|  |  |
| --- | --- |
| Министерство образования Республики Беларусь  Учреждение образования «Полоцкий государственный университет» | |
|  | Факультет информационных технологий  Кафедра технологий программирования |
| Лабораторная работа №5 по курсу «Теория информации»  «Симметричные криптосистемы. Алгоритм IDEA» | |
| Выполнил | Студент гр. 21-ИТ-1  Макеёнок Д.И. |
| Проверила | Васильева Д.М. |
| Полоцк, 2023г. | |

**Ход работы**

Алгоритм IDEA (International Data Encryption Algorithm) относится к классу симметричных шифраторов. Данный алгоритм был разработан в 1990 г. в качестве альтернативы алгоритму DES (Data Encryption Standard). В основе алгоритма лежит идея смешанного преобразования, которое случайным образом равномерно распределяет исходный текст по всему пространству шифротекста.

**Задание:**

1. Разработать программное средство, выполняющее шифрование по алгоритму IDEA заданного файла с произвольным содержимым. Ключ шифрования подаётся в виде бинарного файла длиной 16 байт.

2. Разработать программное средство, выполняющее дешифрование заданного файла, зашифрованного по алгоритму IDEA. Ключ шифрования подается в виде бинарного файла длиной 16 байт.

Листинг 1 - реализация Program.cs:

using System.Text;

using lab5;

var inputText = File.ReadAllText(@"..\source.txt");

var key = Idea.CreateRandomKey(16);

var ideaCrypt = new Idea(key, true);

var encoded = ideaCrypt.Crypt(inputText.Select(x => (byte)x).ToArray());

Console.WriteLine(Encoding.ASCII.GetString(encoded.ToArray()));

Console.WriteLine("--------------");

var ideaDecrypt = new Idea(key, false);

var decoded = ideaDecrypt.Crypt(encoded);

Console.WriteLine(Encoding.ASCII.GetString(decoded.ToArray()));

Листинг 2 - реализация Idea.cs:

using System.Security.Cryptography;

using System.Text;

namespace lab5

{

public class Idea

{

public static int Rounds = 8;

public int[] SubKey;

public static byte[] GetHash(string inputString)

{

using var algorithm = SHA256.Create();

return algorithm.ComputeHash(Encoding.UTF8.GetBytes(inputString));

}

public static string GetHashString(string inputString)

{

var sb = new StringBuilder();

foreach (var b in GetHash(inputString))

sb.Append(b.ToString("X2"));

return sb.ToString();

}

public static string CreateRandomKey(int maxLength = 100)

{

var r = new Random();

return GetHashString(Enumerable.Range(0, maxLength).Aggregate(string.Empty, (current, \_) => current + (char)r.Next(68, 90)))[..maxLength];

}

public Idea(string charKeyLength, bool encrypt)

{

var key = GenerateUserKeyFromCharKey(charKeyLength);

var tempSubKey = ExpandUserKey(key);

SubKey = encrypt ? tempSubKey : InvertSubKey(tempSubKey);

}

public byte[] Crypt(byte[] data, int dataPos = 0)

{

for (var i = 0; i < data.Length - 8; i += 8)

{

var x0 = ((data[dataPos + 0] & 0xFF) << 8) | (data[dataPos + 1] & 0xFF);

var x1 = ((data[dataPos + 2] & 0xFF) << 8) | (data[dataPos + 3] & 0xFF);

var x2 = ((data[dataPos + 4] & 0xFF) << 8) | (data[dataPos + 5] & 0xFF);

var x3 = ((data[dataPos + 6] & 0xFF) << 8) | (data[dataPos + 7] & 0xFF);

var p = 0;

for (var round = 0; round < Rounds; round++)

{

var y0 = Mul(x0, SubKey[p++]);

var y1 = Add(x1, SubKey[p++]);

var y2 = Add(x2, SubKey[p++]);

var y3 = Mul(x3, SubKey[p++]);

var t0 = Mul(y0 ^ y2, SubKey[p++]);

var t1 = Add(y1 ^ y3, t0);

var t2 = Mul(t1, SubKey[p++]);

var t3 = Add(t0, t2);

x0 = y0 ^ t2;

x1 = y2 ^ t2;

x2 = y1 ^ t3;

x3 = y3 ^ t3;

}

var r0 = Mul(x0, SubKey[p++]);

var r1 = Add(x2, SubKey[p++]);

var r2 = Add(x1, SubKey[p++]);

var r3 = Mul(x3, SubKey[p++]);

data[dataPos + 0] = (byte)(r0 >> 8);

data[dataPos + 1] = (byte)r0;

data[dataPos + 2] = (byte)(r1 >> 8);

data[dataPos + 3] = (byte)r1;

data[dataPos + 4] = (byte)(r2 >> 8);

data[dataPos + 5] = (byte)r2;

data[dataPos + 6] = (byte)(r3 >> 8);

data[dataPos + 7] = (byte)r3;

dataPos += 8;

dataPos = Math.Clamp(dataPos, 0, data.Length - 8);

}

return data;

}

private static int[] ExpandUserKey(byte[] userKey)

{

var key = new int[Rounds \* 6 + 4];

for (var i = 0; i < userKey.Length / 2; i++)

key[i] = ((userKey[2 \* i] & 0xFF) << 8) | (userKey[2 \* i + 1] & 0xFF);

for (var i = userKey.Length / 2; i < key.Length; i++)

key[i] = ((key[(i + 1) % 8 != 0 ? i - 7 : i - 15] << 9) | (key[(i + 2) % 8 < 2 ? i - 14 : i - 6] >> 7)) & 0xFFFF;

return key;

}

private static int[] InvertSubKey(int[] key)

{

var invKey = new int[key.Length];

var p = 0;

var i = Rounds \* 6;

invKey[i + 0] = MulInv(key[p++]);

invKey[i + 1] = AddInv(key[p++]);

invKey[i + 2] = AddInv(key[p++]);

invKey[i + 3] = MulInv(key[p++]);

for (var r = Rounds - 1; r >= 0; r--)

{

i = r \* 6;

var m = r > 0 ? 2 : 1;

var n = r > 0 ? 1 : 2;

invKey[i + 4] = key[p++];

invKey[i + 5] = key[p++];

invKey[i + 0] = MulInv(key[p++]);

invKey[i + m] = AddInv(key[p++]);

invKey[i + n] = AddInv(key[p++]);

invKey[i + 3] = MulInv(key[p++]);

}

return invKey;

}

private static int Add(int a, int b)

{

return (a + b) & 0xFFFF;

}

private static int Mul(int a, int b)

{

long r = (long)a \* b;

if (r != 0)

{

return (int)(r % 0x10001) & 0xFFFF;

}

else

{

return (1 - a - b) & 0xFFFF;

}

}

private static int AddInv(int x)

{

return (0x10000 - x) & 0xFFFF;

}

private static int MulInv(int x)

{

if (x <= 1)

return x;

var y = 0x10001;

var t0 = 1;

var t1 = 0;

while (true)

{

t1 += y / x \* t0;

y %= x;

if (y == 1)

return 0x10001 - t1;

t0 += x / y \* t1;

x %= y;

if (x == 1)

return t0;

}

}

private static byte[] GenerateUserKeyFromCharKey(string charKey)

{

var nofChar = 0x7E - 0x21 + 1;

var a = new int[8];

foreach (var t in charKey)

{

var c = (int)t;

for (var i = a.Length - 1; i >= 0; i--)

{

c += a[i] \* nofChar;

a[i] = c & 0xFFFF;

c >>= 16;

}

}

var key = new byte[16];

for (var i = 0; i < 8; i++)

{

key[i \* 2] = (byte)(a[i] >> 8);

key[i \* 2 + 1] = (byte)a[i];

}

return key;

}

}

}

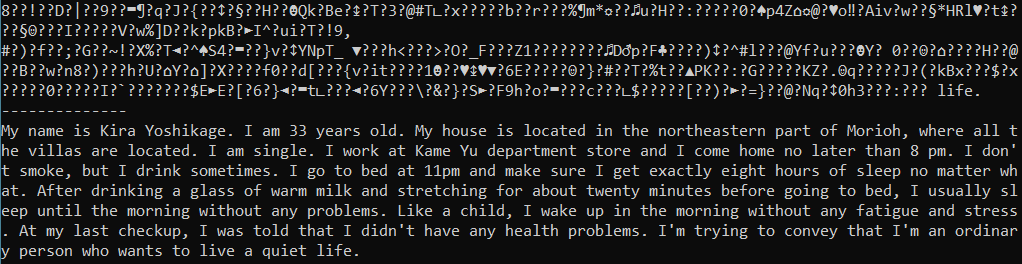


Рисунок 1. Результат работы программы