

**Muhammad Hamees**

**BCSM-F19-206**

**SECTION-7D**

**INFORMATION SECURITY SUBMITTED TO: DR.IMRAN**

**ASSIGNMENT: 02**

**Implement DES algorithm using C++/Java Programming language.**

**Implement DES algorithm using C++.**

# Generating keys

The algorithm involves 16 rounds of encryption, with each round using a different key. Therefore, 16 keys are generated.

#include <iostream> #include <string> using namespace std; string round\_keys[16]; string shift\_left\_once(string key\_chunk){ string shifted=""; for(int i = 1; i < 28; i++){ shifted += key\_chunk[i];

}

shifted += key\_chunk[0]; return shifted;

}

string shift\_left\_twice(string key\_chunk){ string shifted=""; for(int i = 0; i < 2; i++){ for(int j = 1; j < 28; j++){

shifted += key\_chunk[j];

}

shifted += key\_chunk[0]; key\_chunk= shifted; shifted ="";

}

return key\_chunk;

}

void generate\_keys(string key){

int pc1[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

};

int pc2[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

};

string perm\_key ="";

for(int i = 0; i < 56; i++){ perm\_key+= key[pc1[i]-1];

}

string left= perm\_key.substr(0, 28); sring right= perm\_key.substr(28, 28); for(int i=0; i<16; i++){ if(i == 0 || i == 1 || i==8 || i==15 ){ left= shift\_left\_once(left); ight= shift\_left\_once(right);

}

else{

left= shift\_left\_twice(left); right= shift\_left\_twice(right);

}

string combined\_key = left + right; string round\_key = ""; for(int i = 0; i < 48; i++){

round\_key += combined\_key[pc2[i]-1];

}

round\_keys[i] = round\_key; cout<<"Key "<<i+1<<": "<<round\_keys[i]<<endl;

}

}

int main(){

string key = "10101010101110110000100100011000001001110011"

"01101100110011011101"; generate\_keys(key);

}

# Encrypting plain text to obtain ciphertext

Here, the entire algorithm is implemented. The plain text has been transposed, divided into two halves, and undergone 16 rounds of encryption. It has been combined and transposed again, which does the effect of the first transposed function to obtain the ciphertext.

#include <iostream>

#include <string> #include <cmath> using namespace std; string round\_keys[16]; string pt; string convertDecimalToBinary(int decimal)

{

string binary;

while(decimal != 0) {

binary = (decimal % 2 == 0 ? "0" : "1") + binary;

decimal = decimal/2;

}

while(binary.length() < 4){ binary = "0" + binary;

}

return binary;

}

int convertBinaryToDecimal(string binary)

{

int decimal = 0;

int counter = 0; int size = binary.length(); for(int i = size-1; i >= 0; i--)

{

if(binary[i] == '1'){ decimal += pow(2, counter); }

counter++;

}

return decimal;

}

string shift\_left\_once(string key\_chunk){ string shifted=""; for(int i = 1; i < 28; i++){ shifted += key\_chunk[i];

}

shifted += key\_chunk[0]; return shifted;

}

string shift\_left\_twice(string key\_chunk){ string shifted=""; for(int i = 0; i < 2; i++){ for(int j = 1; j < 28; j++){ shifted += key\_chunk[j];

}

shifted += key\_chunk[0]; key\_chunk= shifted; shifted ="";

}

return key\_chunk;

}

string Xor(string a, string b){

string result = ""; int size = b.size(); for(int i = 0; i < size; i++){

if(a[i] != b[i]){

result += "1";

}

else{ result += "0";

}

}

return result;

}

void generate\_keys(string key){

int pc1[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

};

int pc2[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

};

string perm\_key ="";

for(int i = 0; i < 56; i++){ perm\_key+= key[pc1[i]-1];

}

string left= perm\_key.substr(0, 28); string right= perm\_key.substr(28, 28);

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

if(i == 0 || i == 1 || i==8 || i==15 ){ left= shift\_left\_once(left); right= shift\_left\_once(right);

} else{

left= shift\_left\_twice(left); right= shift\_left\_twice(right);

}

string combined\_key = left + right; string round\_key = ""; for(int i = 0; i < 48; i++){ round\_key += combined\_key[pc2[i]-1];

}

round\_keys[i] = round\_key;

}

}

string DES(){ int initial\_permutation[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

};

int expansion\_table[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

};

int substition\_boxes[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

}};

int permutation\_tab[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

};

int inverse\_permutation[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

};

string perm = ""; for(int i = 0; i < 64; i++){ perm += pt[initial\_permutation[i]-1];

}

string left = perm.substr(0, 32); string right = perm.substr(32, 32); for(int i=0; i<16; i++) { string right\_expanded = ""; for(int i = 0; i < 48; i++) { right\_expanded += right[expansion\_table[i]-1]; };

string xored = Xor(round\_keys[i], right\_expanded); string res = ""; for(int i=0;i<8; i++){ string row1= xored.substr(i\*6,1) + xored.substr(i\*6 + 5,1); int row = convertBinaryToDecimal(row1);

string col1 = xored.substr(i\*6 + 1,1) + xored.substr(i\*6 + 2,1) + xored.substr(i\*6 +

3,1) + xored.substr(i\*6 + 4,1);;

int col = convertBinaryToDecimal(col1); int val = substition\_boxes[i][row][col]; res += convertDecimalToBinary(val);

}

string perm2 ="";

for(int i = 0; i < 32; i++){ perm2 += res[permutation\_tab[i]-1];

}

xored = Xor(perm2, left); left = xored; if(i < 15){ string temp = right; right = xored; left = temp;

}

}

string combined\_text = left + right; string ciphertext =""; for(int i = 0; i < 64; i++){ ciphertext+= combined\_text[inverse\_permutation[i]-1];

}

return ciphertext;

}

int main(){

string key=

"1010101010111011000010010001100000100111001101101100110011011101";

pt=

"1010101111001101111001101010101111001101000100110010010100110110"; generate\_keys(key);

cout<<"Plain text: "<<pt<<endl; string ct= DES(); cout<<"Ciphertext: "<<ct<<endl;

}

# Decrypting ciphertext to obtain plain text

|  |  |  |
| --- | --- | --- |
| To decrypt the ciphertext, reverse the order of the keys in | round keys | (i.e., key 16 becomes key |
| 1, and so on) and apply the DES() function again. | | |

#include <iostream>

#include <string> #include <cmath> using namespace std; string round\_keys[16]; string pt; string convertDecimalToBinary(int decimal)

{

string binary; while(decimal != 0) { binary = (decimal % 2 == 0 ? "0" : "1") + binary; decimal = decimal/2;

}

while(binary.length() < 4){ binary = "0" + binary;

}

return binary;

}

int convertBinaryToDecimal(string binary)

{

int decimal = 0; int counter = 0; int size = binary.length();

for(int i = size-1; i >= 0; i--)

{

if(binary[i] == '1'){ decimal += pow(2, counter);

}

counter++;

}

return decimal;

}

string shift\_left\_once(string key\_chunk){ string shifted=""; for(int i = 1; i < 28; i++){ shifted += key\_chunk[i];

}

shifted += key\_chunk[0]; return shifted;

}

string shift\_left\_twice(string key\_chunk){ string shifted=""; for(int i = 0; i < 2; i++){ for(int j = 1; j < 28; j++){ shifted += key\_chunk[j];

}

shifted += key\_chunk[0]; key\_chunk= shifted; shifted ="";

}

return key\_chunk;

}

string Xor(string a, string b){

string result = ""; int size = b.size(); for(int i = 0; i < size; i++){

if(a[i] != b[i]){

result += "1";

}

else{ result += "0";

}

}

return result;

}

void generate\_keys(string key){ int pc1[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

};

int pc2[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

};

string perm\_key ="";

for(int i = 0; i < 56; i++){ perm\_key+= key[pc1[i]-1];

}

string left= perm\_key.substr(0, 28); string right= perm\_key.substr(28, 28); for(int i=0; i<16; i++){ if(i == 0 || i == 1 || i==8 || i==15 ){ left= shift\_left\_once(left); right= shift\_left\_once(right);

} else{

left= shift\_left\_twice(left); right= shift\_left\_twice(right);

}

string combined\_key = left + right; string round\_key = ""; for(int i = 0; i < 48; i++){ round\_key += combined\_key[pc2[i]-1];

}

round\_keys[i] = round\_key;

}

}

string DES(){ int initial\_permutation[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

};

int expansion\_table[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

};

int substition\_boxes[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

}};

int permutation\_tab[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

};

int inverse\_permutation[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

};

string perm = ""; for(int i = 0; i < 64; i++){ perm += pt[initial\_permutation[i]-1];

}

string left = perm.substr(0, 32); string right = perm.substr(32, 32); for(int i=0; i<16; i++) { string right\_expanded = ""; for(int i = 0; i < 48; i++) { right\_expanded += right[expansion\_table[i]-1];

};

string xored = Xor(round\_keys[i], right\_expanded);

string res = ""; for(int i=0;i<8; i++){ string row1= xored.substr(i\*6,1) + xored.substr(i\*6 + 5,1); int row = convertBinaryToDecimal(row1);

string col1 = xored.substr(i\*6 + 1,1) + xored.substr(i\*6 + 2,1) + xored.substr(i\*6 + 3,1) + xored.substr(i\*6 + 4,1);;

int col = convertBinaryToDecimal(col1); int val = substition\_boxes[i][row][col]; res += convertDecimalToBinary(val);

}

string perm2 ="";

for(int i = 0; i < 32; i++){ perm2 += res[permutation\_tab[i]-1];

}

xored = Xor(perm2, left); left = xored; if(i < 15){ string temp = right; right = xored; left = temp;

}

}

string combined\_text = left + right; string ciphertext =""; for(int i = 0; i < 64; i++){ ciphertext+= combined\_text[inverse\_permutation[i]-1]; }

return ciphertext;

}

int main(){

string key=

"1010101010111011000010010001100000100111001101101100110011011101"; pt= "1010101111001101111001101010101111001101000100110010010100110110"; string apt = pt;

generate\_keys(key); cout<<"Plain text: "<<pt<<endl; string ct= DES(); cout<<"Ciphertext: "<<ct<<endl;

int i = 15; int j = 0; while(i > j)

{

string temp = round\_keys[i]; round\_keys[i] = round\_keys[j]; round\_keys[j] = temp;

i--; j++;

}

pt = ct;

string decrypted = DES(); cout<<"Decrypted text:"<<decrypted<<endl; if (decrypted == apt){ cout<<"Plain text encrypted and decrypted successfully."<<endl;

}

}

**Implement DES algorithm using Java.**

# Generating Keys

The algorithm performs 16 rounds of encryption and for each round, a unique key is generated. Before moving to the steps, it is important to know that in plaintext the bits are labeled from 1 to 64 where 1 is the most significant bit and 64 is the least significant bit.

nt pc1[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

};

int pc2[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

};

# DesProgram.java

import java.io.FileInputStream; import java.io.FileOutputStream; import java.io.IOException; import java.io.InputStream; import java.io.OutputStream; import java.security.InvalidAlgorithmParameterException; import java.security.InvalidKeyException; import java.security.NoSuchAlgorithmException; import java.security.spec.AlgorithmParameterSpec; import javax.crypto.Cipher; import javax.crypto.CipherInputStream; import javax.crypto.CipherOutputStream; import javax.crypto.KeyGenerator; import javax.crypto.NoSuchPaddingException; import javax.crypto.SecretKey;

import javax.crypto.spec.IvParameterSpec; public class DesProgram

{

private static Cipher encrypt; private static Cipher decrypt; private static final byte[] initialization\_vector = { 22, 33, 11, 44, 55, 99, 66, 77 }; public static void main(String[] args)

{

String textFile = "C:/Users/Anubhav/Desktop/DemoData.txt";

String encryptedData = "C:/Users/Anubhav/Desktop/encrypteddata.txt";

String decryptedData = "C:/Users/Anubhav/Desktop/decrypteddata.txt";

try

{

SecretKey scrtkey = KeyGenerator.getInstance("DES").generateKey(); AlgorithmParameterSpec aps = new IvParameterSpec(initialization\_vector); encrypt = Cipher.getInstance("DES/CBC/PKCS5Padding"); encrypt.init(Cipher.ENCRYPT\_MODE, scrtkey, aps); decrypt = Cipher.getInstance("DES/CBC/PKCS5Padding"); decrypt.init(Cipher.DECRYPT\_MODE, scrtkey, aps); encryption(new FileInputStream(textFile), new FileOutputStream(encryptedData)); decryption(new FileInputStream(encryptedData), new FileOutputStream(decryptedData));

System.out.println("The encrypted and decrypted files have been created successfully.");

}

catch (NoSuchAlgorithmException | NoSuchPaddingException | InvalidKeyException | InvalidAlgorithmParameterException | IOException e)

{

//prints the message (if any) related to exceptions

e.printStackTrace();

}

}

private static void encryption(InputStream input, OutputStream output) throws IOException

{

output = new CipherOutputStream(output, encrypt); writeBytes(input, output);

}

private static void decryption(InputStream input, OutputStream output) throws IOException

{

input = new CipherInputStream(input, decrypt); writeBytes(input, output);

}

private static void writeBytes(InputStream input, OutputStream output) throws IOException

{

byte[] writeBuffer = new byte[512];

int readBytes = 0; while ((readBytes = input.read(writeBuffer)) >= 0)

{

output.write(writeBuffer, 0, readBytes);

}

output.close(); input.close();

}

}