­Министерство высшего и профессионального образования РФ

Государственное образовательное учреждение

высшего профессионального образования

Ульяновский государственный технический университет

Кафедра «Вычислительная техника»

Дисциплина "МиСЗКИ"

Лабораторная работа №1

Хеширование паролей. Генераторы случайных чисел

Выполнил:

студент группы БЭВМд-41

Ключников Дмитрий

Проверил:

Мартынов А. И.

Ульяновск 2012

***Задание:***

1. Изучить алгоритмы хеширования паролей.
2. Изучить известные алгоритмы работы генераторов случайных чисел.
3. Реализовать свой упрощенный вариант алгоритма хеширования пароля согласно варианту
4. Реализовать свой алгоритм генератора случайных чисел согласно варианту.
5. Проанализировать выходную последовательность, выдаваемую генератором при различных параметрах.

***Дополнительные требования к лабораторной работе:***

1. Паролем может быть любая последовательность символов (русских и английских, цифр, знаков препинания и т.д.).
2. Текст программы оформляется прилично (удобочитаемо, с описанием ВСЕХ функций, переменных и критических мест).
3. В процессе работы программа ОБЯЗАТЕЛЬНО выдает информацию о состоянии процесса генерации.
4. Интерфейс программы может быть произвольным, но удобным и понятным (разрешается использование библиотек VCL)
5. Среда разработки и язык программирования могут быть произвольными.

***Задание по варианту:***

Тип генератора: Конгруэнтный генератор

Диапазон значений: 0-255

**Генератор**

***Конгруэнтный генератор***

Подобные генераторы используют рекуррентную последовательность

***I(n+1)=(a\*I(n)+c)(mod m)***

Число **a** называется мультипликатором, число **c** инкрементом, а число **m** - модулем.

***Генератор Парка-Миллера***

Самая простая последовательность, которую можно предложить для реализации генератора равномерного распределения:

***I(j+1)=a\*I(j)(mod m)***

при соответствующем выборе констант. Константы были предложены Park и Miller: **a=75=16807, m=231-1=2147483647**

**Хеширование пароля**

**Ч**то отличает эту функцию? Правильно, бросается в глаза таблица замены (подстановки). Зачем она в таком простом алгоритме? Прежде всего затем, что таблица замены - это сложная функция, которую просто описали несколько иным способом. При этом такую функцию очень легко реализовать как программно, так и аппаратно, скорость запроса к таблице высока в обоих случаях. У функции есть параметр - число итераций. Статистически, лучшим параметром будет одна итерация для большинства текстов. Хотя для оригинальной функции оптимальными будут три итерации. Сама таблица подстановки идентична таковой в криптоалгоритме Skipjack. Результаты не лучшие, но конструкция имеет право на жизнь.

Алгоритм полностью 32-разрядный, скорость его не уменьшилась. Такую манимуляцию можно провести со всеми алгоритмами семейства, в которых структура - это две подфункции. Достоинства алгоритма - высокая скорость, равная алгоритму с 32-разрядным хэшем, простота реализации и очень хорошие показатели по числу коллизий. Функция, безусловно, лидер в нашем забеге, хотя участвует вне конкурса - ведь хэш в два раза больше, чем у других участников.   
  
Для увеличения лавинного эффекта можно использовать простую модификацию с дополнительным преобразованием финального 64-битного значения.

**Код программы**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace CommonLibrary

{

/// <summary>

/// Линейный конгруэнтный генератор псевдослучайных чисел

/// </summary>

public class CongruentialGenerator

{

/// <summary>

/// Последнее значение

/// </summary>

private ulong lastValue;

/// <summary>

/// Мультипликатор

/// </summary>

public ulong a = 6364136223846793005;

/// <summary>

/// Инкремент

/// </summary>

public ulong c = 1442695040888963407;

/// <summary>

/// Модуль

/// </summary>

public double m = Math.Pow(2, 32);

/// <summary>

/// Линейный конгруэнтный генератор псевдослучайных чисел

/// </summary>

/// <param name="firstValue">Начальное значение</param>

public CongruentialGenerator(ulong firstValue)

{

this.lastValue = firstValue;

}

/// <summary>

/// Получение очередного псевдослучайного числа

/// </summary>

/// <returns>Псевдослучайное число</returns>

public ulong Next()

{

this.lastValue = (ulong)((this.a \* this.lastValue + this.c) % this.m);

return this.lastValue;

}

/// <summary>

/// Получение очередного псевдослучайного числа (double)

/// </summary>

/// <returns>Псевдослучайное число (double)</returns>

public double NextDouble()

{

var res = ((this.a \* this.lastValue + this.c) % this.m);

this.lastValue = (ulong)res;

return res;

}

/// <summary>

/// Получение очередного псевдослучайного числа

/// </summary>

/// <param name="maxValue">Максимальное значение результата</param>

/// <returns>Псевдослучайное число</returns>

public ulong Next(ulong maxValue)

{

return (ulong)(maxValue \* (Next() / this.m));

}

/// <summary>

/// Получение очередного псевдослучайного числа

/// </summary>

/// <param name="minValue">Минимальное значение результата</param>

/// <param name="maxValue">Максимальное значение результата</param>

/// <returns>Псевдослучайное число</returns>

public ulong Next(ulong minValue, ulong maxValue)

{

if (minValue > maxValue)

{

throw new ArgumentException("\"minValue\" can not be greater than \"maxValue\"");

}

return (ulong)(((maxValue - minValue) \* (Next() / this.m)) + minValue);

}

/// <summary>

/// Получение очередного псевдослучайного числа (double)

/// </summary>

/// <param name="minValue">Минимальное значение результата (double)</param>

/// <param name="maxValue">Максимальное значение результата (double)</param>

/// <returns>Псевдослучайное число (double)</returns>

public double Next(double minValue, double maxValue)

{

if (minValue > maxValue)

{

throw new ArgumentException("\"minValue\" can not be greater than \"maxValue\"");

}

if (minValue < 0 || maxValue < 0)

{

throw new ArgumentException("Value can not be less than zero (\"minValue\" or \"maxValue\")");

}

return (((maxValue - minValue) \* (NextDouble() / this.m)) + minValue);

}

/// <summary>

/// Получение очередного псевдослучайного числа (double)

/// </summary>

/// <param name="max">Максимальное значение результата (double)</param>

/// <returns>Псевдослучайное число (double)</returns>

public double Next(double max)

{

if (max < 0)

{

throw new ArgumentException("\"maxValue\" can not be less than zero.");

}

return (max \* (NextDouble() / this.m));

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace CommonLibrary

{

public class MaHash8v64

{

private static uint[] sTable =

{

0xa3,0xd7,0x09,0x83,0xf8,0x48,0xf6,0xf4,0xb3,0x21,0x15,0x78,0x99,0xb1,0xaf,0xf9,

0xe7,0x2d,0x4d,0x8a,0xce,0x4c,0xca,0x2e,0x52,0x95,0xd9,0x1e,0x4e,0x38,0x44,0x28,

0x0a,0xdf,0x02,0xa0,0x17,0xf1,0x60,0x68,0x12,0xb7,0x7a,0xc3,0xe9,0xfa,0x3d,0x53,

0x96,0x84,0x6b,0xba,0xf2,0x63,0x9a,0x19,0x7c,0xae,0xe5,0xf5,0xf7,0x16,0x6a,0xa2,

0x39,0xb6,0x7b,0x0f,0xc1,0x93,0x81,0x1b,0xee,0xb4,0x1a,0xea,0xd0,0x91,0x2f,0xb8,

0x55,0xb9,0xda,0x85,0x3f,0x41,0xbf,0xe0,0x5a,0x58,0x80,0x5f,0x66,0x0b,0xd8,0x90,

0x35,0xd5,0xc0,0xa7,0x33,0x06,0x65,0x69,0x45,0x00,0x94,0x56,0x6d,0x98,0x9b,0x76,

0x97,0xfc,0xb2,0xc2,0xb0,0xfe,0xdb,0x20,0xe1,0xeb,0xd6,0xe4,0xdd,0x47,0x4a,0x1d,

0x42,0xed,0x9e,0x6e,0x49,0x3c,0xcd,0x43,0x27,0xd2,0x07,0xd4,0xde,0xc7,0x67,0x18,

0x89,0xcb,0x30,0x1f,0x8d,0xc6,0x8f,0xaa,0xc8,0x74,0xdc,0xc9,0x5d,0x5c,0x31,0xa4,

0x70,0x88,0x61,0x2c,0x9f,0x0d,0x2b,0x87,0x50,0x82,0x54,0x64,0x26,0x7d,0x03,0x40,

0x34,0x4b,0x1c,0x73,0xd1,0xc4,0xfd,0x3b,0xcc,0xfb,0x7f,0xab,0xe6,0x3e,0x5b,0xa5,

0xad,0x04,0x23,0x9c,0x14,0x51,0x22,0xf0,0x29,0x79,0x71,0x7e,0xff,0x8c,0x0e,0xe2,

0x0c,0xef,0xbc,0x72,0x75,0x6f,0x37,0xa1,0xec,0xd3,0x8e,0x62,0x8b,0x86,0x10,0xe8,

0x08,0x77,0x11,0xbe,0x92,0x4f,0x24,0xc5,0x32,0x36,0x9d,0xcf,0xf3,0xa6,0xbb,0xac,

0x5e,0x6c,0xa9,0x13,0x57,0x25,0xb5,0xe3,0xbd,0xa8,0x3a,0x01,0x05,0x59,0x2a,0x46

};

private static uint LROT14(uint x)

{

return (((x) << 14) | ((x) >> 18));

}

private static uint RROT14(uint x)

{

return (((x) << 18) | ((x) >> 14));

}

private static uint LROT64(uint x)

{

return (((x) << 29) | ((x) >> 34));

}

public static uint GetHashCode(string str)

{

uint sh1, sh2, hash1 = (uint)str.Length, hash2 = (uint)str.Length;

uint digest;

for (int i = 0; i != str.Length; i++)

{

hash1 += sTable[(str[i]) & 255];

hash1 = LROT14(hash1 + ((hash1 << 6) ^ (hash1 >> 11)));

hash2 += sTable[(str[i]) & 255];

hash2 = RROT14(hash2 + ((hash2 << 6) ^ (hash2 >> 11)));

sh1 = hash1;

sh2 = hash2;

hash1 = ((sh1 >> 16) & 0xffff) | ((sh2 & 0xffff) << 16);

hash2 = ((sh2 >> 16) & 0xffff) | ((sh1 & 0xffff) << 16);

}

digest = (((hash1) << 32) | (hash2));

digest ^= LROT64(digest + ((digest << 13) ^ (digest >> 23)));

digest = ((hash2) << 32) | (hash1);

return digest;

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using CommonLibrary;

namespace Lab1

{

public partial class MainForm : Form

{

private CongruentialGenerator generator = null;

public MainForm()

{

InitializeComponent();

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

if (passTB.Text == "")

{

hashTB.Text = "";

return;

}

var hash = MaHash8v64.GetHashCode(passTB.Text);

hashTB.Text = hash + "";

}

private void genB\_Click(object sender, EventArgs e)

{

if (hashTB.Text == "")

{

MessageBox.Show("Задайте пароль!");

return;

}

if (generator == null)

generator = new CongruentialGenerator(uint.Parse(hashTB.Text));

try

{

ulong res = 0;

if (minValueTB.Text != "")

{

if (maxValueTB.Text == "")

{

MessageBox.Show("Задайте максимальное значение");

return;

}

res = generator.Next(ulong.Parse(minValueTB.Text), ulong.Parse(maxValueTB.Text));

}

else

res = maxValueTB.Text != "" ? generator.Next(ulong.Parse(maxValueTB.Text)) : generator.Next();

randomTB.Text = res + "";

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void onlyDigit\_KeyPress(object sender, KeyPressEventArgs e)

{

if (Char.IsDigit(e.KeyChar)) return;

if (e.KeyChar != (char)Keys.Back && e.KeyChar != '-')

e.Handled = true;

else if (e.KeyChar == '-')

if ((sender as TextBox).Text.Length != 0)

e.Handled = true;

}

}

}