Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчёт

по лабораторной работе №5

**Компьютерная реализация хэш-функций на примере MD5.**

Выполнил:

Студент гр. 853504

Пресный В.И,

Проверил:

Олисейчик В.В.

Минск 2021

**Теоретические сведения**

**Хэш-функция MD5**

Алгоритм MD5 получает на входе сообщение произвольной длины и создает в качестве выхода *дайджест сообщения* длиной 128 бит.

Алгоритм состоит из следующих шагов:

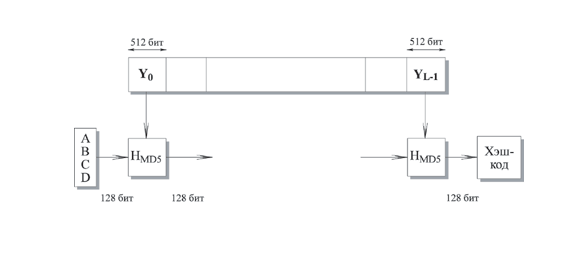


Рисунок 1 –  Логика выполнения MD5

***Шаг 1: добавление недостающих битов***

Сообщение дополняется таким образом, чтобы его длина стала равна 448 по модулю 512 (длина≡448 mod 512).

***Шаг 2: добавление длины***

64-битное представление длины исходного (до добавления) сообщения в битах присоединяется к результату первого шага. Если первоначальная длина больше, чем 264, то используются только последние 64 бита.

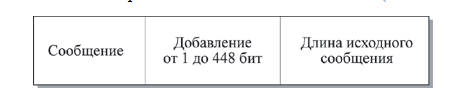


Рисунок 2 –  Структура расширенного сообщения

***Шаг 3: инициализация MD-буфера***

Используется 128-битный буфер для хранения промежуточных и окончательных результатов *хэш-функции*. Буфер может быть представлен как четыре 32-битных регистра (A, B, C, D). Эти регистры инициализируются следующими шестнадцатеричными числами:

А = 01234567; В = 89ABCDEF; C = FEDCBA98; D = 76543210

***Шаг 4: обработка последовательности 512-битных (16-словных) блоков***

Основой алгоритма является модуль, состоящий из четырех циклических обработок, обозначенный как HMD5. Четыре цикла имеют похожую структуру, но каждый цикл использует свою элементарную логическую функцию, обозначаемую fF, fG, fH и fI соответственно.

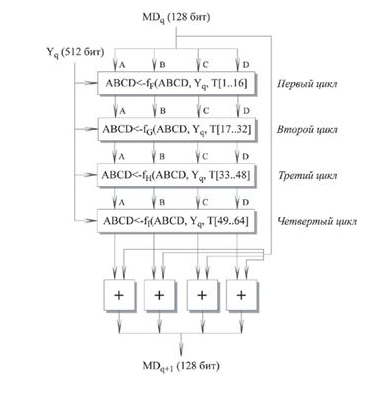


Рисунок 3 –  Обработка очередного 512-битного блока

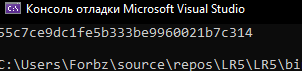
Каждый цикл принимает в качестве входа текущий 512-битный блок Yq, обрабатывающийся в данный момент, и 128-битное значение буфера ABCD, которое является промежуточным значением *дайджеста*, и изменяет содержимое этого буфера. Каждый цикл также использует четвертую часть 64-элементной таблицы T[1 ... 64], построенной на основе функции sin. i-ый элемент T, обозначаемый T[i], имеет значение, равное целой части от 232 \* abs (sin (i)), i задано в радианах. Так как abs (sin (i)) является числом между 0 и 1, каждый элемент Т является целым, которое может быть представлено 32 битами. Таблица обеспечивает "случайный" набор 32-битных значений, которые должны ликвидировать любую регулярность во входных данных.

Для получения MDq+1 выход четырех циклов складывается по модулю 232 с MDq. Сложение выполняется независимо для каждого из четырех слов в буфере.

***Шаг 5: выход***

После обработки всех *L* 512-битных блоков выходом *L*-ой стадии является 128-битный *дайджест сообщения*.

# Результаты выполнения

****

**Код приложения**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LR5

{

public class MD5

{

static int[] s = new int[64] {

7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,

5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20,

4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,

6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21

};

static uint[] K = new uint[64] {

0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee,

0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501,

0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be,

0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821,

0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa,

0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8,

0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed,

0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a,

0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c,

0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70,

0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05,

0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665,

0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039,

0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1,

0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1,

0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391

};

public static uint leftRotate(uint x, int c)

{

return (x << c) | (x >> (32 - c));

}

public static string Calculate(byte[] input)

{

uint a0 = 0x67452301; // A

uint b0 = 0xefcdab89; // B

uint c0 = 0x98badcfe; // C

uint d0 = 0x10325476; // D

var addLength = (64 - (input.Length % 64)) % 64;

addLength = addLength == 0 ? 64 : addLength;

byte[] length = BitConverter.GetBytes(input.Length \* 8);

var extendedMessage = new byte[input.Length + addLength];

Array.Copy(input, extendedMessage, input.Length);

extendedMessage[input.Length] = 128;

Array.Copy(length, 0, extendedMessage, extendedMessage.Length - 8, 4);

for (int i = 0; i < extendedMessage.Length / 64; ++i)

{

uint[] X = new uint[16];

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

X[j] = BitConverter.ToUInt32(extendedMessage, (i \* 64) + (j \* 4));

uint A = a0, B = b0, C = c0, D = d0, F = 0, g = 0;

for (uint k = 0; k < 64; ++k)

{

if (k <= 15)

{

F = (B & C) | (~B & D);

g = k;

}

else if (k >= 16 && k <= 31)

{

F = (D & B) | (~D & C);

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

}

else if (k >= 32 && k <= 47)

{

F = B ^ C ^ D;

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

}

else if (k >= 48)

{

F = C ^ (B | ~D);

g = (7 \* k) % 16;

}

var dtemp = D;

D = C;

C = B;

B = B + leftRotate(A + F + K[k] + X[g], s[k]);

A = dtemp;

}

a0 += A;

b0 += B;

c0 += C;

d0 += D;

}

return GetByteString(a0) + GetByteString(b0) + GetByteString(c0) + GetByteString(d0);

}

private static string GetByteString(uint x)

{

return string.Join("", BitConverter.GetBytes(x).Select(y => y.ToString("x2")));

}

}

}