**Lab 2**

**Q.** **1.** **Write a program to obtain the output as the Cipher text using**

**transposition cipher techniques for the following key and**

**plaintext:**

**Sol-**

#include<bits/stdc++.h>

using namespace std;

bool comp(const vector<char> & x,const vector<char> & y)

{

return (x[0]<y[0]);

}

int main()

{

int i=0,j=1;

int key[]={3,4,2,1,5,6,7};

vector< vector <char> > plaintext(7);

for(int k=0;k<7;k++)

{

plaintext[k]=vector<char>(5);

}

plaintext[0][0]='3';

plaintext[1][0]='4';

plaintext[2][0]='2';

plaintext[3][0]='1';

plaintext[4][0]='5';

plaintext[5][0]='6';

plaintext[6][0]='7';

string text;

text="attackpostponeduntiltwoam";

string::iterator it;

for(it=text.begin();it!=text.end();it++)

{

if(\*it!=' ')

{

plaintext[i][j]=\*it;

if(i%7==6)

{

j++;

i=0;

}

else

{

i++;

}

}

}

for(i;i<7;i++)

{

for(j;j<5;j++)

{

plaintext[i][j]='$';

}

}

sort(plaintext.begin(),plaintext.end(),comp);

cout<<"\nCyphertext is: ";

for(i=0;i<7;i++)

{

for(j=1;j<5;j++)

{

cout<<plaintext[i][j];

}

}

cout<<"\nDecyphered text is: ";

cout<<text;

}

**Q.** **2. Write a program to implement Caesar cipher for the following**

**plaintext using key k=3:**

**which generates the output:**

**Sol-**

#include <iostream>

using namespace std;

string encrypt(string text, int s)

{

string result = "";

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

{

if (isupper(text[i]))

result += char(int(text[i]+s-65)%26 +65);

else

result += char(int(text[i]+s-97)%26 +97);

}

return result;

}

int main()

{

string text;

cout<<"Enter text: ";

getline(cin,text);

int s;

cout<<"Enter the value of shift: ";

cin>>s;

cout << "\nCipher: " << encrypt(text, s);

return 0;

}

**Lab 3**

**Q.** **1. Write a program to implement polyalphabetic cipher for the**

**following plaintext and key:**

**Sol-**

#include<bits/stdc++.h>

using namespace std;

int main()

{

char v\_m[26][26];

int k=97;

string k1="deceptive";

string key,plaintext,ciphertext,text;

int l=97;

for(int i=0;i<26;i++)

{

k=l;

for(int j=0;j<26;j++)

{

v\_m[i][j]=(char)k;

k++;

if(k>122)

k=97;

}

l++;

}

plaintext="wearediscoveredsaveyourselftoday";

k=0;

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

{

key.push\_back(k1[k]);

k++;

if(k>8)

k=0;

}

int row,col;

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

{

row=((int)key[i])%97;

col=((int)plaintext[i])%97;

ciphertext.push\_back(v\_m[row][col]);

}

cout<<"\nAfter encrypting the plaintext-\n";

cout<<"\nKey is : "<<k1;

cout<<"\nCipher text is : "<<ciphertext;

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

{

row=((int)key[i])%97;

for(int j=0;j<26;j++)

{

if(v\_m[row][j]==ciphertext[i])

{

k=j+97;

break;

}

}

text.push\_back((char)k);

}

cout<<"\n\nAfter decrypting the ciphertext-\n";

cout<<"\nKey is : "<<k1;

cout<<"\nDecrypted text is : "<<text;

}

**Q. 2. Write a program to implement Hill cipher for the following**

**plaintext: “pay” and key:**

**Sol-**

#include <bits/stdc++.h>

using namespace std;

void getKeyMatrix(string key, int keyMatrix[][3])

{

int k = 0;

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

keyMatrix[i][j] = (key[k]) % 65;

k++;

}

}

}

void encrypt(int cipherMatrix[][1],

int keyMatrix[][3],

int messageVector[][1])

{

int x, i, j;

for (i = 0; i < 3; i++)

{

for (j = 0; j < 1; j++)

{

cipherMatrix[i][j] = 0;

for (x = 0; x < 3; x++)

{

cipherMatrix[i][j] +=

keyMatrix[i][x] \* messageVector[x][j];

}

cipherMatrix[i][j] = cipherMatrix[i][j] % 26;

}

}

}

void HillCipher(string message, string key)

{

int keyMatrix[3][3];

getKeyMatrix(key, keyMatrix);

int messageVector[3][1];

for (int i = 0; i < 3; i++)

{messageVector[i][0] = (message[i]) % 65;}

int cipherMatrix[3][1];

encrypt(cipherMatrix, keyMatrix, messageVector);

string CipherText;

for (int i = 0; i < 3; i++)

CipherText += cipherMatrix[i][0] + 65;

cout << " Ciphertext:" << CipherText;

}

int main()

{

string message = "hello world";

string key = "computer";

HillCipher(message, key);

return 0;

}

**Lab 4**

**Monoalphabetic Cipher :-**

#include<bits/stdc++.h>

using namespace std;

string encoder(string key)

{

string encoded = "";

bool arr[26] = {0};

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

{

if(key[i] >= 'A' && key[i] <= 'Z')

{

if (arr[key[i]-65] == 0)

{

encoded += key[i];

arr[key[i]-65] = 1;

}

}

else if (key[i] >= 'a' && key[i] <= 'z')

{

if (arr[key[i]-97] == 0)

{

encoded += key[i] - 32;

arr[key[i]-97] = 1;

}

}

}

for (int i=0; i<26; i++)

{

if(arr[i] == 0)

{

arr[i]=1;

encoded += char(i + 65);

}

}

return encoded;

}

string cipheredIt(string msg, string encoded)

{

string cipher="";

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

{

if (msg[i] >='a' && msg[i] <='z')

{

int pos = msg[i] - 97;

cipher += encoded[pos];

}

else if (msg[i] >='A' && msg[i] <='Z')

{

int pos = msg[i] - 65;

cipher += encoded[pos];

}

else

{

cipher += msg[i];

}

}

return cipher;

}

int main()

{

string key;

key = "Computer";

string encoded = encoder(key);

string message = "Attack on Titan";

cout << "Message before Ciphering : " << message << endl;

cout << "Ciphered Text : " << cipheredIt(message,encoded) << endl;

return 0;

}

Execution Time : 0.218 sec

**Caesar Cipher :-**

// A C++ program to illustrate Caesar Cipher Technique

#include <iostream>

using namespace std;

string encrypt(string text, int s)

{

string result = "";

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

{

if (isupper(text[i]))

result += char(int(text[i]+s-65)%26 +65);

else

result += char(int(text[i]+s-97)%26 +97);

}

return result;

}

int main()

{

string text="Attack on Titan";

int s=3;

cout << "\nCipher: " << encrypt(text, s);

return 0;

}

Execution Time : 0.265 sec

**Polyalphabetic Cipher :-**

#include<bits/stdc++.h>

using namespace std;

char mat[26][26];

void encrp(string s,string plaintext)

{

string key="";

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

{

key+=s;

}

if(key.length()<plaintext.length())

{

int i=0;

while(key.length()<plaintext.length())

{

key.append(1,s[i]);

i++;

}

}

// cout<<key;

string ans;

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

{

ans.append(1,mat[key[i]-97][plaintext[i]-97]);

}

cout<<ans<<endl;

}

int main()

{

for(int i=0;i<26;i++)

{

for(int j=0;j<26;j++)

{

mat[i][j]=97+((i+j)%26);

//cout<<mat[i][j]<<" ";

}

//cout<<endl;

}

string key="Computer";

string plaintext="Attack on Titan";

encrp(key,plaintext);

}

Execution Time : 0.231 sec

**Hill Cipher :**

#include <iostream>

using namespace std;

void encrypt(int cipherMatrix[][1],

int keyMatrix[][3],

int messageVector[][1])

{

int x, i, j;

for (i = 0; i < 3; i++)

{

for (j = 0; j < 1; j++)

{

cipherMatrix[i][j] = 0;

for (x = 0; x < 3; x++)

{

cipherMatrix[i][j] +=

keyMatrix[i][x] \* messageVector[x][j];

}

cipherMatrix[i][j] = cipherMatrix[i][j] % 26;

}

}

}

void HillCipher(string message, int keyMatrix[][3])

{

int messageVector[3][1];

for (int i = 0; i < 3; i++)

messageVector[i][0] = (message[i]) % 65;

int cipherMatrix[3][1];

int x, i, j;

for (i = 0; i < 3; i++)

{

for (j = 0; j < 1; j++)

{

cipherMatrix[i][j] = 0;

for (x = 0; x < 3; x++)

{

cipherMatrix[i][j] +=

keyMatrix[i][x] \* messageVector[x][j];

}

cipherMatrix[i][j] = cipherMatrix[i][j] % 26;

}

}

string CipherText;

for (int i = 0; i < 3; i++)

CipherText += cipherMatrix[i][0] + 65;

cout << " Ciphertext:" << CipherText;

}

int main()

{

string message = "Att";

int key[3][3] = {{17,17,5},

{21,18,21},

{2,2,19}};

HillCipher(message, key);

return 0;

}

Execution time: 0.235 sec

**One Time Pad :-**

#include<iostream>

#include<vector>

#include<stdlib.h>

using namespace std;

void to\_upper\_case(vector<char>& text, int len)

{

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

{

if (text[i] >= 97 && text[i] <= 122)

text[i] -= 32;

}

}

void print\_string(vector<char> text, int len)

{

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

{

cout << (char) (text[i] + 65);

}

cout << endl;

return;

}

int main()

{

vector<char> msg;

string s="Attack on Titan";

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

{

msg.push\_back(s[i]);

}

vector<char> enc\_msg;

//vector<char> dec\_msg;

int \*p;

int i;

size\_t len=s.length();

to\_upper\_case(msg, len);

p = (int\*) malloc(msg.size() \* sizeof(int));

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

{

p[i] = rand() % 26;

if (msg[i] >= 65 && msg[i] <= 90)

enc\_msg.push\_back((char) ((msg[i] - 65 + p[i]) % 26));

else if (msg[i] >= 97 && msg[i] <= 122)

enc\_msg.push\_back((char) ((msg[i] - 97 + p[i]) % 26));

else

enc\_msg.push\_back((char) msg[i]);

}

cout << "\nEncoded Message:";

print\_string(enc\_msg, len);

cout << "\nKey for decryption:\n";

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

{

cout << (char) (p[i] + 65);

}

cout << endl;

return 0;

}

Execution time : 0.257 sec

**Rail Fence:-**

// C++ program to illustrate Rail Fence Cipher

// Encryption

#include <bits/stdc++.h>

using namespace std;

string encryptRailFence(string text, int key)

{

char rail[key][(text.length())];

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

for (int j = 0; j < text.length(); j++)

rail[i][j] = '\n';

bool dir\_down = false;

int row = 0, col = 0;

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

{

if (row == 0 || row == key-1)

dir\_down = !dir\_down;

rail[row][col++] = text[i];

dir\_down?row++ : row--;

}

string result;

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

for (int j=0; j < text.length(); j++)

if (rail[i][j]!='\n')

result.push\_back(rail[i][j]);

return result;

}

//driver program to check the above functions

int main()

{

cout << encryptRailFence("Attack on Titan", 3) << endl;

return 0;

}

Execution time: 0.259 sec

**Play Fair :-**

#include<bits/stdc++.h>

using namespace std;

char grid[5][5]; // 5x5 matrix to encipher or decipher

char keyword[9]={'C','o','m','p','u','t','e','r'}; // cypher key

char msg[16]={'A','t','t','a','c','k',' ','o','n',' ','T','i','t','a','n'}; // message

int mark[130],len,r,c; // necessary variables

void createGrid();

void showGrid();

void encipher();

int main()

{

encipher();

return 0;

}

void encipher()

{

memset(mark,0,sizeof(mark));

createGrid();

cout<<"Message to cypher: ";

int l=15; // msg length

char reqText[150]; //generate required text to encipher

int in=0,j=0;

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

{

j=i+1;

if(msg[i]==' ') //ignore all space from string

{

i++;

j++;

}

if(msg[j]==' ') j++; //ignore space

if(toupper(msg[i])=='J') msg[i]='i'; // ignore J

if(toupper(msg[i])==toupper(msg[j])) // if duplicate add 'X' after the first letter

{

reqText[in]=toupper(msg[i]);

reqText[in+1]='X';

in++;

}

else

{

reqText[in]=toupper(msg[i]);

}

in++;

}

if(in%2!=0) reqText[in]='X'; // if one character left, add 'X' after it

cout<<"Cypher text: ";

//encipher starts

int P,Q,R,S,f1,f2;

char x,y;

for(int i=0;i<in;i+=2)

{

x=reqText[i];

y=reqText[i+1];

f1=f2=0;

for(int j=0;j<5;j++)

{

for(int k=0;k<5;k++)

{

if(x==grid[j][k])

{

P=j;

Q=k;

f1=1;

}

if(y==grid[j][k])

{

R=j;

S=k;

f2=1;

}

if(f1 && f2) break;

}

if(f1 && f2) break;

}

if(P==R) //same row

{

if(Q==4) cout<<grid[P][0];

else cout<<grid[P][Q+1];

if(S==4) cout<<grid[R][0];

else cout<<grid[R][S+1];

}

else if(Q==S ) // same column

{

if(P==4) cout<<grid[0][Q];

else cout<<grid[P+1][Q];

if(R==4) cout<<grid[0][S];

else cout<<grid[R+1][S];

}

else //opposite corner

{

cout<<grid[P][S]<<grid[R][Q];

}

}

cout<<endl<<endl;

//system("PAUSE");

// menu();

}

void createGrid()

{

cout<<"Keyword: "<<keyword<<endl;

len=15; // size of input string O(n) :3

mark['J']=1; // ignore J

r=0,c=0; //initialize row and column

// first populate the keyword

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

{

if(!mark[toupper(keyword[i])]) // ignore duplicates

{

mark[toupper(keyword[i])]=1;

grid[r][c++]=toupper(keyword[i]);

if(c%5==0) //increase row column

{

c=0;

r++;

}

}

}

for(int i='A';i<='Z';i++)

{

if(mark[i]==0)

{

grid[r][c++]=i;

mark[i]=1;

if(c%5==0)

{

if(r==4 && c==5) break;

r++;

c=0;

}

}

}

}

void showGrid()

{

cout<<"5x5 Matrix"<<endl;

//show grid

for(int i=0;i<5;i++)

{

for(int j=0;j<5;j++)

{

cout<<grid[i][j]<<" ";

}

cout<<endl;

}

}

Execution time: 0.2 sec

**LAB 6**

**1. Write a program to convert the given text into hexadecimal representation:**

**Input: “Your lips are smoother than vaseline”**

**Output: 596F7572206C6970 732061726520736D 6F6F746865722074 68616E2076617365 6C696E650D0A**

**CODE:**

#include<bits/stdc++.h>

using namespace std;

void string2hex(const string str, string& hexstr, bool capital = true) {

hexstr.resize((str.size() \* 2));

const size\_t a = capital ? 'A' - 1 : 'a' - 1;

for (size\_t i = 0, c = str[0] & 0xFF; i < hexstr.size(); c = str[i / 2] & 0xFF) {

hexstr[i++] = c > 0x9F ? (c / 16 - 9) | a : c / 16 | '0';

hexstr[i++] = (c & 0xF) > 9 ? (c % 16 - 9) | a : c % 16 | '0';

}

}

int main(){

string s = "Your lips are smoother than vaseline";

cout << "\n\tOriginal String: " << s << '\n';

string2hex(s, s);

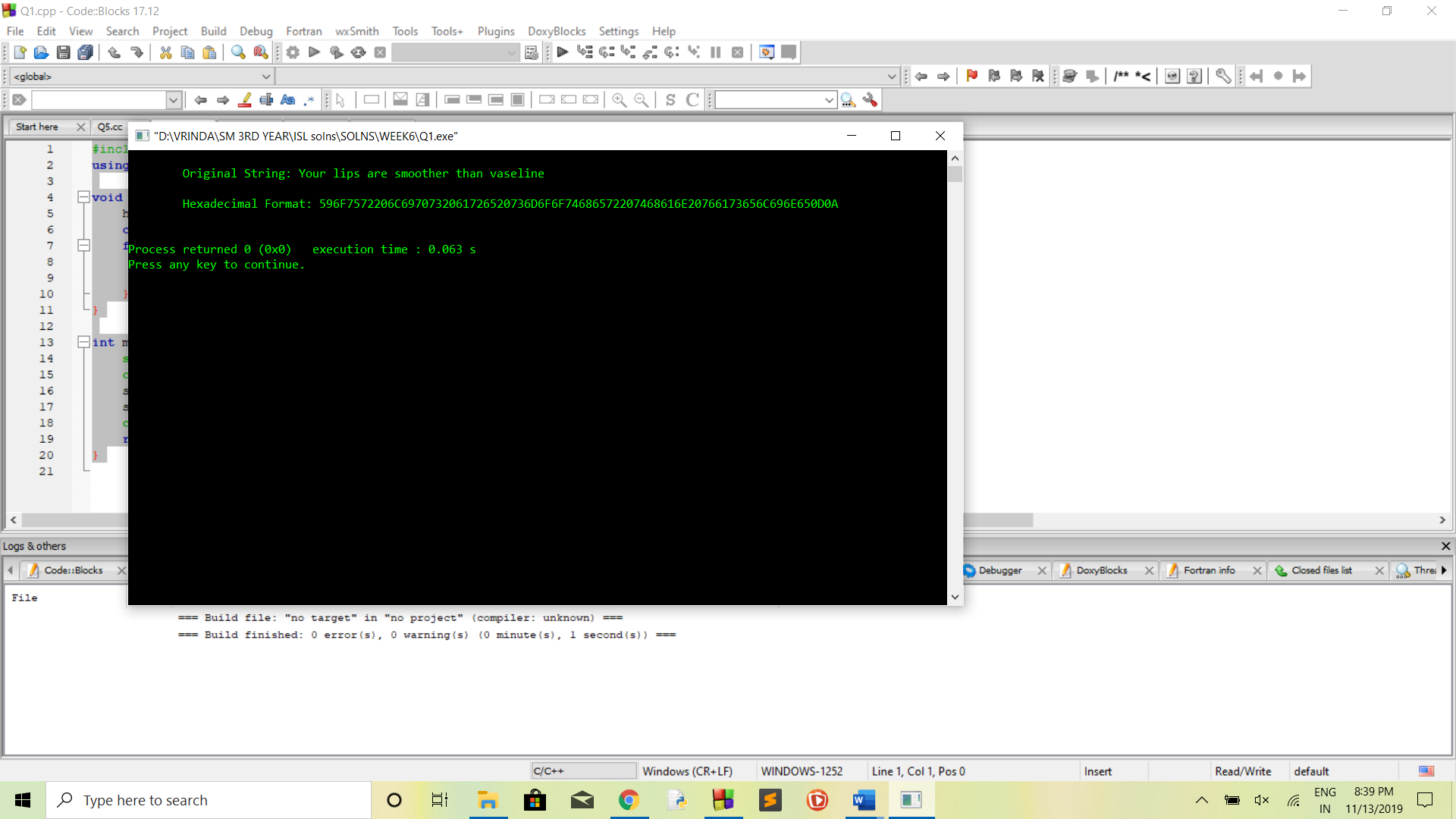
s = s + "0D0A";

cout << "\n\tHexadecimal Format: " << s << "\n\n";

return 0;

}

**OUTPUT:**



1. **Write a program to convert a given hexadecimal key**

**K = 133457799BBCDFF1 into 56-bit key K+ using permutation PC-1.**

**PC-1**

**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**

1. **6 61 53 45 37 29**

**21 13 5 28 20 12 4**

**CODE:**

#include<bits/stdc++.h>

using namespace std;

int main() {

// string K = "133457799BBCDFF1";

string K;

cout << "\n\n\tEnter The Hexadecimal String : ";

cin >> K;

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 KPlus[56];

string hex[16] = {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C", "D", "E", "F"};

string bits[16] = {"0000", "0001", "0010", "0011", "0100", "0101", "0110", "0111", "1000", "1001", "1010", "1011", "1100", "1101", "1110", "1111"};

string bin[(sizeof(K) / sizeof(K[0]))];

for (int i = 0; i < (sizeof(K) / sizeof(K[0])); i++)

{

if (K[i] == '0') bin[i] = bits[0];

if (K[i] == '1') bin[i] = bits[1];

if (K[i] == '2') bin[i] = bits[2];

if (K[i] == '3') bin[i] = bits[3];

if (K[i] == '4') bin[i] = bits[4];

if (K[i] == '5') bin[i] = bits[5];

if (K[i] == '6') bin[i] = bits[6];

if (K[i] == '7') bin[i] = bits[7];

if (K[i] == '8') bin[i] = bits[8];

if (K[i] == '9') bin[i] = bits[9];

if (K[i] == 'A') bin[i] = bits[10];

if (K[i] == 'B') bin[i] = bits[11];

if (K[i] == 'C') bin[i] = bits[12];

if (K[i] == 'D') bin[i] = bits[13];

if (K[i] == 'E') bin[i] = bits[14];

if (K[i] == 'F') bin[i] = bits[15];

}

cout << "\n\n\tK (Hexa Str) : " << K;

cout << "\n\n\tK (64 bits) : ";

for (int l = 0; l < sizeof(K) / sizeof(K[0]); l++) cout << bin[l] << " ";

for (int kp = 0; kp < 56; kp++)

{

KPlus[kp] = int(bin[PC1[kp] / 4][PC1[kp] % 4]) - 48;

}

cout << "\n\n\tK+ (56 bits) : ";

for (int u = 0; u < 56; u++)

{

if ((u+1) % 4 != 0) cout << KPlus[u];

else {

cout << KPlus[u] << " ";

}

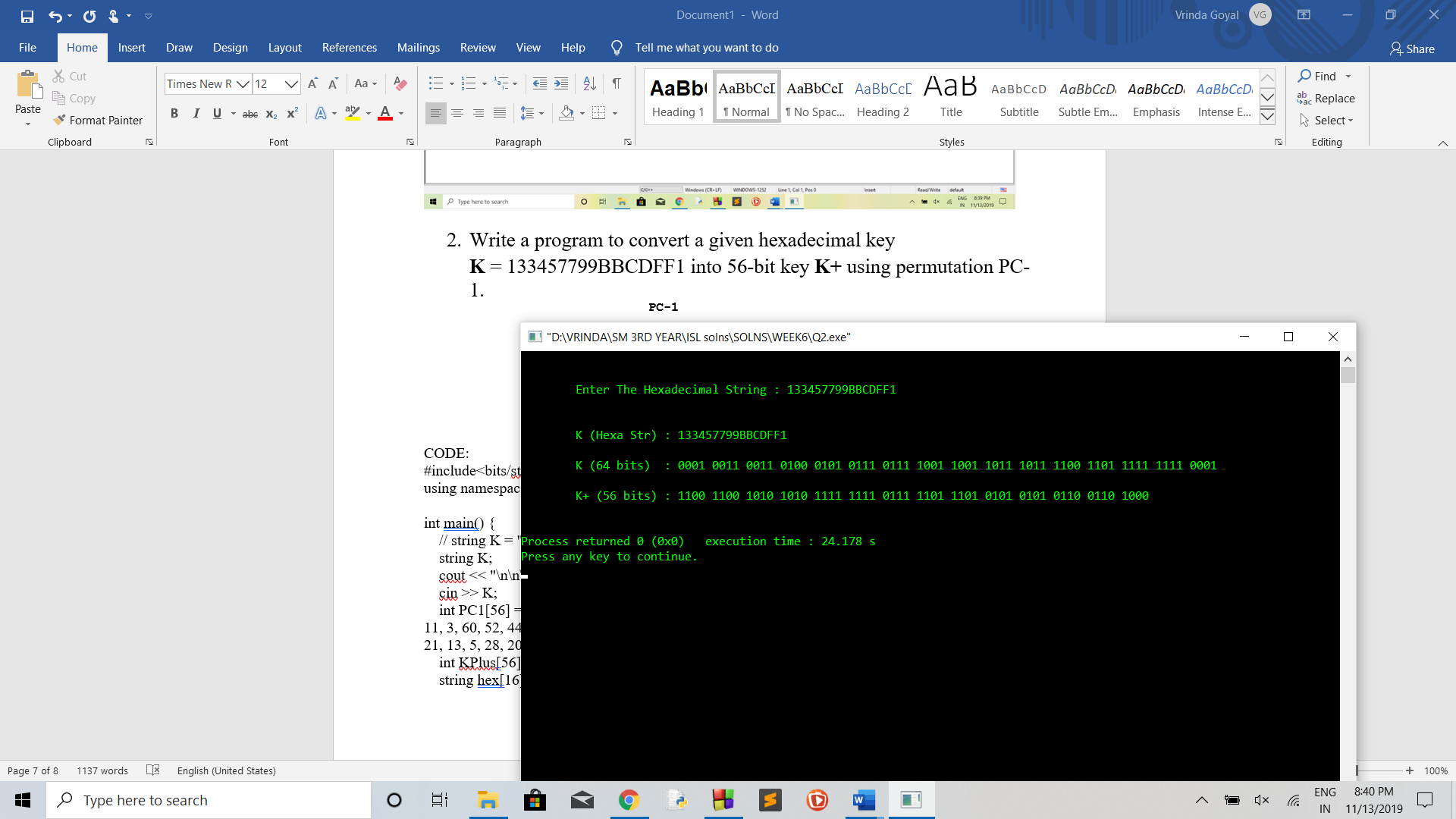
}

cout << "\n\n";

return 0;

}

**OUTPUT:**



1. **From the above key K+, find the C1 and D1, C2 and D2, C3 and D3 using the left shift as per below information:**

**Iteration Number Number of Left Shifts**

**1 1**

**2 1**

**3 2**

**4 2**

**5 2**

**6 2**

**7 2**

**8 2**

**9 1**

**10 2**

**11 2**

**12 2**

**13 2**

**14 2**

**15 2**

**16 1**

**CODE:**

#include<bits/stdc++.h>

using namespace std;

int main() {

int KPlus[56] = {1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0};

int C0[28], D0[28];

int itr[16] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};

cout << "\n\n\tK+ (56-bits) : ";

for (int i = 0; i < 56; i++)

{

if ((i + 1) % 4 != 0) cout << KPlus[i];

else cout << KPlus[i] << " ";

}

cout << endl;

int it = 0;

for (int cd = 0; cd < 28; cd++)

{

C0[cd] = KPlus[it++];

}

for (int cd = 0; cd < 28; cd++)

{

D0[cd] = KPlus[it++];

}

int C[16][28], D[16][28];

for (int iter = 0; iter < 16; iter++)

{

int z1 = 0;

int z2 = 0;

if (iter == 0) {

for (int x = itr[iter]; x < 28; x++)

{

C[iter][z1++] = C0[x];

D[iter][z2++] = D0[x];

}

for (int y = 0; y < itr[iter]; y++)

{

C[iter][z1++] = C0[y];

D[iter][z2++] = D0[y];

}

}

else {

for (int x = itr[iter]; x < 28; x++)

{

C[iter][z1++] = C[iter - 1][x];

D[iter][z2++] = D[iter - 1][x];

}

for (int y = 0; y < itr[iter]; y++)

{

C[iter][z1++] = C[iter - 1][y];

D[iter][z2++] = D[iter - 1][y];

}

}

}

cout << "\n\tC0 (28-bits) : ";

for (int c = 0; c < 28; c++)

{

if ((c + 1) % 4 != 0) cout << C0[c];

else cout << C0[c] << " ";

}

cout << "\n\tD0 (28-bits) : ";

for (int d = 0; d < 28; d++)

{

if ((d + 1) % 4 != 0) cout << D0[d];

else cout << D0[d] << " ";

}

cout << endl;

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

{

if (i + 1 <= 9) cout << "\n\tC" << i + 1 << " (28-bits) : ";

else cout << "\n\tC" << i + 1 << " (28-bits) : ";

for (int x = 0; x < 28; x++)

{

if ((x + 1) % 4 != 0) cout << C[i][x];

else cout << C[i][x] << " ";

}

if (i + 1 <= 9) cout << "\n\tD" << i + 1 << " (28-bits) : ";

else cout << "\n\tD" << i + 1 << " (28-bits) : ";

for (int y = 0; y < 28; y++)

{

if ((y + 1) % 4 != 0) cout << D[i][y];

else cout << D[i][y] << " ";

}

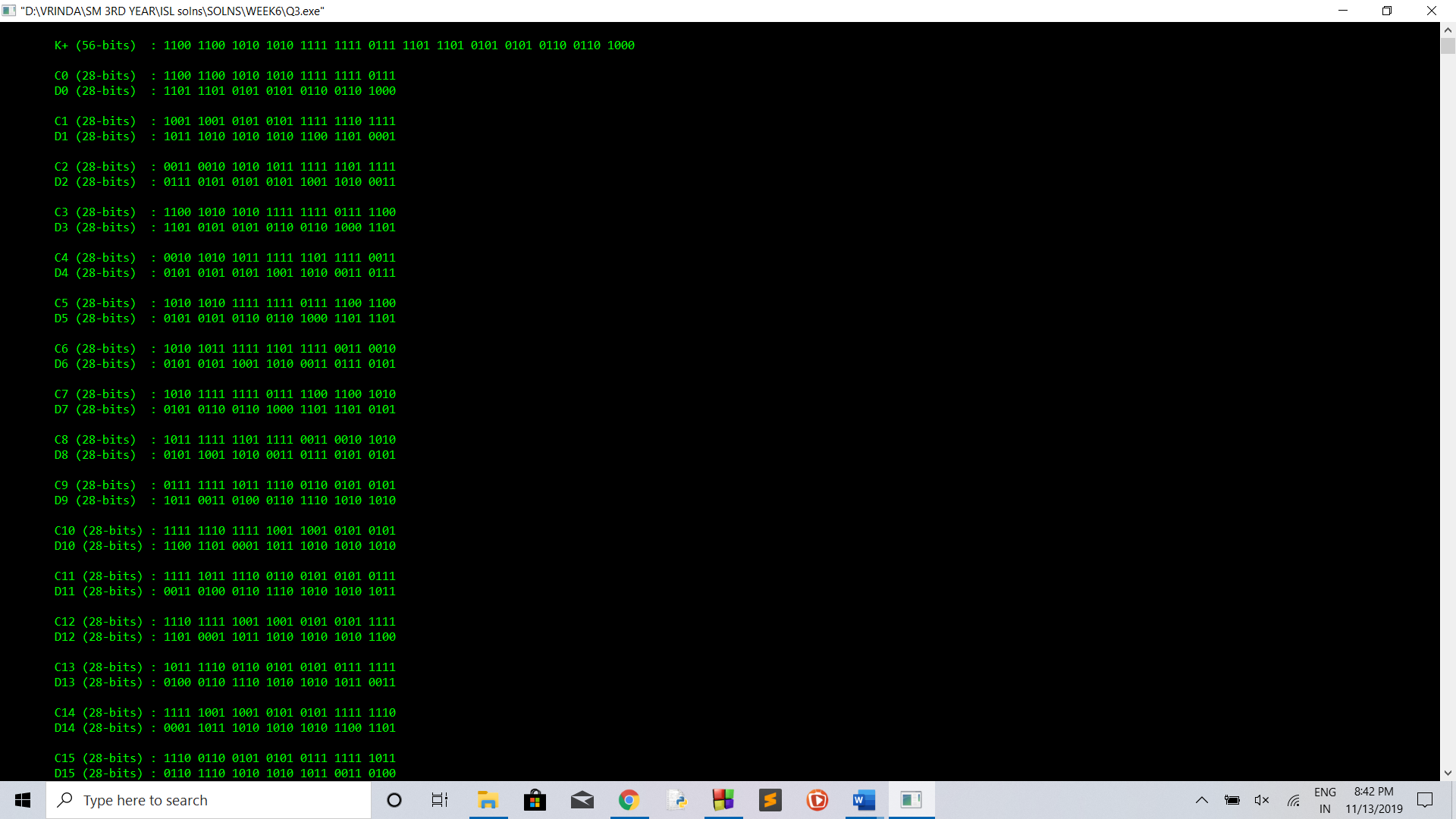
cout << "\n"; }

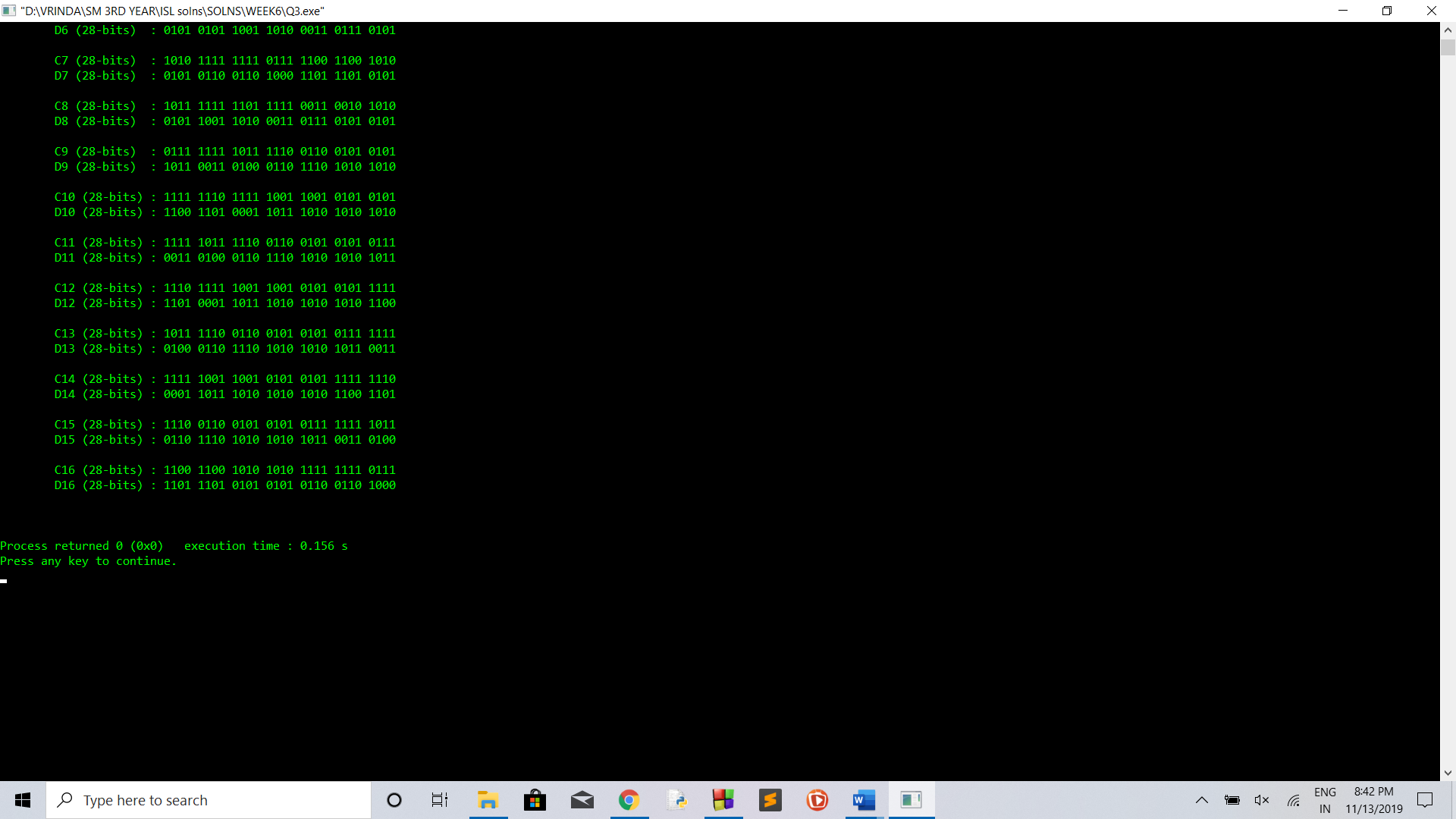
cout << endl << endl;

return 0;

}

**OUTPUT:**





1. **Write a program to convert the 56-bit key K+ obtained from Q.3 into 48-bit key K1 using permutation PC-2.**

**PC-2**

**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**

**Note: For the first key K1, we use C1 D1, for the second key K2, we use C2 D2 , and for the third key K3, we use C3 D3.**

**CODE:**

#include<bits/stdc++.h>

using namespace std;

int main() {

int KPlus[56] = {1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0};

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};

int itr[16] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};

// 1. Generating C1, C2, ......., C16 & D1, D2, ...... D16 from K+

// 1.1. Creating C0 & D0 from K+

int C0[28], D0[28];

int it = 0;

for (int cd = 0; cd < 28; cd++)

{

C0[cd] = KPlus[it++];

}

for (int cd = 0; cd < 28; cd++)

{

D0[cd] = KPlus[it++];

}

// 1.2. Creating C1, C2, ......., C16 & D1, D2, ...... D16 from C0 & D0

int C[16][28], D[16][28];

for (int iter = 0; iter < 16; iter++)

{

int z1 = 0;

int z2 = 0;

if (iter == 0) {

for (int x = itr[iter]; x < 28; x++)

{

C[iter][z1++] = C0[x];

D[iter][z2++] = D0[x];

}

for (int y = 0; y < itr[iter]; y++)

{

C[iter][z1++] = C0[y];

D[iter][z2++] = D0[y];

}

}

else {

for (int x = itr[iter]; x < 28; x++)

{

C[iter][z1++] = C[iter - 1][x];

D[iter][z2++] = D[iter - 1][x];

}

for (int y = 0; y < itr[iter]; y++)

{

C[iter][z1++] = C[iter - 1][y];

D[iter][z2++] = D[iter - 1][y];

}

}

}

// 2. Generating K1, K2, ......., K16 in 56 bit format from C1, C2, ......., C16 & D1, D2, ...... D16

int K\_56[16][56];

for (int w = 0; w < 16; w++)

{

int out = 0;

for (int w1 = 0; w1 < 28; w1++)

{

K\_56[w][out++] = C[w][w1];

}

for (int w2 = 0; w2 < 28; w2++)

{

K\_56[w][out++] = D[w][w2];

}

}

// 3. Generating K1, K2, ......., K16 in 48 bit format from K1, K2, ......., K16 in 56 bit format using PC-2

int K\_48[16][48];

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

{

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

{

K\_48[i][j] = K\_56[i][PC2[j]];

}

}

// Printing The Final Outputs :

cout << "\n\n\tK+ (56-bits) : ";

for (int i = 0; i < 56; i++)

{

if ((i + 1) % 4 != 0) cout << KPlus[i];

else cout << KPlus[i] << " ";

}

cout << "\n\tC0 (28-bits) : ";

for (int c = 0; c < 28; c++)

{

if ((c + 1) % 4 != 0) cout << C0[c];

else cout << C0[c] << " ";

}

cout << "\n\tD0 (28-bits) : ";

for (int d = 0; d < 28; d++)

{

if ((d + 1) % 4 != 0) cout << D0[d];

else cout << D0[d] << " ";

}

cout << "\n";

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

{

if (i + 1 <= 9) cout << "\n\tC" << i + 1 << " (28-bits) : ";

else cout << "\n\tC" << i + 1 << " (28-bits) : ";

for (int x = 0; x < 28; x++)

{

if ((x + 1) % 4 != 0) cout << C[i][x];

else cout << C[i][x] << " ";

}

if (i + 1 <= 9) cout << "\n\tD" << i + 1 << " (28-bits) : ";

else cout << "\n\tD" << i + 1 << " (28-bits) : ";

for (int y = 0; y < 28; y++)

{

if ((y + 1) % 4 != 0) cout << D[i][y];

else cout << D[i][y] << " ";

}

if (i + 1 <= 9) cout << "\n\tK" << i + 1 << " (56-bits) : ";

else cout << "\n\tK" << i + 1 << " (56-bits) : ";

for (int j = 0; j < 56; j++)

{

if ((j + 1) % 4 != 0) cout << K\_56[i][j];

else cout << K\_56[i][i] << " ";

}

if (i + 1 <= 9) cout << "\n\tK" << i + 1 << " (48-bits) : ";

else cout << "\n\tK" << i + 1 << " (48-bits) : ";

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

{

if ((k + 1) % 4 != 0) cout << K\_48[i][k];

else cout << K\_48[i][k] << " ";

}

cout << "\n";

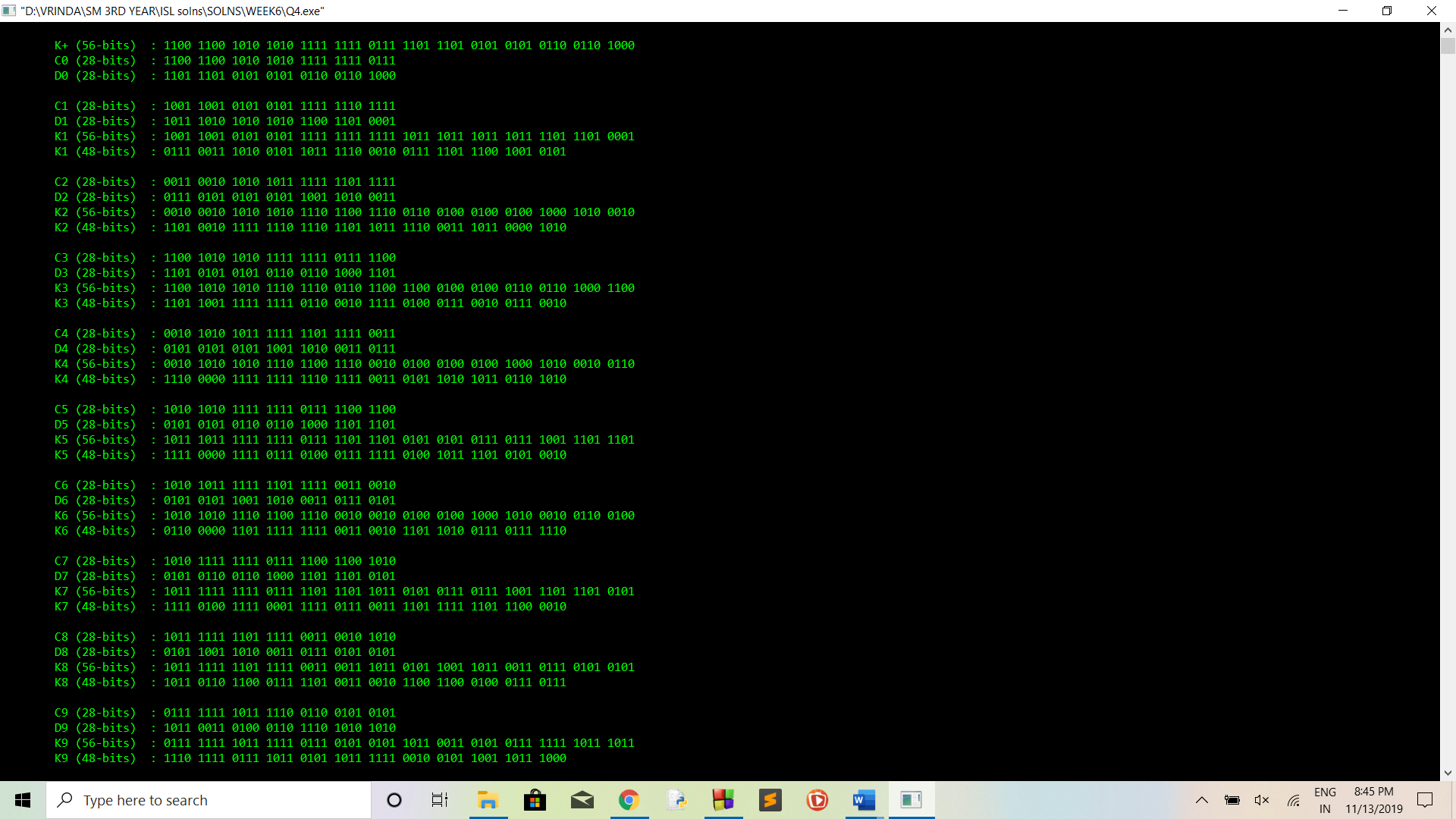
}

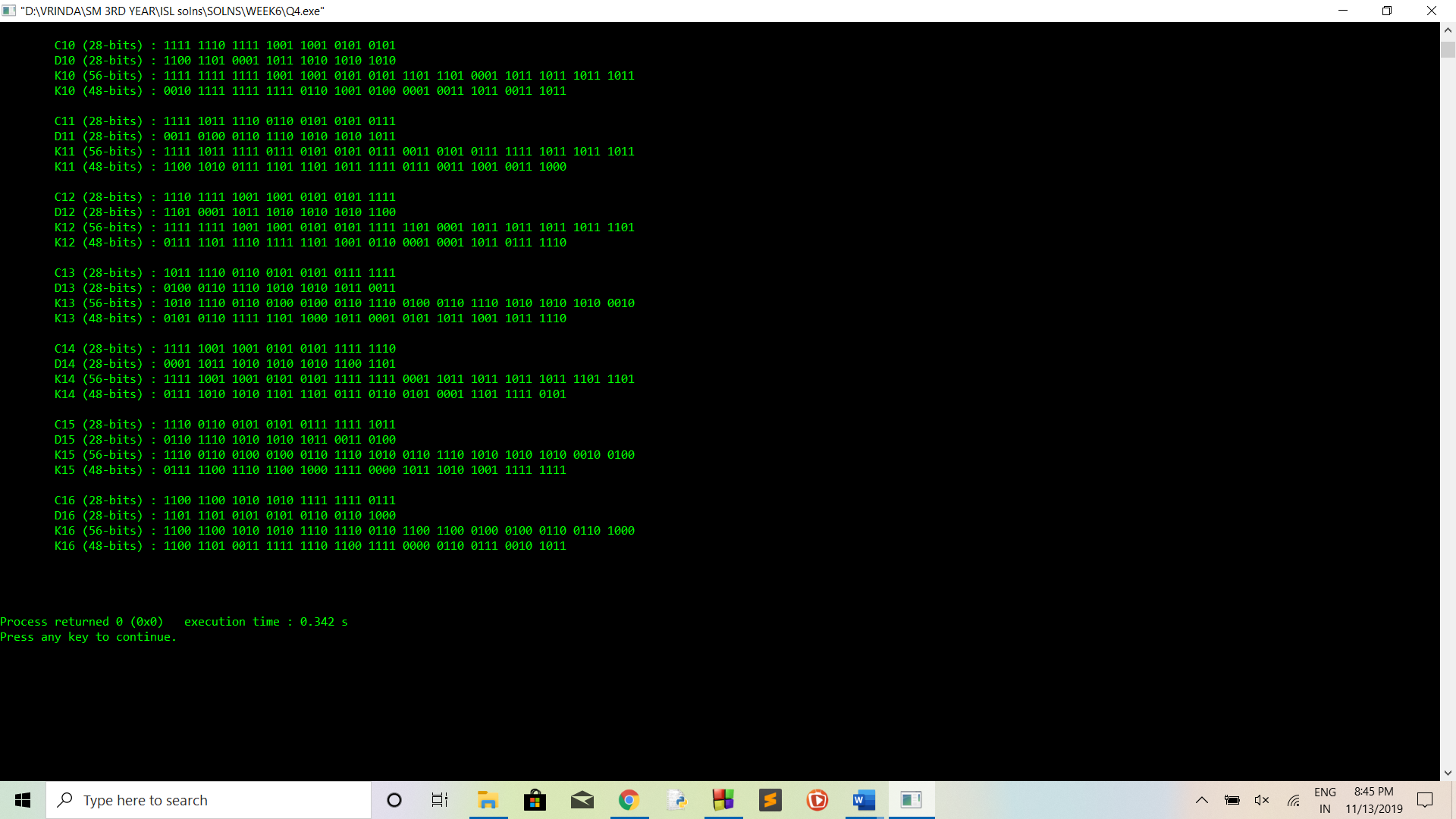
cout << "\n\n\n";

return 0;

}

**OUTPUT:**





**LAB 7**

1. **Write a program to encode a given message M = 0123456789ABCDEF,**

**using Data Encryption Standard (DES) algorithm. Find the output for first iteration (n=1). Use the key as (for first iteration):**

**K1 = 000110 110000 001011 101111 111111 000111 000001 110010**

**This objective could be obtained in the following steps:**

1. **Convert the M into hexadecimal and write into binary form**
2. **Obtain an initial permutation IP from binary message obtained in part (a.) using the permutation:**

**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**

1. **Find L0 and R0 then convert R0 from 32 bits to 48 bits using the function E (E BIT-SELECTION TABLE) by repeating some of the bits in R0 , denoted by E[R0] (48 bits) which is determined by :**

**E BIT-SELECTION TABLE**

**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**

1. **Find R1 using the formula**

**R1 = L0 XOR f (R0, K1),**

**Where f (R0, K1) = f3 (f2 (f1 (R0, K1))),**

**f1 (R0, K1) = E (R0) XOR K1,**

**f2(X) = (S1 (B1) S2 (B2)...S8 (B8)),**

**f3(Y) =P(Y),**

**X= E (R0) XOR K1, Y= f2(X).**

1. **Convert the 48 bits output obtained from part (d.) into 32 bits using the**

**S-Boxes.**

**Here B1B2B3B4B5B6B7B8, is converted into S1(B1)S2(B2)S3(B3)S4(B4)S5(B5)S6(B6)S7(B7)S8(B8)**

**where,**

**The tables defining the functions S1,...,S8 are the following:**

**S1**

**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**

**S2**

**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**

**S3**

**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**

**S4**

**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**

**S5**

**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**

**S6**

**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**

**S7**

**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**

**S8**

**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**

1. **Find the permutation of 32 bits of output of part (e.) using**

**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**

1. **Find the L1 and R1 using the formula:**

**L1 = R0**

**R1 = L0 XOR f (R0, K1),**

**where, f (R0, K1) is the output of part (f.).**

**CODE:**

#include<bits/stdc++.h>

using namespace std;

int main()

{

int 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};

string M = "0123456789ABCDEF";

int K1[48] = {0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0};

string hex[16] = {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C", "D", "E", "F"};

string bits[16] = {"0000", "0001", "0010", "0011", "0100", "0101", "0110", "0111", "1000", "1001", "1010", "1011", "1100", "1101", "1110", "1111"};

int E\_BIT\_SELECTION\_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};

string bin[(sizeof(M) / sizeof(M[0]))];

for (int i = 0; i < (sizeof(M) / sizeof(M[0])); i++) {

if (M[i] == '0') bin[i] = bits[0];

if (M[i] == '1') bin[i] = bits[1];

if (M[i] == '2') bin[i] = bits[2];

if (M[i] == '3') bin[i] = bits[3];

if (M[i] == '4') bin[i] = bits[4];

if (M[i] == '5') bin[i] = bits[5];

if (M[i] == '6') bin[i] = bits[6];

if (M[i] == '7') bin[i] = bits[7];

if (M[i] == '8') bin[i] = bits[8];

if (M[i] == '9') bin[i] = bits[9];

if (M[i] == 'A') bin[i] = bits[10];

if (M[i] == 'B') bin[i] = bits[11];

if (M[i] == 'C') bin[i] = bits[12];

if (M[i] == 'D') bin[i] = bits[13];

if (M[i] == 'E') bin[i] = bits[14];

if (M[i] == 'F') bin[i] = bits[15];

}

cout << "\n\n\tM (64 bits) : ";

for (int l = 0; l < sizeof(M) / sizeof(M[0]); l++) cout << bin[l] << " ";

int IP\_64[64];

for (int kp = 0; kp < 64; kp++)

{

IP\_64[kp] = int(bin[(Permutation[kp] - 1) / 4][Permutation[kp] % 4]) - 48;

}

cout << "\n\n\tIP (64 bits) : ";

for (int l = 0; l < 64; l++)

{

if ((l + 1) % 4 != 0) cout << IP\_64[l];

else cout << IP\_64[l] << " ";

}

int L0[32], R0[32];

int it = 0;

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

{

L0[cd] = IP\_64[it++];

}

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

{

R0[cd] = IP\_64[it++];

}

cout << "\n\n\tL0 (32 bits) : ";

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

{

if ((l + 1) % 4 != 0) cout << L0[l];

else cout << L0[l] << " ";

}

cout << "\n\n\tR0 (32 bits) : ";

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

{

if ((l + 1) % 4 != 0) cout << R0[l];

else cout << R0[l] << " ";

}

int R0\_48[48];

int L0\_48[48];

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

{

L0\_48[x] = L0[E\_BIT\_SELECTION\_TABLE[x] - 1];

R0\_48[x] = R0[E\_BIT\_SELECTION\_TABLE[x] - 1];

}

cout << "\n\n\tL0 (48 bits) : ";

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

{

if ((l + 1) % 4 != 0) cout << L0\_48[l];

else cout << L0\_48[l] << " ";

}

cout << "\n\n\tR0 (48 bits) : ";

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

{

if ((l + 1) % 4 != 0) cout << R0\_48[l];

else cout << R0\_48[l] << " ";

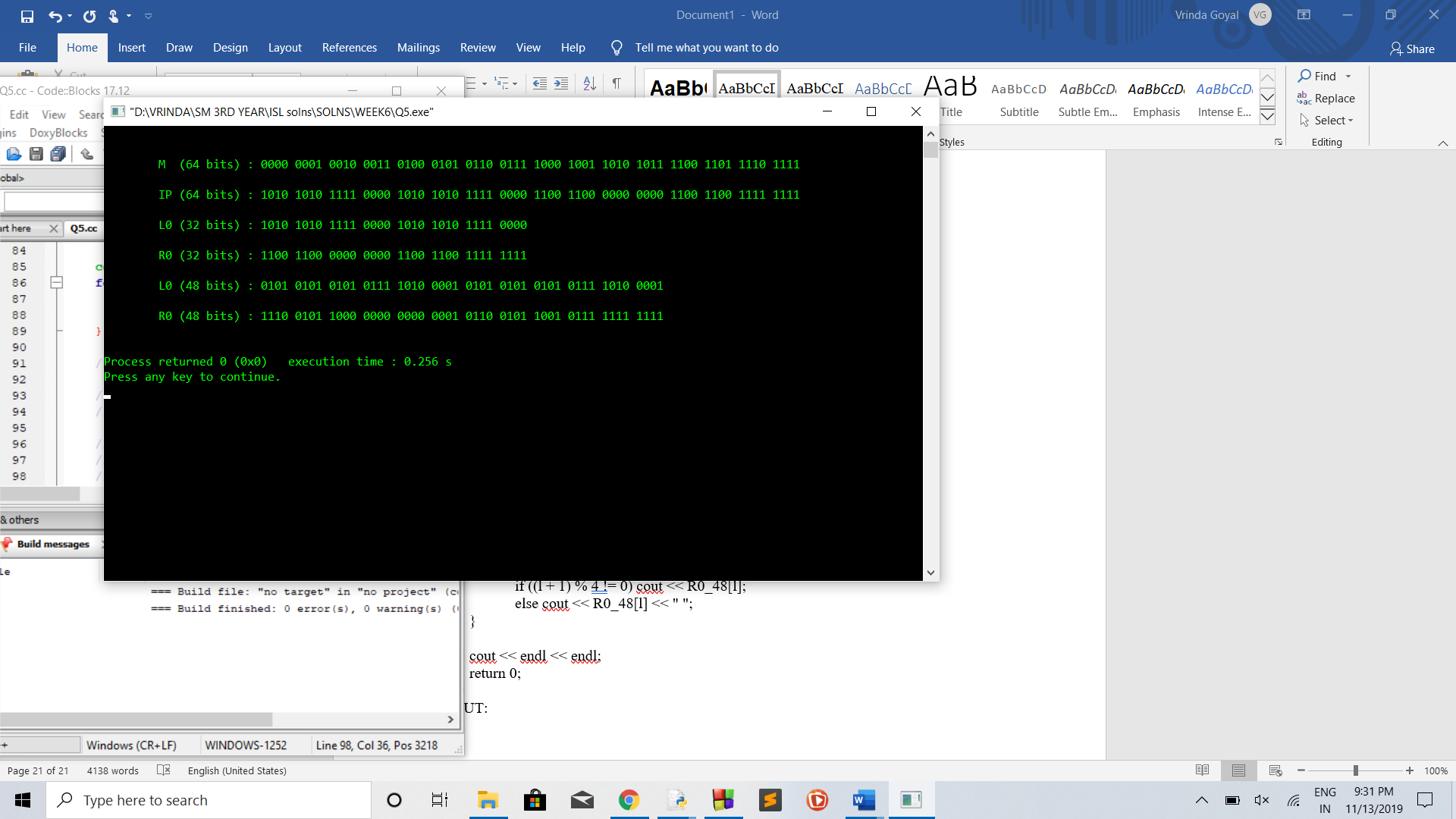
}

cout << endl << endl;

return 0;

}

**OUTPUT:**



**LAB 8 and 9**

1. **Explore the RSA algorithm and find the encrypted message for a letter “l” manually using public key cryptography: RSA, use p=5, q=7.**

**CODE:**

from decimal import Decimal

def gcd(a,b):

if b==0:

return a

else:

return gcd(b,a%b)

p = int(input('Enter the value of p = '))

q = int(input('Enter the value of q = '))

no = int(input('Enter the value of text = '))

n = p\*q

t = (p-1)\*(q-1)

for e in range(2,t):

if gcd(e,t)== 1:

break

for i in range(1,10):

x = 1 + i\*t

if x % e == 0:

d = int(x/e)

break

ctt = Decimal(0)

ctt =pow(no,e)

ct = ctt % n

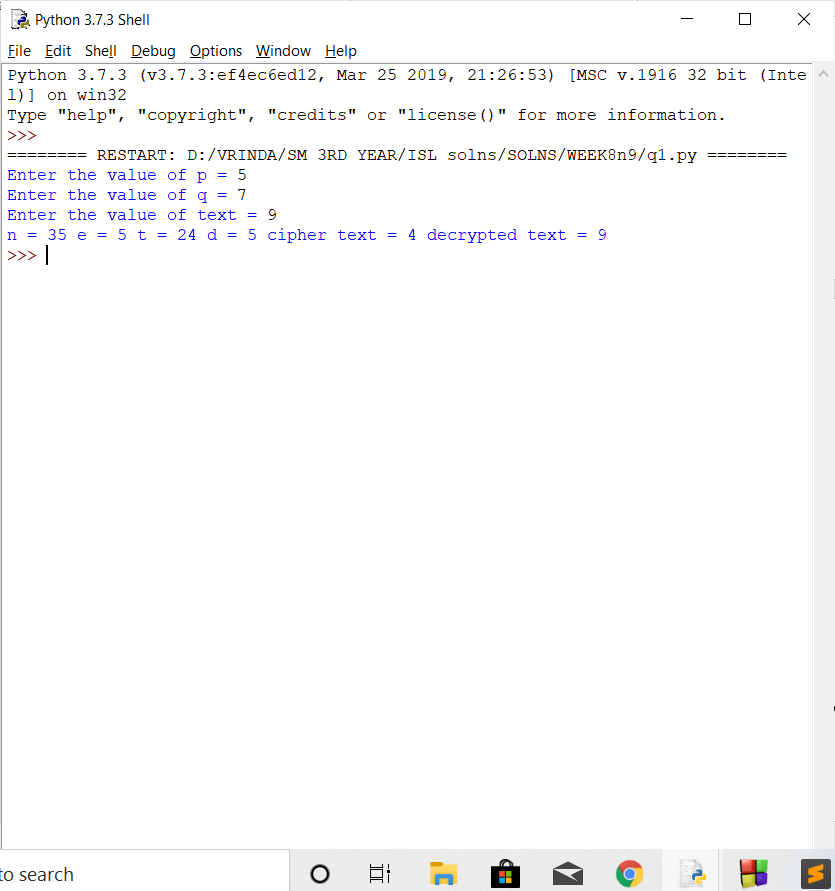
dtt = Decimal(0)

dtt = pow(ct,d)

dt = dtt % n

print('n = '+str(n)+' e = '+str(e)+' t = '+str(t)+' d = '+str(d)+' cipher text = '+str(ct)+' decrypted text = '+str(dt))

**OUTPUT:**



1. **Find the encoded text for a word “hello” using same private key (n, d) and public key (n, e) used in Q. 2.**

**CODE:**

from decimal import Decimal

def gcd(a,b):

if b==0:

return a

else:

return gcd(b,a%b)

p = int(input('Enter the value of p = '))

q = int(input('Enter the value of q = '))

no = int(input('Enter the value of text = '))

n = p\*q

t = (p-1)\*(q-1)

for e in range(2,t):

if gcd(e,t)== 1:

break

for i in range(1,10):

x = 1 + i\*t

if x % e == 0:

d = int(x/e)

break

ctt = Decimal(0)

ctt =pow(no,e)

ct = ctt % n

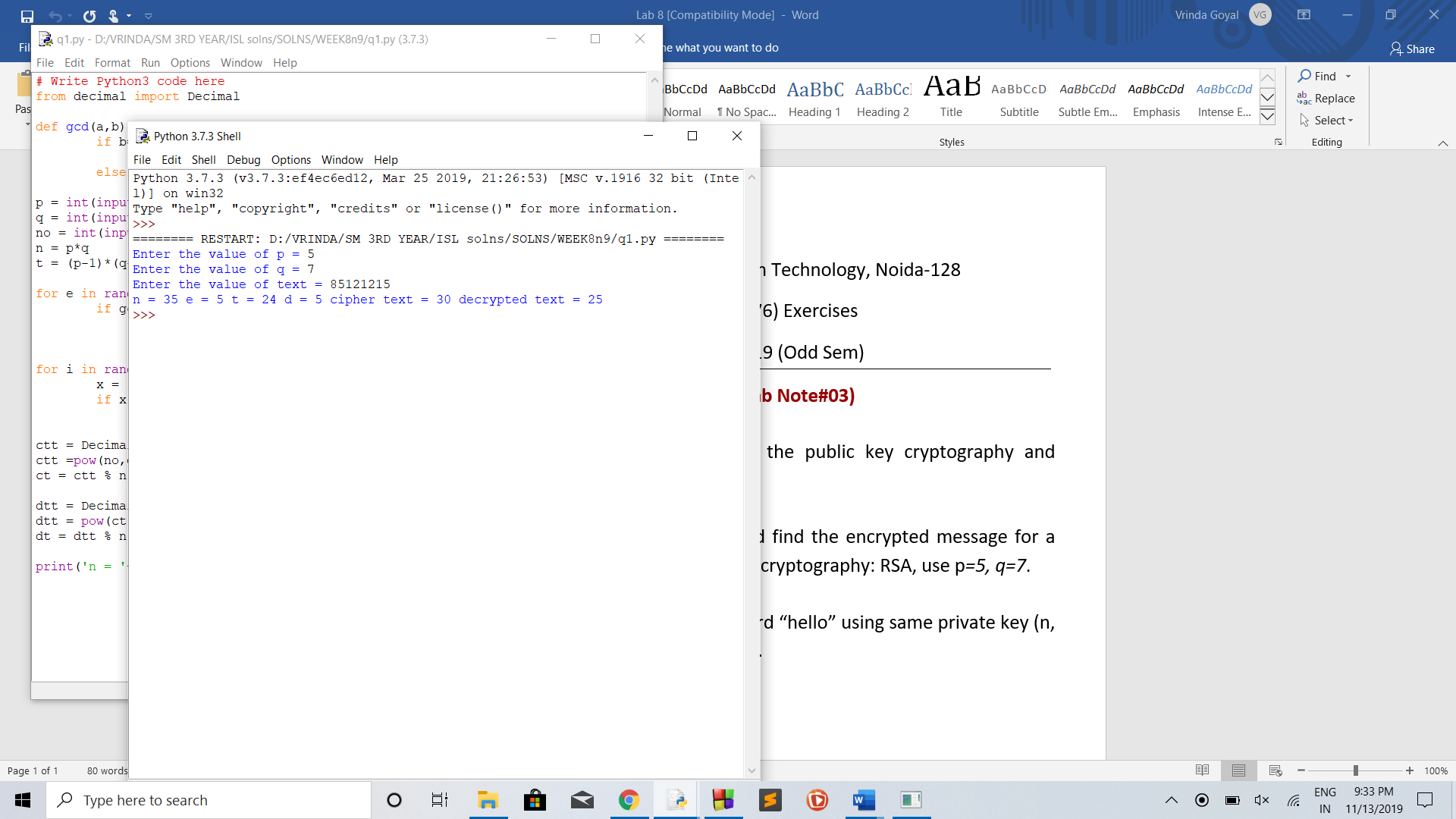
dtt = Decimal(0)

dtt = pow(ct,d)

dt = dtt % n

print('n = '+str(n)+' e = '+str(e)+' t = '+str(t)+' d = '+str(d)+' cipher text = '+str(ct)+' decrypted text = '+str(dt))

**OUTPUT:**



**Implementation of Diffie Hellman Key Exchange Protocol (Client - Server Model)**

**CODE:**

1. **Client Side**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

#include <limits.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <math.h>

#include <ctype.h>

#define SERV\_PORT 19000

#define MAX\_LEN 1024

#define MAXSIZE 1000000

#define CAESAR\_MOD 41

#define M\_ITERATION 15

typedef struct GlobalInfo {

int prime;

int generator;

} GlobalInfo;

/\* Function to compute (a ^ b) mod p \*/

int compute\_exp\_modulo(int a, int b, int p) {

long long x = 1, y = a;

while (b > 0) {

if (b % 2 == 1)

x = (x \* y) % p;

y = (y \* y) % p;

b /= 2;

}

return (int)(x % p);

}

/\* Function to check primality of random generated numbers using Miller-Rabin Test \*/

int MillerRabinTest(int value, int iteration) {

if (value < 2)

return 0;

int q = value - 1, k = 0;

while (!(q % 2)) {

q /= 2;

k++;

}

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

int a = rand() % (value - 1) + 1;

int current = q;

int flag = 1;

int mod\_result = compute\_exp\_modulo(a, current, value);

for (int i = 1; i <= k; i++) {

if (mod\_result == 1 || mod\_result == value - 1) {

flag = 0;

break;

}

mod\_result = (int)((long long)mod\_result \* mod\_result % value);

}

if (flag)

return 0;

}

return 1;

}

/\* Generate a prime number that is going to be shared

\* globally between client and server

\*/

int GeneratePrime() {

printf("\* Running Miller-Rabin test to find a large prime number...\n\n");

srand(time(NULL));

while(1) {

int current\_value = rand() % INT\_MAX;

if (!(current\_value % 2))

current\_value++;

if (MillerRabinTest(current\_value, M\_ITERATION) == 1)

return current\_value;

}

}

/\* Generate the primitive root by checking for random numbers \*/

int GeneratePrimitiveRoot(int p) {

/\* Construct sieve of primes \*/

int sieve[MAXSIZE];

memset(sieve, 0, sizeof(sieve));

sieve[0] = sieve[1] = 1;

for (int i = 4; i < MAXSIZE; i += 2)

sieve[i] = 1;

for (int i = 3; i < MAXSIZE; i += 2) {

if (!sieve[i]) {

for (int j = 2 \* i; j < MAXSIZE; j += i)

sieve[j] = 1;

}

}

while (1) {

int a = rand() % (p - 2) + 2;

int phi = p - 1, flag = 1, root = sqrt(phi);

for (int i = 2; i <= root; i++) {

if (!sieve[i] && !(phi % i)) {

int mod\_result = compute\_exp\_modulo(a, phi / i, p);

if (mod\_result == 1) {

flag = 0;

break;

}

if (MillerRabinTest(phi / i, M\_ITERATION) && !(phi % (phi / i))) {

int mod\_result = compute\_exp\_modulo(a, phi / (phi / i), p);

if (mod\_result == 1) {

flag = 0;

break;

}

}

}

}

if (flag)

return a;

}

}

/\* Convert character to encrypted character \*/

char caesar\_encrypt(char c, int key) {

char dict[] = {' ','A','B','C','D','E','F','G','H','I','J','K','L','M','N',

'O','P','Q','R','S','T','U','V','W','X','Y','Z',',','.',

'?','0','1','2','3','4','5','6','7','8','9','!'};

for (int i = 0; i < CAESAR\_MOD; i++) {

if (dict[i] == c)

return dict[(i + key) % CAESAR\_MOD];

}

return c;

}

void send\_message(int sockfd, char message[MAX\_LEN], int len) {

int n\_sent = 0;

while (n\_sent < len) {

int temp;

if ((temp = send(sockfd, message + n\_sent, len - n\_sent, 0)) <= 0) {

perror("Error ");

exit(-1);

}

n\_sent += temp;

}

}

int recv\_message(int sockfd, char buffer[MAX\_LEN], int recv\_size) {

int n\_recv = 0;

while (n\_recv < recv\_size) {

int temp;

if ((temp = recv(sockfd, buffer + n\_recv, MAX\_LEN - n\_recv, 0)) <= 0) {

if (temp == 0)

break;

perror("Error ");

exit(-1);

}

n\_recv += temp;

}

return n\_recv;

}

/\* Client main program \*/

int main(int argc, char \*argv[]) {

if (argc < 3) {

printf("Please enter command line arguments [IP\_ADDRESS] [FILENAME]\n");

exit(-1);

}

printf("---------------------------------------------------------------\n");

printf("CLIENT\n");

printf("---------------------------------------------------------------\n");

/\* Generate a prime number and its primitive root (publicly known) \*/

GlobalInfo g;

g.prime = GeneratePrime();

printf("\*\* Global prime - %d\n", g.prime);

g.generator = GeneratePrimitiveRoot(g.prime);

printf("\*\* Global primitive root - %d\n\n", g.generator);

/\* Choose a private key for the client \*/

int private\_key = rand() % (g.prime - 1) + 1;

int public\_key = compute\_exp\_modulo(g.generator, private\_key, g.prime);

printf("\*\*\* Client private key : %d\n", private\_key);

printf("\*\*\* Client public key : %d\n\n", public\_key);

/\* Establish socket connection with the server \*/

int sockfd;

if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

perror("Error ");

exit(-1);

}

struct sockaddr\_in serv\_addr;

memset(&serv\_addr, 0, sizeof(serv\_addr));

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_addr.s\_addr = inet\_addr(argv[1]);

serv\_addr.sin\_port = htons(SERV\_PORT);

/\* Connect to the server \*/

if (connect(sockfd, (struct sockaddr \*) &serv\_addr, sizeof(serv\_addr)) < 0) {

perror("Error ");

exit(-1);

}

/\* Send public\_key, generator and prime to the server \*/

char message[MAX\_LEN];

memset(message, 0, sizeof(message));

int n = sprintf(message, "%d\n%d\n%d\n", public\_key, g.prime, g.generator);

send\_message(sockfd, message, n);

/\* Receive server public key \*/

n = recv\_message(sockfd, message, sizeof(int) + sizeof(char));

int public\_key\_server = atoi(message);

printf("\*\*\*\* Server public key : %d\n\n", public\_key\_server);

/\* Compute shared key and caesar key \*/

int shared\_key = compute\_exp\_modulo(public\_key\_server, private\_key, g.prime);

int caesar\_key = shared\_key % CAESAR\_MOD;

printf("\*\*\*\*\* Shared key : %d\n", shared\_key);

printf("\*\*\*\*\* Caesar key : %d\n\n", caesar\_key);

/\* Send file contents to server after encryption \*/

FILE \*input;

if ((input = fopen(argv[2], "r")) == NULL) {

perror("Error ");

exit(-1);

}

printf("\*\*\*\*\*\* Sending file to server in encrypted format...\n");

while ((n = fread(message, sizeof(char), MAX\_LEN, input)) > 0) {

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

message[i] = caesar\_encrypt(toupper(message[i]), caesar\_key);

send\_message(sockfd, message, n);

}

fclose(input);

printf("\*\*\*\*\*\* Finished sending the data to server!\n\n");

printf("---------------------------------------------------------------\n");

close(sockfd);

return 0;

}

1. **Server Side**

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <arpa/inet.h>

#include <string.h>

#include <unistd.h>

#define SERV\_PORT 19000

#define MAX\_LEN 1024

#define CAESAR\_MOD 41

#define LISTEN\_Q 5

typedef struct GlobalInfo {

int prime;

int generator;

} GlobalInfo;

int compute\_exp\_modulo(int a, int b, int p) {

long long x = 1, y = a;

while (b > 0) {

if (b % 2 == 1)

x = (x \* y) % p;

y = (y \* y) % p;

b /= 2;

}

return (int)(x % p);

}

/\* Convert encrypted character to plaintext character \*/

char caesar\_decrypt(char c, int key) {

char dict[] = {' ','A','B','C','D','E','F','G','H','I','J','K','L','M','N',

'O','P','Q','R','S','T','U','V','W','X','Y','Z',',','.',

'?','0','1','2','3','4','5','6','7','8','9','!'};

for (int i = 0; i < CAESAR\_MOD; i++) {

if (dict[i] == c)

return dict[(CAESAR\_MOD + i - key) % CAESAR\_MOD];

}

return c;

}

void send\_message(int sockfd, char message[MAX\_LEN], int len) {

int n\_sent = 0;

while (n\_sent < len) {

int temp;

if ((temp = send(sockfd, message + n\_sent, len - n\_sent, 0)) <= 0) {

perror("Error ");

exit(-1);

}

n\_sent += temp;

}

}

int recv\_message(int sockfd, char buffer[MAX\_LEN], int recv\_size) {

int n\_recv = 0;

while (n\_recv < recv\_size) {

int temp;

if ((temp = recv(sockfd, buffer + n\_recv, MAX\_LEN - n\_recv, 0)) <= 0) {

if (temp == 0)

break;

perror("Error ");

exit(-1);

}

n\_recv += temp;

}

return n\_recv;

}

/\* Server main program \*/

int main(int argc, char \*argv[]) {

if (argc < 2) {

printf("Please enter command line arguments [IP\_ADDRESS]\n");

exit(-1);

}

printf("----------------------------------------------------------------\n");

printf("SERVER\n");

printf("----------------------------------------------------------------\n");

printf("Server started! Waiting for connection from the client...\n\n");

GlobalInfo g;

/\* Establish socket connection with the client \*/

int server\_sockfd, cli\_sockfd;

if ((server\_sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

perror("Error ");

exit(-1);

}

struct sockaddr\_in serv\_addr;

memset(&serv\_addr, 0, sizeof(serv\_addr));

serv\_addr.sin\_addr.s\_addr = inet\_addr(argv[1]);

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(SERV\_PORT);

if (bind(server\_sockfd, (struct sockaddr \*) &serv\_addr, sizeof(serv\_addr)) < 0) {

perror("Error ");

exit(-1);

}

listen(server\_sockfd, LISTEN\_Q);

while (1) {

if ((cli\_sockfd = accept(server\_sockfd, NULL, NULL)) < 0) {

perror("Error ");

exit(-1);

}

printf("\* Client connected!...\n\n");

char buffer[MAX\_LEN];

memset(buffer, 0, sizeof(buffer));

/\* Receive key, generator and prime from client \*/

int recv\_size = sizeof(int) \* 3 + sizeof(char) \* 3;

int n = recv\_message(cli\_sockfd, buffer, recv\_size);

int public\_key\_client = atoi(buffer);

int i = 0;

while (buffer[i] != '\n')

i++;

g.prime = atoi(buffer + ++i);

while (buffer[i] != '\n')

i++;

g.generator = atoi(buffer + ++i);

printf("\*\* Client public key : %d\n", public\_key\_client);

printf("\*\* Global prime : %d\n", g.prime);

printf("\*\* Global primitive root : %d\n\n", g.generator);

/\* Generate server private key and public key \*/

int private\_key = rand() % (g.prime - 1) + 1;

int public\_key = compute\_exp\_modulo(g.generator, private\_key, g.prime);

printf("\*\*\* Server private key : %d\n", private\_key);

printf("\*\*\* Server public key : %d\n\n", public\_key);

/\* Send public\_key to the client \*/

n = sprintf(buffer, "%d\n", public\_key);

send\_message(cli\_sockfd, buffer, n);

/\* Compute shared key and caesar key \*/

int shared\_key = compute\_exp\_modulo(public\_key\_client, private\_key, g.prime);

int caesar\_key = shared\_key % CAESAR\_MOD;

printf("\*\*\*\* Shared key : %d\n", shared\_key);

printf("\*\*\*\* Caesar key : %d\n\n", caesar\_key);

/\* Receive encrypted text from client and write to output.txt file \*/

printf("\*\*\*\*\* Receiving file from client...\n\n");

FILE \*output;

if((output = fopen("output.txt","w")) < 0) {

perror("Error ");

exit(-1);

}

while ((n = recv\_message(cli\_sockfd, buffer, MAX\_LEN)) > 0) {

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

printf("%c", buffer[i]);

buffer[i] = caesar\_decrypt(buffer[i], caesar\_key);

}

fwrite(buffer, sizeof(char), n, output);

}

fclose(output);

printf("\n\n\*\*\*\*\* Finished receiving data from client!\n\n");

printf("----------------------------------------------------------------\n");

close(cli\_sockfd);

}

return 0;

}

**ASSIGNMENT-9**

Packet capture (PCAP) files contain information about the network traffic that can be explored in detail. The goal of this exercise is to examine PCAP files containing attacks, and analyze the traffic in detail. We look into details of transport- (TCP) and application-layer (SMB, RPC, HTTP). Wireshark (and the command-line tool, tshark) will be used in the tasks.

**Some Useful Examples of Appying Filters on pcap files:-**

• ip.addr == 10.0.0.1 [Sets a filter for any packet with 10.0.0.1, as either the source or dest]

• ip.addr==10.0.0.1 && ip.addr==10.0.0.2 [sets a conversation filter between the two defined IP addresses]

• http or dns [sets a filter to display all http and dns]

• tcp.port==4000 [sets a filter for any TCP packet with 4000 as a source or dest port]

• tcp.flags.reset==1 [displays all TCP resets]

• http.request [displays all HTTP GET requests]

• tcp contains reviews [displays all TCP packets that contain the word ‘reviews’. Excellent when searching on a specific string or user ID]

• !(arp or icmp or dns) [masks out arp, icmp, dns, or whatever other protocols may be background noise. Allowing you to focus on the traffic of interest]

• tcp.flags.syn eq 1 [TCP packets with SYN flag set to 1]

Q1. Refer to file Ass9\_1.pcap and answer following questions:-

1. Find IP addresses of FTP server and Client

Sol.

Source(client): 10.121.70.151

Dest.(Server) : 10.234.125.254

1. What is the meaning of FTP Err Code 530.

Sol. Login Error.

1. How many login errors occurred within a minute?

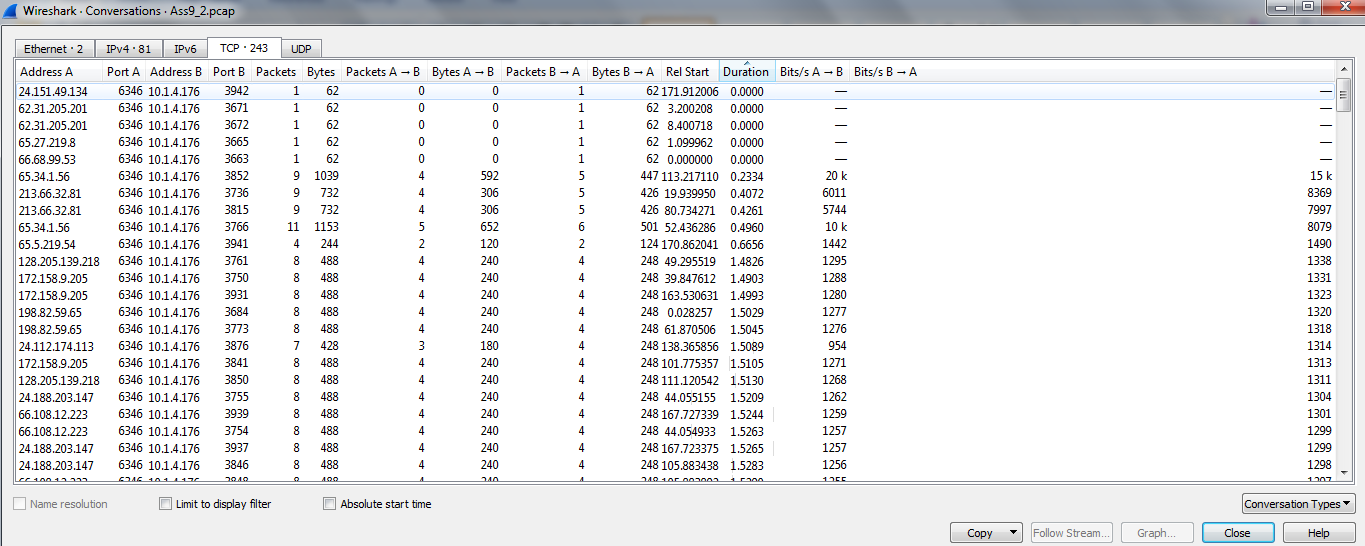
Sol. 16

Q2. Refer to file Ass9\_2.pcap and answer following questions:-

1. Explore what’s going on with wireshark gui:-

• Statistics -> Conversation List -> TCP (\*)

Sol.



1. Explore web to find out which application use TCP/6346?

Sol. GNUtella

Q3. Analyze and point out the difference between the captures in Ass9\_3a.pcap and Ass9\_3b.pcap

Sol.

**Ass9\_3b.pcap:**

**Statistics**

|  |  |  |  |
| --- | --- | --- | --- |
| Measurement | Captured | Displayed | Marked |
| Packets | 100 | 100 (100.0%) | — |
| Time span, s | 0.016 | 0.016 | — |
| Average pps | 6167.1 | 6167.1 | — |
| Average packet size, B | 54 | 54 | — |
| Bytes | 5400 | 5400 (100.0%) | 0 |
| Average bytes/s | 333 k | 333 k | — |
| Average bits/s | 2664 k | 2664 k | — |

**Ass9\_3a.pcap:**

**Statistics**

|  |  |  |  |
| --- | --- | --- | --- |
| Measurement | Captured | Displayed | Marked |
| Packets | 29 | 29 (100.0%) | — |
| Time span, s | 2.820 | 2.820 | — |
| Average pps | 10.3 | 10.3 | — |
| Average packet size, B | 60 | 60 | — |
| Bytes | 1740 | 1740 (100.0%) | 0 |
| Average bytes/s | 617 | 617 | — |
| Average bits/s | 4936 | 4936 | — |