# 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.

1. **Ход работы**:
2. **Задание**:

Реализовать приложение с графическим интерфейсом, позволяющее выполнять следующие действия.

1. Генерировать псевдослучайную последовательность с помощью заданного в варианте алгоритма:
2. все входные параметры генератора должны задаваться из файла или вводиться в приложении;
3. сгенерированная последовательность, состоящая из 0 и 1, должна сохраняться в файл;
4. Проверять полученную псевдослучайную последовательность на равномерность и случайность с помощью трех рассмотренных тестов:

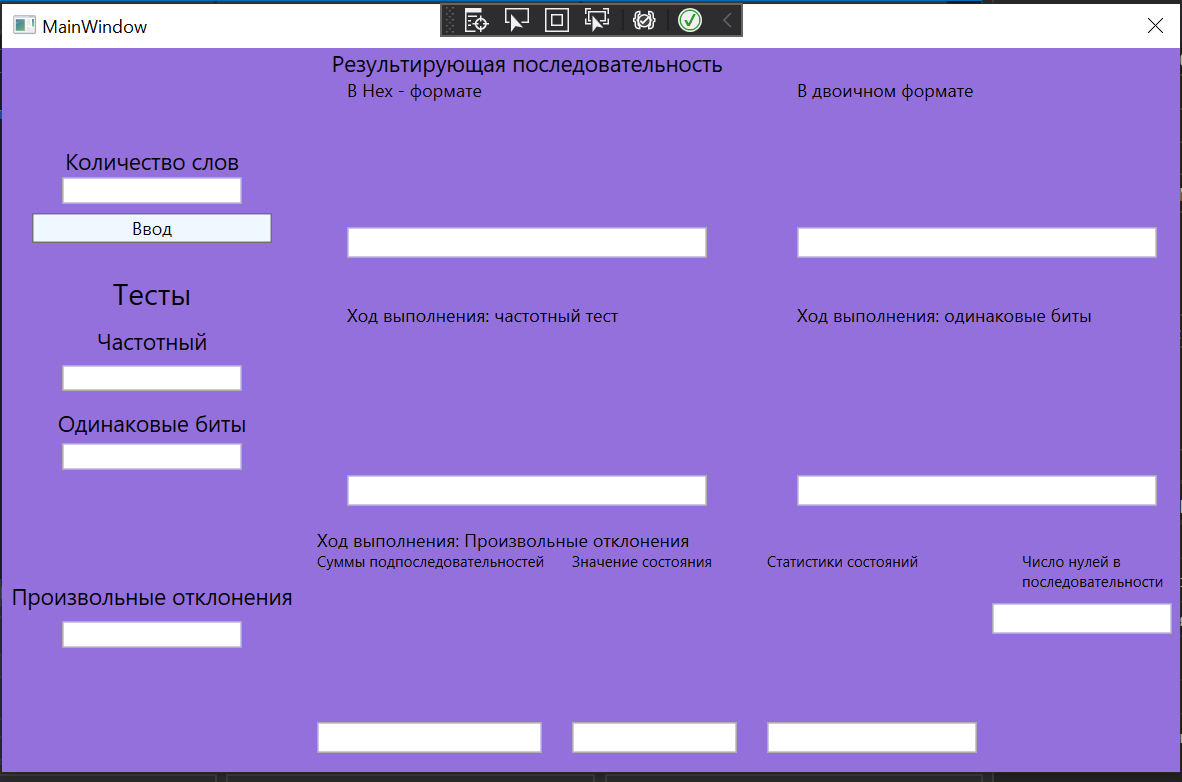
1. результат проверки каждого теста должен отображаться в приложении;

2. все вычисляемые промежуточные значения (все шаги алгоритма теста) могут отображаться в приложении или сохраняться в файл.

С помощью реализованного приложения выполнить следующие задания.

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.
4. **Вариант:** Алгоритм ANSI X9.17
5. **Описание разработанного программного средства**

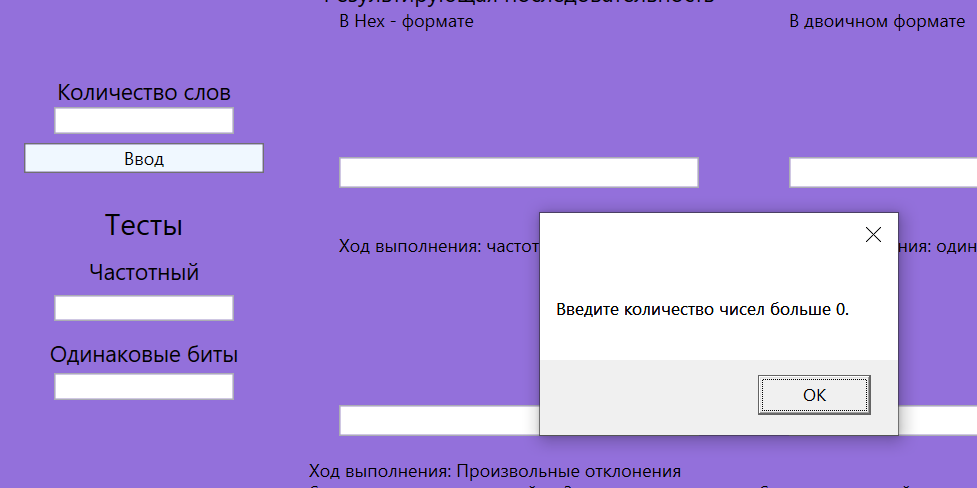
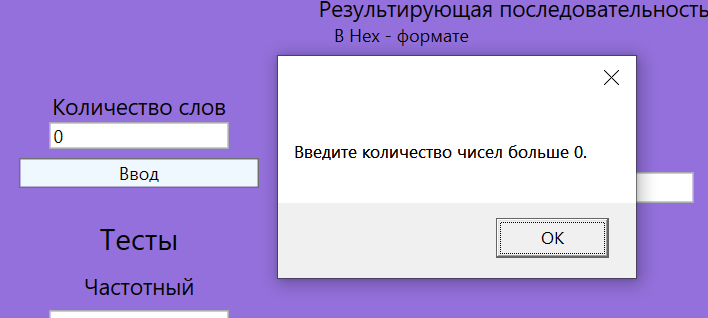
Разработанная программа способна генерировать псевдослучайную последовательность чисел заданной длины. Программа выводит сгенерированную последовательность в 16-ричном (для наглядности отличия 64-битных чисел друг от друга) и в двоичном формате, так же занося двоичный формат в выходной файл. Программа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

1. **Интерфейс приложения**
2. **Исследования**:

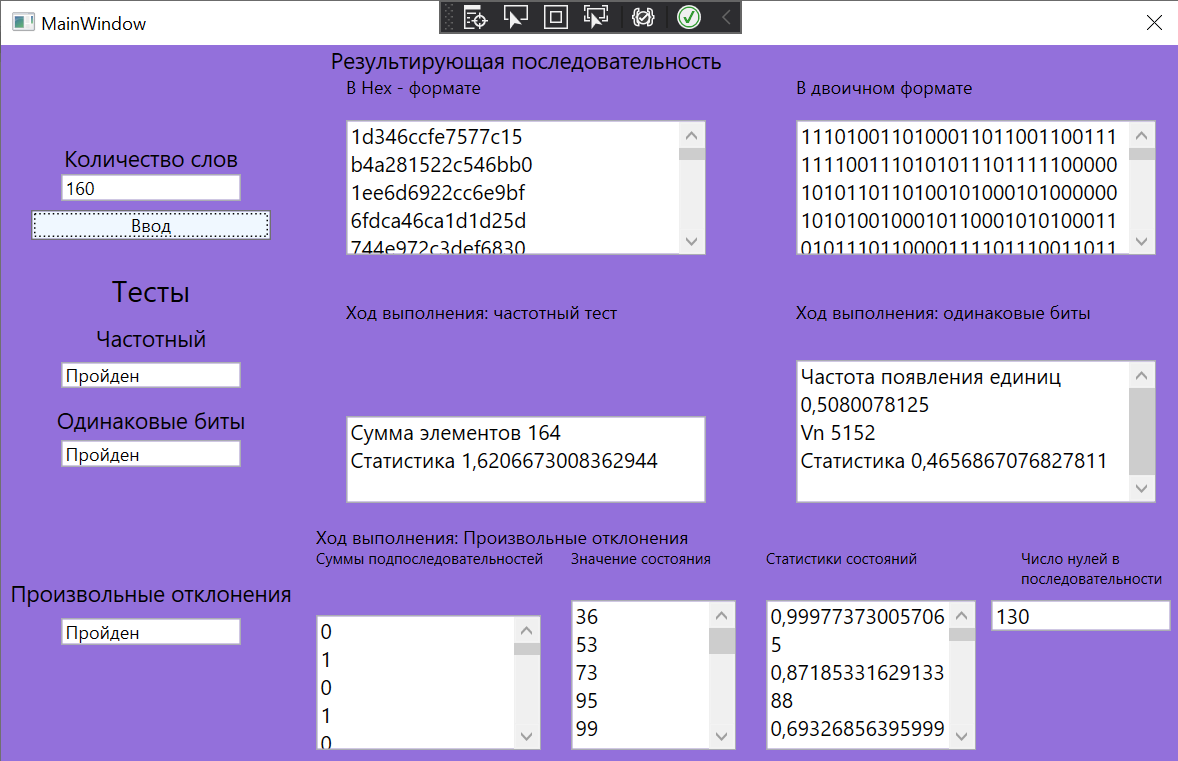
6.1) Демонстрация работоспособности на примере хэширования нескольких файлов.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № теста | Длина сообщения | Результаты | | Частотный тест | Тест на последовательность одинаковых бит | Расширенный тест на отклонения |
| 16-ричная форма | 2-чная форма |
| 1 | 2 | bccc5ed626046afe 9cceee895d70c197 | 10111100110011000101111011010110001001100000010001101010111111101001110011001110111011101000100101011101011100001100000110010111 | Пройден | Пройден | Пройден |
| 2 | 10 | 444987a9e5de38d9 b2c6abafbee7c88c a4d74487093661c6 1e5e853f93c3bbfc d033a88c925bbea7 53ef4ff0562bcede 799424a0981fb9cb b4db18496f08ddff d931f2a9605dd948 c46d75c37220f71b | 1000100010010011000011110101001111001011101111000111000110110011011001011000110101010111010111110111110111001111100100010001100101001001101011101000100100001110000100100110110011000011100011011110010111101000010100111111100100111100001110111011111111001101000000110011101010001000110010010010010110111011111010100111101001111101111010011111111000001010110001010111100111011011110111100110010100001001001010000010011000000111111011100111001011101101001101101100011000010010010110111100001000110111011111111111011001001100011111001010101001011000000101110111011001010010001100010001101101011101011100001101110010001000001111011100011011 | Пройден | Пройден | Пройден |
| 3 | 50 |  |  | Пройден | Пройден | Пройден |

Пример неправильного ввода:

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



1. **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;

1. **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 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); // 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;

}

1. **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.