

# ***INTERNET SECURITY LAB***

## ***ETCS-451***

Faculty name: Mr Alok

Student name: Chandrashekhhar

Roll No.: 47214802717

Semester: 7<sup>th</sup>

Group: 7C8



Maharaja Agrasen Institute of Technology, PSP Area,

Sector – 22, Rohini, New Delhi – 110085

## Index

Exp No	Experiment Name	Date of performance	Date of checking	Marks	Sign.
1.	Make an experiment to implement WEP/WPA 2 PSK, 802.1x EAP Security Protocol.				
2.	To implement Windows Firewall.				
3.	PERFORM THE FOLLOWING ENCRYPTION AND DECRYPTION ALGORITHMS: 1. Shift Cipher 2. Substitution Cipher 3. Vigenere Cipher 4. Affine Cipher 5. Hill Cipher 6. Permutation Cipher 7. Playfair				
4.	Implement RSA Algorithm				
5.	Implement DES Algorithm				
6.	To implement Diffie-Hellman Algorithm				
7.	Study of NeSSi2 Simulation Tool based on parameters of IS.				
8.	To implement VPN through Network Simulator Tool.				

## EXPERIMENT – 1

**Aim:** Make an experiment to implement WEP/WPA 2 PSK, 802.1x EAP Security Protocol.

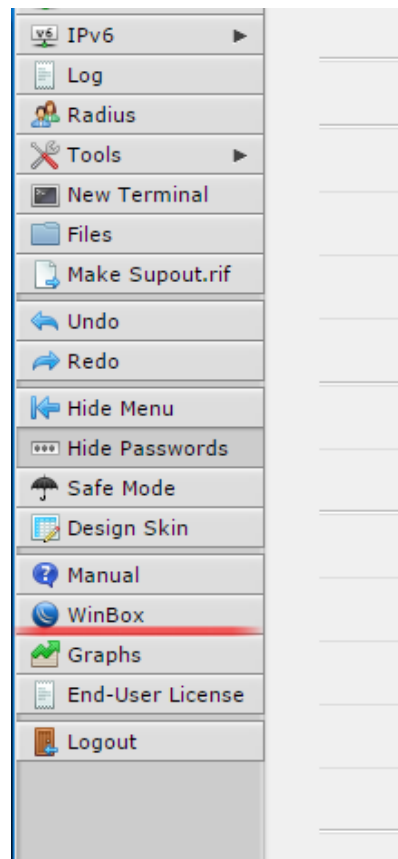
### Theory:

Winbox is a small utility that allows administration of MikrotikRouterOS using a fast and simple GUI. It is a native Win32 binary, but can be run on **Linux** and **MacOS (OSX)** using Wine.

All Winbox interface functions are as close as possible to Console functions, that is why there are no Winbox sections in the manual.

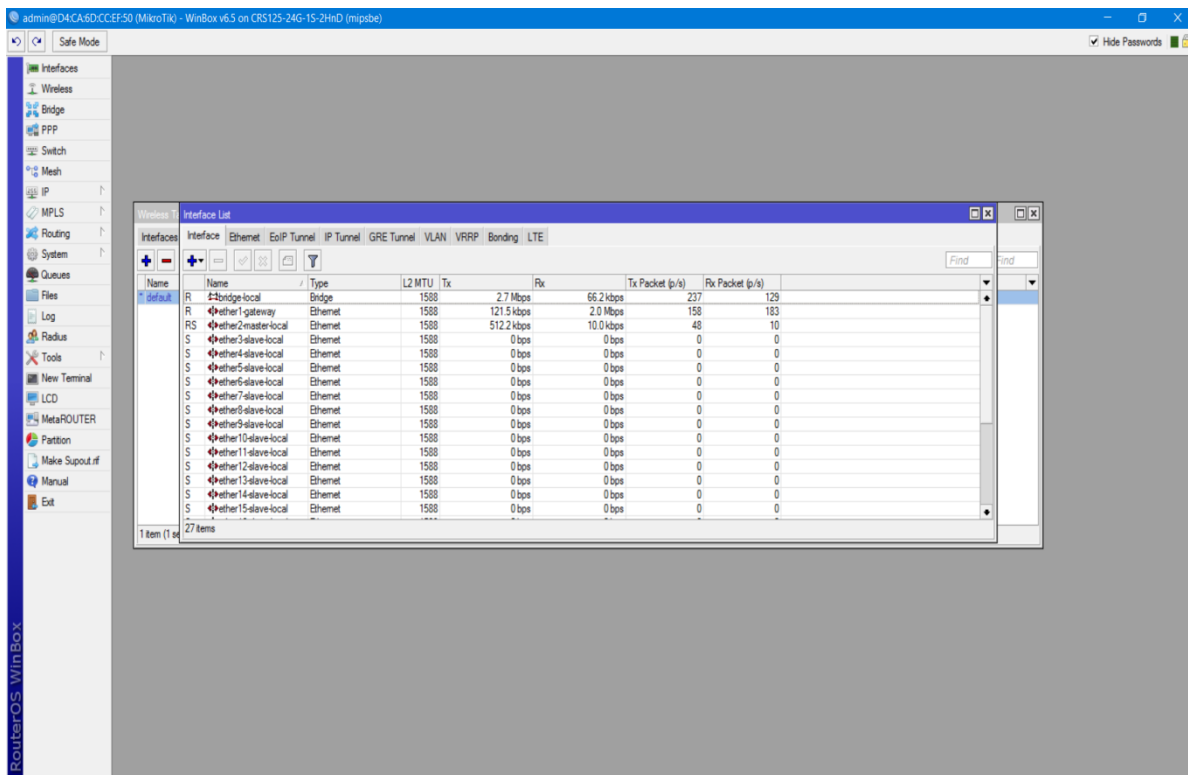
Some of advanced and system critical configurations are not possible from winbox, like MAC address change on an interface.

Winbox loader can be downloaded directly from the router or from the [mikrotik download page](#). When downloading from the router, open a web browser and enter router's IP address, RouterOS welcome page will be displayed. Click on the menu item that says **Winbox** to download **winbox.exe** from MikroTik download server.



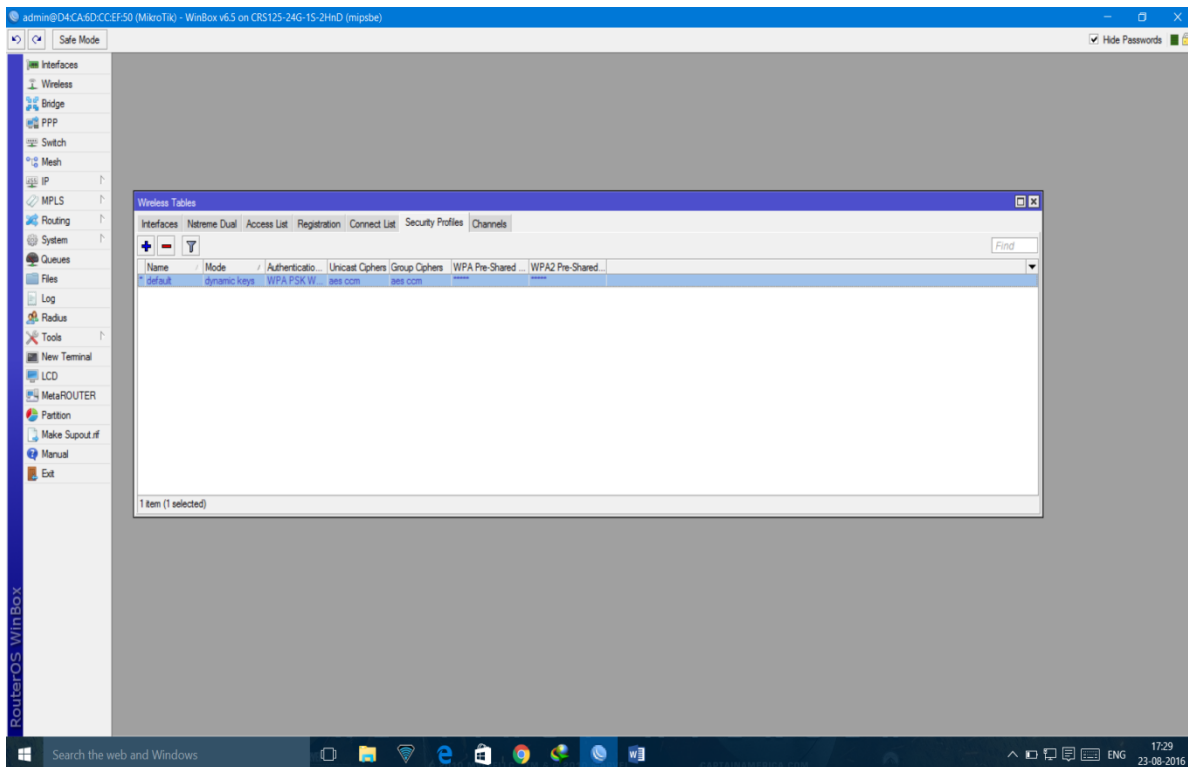
To connect to the router enter IP or MAC address of the router, specify username and password (if any) and click on **Connect** button. You can also enter the port number after the IP address,

separating them with a colon, like this 192.168.88.1:9999. The port can be changed in RouterOS **services** menu.



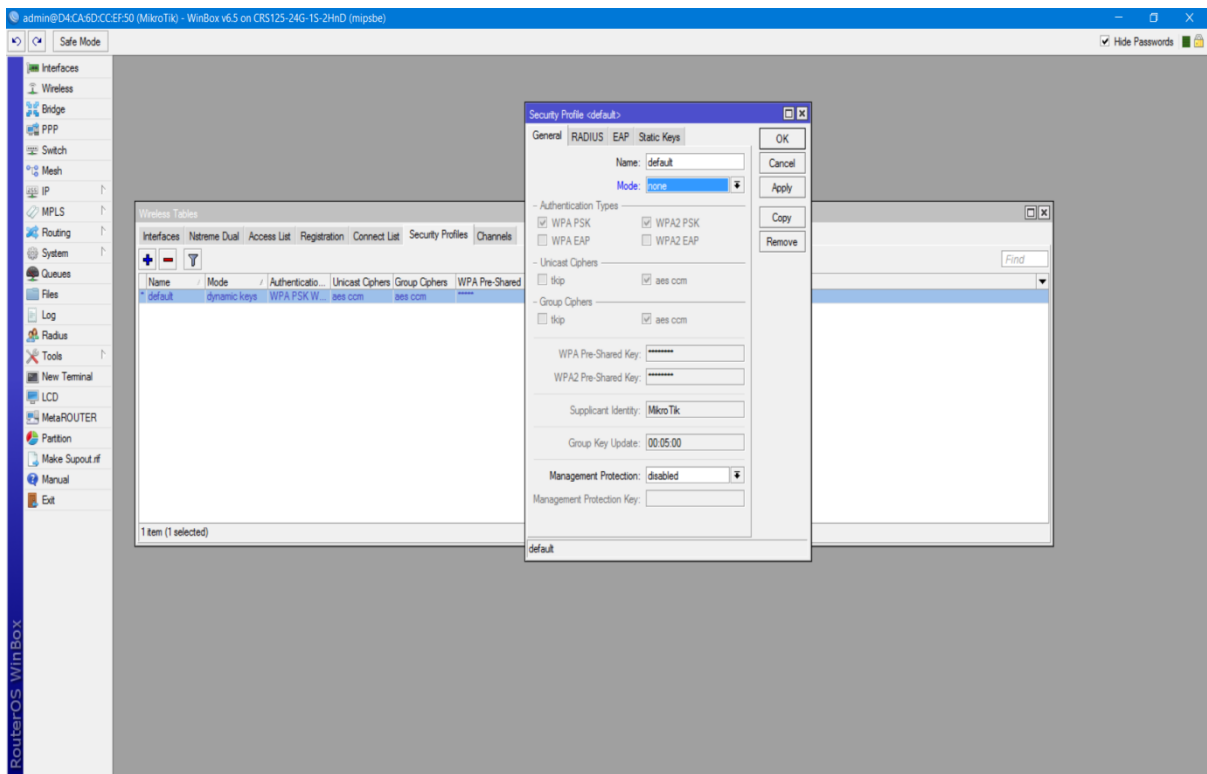
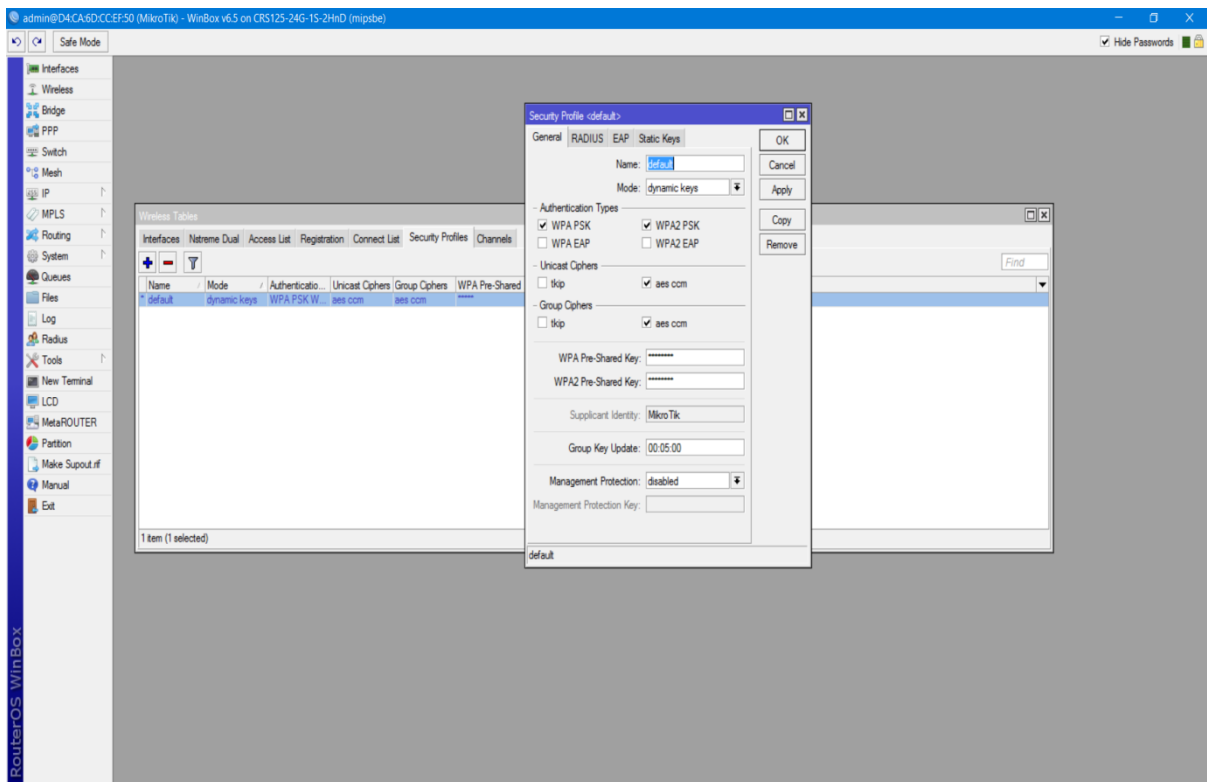
The screenshot shows the RouterOS WinBox interface. The left sidebar contains a menu with options: Interfaces, Wireless, Bridge, PPP, Switch, Mesh, IP, MPLS, Routing, System, Queues, Files, Log, Radius, Tools, New Terminal, LCD, MetaROUTER, Partition, Make Supout.tif, Manual, and Exit. The main window displays the 'Interface List' window, which is a table showing the status of various network interfaces. The table has columns for Name, Type, L2 MTU, Tx, Rx, Tx Packet (p/s), and Rx Packet (p/s). The 'default' interface is selected, showing a list of 15 slave interfaces (ether1-slave-local through ether15-slave-local) and their respective statistics.

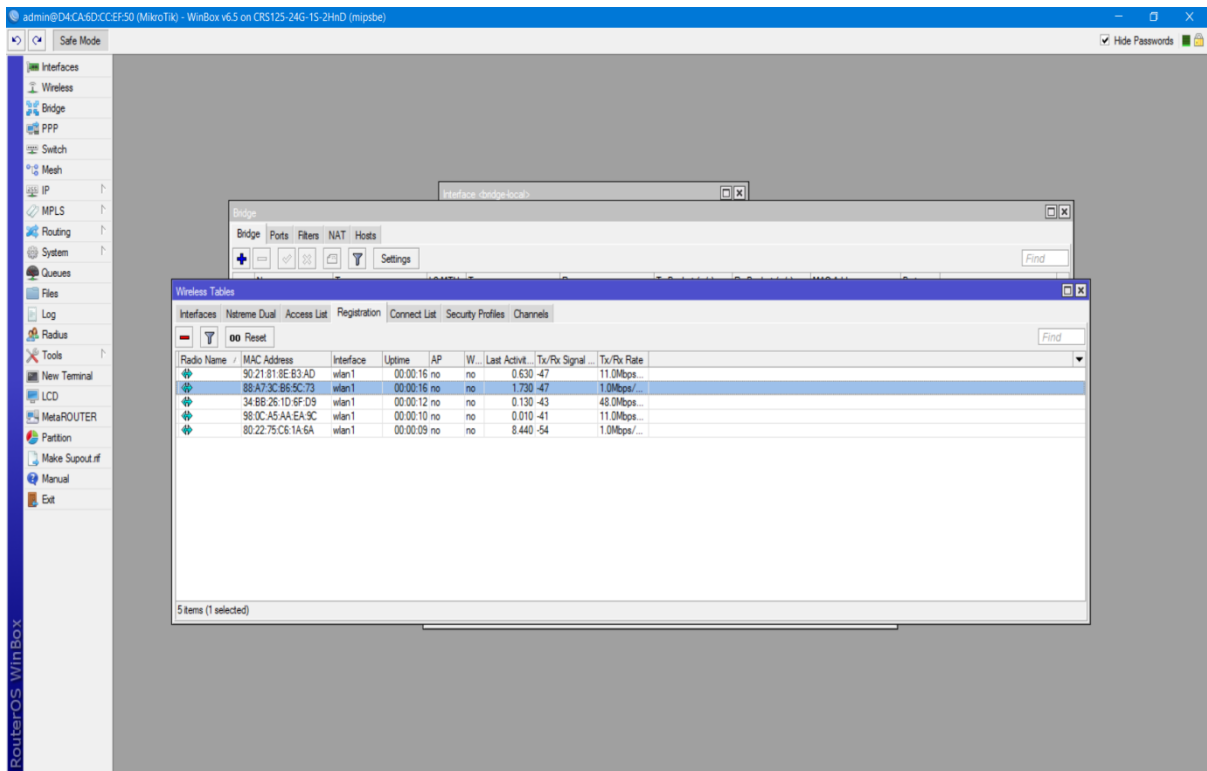
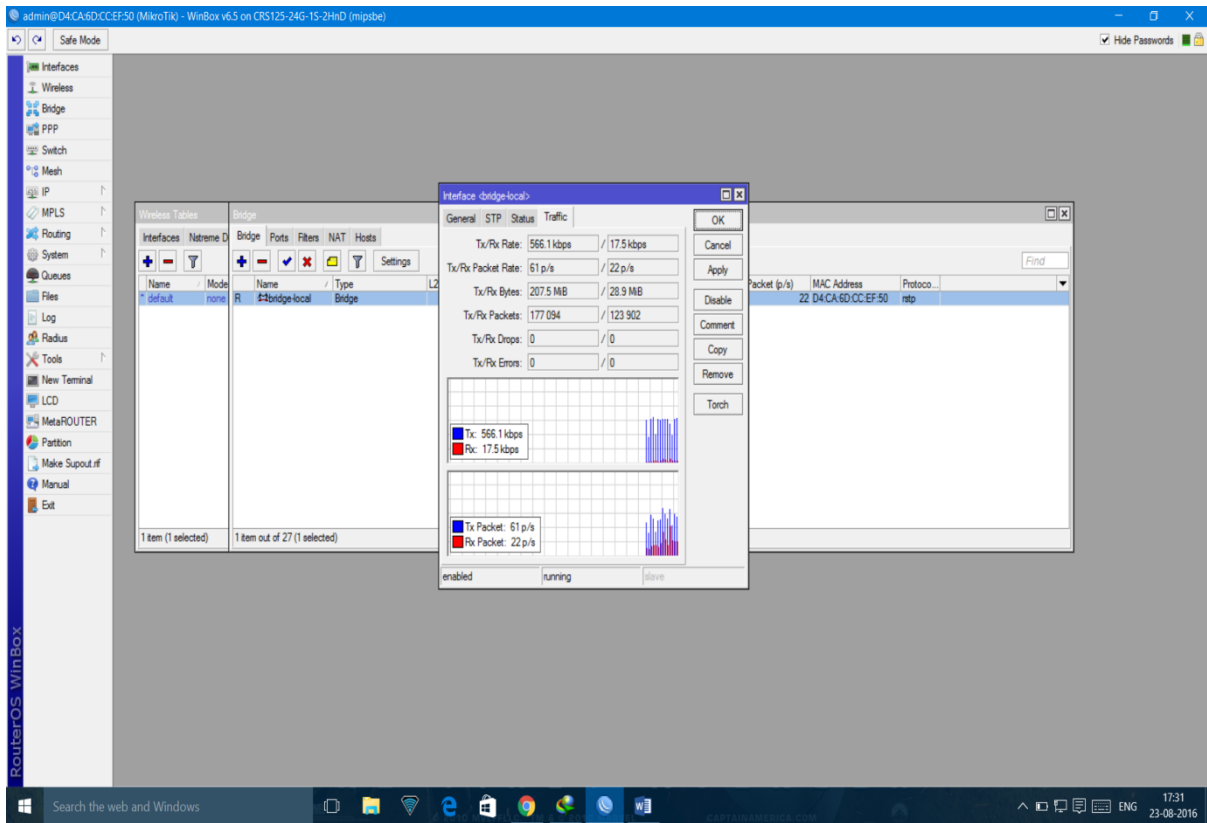
Name	Type	L2 MTU	Tx	Rx	Tx Packet (p/s)	Rx Packet (p/s)
R ether1-gateway	Ethernet	1500	121.5 kbps	2.0 Mbps	158	183
RS ether2-master-local	Ethernet	1500	512.2 kbps	10.0 kbps	48	10
S ether3-slave-local	Ethernet	1500	0 bps	0 bps	0	0
S ether4-slave-local	Ethernet	1500	0 bps	0 bps	0	0
S ether5-slave-local	Ethernet	1500	0 bps	0 bps	0	0
S ether6-slave-local	Ethernet	1500	0 bps	0 bps	0	0
S ether7-slave-local	Ethernet	1500	0 bps	0 bps	0	0
S ether8-slave-local	Ethernet	1500	0 bps	0 bps	0	0
S ether9-slave-local	Ethernet	1500	0 bps	0 bps	0	0
S ether10-slave-local	Ethernet	1500	0 bps	0 bps	0	0
S ether11-slave-local	Ethernet	1500	0 bps	0 bps	0	0
S ether12-slave-local	Ethernet	1500	0 bps	0 bps	0	0
S ether13-slave-local	Ethernet	1500	0 bps	0 bps	0	0
S ether14-slave-local	Ethernet	1500	0 bps	0 bps	0	0
S ether15-slave-local	Ethernet	1500	0 bps	0 bps	0	0



The screenshot shows the RouterOS WinBox interface. The left sidebar contains a menu with options: Interfaces, Wireless, Bridge, PPP, Switch, Mesh, IP, MPLS, Routing, System, Queues, Files, Log, Radius, Tools, New Terminal, LCD, MetaROUTER, Partition, Make Supout.tif, Manual, and Exit. The main window displays the 'Wireless Tables' window, which is a table showing the status of various wireless interfaces. The table has columns for Name, Mode, Authentication, Unicast Ciphers, Group Ciphers, WPA Pre-Shared, and WPA2 Pre-Shared. The 'default' interface is selected, showing a list of 15 slave interfaces (ether1-slave-local through ether15-slave-local) and their respective statistics.

Name	Mode	Authentication	Unicast Ciphers	Group Ciphers	WPA Pre-Shared	WPA2 Pre-Shared
default	dynamic keys	WPA PSK W	aes com	aes com	-----	-----





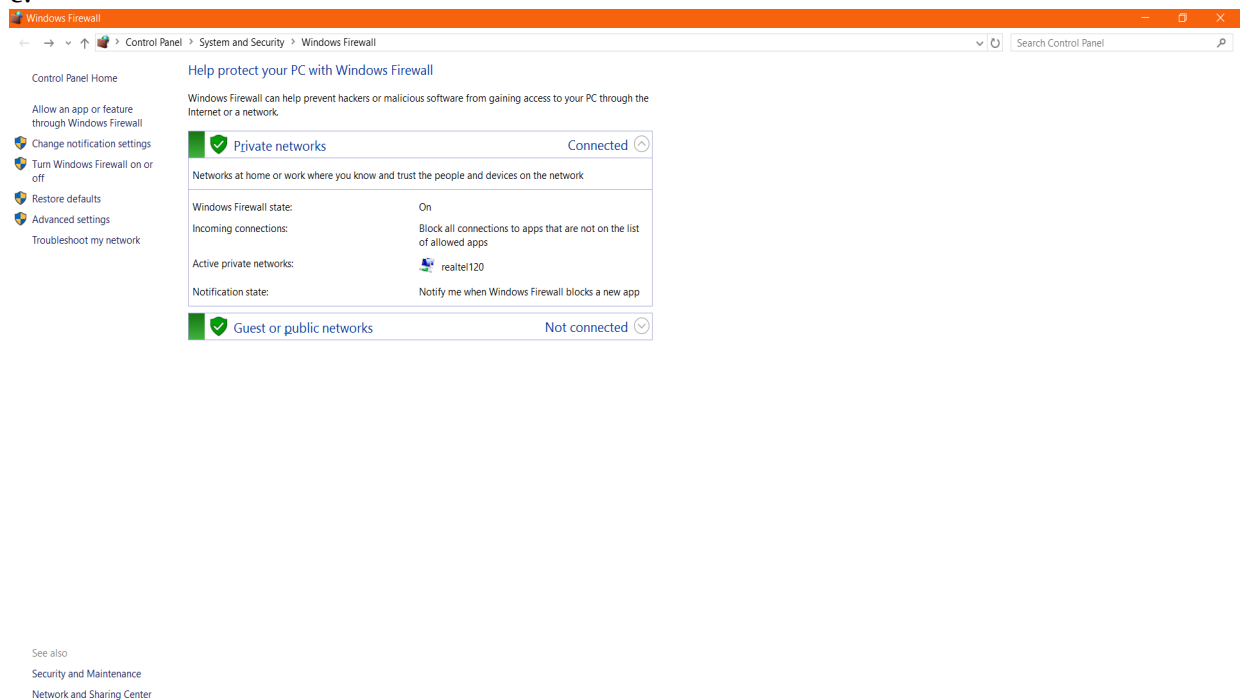
## EXPERIMENT – 2

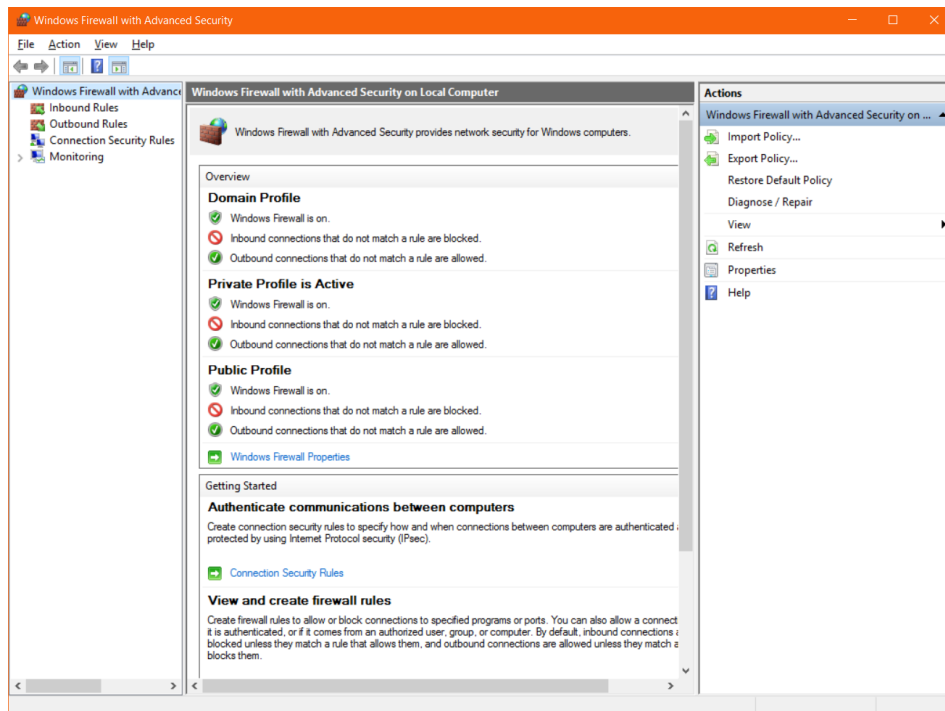
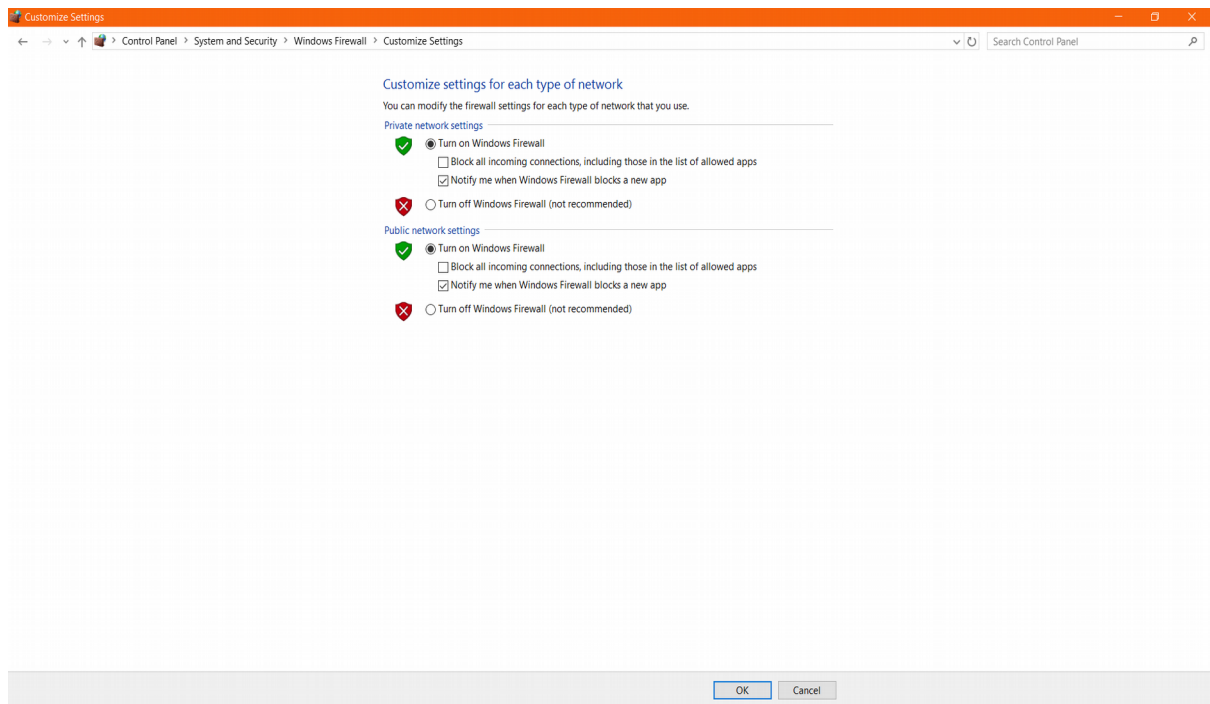
**Aim:** To implement Windows Firewall.

**Theory:**

To enable Windows Firewall and configure the default behaviour on Windows 7, Windows Vista, Windows Server 2008, or Windows Server 2008 R2

1. Open the Group Policy Management Console to Windows Firewall with Advanced Security.
2. In the details pane, in the **Overview** section, click **Windows Firewall Properties**.
3. For each network location type (Domain, Private, Public), perform the following steps.
  - a. Click the tab that corresponds to the network location type.
  - b. Change **Firewall state** to **On (recommended)**.
  - c. Change **Inbound connections** to **Block (default)**.
  - d. Change **Outbound connections** to **Allow (default)**.
  - e.







## EXPERIMENT - 3

### AIM: PERFORM THE FOLLOWING ENCRYPTION AND DECRYPTION ALGORITHMS:

1. Shift Cipher
2. Substitution Cipher
3. Vigenere Cipher
4. Affine Cipher
5. Hill Cipher
6. Permutation Cipher
7. Playfair

#### 1 SHIFT CIPHER:

It is a mono-alphabetic cipher wherein each letter of the plaintext is substituted by another letter to form the ciphertext. It is a simplest form of substitution cipher scheme. This cryptosystem is generally referred to as the Shift Cipher. The concept is to replace each alphabet by another alphabet which is 'shifted' by some fixed number between 0 and 25. For this type of scheme, both sender and receiver agree on a 'secret shift number' for shifting the alphabet. This number which is between 0 and 25 becomes the key of encryption. The name 'Caesar Cipher' is occasionally used to describe the Shift Cipher when the 'shift of three' is used.

```
#include<iostream>

using namespace std;

int main()
{
    char message[100], ch;
    int i, key;
    cout<<"Chandrashekhar, 47214802717"<<endl<<endl;
    cout << "Enter a message to encrypt: ";
    cin.getline(message, 100);
    cout << "Enter key: ";
    cin >> key;

    for(i = 0; message[i] != '\0'; ++i){
```

```

ch = message[i];
if(ch >= 'a' && ch <= 'z'){
    ch = ch + key;
    if(ch > 'z'){
        ch = ch - 'z' + 'a' - 1;
    }
    message[i] = ch;
}
else if(ch >= 'A' && ch <= 'Z'){
    ch = ch + key;
    if(ch > 'Z'){
        ch = ch - 'Z' + 'A' - 1;
    }
    message[i] = ch;
}
}

cout << "Encrypted message: " << message<<endl;

for(i = 0; message[i] != '\0'; ++i){
    ch = message[i];
    if(ch >= 'a' && ch <= 'z'){
        ch = ch - key;
        if(ch < 'a'){
            ch = ch + 'z' - 'a' + 1;
        }
        message[i] = ch;
    }
    else if(ch >= 'A' && ch <= 'Z'){
        ch = ch - key;
        if(ch < 'A'){
            ch = ch + 'Z' - 'A' + 1;
        }
        message[i] = ch;
    }
}

```

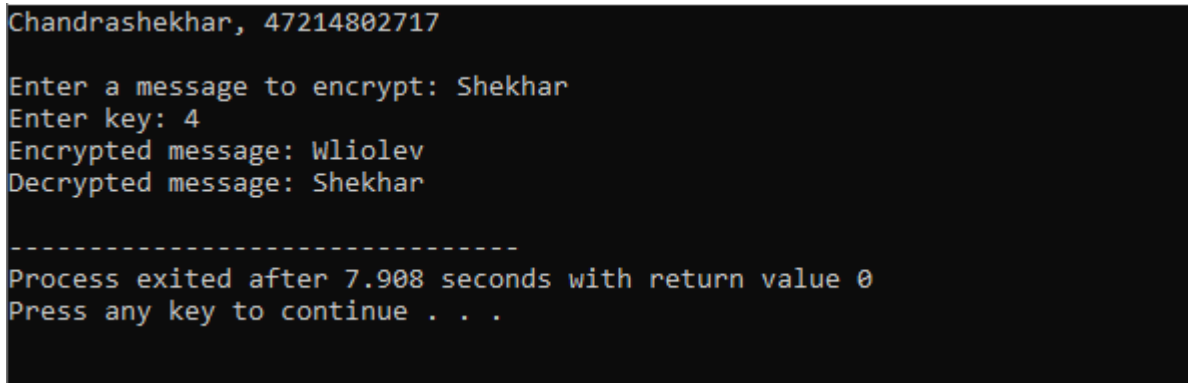
```

    }
}

cout << "Decrypted message: " << message<<endl;

return 0;
}

```



```

Chandrashekhar, 47214802717

Enter a message to encrypt: Shekhar
Enter key: 4
Encrypted message: Wliolev
Decrypted message: Shekhar

-----
Process exited after 7.908 seconds with return value 0
Press any key to continue . . .

```

## 2 SUBSTITUTION CIPHER:

In cryptography, a simple substitution cipher is a cipher that has been in use for many hundreds of years. It is an improvement to the [Caesar Cipher](#). Instead of shifting the alphabets by some number, this scheme substitutes every plaintext character for a different ciphertext character.

With 26 letters in alphabet, the possible permutations are 26! (Factorial of 26) which is equal to  $4 \times 10^{26}$ . The sender and the receiver may choose any one of these possible permutation as a ciphertext alphabet. This permutation is the secret key of the scheme.

```

#include <string>

#include <iostream>

using namespace std;

void encrypt(char * input, unsigned int offset) {
    for (int i = 0; input[i] != 0; i++) {
        //      Skip spaces...
        if (input[i] == ' ')
            continue;

        char firstLetter = islower(input[i]) ? 'a' : 'A';
        unsigned int
            alphaOffset = input[i] - firstLetter,
            newAlphaOffset = alphaOffset+offset;

        input[i] = firstLetter + newAlphaOffset % 26;
    }
}

```

```
    }  
}
```

```
void decrypt(char * input, unsigned int offset) {  
    for (int i = 0; input[i] != 0; i++) {  
        //      Skip spaces...  
        if (input[i] == ' ')  
            continue;  
  
        char firstLetter = islower(input[i]) ? 'a' : 'A';  
        unsigned int alphaOffset = input[i] - firstLetter;  
        int newAlphaOffset = alphaOffset - offset;  
        if (newAlphaOffset < 0) {  
            newAlphaOffset += 26;  
        }  
        input[i] = firstLetter + (newAlphaOffset % 26);  
    }  
}
```

```
int main() {  
    std::string alphabeticalString = "abcdefghijklmnopqrstuvwxyz";  
    unsigned int encryptionOffset;  
    cout<<"Chandrashekhar, 47214802717"<<endl;  
    cout << "string to encrypt:";  
    getline(cin, alphabeticalString);  
    cout << "encryption key (between 0 and 25):";  
    cin >> encryptionOffset;  
    encrypt(const_cast<char*>(alphabeticalString.c_str()), encryptionOffset);  
    cout << "Encrypted Message: " << alphabeticalString << endl;  
    decrypt(const_cast<char*>(alphabeticalString.c_str()), encryptionOffset);  
    cout << "Decrypted Message: " << alphabeticalString <<endl;  
    return 0;  
}
```

```
Chandrashekhar, 47214802717
string to encrypt:Shekhar
encryption key (between 0 and 25):12
Encrypted Message: Etqwtmd
Decrypted Message: Shekhar

-----
Process exited after 13.47 seconds with return value 0
Press any key to continue . . .
```

### 3 VIGENERE CIPHER:

Vigenere Cipher is a method of encrypting alphabetic text. It uses a simple form of [polyalphabetic substitution](#). A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets. The encryption of the original text is done using the [Vigenère square or Vigenère table](#).

- The table consists of the alphabets written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible [Caesar Ciphers](#).
- At different points in the encryption process, the cipher uses a different alphabet from one of the rows.
- The alphabet used at each point depends on a repeating keyword.

```
#include <iostream>

#include <cstring>

using namespace std;

class Vig {
public:
    string k;

    Vig(string k) {
        for (int i = 0; i < k.size(); ++i) {
            if (k[i] >= 'A' && k[i] <= 'Z')
                this->k += k[i];
            else if (k[i] >= 'a' && k[i] <= 'z')
                this->k += k[i] + 'A' - 'a';
        }
    }

    string encryption(string t) {
```

```

string output;
for (int i = 0, j = 0; i < t.length(); ++i) {
    char c = t[i];
    if (c >= 'a' && c <= 'z')
        c += 'A' - 'a';
    else if (c < 'A' || c > 'Z')
        continue;
    output += (c + k[j] - 2 * 'A') % 26 + 'A'; //added 'A' to bring it in range of ASCII alphabet [ 65-90 | A-Z ]
    j = (j + 1) % k.length();
}
return output;
}

string decryption(string t) {
    string output;
    for (int i = 0, j = 0; i < t.length(); ++i) {
        char c = t[i];
        if (c >= 'a' && c <= 'z')
            c += 'A' - 'a';
        else if (c < 'A' || c > 'Z')
            continue;
        output += (c - k[j] + 26) % 26 + 'A'; //added 'A' to bring it in range of ASCII alphabet [ 65-90 | A-Z ]
        j = (j + 1) % k.length();
    }
    return output;
}

};

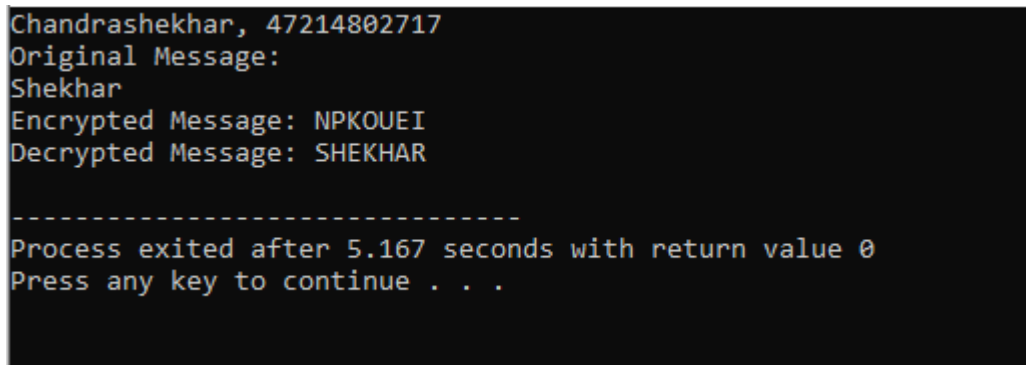
int main() {
    Vig v("VIGENERE"); //pass the keyword
    string ori;
    cout<<"Chandrashekhar, 47214802717"<<endl;

```

```

cout << "Original Message: " << ori << endl;
getline(cin, ori);
string encrypt = v.encryption(ori);
string decrypt = v.decryption(encrypt);
cout << "Encrypted Message: " << encrypt << endl;
cout << "Decrypted Message: " << decrypt << endl;
}

```



```

Chandrashekhar, 47214802717
Original Message:
Shekhar
Encrypted Message: NPKOUEI
Decrypted Message: SHEKHAR

-----
Process exited after 5.167 seconds with return value 0
Press any key to continue . . .

```

#### 4 AFFINE CIPHER:

The Affine cipher is a type of monoalphabetic substitution cipher, wherein each letter in an alphabet is mapped to its numeric equivalent, encrypted using a simple mathematical function, and converted back to a letter. The formula used means that each letter encrypts to one other letter, and back again, meaning the cipher is essentially a standard substitution cipher with a rule governing which letter goes to which.

The whole process relies on working modulo  $m$  (the length of the alphabet used). In the affine cipher, the letters of an alphabet of size  $m$  are first mapped to the integers in the range  $0 \dots m-1$ .

The 'key' for the Affine cipher consists of 2 numbers, we'll call them  $a$  and  $b$ . The following discussion assumes the use of a 26 character alphabet ( $m = 26$ ).  $a$  should be chosen to be relatively prime to  $m$  (i.e.  $a$  should have no factors in common with  $m$ ).

```
#include<iostream>
```

```
#include<string>
```

```
using namespace std;
```

```
//Key values of a and b
```

```
const int a = 17;
```

```
const int b = 20;
```

```
string encryption(string m) {
```

```

string c = "";
for (int i = 0; i < m.length(); i++) {
    if(m[i]!=' ')
        c = c + (char) (((a * (m[i]-'A') ) + b) % 26) + 'A');
    else
        c += m[i];
}
return c;
}

string decryption(string c) {
    string m = "";
    int a_inverse = 0;
    int flag = 0;
    for (int i = 0; i < 26; i++) {
        flag = (a * i) % 26;
        if (flag == 1) {
            a_inverse = i;
        }
    }
    for (int i = 0; i < c.length(); i++) {
        if(c[i] != ' ')
            m = m + (char) (((a_inverse * ((c[i]+'A' - b)) % 26)) + 'A');
        else
            m += c[i];
    }
    return m;
}

int main(void) {
    string msg;
    cout<<"Chandrashekhar, 47214802717"<<endl;
    cout<<"Enter message to encrypt:";
    getline(cin,msg);

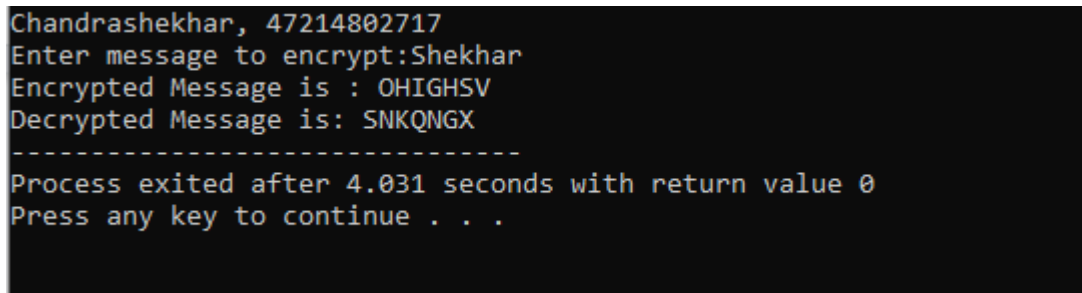
```



```

string c = encryption(msg);
cout << "Encrypted Message is : " << c<<endl;
cout << "Decrypted Message is: " << decryption(c);
return 0;
}

```



```

Chandrashekhar, 47214802717
Enter message to encrypt:Shekhar
Encrypted Message is : OHIGHSV
Decrypted Message is: SNKQNGX
-----
Process exited after 4.031 seconds with return value 0
Press any key to continue . . .

```

## 5 HILL CIPHER:

Based on linear algebra Hill cipher is a polygraphic substitution cipher in cryptography. To encrypt message: The key string and message string are represented as matrix form. They are multiplied then, against modulo 26. The key matrix should have inverse to decrypt the message. To decrypt message: The encrypted message is multiplied by inverse key matrix used for encryption against modulo 26 to get decrypt message.

```

#include<iostream>

#include<math.h>

using namespace std;

float en[3][1], de[3][1], a[3][3], b[3][3], msg[3][1], m[3][3];

void getKeyMatrix() { //get key and message from user
    int i, j;
    char mes[3];

    cout<<"Enter 3x3 matrix for key (should have inverse):\n";
    for(i = 0; i < 3; i++)
        for(j = 0; j < 3; j++) {
            cin>>a[i][j];
            m[i][j] = a[i][j];
        }

    cout<<"\nEnter a string of 3 letter(use A through Z): ";
}

```

```

cin>>mes;

for(i = 0; i < 3; i++)
    msg[i][0] = mes[i] - 65;
}

void encrypt() { //encrypts the message
    int i, j, k;
    for(i = 0; i < 3; i++)
        for(j = 0; j < 1; j++)
            for(k = 0; k < 3; k++)
                en[i][j] = en[i][j] + a[i][k] * msg[k][j];
    cout<<"\nEncrypted string is: ";
    for(i = 0; i < 3; i++)
        cout<<(char)(fmod(en[i][0], 26) + 65); //modulo 26 is taken for each element of the matrix
        obtained by multiplication
    }

void inversematrix() { //find inverse of key matrix
    int i, j, k;
    float p, q;
    for(i = 0; i < 3; i++)
        for(j = 0; j < 3; j++) {
            if(i == j)
                b[i][j]=1;
            else
                b[i][j]=0;
        }
    for(k = 0; k < 3; k++) {
        for(i = 0; i < 3; i++) {
            p = m[i][k];
            q = m[k][k];
            for(j = 0; j < 3; j++) {
                if(i != k) {
                    m[i][j] = m[i][j]*q - p*m[k][j];

```

```

b[i][j] = b[i][j]*q - p*b[k][j];
}
}
}
}
for(i = 0; i < 3; i++)
for(j = 0; j < 3; j++)
b[i][j] = b[i][j] / m[i][i];
cout<<"\n\nInverse Matrix is:\n";
for(i = 0; i < 3; i++) {
for(j = 0; j < 3; j++)
cout<<b[i][j]<<" ";
cout<<"\n";
}
}

void decrypt() { //decrypt the message
int i, j, k;
inversematrix();
for(i = 0; i < 3; i++)
for(j = 0; j < 1; j++)
for(k = 0; k < 3; k++)
de[i][j] = de[i][j] + b[i][k] * en[k][j];
cout<<"\nDecrypted string is: ";
for(i = 0; i < 3; i++)
cout<<(char)(fmod(de[i][0], 26) + 65); //modulo 26 is taken to get the original message
cout<<"\n";
}

int main() {
cout<<"Chandrashekhar, 47214802717"<<endl;
getKeyMatrix();
encrypt();
decrypt();
}

```

```
}
```

```
Chandrashekhar, 47214802717
Enter 3x3 matrix for key (should have inverse):
4 6 5
2 8 4
5 8 6

Enter a string of 3 letter(use A through Z):
ABC

Encrypted string is: QQU

Inverse Matrix is:
-2 -0.5 2
-1 0.125 0.75
3 0.25 -2.5

Decrypted string is: ABC

-----
Process exited after 27.19 seconds with return value 0
Press any key to continue . . .
```

## 6 PERMUTATION CIPHER

The Permutation Cipher is another form of Transposition Cipher. It is similar to Columnar Transposition in some ways, in that the columns are written in the same way, including how the keyword is used. However, the Permutation Cipher acts on blocks of letters (the lengths of the keyword), rather than the whole ciphertext.

### ENCRYPTION

We choose a keyword, and split the plaintext into blocks that are the same length as the keyword. We write this in columns beneath the keyword. We then label each keyword letter in alphabetical order (if there are duplicates we take them in order of appearance). So far this is identical to Columnar Transposition. Now we reorder the columns, so that the numbers are in order (the letters of the keyword are in alphabetical order). We now read across the rows. As an example we shall encrypt the plaintext "the quick brown fox jumped over the lazy dog" using the keyword bad.

### DECRYPTION

To decrypt a ciphertext encoded with the Permutation Cipher, we have to write out the ciphertext in columns (the same number as the length of the keyword). We then order the keyword alphabetically, and write the ordered keyword at the top of the columns. We then rearrange the columns to reform the keyword, and read off the plaintext in rows.

```
#include<bits/stdc++.h>
```

```
using namespace std;
```

```
string const key = "HACK";
```

```
map<int,int> keyMap;
```

```
void setPermutationOrder()
```

```
{  
    // Add the permutation order into map  
    for(int i=0; i < key.length(); i++)  
    {  
        keyMap[key[i]] = i;  
    }  
}
```

```
// Encryption
```

```
string encryptMessage(string msg)
```

```
{  
    int row,col,j;  
    string cipher = "";  
    col = key.length();  
    row = msg.length()/col;
```

```
    if (msg.length() % col)  
        row += 1;
```

```
    char matrix[row][col];
```

```
    for (int i=0,k=0; i < row; i++)
```

```
    {  
        for (int j=0; j<col; )
```

```
        {  
            if(msg[k] == '\0')
```

```
            {  
                /* Adding the padding character '_' */  
                matrix[i][j] = '_';
```

```
j++;
```

```
}
```

```
if( isalpha(msg[k]) || msg[k]==' ')
```

```
{
```

```
matrix[i][j] = msg[k];
```

```
j++;
```

```
}
```

```
k++;
```

```
}
```

```
}
```

```
for (map<int,int>::iterator ii = keyMap.begin(); ii!=keyMap.end(); ++ii)
```

```
{
```

```
j=ii->second;
```

```
for (int i=0; i<row; i++)
```

```
{
```

```
if( isalpha(matrix[i][j]) || matrix[i][j]==' ' || matrix[i][j]=='_')
```

```
cipher += matrix[i][j];
```

```
}
```

```
}
```

```
return cipher;
```

```
}
```

```
// Decryption
```

```
string decryptMessage(string cipher)
```

```
{
```

```
int col = key.length();
```

```
int row = cipher.length()/col;
```

```
char cipherMat[row][col];
```

```
for (int j=0,k=0; j<col; j++)
```

```
for (int i=0; i<row; i++)
```

```
cipherMat[i][j] = cipher[k++];
```

```
int index = 0;
```

```
for( map<int,int>::iterator ii=keyMap.begin(); ii!=keyMap.end(); ++ii)
```

```
ii->second = index++;
```

```
char decCipher[row][col];
```

```
map<int,int>::iterator ii=keyMap.begin();
```

```
int k = 0;
```

```
for (int l=0,j; key[l]!='\0'; k++)
```

```
{
```

```
j = keyMap[key[l++]];
```

```
for (int i=0; i<row; i++)
```

```
{
```

```
decCipher[i][k]=cipherMat[i][j];
```

```
}
```

```
}
```

```
string msg = "";
```

```
for (int i=0; i<row; i++)
```

```
{
```

```
for(int j=0; j<col; j++)
```

```
{
```

```
if(decCipher[i][j] != '_')
```

```
msg += decCipher[i][j];
```

```
}
```

```
}
```

```
return msg;
```

```

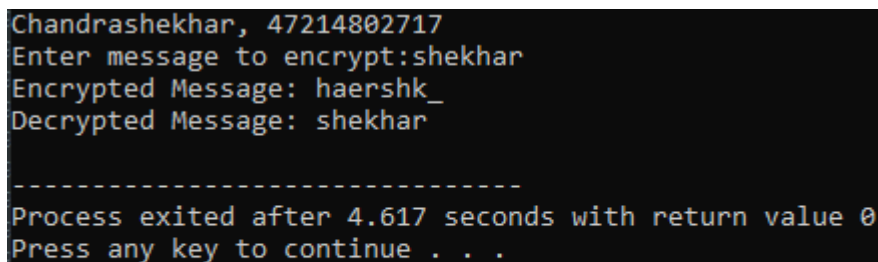
}

// Driver Program
int main(void)
{

    string msg;
    cout<<"Chandrashekhar, 47214802717"<<endl;
    cout<<"Enter message to encrypt:";
    getline(cin,msg);
    setPermutationOrder();
    string cipher = encryptMessage(msg);
    cout << "Encrypted Message: " << cipher << endl;
    cout << "Decrypted Message: " << decryptMessage(cipher) << endl;

    return 0;
}

```



```

Chandrashekhar, 47214802717
Enter message to encrypt:shekhar
Encrypted Message: haershk_
Decrypted Message: shekhar

-----
Process exited after 4.617 seconds with return value 0
Press any key to continue . . .

```

## 7 PLAYFAIR CIPHER:

In the playfair cipher, initially a key table is created. The key table is a 5×5 grid of alphabets that acts as the key for encrypting the plaintext. Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table as we need only 25 alphabets instead of 26. If the plaintext contains J, then it is replaced by I. The sender and the receiver decide on a particular key, say 'tutorials'. In a key table, the first characters (going left to right) in the table is the phrase, excluding the duplicate letters. The rest of the table will be filled with the remaining letters of the alphabet, in natural order.



```

#include <iostream>
#include <string>
using namespace std;
class playfair {
public:
    string msg; char n[5][5];
    void play( string k, string t, bool m, bool e ) {
        createEncoder( k, m );
        getText( t, m, e );
        if( e )
            play( 1 );
        else
            play( -1 );
        print();
    }
private:
    void play( int dir ) {
        int j,k,p,q;
        string nmsg;
        for( string::const_iterator it = msg.begin(); it != msg.end(); it++ ) {
            if( getPos( *it++, j, k ))
                if( getPos( *it, p, q)) {
                    //for same row
                    if( j == p ) {
                        nmsg+= getChar( j, k + dir );
                        nmsg += getChar( p, q + dir );
                    }
                    //for same column
                    else if( k == q ) {
                        nmsg += getChar( j + dir, k );
                        nmsg += getChar( p + dir, q );
                    } else {

```

```

nmsg += getChar( p, k );
nmsg += getChar( j, q ); } } }
msg = nmsg;
}
void print() {
cout << "\n\n Solution:" << endl;
string::iterator it = msg.begin(); int count = 0;
while( it != msg.end() ) {
cout << *it;
it++;
cout << *it << " ";
it++;
if( ++count >= 26 )
cout << endl;
count = 0;
}
cout << endl << endl;
}
char getChar( int a, int b ) { //get the characters
return n[ (b + 5) % 5 ][ (a + 5) % 5 ];
}
bool getPos( char l, int &c, int &d ) { //get the position
for( int y = 0; y < 5; y++ )
for( int x = 0; x < 5; x++ )
if( n[y][x] == l ) {
c = x;
d = y;
return true;
}
return false;
}
void getText( string t, bool m, bool e ) { //get the original message

```

```

for( string::iterator it = t.begin(); it != t.end(); it++ ) {
//to choose J = I or no Q in the alphabet.
*it = toupper( *it );
if( *it < 65 || *it > 90 )
continue;
if( *it == 'J' && m )
*it = 'I';
else if( *it == 'Q' && !m )
continue;
msg += *it;
}
if( e ) {
string nmsg = ""; size_t len = msg.length();
for( size_t x = 0; x < len; x += 2 ) {
nmsg += msg[x];
if( x + 1 < len ) {
if( msg[x] == msg[x + 1] ) nmsg += 'X';
nmsg += msg[x + 1]; } }
msg = nmsg; }
if( msg.length() & 1 )
msg += 'X'; }
void createEncoder( string key, bool m ) { //creation of the key table
if( key.length() < 1 )
key= "KEYWORD";
key += "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
string s= "";
for( string::iterator it = key.begin(); it != key.end(); it++ ) {
*it = toupper( *it );
if( *it < 65 || *it > 90 )
continue;
if( ( *it == 'J' && m ) || ( *it == 'Q' && !m ) )
continue;

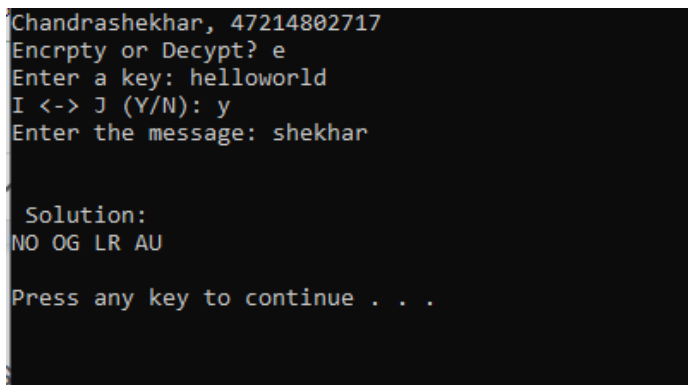
```

```

if( s.find( *it ) == -1 )
s += *it;
}
copy( s.begin(), s.end(), &n[0][0] );
}
};

int main( int argc, char* argv[] ) {
string k, i, msg;
bool m, c;
cout<<"Chandrashekhar, 47214802717"<<endl;
cout << "Encrypt or Decrypt? ";
getline( cin, i );
c = ( i[0] == 'e' || i[0] == 'E' );
cout << "Enter a key: ";
getline( cin, k);
cout << "I <-> J (Y/N): ";
getline( cin, i );
m = ( i[0] == 'y' || i[0] == 'Y' );
cout << "Enter the message: ";
getline( cin, msg );
playfair pf;
pf.play( k, msg,m, c );
return system( "pause" );
}

```



```

Chandrashekhar, 47214802717
Enchrpty or Decrypt? e
Enter a key: helloworld
I <-> J (Y/N): y
Enter the message: shekhar

Solution:
NO OG LR AU

Press any key to continue . . .

```

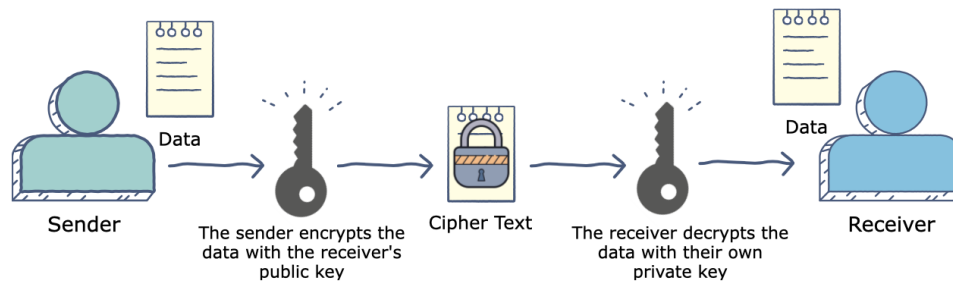
## Experiment No – 4

### Aim – Implement RSA Algorithm

RSA algorithm is a public key encryption technique and is considered as the most secure way of encryption. It was invented by Rivest, Shamir and Adleman in year 1978 and hence name **RSA** algorithm.

The RSA algorithm holds the following features –

- RSA algorithm is a popular exponentiation in a finite field over integers including prime numbers.
- The integers used by this method are sufficiently large making it difficult to solve.
- There are two sets of keys in this algorithm: private key and public key.



### How it works

The RSA algorithm ensures that the keys, in the above illustration, are as secure as possible. The following steps highlight how it works:

#### Generating the keys

- Select two large prime numbers,  $x$  and  $y$ . The prime numbers need to be large so that they will be difficult for someone to figure out.
- Calculate  $n = x * y$
- Calculate the **totient** function;  $\phi(n) = (x-1)(y-1)$
- Select an integer  $e$ , such that  $e$  is **co prime** to  $\phi(n)$  and  $1 < e < \phi(n)$ . The pair of numbers  $(n, e)$  makes up the public key.
- Calculate  $d$  such that  $e * d \equiv 1 \pmod{\phi(n)}$ .  $d$  can be found using the extended euclidean algorithm. The pair  $(n, d)$  makes up the private key.

### Encryption

$$C = P^e \pmod n$$

## Decryption

$$P = C^d \bmod n$$

## How secure is RSA?

*RSA private key is not 100% secure. But if the private key uses larger value of  $n = p \cdot q$ , it will take a very long time to crack the private key.*

The security of RSA public key encryption algorithm is mainly based on the integer factorization problem, which can be described as:

Given integer  $n$  as the product of 2 distinct prime numbers  $p$  and  $q$ , find  $p$  and  $q$ .

If the above problem could be solved, the RSA encryption is not secure at all. This is because the public key  $\{n, e\}$  is known to the public. Any one can use the public key  $\{n, e\}$  to figure out the private key  $\{n, d\}$  using these steps:

- Compute  $p$  and  $q$  by factorizing  $n$ .
- Compute  $m = (p-1) \cdot (q-1)$ .
- Compute  $d$  such that  $d \cdot e \bmod m = 1$  to obtain the private key  $\{n, d\}$

If  $n$  is small, the integer factorization problem is easy to solve by testing all possible prime numbers in the range of  $(1, n)$ .

For example, given 35 as  $n$ , we can list all prime numbers in the range of  $(1, 35)$ : 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, and 31, and try all combinations of them to find 5 and 7 are factors of 35.

As the value of  $n$  gets larger, the integer factorization problem gets harder to solve. But it is still solvable with the use of computers. For example, the RSA-100 number with 100 decimal digits, or 330 bits, has been factored by Arjen K. Lenstra in 1991:

```
RSA-100 = 15226050279225333605356183781326374297180681149613
          80688657908494580122963258952897654000350692006139

p*q = 37975227936943673922808872755445627854565536638199
     * 40094690950920881030683735292761468389214899724061
```

If you are using the above RSA-100 number as  $n$ , your private key is not private any more.

As of today, the highest value of  $n$  that has been factored is RSA-678 number with 232 decimal digits, or 768 bits, factored by Thorsten Kleinjung et al. in 2009:

```
RSA-768 = 12301866845301177551304949583849627207728535695953
          34792197322452151726400507263657518745202199786469
          38995647494277406384592519255732630345373154826850
          79170261221429134616704292143116022212404792747377
```

```
94080665351419597459856902143413
```

```
p*q = 33478071698956898786044169848212690817704794983713
      76856891243138898288379387800228761471165253174308
      7737814467999489
× 36746043666799590428244633799627952632279158164343
  08764267603228381573966651127923337341714339681027
  0092798736308917
```

As our computers are getting more powerful, factoring  $n$  of 1024 bits will soon become reality. This is why experts are recommending us:

1. Stop using RSA keys with  $n$  of 1024 bits now.
2. Use RSA keys with  $n$  of 2048 bits to keep your data safe up to year 2030.
3. Use RSA keys with  $n$  of 3072 bits to keep your data safe beyond year 2031.

## Code

```
from math import gcd
from random import choice, randint

#Function to Check Prime Number

def isPrime(n) :
    if (n <= 1) :
        return False
    if (n <= 3) :
        return True
    if (n % 2 == 0 or n % 3 == 0) :
        return False
    i = 5
    while(i * i <= n) :
        if (n % i == 0 or n % (i + 2) == 0) :
            return False
        i = i + 6
    return True
```

```
#Generation of Prime Numbers
```

```
#Range between which p and q lie
```

```
start = 100
```

```
stop = 999
```

```
p = randint(start,stop)
```

```
q = randint(start,stop)
```

```
#Generation of Random Prime P
```

```
while not isPrime(p):
```

```
    p = randint(start,stop)
```

```
#Generation of Random prime Q not same as P
```

```
while not isPrime(q) and q!=p:
```

```
    q = randint(start,stop)
```

```
#Generation of Phi
```

```
n = p*q
```

```
phi = (p-1)*(q-1)
```

```
#Generation of E (Public Key)
```

```
for e in range(2,phi):
```

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

```
        break
```

```
#Generate d
```

```
#Mod Inverse Function
```



```

def modInverse(a, m) :
    a = a % m;
    for x in range(1, m) :
        if ((a * x) % m == 1) :
            return x
    return 1

d = modInverse(e,phi)

#Original Message
message = 99

#Encrypted Message
enc_message = (message**e)%n

#Decrypted Message
dec_message = (enc_message**d)%n

#Output
print("Integer P: ",p)
print("Integer Q: ",q)
print("Integer n (p*q) :",n)
print("Phi (p-1)(q-1) :",phi)
print("E (Public Key): ",e)
print("D (Private Key: ",d)

print("Message ",message)
print("Encrypted Message: ",enc_message)
print("Decrypte Message: ",dec_message)

```

## Output

```
In [271]: print("Integer P: ",p)
          print("Integer Q: ",q)
          print("Integer n (p*q) :",n)
          print("Phi (p-1)(q-1) :",phi)
          print("E (Public Key): ",e)
          print("D (Private Key: ",d)

          print("Message ",message)
          print("Encrypted Message: ",enc_message)
          print("Decrypte Message: ",dec_message)
```

```
Integer P: 293
Integer Q: 401
Integer n (p*q) : 117493
Phi (p-1)(q-1) : 116800
E (Public Key): 168967
D (Private Key: 53303
Message 99
Encrypted Message: 49837
Decrypte Message: 99
```

## Experiment No – 5

**Aim** – Implement DES Algorithm

### Feistel Cipher

Feistel Cipher model is a structure or a design used to develop many block ciphers such as DES. Feistel cipher may have invertible, non-invertible and self-invertible components in its design. Same encryption as well as decryption algorithm is used. A separate key is used for each round. However same round keys are used for **Encryption Process**

### Feistel cipher algorithm

- Create a list of all the Plain Text characters.
- Convert the Plain Text to Ascii and then 8-bit binary format.
- Divide the binary Plain Text string into two halves: left half (L1) and right half (R1)
- Generate a random binary keys (K1 and K2) of length equal to the half the length of the Plain Text for the two rounds.
- First Round of Encryption

a. Generate function f1 using R1 and K1 as follows:

$$f1 = \text{xor}(R1, K1)$$

b. Now the new left half(L2) and right half(R2) after round 1 are as follows:

$$R2 = \text{xor}(f1, L1)$$

$$L2 = R1$$

- Second Round of Encryption

a. Generate function f2 using R2 and K2 as follows:

$$f2 = \text{xor}(R2, K2)$$

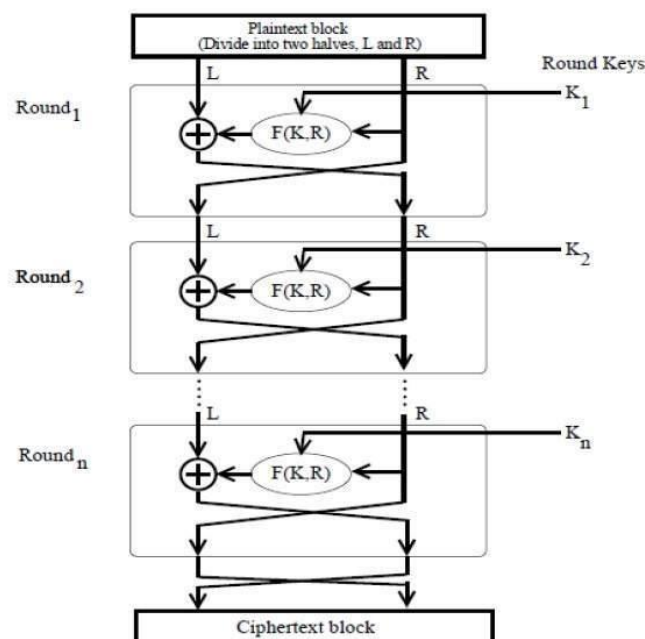
b. Now the new left half(L2) and right half(R2) after round 1 are as follows:

$$R3 = \text{xor}(f2, L2)$$

$$L_3 = R_2$$

- Concatenation of  $R_3$  to  $L_3$  is the Cipher Text
- Same algorithm is used for decryption to retrieve the Plain Text from the Cipher Text.

## Encryption Process



The encryption process uses the Feistel structure consisting multiple rounds of processing of the plaintext, each round consisting of a “substitution” step followed by a permutation step.

Feistel Structure is shown in the following illustration –

The input block to each round is divided into two halves that can be denoted as L and R for the left half and the right half.

The permutation step at the end of each round swaps the modified L and unmodified R. Therefore, the L for the next round would be R of the current round. And R for the next round be the output L of the current round.

- Above substitution and permutation steps form a ‘round’. The number of rounds are specified by the algorithm design.
- Once the last round is completed then the two sub blocks, ‘R’ and ‘L’ are concatenated in this order to form the ciphertext block.

The difficult part of designing a Feistel Cipher is selection of round function 'f'. In order to be unbreakable scheme, this function needs to have several important properties that are beyond the scope of our discussion.

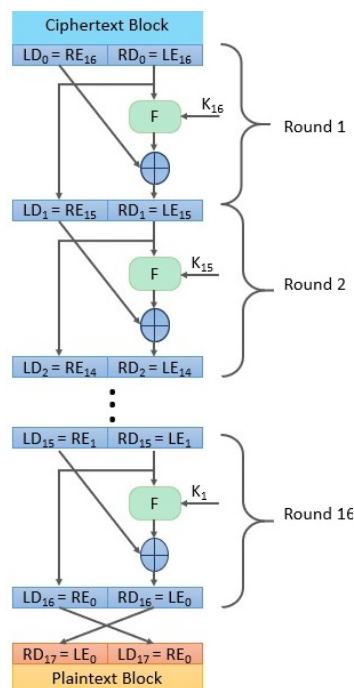
## Decryption Process

The process of decryption in Feistel cipher is almost similar. Instead of starting with a block of plaintext, the ciphertext block is fed into the start of the Feistel structure and then the process thereafter is exactly the same as described in the given illustration.

The process is said to be almost similar and not exactly same. In the case of decryption, the only difference is that the subkeys used in encryption are used in the reverse order.

The final swapping of 'L' and 'R' in last step of the Feistel Cipher is essential. If these are not swapped then the resulting ciphertext could not be decrypted using the same algorithm.

**Number of Rounds:** The number of rounds used in a Feistel Cipher depends on desired security from the system. More number of rounds provide more secure system. But at the same time, more rounds mean the inefficient slow encryption and decryption processes. Number of rounds in the systems thus depend upon efficiency–security tradeoff.

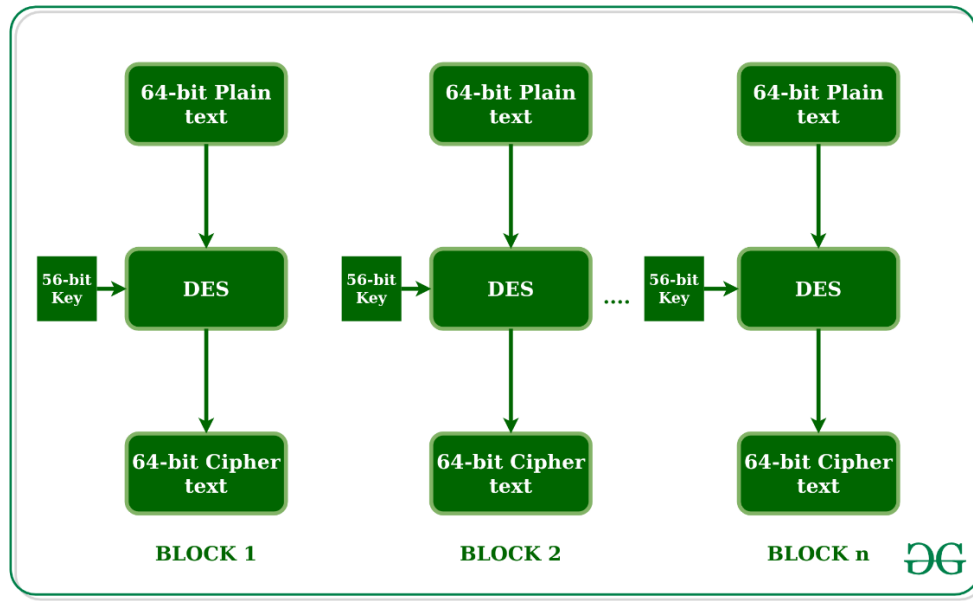


## Data Encryption Standard (DES)

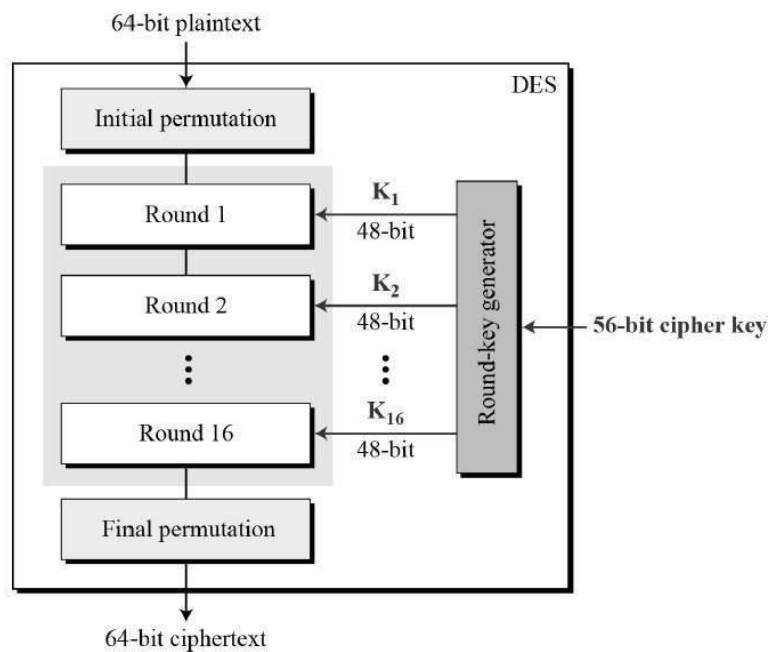
The Data Encryption Standard (DES) is a [symmetric-key algorithm](#) for the [encryption](#) of digital data. Although its short key length of 56 bits makes it too insecure for applications, it has been highly influential in the advancement of [cryptography](#).

**Data encryption standard (DES)** has been found vulnerable against very powerful attacks and therefore, the popularity of DES has been found slightly on decline.

DES is a block cipher and encrypts data in blocks of size of 64 bit each, means 64 bits of plain text goes as the input to DES, which produces 64 bits of cipher text. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is 56 bits. The basic idea is show in figure.



General Structure of DES is depicted in the following illustration –

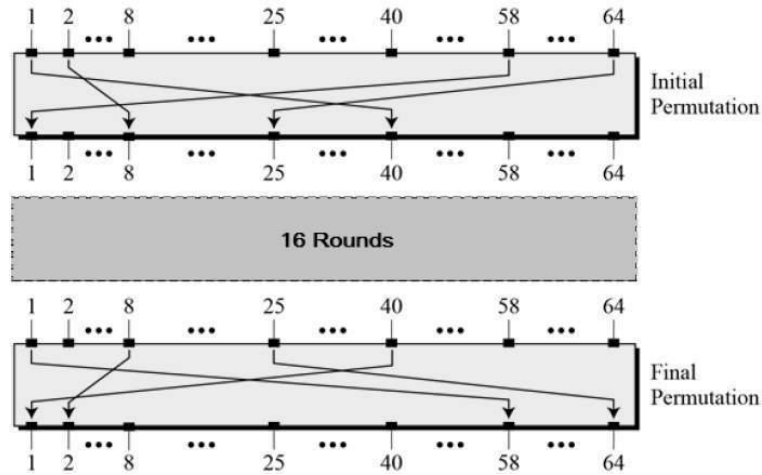


Since DES is based on the Feistel Cipher, all that is required to specify DES is –

- Round function
- Key schedule
- Any additional processing – Initial and final permutation

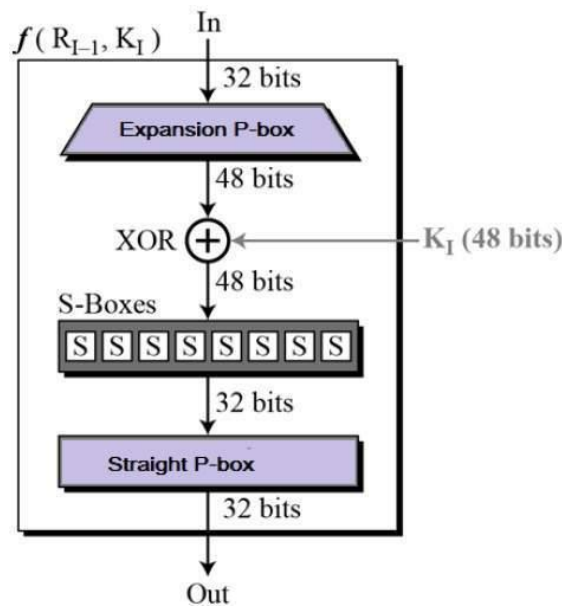
#### Initial and Final Permutation

The Initial permutation (IP) happens only once and it happens before the first round. It suggests how the transposition in IP should proceed. The initial and final permutations are straight Permutation boxes (P-boxes) that are inverses of each other. They have no cryptography significance in DES. The initial and final permutations are shown as follows –

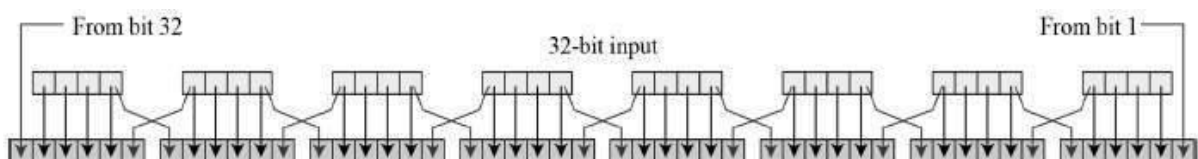


## Round Function

The heart of this cipher is the DES function,  $f$ . The DES function applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output.



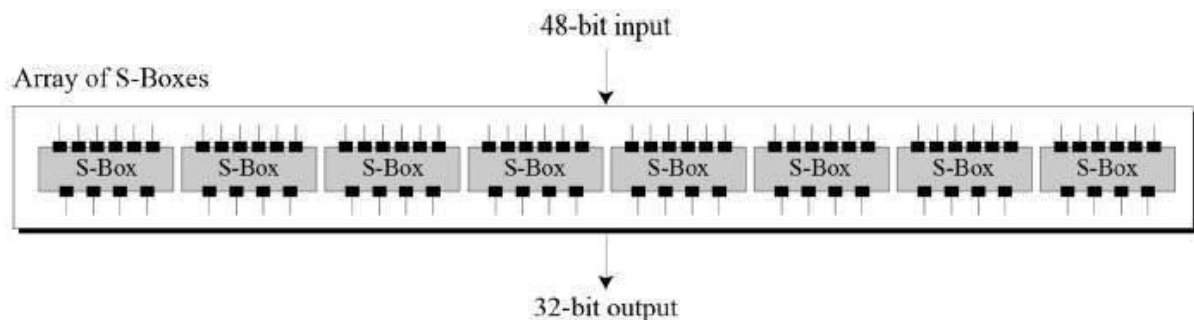
- **Expansion Permutation Box** – Since right input is 32-bit and round key is a 48-bit, we first need to expand right input to 48 bits. Permutation logic is graphically depicted in the following illustration –



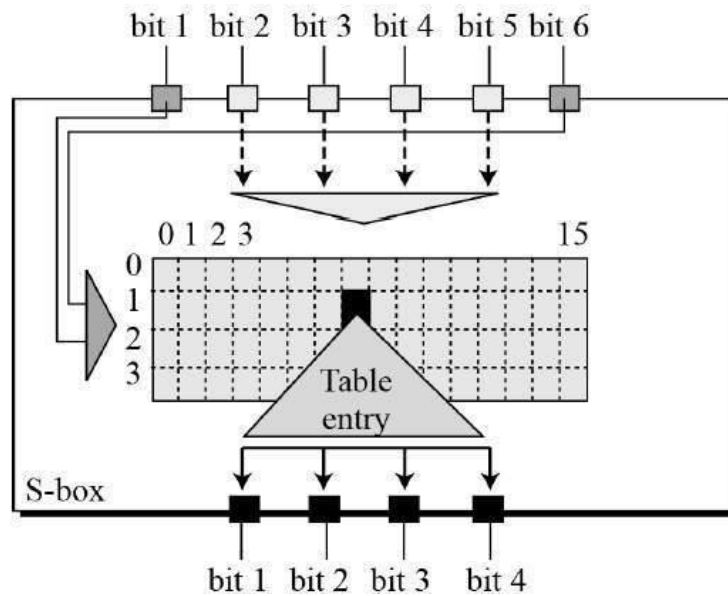
- The graphically depicted permutation logic is generally described as table in DES specification illustrated as shown –

32	01	02	03	04	05
04	05	06	07	08	09
08	09	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	31	31	32	01

- **XOR (Whitener).** – After the expansion permutation, DES does XOR operation on the expanded right section and the round key. The round key is used only in this operation.
- **Substitution Boxes.** – The S-boxes carry out the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output. Refer the following illustration –



- The S-box rule is illustrated below –



- There are a total of eight S-box tables. The output of all eight s-boxes is then combined in to 32 bit section.
- **Straight Permutation** – The 32 bit output of S-boxes is then subjected to the straight permutation with rule shown in the following illustration:

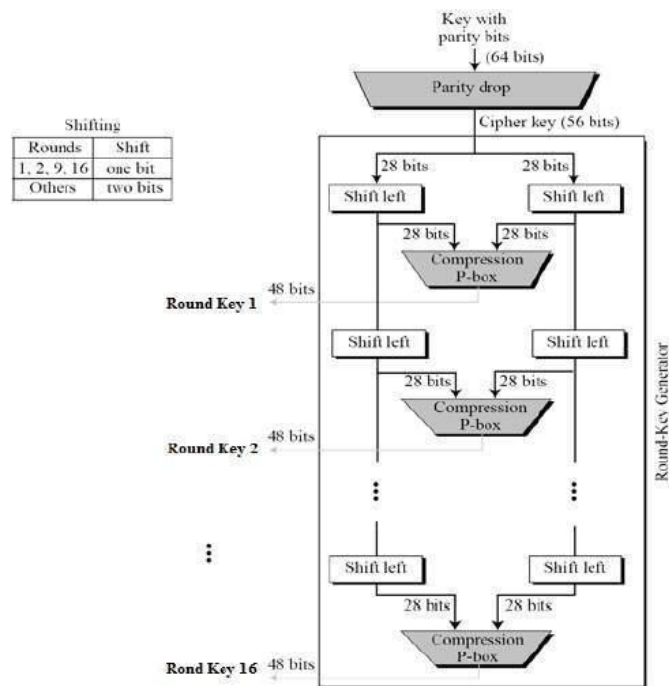
16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

### Key Generation

The round-key generator creates sixteen 48-bit keys out of a 56-bit cipher key. The process of key generation is depicted in the following illustration

—





The logic for Parity drop, shifting, and Compression P-box is given in the DES description.

## DES Analysis

The DES satisfies both the desired properties of block cipher. These two properties make cipher very strong.

- **Avalanche effect** – A small change in plaintext results in the very great change in the ciphertext.
- **Completeness** – Each bit of ciphertext depends on many bits of plaintext.

During the last few years, cryptanalysis have found some weaknesses in DES when key selected are weak keys. These keys shall be avoided.

DES has proved to be a very well designed block cipher. There have been no significant cryptanalytic attacks on DES other than exhaustive key search.

## Code

```
def hex2bin(s):
    mp = {'0': "0000",
          '1': "0001",
          '2': "0010",
          '3': "0011",
          '4': "0100",
```

```
'5' : "0101",
'6' : "0110",
'7' : "0111",
'8' : "1000",
'9' : "1001",
'A' : "1010",
'B' : "1011",
'C' : "1100",
'D' : "1101",
'E' : "1110",
'F' : "1111" }

bin = ""

for i in range(len(s)):
    bin = bin + mp[s[i]]

return bin
```

```
def bin2hex(s):
    mp = {"0000" : '0',
          "0001" : '1',
          "0010" : '2',
          "0011" : '3',
          "0100" : '4',
          "0101" : '5',
          "0110" : '6',
          "0111" : '7',
          "1000" : '8',
          "1001" : '9',
          "1010" : 'A',
          "1011" : 'B',
          "1100" : 'C',
```

```

    "1101" : 'D',
    "1110" : 'E',
    "1111" : 'F' }
hex = ""
for i in range(0,len(s),4):
    ch = ""
    ch = ch + s[i]
    ch = ch + s[i + 1]
    ch = ch + s[i + 2]
    ch = ch + s[i + 3]
    hex = hex + mp[ch]

return hex

```

```

def bin2dec(binary):

```

```

    binary1 = binary
    decimal, i, n = 0, 0, 0
    while(binary != 0):
        dec = binary % 10
        decimal = decimal + dec * pow(2, i)
        binary = binary//10
        i += 1
    return decimal

```

```

def dec2bin(num):

```

```

    res = bin(num).replace("0b", "")
    if(len(res)%4 != 0):
        div = len(res) / 4
        div = int(div)

```

```
    counter =(4 * (div + 1)) - len(res)
    for i in range(0, counter):
        res = '0' + res
    return res
```

```
def permute(k, arr, n):
    permutation = ""
    for i in range(0, n):
        permutation = permutation + k[arr[i] - 1]
    return permutation
```

```
def shift_left(k, nth_shifts):
    s = ""
    for i in range(nth_shifts):
        for j in range(1,len(k)):
            s = s + k[j]
        s = s + k[0]
        k = s
        s = ""
    return k
```

```
def xor(a, b):
    ans = ""
    for i in range(len(a)):
        if a[i] == b[i]:
            ans = ans + "0"
        else:
            ans = ans + "1"
    return ans
```

```
initial_perm = [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]
```

```
exp_d = [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 ]
```

```
per = [ 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 ]
```

```
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] ] ]
```

```
final_perm = [ 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 ]
```

```
def encrypt(pt, rkb, rk):
```

```
    pt = hex2bin(pt)
```

```
    pt = permute(pt, initial_perm, 64)
```

```
    print("After initial permutation", bin2hex(pt))
```

```
    left = pt[0:32]
```

```
    right = pt[32:64]
```

```
    for i in range(0, 16):
```

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

```
        right_expanded = permute(right, exp_d, 48)
```

```
        # XOR RoundKey[i] and right_expanded
```

```
        xor_x = xor(right_expanded, rkb[i])
```

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

```

sbox_str = ""
for j in range(0, 8):
    row = bin2dec(int(xor_x[j * 6] + xor_x[j * 6 + 5]))
    col = bin2dec(int(xor_x[j * 6 + 1] + xor_x[j * 6 + 2] + xor_x[j * 6 + 3] + xor_x[j * 6 + 4]))
    val = sbox[j][row][col]
    sbox_str = sbox_str + dec2bin(val)

# Straight D-box: After substituting rearranging the bits
sbox_str = permute(sbox_str, per, 32)

# XOR left and sbox_str
result = xor(left, sbox_str)
left = result

# Swapper
if(i != 15):
    left, right = right, left
print("Round ", i + 1, " ", bin2hex(left), " ", bin2hex(right), " ", rk[i])

# Combination
combine = left + right

# Final permutaion: final rearranging of bits to get cipher text
cipher_text = permute(combine, final_perm, 64)
return cipher_text

pt = "40214807218ABCD6"
key = "AABB09182736CCDD"

# Key generation

```



```
# --hex to binary
```

```
key = hex2bin(key)
```

```
# --parity bit drop table
```

```
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 ]
```

```
# getting 56 bit key from 64 bit using the parity bits
```

```
key = permute(key, keyp, 56)
```

```
# Number of bit shifts
```

```
shift_table = [1, 1, 2, 2,
```

```
2, 2, 2, 2,
```

```
1, 2, 2, 2,
```

```
2, 2, 2, 1 ]
```

```
# Key- Compression Table : Compression of key from 56 bits to 48 bits
```

```
key_comp = [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 ]

# Splitting

left = key[0:28] # rkb for RoundKeys in binary

right = key[28:56] # rk for RoundKeys in hexadecimal

rkb = []

rk = []

for i in range(0, 16):

    # Shifting the bits by nth shifts by checking from shift table

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

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

    # Combination of left and right string

    combine\_str = left + right

    # Compression of key from 56 to 48 bits

    round\_key = permute(combine\_str, key\_comp, 48)

    rkb.append(round\_key)

    rk.append(bin2hex(round\_key))

print("Encryption")

cipher\_text = bin2hex(encrypt(pt, rkb, rk))

print("Cipher Text : ", cipher\_text)

print("Decryption")

rkb\_rev = rkb[::-1]

rk\_rev = rk[::-1]

text = bin2hex(encrypt(cipher\_text, rkb\_rev, rk\_rev))

```
print("Plain Text : ",text)
```

## OUTPUT

### Encryption

After initial permutation 85C0C81AE05264A8

Round 1	E05264A8	5A30E514	194CD072DE8C
Round 2	5A30E514	A0CA78F3	4568581ABCCE
Round 3	A0CA78F3	B441FB5B	06EDA4ACF5B5
Round 4	B441FB5B	7201A7FF	DA2D032B6EE3
Round 5	7201A7FF	5ABD9E8C	69A629FEC913
Round 6	5ABD9E8C	D22DD7C6	C1948E87475E
Round 7	D22DD7C6	79E3C2B7	708AD2DDB3C0
Round 8	79E3C2B7	9A0BFBE9	34F822F0C66D
Round 9	9A0BFBE9	FE1990B2	84BB4473DCCC
Round 10	FE1990B2	2B96BB29	02765708B5BF
Round 11	2B96BB29	EE29D0B3	6D5560AF7CA5
Round 12	EE29D0B3	2B407732	C2C1E96A4BF3
Round 13	2B407732	3DE42C1A	99C31397C91F
Round 14	3DE42C1A	3ED55242	251B8BC717D0
Round 15	3ED55242	3EC12BE3	3330C5D9A36D
Round 16	E442D08D	3EC12BE3	181C5D75C66D

Cipher Text : 2B9AC18984CA7667

### Decryption

After initial permutation E442D08D3EC12BE3

Round 1	3EC12BE3	3ED55242	181C5D75C66D
Round 2	3ED55242	3DE42C1A	3330C5D9A36D
Round 3	3DE42C1A	2B407732	251B8BC717D0
Round 4	2B407732	EE29D0B3	99C31397C91F
Round 5	EE29D0B3	2B96BB29	C2C1E96A4BF3
Round 6	2B96BB29	FE1990B2	6D5560AF7CA5
Round 7	FE1990B2	9A0BFBE9	02765708B5BF
Round 8	9A0BFBE9	79E3C2B7	84BB4473DCCC
Round 9	79E3C2B7	D22DD7C6	34F822F0C66D
Round 10	D22DD7C6	5ABD9E8C	708AD2DDB3C0
Round 11	5ABD9E8C	7201A7FF	C1948E87475E
Round 12	7201A7FF	B441FB5B	69A629FEC913
Round 13	B441FB5B	A0CA78F3	DA2D032B6EE3
Round 14	A0CA78F3	5A30E514	06EDA4ACF5B5
Round 15	5A30E514	E05264A8	4568581ABCCE
Round 16	85C0C81A	E05264A8	194CD072DE8C

Plain Text : 40214807218ABCD6

## EXPERIMENT – 6

**Aim:** To implement Diffie-Hellman Algorithm

**Source Code:**

```
#include<stdio.h>

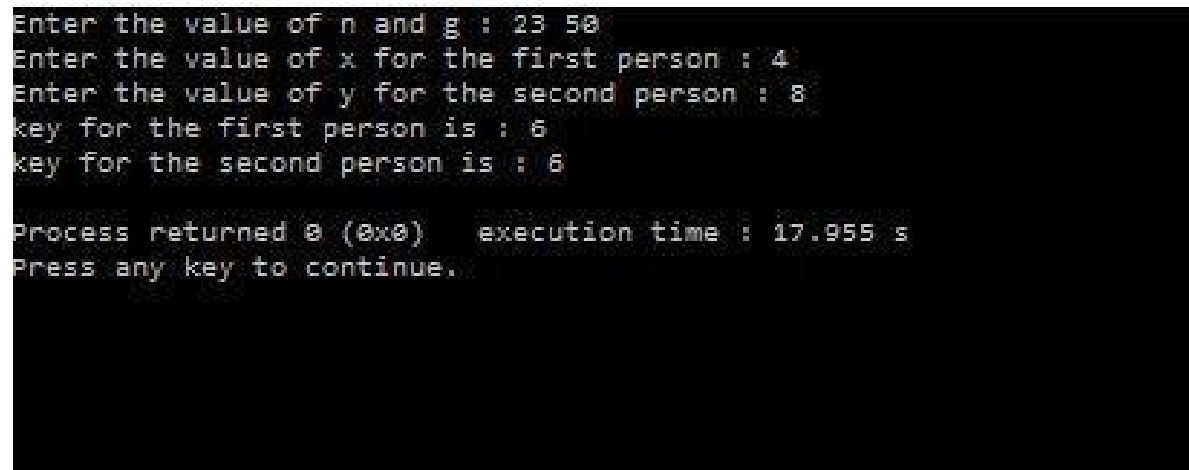
long int power(int a,int b,int mod)
{
    long long int t;
    if(b==1)
        return a;
    t=power(a,b/2,mod);
    if(b%2==0)
        return (t*t)%mod;
    else
        return (((t*t)%mod)*a)%mod;
}

long long int calculateKey(int a,int x,int n)
{
    return power(a,x,n);
}

int main()
{
    int n,g,x,a,y,b;
    // both the persons will be agreed upon the common n and g
    printf("Enter the value of n and g : ");
    scanf("%d%d",&n,&g);
    // first person will choose the x
    printf("Enter the value of x for the first person : ");
    scanf("%d",&x); a=power(g,x,n);
```

```
// second person will choose the y
printf("Enter the value of y for the second person : ");
scanf("%d",&y); b=power(g,y,n);
printf("key for the first person is : %lld\n",power(b,x,n));
printf("key for the second person is : %lld\n",power(a,y,n));
return 0;
}
```

### Output:



```
Enter the value of n and g : 23 50
Enter the value of x for the first person : 4
Enter the value of y for the second person : 8
key for the first person is : 6
key for the second person is : 6

Process returned 0 (0x0)   execution time : 17.955 s
Press any key to continue.
```

## EXPERIMENT – 7

**Aim:** Study of NeSSi<sup>2</sup> Simulation Tool based on parameters of IS.

### Theory:

NeSSi consists of three distinct components, the **Graphical User Interface**, the **Simulation Backend** and the **Simulation Result Database**. Each of these modules may be run on separate machines depending on the computational requirements; furthermore, this modular design facilitates network security researchers using NeSSi to easily exchange

### **Graphical User Interface**

The graphical frontend of NeSSi allows the user to create and edit network topologies, attach runtime information, and schedule them for execution at the simulation backend. On the other hand, finished (or even currently executing, long-running) simulations can be retrieved from the database server and the corresponding simulation results are visualized in the GUI.

### **Simulation Backend**

The actual simulation is performed on machine with hardware dedicated solely to this purpose, the simulation backend. At the DAI-Labor for example, the NeSSi simulation backend runs on a Sun XFire 4600 blade server (8 blades, 8 cores per blade). Once a session is submitted for execution from the GUI, the simulation backend parses the desired session parameters (which event types to log, how many runs to execute etc.), creates a corresponding simulation environment, sets up the database connection and schedules the simulation to run as soon as the necessary processing resources are available.

4. **Parallel Execution Model.** Simulations in large-scale networks are very costly in terms of processing time and memory consumption. Therefore, NeSSi has been designed as a distributed simulation, allowing the subdivision of tasks to different computers and processes in a parallel-execution model.
5. **Discrete Event Simulation.** NeSSi<sup>2</sup> is a discrete-event-based simulation tool, which allows to plan and to schedule time-based events such as network failures, attacks, etc.

### **Simulation Result Database Server**

In NeSSi, we refer to a scenario where we generate traffic via pre-defined profiles on a single network over a certain amount of time as a *session*. The accurate reproduction of a session enables users to use different detection methods or various deployments of detection units for the same traffic data set. This allows the comparison of performance and detection efficiency of different security framework setups.

For these purposes, we use a **distributed database** in which the traffic generated during a session is stored. For each session, the agents log the traffic and detection data and send it to the database that occurs in a simulated scenario between a start and end time. The data types to be logged are specified by the user in the session parameters. The network model is saved in an XML file. This network file is stored and annotated with a version number based on its hash code in order to link a network uniquely to a session. Additionally, attack related events can be stored in the database for evaluation purposes.

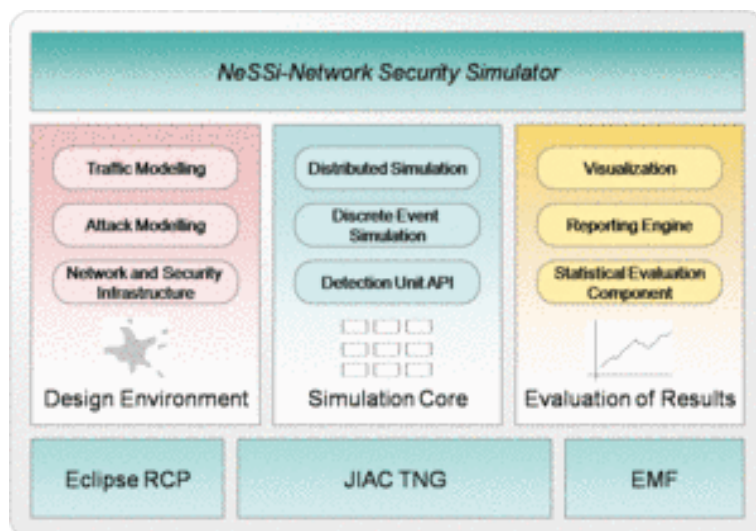
### ***Standard Components and Plugin API***

NeSSi<sup>2</sup> has been designed as a modularized application. Building on the Eclipse framework, it uses the inherent plugin mechanism to allow users to easily extend the functionality of NeSSi<sup>2</sup> and share it with other developers.

Often, security researchers have very specific demands regarding the protocols and features the simulation tool needs to offer. Naturally, NeSSi<sup>2</sup> provides a rich set of basic protocols and detection unit implementations; nevertheless, the special needs of various application areas (wireless networks, sensor networks, MPLS etc.) necessitates a plugin API to allow the user to adapt NeSSi<sup>2</sup> to his needs and add the functionality that is not provided by NeSSi out-of-the-box.

Hence, the NeSSi<sup>2</sup> extension API allows the creation of

- o New device types with user-defined properties
- o New protocols defining the behavior of the network at runtime
- o Application definitions, allowing dynamic behavior to be defined, attached to a device or link, and scheduled for execution in the simulation



## EXPERIMENT – 8

**Aim:** To implement VPN through Network Simulator Tool.

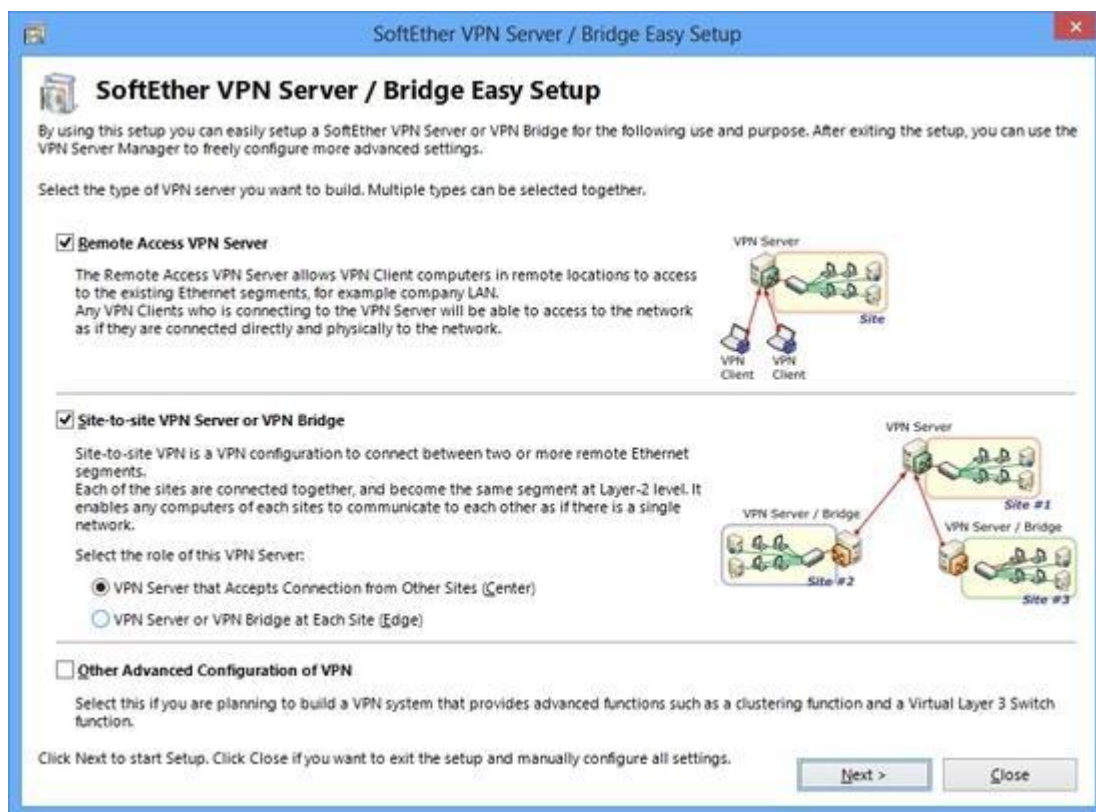
### Steps:

SoftEther VPN can construct distributed virtual Ethernet segment. If you can make some geologically distributed computers enable to communicate each other as if they are connected to the single Ethernet network, using SoftEther VPN is the easiest way.

First, set up a VPN Server. Next, set up VPN Clients on each member PCs. Finally start VPN connections on each VPN client. Then each client can use any kinds of IP-based or Ethernet-based protocols via the VPN even if they are distributed around the world.

### Step 1 – Set up SoftEther VPN Server

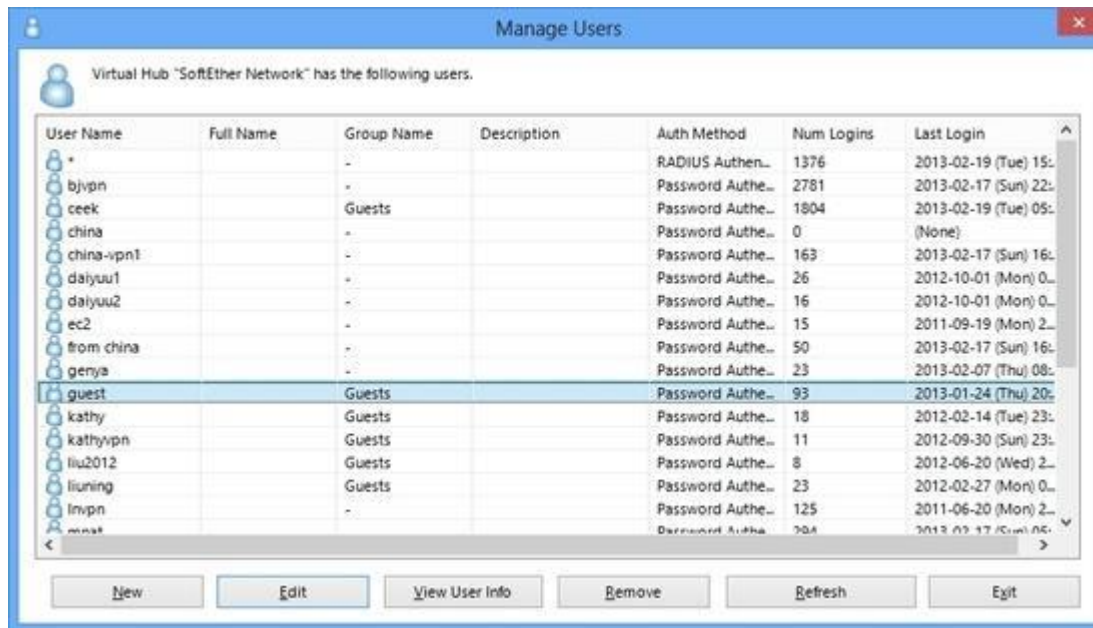
Designate a computer in the group as the VPN Server. [Set up SoftEther VPN Server](#) on that computer. It is very easy by using Installer and Initial Setup Wizard based GUI.



### Step 2 – Create Users

On the VPN Server [you can add several user objects](#) on the [Virtual Hub](#). Each user object has a password. After that, distribute pairs of username and password to each member of the VPN.

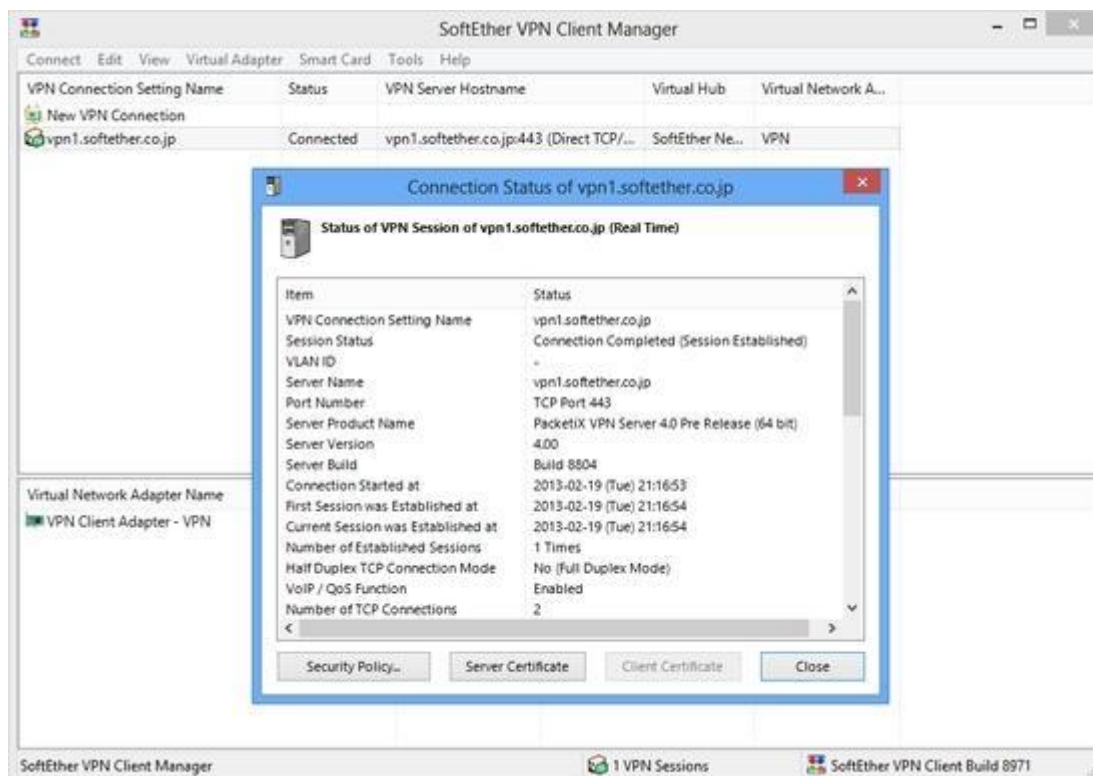




### Step 3 – Set up VPN Client on Each Member's PC

On each member's PC install SoftEther VPN Client. Enter the server address, username and password for each PC.

If a member of the VPN is Mac OS X, iPhone or Android, set up L2TP/IPsec VPN client on each PC instead of SoftEther VPN. Another solution is to use OpenVPN Client on Mac OS X, iPhone or Android to connect to SoftEther VPN Server.

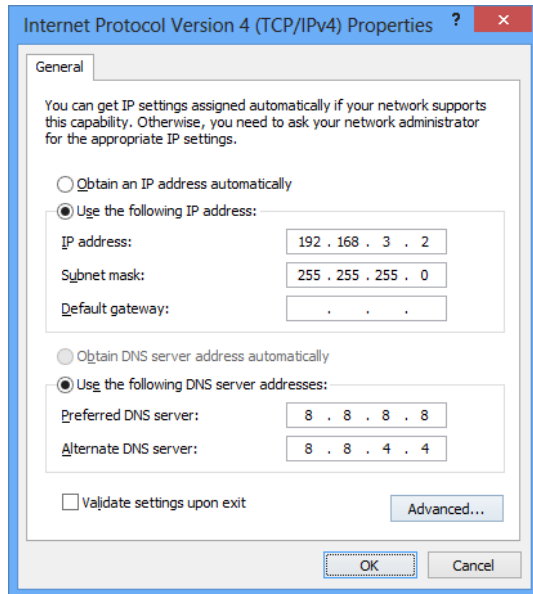


#### ***Step 4 – Set up IP Addresses***

The characteristics of SoftEther's virtual private network is exactly same to a physical Ethernet segment. So you should decide the IP addresses of every member PCs.

Like the physical Ethernet, the simplest way is to set up private IP addresses to each PC, for example 192.168.0.0/24. Make sure not to overlap to physical-using private IPs.

Another solution is to use DHCP server for automated IP address allocation. You can activate Virtual DHCP Server Function on the SoftEther VPN Server and it will distribute 192.168.30.0/24 by default.



#### ***Step 5 – Communicate Like Physical Ethernet***

Once every computer is connected to the Virtual Hub on SoftEther VPN Server, all computers and smart-phones can now communicate mutually as if they are all connected to the single Ethernet network. You can enjoy File Sharing protocols, Remote Printing applications, Remote Desktop applications, SQL Database applications and any other LAN-based applications despite the distances and differences of physical location.

