Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

|  |
| --- |
| Институт космических и информационных технологий |
| *институт* |
| Кафедра Прикладной математики и компьютерной безопасности |
| *кафедра* |

**ЛАБОРАТОРНАЯ РАБОТА №8**

|  |
| --- |
| Хэш-функция ГОСТ Р 34.11-2012 |
| *тема* |

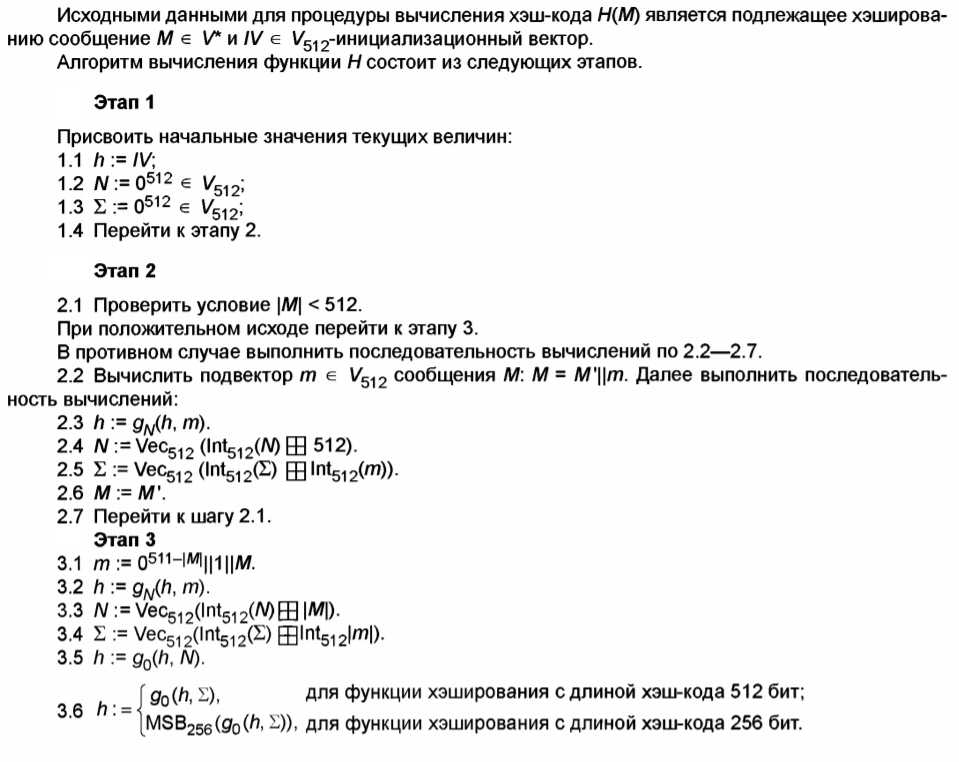
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Руководитель | | |  |  |  | В.И.Вайнштейн |
|  | |  |  | *подпись, дата* |  | *инициалы, фамилия* |
| Студент | КИ15-01 №031508683 | |  |  |  | М.С.Димаксян |
|  | *номер группы, зачетной книжки* | |  | *подпись, дата* |  | *инициалы, фамилия* |

Красноярск 2019

**Задание:**

Изучить основные принципы построения и реализации алгоритма хэширования ГОСТ Р 34.11-2012.

**Теория:**



**Исходный код:**

**Функции интерфейса:**

private void f8\_buttonCountHashFromTextbox\_Click(object sender, EventArgs e)

{

byte[] tb = Encoding.Default.GetBytes(f8\_textBox\_original.Text);

f8\_TextBox\_Hash256.Text = GetHashGOST(tb, 256);

f8\_TextBox\_Hash512.Text = GetHashGOST(tb, 512);

}

private void f8\_buttonCountHashFromFile\_Click(object sender, EventArgs e)

{

OpenFileDialog Load = new OpenFileDialog();

Load.InitialDirectory = "D:\\Учеба\\Крипта 2\\Labs\\Labs\\bin\\Debug\\TextFiles";

Load.DefaultExt = "txt";

Load.FileName = "in.txt";

Load.Filter = "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*";

Load.FilterIndex = 2;

Load.RestoreDirectory = true;

Load.Multiselect = false;

string sFileName;

if (Load.ShowDialog() == DialogResult.OK)

{

sFileName = Load.FileName;

byte[] file = File.ReadAllBytes(Load.FileName);

f8\_TextBox\_Hash256.Text = GetHashGOST(file, 256);

f8\_TextBox\_Hash512.Text = GetHashGOST(file, 512);

}

}

**Хэш-функция:**

private string GetHashGOST(byte[] arr, int hashLen)

{

arr = ReorderArray(arr);

// Векторы для 512-битового и 256-битового хэша

byte[] vector0 = new byte[64]; // 0^512

byte[] vector1 = vector0.Select(x => (byte)1).ToArray(); // (00000001)^512

// Этап 1 - инициализация начальных параметров

byte[] IV = hashLen == 256 ? vector1 : vector0;

byte[] h = IV;

byte[] N = vector0;

byte[] Z = vector0;

byte[] m;

// Этап 2 - сжатие сообщения до размера < 512 бит

while (arr.Length > 64)

{

m = arr.Take(64).ToArray();

h = CompressionFunction(N, h, m);

N = AddMod512(N, ((BigInteger)512).ToByteArray());

Z = AddMod512(Z, m);

arr= arr.Skip(64).ToArray();

}

// Этап 3 - вычисление хэш-кода

m = new byte[64];

Array.Copy(arr, 0, m, 0, arr.Length);

m[arr.Length] = 1;

h = CompressionFunction(N, h, m);

N = AddMod512(N, ((BigInteger)arr.Length \* 8).ToByteArray());

Z = AddMod512(Z, m);

h = CompressionFunction(vector0, h, N);

h = CompressionFunction(vector0, h, Z);

// Для 256-битового хэша

if (hashLen == 256) h = h.Skip(32).ToArray();

string str = "";

foreach (var bt in h)

{

var tmp = bt.ToString("x");

if (tmp.Length < 2)

tmp = "0" + tmp;

str = tmp + str;

}

return str;

}

**Вспомогательные функции и константы:**

// пи подстановка

private static readonly byte [] pi = {

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

};

// l-подстановка

private static readonly ulong[] l = {

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

};

// перестановка байт

private static readonly byte[] t = {

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

};

// итерационные константы

private static 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 static byte[] AddMod512(byte[] a, byte[] b) // Сложение в кольце 2^512

{

byte[] tmp = ((new BigInteger(a) + new BigInteger(b)) % BigInteger.Pow(2, 512)).ToByteArray();

byte[] res = new byte[64];

try

{

Array.Copy(tmp, 0, res, 0, tmp.Length);

}

catch (Exception ex)

{

Array.Copy(tmp, 0, res, 0, 64);

}

return res;

}

private static byte[] CompressionFunction(byte[] N, byte[] h, byte[] m) // Функция сжатия

{

byte[] result = TransS(TransX(h, N));

result = TransP(result);

result = TransL(result);

result = TransE(result, m);

result = TransX(result, h);

result = TransX(result, m);

return result;

}

private static byte[] TransE(byte[] K, byte[] m) // Е преобразование

{

byte[] result = m;

for (int i = 0; i < 12; i++)

{

result = TransX(K, result);

result = TransS(result);

result = TransP(result);

result = TransL(result);

K = GetNextK(K, i);

}

result = TransX(K, result);

return result;

}

private static byte[] GetNextK(byte[] K, int i)

{

// Вычисляем следующий итерационный ключ

// K - предыдущий ключ, i - номер итерации

byte[] result = TransX(K, C[i].Reverse().ToArray());

result = TransS(result);

result = TransP(result);

result = TransL(result);

return result;

}

private static byte[] TransL(byte[] a) // L-преобразование

{

byte[] result = new byte[64];

for (int i = 0; i < 8; i++)

{

ulong t = 0;

byte[] tempArray = new byte[8];

Array.Copy(a, i \* 8, tempArray, 0, 8);

tempArray = tempArray.ToArray();

BitArray tempBits1 = new BitArray(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 = t ^ l[j];

}

}

byte[] resPart = BitConverter.GetBytes(t);

Array.Copy(resPart, 0, result, i \* 8, 8);

}

return result;

}

private static byte[] TransP(byte[] a) // P-преобразование

{

byte[] result = new byte[64];

for (int i = 0; i < 64; i++)

{

result[i] = a[t[i]];

}

return result;

}

private static byte[] TransS(byte[] a) // S-преобразование

{

byte[] result = new byte[64];

for (int i = 0; i < 64; i++)

{

result[i] = pi[a[i]];

}

return result;

}

private static byte[] TransX(byte[] k, byte[] a) // X-преобразование

{

byte[] result = new byte[64];

for (int i = 0; i < 64; i++)

{

result[i] = (byte)(k[i] ^ a[i]);

}

return result;

}

private byte[] ReorderArray (byte[] array)

{

byte[] tmp = new byte[array.Length];

for (int i = 0; i < array.Length; i++)

tmp[i] = array[array.Length - i - 1];

return tmp;

}

**Результат работы программы:**

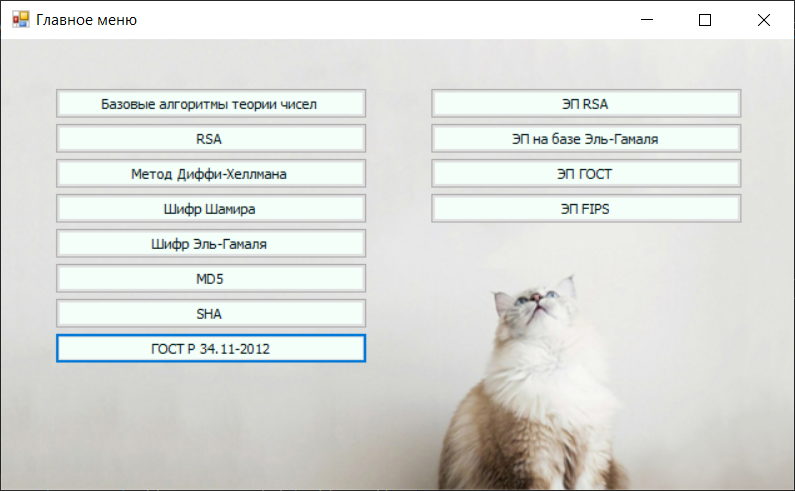


Рисунок 1. Главное меню программы.

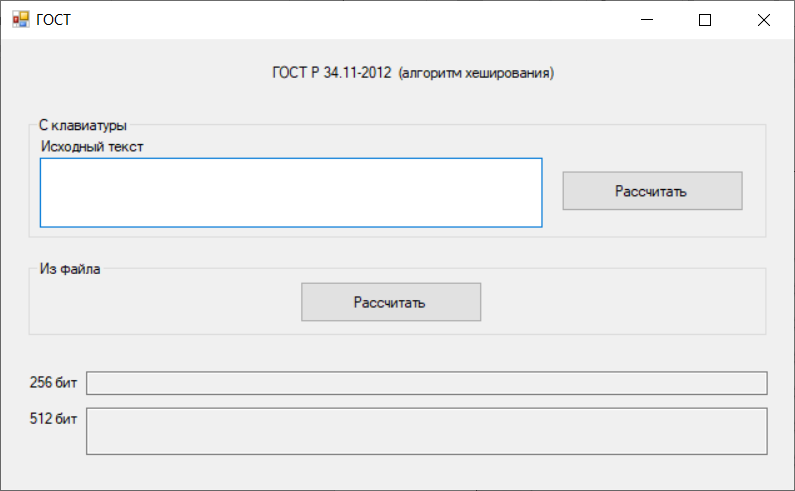


Рисунок 2. Окно хэш-функции ГОСТ Р 34.11-2012.

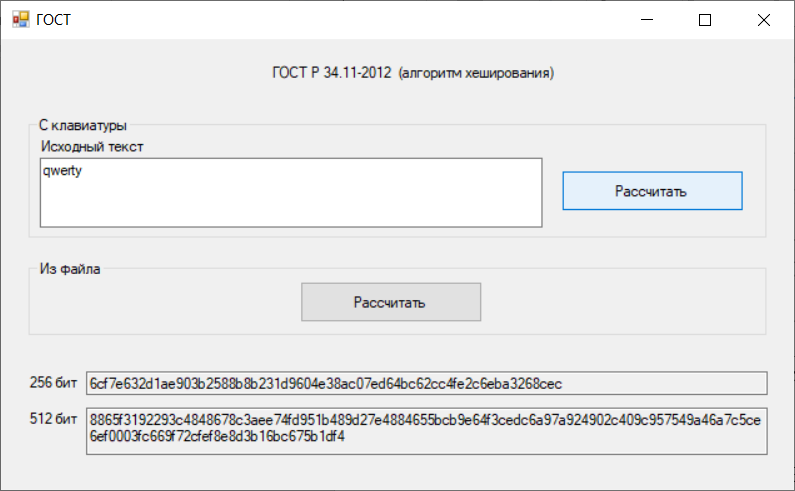


Рисунок 3. Расчет из текстового поля.

Рассчитаем контрольные примеры, приведенные в документе ГОСТ Р 34.11‑2012.

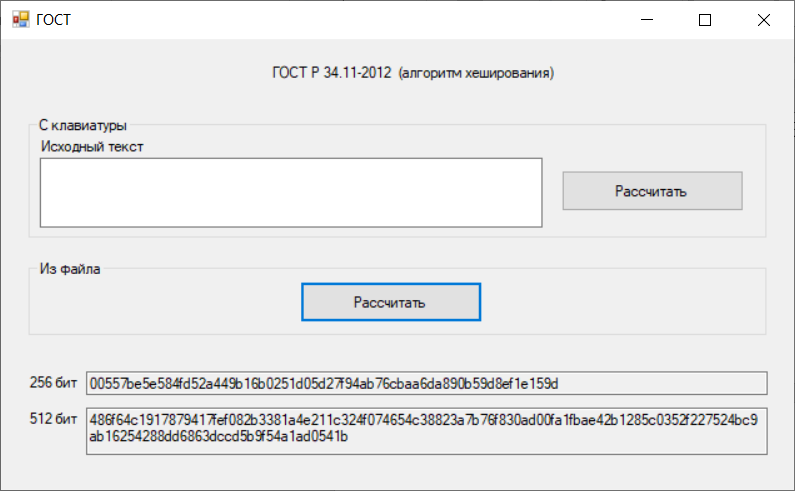


Рисунок 4. Расчет примера 1 из файла.

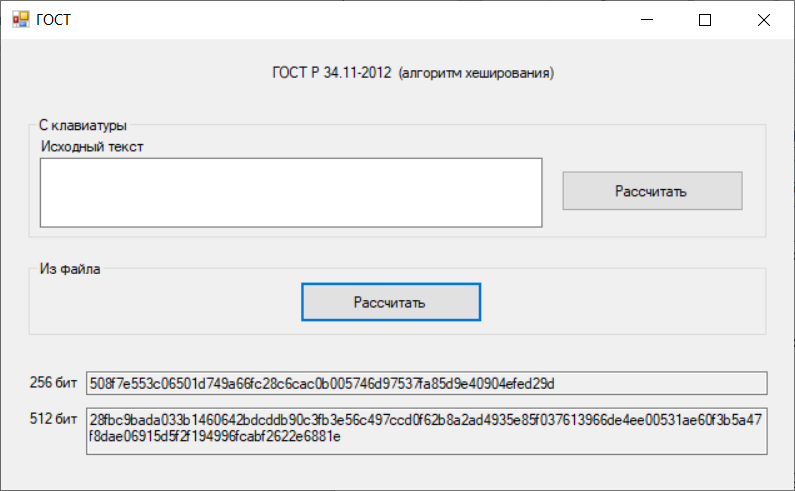


Рисунок 5. Расчет примера 2 из файла.

**Вывод:**

В ходе выполнения лабораторной работы я ознакомилась с функцией хэширования ГОСТ Р 34.11-2012, реализовала на практике программу получения хэша на языке C#. В процессе написания программы я также получила ценный опыт в отладке алгоритмов хэширования.