

**AHSANULLAH UNIVERSITY OF SCIENCE AND TECHNOLOGY**

Department of Computer Science and Engineering

Program: Bachelor of Science in Computer Science and Engineering

Course Code: CSE 4174

Course Title: Cyber Security Lab

Academic Semester: Spring 2023

Assignment Topic: Data Encryption Standard (DES)

Submitted on:1st January 2024

Submitted by

Name: SHUVASHIS SARKER

Student ID: ***20200104116***

Lab Section: C1

**Question:**

1. **Devise a program for implementation of CFB-64 mode of DES..**

*Code: Here is a sample Java code that takes a plaintext and a key as inputs and performs 64 bit encryption and decryption with the DES in CFB mode:*

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package pkg20200104116\_des;

import java.util.HashMap;

import java.util.Map;

import java.util.Arrays;

public class DES {

public static String hex2Binary(String s) {

Map<Character, String> hexToBinaryMap = new HashMap<>();

hexToBinaryMap.put('0', "0000");

hexToBinaryMap.put('1', "0001");

hexToBinaryMap.put('2', "0010");

hexToBinaryMap.put('3', "0011");

hexToBinaryMap.put('4', "0100");

hexToBinaryMap.put('5', "0101");

hexToBinaryMap.put('6', "0110");

hexToBinaryMap.put('7', "0111");

hexToBinaryMap.put('8', "1000");

hexToBinaryMap.put('9', "1001");

hexToBinaryMap.put('A', "1010");

hexToBinaryMap.put('B', "1011");

hexToBinaryMap.put('C', "1100");

hexToBinaryMap.put('D', "1101");

hexToBinaryMap.put('E', "1110");

hexToBinaryMap.put('F', "1111");

StringBuilder binary = new StringBuilder();

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

binary.append(hexToBinaryMap.get(s.charAt(i)));

}

return binary.toString();

}

public static String binary2Hex(String s) {

Map<String, Character> binaryToHexMap = new HashMap<>();

binaryToHexMap.put("0000", '0');

binaryToHexMap.put("0001", '1');

binaryToHexMap.put("0010", '2');

binaryToHexMap.put("0011", '3');

binaryToHexMap.put("0100", '4');

binaryToHexMap.put("0101", '5');

binaryToHexMap.put("0110", '6');

binaryToHexMap.put("0111", '7');

binaryToHexMap.put("1000", '8');

binaryToHexMap.put("1001", '9');

binaryToHexMap.put("1010", 'A');

binaryToHexMap.put("1011", 'B');

binaryToHexMap.put("1100", 'C');

binaryToHexMap.put("1101", 'D');

binaryToHexMap.put("1110", 'E');

binaryToHexMap.put("1111", 'F');

StringBuilder hex = new StringBuilder();

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

String ch = s.substring(i, i + 4);

hex.append(binaryToHexMap.get(ch));

}

return hex.toString();

}

public static int binary2Decimal(int binary) {

int binary1 = binary;

int decimal = 0, i = 0;

while (binary != 0) {

int dec = binary % 10;

decimal = decimal + dec \* (int) Math.pow(2, i);

binary = binary / 10;

i++;

}

return decimal;

}

public static String decimal2Binary(int num) {

String binary = Integer.toBinaryString(num);

if (binary.length() % 4 != 0) {

int div = binary.length() / 4;

int counter = (4 \* (div + 1)) - binary.length();

StringBuilder paddedBinary = new StringBuilder();

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

paddedBinary.append('0');

}

paddedBinary.append(binary);

binary = paddedBinary.toString();

}

return binary;

}

public static String permute(String k, int[] arr, int n) {

StringBuilder permutation = new StringBuilder();

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

permutation.append(k.charAt(arr[i] - 1));

}

return permutation.toString();

}

public static String shiftLeft(String k, int nthShifts) {

for (int shift = 0; shift < nthShifts; shift++) {

StringBuilder s = new StringBuilder();

for (int j = 1; j < k.length(); j++) {

s.append(k.charAt(j));

}

s.append(k.charAt(0));

k = s.toString();

}

return k;

}

public static String xor(String a, String b) {

StringBuilder ans = new StringBuilder();

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

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

ans.append("0");

} else {

ans.append("1");

}

}

return ans.toString();

}

public static int[] initialPermutation = {

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 int[] expansionPermutation = {

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 int[] permutation = {

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 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}

}

};

public static int[] finalPermutation = {

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 String encryptCFB(String pt, String[] rkb, String[] rk, int[][][] sbox, int[] initialPermutation, int[] expansionPermutation, int[] permutation, int[] finalPermutation, String iv) {

pt = hex2Binary(pt);

// Initial Permutation

iv = hex2Binary(iv);

iv = permute(iv, initialPermutation, 64);

System.out.println("After initial permutation: " + binary2Hex(iv));

// Splitting

String left, right;

for (int i = 0; i < pt.length(); i += 64) {

left = iv.substring(0, 32);

right = iv.substring(32, 64);

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

// Expansion D-box: Expanding the 32 bits data into 48 bits

String rightExpanded = permute(right, expansionPermutation, 48);

// XOR RoundKey[i] and right\_expanded

String xorX = xor(rightExpanded, rkb[j]);

// S-boxes: substituting the value from s-box table by calculating row and column

String sBoxStr = "";

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

int row = binary2Decimal(Integer.parseInt(xorX.substring(k \* 6, k \* 6 + 1) + xorX.substring(k \* 6 + 5, k \* 6 + 6)));

int col = binary2Decimal(Integer.parseInt(xorX.substring(k \* 6 + 1, k \* 6 + 2) + xorX.substring(k \* 6 + 2, k \* 6 + 4) + xorX.substring(k \* 6 + 4, k \* 6 + 5)));

int val = sbox[k][row][col];

sBoxStr += decimal2Binary(val);

}

// Straight D-box: After substituting rearranging the bits

sBoxStr = permute(sBoxStr, permutation, 32);

// XOR left and sBoxStr

String result = xor(left, sBoxStr);

left = result;

// Swapper

if (j != 15) {

left = right;

right = result;

}

System.out.println("Round " + (j + 1) + " " + binary2Hex(left) + " " + binary2Hex(right) + " " + rk[j]);

}

// Combine and XOR with plaintext

String combine = left + right;

combine = xor(combine, pt.substring(i, i + 64));

// Final permutation: final rearranging of bits to get ciphertext

String ciphertextBlock = permute(combine, finalPermutation, 64);

// Update IV for the next iteration

iv = ciphertextBlock;

System.out.println("Ciphertext Block: " + binary2Hex(ciphertextBlock));

}

return iv; // return the last IV block

}

public static String decryptCFB(String ct, String[] rkb, String[] rk, int[][][] sbox, int[] initialPermutation, int[] expansionPermutation, int[] permutation, int[] finalPermutation, String iv) {

ct = hex2Binary(ct);

// Initial Permutation for IV

iv = hex2Binary(iv);

iv = permute(iv, initialPermutation, 64);

System.out.println("After initial permutation (IV): " + binary2Hex(iv));

// Splitting IV into left and right parts

String leftIV = iv.substring(0, 32);

String rightIV = iv.substring(32, 64);

// Splitting ciphertext

String left, right;

for (int i = 0; i < ct.length(); i += 64) {

// Perform DES encryption on IV

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

// Expansion D-box: Expanding the 32 bits data into 48 bits

String rightIVExpanded = permute(rightIV, expansionPermutation, 48);

// XOR RoundKey[i] and right\_expanded

String xorX = xor(rightIVExpanded, rkb[j]);

// S-boxes: substituting the value from s-box table by calculating row and column

String sBoxStr = "";

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

int row = binary2Decimal(Integer.parseInt(xorX.substring(k \* 6, k \* 6 + 1) + xorX.substring(k \* 6 + 5, k \* 6 + 6)));

int col = binary2Decimal(Integer.parseInt(xorX.substring(k \* 6 + 1, k \* 6 + 2) + xorX.substring(k \* 6 + 2, k \* 6 + 4) + xorX.substring(k \* 6 + 4, k \* 6 + 5)));

int val = sbox[k][row][col];

sBoxStr += decimal2Binary(val);

}

// Straight D-box: After substituting rearranging the bits

sBoxStr = permute(sBoxStr, permutation, 32);

// XOR leftIV and sBoxStr

String result = xor(leftIV, sBoxStr);

leftIV = result;

// Swapper

if (j != 15) {

leftIV = rightIV;

rightIV = result;

}

System.out.println("Round " + (j + 1) + " " + binary2Hex(leftIV) + " " + binary2Hex(rightIV) + " " + rk[j]);

}

// Combine and XOR with ciphertext

String combine = leftIV + rightIV;

combine = xor(combine, ct.substring(i, i + 64));

// Final permutation: final rearranging of bits to get plaintext

String plaintextBlock = permute(combine, finalPermutation, 64);

// Update IV for the next iteration

leftIV = ct.substring(i, i + 32);

rightIV = ct.substring(i + 32, i + 64);

//System.out.println("Plaintext Block: " + binary2Hex(plaintextBlock));

}

return iv; // return the last IV block

}

public static void main(String[] args) {

String pt = "123456ABCD132536";

String key = "AABB09182736CCDD";

String iv = "0123456789ABCDEF"; // Initialization Vector

// Key generation

key = hex2Binary(key);

// Parity bit drop table

int[] keyp = {

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

};

key = permute(key, keyp, 56);

int[] shiftTable = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};

int[] keyComp = {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};

String left = key.substring(0, 28);

String right = key.substring(28, 56);

String[] rkb = new String[16];

String[] rk = new String[16];

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

left = shiftLeft(left, shiftTable[i]);

right = shiftLeft(right, shiftTable[i]);

String combineStr = left + right;

String roundKey = permute(combineStr, keyComp, 48);

rkb[i] = roundKey;

rk[i] = binary2Hex(roundKey);

}

// Encryption (CFB Mode)

System.out.println("Encryption (CFB Mode)");

String ciphertext = binary2Hex(encryptCFB(pt, rkb, rk, sBox, initialPermutation, expansionPermutation, permutation, finalPermutation, iv));

System.out.println("Ciphertext: " + ciphertext);

// Decryption (CFB Mode)

System.out.println("\nDecryption (CFB Mode)");

String decryptedText = binary2Hex(decryptCFB(ciphertext, rkb, rk, sBox, initialPermutation, expansionPermutation, permutation, finalPermutation, iv));

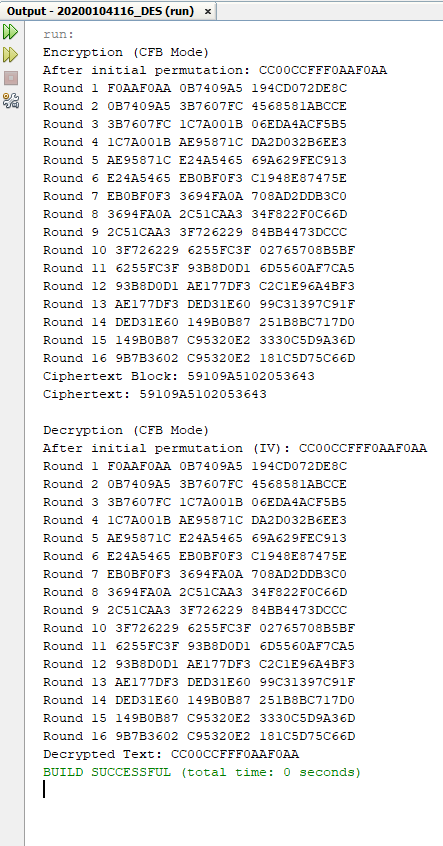
System.out.println("Decrypted Text: " + decryptedText);

}

}

// TODO code application logic here

**OUTPUT:**

****