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 №7**

|  |
| --- |
| Хэш-функция SHA-1 |
| *тема* |

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

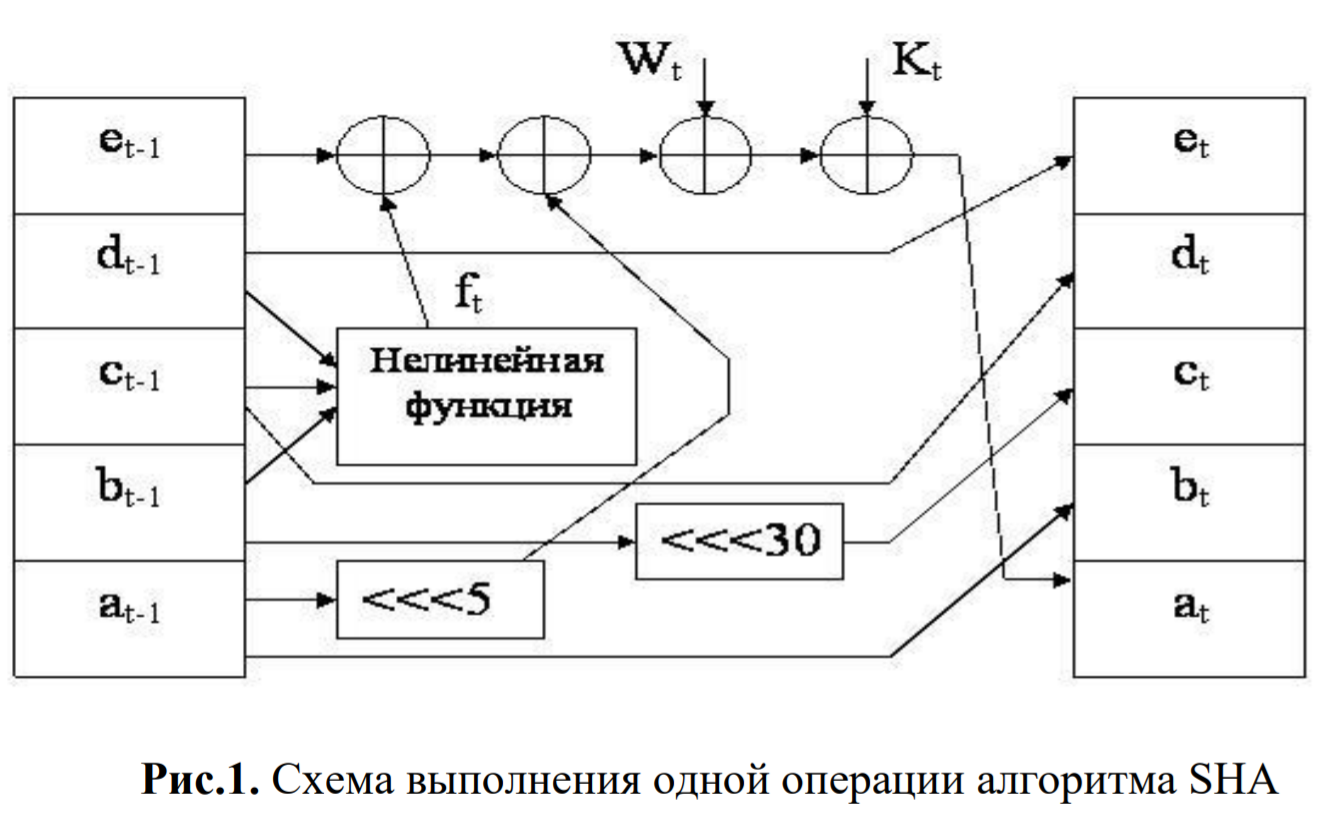
Красноярск 2019

**Задание:**

Изучить основные принципы построения и реализации алгоритма хэширования SHA-1.

**Теория:**

Алгоритм безопасного хэширования SНА (Secure Hash Algorithm) разработан НИСТ и АНБ США в рамках стандарта безопасного хэширования SHS (Secure Hash Standard) в 1992 г. Алгоритм хэширования SНА предназначен для использования совместно с алгоритмом цифровой подписи DSА. Рассмотрим подробнее работу алгоритма хэширования SНА.



**Исходный код:**

**Функции интерфейса:**

private void f7\_buttonCountHashFromTextbox\_Click(object sender, EventArgs e)

{

byte[] tb = Encoding.Default.GetBytes(f7\_textBox\_original.Text);

f7\_TextBox\_Hash.Text = GetHashSHA1(tb);

}

private void f7\_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);

f7\_TextBox\_Hash.Text = GetHashSHA1(file);

}

}

**Хэш-функция:**

private string GetHashSHA1(byte[] arr)

{

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

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

arrEx.CopyTo(arrExtended, 0);

UInt64 length = Convert.ToUInt64(arr.Length);

byte[] len = BitConverter.GetBytes(swapEndianness64(length \* 8));

len.CopyTo(arrExtended, arrEx.Length);

uint[] buf = new uint[5];

buf[0] = 0x67452301;

buf[1] = 0xEFCDAB89;

buf[2] = 0x98BADCFE;

buf[3] = 0x10325476;

buf[4] = 0xC3D2E1F0;

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

{

uint[] tmp = new uint[5];

buf.CopyTo(tmp, 0);

byte[] piece = new byte[64];

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

uint[] w = new uint[80];

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

{

w[i] = swapEndianness(BitConverter.ToUInt32(piece, i \* 4));

}

for (int i = 16; i < 80; i++)

{

w[i] = BitShift(w[i - 3] ^ w[i - 8] ^ w[i - 14] ^ w[i - 16], 1);

}

uint k;

for (int i = 0; i < 80; i++)

{

uint temp = 0;

switch (i / 20)

{

case 0:

k = 0x5A827999;

temp = BitShift(tmp[0], 5) + fun1(tmp[1], tmp[2], tmp[3]) + tmp[4] + k + w[i];

break;

case 1:

k = 0x6ED9EBA1;

temp = BitShift(tmp[0], 5) + fun2(tmp[1], tmp[2], tmp[3]) + tmp[4] + k + w[i];

break;

case 2:

k = 0x8F1BBCDC;

temp = BitShift(tmp[0], 5) + fun3(tmp[1], tmp[2], tmp[3]) + tmp[4] + k + w[i];

break;

case 3:

k = 0xCA62C1D6;

temp = BitShift(tmp[0], 5) + fun2(tmp[1], tmp[2], tmp[3]) + tmp[4] + k + w[i];

break;

}

tmp[4] = tmp[3];

tmp[3] = tmp[2];

tmp[2] = BitShift(tmp[1], 30);

tmp[1] = tmp[0];

tmp[0] = temp;

}

buf[0] += tmp[0];

buf[1] += tmp[1];

buf[2] += tmp[2];

buf[3] += tmp[3];

buf[4] += tmp[4];

}

byte[] result = new byte[20];

Array.Copy(BitConverter.GetBytes((buf[4])), 0, result, 0, 4);

Array.Copy(BitConverter.GetBytes((buf[3])), 0, result, 4, 4);

Array.Copy(BitConverter.GetBytes((buf[2])), 0, result, 8, 4);

Array.Copy(BitConverter.GetBytes((buf[1])), 0, result, 12, 4);

Array.Copy(BitConverter.GetBytes((buf[0])), 0, result, 16, 4);

string str = "";

foreach (var bt in result)

{

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

if (tmp.Length < 2)

tmp = "0" + tmp;

str = tmp + str;

}

return str;

}

**Вспомогательные функции:**

uint swapEndianness(uint x)

{

return ((x & 0x000000ff) << 24) + // First byte

((x & 0x0000ff00) << 8) + // Second byte

((x & 0x00ff0000) >> 8) + // Third byte

((x & 0xff000000) >> 24); // Fourth byte

}

UInt64 swapEndianness64(UInt64 x)

{

return ((x & 0x00000000000000ff) << 56) | // First byte

((x & 0x000000000000ff00) << 40) | // Second byte

((x & 0x0000000000ff0000) << 24) | // Third byte

((x & 0x00000000ff000000) << 8) | // Fourth byte

((x & 0x000000ff00000000) >> 8) | // First byte

((x & 0x0000ff0000000000) >> 24) | // Second byte

((x & 0x00ff000000000000) >> 40) | // Third byte

((x & 0xff00000000000000) >> 56);

}

private uint fun1(uint x, uint y, uint z)

{

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

}

private uint fun2(uint x, uint y, uint z)

{

return x ^ y ^ z;

}

private uint fun3(uint x, uint y, uint z)

{

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

}

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:**

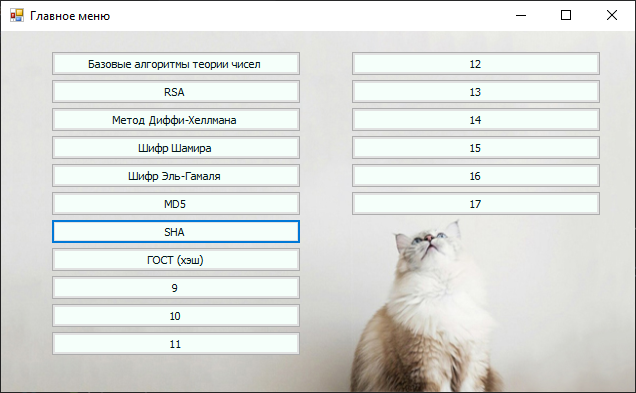


Рисунок 1. Главное меню программы.

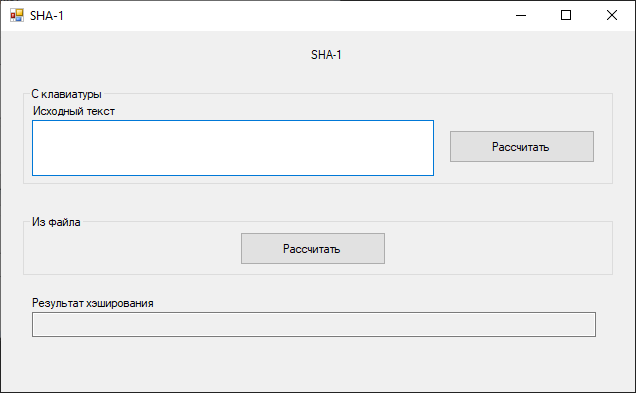


Рисунок 2. Окно хэш-функции SHA-1.

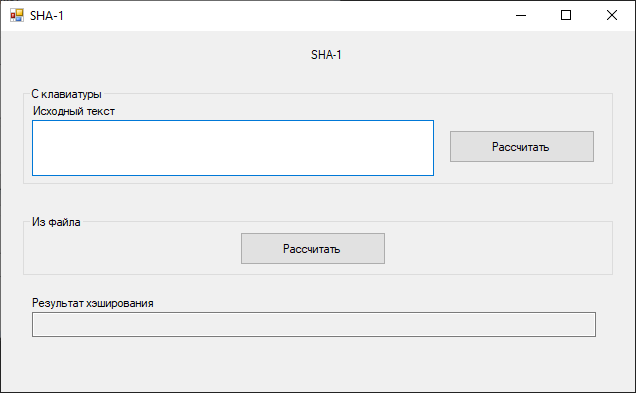


Рисунок 3. Расчет из текстового поля.

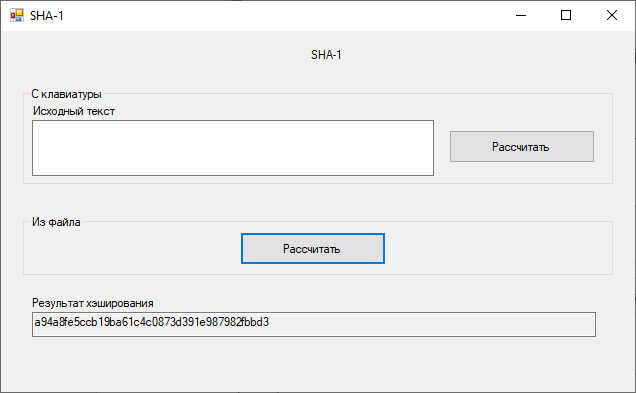


Рисунок 4. Расчет из файла.

**Вывод:**

В ходе выполнения лабораторной работы я ознакомилась с функцией хэширования SHA-1, реализовала на практике программу получения хэша на языке C#.