**EXPERIMENT-1**

Write a program to implement Caesar Cipher.

def cc\_enc(mess, ky):

c = ""

for mi in mess:

if mi.isupper():

ascii\_val = ord(mi)

ascii\_val = ascii\_val + k

ascii\_val = ascii\_val - 65

ascii\_val = ascii\_val % 26

ascii\_val = ascii\_val + 65

c = c + chr(ascii\_val)

elif mi.islower():

ascii\_val = ord(mi)

ascii\_val = ascii\_val + k

ascii\_val = ascii\_val - 97

ascii\_val = ascii\_val % 26

ascii\_val = ascii\_val + 97

c = c + chr(ascii\_val)

else:

c = c + mi

print("Cipher-Text is : ", c)

def cc\_dec(mess, ky):

p = ""

for mi in mess:

if mi.isupper():

ascii\_val = ord(mi)

ascii\_val = ascii\_val - k

if ascii\_val < 65:

ascii\_val = ascii\_val + 26

p = p + chr(ascii\_val)

elif mi.islower():

ascii\_val = ord(mi)

ascii\_val = ascii\_val - k

if ascii\_val < 97:

ascii\_val = ascii\_val + 26

p = p + chr(ascii\_val)

else:

p = p + mi

print("Plain-Text is : ", p)

print("Specify the Message : ")

m = input()

print("Specify the Key : ")

k = int(input())

print("1 for Encryption and 2 for Decryption")

print("Select Operation : ")

ch = int(input())

if ch == 1:

cc\_enc(m, k)

elif ch == 2:

cc\_dec(m, k)

else:

print("Invalid Operation !")

**OUTPUT:**

Specify the Message :

Mjqqt Btwqi

Specify the Key :

5

1 for Encryption and 2 for Decryption

Select Operation :

2

Plain-Text is : Hello World

**EXPERIMENT-2**

Write a program of Data Encryption Standard (DES) algorithm for data encrytion

import java.security.InvalidKeyException; import java.security.NoSuchAlgorithmException;

import javax.crypto.BadPaddingException; import javax.crypto.Cipher;

import javax.crypto.IllegalBlockSizeException; import javax.crypto.KeyGenerator;

import javax.crypto.NoSuchPaddingException; import javax.crypto.SecretKey;

public class DES

{

public static void main(String[] argv) {

try{

System.out.println("Message Encryption Using DES Algorithm\n");

KeyGenerator keygenerator = KeyGenerator.getInstance("DES"); SecretKey myDesKey = keygenerator.generateKey();

Cipher desCipher;

desCipher = Cipher.getInstance("DES/ECB/PKCS5Padding"); desCipher.init(Cipher.ENCRYPT\_MODE, myDesKey); byte[] text = "Secret Information ".getBytes(); System.out.println("Message [Byte Format] : " + text); System.out.println("Message : " + new String(text));

byte[] textEncrypted = desCipher.doFinal(text); System.out.println("Encrypted Message: " + textEncrypted); desCipher.init(Cipher.DECRYPT\_MODE, myDesKey); byte[] textDecrypted = desCipher.doFinal(textEncrypted);

System.out.println("Decrypted Message: " + new

String(textDecrypted));

}catch(NoSuchAlgorithmException e){ e.printStackTrace();

}catch(NoSuchPaddingException e){ e.printStackTrace();

}catch(InvalidKeyException e){ e.printStackTrace();

}

catch(IllegalBlockSizeException e)

{ e.printStackTrace();

}

catch(BadPaddingException e)

{ e.printStackTrace();

}

}

}

# OUTPUT:

Message Encryption Using DES Algorithm

Message [Byte Format] : [B@4dcbadb4 Message : Secret Information Encrypted Message: [B@504bae78 Decrypted Message: Secret Information

**EXPERIMENT-3**

Write a program to implement Playfair Cipher

**import** java.awt.Point;

**import** java.util.Scanner;

**public** **class** PlayfairCipher

{

//length of digraph array

**private** **int** length = 0;

//creates a matrix for Playfair cipher

**private** String [ ]  table;

//main() method to test Playfair method

**public** **static** **void** main(String args[] )

{

PlayfairCipher pf = **new** PlayfairCipher();

}

//main run of the program, Playfair method

//constructor of the class

**private** PlayfairCipher()

{

//prompts user for the keyword to use for encoding & creates tables

System.out.print("Enter the key for playfair cipher: ");

Scanner sc = **new** Scanner(System.in);

String key = parseString(sc);

**while**(key.equals(""))

key = parseString(sc);

table = **this**.cipherTable(key);

//prompts user for message to be encoded

System.out.print("Enter the plaintext to be encipher: ");

//System.out.println("using the previously given keyword");

String input = parseString(sc);

**while**(input.equals(""))

input = parseString(sc);

//encodes and then decodes the encoded message

String output = cipher(input);

String decodedOutput = decode(output);

//output the results to user

**this**.keyTable(table);

**this**.printResults(output,decodedOutput);

}

//parses an input string to remove numbers, punctuation,

//replaces any J's with I's and makes string all caps

**private** String parseString(Scanner sc)

{

String parse = sc.nextLine();

//converts all the letters in upper case

parse = parse.toUpperCase();

//the string to be substituted by space for each match (A to Z)

parse = parse.replaceAll("[^A-Z]", "");

//replace the letter J by I

parse = parse.replace("J", "I");

**return** parse;

}

//creates the cipher table based on some input string (already parsed)

**private** String cipherTable(String key)

{

//creates a matrix of 5\*5

String playfairTable = **new** String[5][5];

String keyString = key + "ABCDEFGHIKLMNOPQRSTUVWXYZ";

//fill string array with empty string

**for**(**int** i = 0; i < 5; i++)

**for**(**int** j = 0; j < 5; j++)

playfairTable[i][j] = "";

**for**(**int** k = 0; k < keyString.length(); k++)

{

**boolean** repeat = **false**;

**boolean** used = **false**;

**for**(**int** i = 0; i < 5; i++)

{

**for**(**int** j = 0; j < 5; j++)

{

**if**(playfairTable[i][j].equals("" + keyString.charAt(k)))

{

repeat = **true**;

}

**else** **if**(playfairTable[i][j].equals("") && !repeat && !used)

{

playfairTable[i][j] = "" + keyString.charAt(k);

used = **true**;

}

}

}

}

**return** playfairTable;

}

//cipher: takes input (all upper-case), encodes it, and returns the output

**private** String cipher(String in)

{

length = (**int**) in.length() / 2 + in.length() % 2;

//insert x between double-letter digraphs & redefines "length"

**for**(**int** i = 0; i < (length - 1); i++)

{

**if**(in.charAt(2 \* i) == in.charAt(2 \* i + 1))

{

in = **new** StringBuffer(in).insert(2 \* i + 1, 'X').toString();

length = (**int**) in.length() / 2 + in.length() % 2;

}

}

//------------makes plaintext of even length--------------

//creates an array of digraphs

String[] digraph = **new** String[length];

//loop iterates over the plaintext

**for**(**int** j = 0; j < length ; j++)

{

//checks the plaintext is of even length or not

**if**(j == (length - 1) && in.length() / 2 == (length - 1))

//if not addends X at the end of the plaintext

in = in + "X";

digraph[j] = in.charAt(2 \* j) +""+ in.charAt(2 \* j + 1);

}

//encodes the digraphs and returns the output

String out = "";

String[] encDigraphs = **new** String[length];

encDigraphs = encodeDigraph(digraph);

**for**(**int** k = 0; k < length; k++)

out = out + encDigraphs[k];

**return** out;

}

//---------------encryption logic-----------------

//encodes the digraph input with the cipher's specifications

**private** String[] encodeDigraph(String di[])

{

String[] encipher = **new** String[length];

**for**(**int** i = 0; i < length; i++)

{

**char** a = di[i].charAt(0);

**char** b = di[i].charAt(1);

**int** r1 = (**int**) getPoint(a).getX();

**int** r2 = (**int**) getPoint(b).getX();

**int** c1 = (**int**) getPoint(a).getY();

**int** c2 = (**int**) getPoint(b).getY();

//executes if the letters of digraph appear in the same row

//in such case shift columns to right

**if**(r1 == r2)

{

c1 = (c1 + 1) % 5;

c2 = (c2 + 1) % 5;

}

//executes if the letters of digraph appear in the same column

//in such case shift rows down

**else** **if**(c1 == c2)

{

r1 = (r1 + 1) % 5;

r2 = (r2 + 1) % 5;

}

//executes if the letters of digraph appear in the different row and different column

//in such case swap the first column with the second column

**else**

{

**int** temp = c1;

c1 = c2;

c2 = temp;

}

/performs the table look-up and puts those values into the encoded array

encipher[i] = table[r1][c1] + "" + table[r2][c2];

}

**return** encipher;

}

//-----------------------decryption logic---------------------

// decodes the output given from the cipher and decode methods (opp. of encoding process)

**private** String decode(String out)

{

String decoded = "";

**for**(**int** i = 0; i < out.length() / 2; i++)

{

**char** a = out.charAt(2\*i);

**char** b = out.charAt(2\*i+1);

**int** r1 = (**int**) getPoint(a).getX();

**int** r2 = (**int**) getPoint(b).getX();

**int** c1 = (**int**) getPoint(a).getY();

**int** c2 = (**int**) getPoint(b).getY();

**if**(r1 == r2)

{

c1 = (c1 + 4) % 5;

 = (c2 + 4) % 5;

}

**else** **if**(c1 == c2)

r1 = (r1 + 4) % 5;

r2 = (r2 + 4) % 5;

**else**

{

//swapping logic

**int** temp = c1;

c1 = c2;

c2 = temp;

}

decoded = decoded + table[r1][c1] + table[r2][c2];

}

//returns the decoded message

**return** decoded;

}

// returns a point containing the row and column of the letter

**private** Point getPoint(**char** c)

{

Point pt = **new** Point(0,0);

**for**(**int** i = 0; i < 5; i++)

**for**(**int** j = 0; j < 5; j++)

**if**(c == table[i][j].charAt(0))

pt = **new** Point(i,j);

**return** pt;

}

//function prints the key-table in matrix form for playfair cipher

**private** **void** keyTable(String[][] printTable)

{

System.out.println("Playfair Cipher Key Matrix: ");

System.out.println();

//loop iterates for rows

**for**(**int** i = 0; i < 5; i++)

{

//loop iterates for column

**for**(**int** j = 0; j < 5; j++)

{

//prints the key-table in matrix form

System.out.print(printTable[i][j]+" ");

}

System.out.println();

}

System.out.println();

}

//method that prints all the results

**private** **void** printResults(String encipher, String dec)

  System.out.print("Encrypted Message: ");

//prints the encrypted message

System.out.println(encipher);

System.out.println();

System.out.print("Decrypted Message: ");

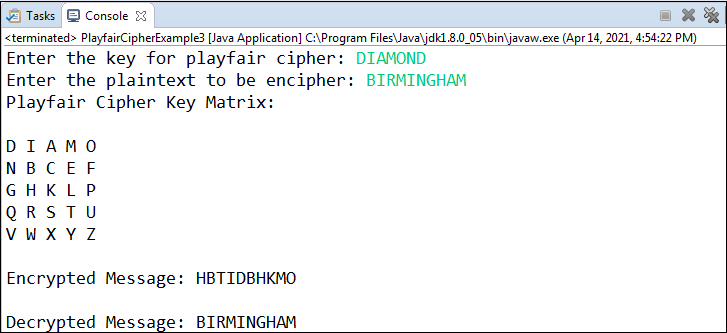
//prints the decryted message

System.out.println(dec);

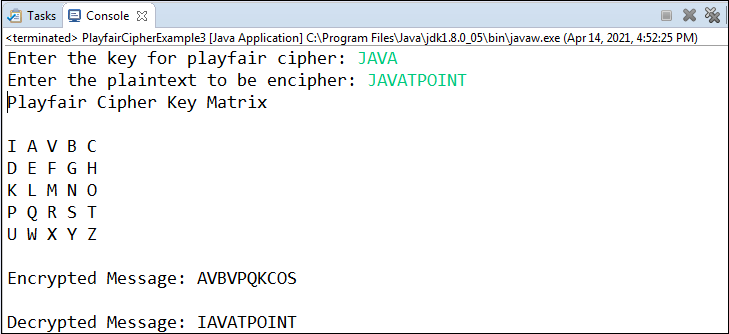
}

}

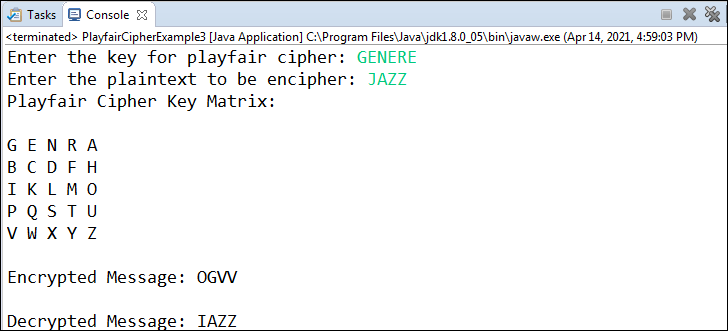
**Output 1:**



**Output 2:**



**Output 3:**



**EXPERIMENT-4**

Write a program to implement Autokey Cipher

/\*

Autokey encryption and decryption

\*/

import java.util.\*;

import java.lang.\*;

import java.math.\*;

public class Autokey{

private static String alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

public static void main(String[] args){

String text = "FOLLOWDIRECTION";

String key = "P"; //15 - P

if(key.matches("[-+]?\\d\*\\.?\\d+"))

key = ""+alpha.charAt(Integer.parseInt(key));

String enc = AutoEncryption(text,key);

System.out.println("Plaintext : "+text);

System.out.println("Encrypted : "+enc);

System.out.println("Decrypted : "+AutoDecryption(enc,key));

}

public static String AutoEncryption(String text,String key){

int len = text.length();

String subkey = key + text;

subkey = subkey.substring(0,subkey.length()-key.length());

String sb = "";

for(int x=0;x<len;x++){

int get1 = alpha.indexOf(text.charAt(x));

int get2 = alpha.indexOf(subkey.charAt(x));

int total = (get1 + get2)%26;

sb += alpha.charAt(total);

}

return sb;

}

public static String AutoDecryption(String text,String key){

int len = text.length();

String current = key;

String sb ="";

for(int x=0;x<len;x++){

int get1 = alpha.indexOf(text.charAt(x));

int get2 = alpha.indexOf(current.charAt(x));

int total = (get1 - get2)%26;

total = (total<0)? total + 26 : total;

sb += alpha.charAt(total);

current += alpha.charAt(total);

}

return sb;

}

}

**EXPERIMENT-5**

Write a program to implement Hill Cipher. (Use any matrix but find inverse yourself)

#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() {

   getKeyMatrix();

   encrypt();

   decrypt();

}

**OUTPUT**

Enter 3x3 matrix for key (should have inverse):

1

0

1

2

4

0

3

5

6

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

Encrypted string is: CER

Inverse Matrix is:

1.09091 0.227273 -0.181818

-0.545455 0.136364 0.0909091

-0.0909091 -0.227273 0.181818

Decrypted string is: ABC

**EXPERIMENT-6**

Write a program to implement Rail fencing technique

**a. For Encryption**

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

**using** **namespace** std;

**int** main(){

**int** t,n,m,i,j,k,sum=0;

string s;

cout<<"Enter the message"<<'\n';

cin>>s;

cout<<"Enter key"<<'\n';

cin>>n;

vector<vector<**char**>> a(n,vector<**char**>(s.size(),' '));

j=0;

**int** flag=0;

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

a[j][i] = s[i];

**if**(j==n-1){

flag=1;

}

**else** **if**(j==0)

flag=0;

**if**(flag==0){

j++;

}

**else** j--;

}

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

**for**(j=0;j<s.size();j++){

**if**(a[i][j]!=' ')

cout<<a[i][j];

}

}

cout<<'\n';

**return** 0;

}

**OUTPUT:**

Enter the message

HELLOWORLD

Enter key

3

HOLELWRDLO

**b. For Decryption**

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

**using** **namespace** std;

**int** main(){

**int** t,n,m,i,j,k,sum=0;

string s;

cout<<"Enter the message to decrypt"<<'\n';

cin>>s;

cout<<"Enter key"<<'\n';

cin>>n;

vector<vector<**char**>> a(n,vector<**char**>(s.size(),' '));

j=0;

**int** flag=0;

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

a[j][i] = '0';

**if**(j==n-1){

flag=1;

}

**else** **if**(j==0)

flag=0;

**if**(flag==0){

j++;

}

**else** j--;

}

**int** temp =0;

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

**for**(j=0;j<s.size();j++){

**if**(a[i][j]=='0')

a[i][j]= s[temp++];

}

}

flag=0;

j=0;

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

cout<<a[j][i];

**if**(j==n-1){

flag=1;

}

**else** **if**(j==0)

flag=0;

**if**(flag==0){

j++;

}

**else** j--;

}

cout<<'\n';

**return** 0;

}

**OUTPUT:**

Enter the message to decrypt

HOLELWRDLO

Enter key

3

HELLOWORLD

**EXPERIMENT-7**

Write a program to implement Simple Columnar Transposition Technique.

#include<stdio.h>

#include<string.h>

void cipher(int i, int c);

int findMin();

void makeArray(int, int);

char arr[22][22], darr[22][22], emessage[111], retmessage[111], key[55];

char temp[55], temp2[55];

int k = 0;

int main()

{

char \*message;

int i, j, klen, emlen, flag = 0;

int r, c, index, rows;

printf("Enter the key**\n**");

fflush(stdin);

gets(key);

printf("**\n**Enter message to be ciphered**\n**");

fflush(stdin);

gets(message);

strcpy(temp, key);

klen = strlen(key);

k = 0;

for (i = 0;; i++)

{

if (flag == 1)

break;

for (j = 0; key[j] != NULL; j++)

{

if (message[k] == NULL)

{

flag = 1;

arr[i][j] = '-';

}

else

{

arr[i][j] = message[k++];

}

}

}

r = i;

c = j;

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

{

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

{

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

}

printf("**\n**");

}

k = 0;

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

{

index = findMin();

cipher(index, r);

}

emessage[k] = '**\0**';

printf("**\n**Encrypted message is**\n**");

for (i = 0; emessage[i] != NULL; i++)

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

printf("**\n\n**");

//deciphering

emlen = strlen(emessage);

//emlen is length of encrypted message

strcpy(temp, key);

rows = emlen / klen;

//rows is no of row of the array to made from ciphered message

j = 0;

for (i = 0, k = 1; emessage[i] != NULL; i++, k++)

{

//printf("\nEmlen=%d",emlen);

temp2[j++] = emessage[i];

if ((k % rows) == 0)

{

temp2[j] = '**\0**';

index = findMin();

makeArray(index, rows);

j = 0;

}

}

printf("**\n**Array Retrieved is**\n**");

k = 0;

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

{

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

{

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

//retrieving message

retmessage[k++] = darr[i][j];

}

printf("**\n**");

}

retmessage[k] = '**\0**';

printf("**\n**Message retrieved is**\n**");

for (i = 0; retmessage[i] != NULL; i++)

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

return (0);

}

void cipher(int i, int r)

{

int j;

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

{

{

emessage[k++] = arr[j][i];

}

}

// emessage[k]='\0';

}

void makeArray(int col, int row)

{

int i, j;

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

{

darr[i][col] = temp2[i];

}

}

int findMin()

{

int i, j, min, index;

min = temp[0];

index = 0;

for (j = 0; temp[j] != NULL; j++)

{

if (temp[j] < min)

{

min = temp[j];

index = j;

}

}

temp[index] = 123;

return (index);

}

**OUTPUT:**

$ g++ TranspositionTechnique.cpp

$ a.out

Enter the key

hello

Enter the message to be ciphered

how are you

h o w a

r e y o

u - - - -

Encrypted message is

oe-hruw - y-ao-

Array Retrieved is

h o w a

r e y o

u - - - -

Message retrieved is

how are you----

------------------

(program exited with code: 0)

Press return to continue

**EXPERIMENT-9**

Case study of System threat attacks- Denial of Services

# Introduction:

A Denial of Service (DoS) attack attempts to make an online service or a website unavailable by overloading it with vast floods of internet traffic generated from multiple sources. Exploited machines can include computers and other networked resources such as IoT devices.

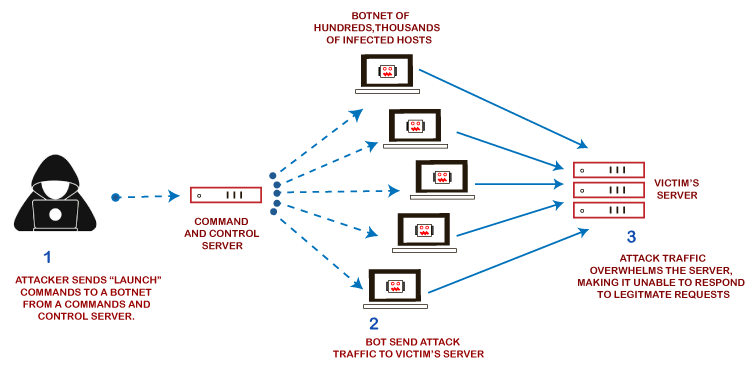
A Denial of Service (DoS) attack, in which one computer and one Internet connection are used to flood a targeted resource with packets, but a DDoS attack uses many computers and many Internet connections, often distributed globally in what is referred to as a **botnet**.

A large-scale volumetric DDoS attack can generate traffic measured in tens of Gigabits (and even hundreds of Gigabits) per second. A regular network will not be able to handle such traffic.

Attackers build a network of hacked machines known as **botnets** by spreading malicious code through emails, websites, and social media. Once these computers are infected, they can be controlled remotely, without their owners' knowledge, and used as an army to launch an attack against any target.

# How DoS attack works?

DoS attacks are carried out with networks of Internet-connected machines. A DDoS attack can be generated in the following step by step way, such as:



1. These networks consist of computers and other devices such as IoT devices that have been infected with malware, allowing them to be controlled remotely by an attacker. These individual devices are referred to as bots or zombies, and a group of bots is called a **botnet**.
2. Once a botnet has been established, the attacker can direct an attack by sending remote instructions to each bot. It can use for sending more connection requests than a server can handle at a time.
3. Attackers can have computers send a victim resource huge amounts of random data to use up the target's bandwidth.
4. When the botnet targets a victim's server or network, each bot sends requests to the target's IP address, potentially causing the server or network to become overload, resulting in a denial-of-service to regular traffic.

Due to the distributed nature of these machines, they can use to generate distributed high traffic, which may be difficult to handle. It finally results in a complete blockage of a service.

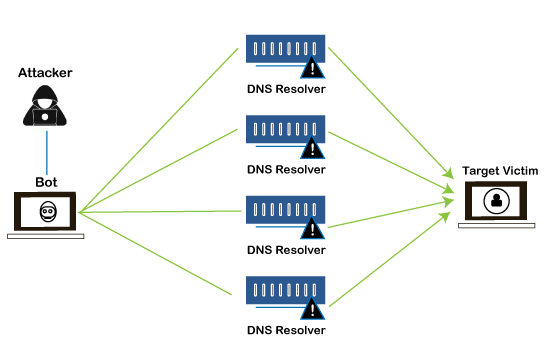
# Types of DoS Attacks

Denial of Service attacks can be broadly categorized into these three categories:

# 1. Volume-Based Attacks

Volume-based attacks use massive amounts of fake traffic to overwhelm a resource such as a website or a server.

It includes TCP floods, UDP floods, ICMP floods, and other spoofed-packet floods. These are also called Layer 3 & 4 Attacks. Here, an attacker tries to saturate the bandwidth of the target site. The attack magnitude is measured in **Bits per Second** (bps).

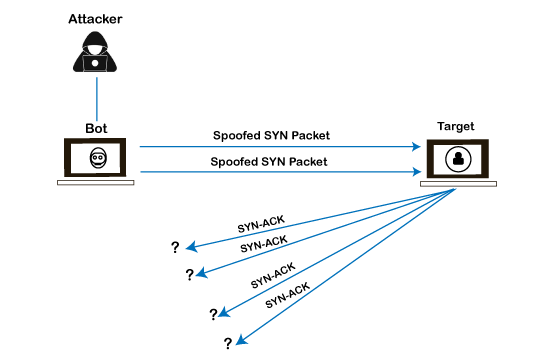


* **Amplification Attack:** The attacker makes a request that generates a significant response which includes DNS requests for large TXT records and HTTP GET requests for large files like images, PDFs, or any other data files.
* **UDP Flood:** A UDP flood is used to flood random ports on a remote host with numerous UDP packets, more specifically port number 53. Specialized firewalls are used to filter out or block malicious UDP packets.
* **ICMP Flood:** This is similar to UDP flood and flooded a remote host with numerous ICMP Echo Requests. This type of attack can consume both outgoing and incoming bandwidth, and a high volume of ping requests will result in overall system slowdown.
* **HTTP Flood:** The attacker sends HTTP GET and POST requests to a targeted web server in a large volume that the server cannot handle and leads to denial of additional connections from legitimate clients.

**2. Protocol Attacks**

Protocol or network-layer DDoS attacks send large numbers of packets to targeted network infrastructures and infrastructure management tools.

It includes SYN floods, Ping of Death, fragmented packet attacks, Smurf DDoS, etc. This type of attack consumes existing server resources and other resources, such as firewalls and load balancers. The attack magnitude is measured in **Packets per Second** (PPS).

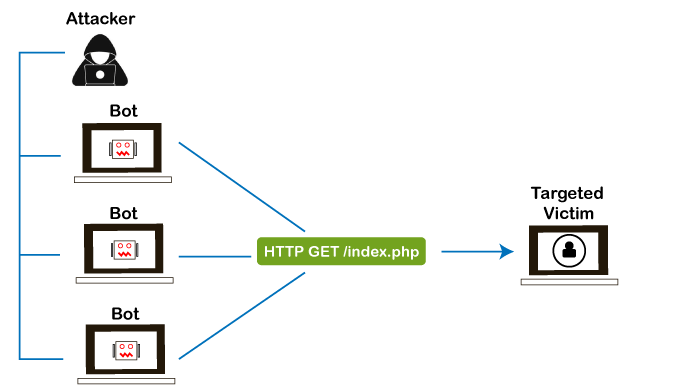


* **SYN Flood:** The attacker sends TCP connection requests faster than the targeted machine can process them, causing network saturation. Administrators can tweak TCP stacks to mitigate the effect of SYN floods. To reduce the effect of SYN floods, you can reduce the timeout until a stack frees memory allocated to a connection or selectively dropping incoming connections using a **firewall** or **iptables**.
* **DNS Flood:** DNS floods are used for attacking both the infrastructure and a DNS application to overload a target system and consume all its available network bandwidth.
* **Ping of Death:** The attacker sends malformed or oversized packets using a simple ping command. IP allows sending 65,535 bytes packets but sending a ping packet larger than 65,535 bytes violates the Internet Protocol and could cause memory overflow on the target system and finally crash the system. Many sites block ICMP ping messages altogether at their firewalls to avoid Ping of Death attacks and their variants.

**3. Application Layer Attacks**

Flooding applications with maliciously crafted requests conduct Application-layer attacks. The size of application-layer attacks is measured in requests per second (rps).

It includes Slowloris, Zero-day DDoS attacks, DDoS attacks that target Apache, Windows, or OpenBSD vulnerabilities, and more. Here the goal is to crash the webserver.



* **Application Attack:** This is also called Layer 7 Attack, where the attacker makes excessive log-in, database-lookup, or search requests to overload the application. It is tough to detect Layer 7 attacks because they resemble legitimate website traffic.
* **Slowloris:** The attacker sends many HTTP headers to a targeted web server but never completes a request. The targeted server keeps each of these false connections open and eventually overflows the maximum concurrent connection pool, leading to a denial of additional connections from legitimate clients.
* **NTP Amplification:** The attacker exploits publically accessible Network Time Protocol (NTP) servers to overwhelm the targeted server with User Datagram Protocol (UDP) traffic.
* **Zero-day DDoS Attacks:** Zero-day vulnerability is a system or application flaw previously unknown to the vendor and has not been fixed or patched. These are new types of attacks coming into existence day by day, such as exploiting vulnerabilities for which no patch has yet been released.

### How to Fix a DoS Attack

We must be careful while approaching and selecting a DDoS protection service provider. Many service providers want to take advantage of your situation. If you inform them that we are under a DDoS attack, they will start offering you various services at unreasonably high costs.

If we see a low magnitude of the DDoS, you can find many firewall-based solutions that can help you filter out DDoS-based traffic. If you have a high volume of DDoS attacks like in gigabits or even more, we should take the help of a DDoS protection service provider that offers a more holistic, proactive, and g enuine approach.

There are quite a few DDoS protection options that we can apply depending on the type of DoS attack.

**1. Blocking vulnerable ports**

Your DDoS protection starts from identifying and closing all the possible OS and application-level vulnerabilities in your system, closing all the possible ports, removing unnecessary access from the system, and hiding your server behind a proxy or CDN system.

**2. Configure firewalls and routers**

Firewalls and routers should be configured to reject bogus traffic, and you should keep your routers and firewalls updated with the latest security patches. These remain your initial line of defense.

Application front-end hardware integrated into the network before traffic reaches a server analyzes and screens data packets classifying the data as a priority, regular, or dangerous as they enter a system and can be used to block threatening data.

**3. Consider artificial intelligence**

While present defenses of advanced firewalls and intrusion detection systems are common, AI is being used to develop new systems.

The systems that can quickly route Internet traffic to the cloud, where it's analyzed, and malicious web traffic blocked before it reaches a company's computers. Such AI programs could identify and defend against known DDoS indicative patterns. Plus, the self-learning capabilities of AI would help predict and identify future DDoS patterns.

Researchers are exploring the use of blockchain, the same technology behind Bitcoin and other cryptocurrencies, to permit people to share their unused bandwidth to absorb the malicious traffic created in a DDoS attack and render it ineffective.

**4. Secure IoT devices**

If you have IoT devices, you should make sure your devices are formatted for maximum protection. Secure passwords should be used for all devices. IoT devices have been vulnerable to weak passwords, with many devices operating with easily discovered default passwords.

A strong firewall is also important. Protecting your devices is an essential part of Cyber Safety.

**5. Application front end hardware**

Application front-end hardware is intelligent hardware placed on the network before traffic reaches the servers. It can be used on networks in conjunction with routers and switches. Application front-end hardware analyzes data packets as they enter the system and then identifies them as a priority, regular, or dangerous. There are more than 25 bandwidth management vendors.

**6. Blackhole and sinkhole**

With **blackhole** routing, all the traffic to the attacked DNS or IP address is sent to a "black hole" (null interface or a non-existent server). It is managed by the ISP to be more efficient and avoid affecting network connectivity.

A DNS **sinkhole** routes traffic to a valid IP address which analyzes traffic and rejects bad packets. Sinkholing is not efficient for most severe attacks.

**7. IPS based prevention**

Intrusion prevention systems (IPS) are effective if the attacks have signatures associated with them. However, the trend among the attacks is to have legitimate content but bad intent. Intrusion-prevention systems which work on content recognition cannot block behavior-based DoS attacks.

An ASIC-based IPS may detect and block denial-of-service attacks because they have the processing power and the granularity to analyze the attacks and act like a circuit breaker in an automated way.

A rate-based IPS (RBIPS) must analyze traffic granularly and continuously monitor the traffic pattern and determine if there is a traffic anomaly. It must let the legitimate traffic flow while blocking the DoS attack traffic.

**8. DDS based defense**

More focused on the problem than IPS, a DoS defense system (DDS) can block connection-based DoS attacks and those with legitimate content but bad intent. A DDS can also address both protocol attacks (such as teardrop and ping of death) and rate-based attacks (such as ICMP floods and SYN floods). DDS has a purpose-built system that can quickly identify and obstruct denial of service attacks at a more incredible speed than software that is based system.

**9. Switches**

Most switches have some rate-limiting and ACL capability. Some switches provide automatic or system-wide rate limiting, traffic shaping, delayed binding (TCP splicing), deep packet inspection, and Bogon filtering (bogus IP filtering) to detect and remediate DoS attacks through automatic rate filtering and WAN Link failover and balancing.

These schemes will work as long as the DoS attacks can be prevented by using them. For example, SYN flood can be prevented using delayed binding or TCP splicing. Similarly, content-based DoS may be prevented using deep packet inspection. Attacks originating from dark addresses or going to dark addresses can be prevented using bogon filtering. Automatic rate filtering can work as long as set rate thresholds have been set correctly. Wan-link failover will work as long as both links have DoS/DDoS prevention mechanism.

**EXPERIMENT-10**

Case study of Sniffing and Spoofing attacks

**Introduction:**

**SNIFFING**

The practice or technique of monitoring, gathering, capturing, and logging some or all data packets passing through a given computer network is called sniffing or packet sniffing.  A packet sniffer is composed of two parts namely; a network adapter and software that is used by a network to observe or troubleshoot network traffic.

Attackers use these sniffers to seize data packets that contain valuable information and analyze the network traffic. Sniffing is categorized into active sniffing and passive sniffing. In Active sniffing, there is the constant activity by the attacker to obtain information and sniff the traffic from the switch network. In passive sniffing, the attacker is hidden and sniffs through the hub.

**SPOOFING**

Any kind of behavior where an attacker mask as an authentic user or a device to secure something beneficial or crucial information for their gain is called spoofing. There are various kinds of spoofing such as website spoofing, E-mail spoofing, and IP spoofing. Other common methods include ARP spoofing attacks and DNS server spoofing attacks.

An E-mail spoofing targets the user while an IP spoofing is predominantly targeted at a network.

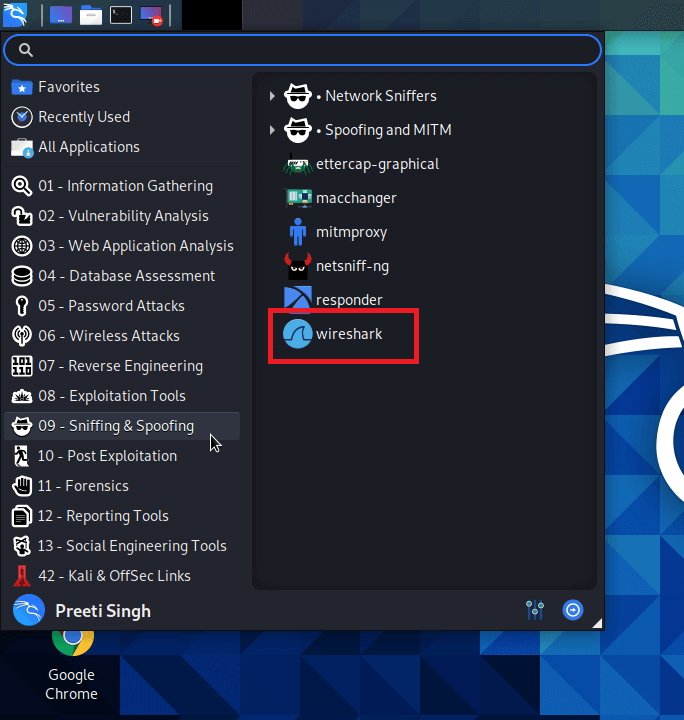
In an IP spoofing attack, the attacker attempts to obtain illicit and illegal access to a network through messages with a bogus or spoofed IP address to deceive and show it off as a message from a trusted source. This is achieved by using a genuine host’s IP address and varying the packet headers led from their personal system to mimic it as an original and a trusted computer’s IP address.

**Tools for Sniffing and Spoofing**

Linux has several tools for sniffing and spoofing network traffic. The following are the top 5 tools for sniffing and spoofing:

## 1. Wireshark

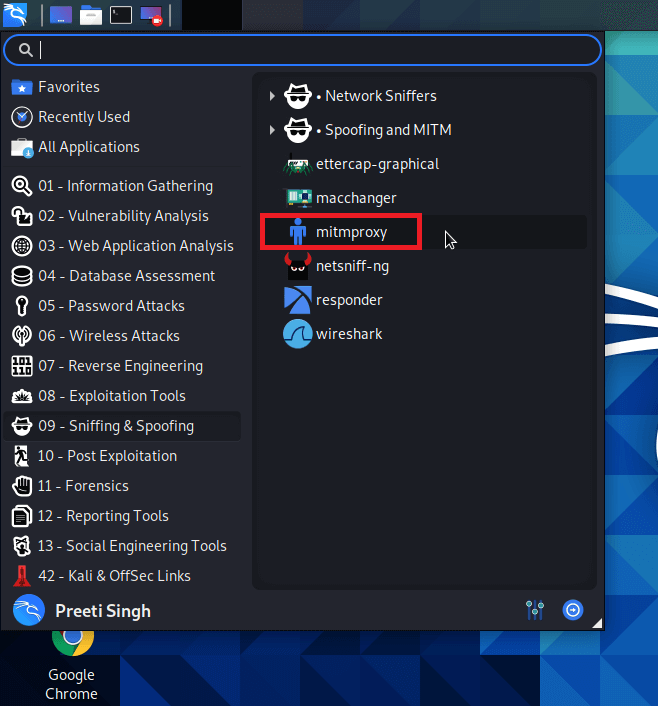
One of the best and widely used tools for sniffing and spoofing is Wireshark. Wireshark is a network traffic analysis tool with a plethora of capabilities. Wireshark's extensive library of protocol dissectors is one of its most distinguishing features. These allow the tool to analyze a wide range of protocols, break out the several fields in every packet and display them within an accessible **graphical user interface (GUI).** Users with even basic network understanding will be able to understand what they are looking for. In addition, Wireshark has a number of traffic analysis tools, such as statistical analysis and the capability to follow network sessions or decrypt **SSL/TLS traffic.**



Wireshark is a useful sniffing tool since it gives us a lot of information about network traffic, either from a **capture file** or a **live capture.** This can help with understanding the network layout, capturing leaked credentials, and another task.

## 2. Mitmproxy

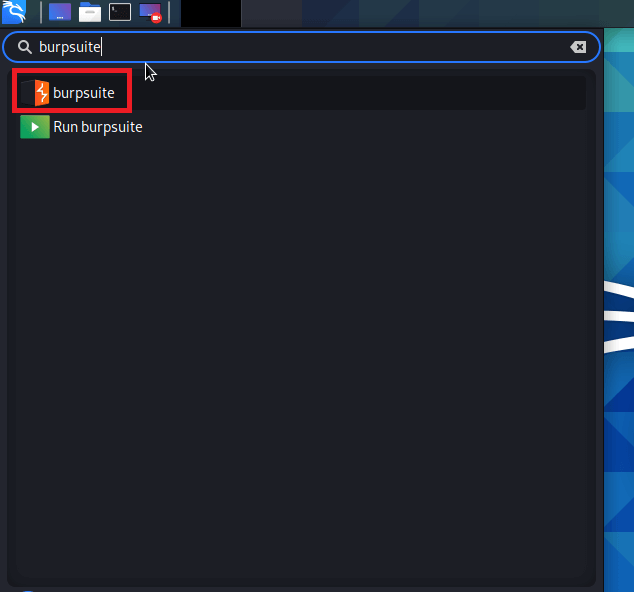
The attacker interjects oneself into communication between a client and a server in a **man-in-the-middle (MitM)** attack. All traffic passing across that connection is intercepted by the attacker, giving them the ability to listen in on traffic and change data passing through the network.



With the help of the **mitmproxy** in Kali Linux, we can easily perform the MitM attacks on web traffic. It enables the **on-the-fly** capture and modification of **HTTP traffic,** supports client and server traffic replay, and comprises the capability to automate attacks using **Python.** Mitmproxy may also support the interception of **HTTPs** traffic with **SSL** certificates generated **on-the-fly.**

## 3. Burp Suite

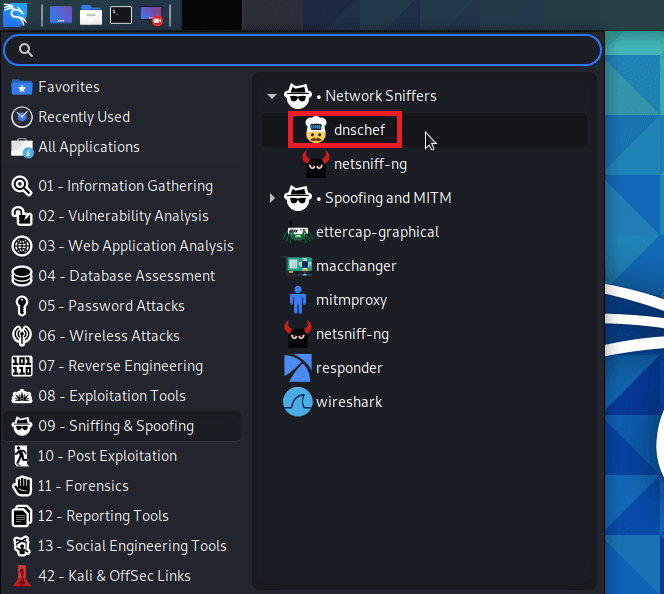
Burp Suite is a collection of various penetration tools. Its main focus is on web application security analysis. **Burp Proxy** is the main tool in **Burp Suite** which is helpful in Sniffing and Spoofing. Burp Proxy permits us to intercept and modify HTTP connections, as well as provides supports for intercept HTTPs connection.



It works on a freemium model. Although the basic tools are free, attacks must be carried out manually without the ability to save work. Paying for a license grants us access to a larger tool **(including a web vulnerability scanner)** as well as automation support.

## 4. dnschef

The **dnschef** tool is a DNS proxy that may be used to **analyze malware** and **penetration testing.** A highly configurable DNS proxy, dnschef, is used for analyzing network traffic. This DNS proxy can generate fake requests and utilize these requests to be sent to a local machine rather than a real server. This tool works on multiple platforms and can generate phony requests and responses depending on domain lists. Different DNS record types are also supported by the dnschef tool.

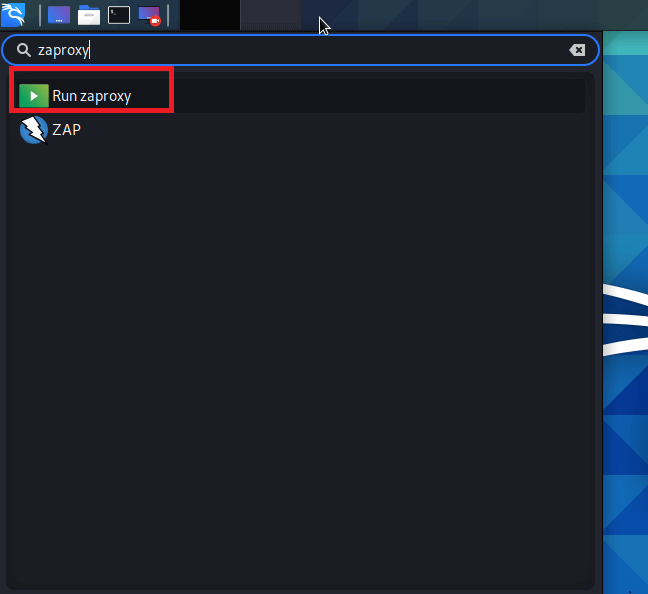


When it is impossible to force an application to use another proxy server, a DNS proxy must be used instead. If a mobile application disregards the HTTP proxy setting, dnschef can deceive it by faking requests and answers to a specified target.

## 5. Zaproxy

On Kali Linux, the executable named **Zaproxy** is **OWASP's** Zed Attack Proxy **(ZAP).** Like Burp Suite, **ZAP** is a penetration testing tool that aids in detecting and exploiting vulnerabilities within web applications.

Because of its ability to intercept and modify **HTTP(S)** traffic, **ZAP** is a handy tool for sniffing and spoofing. ZAP has a lot of functionality and is a completely free tool for carrying out these attacks.



**EXPERIMENT-11**

Write a program to implement Simple RSA algorithm with small numbers.

def gcd(a, b):

while a != b:

if a > b:

a = a - b

if b > a:

b = b - a

return a

p = int(input("Enter a Prime Number - P : "))

q = int(input("Enter another Prime Number - Q : "))

n = p \* q

print("Value of MODULUS - N : ", n)

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

print("Value of Totient of N - T\_N : ", t\_n)

e = -1

for i in range(4, t\_n):

if gcd(i, t\_n) == 1:

e = i

break

print("Public Key used for Encryption - E : ", e)

i = 1

d = -1

while True:

res = 1 + i \* t\_n

if res % e == 0:

d = res // e

break

i = i + 1

print("Private Key used for Decryption - D : ", d)

msg = int(input("Enter the Message - an Integer : "))

c\_text = (msg \*\* e) % n

print("Cipher Text Message is : ", c\_text)

p\_text = (c\_text \*\* d) % n

print("Plain Text Message is : ", p\_text)

**Output:**

Enter a Prime Number - P : 17

Enter another Prime Number - Q : 11

Value of MODULUS - N : 187

Value of Totient of N - T\_N : 160

Public Key used for Encryption - E : 7

Private Key used for Decryption - D : 23

Enter the Message - an Integer : 5

Cipher Text Message is : 146

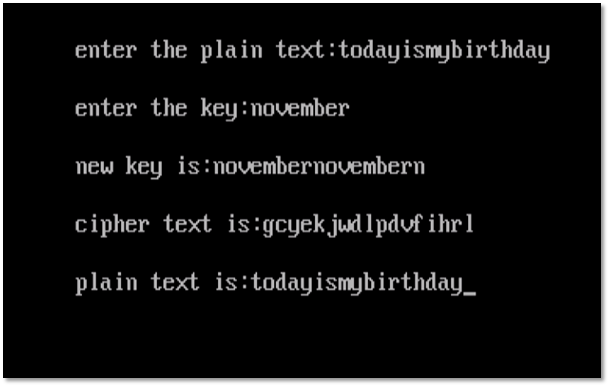
Plain Text Message is : 5

**EXPERIMENT-12**

Write a program to implement poly alphabetic Cipher

//POLY ALPHABATIC CIPHER  
#include<stdio.h>  
#include<conio.h>  
#include<string.h>  
void main()  
{  
    char pt[20]={'\0'},ct[20]={'\0'},key[20]={'\0'},rt[20]={'\0'};  
    int i,j;  
    clrscr();  
    printf("\n enter the plain text:");  
    scanf("%s",pt);  
    printf("\n enter the key:");  
    scanf("%s",key);  
  
    //length of plaintext equal to length of key  
    j=0;  
    for(i=strlen(key);i<strlen(pt);i++)  
    {  
    if(j==strlen(key))  
    {  
    j=0;  
    }  
    key[i]=key[j];  
    j++;  
    }  
    printf("\n new key is:%s",key);  
  
    //converting plain text to cipher text (encryption)  
    for(i=0;i<strlen(pt);i++)  
    {  
        ct[i]=(((pt[i]-97)+(key[i]-97))%26)+97;  
    }  
    printf("\n \n cipher text is:%s",ct);  
  
    //converting cipher text to plain text (decryption)  
    for(i=0;i<strlen(ct);i++)  
    {  
    if(ct[i]<key[i])  
    {  
        rt[i]=26+((ct[i]-97)-(key[i]-97))+97;  
    }  
    else  
        rt[i]=(((ct[i]-97)-(key[i]-97))%26)+97;  
    }  
    printf("\n \n plain text is:%s",rt);  
    getch();  
}

**Output:**

****