МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра «Информатики и систем управления»

ОТЧЕТ

по лабораторной работе

по дисциплине

Методы и средства защиты информации

ПРЕПОДАВАТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Капранов С.Н. (подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гореев А.Д. (подпись) (фамилия, и.,о.)

19-ИСТ-2

(шифр группы)

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород

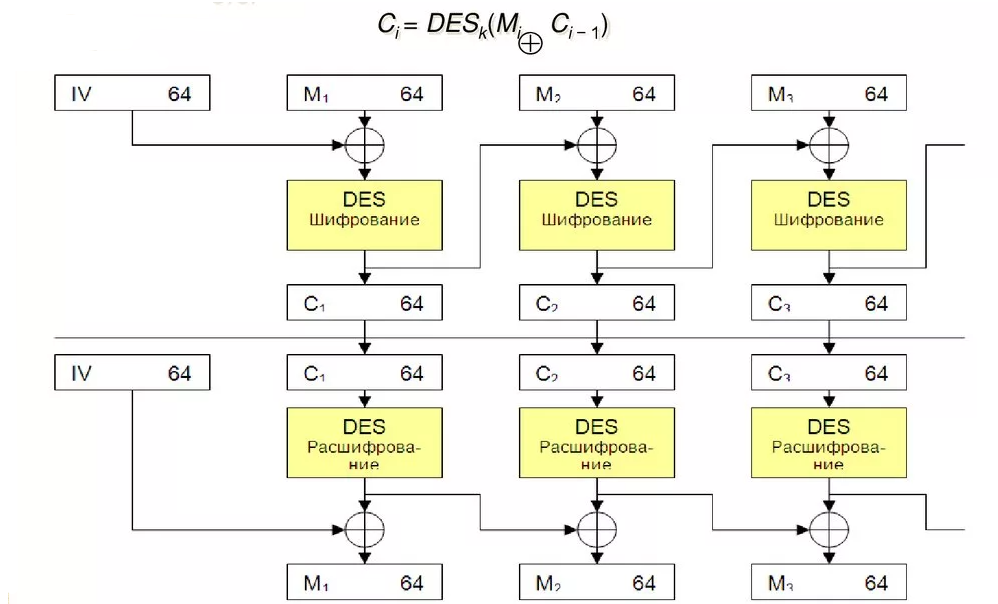
2021

**Вариант № 7**

**Задание №4.1** Реализовать стандарт шифрования данных DES в режиме «Сцепление блоков шифрованного текста».

**Описание алгоритма**

Используется похожий режим шифрования, как и «Электронная кодовая книга». Первый блок складывается по модулю 2 с вектором инициализации и потом с ключом. Последующие блоки складываются по модулю два с предыдущим блоком.



**Листинг программы**

#include <iostream>

#include <string>

#include <windows.h>

#include <cmath>

using namespace std;

// 1-я таблица перестановок для преобразования 64 бит в 56

const int firstPermutTab[56] = {

57,49,41,33,25,17,9,

1,58,50,42,34,26,18,

10,2,59,51,43,35,27,

19,11,3,60,52,44,36,

63,55,47,39,31,23,15,

7,62,54,46,38,30,22,

14,6,61,53,45,37,29,

21,13,5,28,20,12,4

};

// 2-я таблица перестановок для преобразования 56 в 48 бит при переходе

// со стороны ключа на сторону открытого текста

const int secondPermutTab[48] = {

14,17,11,24,1,5,

3,28,15,6,21,10,

23,19,12,4,26,8,

16,7,27,20,13,2,

41,52,31,37,47,55,

30,40,51,45,33,48,

44,49,39,56,34,53,

46,42,50,36,29,32

};

// Начальная таблица перестановок для текста

const int firstPermutTabText[64] = {

58,50,42,34,26,18,10,2,

60,52,44,36,28,20,12,4,

62,54,46,38,30,22,14,6,

64,56,48,40,32,24,16,8,

57,49,41,33,25,17,9,1,

59,51,43,35,27,19,11,3,

61,53,45,37,29,21,13,5,

63,55,47,39,31,23,15,7

};

// Таблица функции расширения E

const int expandTab[48] = {

32,1,2,3,4,5,4,5,

6,7,8,9,8,9,10,11,

12,13,12,13,14,15,16,17,

16,17,18,19,20,21,20,21,

22,23,24,25,24,25,26,27,

28,29,28,29,30,31,32,1

};

// Функции преобразования S(i)

// 48 бит делится на подблоки по 6 бит (S-box).

// Функция S преобразует 6 бит в 4 бита.

const int compressTab[8][4][16] =

{ {

14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7,

0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8,

4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0,

15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13

},

{

15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10,

3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5,

0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15,

13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9

},

{

10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8,

13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1,

13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7,

1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12

},

{

7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15,

13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9,

10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4,

3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14

},

{

2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9,

14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6,

4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14,

11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3

},

{

12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11,

10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8,

9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6,

4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13

},

{

4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1,

13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6,

1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2,

6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12

},

{

13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7,

1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2,

7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8,

2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11

} };

// Таблица перестановок P

const int permutRight32[32] = {

16,7,20,21,29,12,28,17,

1,15,23,26,5,18,31,10,

2,8,24,14,32,27,3,9,

19,13,30,6,22,11,4,25

};

// Обратная таблица перестановок

const int secondPermutTabText[64] = {

40,8,48,16,56,24,64,32,

39,7,47,15,55,23,63,31,

38,6,46,14,54,22,62,30,

37,5,45,13,53,21,61,29,

36,4,44,12,52,20,60,28,

35,3,43,11,51,19,59,27,

34,2,42,10,50,18,58,26,

33,1,41,9,49,17,57,25

};

// Массив для хранения 16 сгенерированных ключей

string roundKeys[16];

// строка для шифрования блока текста, длиной 64 бита

string encrStr;

// функция перевода в верхний регистр

string highRegister(string& word)

{

for (int i = 0; i < word.length(); i++)

{

if (word[i] >= 'а' && word[i] <= 'я' || word[i] >= 'a' && word[i] <= 'z')

{

word[i] = word[i] - 32;

}

}

return word;

}

// Функция перевода чисел из двоичной системы счисления в десятичную

int binToDec(string& bin)

{

int dec = 0;

for (int i = bin.length() - 1, j = 0; i >= 0; i--, j++)

{

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

{

dec += pow(2, j);

}

}

return dec;

}

// функция перевода строки в двоичную систему

string strToBin(string& str)

{

string binStr = "";

for (int i = 0; i < str.length(); i++)

{

int val = abs(int(str[i]));

string bin = "";

while (val > 0)

{

(val % 2) ? bin.push\_back('1') : bin.push\_back('0');

val /= 2;

}

while (bin.size() < 8) {

bin += '0';

}

reverse(bin.begin(), bin.end());

binStr += bin;

}

return binStr;

}

// Функция перевода текста из двоичной системы в символьную

string binToStr(string& str)

{

string result = "";

string block = "";

int count = 0;

for (int i = 0; i < str.length() / 8; i++)

{

block = "";

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

{

block += str[count];

count++;

}

int dec = binToDec(block);

if (dec - 'A' < 0 && dec != ' ')

{

dec = dec \* (-1);

}

result += dec;

}

return result;

}

// Функция перевода числа из десятичной в двоичную систему счисления

string convDecToBin(int& dec)

{

// строка для записи двоичного числа

string bin;

// алгоритм перевода

while (dec != 0)

{

bin += (dec % 2 == 0 ? "0" : "1");

dec = dec / 2;

}

while (bin.length() < 4)

{

bin += "0";

}

return bin;

}

// Функция для выполнения кругового сдвига влево на 1 разделенной части ключа

string shiftLeft1(string& key) {

string shift = "";

for (int i = 1; i < 28; i++)

{

shift += key[i];

}

shift += key[0];

return shift;

}

// Функция для выполнения кругового сдвига влево на 2 разделенной части ключа

string shiftLeft2(string& key) {

string shift = "";

for (int i = 0; i < 2; i++)

{

for (int j = 1; j < 28; j++)

{

shift += key[j];

}

shift += key[0];

key = shift;

shift = "";

}

return key;

}

// Функция XOR (Исключающее ИЛИ)

string XOR(string &str1, string &str2)

{

string res = "";

int counter = 0;

if (str1.size() % str2.size() == 0)

{

counter = str1.size() / str2.size();

}

else

{

counter = str1.size() / str2.size() + 1;

}

while (counter > 0)

{

for (int i = 0; i < str2.size(); i++)

{

res += (str1[i] == str2[i]) ? '0' : '1';

}

counter--;

}

return res;

}

//Функция для генерации 16 ключей

void genKeys(string& key)

{

// Сжатие ключа с помощью 1-й таблицы перестановок

string permKey = "";

for (int i = 0; i < 56; i++)

{

permKey += key[firstPermutTab[i] - 1];

}

// Деление ключа на две равные части по 28 бит

string left = permKey.substr(0, 28);

string right = permKey.substr(28, 28);

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

{

// Для циклов 1, 2, 9, 16 сдвиг на единицу

if (i == 0 || i == 1 || i == 8 || i == 15)

{

left = shiftLeft1(left);

right = shiftLeft1(right);

}

// Для всех остальных циклов сдвиг на 2

else

{

left = shiftLeft2(left);

right = shiftLeft2(right);

}

// Объединение 2 частей

string combKey = left + right;

string roundKey = "";

// Используем 2-ю таблицу перестановок

for (int i = 0; i < 48; i++)

{

roundKey += combKey[secondPermutTab[i] - 1];

}

roundKeys[i] = roundKey;

}

}

// Функция шифрования

string encryption(string &vector)

{

//Начальную перестановка

string res = "";

for (int i = 0; i < 64; i++)

{

res += encrStr[firstPermutTabText[i] - 1];

}

// Деление результата на две равные половины

string left = res.substr(0, 32);

string right = res.substr(32, 32);

// Шифруем текст 16 раз

// переменная для хранения зашифрованного текста

string encrText = "";

//предыдущий блок

string previousEncryp = "";

string result = "";

int count = 0;

//проводим 16 раундов шифрования

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

{

string rightExpand = "";

// Правая половина текста расширяется

for (int i = 0; i < 48; i++)

{

rightExpand += right[expandTab[i] - 1];

};

// Применяем операцию "Исключающее ИЛИ"

string strXOR = XOR(roundKeys[i], rightExpand);

string res = "";

// Результат делится на 8 равных частей и пропускается через 8 S-блоков.

//После прохождения каждый блок уменьшается с 6 до 4 бит.

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

{

// Поиск индексов строк и столбцов для S-преобразований

string rowStr = strXOR.substr(i \* 6, 1) + strXOR.substr(i \* 6 + 5, 1);

int row = binToDec(rowStr);

string colStr = strXOR.substr(i \* 6 + 1, 1) + strXOR.substr(i \* 6 + 2, 1) + strXOR.substr(i \* 6 + 3, 1) + strXOR.substr(i \* 6 + 4, 1);

int col = binToDec(colStr);

int val = compressTab[i][row][col];

res += convDecToBin(val);

}

if (count == 0)

{

result = XOR(res, vector);

previousEncryp = result;

}

if (count > 0) //с предыдущими блоками

{

result = XOR(res, previousEncryp);

previousEncryp = "";

previousEncryp = result;

}

// P преобразования с помощью таблицы перестановок P

string perm2 = "";

for (int i = 0; i < 32; i++)

{

perm2 += result[permutRight32[i] - 1];

}

// Результат сравнивается с левой половиной с помощью XOR (исключающее ИЛИ)

strXOR = XOR(perm2, left);

//левая и правая части тектса меняются местами

left = strXOR;

if (i < 15)

{

string temp = right;

right = strXOR;

left = temp;

}

// объединение обеих сторон для создания полного зашифрованного текста

string combined\_text = left + right;

encrText = "";

// Применяется обратная перестановка

for (int i = 0; i < 64; i++)

{

encrText += combined\_text[secondPermutTabText[i] - 1];

}

}

//Возвращение зашифрованного текста

return encrText;

}

void Encrypt()

{

cout << "Введите строку: ";

string str;

getline(cin, str);

string text = highRegister(str);

cout << "Введите ключ, состоящий из 8 символов: ";

string key;

getline(cin, key);

cout << "Введите вектор, состоящий из 8 символов: ";

string vector;

getline(cin, vector);

key = strToBin(key);

int symbol = 0;

//если исходный текст не делится на блоки по 64 бита, то он

//дополняется символами с конца

for (int i = 0; i < (text.length() % 8); i++)

{

text += " ";

symbol++;

}

//зашифрованное сообщение

string encrypted = "";

int count = 0;

//шифрование сообщения

for (int i = 0; i < (text.length() / 8); i++)

{

string block = "";

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

{

block += text[count];

count++;

}

encrStr = strToBin(block);

genKeys(key);

encrypted += encryption(vector);

}

cout << endl;

cout << "Зашифрованное сообщение: " << encrypted << endl

<< endl;

string encryptedStr = binToStr(encrypted);

cout << "Зашифрованное сообщение: " << encryptedStr << endl

<< endl;

int i = 15;

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

{

string temp = roundKeys[i];

roundKeys[i] = roundKeys[j];

roundKeys[j] = temp;

i--;

}

count = 0;

//расшифрованное сообщение

string decrypted;

//расшифровка сообщения

for (int k = 0; k < (text.length() / 8); k++)

{

string binBlock = "";

for (int m = 0; m < 64; m++)

{

binBlock += encrypted[count];

count++;

}

encrStr = binBlock;

//при расшифровке данных все действия выполняются в

//обратном порядке

string result = encryption(vector);

decrypted += result;

}

cout << "Расшифрованное сообщение: " << endl << decrypted

<< endl << endl;

decrypted = binToStr(decrypted);

cout << "Расшифрованное сообщение: " << endl << decrypted

<< endl;

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

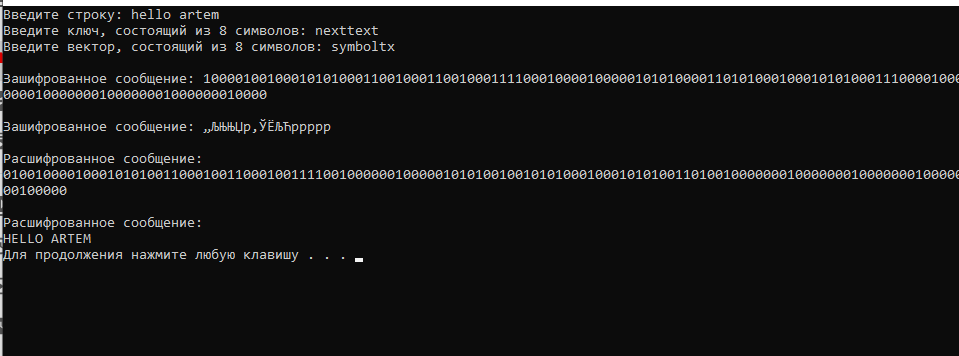
Encrypt();

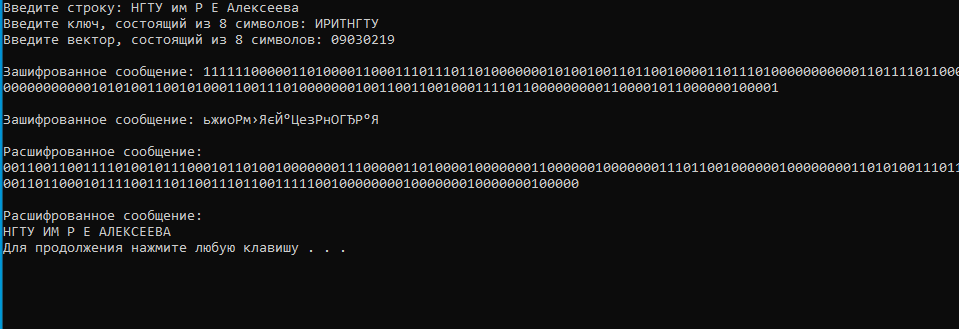
system("pause");

return 0;

}

**Примеры работы программы**



****

