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 №6**

|  |
| --- |
| Хэш-функция MD5 |
| *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 MD5

**Теория:**

Алгоритм состоит из пяти шагов:  
1)Append Padding Bits  
В исходную строку дописывают единичный байт 0х80, а затем дописывают нулевые биты, до тех пор, пока длина сообщения не будет сравнима с 448 по модулю 512. То есть дописываем нули до тех пор, пока длина нового сообщения не будет равна [длина] = (512\*N+448),  
где N — любое натуральное число, такое, что это выражение будет наиболее близко к длине блока.  
2)Append Length  
Далее в сообщение дописывается 64-битное представление длины исходного сообщения.  
3)Initialize MD Buffer  
На этом шаге инициализируется буффер  
word A: 01 23 45 67  
word B: 89 ab cd ef  
word C: fe dc ba 98  
word D: 76 54 32 10  
Как можно заметить буффер состоит из четырех констант, предназначенный для сбора хэша.  
4)Process Message in 16-Word Blocks  
На четвертом шаге в первую очередь определяется 4 вспомогательные логические функции, которые преобразуют входные 32-битные слова, в, как ни странно, в 32-битные выходные.  
F(X,Y,Z) = XY v not(X) Z G(X,Y,Z) = XZ v Y not(Z)  
H(X,Y,Z) = X xor Y xor Z I(X,Y,Z) = Y xor (X v not(Z))  
Также на этом шаге реализуется так называемый «белый шум» — усиление алгоритма, состоящее 64 элементного массива, содержащего псевдослучайные числа, зависимые от синуса числа i:  
T[i]=4,294,967,296\*abs(sin(i))  
Далее начинается «магия». Копируем каждый 16-битный блок в массив X[16] и производим манипуляции:  
AA = A  
BB = B  
CC = C  
DD = D  
Далее каждый из четырех раунд состоит из 16 элементарных преобразований, которые в общем виде можно представить в виде [abcd k s i], которое, в свою очередь, можно представить как A = B + ((A + F(B,C,D) + X[k] + T[i]) <<< s), где  
A, B, C, D — регистры  
F(B,C,D) — одна из логических функций  
X[k] — k-тый элемент 16-битного блока.  
T[i] — i-тый элемент таблицы «белого шума»  
<<< s — операция циклического сдвига на s позиций влево.  
Приводить все раунды не имеет смысла, все их можно посмотреть [тут](http://md5x.ru/images/rfc/rfc1321.txt)  
Ну и в конце суммируем результаты вычислений:  
A = A + AA  
B = B + BB  
C = C + CC  
D = D + DD  
5) Output  
Выводя побайтово буффер ABCD начиная с A и заканчивая D получим наш хэш.

**Исходный код:**

**Функции интерфейса:**

private void f6\_buttonCountHashFromTextbox\_Click(object sender, EventArgs e)

{

byte[] tb = Encoding.Default.GetBytes(f6\_textBox\_original.Text);

f6\_TextBox\_Hash.Text = GetHashMD5(tb);

}

private void f6\_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);

f6\_TextBox\_Hash.Text = GetHashMD5(file);

}

}

**Хэш-функция:**

private string GetHashMD5(byte[] arr)

{

//Step 1

int bytesToAddAmount = (arr.Length + 1) % 64 > 56 ? (arr.Length + 1) % 64 - 56 : -(arr.Length + 1) % 64 + 56;

byte[] arrEx = new byte[arr.Length + bytesToAddAmount + 1];

arr.CopyTo(arrEx, 0);

arrEx[arr.Length] = 0x80; //Добавляем единицу

//Step 2

byte[] arrExtended = new byte[arrEx.Length + 8]; //Добавляем слоты для длины файла

arrEx.CopyTo(arrExtended, 0);

byte [] len = BitConverter.GetBytes(arr.Length \* 8);

len.CopyTo(arrExtended, arrEx.Length);

//Step 3

uint[] buf = new uint[4];

buf[0] = 0x67452301;

buf[1] = 0xEFCDAB89;

buf[2] = 0x98BADCFE;

buf[3] = 0x10325476;

//Step 4

//Функции funF, funG, funH, funI определены

uint[] T = new uint[64];

for (int i = 1; i <= 64; i++)

{

T[i - 1] = (uint)(0x100000000 \* Math.Abs(Math.Sin(i)));

}

int[] shiftValues = { 7, 12, 17, 22, 5, 9, 14, 20, 4, 11, 16, 23, 6, 10, 15, 21 }; //константы для сдвигов

for (int circle = 0; circle < arrExtended.Length / 64; circle++)

{

//HMD5

uint[] tmp = new uint[4];

buf.CopyTo(tmp, 0);

byte[] piece = new byte[64];

Array.Copy(arrExtended, 64 \* circle, piece, 0, 64);

for (int i = 0; i < 16; i++)

{

var ff = piece.Skip(4 \* i).Take(4).ToArray();

tmp[0] += funF(tmp[1], tmp[2], tmp[3]) + T[i] + BitConverter.ToUInt32(piece.Skip(4 \* i).Take(4).ToArray(), 0);

tmp[0] = BitShift(tmp[0], shiftValues[i % 4]);

tmp[0] += tmp[1];

tmp = Shift(tmp);

}

for (int i = 0; i < 16; i++)

{

int g = (5 \* i + 1) % 16;

tmp[0] += funG(tmp[1], tmp[2], tmp[3]) + T[16 + i] + BitConverter.ToUInt32(piece.Skip(4 \* g).Take(4).ToArray(), 0);

tmp[0] = BitShift(tmp[0], shiftValues[4 + (i % 4)]);

tmp[0] += tmp[1];

tmp = Shift(tmp);

}

for (int i = 0; i < 16; i++)

{

int g = (3 \* i + 5) % 16;

tmp[0] += funH(tmp[1], tmp[2], tmp[3]) + T[32 + i] + BitConverter.ToUInt32(piece.Skip(4 \* g).Take(4).ToArray(), 0);

tmp[0] = BitShift(tmp[0], shiftValues[8 + (i % 4)]);

tmp[0] += tmp[1];

tmp = Shift(tmp);

}

for (int i = 0; i < 16; i++)

{

int g = (7 \* i) % 16;

tmp[0] += funI(tmp[1], tmp[2], tmp[3]) + T[48 + i] + BitConverter.ToUInt32(piece.Skip(4 \* g).Take(4).ToArray(), 0);

tmp[0] = BitShift(tmp[0], shiftValues[12 + (i % 4)]);

tmp[0] += tmp[1];

tmp = Shift(tmp);

}

buf[0] += tmp[0];

buf[1] += tmp[1];

buf[2] += tmp[2];

buf[3] += tmp[3];

}

//Step 5

return OutHash(buf[0]) + OutHash(buf[1]) + OutHash(buf[2]) + OutHash(buf[3]);

}

**Вспомогательные функции:**

private uint funF(uint x, uint y, uint z)

{

return (x & y) | (~x & z);

}

private uint funG(uint x, uint y, uint z)

{

return (x & z) | (~z & y);

}

private uint funH(uint x, uint y, uint z)

{

return x ^ y ^ z;

}

private uint funI(uint x, uint y, uint z)

{

return y ^ (~z | x);

}

private uint[] Shift(uint[] arr)

{

var tmp = arr[0];

arr[0] = arr[3];

arr[3] = arr[2];

arr[2] = arr[1];

arr[1] = tmp;

return arr;

}

private uint BitShift(uint numb, int pos)

{

uint hi = (uint)(numb >> (32 - pos));

return (numb << pos) | hi;

}

private string OutHash(uint reg)

{

var bytes = BitConverter.GetBytes(reg);

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

if (tmp.Length != 8)

{

var k = 8 - tmp.Length;

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

{

tmp = "0" + tmp;

}

}

return new string(new char[] { tmp[6], tmp[7], tmp[4], tmp[5], tmp[2], tmp[3], tmp[0], tmp[1] });

}

**Результат работы программы:**

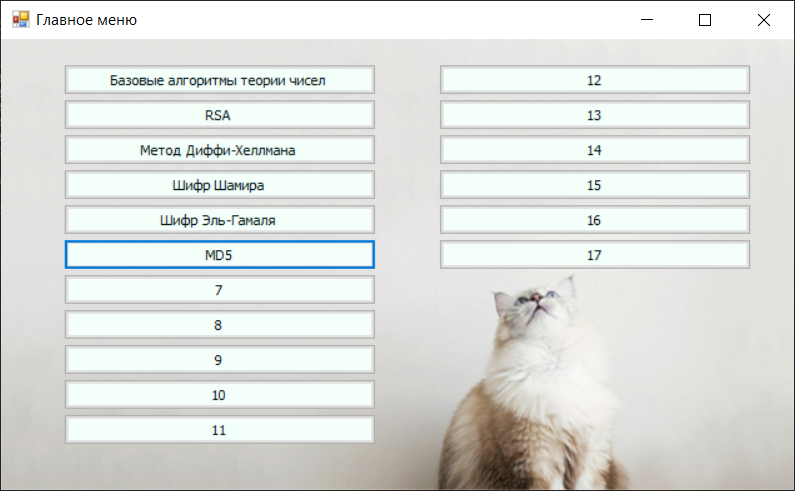


Рисунок 1. Главное меню программы.

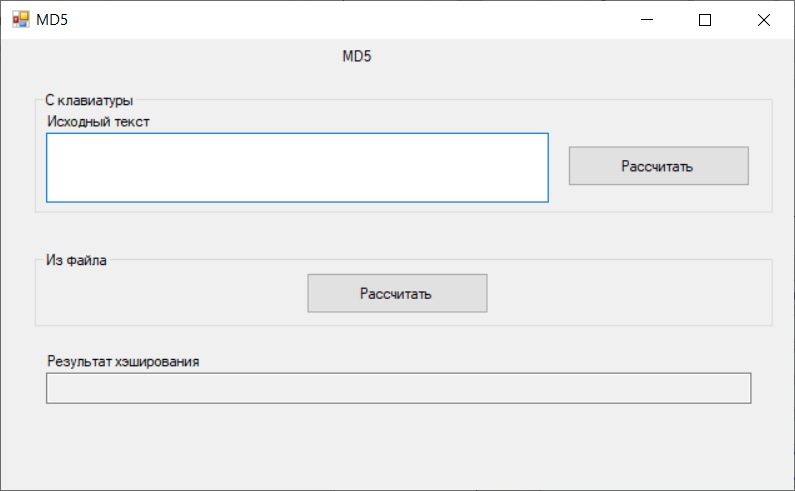


Рисунок 2. Окно хэш-функции MD5.

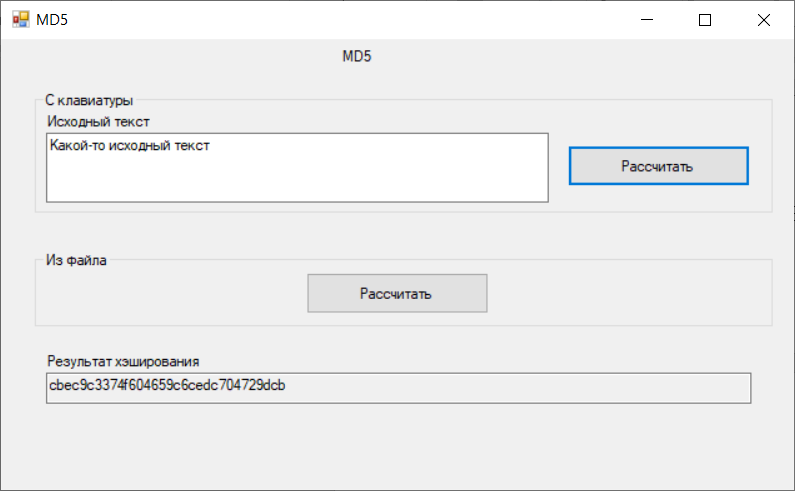


Рисунок 3. Расчет из текстового поля.

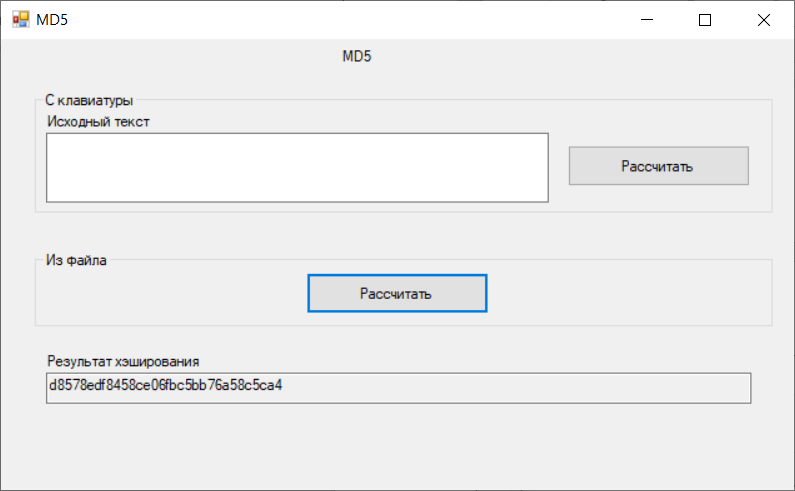


Рисунок 4. Расчет из файла.

**Вывод:**

В ходе выполнения лабораторной работы я ознакомилась с функцией хэширования MD5, реализовала на практике программу получения хэша на языке C#.