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-61
СТУДЕНТЫ:	Ершов П.К., Мамонова Е.В., Цыденов З.Б.
ВАРИАНТ:	2
ПРЕПОДАВАТЕЛЬ:	Авдеенко Т.В.

Новосибирск

2020

1. Цель работы

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

2. Задание

I. 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.

II. С помощью реализованного приложения выполнить следующие задания.

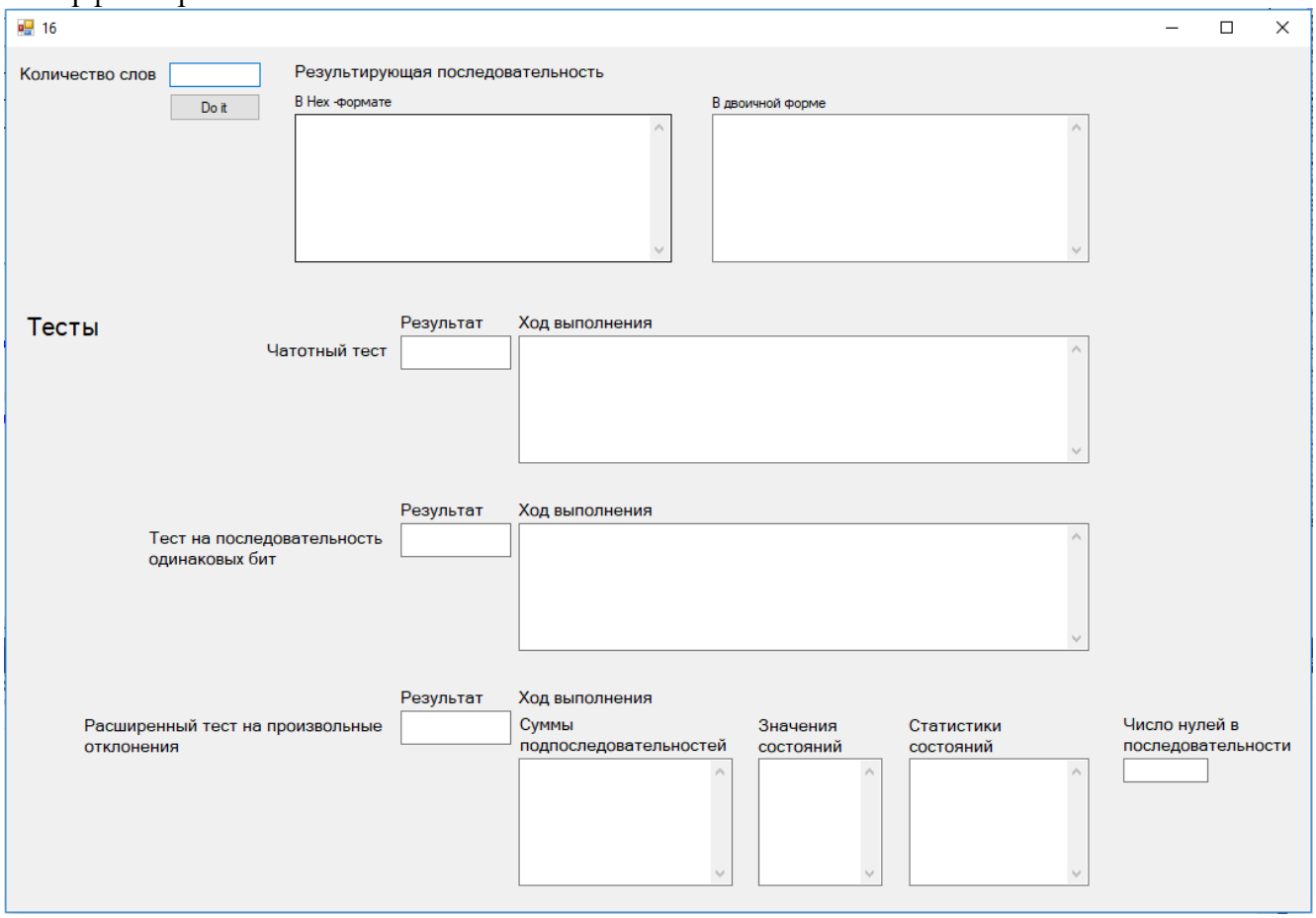
1. Протестировать правильность работы разработанного приложения.
2. Сгенерировать последовательность из не менее 10 000 бит и исследовать ее на равномерность и случайность.
3. Сделать вывод о случайности сгенерированной последовательности и о возможности ее использования в качестве криптографически безопасной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

Вариант: Алгоритм ANSI X9.17

3. 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.

Интерфейс приложения:



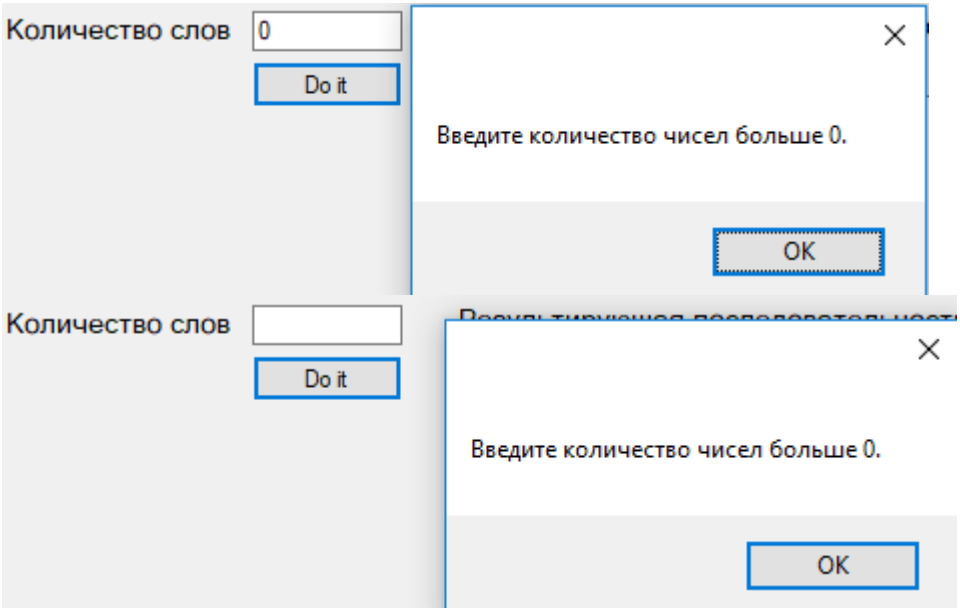
4. Исследования

4.1. Демонстрация работоспособности на примере хэширования нескольких файлов.

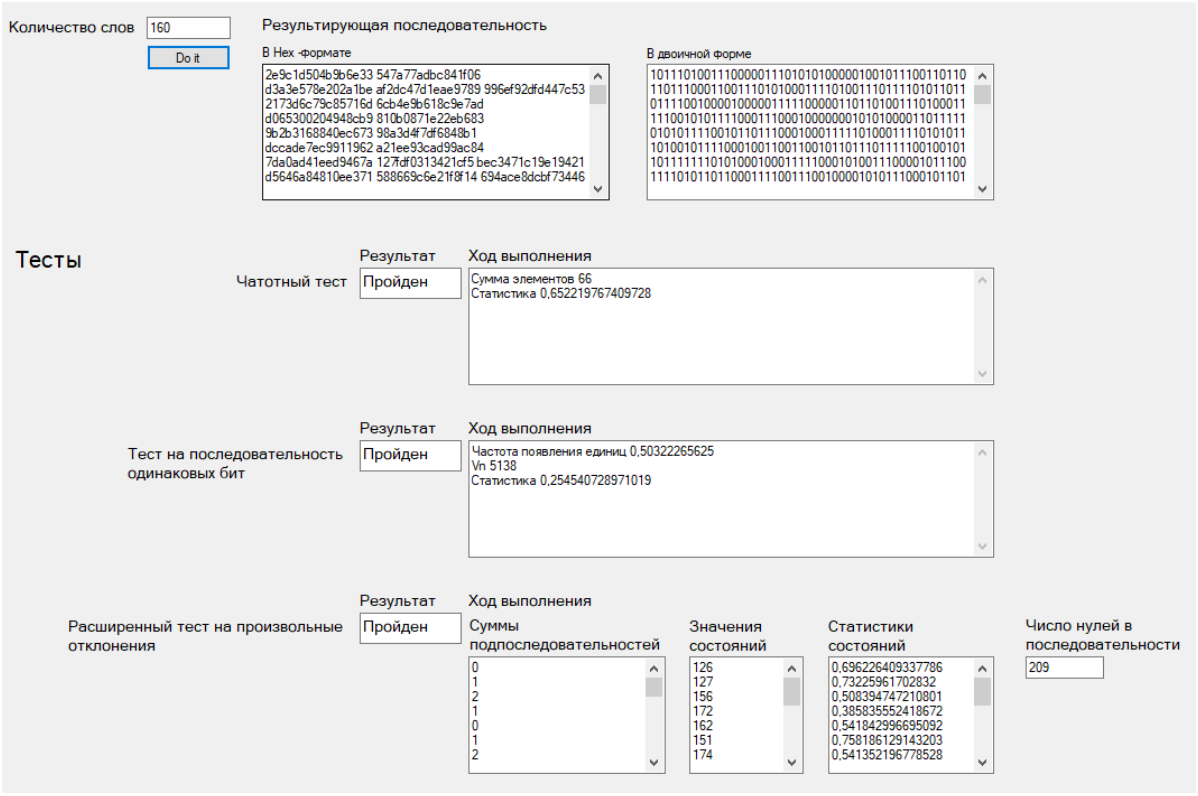
№ теста	Длина сообщения	Результаты		Частотный тест	Тест на последовательность одинаковых бит	Расширенный тест на отклонения
		16-ричная форма	2-чная форма			
1	2	bccc5ed626046afe 9cceee895d70c197	1011110011001 1000101111011 0101100010011 0000001000110 1010111111101 0011100110011 1011101110100 0100101011101 0111000011000 00110010111	Пройден	Пройден	Пройден
2	10	444987a9e5de38d9 b2c6abafbee7c88c a4d74487093661c6 1e5e853f93c3bbfc d033a88c925bbea7 53ef4ff0562bcede 799424a0981fb9cb b4db18496f08ddff d931f2a9605dd948 c46d75c37220f71b	100010001001001 100001111010100 111100101110111 100011100011011 001101100101100 011010101011101 011111011111011 100111110010001 000110010100100 110101110100010 010000111000010 010011011001100 001110001101111 001011110100001 010011111110010 011110000111011 101111111100110	Пройден	Пройден	Пройден

			100000011001110 101000100011001 001001001011011 101111101010011 110100111110111 101001111111100 000101011000101 011110011101101 111011110011001 010000100100101 000001001100000 011111101110011 100101110110100 110110110001100 001001001011011 110000100011011 101111111111101 100100110001111 100101010100101 100000010111011 101100101001000 110001000110110 101110101110000 110111001000100 000111101110001 1011			
3	50			Пройден	Пройден	Пройден

Примеры ошибочной длины последовательности:



4.2. Исследование последовательности длиной более 10000 бит.



5. Код программы

Cripto.cs

```
class Cripto
    {
        const double F_S = 1.82138636; // значение статистики для частотного теста

        public 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 byte[] TriDES(byte[] Data, byte[] Key, byte[] IV) // 3DES алгоритм
        {
            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 оперции 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); //

            for (int i = 0; i < m; i++) // основной цикл
            {
                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); // зашифрованные дата и время
                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
                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
                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;
        }

        public 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;
        }

        public 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 double[] freq_test(int[] Seq) // частотный тест
        {
            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;
    }

    public double[] same_bits_test(int[] Seq) // тест на последовательность одинаковых бит
    {
        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)

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

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

    public int[] state_check(int[] S, int[] eps) // подсчёт количества встреч чисел от -9 до 9 (кроме 0) в последовательности
    {
        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;
    }

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

    public double[] rand_dev_test(int[] Seq) // расширенный тест на произвольные отклонения
    {
        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

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

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

        L = k - 1;

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

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

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

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

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

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

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

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

        if (Y_check(Y))
            res[S.Length + 2 * eps.Length + 1] = 1;
        else
            res[S.Length + 2 * eps.Length + 1] = 0;

        return res;
    }
}
```

Form1.cs

```
private void button1_Click(object sender, EventArgs e)
        {
            string mess = textBox1.Text; // получаем количество 64-битных чисел
            if (mess == "" || Convert.ToInt32(mess) == 0) // проверка на корректное количество чисел
                MessageBox.Show("Введите количество чисел больше 0.");
            else
            {
                Cripto Cr = new Cripto(); // создаём основной класс
                // очищаем все textBox при нажании на кнопку
                textBox2.Clear();
                textBox3.Clear();
                textBox4.Clear();
                textBox5.Clear();
                textBox6.Clear();
                textBox7.Clear();
                textBox8.Clear();
                textBox9.Clear();
                textBox10.Clear();
                textBox11.Clear();
                textBox12.Clear();

                ulong[] ansi = Cr.ANSI(Convert.ToInt32(mess)); // массив псевдослучайных чисел
                int[] Seq = Cr.num_convert(ansi); // подготовка последовательности преобразуем все числа
                                                  //в массив int значений(1, 0)
                string[] out_seq = new string[1]; // выходной массив для записи строки двоичных чисел в файл

                for (int i = 0; i < Seq.Length; i++) // получение строки двоичных чисел
                    out_seq[0] += Seq[i].ToString();
                string binary = "";
                for (int i = 0; i < ansi.Length; i++) // вывод чисел в hex и в двоичном формате
                {
                    textBox2.Text += ansi[i].ToString("x") + " ";
                    binary += Convert.ToString((long)ansi[i], toBase: 2);

                }
                textBox3.Text = binary;

                Cr.Write("out.txt", out_seq); // запись в файл выходной двоичной последовательности
                int[] Seq_m = Cr.zero_invert(Seq); // получение массива значений -1 и 1

                double[] Freq_res = Cr.freq_test(Seq_m); // частотный тест
                // вывод шагов теста
                textBox4.Text += "Сумма элементов " + Freq_res[0].ToString() + Environment.NewLine;
                textBox4.Text += "Статистика " + Freq_res[1].ToString() + Environment.NewLine;
                // если частотный тест не пройден, то нет смысла в других тестах
                if (Freq_res[2] == 1)
                {
                    textBox7.Text = "Пройден ";

                    double[] same_res = Cr.same_bits_test(Seq); // тест на последовательность одинаковых бит
                    // вывод шагов теста
                    textBox5.Text += "Частота появления единиц " + same_res[0].ToString() + Environment.NewLine;
                    textBox5.Text += "Vn " + same_res[1].ToString() + Environment.NewLine;
                    textBox5.Text += "Статистика " + same_res[2].ToString() + Environment.NewLine;
                    // проверка прохождения теста
                    if (same_res[3] == 1)
                        textBox8.Text = "Пройден";
                    else
                        textBox8.Text = "Не пройден";

                    double[] rand_res = Cr.rand_dev_test(Seq_m); // расширенный тест на произвольные отклонения

                    // вывод шагов теста
                    string rt = "";

                    for (int i = 0; i < rand_res.Length - 2 - 2 * 18; i++)
                        rt += rand_res[i].ToString() + Environment.NewLine;
                    textBox6.Text = rt;

                    rt = "";
                    for (int i = rand_res.Length - 2 - 2 * 18; i < rand_res.Length - 2 - 18; i++)
                        rt += rand_res[i].ToString() + Environment.NewLine;
                    textBox10.Text = rt;

                    rt = "";
                    for (int i = rand_res.Length - 2 - 18; i < rand_res.Length - 2; i++)
                        rt += rand_res[i].ToString() + Environment.NewLine;
                    textBox11.Text = rt;

                    textBox12.Text = rand_res[rand_res.Length - 2].ToString();
                    // проверка прохождения теста
                    if (rand_res[rand_res.Length - 1] == 1)
                        textBox9.Text = "Пройден";
                    else
                        textBox9.Text = "Не пройден";
                }
                else
                {
                    textBox7.Text = "Не пройден ";
                    MessageBox.Show("Дальнейшие тесты не требуются, последовательность не случайная.");
                }
            }
        }
```

6. Выводы

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.