**7. Take a 64-bit plain text and encrypt the same using DES algorithm.**

#include <iostream >

using namespace std;

string hex2bin(string s)

{

unordered\_map<char, string> mp;

mp['0'] = "0000";

mp['1'] = "0001";

mp['2'] = "0010";

mp['3'] = "0011";

mp['4'] = "0100";

mp['5'] = "0101";

mp['6'] = "0110";

mp['7'] = "0111";

mp['8'] = "1000";

mp['9'] = "1001";

mp['A'] = "1010";

mp['B'] = "1011";

mp['C'] = "1100";

mp['D'] = "1101";

mp['E'] = "1110";

mp['F'] = "1111";

string bin = "";

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

bin += mp[s[i]];

}

return bin;

}

string bin2hex(string s)

{

// binary to hexadecimal conversion

unordered\_map<string, string> mp;

mp["0000"] = "0";

mp["0001"] = "1";

mp["0010"] = "2";

mp["0011"] = "3";

mp["0100"] = "4";

mp["0101"] = "5";

mp["0110"] = "6";

mp["0111"] = "7";

mp["1000"] = "8";

mp["1001"] = "9";

mp["1010"] = "A";

mp["1011"] = "B";

mp["1100"] = "C";

mp["1101"] = "D";

mp["1110"] = "E";

mp["1111"] = "F";

string hex = "";

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

string ch = "";

ch += s[i];

ch += s[i + 1];

ch += s[i + 2];

ch += s[i + 3];

hex += mp[ch];

}

return hex;

}

string permute(string k, int\* arr, int n)

{

string per = "";

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

per += k[arr[i] - 1];

}

return per;

}

string shift\_left(string k, int shifts)

{

string s = "";

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

for (int j = 1; j < 28; j++) {

s += k[j];

}

s += k[0];

k = s;

s = "";

}

return k;

}

string xor\_(string a, string b)

{

string ans = "";

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

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

ans += "0";

}

else {

ans += "1";

}

}

return ans;

}

string encrypt(string pt, vector<string> rkb,

vector<string> rk)

{

pt = hex2bin(pt);

int initial\_perm[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 };

pt = permute(pt, initial\_perm, 64);

cout << "After initial permutation: " << bin2hex(pt)

<< endl;

string left = pt.substr(0, 32);

string right = pt.substr(32, 32);

cout << "After splitting: L0=" << bin2hex(left)

<< " R0=" << bin2hex(right) << endl;

int exp\_d[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 s[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};

int per[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 };

cout << endl;

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

string right\_expanded = permute(right, exp\_d, 48);

string x = xor\_(rkb[i], right\_expanded);

string op = "";

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

int row = 2 \* int(x[i \* 6] - '0')

+ int(x[i \* 6 + 5] - '0');

int col = 8 \* int(x[i \* 6 + 1] - '0')

+ 4 \* int(x[i \* 6 + 2] - '0')

+ 2 \* int(x[i \* 6 + 3] - '0')

+ int(x[i \* 6 + 4] - '0');

int val = s[i][row][col];

op += char(val / 8 + '0');

val = val % 8;

op += char(val / 4 + '0');

val = val % 4;

op += char(val / 2 + '0');

val = val % 2;

op += char(val + '0');

}

op = permute(op, per, 32);

x = xor\_(op, left);

left = x;

if (i != 15) {

swap(left, right);

}

cout << "Round " << i + 1 << " " << bin2hex(left)

<< " " << bin2hex(right) << " " << rk[i]

<< endl;

}

string combine = left + right;

int final\_perm[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 cipher

= bin2hex(permute(combine, final\_perm, 64));

return cipher;

}

int main()

{

string pt, key;

pt = "123456ABCD132536";

key = "AABB09182736CCDD";

key = hex2bin(key);

int keyp[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 };

key = permute(key, keyp, 56); // key without parity

int shift\_table[16] = { 1, 1, 2, 2, 2, 2, 2, 2,

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

int key\_comp[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 left = key.substr(0, 28);

string right = key.substr(28, 28);

vector<string> rkb;

vector<string> rk;

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

left = shift\_left(left, shift\_table[i]);

right = shift\_left(right, shift\_table[i]);

string combine = left + right;

string RoundKey = permute(combine, key\_comp, 48);

rkb.push\_back(RoundKey);

rk.push\_back(bin2hex(RoundKey));

}

cout << "\nEncryption:\n\n";

string cipher = encrypt(pt, rkb, rk);

cout << "\nCipher Text: " << cipher << endl;

cout << "\nDecryption\n\n";

reverse(rkb.begin(), rkb.end());

reverse(rk.begin(), rk.end());

string text = encrypt(cipher, rkb, rk);

cout << "\nPlain Text: " << text << endl;

}

**OUTPUT:**

Encryption:

After initial permutation: 14A7D67818CA18AD

After splitting: L0=14A7D678 R0=18CA18AD

Round 1 18CA18AD 5A78E394 194CD072DE8C

Round 2 5A78E394 4A1210F6 4568581ABCCE

Round 3 4A1210F6 B8089591 06EDA4ACF5B5

Round 4 B8089591 236779C2 DA2D032B6EE3

Round 5 236779C2 A15A4B87 69A629FEC913

Round 6 A15A4B87 2E8F9C65 C1948E87475E

Round 7 2E8F9C65 A9FC20A3 708AD2DDB3C0

Round 8 A9FC20A3 308BEE97 34F822F0C66D

Round 9 308BEE97 10AF9D37 84BB4473DCCC

Round 10 10AF9D37 6CA6CB20 02765708B5BF

Round 11 6CA6CB20 FF3C485F 6D5560AF7CA5

Round 12 FF3C485F 22A5963B C2C1E96A4BF3

Round 13 22A5963B 387CCDAA 99C31397C91F

Round 14 387CCDAA BD2DD2AB 251B8BC717D0

Round 15 BD2DD2AB CF26B472 3330C5D9A36D

Round 16 19BA9212 CF26B472 181C5D75C66D

Cipher Text: C0B7A8D05F3A829C

Decryption

After initial permutation: 19BA9212CF26B472

After splitting: L0=19BA9212 R0=CF26B472

Round 1 CF26B472 BD2DD2AB 181C5D75C66D

Round 2 BD2DD2AB 387CCDAA 3330C5D9A36D

Round 3 387CCDAA 22A5963B 251B8BC717D0

Round 4 22A5963B FF3C485F 99C31397C91F

Round 5 FF3C485F 6CA6CB20 C2C1E96A4BF3

Round 6 6CA6CB20 10AF9D37 6D5560AF7CA5

Round 7 10AF9D37 308BEE97 02765708B5BF

Round 8 308BEE97 A9FC20A3 84BB4473DCCC

Round 9 A9FC20A3 2E8F9C65 34F822F0C66D

Round 10 2E8F9C65 A15A4B87 708AD2DDB3C0

Round 11 A15A4B87 236779C2 C1948E87475E

Round 12 236779C2 B8089591 69A629FEC913

Round 13 B8089591 4A1210F6 DA2D032B6EE3

Round 14 4A1210F6 5A78E394 06EDA4ACF5B5

Round 15 5A78E394 18CA18AD 4568581ABCCE

Round 16 14A7D678 18CA18AD 194CD072DE8C

Plain Text: 123456ABCD132536

**8. Using RSA algorithm encrypt a text data and decrypt the same.**

#include <bits/stdc++.h>

using namespace std;

int gcd(int a, int h)

{

int temp;

while (1) {

temp = a % h;

if (temp == 0)

return h;

a = h;

h = temp;

}

}

int main()

{

double p = 3;

double q = 7;

double n = p \* q;

double e = 2;

double phi = (p - 1) \* (q - 1);

while (e < phi) {

if (gcd(e, phi) == 1)

break;

else

e++;

}

int k = 2;

double d = (1 + (k \* phi)) / e;

double msg = 12;

printf("Message data = %lf", msg);

double c = pow(msg, e);

c = fmod(c, n);

printf("\nEncrypted data = %lf", c);

double m = pow(c, d);

m = fmod(m, n);

printf("\nOriginal Message Sent = %lf", m);

return 0;

}

**OUTPUT:**

Message data = 12.000000

Encrypted data = 3.000000

Original Message Sent = 12.000000