**EX.NO.4: DATA ENCRYPTION STANDARD (DES)**

**CODE:**

import java.util.\*;

class DES {

String plainText = new String();

String cipherText = new String();

String key = new String();

String keys[] = new String[16];

// Initial Permutation Table

int[] IP = { 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 };

// Inverse Initial Permutation Table

int[] FP = { 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 };

// first key-Permutation Table

int[] PC1 = { 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 };

// second key-Permutation Table

int[] PC2 = { 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 };

// Expansion D-box Table

int[] EP = { 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 };

// Straight Permutation Table

int[] P = { 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 };

// S-box Table

int[][][] sbox = {

{ { 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 } }

};

int[] shiftBits = { 1, 1, 2, 2, 2, 2, 2, 2,

1, 2, 2, 2, 2, 2, 2, 1 };

String leftCircularShift(String input, int numBits) {

int n = input.length();

String shifted = "";

shifted+=input.substring(numBits,n);

shifted+=input.substring(0,numBits);

return shifted;

}

String permutation(int[] table, String input) {

String output = "";

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

output += input.charAt(table[i] - 1);

}

return output;

}

String asciiToBinary(String ascii) {

String bin\_T = "";

int n = ascii.length();

for (int i = 0; i < n; i++) {

int val = Integer.valueOf(ascii.charAt(i));

String bin = "";

while (val > 0) {

if (val % 2 == 1)

bin += '1';

else

bin += '0';

val /= 2;

}

bin = reverse(bin);

if (bin.length() < 8) {

int l = 8 - bin.length();

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

bin = '0' + bin;

}

bin\_T += bin + "";

}

return bin\_T;

}

String reverse(String input) {

char[] a = input.toCharArray();

int l, r = 0;

r = a.length - 1;

for (l = 0; l < r; l++, r--) {

char temp = a[l];

a[l] = a[r];

a[r] = temp;

}

return String.valueOf(a);

}

void generateKeys() {

key = asciiToBinary(key);

key = permutation(PC1, key);

System.out.println("\nOutput of PC1 (56-bit) in hex is "+binaryToHex(key).toUpperCase());

String roundKey = key;

System.out.println("\nThe round keys (48-bit) in hex are: ");

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

roundKey =

leftCircularShift(roundKey.substring(0, 28), shiftBits[i]) +

leftCircularShift(roundKey.substring(28, 56), shiftBits[i]);

keys[i] = permutation(PC2, roundKey);

System.out.println("Key "+(i+1)+": "+binaryToHex(keys[i]).toUpperCase());

}

}

String xor(String a, String b) {

String ans = "";

int n = a.length();

for (int i = 0; i < n; i++) {

if (a.charAt(i) == b.charAt(i))

ans += "0";

else

ans += "1";

}

return ans;

}

int binaryToDecimal(String binary) {

int dec = 0;

int base = 1;

int len = binary.length();

for (int i = len - 1; i >= 0; i--) {

if (binary.charAt(i) == '1')

dec += base;

base = base \* 2;

}

return dec;

}

String sBox(String input) {

String output = "";

String bin = "";

for (int i = 0; i < 48; i += 6) {

String temp = input.substring(i, i + 6);

int num = i / 6;

int row = binaryToDecimal(temp.charAt(0) + "" + temp.charAt(5));

int col = binaryToDecimal(temp.substring(1, 5));

bin = Integer.toBinaryString(sbox[num][row][col]);

if (bin.length() < 4) {

int l = 4 - bin.length();

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

bin = '0' + bin;

}

output += bin;

}

return output;

}

String round(String input, String key, int r\_num) {

String left = input.substring(0, 32);

String right = input.substring(32, 64);

String temp = right;

// Expansion permutation 32 to 48 bit

temp = permutation(EP, temp);

// xor temp and round key

temp = xor(temp, key);

// lookup in s-box table

temp = sBox(temp);

// Straight D-box

temp = permutation(P, temp);

// xor

left = xor(left, temp);

System.out.println("Round " +(r\_num + 1) +": "+binaryToHex(right).toUpperCase() +" " +binaryToHex(left).toUpperCase());

// swapping

return right + left;

}

void encrypt() {

plainText = asciiToBinary(plainText);

cipherText = plainText;

// initial permutation

cipherText = permutation(IP, cipherText);

System.out.println("\nOutput of initial permutation IP (in hex) : " + binaryToHex(cipherText).toUpperCase()+"\n");

// 16 rounds

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

cipherText = round(cipherText, keys[i], i);

}

// 32-bit swap

cipherText = cipherText.substring(32, 64) + cipherText.substring(0, 32);

// final permutation

cipherText = permutation(FP, cipherText);

cipherText = binaryToHex(cipherText);

System.out.println("\nThe cipher text is (in hex) " + cipherText.toUpperCase());

}

String binaryToHex(String binary) {

String hexStr = "";

for (int i = 0; i < binary.length(); i = i + 4) {

int decimal = Integer.parseInt(binary.substring(i, i + 4), 2);

hexStr = hexStr + Integer.toString(decimal, 16);

}

return hexStr;

}

String hexToBinary(String hex) {

String bin = "", temp = "";

int n = Integer.parseInt(hex.substring(0, 7), 16);

bin = Integer.toBinaryString(n);

if (bin.length() < 28) {

int l = 28 - bin.length();

for (int j = 0; j < l; j++) bin = '0' + bin;

}

n = Integer.parseInt(hex.substring(7, 14), 16);

temp = Integer.toBinaryString(n);

if (temp.length() < 28) {

int l = 28 - temp.length();

for (int j = 0; j < l; j++) temp = '0' + temp;

}

bin += temp;

n = Integer.parseInt(hex.substring(14, 16), 16);

temp = Integer.toBinaryString(n);

if (temp.length() < 8) {

int l = 8 - temp.length();

for (int j = 0; j < l; j++) temp = '0' + temp;

}

bin += temp;

return bin;

}

String binaryToAscii(String binary) {

String res = "";

for (int i = 0; i < binary.length(); i = i + 8) {

char ch = (char) binaryToDecimal(binary.substring(i, i + 8));

res += ch;

}

return res;

}

void decrypt() {

cipherText = hexToBinary(cipherText);

plainText = cipherText;

// initial permutation

plainText = permutation(IP, plainText);

System.out.println("\nOutput of initial permutation IP (in hex) : " + binaryToHex(plainText).toUpperCase());

// 16-rounds

for (int i = 15; i > -1; i--) {

plainText = round(plainText, keys[i], 15 - i);

}

// 32-bit swap

plainText = plainText.substring(32, 64) + plainText.substring(0, 32);

plainText = permutation(FP, plainText);

plainText = binaryToAscii(plainText);

System.out.println("\nThe plain text is (in ASCII) " + plainText.toUpperCase());

}

boolean validateString(String str){

return str.length()==8;

}

boolean validateCipherText(String str){

return str.length()==16;

}

public static void main(String args[]) {

Scanner sc = new Scanner(System.in);

DES des = new DES();

System.out.println("\nDATA ENCRYPTION STANDARD - DES");

System.out.println("\nKEY - GENERATION");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.print("\nEnter the key of length 8 (in ASCII): ");

des.key = sc.next();

while (!des.validateString(des.key)) {

System.out.println("\nInvalid key");

System.out.print("\nEnter the key: ");

des.key = sc.next();

}

des.generateKeys();

System.out.println("\nENCRYPTION");

System.out.println("\*\*\*\*\*\*\*\*\*\*");

System.out.print("\nEnter the plainText of length 8 (in ASCII): ");

des.plainText = sc.next();

while (!des.validateString(des.plainText)) {

System.out.println("\nInvalid plain text length");

System.out.print("\nEnter the plainText of length 8 (in ASCII): ");

des.plainText = sc.next();

}

des.encrypt();

System.out.println("\nDECRYPTION");

System.out.println("\*\*\*\*\*\*\*\*\*\*");

System.out.print("\nEnter the cipherText of length 16 (in hex): ");

des.cipherText = sc.next();

while (!des.validateCipherText(des.cipherText)) {

System.out.println("\nInvalid cipher text length");

System.out.print("\nEnter the cipherText of length 8 (in ASCII): ");

des.cipherText = sc.next();

}

System.out.print("\nEnter the key of length 8 (in ASCII): ");

des.key = sc.next();

while (!des.validateString(des.key)) {

System.out.println("\nInvalid key");

System.out.print("\nEnter the key: ");

des.key = sc.next();

}

des.generateKeys();

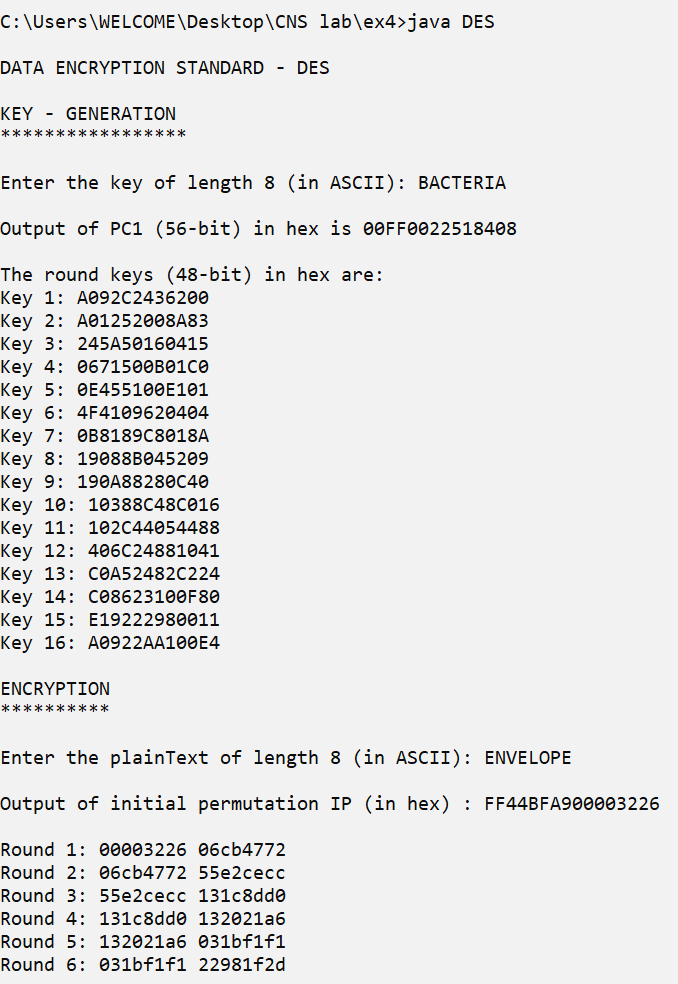
des.decrypt();

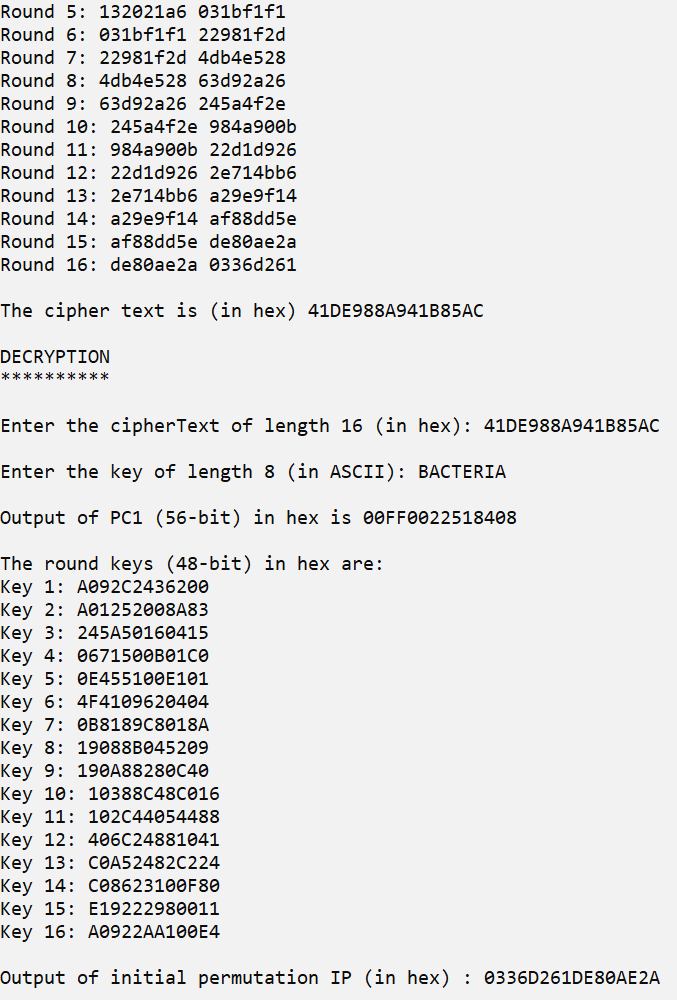
}

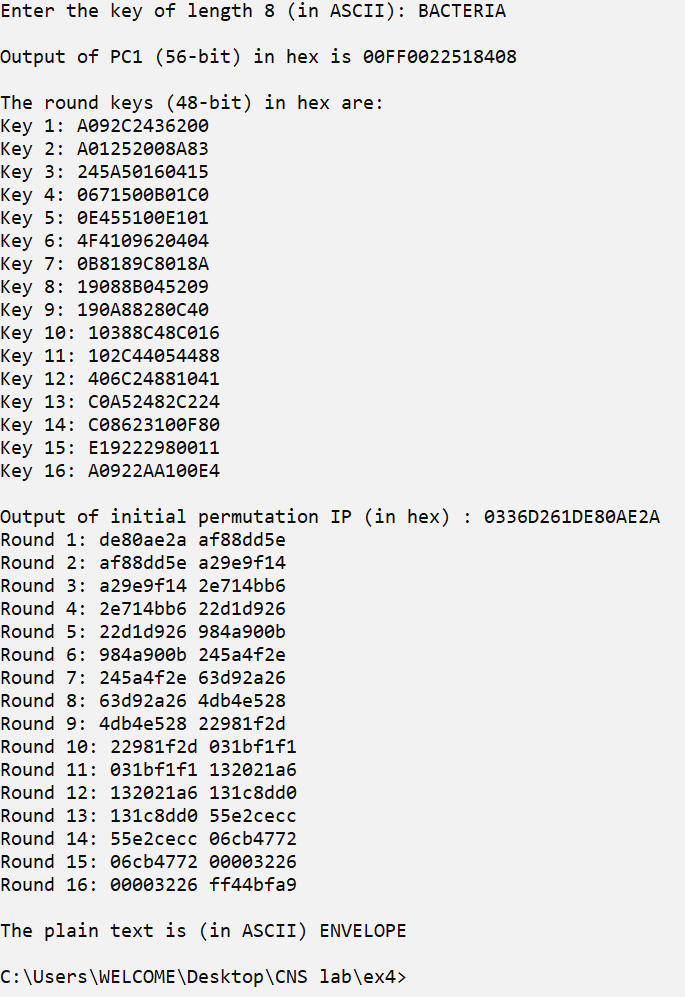
}

**OUTPUT:**

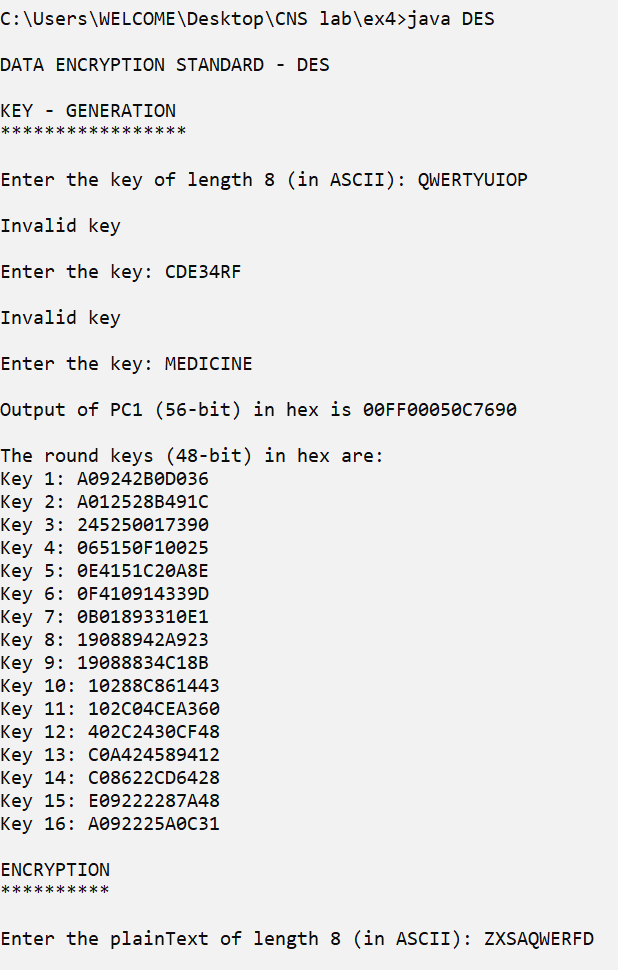
**Example 1:**

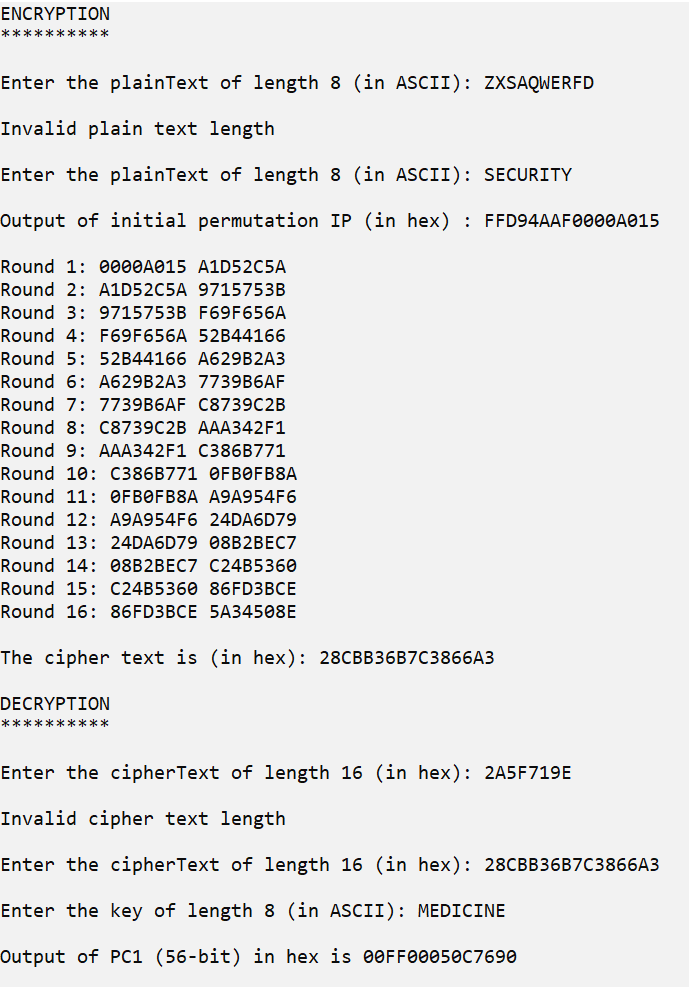
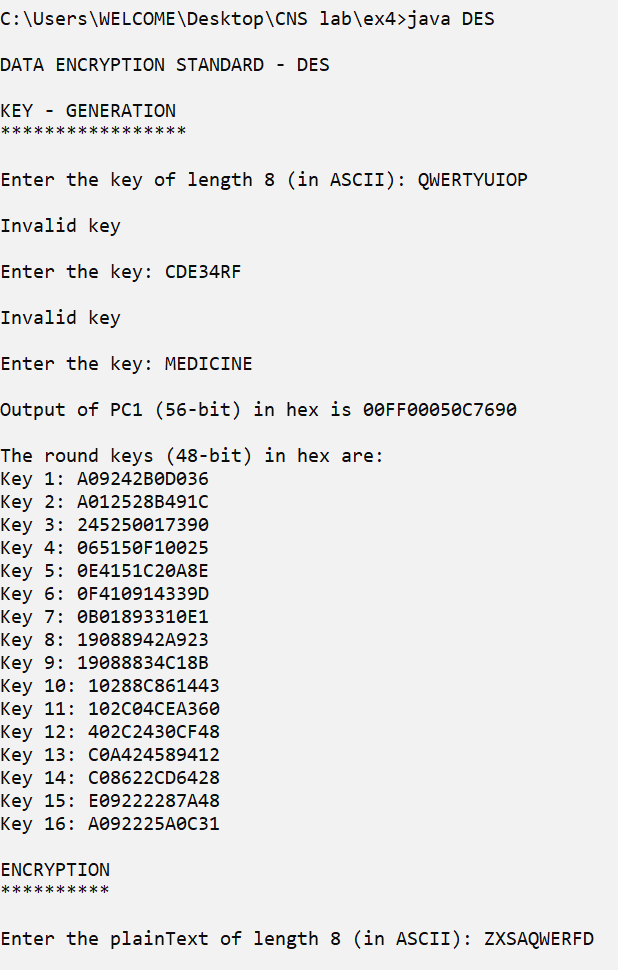


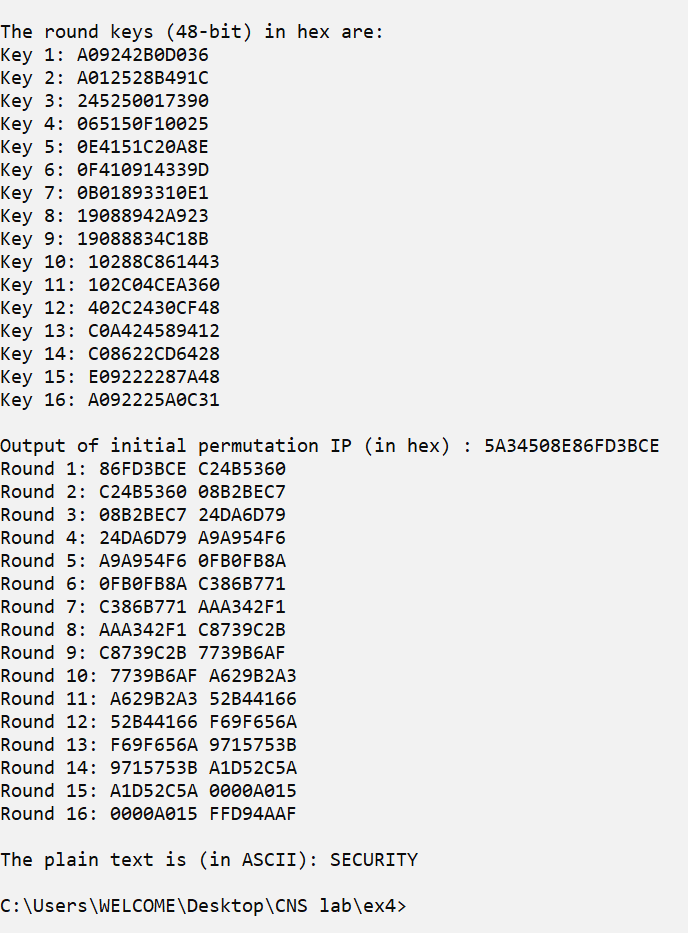
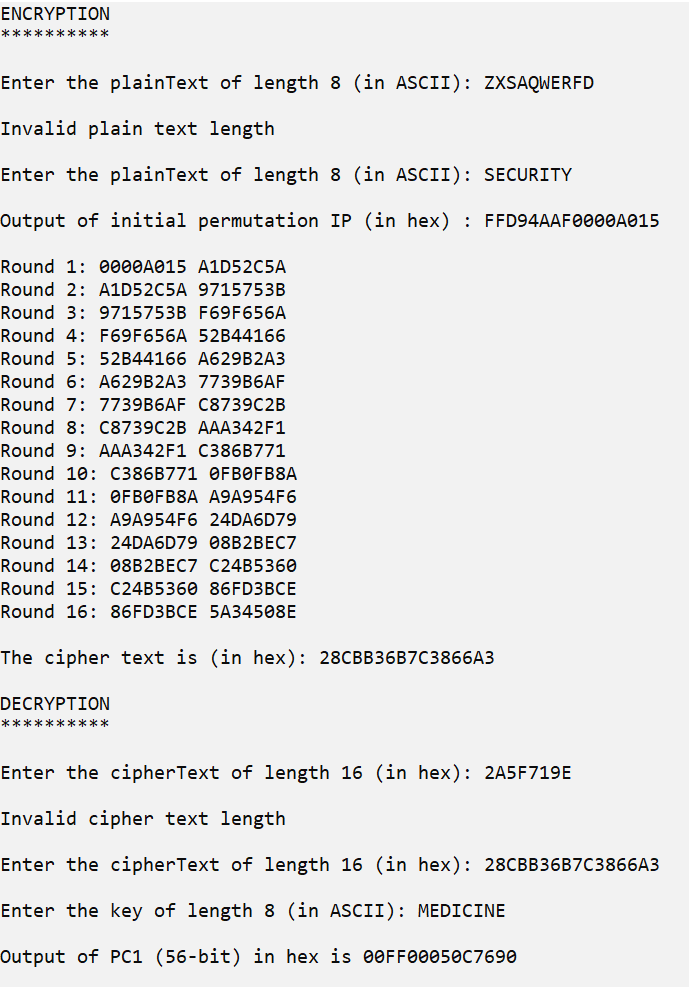


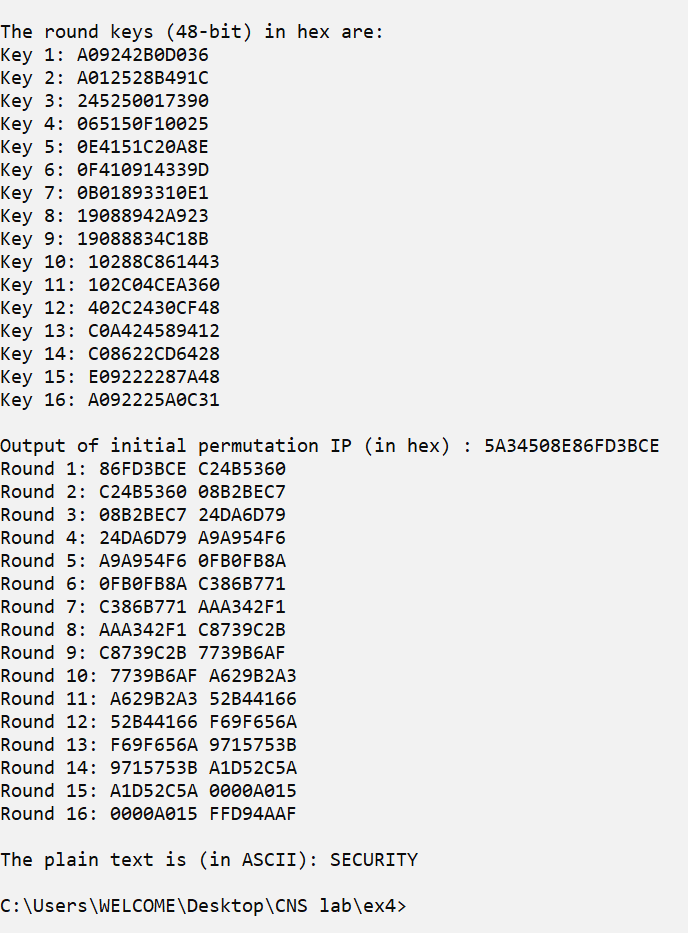


**Example 2:**

****

****

****

****