Kpi-best

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМ. І.СІКОРСЬКОГО»

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

**Лабораторна робота №2**

з дисципліни «Безпека програм і даних»

на тему: «АЛГОРИТМИ ЗАХИСТУ ДАНИХ НА ОСНОВІ БУЛЕВИХ ПЕРЕТВОРЕНЬ»

Виконав:

студент 4-го курсу

факультету ІОТ

групи ІП-54

Макаренко Антон

Перевірив:

Марковський О.П.

Київ 2018

**Завдання:**

**Хеш-алгоритми** SHA-1

**Лістинг:**

**Program.cs:**

using System;

using System.Linq;

using System.Text;

namespace Lab2

{

class Program

{

static void Main(string[] args)

{

var str = "123";

var result = SHA1Algorithm.SHA1(str);

Console.WriteLine(string.Join("", result.Select(x => x.ToString("X"))));

}

}

}

**SHA1Algorithm.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Lab2

{

public static class SHA1Algorithm

{

private static readonly uint[] H = new uint[] {

0x67452301,

0xEFCDAB89,

0x98BADCFE,

0x10325476,

0xC3D2E1F0

};

private static readonly uint[] K = new uint[] {

0x5A827999,

0x6ED9EBA1,

0x8F1BBCDC,

0xCA62C1D6

};

private static uint F(uint j, uint x, uint y, uint z)

{

if (j < 20)

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

else if (j < 40)

return x ^ y ^ z;

else if (j < 60)

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

else if (j < 80)

return x ^ y ^ z;

else

return 0;

}

private static long extension(List<byte> message)

{

var bitlen = message.Count \* 8;

message.Add(0x80);

while ((message.Count \* 8) % 512 != 448)

{

message.Add(0);

}

return bitlen;

}

private static void AddingLength(long bitlen, List<byte> message)

{

var temp = message.Count;

message.AddRange(new byte[8]);

for (var i = message.Count - 1; i >= temp; i--)

{

message[i] = (byte)bitlen;

bitlen >>= 8;

}

}

private static uint[] MessageProcessing(long bitlen, List<byte> message)

{

uint[] h = new uint[5];

H.CopyTo(h,0);

uint[] W = new uint[80];

uint A, B, C, D, E, T;

byte[] M = new byte[64];

for (var i = 0; i < message.Count; i += 64)

{

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

{

M[k] = message[k + i];

}

for (var k = 0; k < 16; k++)

{

W[k] = ((uint)M[k \* 4]) << 24;

W[k] |= ((uint)M[k \* 4 + 1]) << 16;

W[k] |= ((uint)M[k \* 4 + 2]) << 8;

W[k] |= ((uint)M[k \* 4 + 3]);

}

for (var k = 16; k < 80; k++)

{

W[k] = RotateLeft((W[k - 3] ^ W[k - 8] ^ W[k - 14] ^ W[k - 16]), 1);

}

A = h[0];

B = h[1];

C = h[2];

D = h[3];

E = h[4];

for (uint j = 0; j < 80; j++)

{

T = RotateLeft(A, 5) + F(j, B, C, D) + E + W[j] + K[j / 20];

E = D;

D = C;

C = RotateLeft(B, 30);

B = A;

A = T;

}

h[0] += A; h[1] += B; h[2] += C; h[3] += D; h[4] += E;

}

return h;

}

private static uint RotateLeft(uint original, int bits)

{

return (original << bits) | (original >> (32 - bits));

}

public static uint[] SHA1(string message)

{

List<byte> byteMessage = Encoding.ASCII.GetBytes(message).ToList();

var bitlen = extension(byteMessage);

AddingLength(bitlen, byteMessage);

return MessageProcessing(bitlen, byteMessage);

}

}

}