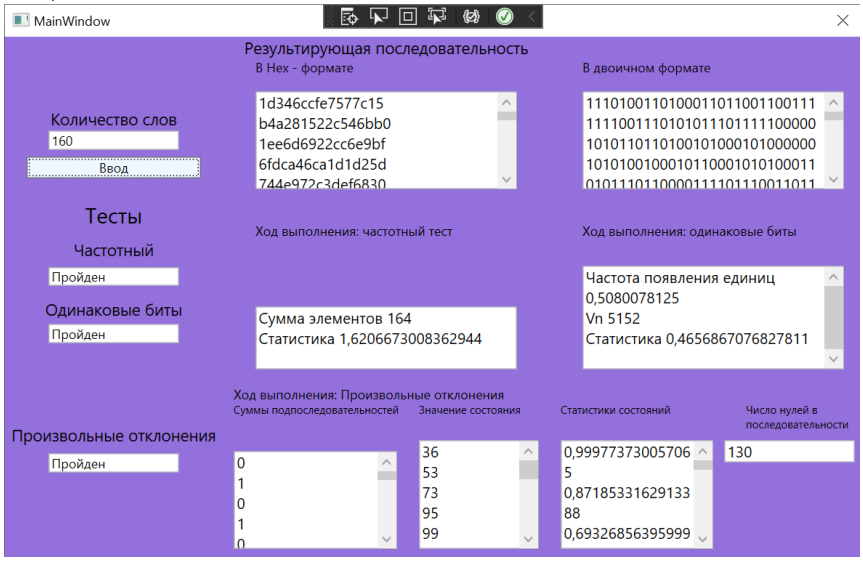
3	50			Пройден	Пройден	Пройден

Пример неправильного ввода:





6.2) **Исследование последовательности длиной более 10000 бит**.



7. Шаги алгоритма:

ANSI_X9_17

1. Из программы (MainWindow.xaml.cs) получаем количество 64-битных слов. Запускаем алгоритм генерации ANSI_X9_17

```
string mess = WordCountTextBox.Text; // получаем количество 64-битных чисел
 ulong[] ansi = aNSI_X9_17.ANSI(Convert.ToInt32(mess)); // массив псевдослучайных чисел
```

2. Выполнение алгоритма ANSI_X9_17 происходит в файле ANSI_X9_17.cs.

a) Фиксируем значение даты и времени (98): `DateTime date1 = DateTime.Now; // структура для извлечения даты и времени`

b) Для i = 1,m. Вычисляем значение i-го входного слова. Вычисляем значение нового параметра s1. В результате предыдущего шага формируется выходная последовательность из m слов x1, x2…xm.

```
 for (int i = 0; i < m; i++) // основной цикл
{

                //---------------------Для алгоритма 3DES---------------------------------
 -------------------------------
                byte[] d = BitConverter.GetBytes((ulong)date1.ToBinary()); // получаем
 дату и время в 64-битном формате
                byte[] IV = new byte[128]; // 128-битный вектор инициализации для
 шифрования чисел 3DES
                rand.NextBytes(IV); // получаем случайный вектор (использование случайного
 вектора позволяет избежать дублирующихся блоков в шифротексте)
                byte[] buf = TriDES(d, key_gen(), IV); // key_gen - генерирует составной
 128 битный ключ
                //------------------------------------------------------------------------
 -------------------------------

                ulong b1 = BitConverter.ToUInt64(buf, 0); // преобразуем зашифрованную
 дату и время в ulong для битовых операций
                ulong s = BitConverter.ToUInt64(s_0, 0); //  преобразуем секретное число в
 ulong для битовых операций

                //---------------------вычисляем новое значение параметра Si--------------
 -------------------------------
                s ^= b1; // проводим xor операцию с секретным числом и зашифрованными
 датой, временем
                //------------------------------------------------------------------------
 -------------------------------

                buf = BitConverter.GetBytes(s); // переводим результат xor обратно в
 массив байтов для повторного шифрования

                rand.NextBytes(IV); // получаем новый случайный вектор инициализации
                buf = TriDES(buf, key_gen(), IV); // шифруем результат xor

                //----------------------Вычисляем значение i-го выходного слова-----------
 -------------------------------
                X[i] = BitConverter.ToUInt64(buf, 0); // получаем одно из результирующих
 псевдослучайных чисел
                b1 ^= X[i]; // проводим операцию xor с результирующим числом и предыдущим
 xor
                //------------------------------------------------------------------------
 -------------------------------

                buf = BitConverter.GetBytes(b1); // переводим xor в массив байтов
                rand.NextBytes(IV); // шифруем результат второго xor чтобы получить новое
 начальное случайное число для следующей итерации цикла
            }
            return X;
```

8. Код программы:

Methods.cs

```
public static class Methods
   {
        public static int[] num_convert(ulong[] X) // перевод массива чисел ulong в int массив
 0 и 1
        {
            int[] bin_seq = new int[X.Length * 64]; // результирующий массив длиной в 64*число
 элементов во входном массиве
            int k = 0;
           for (int i = 0; i < X.Length; i++)
            {
                BitArray buf = new BitArray(BitConverter.GetBytes(X[i])); // превод элемента
 массива ulong в масив байтов, а затем в массив битов
                for (int j = 0; j < buf.Length; j++)
                {
                    if (buf[j])
                        bin_seq[k] = 1;
                    else
                        bin_seq[k] = 0;
                    k++;
                }
            }
            return bin_seq;
        }
        public static void Write(string fileName, string[] outw)  // Запись входной строки и
 ключа в файл
        {
            using (StreamWriter sw = new StreamWriter(fileName))
            {
                foreach (string str in outw)
                    sw.WriteLine(str);
            }
        }
        public static int[] zero_invert(int[] X) // функция замены нулей на -1
        {
            int[] bin_seq = new int[X.Length];

            for (int i = 0; i < bin_seq.Length; i++)
            {
                bin_seq[i] = X[i];
                if (bin_seq[i] == 0)
                    bin_seq[i] = -1;
            }

            return bin_seq;
        }
```

Test.cs

```
public class Test

    {
        const double F_S = 1.82138636; // значение статистики для частотного теста
        double[] Stat = new double[16] { 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 };

        // ------------------------------------частотный тест----------------------------------------
 --
        //Входные данные: последовательность из -1 и 1
        //Выходные данные: шаги выполнения - статистика, сумма элементов, результат прохождения
 теста.
```

```
        public double[] freq_test(int[] Seq)
        {
            int S = 0; // сумма всех элементов последовательности
            double[] res = new double[3]; // результирующий массив (нужет для регистрации всех шагов
теста)

            for (int i = 0; i < Seq.Length; i++) // получаем сумму элементов последовательности
                S += Seq[i];

            double Stat = Math.Abs(S) / Math.Sqrt(Seq.Length); // находим статистику
            res[0] = S; // сохраняем сумму
            res[1] = Stat; // сохраняем статистику
            if (Stat <= F_S) // если статистика меньше или равна тестовой - тест пройден
                res[2] = 1;
            else
                res[2] = 0;

            return res;
        }
        //-------------------------------------------------------------------------------------
---


        //--------------------------------тест на последовательность одинаковых бит-----------------
----
        //Входные данные: последовательность из 0 и 1
        //Выходные данные: частота встреч единиц в последовательности, значение Vn (все ситуации,
когда соседние элементы не равны друг другу), результат теста

        //Тест определяет, является ли количество цепочек из нулей
        //и единиц различной длины в последовательности приблизительно таким же, как должно быть в
истинно случайной последовательности
        public double[] same_bits_test(int[] Seq)
        {
            double pi = 0; // частота появленяи единиц в последовательности
            int V = 1; // количество ситуаций, при которых соседние числа последовательности не
равны друг другу

            double[] res = new double[4]; // результирующий массив (нужет для регистрации всех шагов
теста)

            // находим частоту встречи единиц в последовательности
            for (int i = 0; i < Seq.Length; i++)
                pi += Seq[i];
            pi /= Seq.Length;

            // находим все ситуации, когда соседние элементы не равны друг другу
            for (int i = 0; i < Seq.Length - 1; i++)
                if (Seq[i] != Seq[i + 1])
                    V += 1;

            double Stat = Math.Abs(V - 2 * pi * Seq.Length * (1 - pi)) /
                          (2 * pi * (1 - pi) * Math.Sqrt(2 * Seq.Length)); // статистика
            res[0] = pi;
            res[1] = V;
            res[2] = Stat;

            if (Stat <= F_S)
                res[3] = 1;
            else
                res[3] = 0;

            return res;
        }
        //-------------------------------------------------------------------------------------
-----
```

```
      //-------------------------расширенный тест на произвольные отклонения--------------------
------
      //-подсчёт количества встреч чисел от -9 до 9 (кроме 0) в последовательности
      public int[] state_check(int[] S, int[] eps)
      {
          for (int i = -9; i < 0; i++)
              for (int j = 0; j < S.Length; j++)
                  if (S[j] == i)
                      eps[i + 9]++;

          for (int i = 1; i < 10; i++)
              for (int j = 0; j < S.Length; j++)
                  if (S[j] == i)
                      eps[i + 8]++;
          return eps;
      }
      // получение статистик для ситуаций, полученных в state_check
      public bool Y_check(double[] Y)
      {
          for (int i = 0; i < Y.Length; i++)
              if (Y[i] > F_S)
                  return false;
          return true;
      }

      //Входные данные: последовательность из -1 и 1
      //Выходные данные: возрастающие суммы, число встреч, статистики, число нулей, результат
прохождения етста

      //Этот тест оценивает общее число посещений определенного состояния при произвольном обходе
кумулятивной суммы. Цель этого теста
      //– определить отклонения от ожидаемого числа посещений различных
      //состояний при произвольном обходе.Фактически данный тест состоит
      //из 18 тестов, по одному для каждого состояния: –9, –8, …, –1, 1, 2, …,9.
      public double[] rand_dev_test(int[] Seq)
      {
          int[] S = new int[Seq.Length + 2]; // новая последовательность возростающих сумм
(содержит 0 в начальном и последнем элементах)
          int[] eps = new int[18]; // количество встреч чисел от -9 до 9 (кроме 0) в S
          double[] Y = new double[18]; // статистики для eps
          int k = 0; // число нулей в S
          int L = 0; // число нулей - 1 в S

          // получаем возростающие суммы
          for (int i = 1; i < Seq.Length + 1; i++)
              S[i] += S[i - 1] + Seq[i - 1];

          // получаем число нулей
          for (int i = 0; i < S.Length; i++)
              if (S[i] == 0)
                  k++;

          L = k - 1;

          eps = state_check(S, eps); // получаем число встреч чисел

          // получаем статистики для чисел от -9 до -1
          for (int i = -9; i < 0; i++)
              Y[i + 9] = Math.Abs(eps[i + 9] - L) / Math.Sqrt(2 * L * (4 * Math.Abs(i) - 2));

          // получаем статистики для чисел от 1 до 9
          for (int i = 1; i < 10; i++)
              Y[i + 8] = Math.Abs(eps[i + 8] - L) / Math.Sqrt(2 * L * (4 * Math.Abs(i) - 2));

          // результирующий массив (нужет для регистрации всех шагов теста)
          double[] res = new double[S.Length + 2 * eps.Length + 2];

          // сохраняем возростающие суммы
          for (int i = 0; i < S.Length; i++)
```

```
            res[i] = S[i];
        // сохраняем число встреч
        for (int i = 0; i < eps.Length; i++)
            res[i + S.Length] = eps[i];
        // сохраняем статистики
        for (int i = 0; i < eps.Length; i++)
            res[i + S.Length + eps.Length] = Y[i];

        res[S.Length + 2 * eps.Length] = L; //число нулей в новой последовательности

        //проверяем все 18 статистик
        //если каждая из статистик менше определенного числа, то тест пройден
        if (Y_check(Y))
            res[S.Length + 2 * eps.Length + 1] = 1;
        else
            res[S.Length + 2 * eps.Length + 1] = 0;

        return res;
    }
```

ANSI_X9_17.cs

```
    public byte[] TriDES(byte[] Data, byte[] Key, byte[] IV)
    {
        MemoryStream mStream = new MemoryStream();  // создаём поток памяти

        TripleDESCryptoServiceProvider tdes = new TripleDESCryptoServiceProvider(); // объект
класса 3DES
        tdes.Padding = PaddingMode.None; //отключаем дополнение выходного шифротекства справа
        tdes.Mode = CipherMode.CFB; // включаепм режим CFB, чтобы установить длину шифротекста в
64 бита

        CryptoStream cStream = new CryptoStream(mStream,
            tdes.CreateEncryptor(Key, IV),
            CryptoStreamMode.Write); // создаём поток для шифрования

        byte[] toEncrypt = Data;

        cStream.Write(toEncrypt, 0, toEncrypt.Length); // помещаем данные в поток шифрования
        cStream.FlushFinalBlock();

        byte[] ret = mStream.ToArray(); // извлекаем шифротекст

        cStream.Close(); // закрываем потоки
        mStream.Close();

        return ret;
    }

    public byte[] key_gen() // генератор 128-битного ключа
    {
        byte[] key = new byte[128]; // основной ключ
        byte[] b1 = new byte[64]; // 64-битная часть ключа
        byte[] b2 = new byte[64]; // 64-битная часть ключа
        byte[] xor = new byte[64]; // 64-битная блок случайных бит для равномерного
преобразования ключа
        byte[] bit = new byte[1]; // случайный бит для выбора совмещения двух частей ключа (1 -
!b1 + b2, 0 - b2 + !b1)
        var rand = new Random(); // структура для рандома

        rand.NextBytes(b1); // создаём случайную половину ключа
        rand.NextBytes(b2); // создаём случайную половину ключа
        rand.NextBytes(xor); // создаём случайную последовательность для операции xor с
половинами ключа
        ulong k1 = BitConverter.ToUInt64(b1, 0); // переводим в ulong формат
        ulong k2 = BitConverter.ToUInt64(b2, 0); // переводим в ulong формат

        ulong xr = BitConverter.ToUInt64(xor, 0); // переводим в ulong формат
        k1 ^= xr; // приводим к равномерному виду
```

```
            xr = BitConverter.ToUInt64(xor, 0);
            k2 ^= xr; // приводим к равномерному виду

            rand.NextBytes(bit); // создаём случайное число (0 или 1), чтобы выбрать, какую половину
ключа поставить первой и какую инвертировать
            if (bit[0] == 1)
            {
                b1 = BitConverter.GetBytes(~k1); // обрачаем число
                b2 = BitConverter.GetBytes(k2);
                key = b1.Concat(b2).ToArray(); // соединяем половины 128-битного ключа
            }
            else
            {
                b1 = BitConverter.GetBytes(k1);
                b2 = BitConverter.GetBytes(~k2); // обрачаем число
                key = b2.Concat(b1).ToArray(); // соединяем половины 128-битного ключа
            }

            return key;
        }

        public ulong[] ANSI(int m) // алгоритм ANSI X9.17
        {
            ulong[] X = new ulong[m]; // результирующая последовательность из m 64-битных чисел

            DateTime date1 = DateTime.Now; // структура для извлечения даты и времени
            var rand = new Random(); // структура для получения рандомных чисел

            byte[] s_0 = new byte[64]; // начальное случайное секретное 64-битное число
            rand.NextBytes(s_0); //     Заполняет элементы указанного массива байтов случайными
числами.

            for (int i = 0; i < m; i++) // основной цикл
            {
                //---------------------Для алгоритма 3DES----------------------------------------
---------------------
                byte[] d = BitConverter.GetBytes((ulong)date1.ToBinary()); // получаем дату и время
в 64-битном формате
                byte[] IV = new byte[128]; // 128-битный вектор инициализации для шифрования чисел
3DES
                rand.NextBytes(IV); // получаем случайный вектор (использование случайного вектора
позволяет избежать дублирующихся блоков в шифротексте)
                byte[] buf = TriDES(d, key_gen(), IV); // key_gen - генерирует составной 128 битный
ключ
                //---------------------------------------------------------------------------------
---------------------

                ulong b1 = BitConverter.ToUInt64(buf, 0); // преобразуем зашифрованную дату и время
в ulong для битовых операций
                ulong s = BitConverter.ToUInt64(s_0, 0); //  преобразуем секретное число в ulong для
битовых операций

                //---------------------вычисляем новое значение параметра Si-----------------------
---------------------
                s ^= b1; // проводим xor операцию с секретным числом и зашифрованными датой,
временем
                //---------------------------------------------------------------------------------
---------------------

                buf = BitConverter.GetBytes(s); // переводим результат xor обратно в массив байтов
для повторного шифрования

                rand.NextBytes(IV); // получаем новый случайный вектор инициализации
                buf = TriDES(buf, key_gen(), IV); // шифруем результат xor

                //----------------------Вычисляем значение i-го выходного слова--------------------
----------------------
                X[i] = BitConverter.ToUInt64(buf, 0); // получаем одно из результирующих
псевдослучайных чисел
```

```
                b1 ^= X[i]; // проводим операцию xor с результирующим числом и предыдущим xor
                //-------------------------------------------------------------------------------
-----------------------

                buf = BitConverter.GetBytes(b1); // переводим xor в массив байтов
                rand.NextBytes(IV); // шифруем результат второго xor чтобы получить новое начальное
случайное число для следующей итерации цикла
            }
            return X;
        }
```

9. Вывод:

В ходе выполненной лабораторной было разработано программное средство, предназначенное для генерации псевдослучайной последовательности алгоритмом ANSI X9.17, а также для исследования полученной последовательности на случайность и равномерность.

В ходе исследования последовательности длиной более 12000 бит, все тесты были успешно пройдены, исходя из чего можно сделать вывод, что последовательность можно считать достаточно случайной для использования в качестве криптографически безопасной псевдослучайной последовательности.