Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Российский химико-технологический университет имени Д.И. Менделеева»
Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 9

Выполнил студент группы.....КС-38.....(Нергарян Геворг Гарегинович)
Ссылка на репозиторий: .....(https://github.com/MUCTR-IKT-CPP/Nergaryan_KC-38_Algos)

Приняли: .....Пысин Максим Дмитриевич
.....Краснов Дмитрий Олегович
.....Лобанов Алексей Владимирович
.....Крашенинников Роман Сергеевич

Дата сдачи: …….04.05.2023……..

Оглавление

Описание задачи.

В рамках лабораторной работы необходимо реализовать 1 из ниже приведенных алгоритмов хеширования:

1. MD5
2. SHA1
3. SHA2
4. Стриборг
5. RIPEMD-160

Доп вариант для тех кто хочет посложнее:

- Luffa
- SHA3

Для реализованной хеш функции провести следующие тесты:

- Провести сгенерировать 1000 пар строк длинной 128 символов отличающихся друг от друга 1,2,4,8,16 символов и сравнить хеши для пар между собой, проведя поиск одинаковых последовательностей символов в хешах и подсчитав максимальную длину такой последовательности. Результаты для каждого количества отличий нанести на график, где по оси x кол-во отличий, а по оси y максимальная длинна одинаковой последовательности.
- Провести N = 10^i(i от 2 до 6) генерацию хешей для случайно сгенерированных строк длинно 256 символов, и выполнить поиск одинаковых хешей в итоговом наборе данных, результаты привести в таблице где первая колонка это N генераций, а вторая таблица наличие и кол-во одинаковых хешей, если такие были.
- Провести по 1000 генераций хеша для строк длинной n (64, 128, 256, 512, 1024, 2048, 4096, 8192)(строки генерировать случайно для каждой серии), подсчитать среднее время и построить зависимость скорости расчета хеша от размера входных данных

Описание метода/модели.

Метод Стрибога - это хэш-функция, которая преобразует входные данные произвольной длины в выходные данные фиксированной длины, обычно в виде битовой строки.

Метод Стрибог был разработан НИИ КрИптографии и Информационной Безопасности (НИИКИБ) Российской Федерации в 2010 году. Этот метод является одним из наиболее распространенных хэш-алгоритмов, используемых в настоящее время.

Алгоритм Стрибог основан на блочном шифре ГОСТ 28147-89, который является стандартом для защиты информации в Российской Федерации. Хэш-функция Стрибог работает в двух режимах: Стрибог-256 и Стрибог-512. Стрибог-256 генерирует хэш-код длиной 256 бит, а Стрибог-512 - длиной 512 бит.

Алгоритм Стрибог состоит из нескольких этапов:

1. Инициализация: устанавливается начальное значение вектора состояния.
2. Добавление данных: входные данные разбиваются на блоки фиксированного размера и добавляются к вектору состояния.
3. Финализация: к вектору состояния применяется дополнительное преобразование, в результате чего получается итоговое значение хэш-кода.

В коде, метод Стрибог используется для создания хэш-функции, которая принимает строку и возвращает 64-значный хэш в формате шестнадцатеричного числа. Код разбивает строку на блоки, использует алгоритм Стрибог-512 для вычисления хэш-кода каждого блока, а затем объединяет результаты для получения итогового хэш-кода.

Выполнение задачи.

Использовался языкC#.

1) Код:

```
using System.Collections;
using System.Text;
//алгоритм хеширования Стриборг
namespace Lab9
{
   internal class Algosss
   {
      class GOST
      {
         //readonly означает, что значение этого массива можно изменять только в конструкторе объекта класса
         // бинарную матрицу A (L)
         private readonly ulong[] A = {
      0x8e20faa72ba0b470, 0x47107ddd9b505a38, 0xad08b0e0c3282d1c, 0xd8045870ef14980e,
      0x6c022c38f90a4c07, 0x3601161cf205268d, 0x1b8e0b0e798c13c8, 0x83478b07b2468764,
      0xa011d380818e8f40, 0x5086e740ce47c920, 0x2843fd2067adea10, 0x14aff010bdd87508,
      0x0ad97808d06cb404, 0x05e23c0468365a02, 0x8c711e02341b2d01, 0x46b60f011a83988e,
      0x90dab52a387ae76f, 0x486dd4151c3dfdb9, 0x24b86a840e90f0d2, 0x125c354207487869,
      0x092e94218d243cba, 0x8a174a9ec8121e5d, 0x4585254f64090fa0, 0xaccc9ca9328a8950,
      0x9d4df05d5f661451, 0xc0a878a0a1330aa6, 0x60543c50de970553, 0x302a1e286fc58ca7,
      0x18150f14b9ec46dd, 0x0c84890ad27623e0, 0x0642ca05693b9f70, 0x0321658cba93c138,
      0x86275df09ce8aaa8, 0x439da0784e745554, 0xafc0503c273aa42a, 0xd960281e9d1d5215,
      0xe230140fc0802984, 0x71180a8960409a42, 0xb60c05ca30204d21, 0x5b068c651810a89e,
      0x456c34887a3805b9, 0xac361a443d1c8cd2, 0x561b0d22900e4669, 0x2b838811480723ba,
      0x9bcf4486248d9f5d, 0xc3e9224312c8c1a0, 0xeffa11af0964ee50, 0xf97d86d98a327728,
      0xe4fa2054a80b329c, 0x727d102a548b194e, 0x39b008152acb8227, 0x9258048415eb419d,
      0x492c024284fbaec0, 0xaa16012142f35760, 0x550b8e9e21f7a530, 0xa48b474f9ef5dc18,
      0x70a6a56e2440598e, 0x3853dc371220a247, 0x1ca76e95091051ad, 0x0edd37c48a08a6d8,
      0x07e095624504536c, 0x8d70c431ac02a736, 0xc83862965601dd1b, 0x641c314b2b8ee083
      };

         // Таблица подстановок (S)
         private readonly byte[] Sbox ={
      0xFC, 0xEE, 0xDD, 0x11, 0xCF, 0x6E, 0x31, 0x16, 0xFB, 0xC4, 0xFA, 0xDA, 0x23, 0xC5, 0x04, 0x4D,
      0xE9, 0x77, 0xF0, 0xDB, 0x93, 0x2E, 0x99, 0xBA, 0x17, 0x36, 0xF1, 0xBB, 0x14, 0xCD, 0x5F, 0xC1,
      0xF9, 0x18, 0x65, 0x5A, 0xE2, 0x5C, 0xEF, 0x21, 0x81, 0x1C, 0x3C, 0x42, 0x8B, 0x01, 0x8E, 0x4F,
      0x05, 0x84, 0x02, 0xAE, 0xE3, 0x6A, 0x8F, 0xA0, 0x06, 0x0B, 0xED, 0x98, 0x7F, 0xD4, 0xD3, 0x1F,
      0xEB, 0x34, 0x2C, 0x51, 0xEA, 0xC8, 0x48, 0xAB, 0xF2, 0x2A, 0x68, 0xA2, 0xFD, 0x3A, 0xCE, 0xCC,
      0xB5, 0x70, 0x0E, 0x56, 0x08, 0x0C, 0x76, 0x12, 0xBF, 0x72, 0x13, 0x47, 0x9C, 0xB7, 0x5D, 0x87,
      0x15, 0xA1, 0x96, 0x29, 0x10, 0x7B, 0x9A, 0xC7, 0xF3, 0x91, 0x78, 0x6F, 0x9D, 0x9E, 0xB2, 0xB1,
      0x32, 0x75, 0x19, 0x3D, 0xFF, 0x35, 0x8A, 0x7E, 0x6D, 0x54, 0xC6, 0x80, 0xC3, 0xBD, 0x0D, 0x57,
```

```
0xDF, 0xF5, 0x24, 0xA9, 0x3E, 0xA8, 0x43, 0xC9, 0xD7, 0x79, 0xD6, 0xF6, 0x7C, 0x22,
0xB9, 0x03,
0xE0, 0x0F, 0xEC, 0xDE, 0x7A, 0x94, 0xB0, 0xBC, 0xDC, 0xE8, 0x28, 0x50, 0x4E, 0x33,
0x0A, 0x4A,
0xA7, 0x97, 0x60, 0x73, 0x1E, 0x00, 0x62, 0x44, 0x1A, 0xB8, 0x38, 0x82, 0x64, 0x9F, 0x26,
0x41,
0xAD, 0x45, 0x46, 0x92, 0x27, 0x5E, 0x55, 0x2F, 0x8C, 0xA3, 0xA5, 0x7D, 0x69, 0xD5,
0x95, 0x3B,
0x07, 0x58, 0xB3, 0x40, 0x86, 0xAC, 0x1D, 0xF7, 0x30, 0x37, 0x6B, 0xE4, 0x88, 0xD9,
0xE7, 0x89,
0xE1, 0x1B, 0x83, 0x49, 0x4C, 0x3F, 0xF8, 0xFE, 0x8D, 0x53, 0xAA, 0x90, 0xCA, 0xD8,
0x85, 0x61,
0x20, 0x71, 0x67, 0xA4, 0x2D, 0x2B, 0x09, 0x5B, 0xCB, 0x9B, 0x25, 0xD0, 0xBE, 0xE5,
0x6C, 0x52,
0x59, 0xA6, 0x74, 0xD2, 0xE6, 0xF4, 0xB4, 0xC0, 0xD1, 0x66, 0xAF, 0xC2, 0x39, 0x4B,
0x63, 0xB6
};

    // Таблица перестановок (P)
    private readonly byte[] Tau ={
0, 8, 16, 24, 32, 40, 48, 56,
1, 9, 17, 25, 33, 41, 49, 57,
2, 10, 18, 26, 34, 42, 50, 58,
3, 11, 19, 27, 35, 43, 51, 59,
4, 12, 20, 28, 36, 44, 52, 60,
5, 13, 21, 29, 37, 45, 53, 61,
6, 14, 22, 30, 38, 46, 54, 62,
7, 15, 23, 31, 39, 47, 55, 63
};

    // Константные значения для функции KeySchedule
    private readonly byte[][] C = {
new byte[64]{
0xb1,0x08,0x5b,0xda,0x1e,0xca,0xda,0xe9,0xeb,0xcb,0x2f,0x81,0xc0,0x65,0x7c,0x1f,
0x2f,0x6a,0x76,0x43,0x2e,0x45,0xd0,0x16,0x71,0x4e,0xb8,0x8d,0x75,0x85,0xc4,0xfc,
0x4b,0x7c,0xe0,0x91,0x92,0x67,0x69,0x01,0xa2,0x42,0x2a,0x08,0xa4,0x60,0xd3,0x15,
0x05,0x76,0x74,0x36,0xcc,0x74,0x4d,0x23,0xdd,0x80,0x65,0x59,0xf2,0xa6,0x45,0x07
},
new byte[64]{
0x6f,0xa3,0xb5,0x8a,0xa9,0x9d,0x2f,0x1a,0x4f,0xe3,0x9d,0x46,0x0f,0x70,0xb5,0xd7,
0xf3,0xfe,0xea,0x72,0x0a,0x23,0x2b,0x98,0x61,0xd5,0x5e,0x0f,0x16,0xb5,0x01,0x31,
0x9a,0xb5,0x17,0x6b,0x12,0xd6,0x99,0x58,0x5c,0xb5,0x61,0xc2,0xdb,0x0a,0xa7,0xca,
0x55,0xdd,0xa2,0x1b,0xd7,0xcb,0xcd,0x56,0xe6,0x79,0x04,0x70,0x21,0xb1,0x9b,0xb7
},
new byte[64]{
0xf5,0x74,0xdc,0xac,0x2b,0xce,0x2f,0xc7,0x0a,0x39,0xfc,0x28,0x6a,0x3d,0x84,0x35,
0x06,0xf1,0x5e,0x5f,0x52,0x9c,0x1f,0x8b,0xf2,0xea,0x75,0x14,0xb1,0x29,0x7b,0x7b,
0xd3,0xe2,0x0f,0xe4,0x90,0x35,0x9e,0xb1,0xc1,0xc9,0x3a,0x37,0x60,0x62,0xdb,0x09,
0xc2,0xb6,0xf4,0x43,0x86,0x7a,0xdb,0x31,0x99,0x1e,0x96,0xf5,0x0a,0xba,0x0a,0xb2
},
new byte[64]{
0xef,0x1f,0xdf,0xb3,0xe8,0x15,0x66,0xd2,0xf9,0x48,0xe1,0xa0,0x5d,0x71,0xe4,0xdd,
0x48,0x8e,0x85,0x7e,0x33,0x5c,0x3c,0x7d,0x9d,0x72,0x1c,0xad,0x68,0x5e,0x35,0x3f,
0xa9,0xd7,0x2c,0x82,0xed,0x03,0xd6,0x75,0xd8,0xb7,0x13,0x33,0x93,0x52,0x03,0xbe,
0x34,0x53,0xea,0xa1,0x93,0xe8,0x37,0xf1,0x22,0x0c,0xbe,0xbc,0x84,0xe3,0xd1,0x2e
```

```
},
new byte[64]{
0x4b,0xea,0x6b,0xac,0xad,0x47,0x47,0x99,0x9a,0x3f,0x41,0x0c,0x6c,0xa9,0x23,0x63,
0x7f,0x15,0x1c,0x1f,0x16,0x86,0x10,0x4a,0x35,0x9e,0x35,0xd7,0x80,0x0f,0xff,0xbd,
0xbf,0xcd,0x17,0x47,0x25,0x3a,0xf5,0xa3,0xdf,0xff,0x00,0xb7,0x23,0x27,0x1a,0x16,
0x7a,0x56,0xa2,0x7e,0xa9,0xea,0x63,0xf5,0x60,0x17,0x58,0xfd,0x7c,0x6c,0xfe,0x57
},
new byte[64]{
0xae,0x4f,0xae,0xae,0x1d,0x3a,0xd3,0xd9,0x6f,0xa4,0xc3,0x3b,0x7a,0x30,0x39,0xc0,
0x2d,0x66,0xc4,0xf9,0x51,0x42,0xa4,0x6c,0x18,0x7f,0x9a,0xb4,0x9a,0xf0,0x8e,0xc6,
0xcf,0xfa,0xa6,0xb7,0x1c,0x9a,0xb7,0xb4,0x0a,0xf2,0x1f,0x66,0xc2,0xbe,0xc6,0xb6,
0xbf,0x71,0xc5,0x72,0x36,0x90,0x4f,0x35,0xfa,0x68,0x40,0x7a,0x46,0x64,0x7d,0x6e
},
new byte[64]{
0xf4,0xc7,0x0e,0x16,0xee,0xaa,0xc5,0xec,0x51,0xac,0x86,0xfe,0xbf,0x24,0x09,0x54,
0x39,0x9e,0xc6,0xc7,0xe6,0xbf,0x87,0xc9,0xd3,0x47,0x3e,0x33,0x19,0x7a,0x93,0xc9,
0x09,0x92,0xab,0xc5,0x2d,0x82,0x2c,0x37,0x06,0x47,0x69,0x83,0x28,0x4a,0x05,0x04,
0x35,0x17,0x45,0x4c,0xa2,0x3c,0x4a,0xf3,0x88,0x86,0x56,0x4d,0x3a,0x14,0xd4,0x93
},
new byte[64]{
0x9b,0x1f,0x5b,0x42,0x4d,0x93,0xc9,0xa7,0x03,0xe7,0xaa,0x02,0x0c,0x6e,0x41,0x41,
0x4e,0xb7,0xf8,0x71,0x9c,0x36,0xde,0x1e,0x89,0xb4,0x44,0x3b,0x4d,0xdb,0xc4,0x9a,
0xf4,0x89,0x2b,0xcb,0x92,0x9b,0x06,0x90,0x69,0xd1,0x8d,0x2b,0xd1,0xa5,0xc4,0x2f,
0x36,0xac,0xc2,0x35,0x59,0x51,0xa8,0xd9,0xa4,0x7f,0x0d,0xd4,0xbf,0x02,0xe7,0x1e
},
new byte[64]{
0x37,0x8f,0x5a,0x54,0x16,0x31,0x22,0x9b,0x94,0x4c,0x9a,0xd8,0xec,0x16,0x5f,0xde,
0x3a,0x7d,0x3a,0x1b,0x25,0x89,0x42,0x24,0x3c,0xd9,0x55,0xb7,0xe0,0x0d,0x09,0x84,
0x80,0x0a,0x44,0x0b,0xdb,0xb2,0xce,0xb1,0x7b,0x2b,0x8a,0x9a,0xa6,0x07,0x9c,0x54,
0x0e,0x38,0xdc,0x92,0xcb,0x1f,0x2a,0x60,0x72,0x61,0x44,0x51,0x83,0x23,0x5a,0xdb
},
new byte[64]{
0xab,0xbe,0xde,0xa6,0x80,0x05,0x6f,0x52,0x38,0x2a,0xe5,0x48,0xb2,0xe4,0xf3,0xf3,
0x89,0x41,0xe7,0x1c,0xff,0x8a,0x78,0xdb,0x1f,0xff,0xe1,0x8a,0x1b,0x33,0x61,0x03,
0x9f,0xe7,0x67,0x02,0xaf,0x69,0x33,0x4b,0x7a,0x1e,0x6c,0x30,0x3b,0x76,0x52,0xf4,
0x36,0x98,0xfa,0xd1,0x15,0x3b,0xb6,0xc3,0x74,0xb4,0xc7,0xfb,0x98,0x45,0x9c,0xed
},
new byte[64]{
0x7b,0xcd,0x9e,0xd0,0xef,0xc8,0x89,0xfb,0x30,0x02,0xc6,0xcd,0x63,0x5a,0xfe,0x94,
0xd8,0xfa,0x6b,0xbb,0xeb,0xab,0x07,0x61,0x20,0x01,0x80,0x21,0x14,0x84,0x66,0x79,
0x8a,0x1d,0x71,0xef,0xea,0x48,0xb9,0xca,0xef,0xba,0xcd,0x1d,0x7d,0x47,0x6e,0x98,
0xde,0xa2,0x59,0x4a,0xc0,0x6f,0xd8,0x5d,0x6b,0xca,0xa4,0xcd,0x81,0xf3,0x2d,0x1b
},
new byte[64]{
0x37,0x8e,0xe7,0x67,0xf1,0x16,0x31,0xba,0xd2,0x13,0x80,0xb0,0x04,0x49,0xb1,0x7a,
0xcd,0xa4,0x3c,0x32,0xbc,0xdf,0x1d,0x77,0xf8,0x20,0x12,0xd4,0x30,0x21,0x9f,0x9b,
0x5d,0x80,0xef,0x9d,0x18,0x91,0xcc,0x86,0xe7,0x1d,0xa4,0xaa,0x88,0xe1,0x28,0x52,
0xfa,0xf4,0x17,0xd5,0xd9,0xb2,0x1b,0x99,0x48,0xbc,0x92,0x4a,0xf1,0x1b,0xd7,0x20
}
};

    private readonly byte[] iv = new byte[64];

    private byte[] N = new byte[64];
```

```
        private byte[] Sigma = new byte[64];

        public int outLen = 0; //Эта переменная используется для задания длины выходного
хеш-кода

        //Конструктор класса ГОСТ
        // Принимает длину желаемого выходного хеша: 256 бит или 512 бит
        public GOST(int outputLenght)
        {
            if (outputLenght == 512)
            {// Заполнение векторов N, Sigma и iv нулями
                for (int i = 0; i < 64; i++)
                {
                    N[i] = 0x00;
                    Sigma[i] = 0x00;
                    iv[i] = 0x00;
                }
                outLen = 512;// Установка длины выходного хеша в 512 бит
            }
            else if (outputLenght == 256)
            {// Заполнение векторов N и Sigma нулями, а iv - единицами
                for (int i = 0; i < 64; i++)
                {
                    N[i] = 0x00;
                    Sigma[i] = 0x00;
                    iv[i] = 0x01;
                }
                outLen = 256;// Установка длины выходного хеша в 256 бит
            }
        }
        // Этот метод складывает два 64-байтных массива и возвращает результат
        // Результат вычислений сохраняется в новый массив размером 64 байта
        // Если результат сложения больше 64 байт, то берется только младший байт каждого
слова.
        // Это происходит из-за использования оператора & для обнуления старшего байта
каждого слова.
        private static byte[] AddModulo512(byte[] a, byte[] b)
        {
            byte[] temp = new byte[64];
            int t = 0;
            byte[] tempA = new byte[64];
            byte[] tempB = new byte[64];
            Array.Copy(a, 0, tempA, 64 - a.Length, a.Length);
            Array.Copy(b, 0, tempB, 64 - b.Length, b.Length);
            int i;
            for (i = 63; i >= 0; i--)
            {
                t = tempA[i] + tempB[i] + (t >> 8);
                temp[i] = (byte)(t & 0xFF);
            }
            return temp;
        }

        //XOR двух последовательностей длиной 512 бит
        private static byte[] AddXor512(byte[] a, byte[] b)
```

```
{
    byte[] c = new byte[64];
    for (int i = 0; i < 64; i++)
        c[i] = (byte)(a[i] ^ b[i]);
    return c;
}

//Функция S является обычной функцией подстановки.
//Каждый байт из 512-битной входной последовательности заменяется
//соответствующим байтом из таблицы подстановок
private byte[] S(byte[] state)
{
    byte[] result = new byte[64];
    for (int i = 0; i < 64; i++)
        result[i] = Sbox[state[i]];
    return result;
}

//P-преобразование. Функция перестановки.
//Для каждой пары байт из входной последовательности
//происходит замена одного байта другим.
private byte[] P(byte[] state)
{
    byte[] result = new byte[64];
    for (int i = 0; i < 64; i++)
    {
        result[i] = state[Tau[i]];
    }
    return result;
}

//L-преобразование.
//Представляет собой умножение 64-битного входного вектора на бинарную матрицу
A.
// Метод для выполнения преобразования L в алгоритме ГОСТ Р 34.11-2012.
// Входные параметры:
// - state - массив байт, представляющий состояние в алгоритме
// Выходные параметры:
// - result - массив байт, представляющий результат преобразования L
private byte[] L(byte[] state)
{
    byte[] result = new byte[64];
    for (int i = 0; i < 8; i++)
    {
        ulong t = 0;
        byte[] tempArray = new byte[8];
        Array.Copy(state, i * 8, tempArray, 0, 8);
        tempArray = tempArray.Reverse().ToArray();
        BitArray tempBits1 = new(tempArray);
        bool[] tempBits = new bool[64];
        tempBits1.CopyTo(tempBits, 0);
        tempBits = tempBits.Reverse().ToArray();
        for (int j = 0; j < 64; j++)
        {
            if (tempBits[j] != false)
```

```
                t ^= A[j];
            }
            byte[] ResPart = BitConverter.GetBytes(t).Reverse().ToArray();
            Array.Copy(ResPart, 0, result, i * 8, 8);
        }
        return result;
    }

    //функция KeySchedule(K, i).
    //Отвечает за формирование временного ключа K на каждом раунде функции E(K, m).
    private byte[] KeySchedule(byte[] K, int i)
    {
        K = AddXor512(K, C[i]);
        K = S(K);
        K = P(K);
        K = L(K);
        return K;
    }

    //Функция E.
    private byte[] E(byte[] K, byte[] m)
    {
        byte[] state = AddXor512(K, m);
        for (int i = 0; i < 12; i++)
        {
            state = S(state);
            state = P(state);
            state = L(state);
            K = KeySchedule(K, i);
            state = AddXor512(state, K);
        }
        return state;
    }

    //Функция сжатия gn
    private byte[] G_n(byte[] N, byte[] h, byte[] m)
    {
        byte[] K = AddXor512(h, N);
        K = S(K);
        K = P(K);
        K = L(K);
        byte[] t = E(K, m);
        t = AddXor512(t, h);
        byte[] newh = AddXor512(t, m);
        return newh;
    }

    //Функция получения Хэш значения.
    public byte[] GetHash(byte[] message)
    {
        byte[] paddedMes = new byte[64];
        int len = message.Length * 8;
        byte[] h = new byte[64];
        Array.Copy(iv, h, 64);
        byte[] N_0 ={
```

```
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
};
   // Очистка массивов N, Sigma и iv в зависимости от длины хеш-значения
   if (outLen == 512)
   {
      for (int i = 0; i < 64; i++)
      {
         N[i] = 0x00;
         Sigma[i] = 0x00;
         iv[i] = 0x00;
      }
   }
   else if (outLen == 256)
   {
      for (int i = 0; i < 64; i++)
      {
         N[i] = 0x00;
         Sigma[i] = 0x00;
         iv[i] = 0x01;
      }
   }

   // Создание массива N_512 для определения длины обрабатываемого блока
   byte[] N_512 = BitConverter.GetBytes(512);
   int inc = 0;
   while (len >= 512)
   {
      inc++;
      byte[] tempMes = new byte[64];
      Array.Copy(message, message.Length - inc * 64, tempMes, 0, 64);
      // Обновление массива h для обрабатываемого блока
      h = G_n(N, h, tempMes);
      N = AddModulo512(N, N_512.Reverse().ToArray());
      Sigma = AddModulo512(Sigma, tempMes);
      len -= 512;
   }
   byte[] message1 = new byte[message.Length - inc * 64];
   Array.Copy(message, 0, message1, 0, message.Length - inc * 64);
   if (message1.Length < 64)
   {
      for (int i = 0; i < (64 - message1.Length - 1); i++)
      {
         paddedMes[i] = 0;
      }
      paddedMes[64 - message1.Length - 1] = 0x01;
      Array.Copy(message1, 0, paddedMes, 64 - message1.Length, message1.Length);
   }
   h = G_n(N, h, paddedMes);
   byte[] MesLen = BitConverter.GetBytes(message1.Length * 8);
   N = AddModulo512(N, MesLen.Reverse().ToArray());
   Sigma = AddModulo512(Sigma, paddedMes);
   h = G_n(N_0, h, N);
```

```
        h = G_n(N_0, h, Sigma);
        if (outLen == 512)
            return h;
        else
        {
            byte[] h256 = new byte[32];
            Array.Copy(h, 0, h256, 0, 32);
            return h256;
        }
    }
}


//Получение случайной строки символов длинной length.
public static string GetRandomLine(int length, Random random)
{
    string line = "";
    for (int i = 0; i < length; i++)
    {

        int randomNumber = random.Next(0, 9 + 1);
        line += randomNumber;
    }
    return line;
}


//Поиск максимальной длины одинаковой последовательности символов строк str1 и str2.
public static int LCSubStr(string str1, string str2)
{
    int[,] LCSuff = new int[str1.Length + 1, str2.Length + 1];
    int mx = 0;
    for (int i = 0; i <= str1.Length; i++)
    {
        for (int j = 0; j <= str2.Length; j++)
        {
            if (i == 0 || j == 0)
            {
                LCSuff[i, j] = 0;
            }
            else if (str1[i - 1] == str2[j - 1])
            {
                LCSuff[i, j] = LCSuff[i - 1, j - 1] + 1;
                mx = Math.Max(mx, LCSuff[i, j]);
            }
            else
                LCSuff[i, j] = 0;
        }
    }
    return mx;
}

//Тест 1
public static void Test1()
{
```

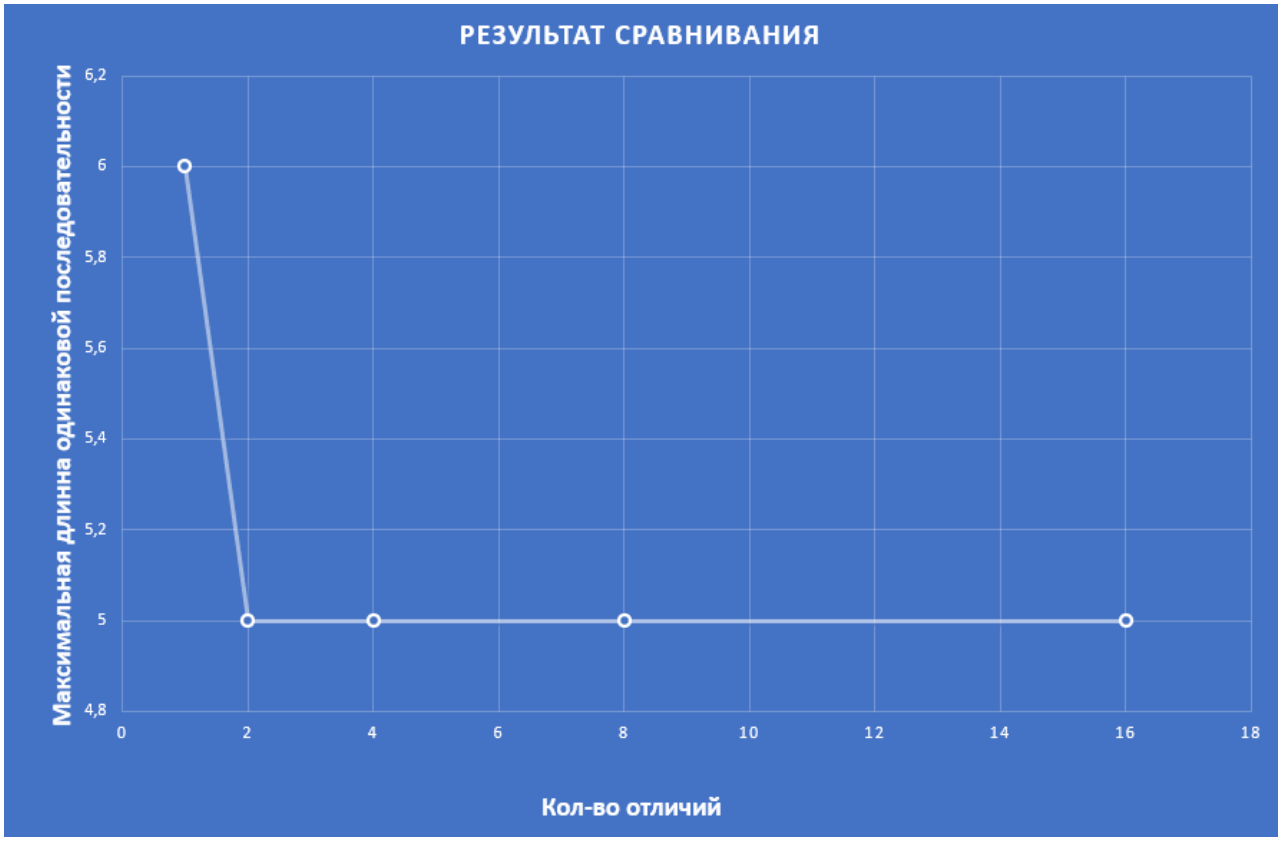
```
    Console.WriteLine("Test №1");
    Console.WriteLine();
    int[] differences = new int[] { 1, 2, 4, 8, 16 };
    Random random = new();
    foreach (int difference in differences)
    {
        int MaxLenght = 0;
        for (int i = 0; i < 1000; i++)
        {
            int[] elementsForChange = Array.Empty<int>();
            string line1 = GetRandomLine(128, random);
            string line2 = line1;
            while (elementsForChange.Length < difference)
            {
                int randomIndex = random.Next(0, line1.Length);
                if (!elementsForChange.Contains(randomIndex))
                {
                    Array.Resize(ref elementsForChange, elementsForChange.Length + 1);
                    elementsForChange[^1] = randomIndex;
                }
            }
            foreach (int element in elementsForChange)
            {
                string oldValue = line2[element].ToString();
                int newValue;
                do
                {
                    newValue = random.Next(0, 9 + 1);
                } while (newValue.ToString() == oldValue);


                line2 = line2.Remove(element, 1).Insert(element, newValue.ToString());

            }

            GOST G = new(256);

            byte[] message1 = Encoding.ASCII.GetBytes(line1);
            byte[] message2 = Encoding.ASCII.GetBytes(line2);

            byte[] res1 = G.GetHash(message1);
            byte[] res2 = G.GetHash(message2);

            string h256_1 = BitConverter.ToString(res1);
            string h256_1_clear = h256_1.Replace("-", string.Empty);
            string h256_2 = BitConverter.ToString(res2);
            string h256_2_clear = h256_2.Replace("-", string.Empty);
            int lenght = LCSubStr(h256_1_clear, h256_2_clear);
            if (lenght > MaxLenght)
                MaxLenght = lenght;
        }
        Console.WriteLine(difference + " " + MaxLenght);
    }
}
```

```
//Тест 2
public static void Test2()
{
    Console.WriteLine();
    Console.WriteLine("Test №2");
    Console.WriteLine();

    GOST G = new(256);
    Random random = new();

    for (int i = 2; i <= 6; ++i)
    {
        int N = (int)Math.Pow(10, i);
        List<string> hashes = new(N);

        for (int j = 0; j < hashes.Count; j++)
        {
            byte[] message = Encoding.ASCII.GetBytes(GetRandomLine(256, random));
            byte[] res = G.GetHash(message);
            string h256_1 = BitConverter.ToString(res);
            string h256_1_clear = h256_1.Replace("-", string.Empty);

            hashes[j] = h256_1_clear;
        }

        List<string> result = hashes.GroupBy(x => x)
                .Where(g => g.Count() > 1)
                .Select(x => x.Key)
                .ToList();
        Console.WriteLine(N + " " + result.Count);
    }
}

//Тест 3
public static void Test3()
{
    Console.WriteLine();
    Console.WriteLine("Test №3");
    Console.WriteLine();

    GOST G = new(256);
    Random random = new();

    List<int> lens = new() { 64, 128, 256, 512, 1024, 2048, 4096, 8192 };

    foreach (int len in lens)
    {
        List<string> strs = new();
        for (int i = 0; i < 1000; ++i)
        {
            strs.Add(GetRandomLine(len, random));
        }

        DateTime start = DateTime.Now;
        for (int i = 0; i < 1000; ++i)
```

```
            {
                byte[] message = Encoding.ASCII.GetBytes(strs[i]);
                G.GetHash(message);
            }
            DateTime end = DateTime.Now;
            TimeSpan time = end - start;

            Console.WriteLine("Len: " + len + " Time: " + time.TotalMilliseconds / 1000.0);
        }
    }

    static void Main(string[] args)
    {
        if (args is null)
        {
            throw new ArgumentNullException(nameof(args));
        }

        Test1();
        Test2();
        Test3();

        Console.WriteLine("Программа закончила вычисления");
        Console.Read();
    }
  }
}
```

Данные выводятся в консоль.

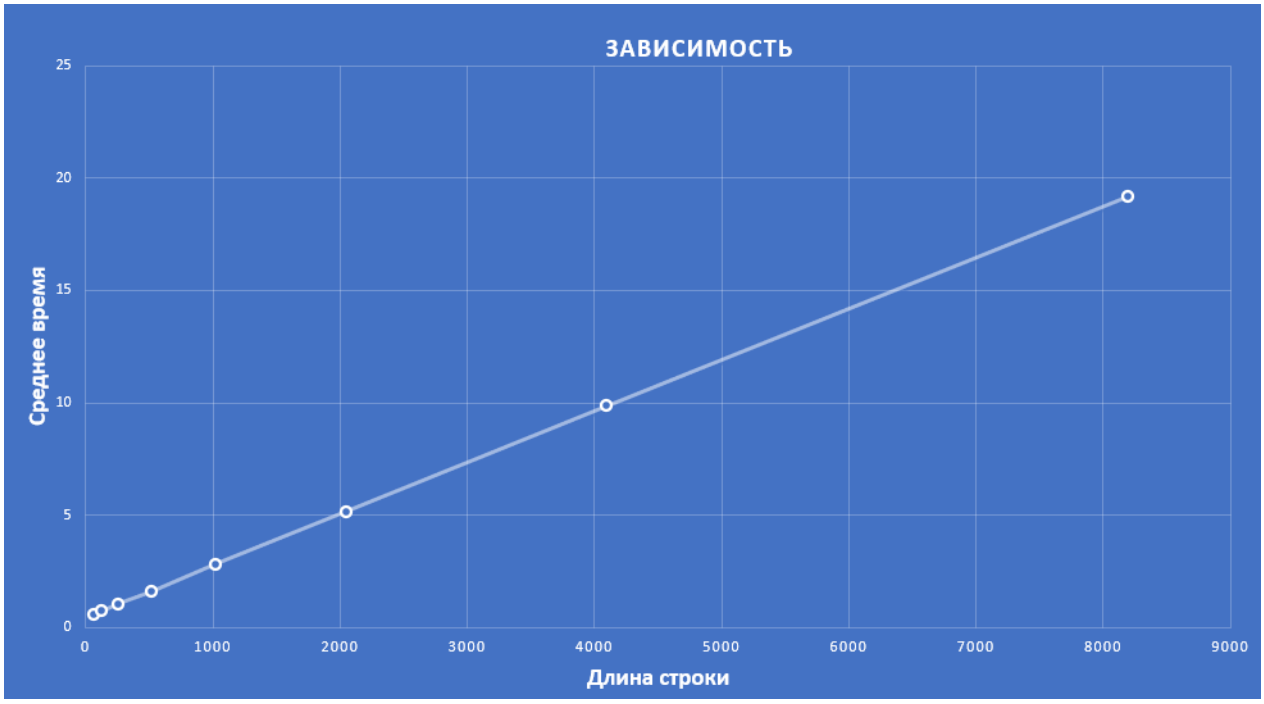
2)



3)

```
Test №2

100 0
1000 0
10000 0
100000 0
1000000 0
```

4)



Заключение.

В ходе выполненной работы был реализован алгоритм хеширования Стриборг. Были проведены три теста:

1. Была сгенерирована 1000 пар строк длиной 128 символов, отличающихся друг от друга 1, 2, 4, 8, 16 символов, и для каждой пары была подсчитана максимальная длина общей подстроки в хешах. Были построены графики зависимости максимальной длины общей подстроки от количества отличий между строками.
2. Была проведена N = 10^i(i от 2 до 6) генерация хешей для случайно сгенерированных строк длиной 256 символов, и для каждого набора данных был выполнен поиск одинаковых хешей. Результаты были представлены в таблице.
3. Были проведены по 1000 генераций хеша для строк длиной 64, 128, 256, 512, 1024, 2048, 4096, 8192 символов, и для каждой длины строки было подсчитано среднее время генерации хеша. Была построена зависимость скорости расчета хеша от размера входных данных.

В результате тестов было установлено, что хеш-функция Стриборг обладает высокой степенью устойчивости к коллизиям и демонстрирует высокую скорость генерации хешей для входных данных различной длины.