Министерство образования Республики Беларусь

Белорусский государственный университет информатики и радиоэлектроники

Кафедра информатики

ОТЧЕТ

по лабораторной работе №2

По дисциплине «Методы защиты информации»

По теме «Симметричная криптография. СТБ 34.101.31-2007»

Выполнил:

Студент гр. 853504

Пресный В.И,

Проверил:

Олисейчик В.В.

Минск 2021

**Задание:** Реализовать программные средства шифрования и дешифрования текстовых файлов при помощи алгоритма СТБ 34.101.31-2007 в различных режимах.

СТБ 34.101.31 – блочный шифр с 256-битным ключом и 8 циклами криптопреобразований, оперирующий с 128-битными блоками. Криптографические алгоритмы стандарта построены на основе базовых режимов шифрования блоков данных. Алгоритмы шифрования, описанные в стандарте:

* режим простой замены;
* режим сцепления блоков;
* режим гаммирования с обратной связью;
* режим счётчика;

Также в стандарте описаны:

* выработка имитовставки;
* одновременное шифрование и имитозащиты данных;
* одновременное шифрование и имитозащиты ключей;
* алгоритм хэширования;

Каждая группа включает алгоритм зашифрования и алгоритм расшифрования с использованием секретного ключа. Предполагается, что стороны будут обмениваться сообщениями, используя один ключ, заранее распределен между ними. В режимах простой замены и сцепления блоков шифруются сообщения, которые содержат хотя бы один блок, а в режимах гаммирования с обратной связью и счётчика — сообщения произвольной длины.

**Алгоритм**

1. **Шифрование блока**

**Входные данные и выходные данные**

Входными данными алгоритмов зашифрования и расшифрования являются блок  {\displaystyle X\in \{0,1\}^{128}}и ключ {\displaystyle \theta \in \{0,1\}^{256}.}

Выходными данными является блок {\displaystyle Y\in \{0,1\}^{128}}— результат зашифрования либо расшифрования слова на ключе  либо 

Входные данные для шифрования подготавливаются следующим образом:

* Слово {\displaystyle X} записывается в виде {\displaystyle X=X\_{1}\|X\_{2}\|X\_{3}\|X\_{4},X\_{i}\in \{0,1\}^{32}.}
* Ключ записывается в виде  и определяются тактовые ключи

**Обозначения и вспомогательные преобразования**

Преобразование  ставит в соответствие слову , слово



* циклический сдвиг влево на r бит.

 операция замены 8-битной входной строки подстановкой с рисунка 1.

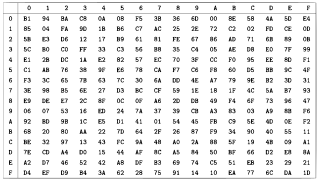


Рисунок 1 – Преобразование Н

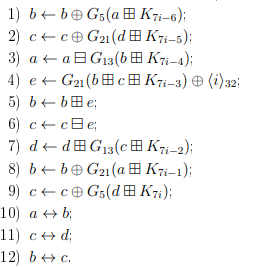
Подстановка {\displaystyle H:\{0,1\}^{8}\rightarrow \{0,1\}^{8}}задается фиксированной таблицей. В таблице используется шестнадцатеричное представление слов {\displaystyle u\in \{0,1\}^{8}.}

{\displaystyle \boxplus } и  {\displaystyle \boxminus -}операции сложения и вычитания по модулю 232{\displaystyle 2^{32}.}

1. **Зашифрование**

Для зашифрования блока {\displaystyle X} на ключе {\displaystyle \theta } выполняются следующие шаги:

1. Установить {\displaystyle a\leftarrow X\_{1},b\leftarrow X\_{2},c\leftarrow X\_{3},d\leftarrow X\_{4}.}
2. Для {\displaystyle i}i = 1,2,… ,8 выполнить:



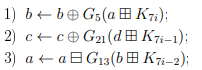
3. Установить {\displaystyle Y\leftarrow b\|d\|a\|c.}

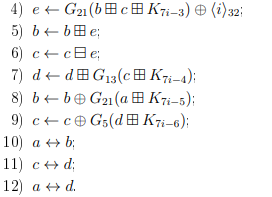
4. Возвратить {\displaystyle Y.}

**3. Расшифрование**

Для расшифрования блока {\displaystyle X} на ключе {\displaystyle \theta } выполняются следующие шаги:

1. Установить {\displaystyle a\leftarrow X\_{1},b\leftarrow X\_{2},c\leftarrow X\_{3},d\leftarrow X\_{4}.}
2. Для {\displaystyle i}i = 8,7,… ,1 выполнить:





3. Установить {\displaystyle Y\leftarrow c\|a\|d\|b.} 

4. Возвратить {\displaystyle Y.} 

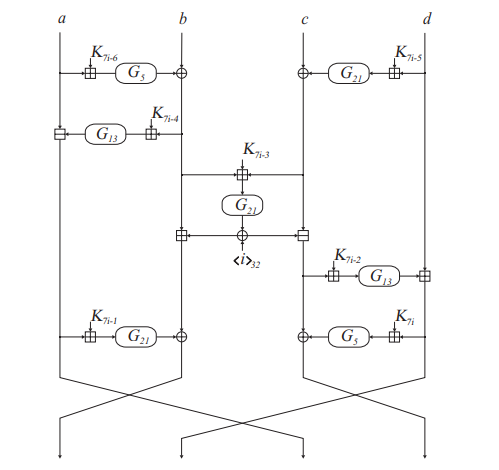
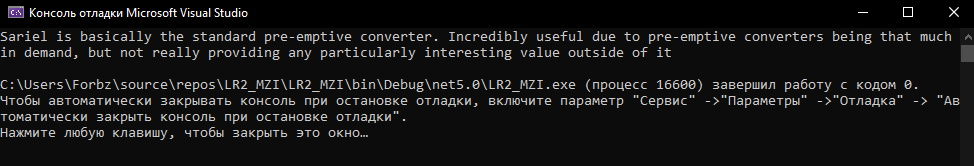


Рисунок 2 – Вычисления на i-м такте шифрования

**Результат работы программы**



**Программнй код**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LR2\_MZI

{

class STB

{

private static readonly byte[,] \_H = new byte[,]

{

{ 0xB1, 0x94, 0xBA, 0xC8, 0x0A, 0x08, 0xF5, 0x3B, 0x36, 0x6D, 0x00, 0x8E, 0x58, 0x4A, 0x5D, 0xE4 },

{ 0x85, 0x04, 0xFA, 0x9D, 0x1B, 0xB6, 0xC7, 0xAC, 0x25, 0x2E, 0x72, 0xC2, 0x02, 0xFD, 0xCE, 0x0D },

{ 0x5B, 0xE3, 0xD6, 0x12, 0x17, 0xB9, 0x61, 0x81, 0xFE, 0x67, 0x86, 0xAD, 0x71, 0x6B, 0x89, 0x0B },

{ 0x5C, 0xB0, 0xC0, 0xFF, 0x33, 0xC3, 0x56, 0xB8, 0x35, 0xC4, 0x05, 0xAE, 0xD8, 0xE0, 0x7F, 0x99 },

{ 0xE1, 0x2B, 0xDC, 0x1A, 0xE2, 0x82, 0x57, 0xEC, 0x70, 0x3F, 0xCC, 0xF0, 0x95, 0xEE, 0x8D, 0xF1 },

{ 0xC1, 0xAB, 0x76, 0x38, 0x9F, 0xE6, 0x78, 0xCA, 0xF7, 0xC6, 0xF8, 0x60, 0xD5, 0xBB, 0x9C, 0x4F },

{ 0xF3, 0x3C, 0x65, 0x7B, 0x63, 0x7C, 0x30, 0x6A, 0xDD, 0x4E, 0xA7, 0x79, 0x9E, 0xB2, 0x3D, 0x31 },

{ 0x3E, 0x98, 0xB5, 0x6E, 0x27, 0xD3, 0xBC, 0xCF, 0x59, 0x1E, 0x18, 0x1F, 0x4C, 0x5A, 0xB7, 0x93 },

{ 0xE9, 0xDE, 0xE7, 0x2C, 0x8F, 0x0C, 0x0F, 0xA6, 0x2D, 0xDB, 0x49, 0xF4, 0x6F, 0x73, 0x96, 0x47 },

{ 0x06, 0x07, 0x53, 0x16, 0xED, 0x24, 0x7A, 0x37, 0x39, 0xCB, 0xA3, 0x83, 0x03, 0xA9, 0x8B, 0xF6 },

{ 0x92, 0xBD, 0x9B, 0x1C, 0xE5, 0xD1, 0x41, 0x01, 0x54, 0x45, 0xFB, 0xC9, 0x5E, 0x4D, 0x0E, 0xF2 },

{ 0x68, 0x20, 0x80, 0xAA, 0x22, 0x7D, 0x64, 0x2F, 0x26, 0x87, 0xF9, 0x34, 0x90, 0x40, 0x55, 0x11 },

{ 0xBE, 0x32, 0x97, 0x13, 0x43, 0xFC, 0x9A, 0x48, 0xA0, 0x2A, 0x88, 0x5F, 0x19, 0x4B, 0x09, 0xA1 },

{ 0x7E, 0xCD, 0xA4, 0xD0, 0x15, 0x44, 0xAF, 0x8C, 0xA5, 0x84, 0x50, 0xBF, 0x66, 0xD2, 0xE8, 0x8A },

{ 0xA2, 0xD7, 0x46, 0x52, 0x42, 0xA8, 0xDF, 0xB3, 0x69, 0x74, 0xC5, 0x51, 0xEB, 0x23, 0x29, 0x21 },

{ 0xD4, 0xEF, 0xD9, 0xB4, 0x3A, 0x62, 0x28, 0x75, 0x91, 0x14, 0x10, 0xEA, 0x77, 0x6C, 0xDA, 0x1D }

};

private static uint ModAdd(params uint[] data)

{

var mod = (ulong)Math.Pow(2, 32);

ulong res = 0;

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

{

res = (res + data[i]) % mod;

}

return (uint)res;

}

private static uint ModSub(uint a, uint b)

{

var mod = (ulong)Math.Pow(2, 32);

return (uint)(((ulong)a - (ulong)(b)) % mod);

}

public static byte GetBits(ref byte a)

{

byte res = 0;

for (int i = 0; i < 4; i++)

{

res += (byte)((a % 2) \* Math.Pow(2, i));

a /= 2;

}

return res;

}

private static byte H(byte u)

{

var R = GetBits(ref u);

var L = GetBits(ref u);

return \_H[L, R];

}

static uint shift(uint a, int n)

{

return (a << n) | (a >> (32 - n));

}

private static uint G(uint \_u, byte r)

{

var u = BitConverter.GetBytes(\_u);

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

{

u[i] = H(u[i]);

}

var res = BitConverter.ToUInt32(u, 0);

var a = shift(res, r);

return a;

}

private static void Swap(ref uint a, ref uint b)

{

var tmp = a;

a = b;

b = tmp;

}

private static uint[] ExtendKey(uint[] data)

{

var res = new uint[data.Length \* 7];

for (int i = 0; i < 7; i++)

{

for (int j = 0; j < 8; j++)

{

res[i \* 8 + j] = data[j];

}

}

return res;

}

public static uint[] Encrypt(uint[] data, uint[] key)

{

var a = data[0];

var b = data[1];

var c = data[2];

var d = data[3];

key = ExtendKey(key);

for (uint i = 1; i <= 8; i++)

{

b = b ^ G(ModAdd(a, key[7 \* i - 7]), 5);

c = c ^ G(ModAdd(d, key[7 \* i - 6]), 21);

a = (ModSub((a), (G(ModAdd(b, key[7 \* i - 5]), 13))));

var e = G(ModAdd(b, c, key[7 \* i - 4]), 21) ^ (i);

b = ModAdd(b, e);

c = (ModSub((c), (e)));

d = ModAdd(d, G(ModAdd(c, key[7 \* i - 3]), 13));

b = b ^ G(ModAdd(a, key[7 \* i - 2]), 21);

c = c ^ G(ModAdd(d, key[7 \* i - 1]), 5);

Swap(ref a, ref b);

Swap(ref c, ref d);

Swap(ref b, ref c);

}

return new uint[] { b, d, a, c };

}

public static uint[] Decrypt(uint[] data, uint[] key)

{

var a = data[0];

var b = data[1];

var c = data[2];

var d = data[3];

key = ExtendKey(key);

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

{

b = b ^ G(ModAdd(a, key[7 \* i - 1]), 5);

c = c ^ G(ModAdd(d, key[7 \* i - 2]), 21);

a = (ModSub((a), (G(ModAdd(b, key[7 \* i - 3]), 13))));

var e = G(ModAdd(b, c, key[7 \* i - 4]), 21) ^ ((uint)i);

b = ModAdd(b, e);

c = (ModSub((c), (e)));

d = ModAdd(d, G(ModAdd(c, key[7 \* i - 5]), 13));

b = b ^ G(ModAdd(a, key[7 \* i - 6]), 21);

c = c ^ G(ModAdd(d, key[7 \* i - 7]), 5);

Swap(ref a, ref b);

Swap(ref c, ref d);

Swap(ref a, ref d);

}

return new uint[] { c, a, d, b };

}

public static uint[] SimpleReplaceEncrypt(uint[] data, uint[] key)

{

List<uint> dat = new List<uint>(data);

while (dat.Count % 4 != 0)

{

dat.Add(0);

}

var res = new uint[dat.Count];

for (int i = 0; i < dat.Count / 4; i++)

{

var take = dat.Skip(i \* 4).Take(4).ToArray();

Encrypt(take, key).CopyTo(res, i \* 4);

}

return res;

}

public static uint[] SimpleReplaceDecrypt(uint[] data, uint[] key)

{

List<uint> dat = new List<uint>(data);

while (dat.Count % 4 != 0)

{

dat.Add(0);

}

var res = new uint[dat.Count];

for (int i = 0; i < dat.Count / 4; i++)

{

var take = dat.Skip(i \* 4).Take(4).ToArray();

Decrypt(take, key).CopyTo(res, i \* 4);

}

return res;

}

}

}