Дніпровський національний університет імені Олеся Гончара Факультет прикладної математики Кафедра обчислювальної математики та математичної кібернетики

Лабораторна робота №3

Захист інформації

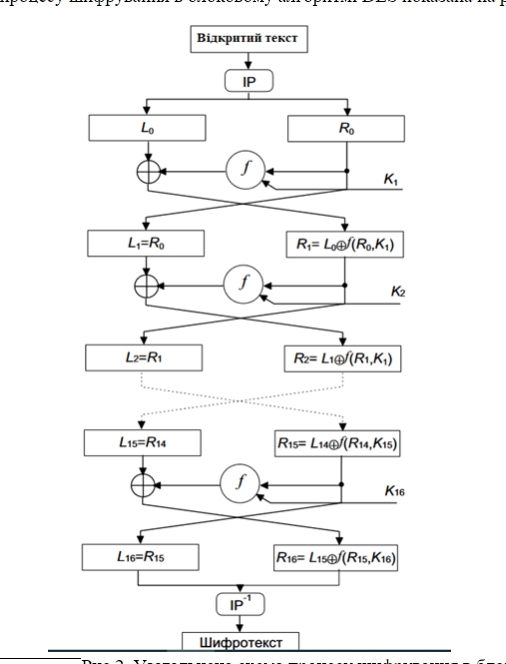
студента групи ПА-19-2

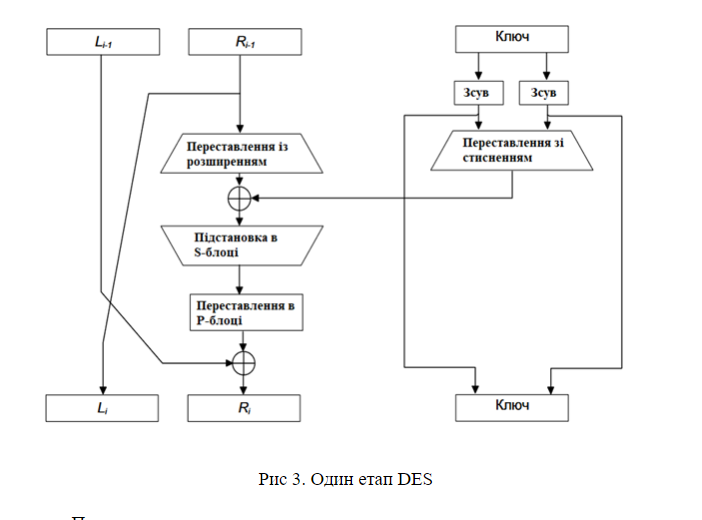
Ільяшенко Єгора

Варіант №7

Дніпро, 2022

Алгоритми DES



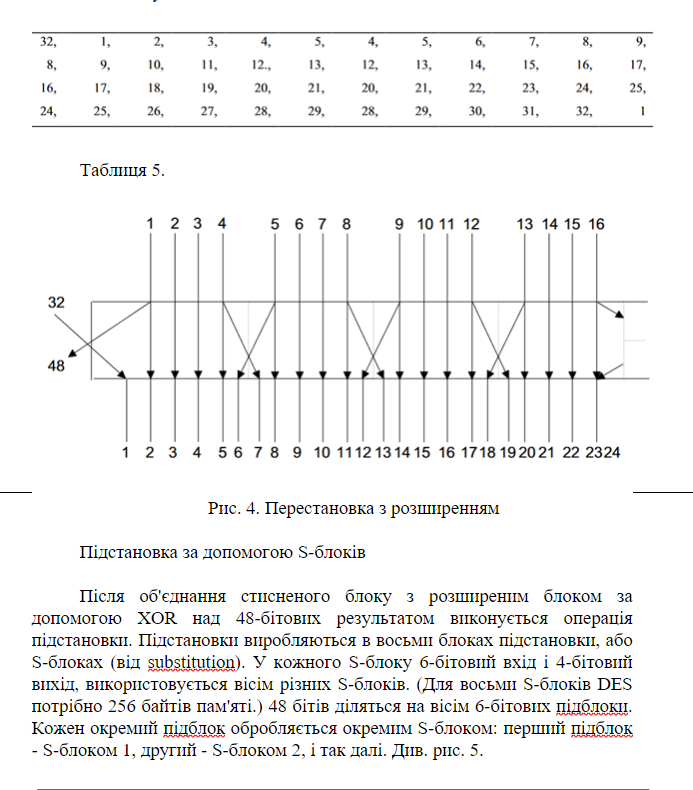


Изображение выглядит как стол

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание



Изображение выглядит как стол

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание



Изображение выглядит как стол

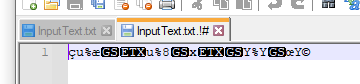
Автоматически созданное описание

Изображение выглядит как стол

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание



Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Код програми:

class SDES

{

BitArray[,] S\_Box1 = new BitArray[4, 4];

BitArray[,] S\_Box2 = new BitArray[4, 4];

BitArray Master\_key;

public SDES(string \_key)

{

Master\_key = new BitArray(10);

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

{

Master\_key[i] = str2bin(\_key[i]);

}

BitArray b0 = new BitArray(2);

b0[0] = false;

b0[1] = false;

BitArray b1 = new BitArray(2);

b0[0] = false;

b0[1] = true;

BitArray b2 = new BitArray(2);

b0[0] = true;

b0[1] = false;

BitArray b3 = new BitArray(2);

b0[0] = true;

b0[1] = true;

S\_Box1[0, 0] = b1;

S\_Box1[0, 1] = b0;

S\_Box1[0, 2] = b3;

S\_Box1[0, 3] = b2;

S\_Box1[1, 0] = b3;

S\_Box1[1, 1] = b2;

S\_Box1[1, 2] = b1;

S\_Box1[1, 3] = b0;

S\_Box1[2, 0] = b0;

S\_Box1[2, 1] = b2;

S\_Box1[2, 2] = b1;

S\_Box1[2, 3] = b3;

S\_Box1[3, 0] = b3;

S\_Box1[3, 1] = b1;

S\_Box1[3, 2] = b3;

S\_Box1[3, 3] = b2;

//---------------------

S\_Box2[0, 0] = b0;

S\_Box2[0, 1] = b1;

S\_Box2[0, 2] = b2;

S\_Box2[0, 3] = b3;

S\_Box2[1, 0] = b2;

S\_Box2[1, 1] = b0;

S\_Box2[1, 2] = b1;

S\_Box2[1, 3] = b3;

S\_Box2[2, 0] = b3;

S\_Box2[2, 1] = b0;

S\_Box2[2, 2] = b1;

S\_Box2[2, 3] = b0;

S\_Box2[3, 0] = b2;

S\_Box2[3, 1] = b1;

S\_Box2[3, 2] = b0;

S\_Box2[3, 3] = b3;

//---------------------

}

public byte Encrypt(byte block)

{

BitArray bits\_block = byte2bits(block);

BitArray[] keys = Generate\_Keys();

return bits2byte(RIP(Fk(Switch(Fk(IP(bits\_block), keys[0])), keys[1])));

//ciphertext = IP-1( fK2 ( SW (fK1 (IP (plaintext)))))

}

public byte Decrypt(byte block)

{

BitArray bits\_block = byte2bits(block);

BitArray[] keys = Generate\_Keys();

return bits2byte(RIP(Fk(Switch(Fk(IP(bits\_block), keys[1])), keys[0])));

//IP-1 ( fK1( SW( fK2( IP(ciphertext)))))

}

BitArray byte2bits(byte block)

{

string bits = decimal2binstr(block);

BitArray result = new BitArray(8);

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

{

result[i] = str2bin(bits[i]);

}

return result;

}

byte bits2byte(BitArray block)

{

string result = "";

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

{

result += bin2str(block[i]);

}

return binstr2decimal(result);

}

BitArray[] Generate\_Keys()

{

BitArray [] keys = new BitArray[2];

BitArray[] temp = Split\_Block(P10(Master\_key));

keys[0] = P8(Circular\_left\_shift(temp[0], 1), Circular\_left\_shift(temp[1], 1));

keys[1] = P8(Circular\_left\_shift(temp[0], 3), Circular\_left\_shift(temp[1], 3)); //1 + 2 = 3

return keys;

}

// decimal to binary string

public string decimal2binstr(byte num)

{

string ret = "";

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

{

if (num % 2 == 1)

ret = "1" + ret;

else

ret = "0" + ret;

num >>= 1;

}

return ret;

}

// binary to decimal string

public byte binstr2decimal(string binstr)

{

byte ret = 0;

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

{

ret <<= 1;

if (binstr[i] == '1')

ret++;

}

return ret;

}

public string bin2str(bool input)

{

if (input)

return "1";

else

return "0";

}

public bool str2bin(char bit)

{

if (bit == '0')

return false;

else if (bit == '1')

return true;

else

throw new Exception("Key should be in binary format [0,1]");

}

//generates permated array P10

BitArray P10(BitArray key)

{

//0 1 2 3 4 5 6 7 8 9

//2 4 1 6 3 9 0 8 7 5

BitArray permutatedArray = new BitArray(10);

permutatedArray[0] = key[2];

permutatedArray[1] = key[4];

permutatedArray[2] = key[1];

permutatedArray[3] = key[6];

permutatedArray[4] = key[3];

permutatedArray[5] = key[9];

permutatedArray[6] = key[0];

permutatedArray[7] = key[8];

permutatedArray[8] = key[7];

permutatedArray[9] = key[5];

return permutatedArray;

}

//generates permuted array P8

BitArray P8(BitArray part1, BitArray part2)

{

//0 1 2 3 4 5 6 7

//5 2 6 3 7 4 9 8

//6 3 7 4 8 5 10 9

BitArray permutatedArray = new BitArray(8);

permutatedArray[0] = part2[0];//5

permutatedArray[1] = part1[2];

permutatedArray[2] = part2[1];//6

permutatedArray[3] = part1[3];

permutatedArray[4] = part2[2];//7

permutatedArray[5] = part1[4];

permutatedArray[6] = part2[4];//9

permutatedArray[7] = part2[3];//8

return permutatedArray;

}

BitArray P4(BitArray part1, BitArray part2)

{

//0 1 2 3

//2 4 3 1

//1 3 2 0

BitArray permutatedArray = new BitArray(4);

permutatedArray[0] = part1[1];

permutatedArray[1] = part2[1];//3

permutatedArray[2] = part2[0];//2

permutatedArray[3] = part1[0];

return permutatedArray;

}

BitArray EP(BitArray input)

{

//0 1 2 3

//4 1 2 3 2 3 4 1

//3 0 1 2 1 2 3 0

BitArray permutatedArray = new BitArray(8);

permutatedArray[0] = input[3];

permutatedArray[1] = input[0];

permutatedArray[2] = input[1];

permutatedArray[3] = input[2];

permutatedArray[4] = input[1];

permutatedArray[5] = input[2];

permutatedArray[6] = input[3];

permutatedArray[7] = input[0];

return permutatedArray;

}

//generates permuted text IP

BitArray IP(BitArray plainText)

{

//0 1 2 3 4 5 6 7

//1 5 2 0 3 7 4 6

BitArray permutatedArray = new BitArray(8);

permutatedArray[0] = plainText[1];

permutatedArray[1] = plainText[5];

permutatedArray[2] = plainText[2];

permutatedArray[3] = plainText[0];

permutatedArray[4] = plainText[3];

permutatedArray[5] = plainText[7];

permutatedArray[6] = plainText[4];

permutatedArray[7] = plainText[6];

return permutatedArray;

}

BitArray RIP(BitArray permutedText)

{

//0 1 2 3 4 5 6 7

//3 0 2 4 6 1 7 5

BitArray permutatedArray = new BitArray(8);

permutatedArray[0] = permutedText[3];

permutatedArray[1] = permutedText[0];

permutatedArray[2] = permutedText[2];

permutatedArray[3] = permutedText[4];

permutatedArray[4] = permutedText[6];

permutatedArray[5] = permutedText[1];

permutatedArray[6] = permutedText[7];

permutatedArray[7] = permutedText[5];

return permutatedArray;

}

BitArray Circular\_left\_shift(BitArray a, int bitNumber)

{

BitArray shifted = new BitArray(a.Length);

int index = 0;

for (int i = bitNumber; index < a.Length; i++)

{

shifted[index++] = a[i%a.Length];

}

return shifted;

}

BitArray[] Split\_Block(BitArray block)

{

BitArray[] splited = new BitArray[2];

splited[0] = new BitArray(block.Length / 2);

splited[1] = new BitArray(block.Length / 2);

int index = 0;

for (int i = 0; i < block.Length/2; i++)

{

splited[0][i] = block[i];

}

for (int i = block.Length / 2; i < block.Length; i++)

{

splited[1][index++] = block[i];

}

return splited;

}

BitArray S\_Boxes(BitArray input, int no)

{

BitArray[,] current\_S\_Box;

if (no == 1)

current\_S\_Box = S\_Box1;

else

current\_S\_Box = S\_Box2;

return current\_S\_Box[binstr2decimal(bin2str(input[0]) + bin2str(input[3])),

binstr2decimal(bin2str(input[1]) + bin2str(input[2]))];

}

BitArray F(BitArray right, BitArray sk)

{

BitArray[] temp = Split\_Block(Xor(EP(right), sk));

return P4(S\_Boxes(temp[0], 1), S\_Boxes(temp[1], 2));

}

BitArray Fk(BitArray IP, BitArray key)

{

BitArray[] temp = Split\_Block(IP);

BitArray Left = Xor(temp[0], F(temp[1], key));

BitArray joined = new BitArray(8);

int index = 0;

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

{

joined[index++] = Left[i];

}

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

{

joined[index++] = temp[1][i];

}

return joined;

}

BitArray Switch(BitArray input)

{

BitArray switched = new BitArray(8);

int index = 0;

for (int i = 4; index < input.Length; i++)

{

switched[index++] = input[i%input.Length];

}

return switched;

}

BitArray Xor(BitArray a, BitArray b)

{

return b.Xor(a);

}

}