Министерство образования и науки 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»


| | |
|---|---|
| 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. Вычислять значение хэш-функции, заданной в варианте:
    1) текст сообщения должен считываться из файла;
    2) полученное значение хэш-функции должно представляться в шестнадцатеричном виде и сохраняться в файл;
    3) при работе программы должна быть возможность просмотра и изменения считанного из файла сообщения и вычисленного значения хэш-функции.
2. Исследовать лавинный эффект на сообщении, состоящем из одного блока:
    1) для бита, который будет изменяться, приложение должно позволять задавать его позицию (номер) в сообщении;
    2) приложение должно уметь после каждого раунда (итерации цикла) вычисления хэш-функции подсчитывать число бит, изменившихся в значении хэш-функции при изменении одного бита в тексте сообщения;
    3) приложение может строить графики зависимости числа бит, изменившихся в значении хэш-функции, от раунда вычисления хэш-функции, либо графики можно строить в стороннем ПО, но тогда приложение должно сохранять в файл необходимую для построения графиков информацию.

II. С помощью реализованного приложения выполнить следующие задания.

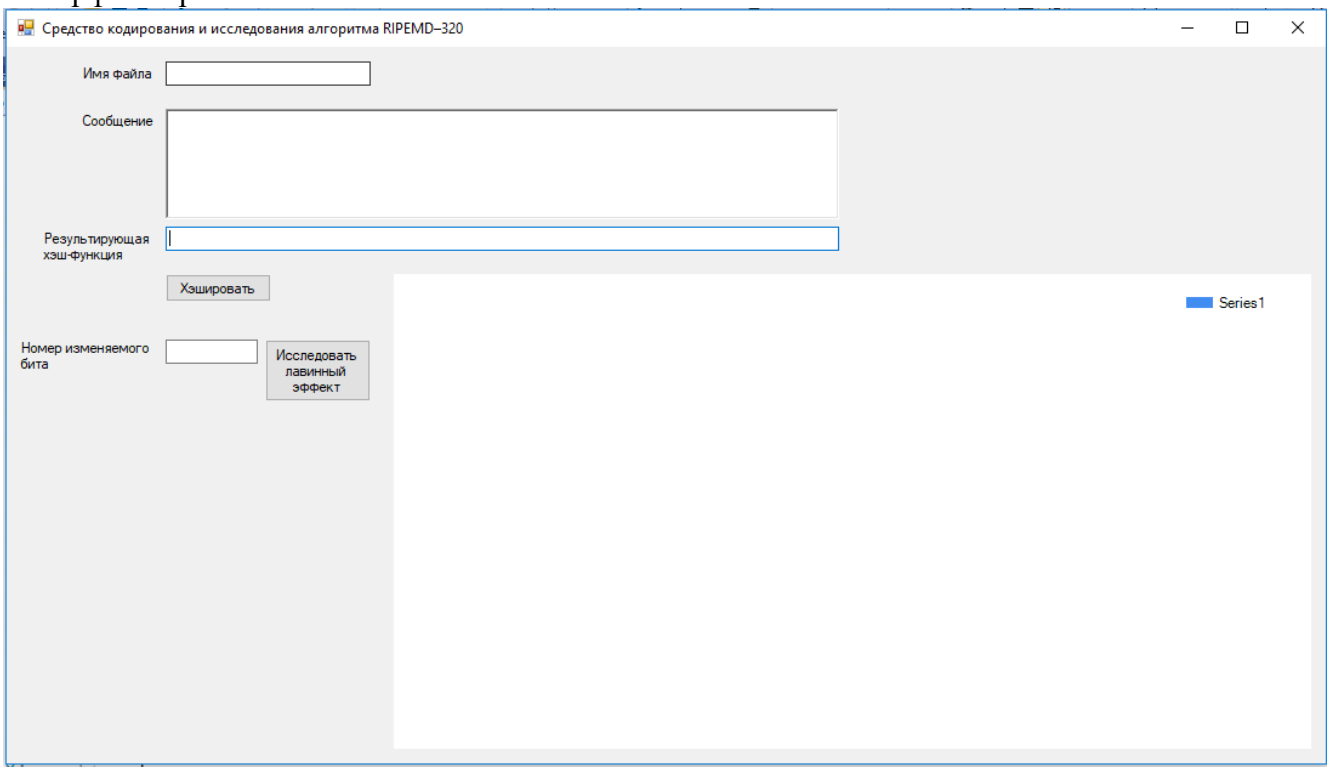
1. Протестировать правильность работы разработанного приложения.
2. Исследовать лавинный эффект при изменении одного бита в сообщении: для различных позиций изменяемого бита в сообщении построить графики зависимостей числа бит, изменившихся в значении хэш-функции, от раунда вычисления хэш-функции (всего в отчете должно быть два-три графика).
3. Сделать выводы о проделанной работе.

**Вариант:** Алгоритм RIPEMD–320

## 3. Описание разработанного программного средства

Разработанная программа способна производить шифрование считанного из указанного файла сообщения по алгоритму RIPEMD–320. Также программа способна проводить исследование лавинного эффекта с выводом графика отличающихся битов. Программа учитывает подаваемые ей параметры (имя файла, номер изменяемого бита, их отсутствие, имя несуществующего файла) и адекватно реагировать на них: выдавать соответствующие сообщения.

Интерфейс приложения:

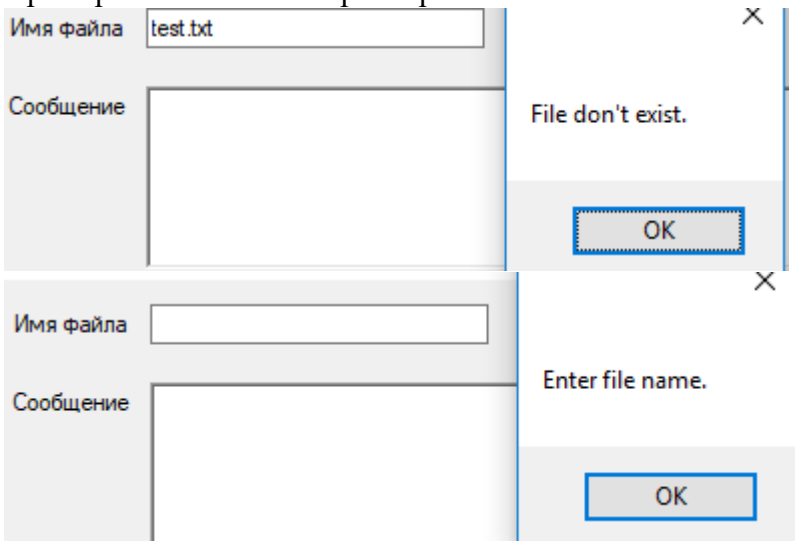



## 4. Исследования

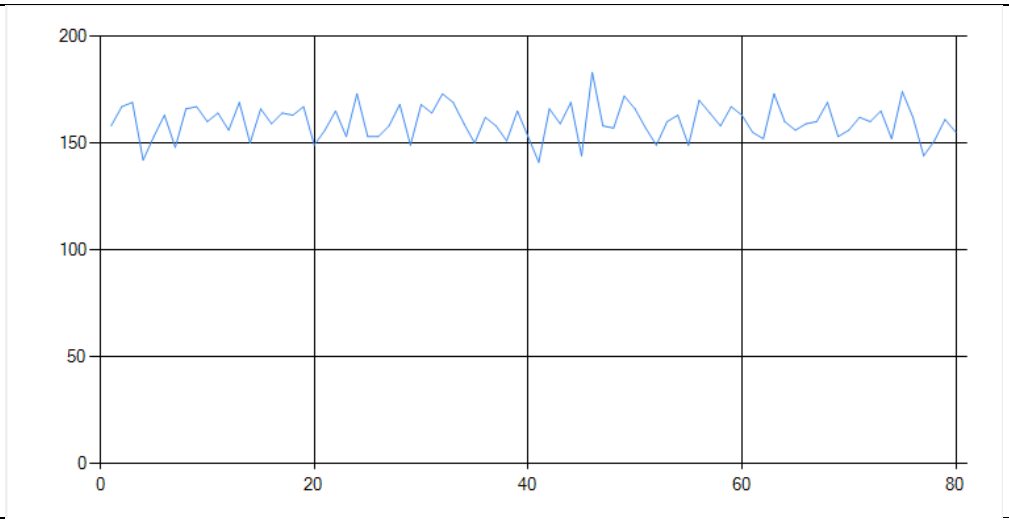
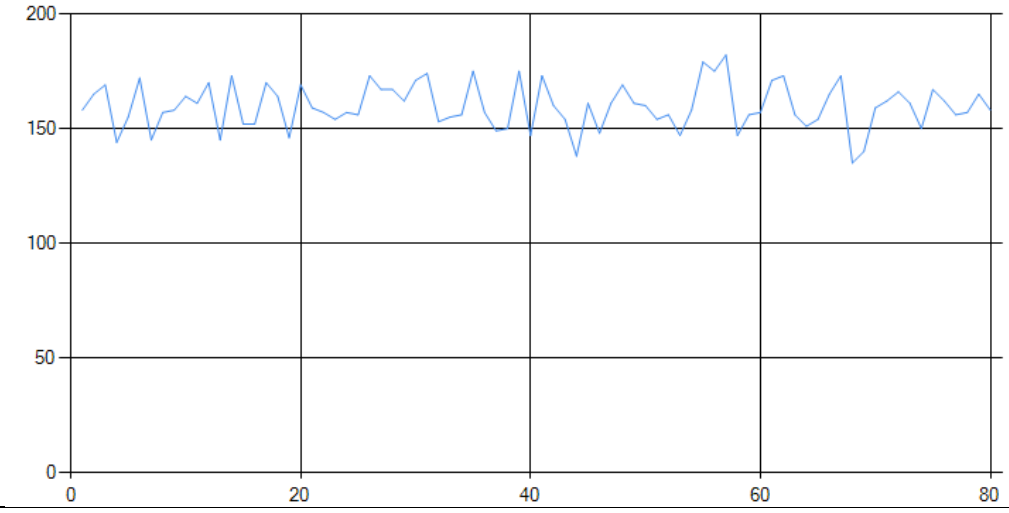
**4.1.** Демонстрация работоспособности на примере хэширования нескольких файлов.

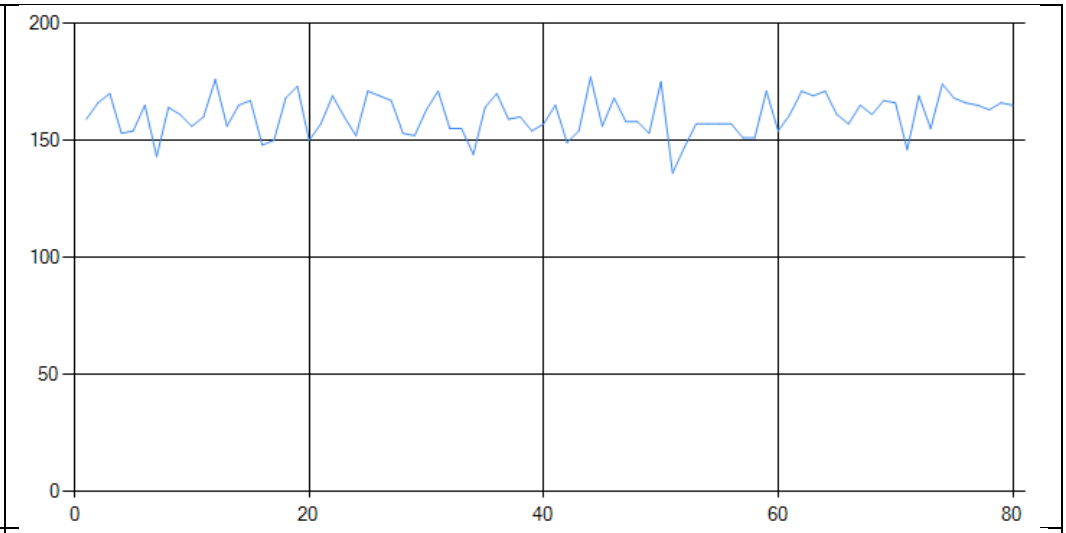
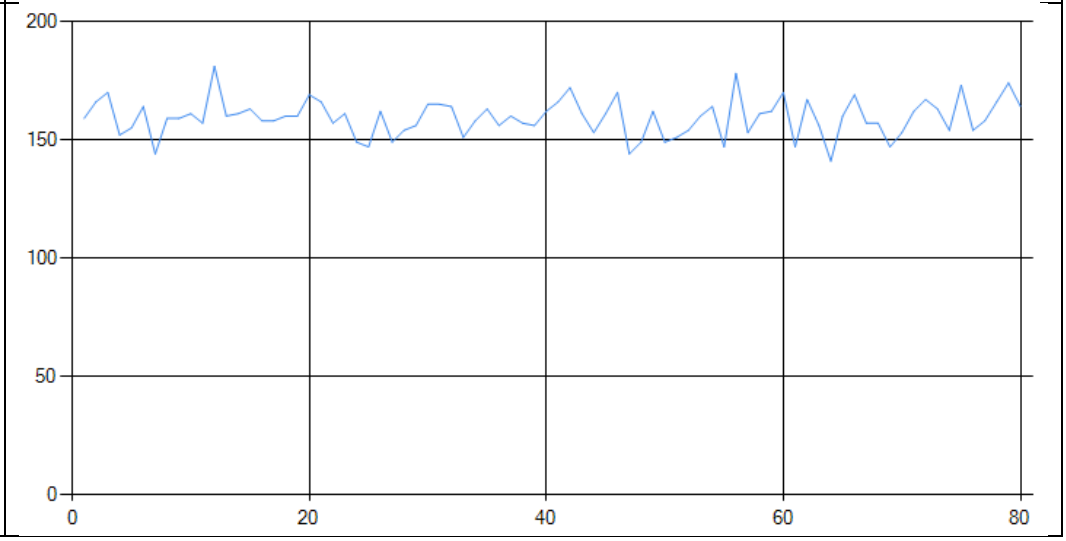
| № теста | Сообщение | Хэши |
|---|---|---|
| 1 | 1231239187123984712-03817 712 380127387120381207452390471274981270471897129834 712089736981273 0126390 8126387126 | ab9e5b95 \| bb298d8c \| 890d5ec6 \| 668ca2c9 \| 45a2b706 \| ee3ef030 \| 598db91 \| eb99c9a2 \| c411434b \| 3da1e0f |
| 2 | Погиб поэт! – невольник чести –Пал, оклеветанный молвой,С свинцом в груди и жаждой мести,Поникнув гордой головой!..Не вынесла душа поэтаПозора мелочных обид,Восстал он против мнений светаОдин, как прежде... и убит!Убит!.. к чему теперь рыданья,Пустых похвал ненужный хорИ жалкий лепет оправданья?Судьбы свершился приговор!Не вы ль сперва так злобно гналиЕго свободный, смелый дарИ для потехи раздувалиЧуть затаившийся пожар?Что ж? веселитесь... – он мученийПоследних вынести не мог:Угас, как светоч, дивный гений,Увял торжественный венок. | 8e8d3600 \| 451b407d \| a0316444 \| 9d00074c \| 78165e37 \| 2010b456 \| 14e9c583 \| 52a0f23f \| 6684015d \| e98dfe26 |
| 3 | мг p29щ 981н 901081n pf2983 281u0`897 `890u08798 u[09u YPIO N*( & {)UJ08y 2908 u79317 rp 91hp890p &GH y9[7 y()# U(&Y P)#U{0 3y(8y#P(&[0i kj p982ph u1bdo8hncvp182 p1u 18[ty[19'u1[83yp12398y0 c1pnc p289py p9PY J8 typ9pih [9 [u[1u ;'u[ au i'qu[8 u92 0923pyv0 iud11 [-1iu 01299 uf0uy fff2p98 hph28c 2 y c28p2p28 opi2uc[0u92[0cuy2p2u98u i [0c 01 inkljhbol;lkp[;i [0iuij [ox0o 09rШРщооЩШОАОР2 33ОГщшрРО3ШЩохОЩШlOjoij[PJ [089u OPIJU[90UIOKMi9u[ j90 ]IP IojlO lJ iu 09i 09 9 `9-lU 1 -1- 0ii -` 9 - -0` -`- -10 [-0 -9 19- U(_l* 0ij9 u-9 lU= )(U90 _ Y897687 | 332c184c \| 127c9142 \| 9652e9a3 \| c4574718 \| 7b5a992e \| e23a61ab \| 2ea024e8 \| c773a1b2 \| 993e84fd \| 4fe81fc8 |
| 4 | | e71aad19 \| 4901881e \| ba0236f8 \| c694d85e \| b75d264a \| 818ba15a \| 4bb052ec \| 97cab4f \| 72e2d116 \| 8a048cf8 |

Примеры ошибочных параметров:




**4.2.** Исследование лавинного эффекта.

| № теста | Номер изменённого бита | Результирующий график<br>Ось X – раунды хэширования<br>Ось Y – изменившиеся биты |
|---|---|---|
| 1 | 0 |  |
| 2 | 10 |  |

| | | |
|---|---|---|
| 3 | 100 |  |
| 4 | 400 |  |

## 5. Код программы

Cripto.cs

```
class Cripto
    {
        int bl = 32;  // длина слова в битах
        int bt = 8;  // длина бита
        int N = 10;  // количество шестнадцатиричных слов в алгоритме
        int Mlen = 512; // длина одного блока в сообщении
        int oper = 80; // число операций в обработке одного 512-битного облока сообщения

         // Массив номеров сообщений 32-битных слов
        int[] R1 = new int[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
                               12, 13, 14, 15, 7, 4, 13, 1, 10, 6,
                               15, 3, 12, 0, 9, 5, 2, 14, 11, 8, 3,
                               10, 14, 4, 9, 15, 8, 1, 2, 7, 0, 6, 13,
                               11, 5, 12, 1, 9, 11, 10, 0, 8, 12, 4,
                               13, 3, 7, 15, 14, 5, 6, 2, 4, 0, 5, 9,
                               7, 12, 2, 10, 14, 1, 3, 8, 11, 6, 15, 13 };
         // Массив номеров сообщений 32-битных слов
        int[] R2 = new int[] { 5, 14, 7, 0, 9, 2, 11, 4, 13, 6, 15, 8,
                               1, 10, 3, 12, 6, 11, 3, 7, 0, 13, 5,
                               10, 14, 15, 8, 12, 4, 9, 1, 2, 15, 5,
                               1, 3, 7, 14, 6, 9, 11, 8, 12, 2, 10, 0,
                               4, 13, 8, 6, 4, 1, 3, 11, 15, 0, 5, 12,
                               2, 13, 9, 7, 10, 14, 12, 15, 10, 4, 1,
                               5, 8, 7, 6, 2, 13, 14, 0, 3, 9, 11 };

         // Массив битовых сдвигов
```

```
        int[] S1 = new int[] { 11, 14, 15, 12, 5, 8, 7, 9, 11, 13, 14, 15,
                               6, 7, 9, 8, 7, 6, 8, 13, 11, 9, 7, 15, 7, 12,
                              15, 9, 11, 7, 13, 12, 11, 13, 6, 7, 14, 9, 13,
                              15, 14, 8, 13, 6, 5, 12, 7, 5, 11, 12, 14, 15,
                              14, 15, 9, 8, 9, 14, 5, 6, 8, 6, 5, 12, 9, 15,
                              5, 11, 6, 8, 13, 12, 5, 12, 13, 14, 11, 8, 5, 6 };
         // Массив битовых сдвигов
        int[] S2 = new int[] { 8, 9, 9, 11, 13, 15, 15, 5, 7, 7, 8, 11, 14,
                              14, 12, 6, 9, 13, 15, 7, 12, 8, 9, 11, 7, 7,
                              12, 7, 6, 15, 13, 11, 9, 7, 15, 11, 8, 6, 6,
                              14, 12, 13, 5, 14, 13, 13, 7, 5, 15, 5, 8, 11,
                              14, 14, 6, 14, 6, 9, 12, 9, 12, 5, 15, 8, 8, 5,
                              12, 9, 12, 5, 14, 6, 8, 13, 6, 5, 15, 13, 11, 11 };

        uint[] k_const1 = new uint[5] { 0x00000000, 0x5a827999, 0x6ed9eba1, 0x8f1bbcdc, 0xa953fd4e }; // массив
для фукнции K1
        uint[] k_const2 = new uint[5] { 0x50a28be6, 0x5c4dd124, 0x6d703ef3, 0x7a6d76e9, 0x00000000 }; // массив
для фукнции K2
        uint[] h = new uint[10] { 0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476, 0xc3d2e1f0,
                                  0x76543210, 0xfedcba98, 0x89abcdef, 0x01234567, 0x3c2d1e0f }; // массив
начальных для выходного массива 4-битовых хэшей
        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 string[] Read(string fileName)  // чтение строки
        {
            List<string> list = new List<string>();
            using (StreamReader reader = new StreamReader(fileName, Encoding.GetEncoding(1251)))
            {
                while (!reader.EndOfStream)
                {
                    string str = reader.ReadLine();
                    list.Add(str);
                }
            }
            return list.ToArray();
        }

        public int[] in_mass(int[] x, int[] y, int j) // поместить массив y в массив x начиная с j-позиции
        {
            for (int i = 0; i < y.Length; i++)
                x[j + i] = y[i];
            return x;
        }


        public int[] TextToBin(string X) // перевод строки в 2-ую форму
        {
            int[] o = new int[1]; // выходной массив двоичных чисел
            int num = 0;
            int a = 0;
            int u = 0;
            for (int i = 0; i < X.Length; i++) // основной цикл по строке X
            {
                int[] b = new int[1];
                int j = 0;
                num = (int)X[i];
                while (num >= 1)
                {
                    a = num % 2;
                    Array.Resize(ref b, b.Length + 1);
                    b[j] = a;
                    j++;
                    num = num / 2;
                };
                Array.Resize(ref b, b.Length - 1); // удаление лишнего элемента
                Array.Resize(ref o, b.Length + u);
                o = in_mass(o, b, u);
                u += b.Length;

            }
            return o;
```

```
        }


        public int[] Tobin(int i) // перевод числа в двоичную форму
        {
            int[] b = new int[1];
            int a = 0;
            int j = 0;
            while (i >= 1)
            {
                a = i % 2;
                b[j] = a;
                j++;
                Array.Resize(ref b, b.Length + 1);

                i = i / 2;
            }
            Array.Resize(ref b, b.Length - 1);
            Array.Reverse(b, 0, b.Length);
            return b;
        }

        public uint shift(uint x, int S) // циклический битовый сдвиг
        {
            return (x << S) ^ (x >> (bl - S)); // проводим операцию XOR между смещённым влево на S бит числом и
числом, смещённым на 32 - S бит
        }

        public uint f(int j, uint x, uint y, uint z) //  битовая функция f
        {
            uint f_out = 0;

            if (0 <= j && j <= 15)
                f_out = x ^ y ^ z;
            if (16 <= j && j <= 31)
                f_out = (x & y) | (~x & z);
            if (32 <= j && j <= 47)
                f_out = (x | ~y) ^ z;
            if (48 <= j && j <= 63)
                f_out = (x ^ z) | (y & ~z);
            if (64 <= j && j <= 79)
                f_out = x ^ (y | ~z);

            return f_out;
        }

        public uint K_f(int j, int f) //  функции K1 и K2
        {
            uint f_out = 0;

            if (f == 0)
                f_out = k_const1[j / 16];
            else
                f_out = k_const2[j / 16];

            return f_out;
        }

        public int[] take_mas(int[] X, int j, int l) // получение части массива X длиной l спозиции j
        {
            int[] f_out = new int[l];

            for (int i = 0; i < l; i++)
                f_out[i] = X[j + i];
            return f_out;
        }

        public double btod(int[] X, int f) // перевод двоичного числа в десятичное (f - параметр разряда числа)
        {
            double res = 0;
            for (int i = 0; i < f; i++)
            {
                res += X[i] * Math.Pow(2, 7 - i);
            }
            return res;
        }
```

```
        public bool sort(int[] X) // проверка упорядоченности массива X
        {
            int i = 0;
            while (i < X.Length - bt)
            {
                if (btod(take_mas(X, i, bt), bt) > btod(take_mas(X, i + bt, bt), bt))
                    return false;
                i += bt;
            }
            return true;
        }

        public int[] l_end(int[] X) // перевод сообщения к порядку little-endian (сортировка от)
        {
            int[] f_out = new int[X.Length]; // выходной массив
            int[] buf = new int[bt]; // буфер длиной

            f_out = in_mass(f_out, X, 0); // перенос входного массива в выходной массив

            while (!sort(f_out))
            {
                int i = 0;
                while (i < f_out.Length - bt)
                {
                    if (btod(take_mas(f_out, i, bt), bt) > btod(take_mas(f_out, i + bt, bt), bt)) // сравнение
соседних 32-битных слов
                    {
                        buf = take_mas(f_out, i, bt);
                        for (int j = 0; j < bt; j++)
                            f_out[i + j] = f_out[i + bt + j];
                        f_out = in_mass(f_out, buf, i + bt);
                    }
                    i += bt;
                }
            }
            return f_out;
        }

        public int[] mes_add(string X) // добавление недостающих бит в сообщение
        {
            int[] a_mes = new int[1]; // дополненное сообщение
            int[] beg_bit = new int[1]; // массив с двоичным представлением длины двоичной формы исходного
сообщения
            int[] l_bit = new int[64]; // младшие биты beg_bit

            a_mes = TextToBin(X); // получение двоичного представления исходного сообщения
            int beg_len = a_mes.Length; // получения длины a_mes

            Array.Resize(ref a_mes, a_mes.Length + 1);
            a_mes[a_mes.Length - 1] = 1; // добавления 1 в конец сообщения

            while (a_mes.Length % 512 != 448) // дополнение сообщения нулями
            {
                Array.Resize(ref a_mes, a_mes.Length + 1);
                a_mes[a_mes.Length - 1] = 0;
            }

            a_mes = l_end(a_mes);
            Array.Resize(ref a_mes, a_mes.Length + 64);

            beg_bit = Tobin(beg_len); // преобразование длины сообщения в двоичную форму

            if (beg_bit.Length < 64) // если длина меньше 64 бит, то недостающие биты заполняем нулями
            {
                Array.Reverse(beg_bit, 0, beg_bit.Length);
                Array.Reverse(l_bit, 0, l_bit.Length);
                l_bit = in_mass(l_bit, beg_bit, 0);
                Array.Reverse(l_bit, 0, l_bit.Length);
                a_mes = in_mass(a_mes, l_bit, a_mes.Length - 64);
            }
            else
            {
                int[] buf = new int[32];
                Array.Reverse(beg_bit, 0, beg_bit.Length);
                l_bit = take_mas(beg_bit, 0, 64);
                Array.Reverse(l_bit, 0, l_bit.Length);
                buf = take_mas(l_bit, 32, 32);
```

```
            Array.Resize(ref l_bit, l_bit.Length - 32);
            a_mes = in_mass(a_mes, buf, a_mes.Length - 64);
            a_mes = in_mass(a_mes, l_bit, a_mes.Length - 32);
        }
        return a_mes;
    }
    public uint[] Pars(int[] X) // парсим сообщение из массива int чисел, в массив uint чисел
    {
        uint[] w = new uint[Mlen / bl]; // выделяем массив на 16 32-разрядных числа
        int i = 0;
        int k = 0;
        while (i < X.Length)
        {
            string buf = "";
            for (int j = 0; j < bl; j++)
                buf += X[i + j].ToString();
            w[k] = Convert.ToUInt32(buf, 2);
            k++;
            i += bl;
        }
        return w;
    }

    public uint[] encoder(string Mess) // функция шифрования (на вход передаём сообщение в строке)
    {
        int[] X = mes_add(Mess); // массив под сообщение в int фомрате
        uint[] H = new uint[N]; // массив результирующих хэшей
        uint[] buf1 = new uint[N / 2]; // массив хэшей A1, B1, C1, D1, E1
        uint[] buf2 = new uint[N / 2]; // массив хэшей A2, B2, C2, D2, E2
        uint[] w = new uint[Mlen / bl]; // массив с одним 512-битным блоком, разбитый на 16 32-битных слова
        uint T = 0; //
        int i = 0;
        for (int k = 0; k < h.Length; k++) // заносим начальные значения хэшей
            H[k] = h[k];

        while (i < X.Length) // цикл по всему сообщению
        {
            w = Pars(take_mas(X, i, Mlen)); // получаем 512-битный блок
            for (int k = 0; k < N / 2; k++) // заносим в массивы хэшей начальные значения
            {
                buf1[k] = H[k];
                buf2[k] = H[k + N / 2];
            }

            for(int j = 0; j < oper; j++) // основной цикл шифрования
            {
                T = shift((buf1[0] ^ f(j, buf1[1], buf1[2], buf1[3])
                    ^ w[R1[j]] ^ K_f(j, 0)), S1[j]) ^ buf1[4];
                buf1[0] = buf1[4];
                buf1[4] = buf1[3];
                buf1[3] = shift(buf1[2], 10);
                buf1[2] = buf1[1];
                buf1[1] = T;
                T = shift((buf2[0] ^ f(oper - 1 - j, buf2[1], buf2[2], buf2[3])
                    ^ w[R2[j]] ^ K_f(j, 1)), S2[j]) ^ buf2[4];
                buf2[0] = buf2[4];
                buf2[4] = buf2[3];
                buf2[3] = shift(buf2[2], 10);
                buf2[2] = buf2[1];
                buf2[1] = T;

                if(j == 15)
                {
                    T = buf1[1];
                    buf1[1] = buf2[1];
                    buf2[1] = T;
                }
                if (j == 31)
                {
                    T = buf1[3];
                    buf1[3] = buf2[3];
                    buf2[3] = T;
                }
                if (j == 47)
                {
                    T = buf1[0];
                    buf1[0] = buf2[0];
```

```
                    buf2[0] = T;
                }
                if (j == 63)
                {
                    T = buf1[2];
                    buf1[2] = buf2[2];
                    buf2[2] = T;
                }
                if (j == 79)
                {
                    T = buf1[4];
                    buf1[4] = buf2[4];
                    buf2[4] = T;
                }
            }
            for(int k = 0; k < N / 2; k++) // получаем промежуточные значения хэшей
            {
                H[k] = H[k] ^ buf1[k];
                H[k + N / 2] = H[k + N / 2] ^ buf2[k];
            }
            i += Mlen; // смещаемся в сообщении на следующие 512 бит
        }

        return H;
    }

    public int bit_compar(uint[] X, uint[] Y) // сравнение двух хэш-функций на отличающиеся биты
    {
        int ch = 0; // число изменившихся бит

        for (int i = 0; i < X.Length; i++)
        {
            // конертируем массив uint сначала в массив байтов, затем в массив битов
            byte[] bufX = BitConverter.GetBytes(X[i]);
            byte[] bufY = BitConverter.GetBytes(Y[i]);
            BitArray X1 = new BitArray(bufX);
            BitArray Y1 = new BitArray(bufY);

            for (int j = 0; j < X1.Length; j++)
                if (X1[j] != Y1[j])
                    ch++;
        }
        return ch;
    }

    public uint[] bit_corrector(uint[] X, int y) // инвертирует бит под номером y
    {
        uint buf = 1;
        X[y / 32] ^= (buf << (bl - 1 - y % bl));
        return X;
    }

    public int[] avalanche_eff(string Mess, uint[] H_f, int n_bit) // функция исследования лавинного эффекта
при изменении бита под номером n_bit)
    {
        int[] X = mes_add(Mess); // массив под сообщение в int фомрате
        uint[] H_test = new uint[N]; // массив первичных хэшей для исследования
        uint[] buf1 = new uint[N / 2]; // массив хэшей A1, B1, C1, D1, E1
        uint[] buf2 = new uint[N / 2]; // массив хэшей A2, B2, C2, D2, E2
        uint[] w = new uint[Mlen / bl]; // массив с одним 512-битным блоком, разбитый на 16 32-битных слова
        int[] stat = new int[oper];
        uint T = 0; //

        w = Pars(take_mas(X, 0, Mlen)); // получаем 512-битный блок


        for (int k = 0; k < h.Length; k++) // заносим начальные значения хэшей
            H_test[k] = h[k];

        for (int k = 0; k < N / 2; k++) // заносим в массивы хэшей начальные значения
        {
            buf1[k] = H_test[k];
            buf2[k] = H_test[k + N / 2];
        }

        w = bit_corrector(w, n_bit); // изменяем бит n_bit в сообщении
```

```
            for (int j = 0; j < oper; j++) // основной цикл шифрования
            {
                T = shift((buf1[0] ^ f(j, buf1[1], buf1[2], buf1[3])
                    ^ w[R1[j]] ^ K_f(j, 0)), S1[j]) ^ buf1[4];
                buf1[0] = buf1[4];
                buf1[4] = buf1[3];
                buf1[3] = shift(buf1[2], 10);
                buf1[2] = buf1[1];
                buf1[1] = T;
                T = shift((buf2[0] ^ f(oper - 1 - j, buf2[1], buf2[2], buf2[3])
                    ^ w[R2[j]] ^ K_f(j, 1)), S2[j]) ^ buf2[4];
                buf2[0] = buf2[4];
                buf2[4] = buf2[3];
                buf2[3] = shift(buf2[2], 10);
                buf2[2] = buf2[1];
                buf2[1] = T;

                if (j == 15)
                {
                    T = buf1[1];
                    buf1[1] = buf2[1];
                    buf2[1] = T;
                }
                if (j == 31)
                {
                    T = buf1[3];
                    buf1[3] = buf2[3];
                    buf2[3] = T;
                }
                if (j == 47)
                {
                    T = buf1[0];
                    buf1[0] = buf2[0];
                    buf2[0] = T;
                }
                if (j == 63)
                {
                    T = buf1[2];
                    buf1[2] = buf2[2];
                    buf2[2] = T;
                }
                if (j == 79)
                {
                    T = buf1[4];
                    buf1[4] = buf2[4];
                    buf2[4] = T;
                }

                for (int k = 0; k < N / 2; k++) // получаем промежуточные значения хэшей
                {
                    H_test[k] ^= buf1[k];
                    H_test[k + N / 2] ^= buf2[k];
                }
                stat[j] = bit_compar(H_f, H_test); // сравниваем полученную хэш-функцию с изначальной
            }

            return stat;
        }
    }
```

Form1.cs

```
        Cripto Cr = new Cripto(); // инициализируем основной класс
        uint[] H = new uint[1]; // массив хэш-функции
        string[] m = new string[1]; // промежуточный массив сообщения (нужен для чтения из файла)
        string mess = ""; // сообщение

        private void Button1_Click(object sender, EventArgs e)
        {
            string file_name = textBox3.Text; // имя файла

            if (file_name == "") // проверка на ввод имени файла
                MessageBox.Show("Enter file name.");
            else
            {
                bool fl = File.Exists(file_name); // проверка на существование файла
```

```
        if (fl)
        {
            richTextBox1.Clear();
            m = Cr.Read(file_name);
            for (int i = 0; i < m.Length; i++)
                mess += m[i];
            for (int i = 0; i < m.Length; i++)
                richTextBox1.Text += m[i];

            if (m.Length == 0)
                mess = "";

            textBox1.Clear();

            H = Cr.encoder(mess); // функция хэширования
            for (int i = 0; i < H.Length; i++)
                textBox1.Text += (H[i]).ToString("x") + " | ";
        }
        else
            MessageBox.Show("File don't exist.");

    }
}

private void Button2_Click(object sender, EventArgs e)
{
    string m = textBox2.Text; // номер бита, который нужно изменить

    if (m == "")
        MessageBox.Show("Enter the bit number.");
    else
    {
        int n_bit = Convert.ToInt32(m); // номер изменяемого бита в int формате
        int[] stat = Cr.avalanche_eff(mess, H, n_bit); // получаем статистику изменения хэш-функции
        chart1.Series.Clear();
        // Add series.
        for (int i = 0; i < stat.Length; i++)
        {
            // Add series.
            chart1.Series.Add(Convert.ToString(i));
            chart1.Series[0].Points.AddXY(i + 1, stat[i]);

        }
        chart1.Series[0].ChartType = SeriesChartType.Line;
        chart1.Legends.Clear();
    }
}
```

## 6. Выводы

В ходе выполненной лабораторной было разработано программное средство, предназначенное для вычисления хэш-функции текстового сообщения алгоритмом RIPEMD–320, а также для исследования лавинного эффекта на примере данного алгоритма.

В ходе исследования лавинного эффекта был, исходя из графиков изменения хэш-функции, можно сделать вывод, что даже изменение одного бита в исходном сообщении приводит к изменению в среднем более половины всех битов хэщ-функции уже на первом раунде хэширования.