**IT8761 – Security Laboratory**

**Reshma Ramesh Babu**

**312217104129**

**Exercise 4**

**Aim:** To implement the Data Encryption Standard (DES) algorithm.

**Code:**

import java.util.\*;

class DES2 {

// Initial Permutation table

private static final byte[] 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

};

// Permuted Choice 1 table

private static final byte[] 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

};

// Permuted Choice 2 table

private static final byte[] 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

};

// Number of rotations in each round

private static final byte[] rotations = {

1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1

};

// Expansion table

private static final byte[] E = {

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

};

// S-boxes (i.e. Substitution boxes)

private static final byte[][] S = { {

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

} };

// Permutation table

private static final byte[] 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

};

// Inverse permutation table

private static final byte[] 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

};

// 28 bits each, used as storage in the KS (Key Structure) rounds to

private static int[] C = new int[28];

private static int[] D = new int[28];

private static int[][] subkey = new int[16][48];

public String encryption(int[] inputBits, int[] keyBits) {

return DES2.permute(inputBits, keyBits, false);

}

public String decryption(int[] inputBits, int[] keyBits) {

return DES2.permute(inputBits, keyBits, true);

}

private static void displayBits(int[] bits) {

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

String output = new String();

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

output += bits[i+j]; System.out.print(Integer.toBinaryString(Integer.parseInt(output, 2)));

}

System.out.println();

}

public void display(boolean isDecrypt) {

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

System.out.print("Round " + (n+1) + ": ");

if(isDecrypt) {

System.out.print("Key = ");

displayBits(subkey[15-n]);

} else {

System.out.print("Key = ");

displayBits(subkey[n]);

}

}

}

private static String permute(int[] inputBits, int[] keyBits, boolean isDecrypt) {

// Initial permutation step takes input bits and permutes into the newBits array

int newBits[] = new int[inputBits.length];

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

newBits[i] = inputBits[IP[i]-1];

}

int L[] = new int[32];

int R[] = new int[32];

int i;

for(i=0 ; i < 28 ; i++) {

C[i] = keyBits[PC1[i]-1];

}

for( ; i < 56 ; i++) {

D[i-28] = keyBits[PC1[i]-1];

}

System.arraycopy(newBits, 0, L, 0, 32);

System.arraycopy(newBits, 32, R, 0, 32);

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

int newR[] = new int[0];

if(isDecrypt) {

newR = fiestel(R, subkey[15-n]);

} else {

newR = fiestel(R, KS(n, keyBits));

}

int newL[] = xor(L, newR);

L = R;

R = newL;

}

int output[] = new int[64];

System.arraycopy(R, 0, output, 0, 32);

System.arraycopy(L, 0, output, 32, 32);

int finalOutput[] = new int[64];

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

finalOutput[i] = output[FP[i]-1];

String hex = new String();

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

String bin = new String();

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

bin += finalOutput[(4\*i)+j];

int decimal = Integer.parseInt(bin, 2);

hex += Integer.toHexString(decimal);

}

return hex.toUpperCase();

}

private static int[] KS(int round, int[] key) {

int C1[] = new int[28];

int D1[] = new int[28];

int rotationTimes = (int) rotations[round];

C1 = leftShift(C, rotationTimes);

D1 = leftShift(D, rotationTimes);

int CnDn[] = new int[56];

System.arraycopy(C1, 0, CnDn, 0, 28);

System.arraycopy(D1, 0, CnDn, 28, 28);

int Kn[] = new int[48];

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

Kn[i] = CnDn[PC2[i]-1];

subkey[round] = Kn;

C = C1;

D = D1;

return Kn;

}

private static int[] fiestel(int[] R, int[] roundKey) {

int expandedR[] = new int[48];

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

expandedR[i] = R[E[i]-1];

int temp[] = xor(expandedR, roundKey);

int output[] = sBlock(temp);

return output;

}

private static int[] xor(int[] a, int[] b) {

int answer[] = new int[a.length];

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

answer[i] = a[i]^b[i];

return answer;

}

private static int[] sBlock(int[] bits) {

int output[] = new int[32];

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

int row[] = new int [2];

row[0] = bits[6\*i];

row[1] = bits[(6\*i)+5];

String sRow = row[0] + "" + row[1];

int column[] = new int[4];

column[0] = bits[(6\*i)+1];

column[1] = bits[(6\*i)+2];

column[2] = bits[(6\*i)+3];

column[3] = bits[(6\*i)+4];

String sColumn = column[0] +""+ column[1] +""+ column[2] +""+ column[3];

int iRow = Integer.parseInt(sRow, 2);

int iColumn = Integer.parseInt(sColumn, 2);

int x = S[i][(iRow\*16) + iColumn];

String s = Integer.toBinaryString(x);

while(s.length() < 4)

s = "0" + s;

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

output[(i\*4) + j] = Integer.parseInt(s.charAt(j) + "");

}

int finalOutput[] = new int[32];

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

finalOutput[i] = output[P[i]-1];

return finalOutput;

}

private static int[] leftShift(int[] bits, int n) {

int answer[] = new int[bits.length];

System.arraycopy(bits, 0, answer, 0, bits.length);

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

int temp = answer[0];

for(int j=0 ; j < bits.length-1 ; j++)

answer[j] = answer[j+1];

answer[bits.length-1] = temp;

}

return answer;

}

private static int[] hexToBits(String hexValue) {

int[] bits = new int[64];

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

String s = Integer.toBinaryString(Integer.parseInt(hexValue.charAt(i) + "", 16));

while(s.length() < 4)

s = "0" + s;

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

bits[(4\*i)+j] = Integer.parseInt(s.charAt(j) + "");

}

return bits;

}

private static int[] asciiToBits(String asciiValue) {

char[] chars = asciiValue.toCharArray();

StringBuffer hex = new StringBuffer();

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

hex.append(Integer.toHexString((int) chars[i]));

return hexToBits(hex.toString());

}

private static String hexToASCII(String hexValue) {

StringBuilder output = new StringBuilder("");

for (int i = 0; i < hexValue.length(); i += 2) {

String str = hexValue.substring(i, i + 2);

output.append((char) Integer.parseInt(str, 16));

}

return output.toString();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int choice;

DES2 des = new DES2();

do {

System.out.print("1. Encryption\n2. Decryption\n3. Exit\nEnter your choice: ");

choice = sc.nextInt();

sc.nextLine();

if(choice == 1) {

System.out.print("Enter plain text: ");

String plainText = sc.nextLine();

System.out.print("Enter the key hex value: ");

String keyText = sc.nextLine();

int[] keyBits = hexToBits(keyText);

int i = 0;

int n = plainText.length();

String encryption = "";

while(i < n) {

int[] inputBits;

if(i + 8 < n)

inputBits = asciiToBits(plainText.substring(i, i+8));

else

inputBits = asciiToBits(plainText.substring(i));

encryption += des.encryption(inputBits, keyBits);

i += 8;

}

des.display(false);

System.out.println("Encrypted Hex Value: " + encryption);

}

else if(choice == 2) {

System.out.print("Enter encrypted hex value: ");

String encryptedHex = sc.nextLine();

System.out.print("Enter key hex value: ");

String keyText = sc.nextLine();

int[] keyBits = hexToBits(keyText);

int i = 0;

int n = encryptedHex.length();

String decryption = "";

while(i < n) {

int[] inputBits;

if(i + 16 < n)

inputBits = hexToBits(encryptedHex.substring(i, i+16));

else

inputBits = hexToBits(encryptedHex.substring(i));

decryption += hexToASCII(des.decryption(inputBits, keyBits));

i += 16;

}

des.display(true);

System.out.println("Decrypted Text: " + decryption);

}

} while(choice != 3);

}

}

**Output:**



