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 №4

по дисциплине

«Средства и методы защиты информации»

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_ Капранов С. Н.\_\_

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_ Сорокин Е. А. \_\_\_\_

(подпись) (фамилия, и.,о.)

\_\_\_\_\_\_\_\_ 18-В1 \_\_\_\_\_\_\_

(шифр группы)

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

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

Нижний Новгород 2021

**Задание №4.3**

Реализовать стандарт шифрования данных DES в режиме в режиме Обратная связь по шифротексту.

**Теория**

DES (англ. Data Encryption Standard) — алгоритм для симметричного шифрования, разработанный фирмой IBM и утверждённый правительством США в 1977 году как официальный стандарт (FIPS 46-3). Размер блока для DES равен 64 битам. В основе алгоритма лежит сеть Фейстеля с 16 циклами (раундами) и ключом, имеющим длину 56 бит. Алгоритм использует комбинацию нелинейных (S-блоки) и линейных (перестановки E, IP, IP-1) преобразований.

**Алгоритм**

**Начальная перестановка**

Исходный текст T (блок 64 бит) преобразуется c помощью начальной перестановки IP.

**Циклы шифрования**

Полученный после начальной перестановки 64-битовый блок IP(T) участвует в 16 циклах преобразования Фейстеля.

16 циклов преобразования Фейстеля:

Разбить IP(T) на две части L0, R0, где L0, R0 — соответственно 32 старших битов и 32 младших битов блока T0. IP(T)= L0R0.

Пусть Ti-1 = Li-1Ri-1 результат (i-1) итерации, тогда результат i-ой итерации Ti = LiRi определяется:

Li = Ri-1

Ri = Li-1 XOR f(Ri-1, ki)

Левая половина Li равна правой половине предыдущего вектора Li-1­Ri-1. А правая половина Ri — это битовое сложение Li-1 и {\displaystyle f(Ri-1, ki) по модулю 2.

В 16-циклах преобразования Фейстеля функция f играет роль шифрования.

**Основная функция шифрования (функция Фейстеля)**

Аргументами функции f являются 32-битовый вектор Ri-1 и 48-битовый ключ ki, который является результатом преобразования 56-битового исходного ключа шифра k. Для вычисления функции f последовательно используются

1. функция расширения E,
2. сложение по модулю 2 с ключом ki,
3. преобразование S, состоящее из 8 преобразований S-блоков S1, S2, S3, S4, S5, S6, S7, S8,
4. перестановка P.

**Генерирование ключей ki**

Первоначально генерируется 56-битный ключ k. Из него, с помощью перестановки получается другой 56-битный блок, правая и левая половины которого обозначаются как C0 и D0 соответственно. Для каждого раунда шифрования получают Ci и Di как результат циклического сдвига влево на 1 или 2 бита в зависимости от раунда для Ci-1 и Di-1 соответственно. Затем получают 48-битный ключ ki как перестановку CiDi

**Конечная перестановка**

Конечная перестановка IP-1 действует на T16 (где T16 = L16R16) и является обратной к первоначальной перестановке.

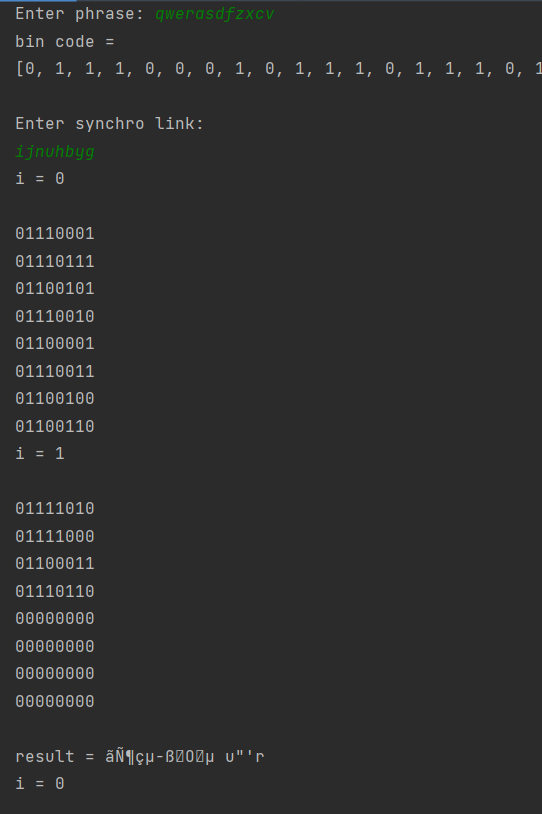
**Расшифрование**

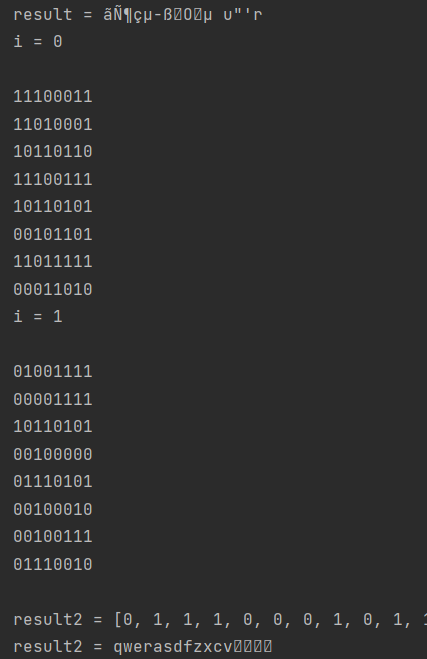
При расшифровании данных все действия выполняются в обратном порядке. В 16 циклах расшифрования, в отличие от шифрования c помощью прямого преобразования сетью Фейстеля, здесь используется обратное преобразование сетью Фейстеля.

**Режим Обратной связи по шифротексту**

Режим обратной связи по шифротексту. В режиме CFB вырабатывается блочная «гамма». Начальный вектор является синхропосылкой и предназначен для того, чтобы разные наборы данных шифровались по-разному с использованием одного и того же секретного ключа. Синхропосылка посылается получателю в открытом виде вместе с зашифрованным файлом. Алгоритм DES, в отличие от предыдущих режимов, используется только как шифрование (в обоих случаях).  
Один из вариантов использования симметричного блочного шифра, при котором для шифрования следующего блока открытого текста он складывается по модулю 2 с перешифрованным (блочным шифром) результатом шифрования предыдущего блока.

**Пример работы программы**





**Код программы**

import java.nio.charset.StandardCharsets;  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.Scanner;  
  
public class Main {  
 // создадим необходимы константные таблицы  
 public static final int[] IP = new int[] {  
 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  
 };  
  
 public static final int[] FP = new int[] {  
 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  
 };  
  
 public static final int[] EP = new int[] {  
 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  
 };  
  
 public static final int[] P = new int[] {  
 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  
 };  
  
 public static final int[] K1P = new int[] {  
 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  
 };  
  
 public static final int[] K2P = new int[] {  
 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  
 };  
  
 public static final int[] CP = new int[] {  
 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  
 };  
  
 public static final int[][][] Sbox = new int[][][] {  
 { // 0  
 {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}  
 },  
 { // 1  
 {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 }  
 },  
 { // 2  
 {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}  
 },  
 { // 3  
 {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}  
 },  
 { // 4  
 {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 }  
 },  
 { // 5  
 {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}  
 },  
 { // 6  
 {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}  
 },  
 { // 7  
 {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}  
 }  
 };  
  
 // преобразуем исходную строку в список бинарных значений  
 public static ArrayList<Integer> convertToBinaryStr(String phrase) {  
  
 StringBuilder binStr = new StringBuilder(phrase);  
 byte[] bstr = phrase.getBytes(StandardCharsets.UTF\_8);  
 ArrayList<Integer> binCode = new ArrayList<>();  
 for (int i = binStr.length()-1; i >= 0 ; i--) {  
 int temp = Integer.parseInt(Integer.toBinaryString(bstr[i])); // двоичное число  
 // теперь нужно преобразовать число в массив цифр  
 // и записать этот массив посимвольно в ArrayList  
 while (temp != 0) {  
 binCode.add(temp%10);  
 temp /= 10;  
 }  
 if (binCode.size()%8 != 0) {  
 while (binCode.size()%8 != 0) {  
 binCode.add(0);  
 }  
 }  
 }  
 Collections.reverse(binCode);  
  
 return binCode;  
 }  
  
 public static void addUpTo64(ArrayList<Integer> list) {  
 if (list.size()%64 != 0) {  
 while (list.size()%64 != 0) {  
 list.add(0);  
 }  
 }  
 }  
  
 // преобразуем список бинарных значений в строку  
 public static StringBuilder convertToCharStr(ArrayList<Integer> list) {  
 StringBuilder temp = new StringBuilder(8);  
 String temp2;  
 StringBuilder result = new StringBuilder();  
 for (int i = 0; i < list.size(); i++) {  
 temp.append(list.get(i));  
 if ((i+1) % 8 == 0) {  
 temp2 = temp.toString();  
 result.append( (char)(Integer.parseInt(temp2, 2)) );  
 temp = new StringBuilder(8);  
 }  
  
 }  
 return result;  
 }  
  
 public static ArrayList<Integer> initialPermutation(ArrayList<Integer> list) {  
 ArrayList<Integer> result= new ArrayList<>();  
 for (int i = 0; i < 64; i++) {  
 result.add(list.get(IP[i]-1));  
 }  
 return result;  
 }  
  
 public static ArrayList<Integer> expansionPermutation(ArrayList<Integer> block32) {  
 ArrayList<Integer> result = new ArrayList<>();  
 for (int i = 0; i < 48; i++ ) {  
 result.add(block32.get(EP[i]-1));  
 }  
 return result;  
 }  
  
 public static void split48bitsTo6bits(  
 ArrayList<Integer> block48, ArrayList<ArrayList<Integer>> blocks6b) {  
 ArrayList<Integer> temp = new ArrayList<>();  
 for (int i = 0; i < 48; i++) {  
 temp.add(block48.get(i));  
 if ((i+1)%6 == 0) {  
 blocks6b.add(temp);  
 temp = new ArrayList<>();  
 }  
 }  
 }  
  
 public static int extremeBits(ArrayList<Integer> list) {  
 StringBuilder str = new StringBuilder();  
 str.append(list.get(0));  
 str.append(list.get(5));  
 return Integer.parseInt(str.toString(),2);  
 }  
  
 public static int middleBits(ArrayList<Integer> list) {  
 StringBuilder str = new StringBuilder();  
 str.append(list.get(1));  
 str.append(list.get(2));  
 str.append(list.get(3));  
 str.append(list.get(4));  
 return Integer.parseInt(str.toString(),2);  
 }  
  
 public static void substitution6bitsTo4bits(  
 ArrayList<ArrayList<Integer>> blocks6b, ArrayList<ArrayList<Integer>> blocks4b) {  
 int block2b;  
 int block4b;  
 ArrayList<Integer> temp = new ArrayList<>();  
 StringBuilder strb;  
 for (int i = 0; i < 8; i++) {  
 block2b = extremeBits(blocks6b.get(i));  
 block4b = middleBits(blocks6b.get(i));  
 strb = new StringBuilder(Integer.toBinaryString(Sbox[i][block2b][block4b]));  
 //не всегда мы получим именно 4 цифры  
 //старшие нули нужно дописать вручную  
 StringBuilder newstr = new StringBuilder();  
 for (int k = 0; k < (4-strb.length()); k++) {  
 newstr.append(0);  
 }  
 newstr.append(strb);  
 for (int j = 0; j < 4; j++) {  
 temp.add(Integer.parseInt(newstr.charAt(j) + ""));  
 }  
 blocks4b.add(temp);  
 temp = new ArrayList<>();  
 }  
 }  
  
 public static ArrayList<Integer> join4bitsTo32bits(ArrayList<ArrayList<Integer>> blocks4b) {  
 ArrayList<Integer> result = new ArrayList<>();  
 for (ArrayList<Integer> list : blocks4b) {  
 for (int i = 0; i < 4; i++) {  
 result.add(list.get(i));  
 }  
 }  
 return result;  
 }  
  
 public static ArrayList<Integer> substitutions(ArrayList<Integer> block48) {  
 ArrayList<ArrayList<Integer>> blocks6b = new ArrayList<>();  
 ArrayList<ArrayList<Integer>> blocks4b = new ArrayList<>();  
 split48bitsTo6bits(block48, blocks6b);  
 substitution6bitsTo4bits(blocks6b, blocks4b);  
 return join4bitsTo32bits(blocks4b);  
 }  
  
 public static ArrayList<Integer> permutation(ArrayList<Integer> block32) {  
 ArrayList<Integer> result = new ArrayList<>();  
 for (int i = 0; i < 32; i++) {  
 result.add(block32.get(P[i]-1));  
 }  
 return result;  
 }  
  
 public static ArrayList<Integer> XOR(ArrayList<Integer> list1, ArrayList<Integer> list2) {  
 ArrayList<Integer> result = new ArrayList<>();  
 for (int i = 0; i < list1.size(); i++) {  
 if ((!list1.get(i).equals(list2.get(i)))) {  
 result.add(1);  
 } else {  
 result.add(0);  
 }  
 }  
 return result;  
 }  
  
 public static ArrayList<Integer> funcF(ArrayList<Integer> block32, ArrayList<Integer> key48) {  
 ArrayList<Integer> block48 = expansionPermutation(block32);  
 block48 = XOR(block48, key48);  
 block32 = substitutions(block48);  
 return permutation(block32);  
 }  
  
 public static ArrayList<ArrayList<Integer>> roundFeistelCipher(  
 ArrayList<ArrayList<Integer>> block, ArrayList<Integer> key48) {  
 ArrayList<Integer> temp = new ArrayList<>();  
 // запишем N2 в temp  
 for (int i = 0; i < 32; i++) {  
 temp.add(block.get(1).get(i));  
 }  
  
 block.set(1, XOR(funcF(block.get(1), key48), block.get(0)));  
  
 for (int i = 0; i < 32; i++) {  
 block.get(0).set(i, temp.get(i));  
 }  
  
 return block;  
 }  
  
 public static void swap(ArrayList<Integer> N1, ArrayList<Integer> N2) {  
 ArrayList<Integer> temp = new ArrayList<>();  
 for (int i = 0; i < 32; i++) {  
 temp.add(N1.get(i));  
 }  
 for (int i = 0; i < 32; i++) {  
 N1.set(i, N2.get(i));  
 N2.set(i, temp.get(i));  
 }  
 }  
  
 public static ArrayList<Integer> feistelCipher(  
 ArrayList<Integer> list, ArrayList<ArrayList<Integer>> keys) {  
 // разобьём на два блока по 32 бита  
 ArrayList<Integer> N1 = new ArrayList<>();  
 ArrayList<Integer> N2 = new ArrayList<>();  
 for (int i = 0; i < 64; i++) {  
 if (i < 32) {  
 N1.add(list.get(i));  
 } else {  
 N2.add(list.get(i));  
 }  
 }  
 // основной цикл в 16 эпох  
 ArrayList<ArrayList<Integer>> block = new ArrayList<>();  
 block.add(N1);  
 block.add(N2);  
// if (mode == 'e') {  
 for (int i = 0; i < 16; i++) {  
 block = roundFeistelCipher(block, keys.get(i));  
 }  
 swap(block.get(0), block.get(1));  
  
// } else if (mode == 'd') {  
// for (int i = 15; i >= 0; i--) {  
// block = roundFeistelCipher(block, keys.get(i));  
// }  
// swap(block.get(0), block.get(1));  
// }  
  
 list = new ArrayList<>();  
 for (int i = 0; i < 64; i++) {  
 if (i < 32) {  
 list.add(block.get(0).get(i));  
 } else {  
 list.add(block.get(1).get(i%32));  
 }  
 }  
 return list;  
 }  
  
 public static void lShift(ArrayList<Integer> block28b, int n) {  
 for (int i = 0; i < n; i++) {  
 block28b.add(block28b.get(0));  
 block28b.remove(0);  
 }  
 }  
  
 public static ArrayList<Integer> join28bitsTo56bits(  
 ArrayList<Integer> block28b1, ArrayList<Integer> block28b2) {  
 ArrayList<Integer> result = new ArrayList<>();  
 for (int i = 0; i < 56; i++) {  
 if (i < 28) {  
 result.add(block28b1.get(i));  
 } else {  
 result.add(block28b2.get(i%28));  
 }  
 }  
 return result;  
 }  
  
 public static ArrayList<Integer> key\_contraction\_permutation(ArrayList<Integer> block56b) {  
 ArrayList<Integer> block48b = new ArrayList<>();  
 for (int i = 0; i < 48; i++) {  
 block48b.add(block56b.get(CP[i]-1));  
 }  
 return block48b;  
 }  
  
 private static void keyExpansionTo48bits(  
 ArrayList<Integer> block28b1,  
 ArrayList<Integer> block28b2,  
 ArrayList<ArrayList<Integer>> keys48b) {  
 ArrayList<Integer> block56b;  
 int n;  
 for (int i = 0; i < 16; i++) {  
 switch (i) {  
 case 0: case 1: case 8: case 15: n = 1; break;  
 default: n = 2; break;  
 }  
 lShift(block28b1, n);  
 lShift(block28b2, n);  
 block56b = join28bitsTo56bits(block28b1, block28b2);  
 keys48b.add(key\_contraction\_permutation(block56b));  
 }  
 }  
  
 public static void keyPermutation56bitsTo28bits(  
 ArrayList<Integer> key, ArrayList<ArrayList<Integer>> keys48b) {  
 ArrayList<Integer> block28b1 = new ArrayList<>();  
 ArrayList<Integer> block28b2 = new ArrayList<>();  
 for (int i = 0; i < 28; i++) {  
 block28b1.add(key.get(K1P[i]));  
 block28b2.add(key.get(K2P[i]));  
 }  
 keyExpansionTo48bits(block28b1, block28b2, keys48b);  
 }  
  
 public static ArrayList<Integer> finalPermutation(ArrayList<Integer> workBloc) {  
 ArrayList<Integer> result= new ArrayList<>();  
 for (int i = 0; i < 64; i++) {  
 result.add(workBloc.get(FP[i]-1));  
 }  
 return result;  
 }  
  
 public static ArrayList<Integer> DES(  
 ArrayList<Integer> binaryCode,  
 char mode,  
 ArrayList<ArrayList<Integer>> keys48b,  
 ArrayList<Integer> binarySynchro) {  
  
 ArrayList<Integer> result = new ArrayList<>();  
  
 // выделяем первый блок  
 ArrayList<Integer> workBloc = new ArrayList<>();  
  
 ArrayList<Integer> preResult;  
  
 int a = 0;  
 while (!binaryCode.isEmpty()) {  
 if (a == 0) {  
 preResult = cipherProc(binarySynchro, keys48b);  
 } else {  
 preResult = cipherProc(workBloc, keys48b);  
 }  
  
 workBloc = new ArrayList<>();  
 for (int i = 0; i < 64; i++) {  
 workBloc.add(binaryCode.get(0));  
 binaryCode.remove(0);  
 }  
 System.out.println("i = " + a);  
 for (int k = 0; k < 64; k++) {  
 if (k%8 == 0) {  
 System.out.println();  
 }  
 System.out.print(workBloc.get(k));  
  
 }  
 System.out.println();  
 a++;  
  
 if (mode == 'e') {  
 workBloc = XOR(workBloc, preResult);  
  
 for (int j = 0; j < 64; j++) {  
 result.add(workBloc.get(j));  
 }  
 } else if (mode == 'd') {  
 ArrayList<Integer> open;  
 open = XOR(workBloc, preResult);  
  
 for (int j = 0; j < 64; j++) {  
 result.add(open.get(j));  
 }  
 }  
  
 }  
  
 return result;  
 }  
  
 public static ArrayList<Integer> cipherProc(  
 ArrayList<Integer> binarySynchro, ArrayList<ArrayList<Integer>> keys48b) {  
  
 binarySynchro = initialPermutation(binarySynchro);  
  
 binarySynchro = feistelCipher(binarySynchro, keys48b);  
  
 binarySynchro = finalPermutation(binarySynchro);  
  
 return binarySynchro;  
  
 }  
  
 public static void main(String[] args) {  
  
 String skey = "DESkey56";  
  
 Scanner scan = new Scanner(System.in);  
  
 System.out.print("Enter phrase: ");  
 String phrase = scan.next();  
  
 // преобразовали фразу в последовательность битов  
 ArrayList<Integer> binaryCode = convertToBinaryStr(phrase);  
  
 //здесь нужно разбить строку на блоки по 64 бита или дополнить до 64 бит  
 addUpTo64(binaryCode);  
  
 System.out.println("bin code = \n" + binaryCode);  
 System.out.println();  
  
 System.out.println("Enter synchro link: ");  
 String synchro = scan.next();  
  
 // преобразовали посылку в последовательность битов  
 ArrayList<Integer> binarySynchro = convertToBinaryStr(synchro);  
 //здесь нужно разбить строку на блоки по 64 бита или дополнить до 64 бит  
 addUpTo64(binarySynchro);  
  
  
 // создадим все 16 ключей  
 // получили ключ в битах  
 ArrayList<Integer> key = convertToBinaryStr(skey);  
 // список для хранения всех 16 ключей  
 ArrayList<ArrayList<Integer>> keys48b = new ArrayList<>();  
 // передаём список, чтобы сохранить в него все ключи  
 keyPermutation56bitsTo28bits(key, keys48b);  
  
  
 ArrayList<Integer> result = DES(binaryCode, 'e', keys48b, binarySynchro);  
 System.out.println();  
 System.out.println("result = " + convertToCharStr(result));  
  
  
 ArrayList<Integer> result2 = DES(result, 'd', keys48b, binarySynchro);  
 System.out.println();  
 System.out.println("result2 = " + result2);  
 System.out.println("result2 = " + convertToCharStr(result2));  
  
 }  
}