Федеральное агентство связи

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра «Информационная безопасность»

Отчёт по лабораторной работе №3

по дисциплине

«Методы и средства защиты компьютерной информации»

Изучение алгоритмов хеширования

Выполнил:

студент группы БВТ1301

Редько Е.Ю.

Проверил:

ассистент кафедры ИБ

Барков В.В.

Москва 2016

**Цель работы**

Изучить принципы работы алгоритмов хеширования на примере реализации одного из алгоритмов.

**Задание**

Используя один из объектно-ориентированных языков программирования реализовать алгоритм хеширования, указанный в индивидуальном варианте.

Проект реализовать в виде библиотеки классов. При реализации использовать интерфейсы. Алгоритм должен быть универсальным с точки зрения данных для хеширования (строка, файл, сокет).

Для созданного проекта необходимо написать модульные тесты, показывающие корректность работы алгоритма.

**Описание алгоритма MD5**  
На вход алгоритма поступает входной поток данных, хеш которого необходимо найти. Длина сообщения может быть любой (в том числе нулевой). Запишем длину сообщения в L. Это число целое и неотрицательное. Кратность каким-либо числам необязательна. После поступления данных идёт процесс подготовки потока к вычислениям.

**Шаг 1. Выравнивание потока**

Сначала дописывают единичный бит в конец потока (байт 80h), затем необходимое число нулевых бит. Входные данные выравниваются так, чтобы их новый размер L ′ {\displaystyle L'} был сравним с 448 по модулю 512, ( L ′ = 512 × N + 448 {\displaystyle L'=512\times N+448} ). Выравнивание происходит, даже если длина уже сравнима с 448.

**Шаг 2. Добавление длины сообщения**

В конец сообщения дописывают 64-битное представление длины данных (количество бит в сообщении) до выравнивания. Сначала записывают младшие 4 байта, затем старшие. Если длина превосходит 2 64 − 1 {\displaystyle 2^{64}-1} , то дописывают только младшие биты (эквивалентно взятию по модулю 2 64 {\displaystyle 2^{64}} ). После этого длина потока станет кратной 512. Вычисления будут основываться на представлении этого потока данных в виде массива слов по 512 бит.

**Шаг 3. Инициализация буфера**

Для вычислений инициализируются 4 переменных размером по 32 бита и задаются начальные значения шестнадцатеричными числами (порядок байтов [little-endian](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D0%B1%D0%B0%D0%B9%D1%82%D0%BE%D0%B2), сначала младший байт):

А = 01 23 45 67; // 67452301h

В = 89 AB CD EF; // EFCDAB89h

С = FE DC BA 98; // 98BADCFEh

D = 76 54 32 10. // 10325476h

В этих переменных будут храниться результаты промежуточных вычислений. Начальное состояние ABCD называется инициализирующим вектором.

Определим ещё функции и константы, которые нам понадобятся для вычислений.

* Потребуются 4 функции для четырёх раундов. Введём функции от трёх параметров — слов, результатом также будет слово:

1-й раунд: FunF ⁡ ( X , Y , Z ) = ( X ∧ Y ) ∨ ( ¬ X ∧ Z ) {\displaystyle \operatorname {FunF} (X,Y,Z)=(X\wedge Y)\vee (\neg X\wedge Z)} ,

2-й раунд: FunG ⁡ ( X , Y , Z ) = ( X ∧ Z ) ∨ ( ¬ Z ∧ Y ) {\displaystyle \operatorname {FunG} (X,Y,Z)=(X\wedge Z)\vee (\neg Z\wedge Y)} ,

3-й раунд: FunH ⁡ ( X , Y , Z ) = X ⊕ Y ⊕ Z {\displaystyle \operatorname {FunH} (X,Y,Z)=X\oplus Y\oplus Z} ,

4-й раунд: FunI ⁡ ( X , Y , Z ) = Y ⊕ ( ¬ Z ∨ X ) {\displaystyle \operatorname {FunI} (X,Y,Z)=Y\oplus (\neg {Z}\vee X)} ,

* Определим таблицу констант T [ 1 … 64 ] {\displaystyle T[1\ldots 64]}  — 64-элементная таблица данных, построенная следующим образом: T [ n ] = int ⁡ ( 2 32 ⋅ | sin ⁡ n | ) {\displaystyle T[n]=\operatorname {int} (2^{32}\cdot |\sin n|)} .
* Каждый 512-битный блок проходит 4 этапа вычислений по 16 раундов. Для этого блок представляется в виде массива *X* из 16 слов по 32 бита. Все раунды однотипны.

**Шаг 4. Вычисление в цикле**

Заносим в блок данных элемент *n* из массива 512-битных блоков. Сохраняются значения A, B, C и D, оставшиеся после операций над предыдущими блоками (или их начальные значения, если блок первый).

AA = A

BB = B

CC = C

DD = D

**Этап 1**

// [abcd k s i] a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s).

[ABCD 0 7 1][DABC 1 12 2][CDAB 2 17 3][BCDA 3 22 4]

[ABCD 4 7 5][DABC 5 12 6][CDAB 6 17 7][BCDA 7 22 8]

[ABCD 8 7 9][DABC 9 12 10][CDAB 10 17 11][BCDA 11 22 12]

[ABCD 12 7 13][DABC 13 12 14][CDAB 14 17 15][BCDA 15 22 16]

**Этап 2**

// [abcd k s i] a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s).

[ABCD 1 5 17][DABC 6 9 18][CDAB 11 14 19][BCDA 0 20 20]

[ABCD 5 5 21][DABC 10 9 22][CDAB 15 14 23][BCDA 4 20 24]

[ABCD 9 5 25][DABC 14 9 26][CDAB 3 14 27][BCDA 8 20 28]

[ABCD 13 5 29][DABC 2 9 30][CDAB 7 14 31][BCDA 12 20 32]

**Этап 3**

// [abcd k s i] a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s).

[ABCD 5 4 33][DABC 8 11 34][CDAB 11 16 35][BCDA 14 23 36]

[ABCD 1 4 37][DABC 4 11 38][CDAB 7 16 39][BCDA 10 23 40]

[ABCD 13 4 41][DABC 0 11 42][CDAB 3 16 43][BCDA 6 23 44]

[ABCD 9 4 45][DABC 12 11 46][CDAB 15 16 47][BCDA 2 23 48]

**Этап 4**

// [abcd k s i] a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s).

[ABCD 0 6 49][DABC 7 10 50][CDAB 14 15 51][BCDA 5 21 52]

[ABCD 12 6 53][DABC 3 10 54][CDAB 10 15 55][BCDA 1 21 56]

[ABCD 8 6 57][DABC 15 10 58][CDAB 6 15 59][BCDA 13 21 60]

[ABCD 4 6 61][DABC 11 10 62][CDAB 2 15 63][BCDA 9 21 64]

Суммируем с результатом предыдущего цикла:

A = AA + A

B = BB + B

C = CC + C

D = DD + D

После окончания цикла необходимо проверить, есть ли ещё блоки для вычислений. Если да, то переходим к следующему элементу массива (*n* + 1) и повторяем цикл.

**Шаг 5. Результат вычислений**

Результат вычислений находится в буфере ABCD, это и есть хеш. Если выводить побайтово, начиная с младшего байта A и закончив старшим байтом D, то мы получим MD5-хеш. 1, 0, 15, 34, 17, 18…

## Схема алгоритма хеширования MD5

Схема алгоритма хеширования MD5 представлена на рисунке 1.



Рисунок 1. Алгоритм хеширования.

**Код программы**

IMD5.cs (интерфейс):

namespace RedkoLib

{

public interface IMD5

{

void GetHash(Stream input, Stream output);

}

}

MD5.cs:

namespace RedkoLib

{

public class MD5 : IMD5

{

public enum MD5InitializerConstant : uint

{

A = 0x67452301,

B = 0xEFCDAB89,

C = 0x98BADCFE,

D = 0X10325476

}

protected readonly static uint[] T = new uint[64]

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

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

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

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

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

0xd62f105d,0x2441453,0xd8a1e681,0xe7d3fbc8,

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

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

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

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

0x289b7ec6,0xeaa127fa,0xd4ef3085,0x4881d05,

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

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

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

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

0xf7537e82,0xbd3af235,0x2ad7d2bb,0xeb86d391};

protected uint[] X = new uint[16];

protected Digest dgFingerPrint;

protected static byte[] m\_byteInput;

public void GetHash(Stream input, Stream output)

{

m\_byteInput = new byte[input.Length];

input.Position = 0;

int size = input.Read(m\_byteInput, 0, (int)input.Length);

string hash = CalculateMD5Value().ToString();

output.Write(Encoding.Default.GetBytes(hash), 0, Encoding.Default.GetBytes(hash).Length);

}

protected Digest CalculateMD5Value()

{

byte[] bMsg;

uint n;

Digest dg = new Digest();

bMsg = CreatePaddedBuffer();

n = (uint)(bMsg.Length \* 8) / 32;

for (uint i = 0; i < n / 16; i++)

{

CopyBlock(bMsg, i);

Transform(ref dg.A, ref dg.B, ref dg.C, ref dg.D);

}

return dg;

}

protected void TransF(ref uint a, uint b, uint c, uint d, uint k, ushort s, uint i)

{

a = b + RotateLeft((a + ((b & c) | (~(b) & d)) + X[k] + T[i - 1]), s);

}

protected void TransG(ref uint a, uint b, uint c, uint d, uint k, ushort s, uint i)

{

a = b + RotateLeft((a + ((b & d) | (c & ~d)) + X[k] + T[i - 1]), s);

}

protected void TransH(ref uint a, uint b, uint c, uint d, uint k, ushort s, uint i)

{

a = b + RotateLeft((a + (b ^ c ^ d) + X[k] + T[i - 1]), s);

}

protected void TransI(ref uint a, uint b, uint c, uint d, uint k, ushort s, uint i)

{

a = b + RotateLeft((a + (c ^ (b | ~d)) + X[k] + T[i - 1]), s);

}

protected void Transform(ref uint A, ref uint B, ref uint C, ref uint D)

{

uint AA, BB, CC, DD;

AA = A;

BB = B;

CC = C;

DD = D;

TransF(ref A, B, C, D, 0, 7, 1); TransF(ref D, A, B, C, 1, 12, 2); TransF(ref C, D, A, B, 2, 17, 3); TransF(ref B, C, D, A, 3, 22, 4);

TransF(ref A, B, C, D, 4, 7, 5); TransF(ref D, A, B, C, 5, 12, 6); TransF(ref C, D, A, B, 6, 17, 7); TransF(ref B, C, D, A, 7, 22, 8);

TransF(ref A, B, C, D, 8, 7, 9); TransF(ref D, A, B, C, 9, 12, 10); TransF(ref C, D, A, B, 10, 17, 11); TransF(ref B, C, D, A, 11, 22, 12);

TransF(ref A, B, C, D, 12, 7, 13); TransF(ref D, A, B, C, 13, 12, 14); TransF(ref C, D, A, B, 14, 17, 15); TransF(ref B, C, D, A, 15, 22, 16);

TransG(ref A, B, C, D, 1, 5, 17); TransG(ref D, A, B, C, 6, 9, 18); TransG(ref C, D, A, B, 11, 14, 19); TransG(ref B, C, D, A, 0, 20, 20);

TransG(ref A, B, C, D, 5, 5, 21); TransG(ref D, A, B, C, 10, 9, 22); TransG(ref C, D, A, B, 15, 14, 23); TransG(ref B, C, D, A, 4, 20, 24);

TransG(ref A, B, C, D, 9, 5, 25); TransG(ref D, A, B, C, 14, 9, 26); TransG(ref C, D, A, B, 3, 14, 27); TransG(ref B, C, D, A, 8, 20, 28);

TransG(ref A, B, C, D, 13, 5, 29); TransG(ref D, A, B, C, 2, 9, 30); TransG(ref C, D, A, B, 7, 14, 31); TransG(ref B, C, D, A, 12, 20, 32);

TransH(ref A, B, C, D, 5, 4, 33); TransH(ref D, A, B, C, 8, 11, 34); TransH(ref C, D, A, B, 11, 16, 35); TransH(ref B, C, D, A, 14, 23, 36);

TransH(ref A, B, C, D, 1, 4, 37); TransH(ref D, A, B, C, 4, 11, 38); TransH(ref C, D, A, B, 7, 16, 39); TransH(ref B, C, D, A, 10, 23, 40);

TransH(ref A, B, C, D, 13, 4, 41); TransH(ref D, A, B, C, 0, 11, 42); TransH(ref C, D, A, B, 3, 16, 43); TransH(ref B, C, D, A, 6, 23, 44);

TransH(ref A, B, C, D, 9, 4, 45); TransH(ref D, A, B, C, 12, 11, 46); TransH(ref C, D, A, B, 15, 16, 47); TransH(ref B, C, D, A, 2, 23, 48);

TransI(ref A, B, C, D, 0, 6, 49); TransI(ref D, A, B, C, 7, 10, 50); TransI(ref C, D, A, B, 14, 15, 51); TransI(ref B, C, D, A, 5, 21, 52);

TransI(ref A, B, C, D, 12, 6, 53); TransI(ref D, A, B, C, 3, 10, 54); TransI(ref C, D, A, B, 10, 15, 55); TransI(ref B, C, D, A, 1, 21, 56);

TransI(ref A, B, C, D, 8, 6, 57); TransI(ref D, A, B, C, 15, 10, 58); TransI(ref C, D, A, B, 6, 15, 59); TransI(ref B, C, D, A, 13, 21, 60);

TransI(ref A, B, C, D, 4, 6, 61); TransI(ref D, A, B, C, 11, 10, 62); TransI(ref C, D, A, B, 2, 15, 63); TransI(ref B, C, D, A, 9, 21, 64);

A = A + AA;

B = B + BB;

C = C + CC;

D = D + DD;

}

public static uint RotateLeft(uint uiNumber, ushort shift)

{

return ((uiNumber >> 32 - shift) | (uiNumber << shift));

}

public static uint ReverseByte(uint uiNumber)

{

return (((uiNumber & 0x000000ff) << 24) |

(uiNumber >> 24) |

((uiNumber & 0x00ff0000) >> 8) |

((uiNumber & 0x0000ff00) << 8));

}

protected byte[] CreatePaddedBuffer()

{

uint pad;

byte[] bMsg;

ulong sizeMsg;

uint sizeMsgBuff;

int temp = (448 - ((m\_byteInput.Length \* 8) % 512));

pad = (uint)((temp + 512) % 512);

if (pad == 0)

pad = 512;

sizeMsgBuff = (uint)((m\_byteInput.Length) + (pad / 8) + 8);

sizeMsg = (ulong)m\_byteInput.Length \* 8;

bMsg = new byte[sizeMsgBuff];

for (int i = 0; i < m\_byteInput.Length; i++)

bMsg[i] = m\_byteInput[i];

bMsg[m\_byteInput.Length] |= 0x80;

for (int i = 8; i > 0; i--)

bMsg[sizeMsgBuff - i] = (byte)(sizeMsg >> ((8 - i) \* 8) & 0x00000000000000ff);

return bMsg;

}

protected void CopyBlock(byte[] bMsg, uint block)

{

block = block << 6;

for (uint j = 0; j < 61; j += 4)

{

X[j >> 2] = (((uint)bMsg[block + (j + 3)]) << 24) |

(((uint)bMsg[block + (j + 2)]) << 16) |

(((uint)bMsg[block + (j + 1)]) << 8) |

(((uint)bMsg[block + (j)]));

}

}

public class Digest

{

public uint A;

public uint B;

public uint C;

public uint D;

public Digest()

{

A = (uint)MD5InitializerConstant.A;

B = (uint)MD5InitializerConstant.B;

C = (uint)MD5InitializerConstant.C;

D = (uint)MD5InitializerConstant.D;

}

public override string ToString()

{

string st;

st = ReverseByte(A).ToString("X8") +

ReverseByte(B).ToString("X8") +

ReverseByte(C).ToString("X8") +

ReverseByte(D).ToString("X8");

return st;

}

}

}

}

TMD5.cs (тест):

namespace RedkoTest

{

[TestClass]

public class TMD5

{

[TestMethod]

public void EmptyHashEquals()

{

string sourceDataString = "";

MemoryStream input = new MemoryStream(Encoding.Default.GetBytes(sourceDataString));

MemoryStream output = new MemoryStream();

RedkoLib.MD5 md5 = new RedkoLib.MD5();

StringComparer comparer = StringComparer.OrdinalIgnoreCase;

md5.GetHash(input, output);

string myHash = Encoding.Default.GetString(output.ToArray()).TrimEnd('\0');

Assert.AreEqual(comparer.Compare(myHash, "d41d8cd98f00b204e9800998ecf8427e"), 0);

}

[TestMethod]

public void HashEquals()

{

string sourceDataString = "test";

RedkoLib.MD5 md5 = new RedkoLib.MD5();

MemoryStream input = new MemoryStream(Encoding.Default.GetBytes(sourceDataString));

MemoryStream output = new MemoryStream();

md5.GetHash(input, output);

string myHash = Encoding.Default.GetString(output.ToArray()).TrimEnd('\0');

System.Security.Cryptography.MD5 md5Hash = System.Security.Cryptography.MD5.Create();

byte[] data = md5Hash.ComputeHash(Encoding.UTF8.GetBytes(sourceDataString));

StringBuilder sBuilder = new StringBuilder();

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

{

sBuilder.Append(data[i].ToString("x2"));

}

string hash = sBuilder.ToString();

StringComparer comparer = StringComparer.OrdinalIgnoreCase;

Assert.AreEqual(comparer.Compare(myHash, hash), 0);

}

}

}

