

# **Information security**

## **Practical 1**

### 1. What is cryptography?

Cryptography is a method of protecting information and communications through the use of codes, so that only those for whom the information is intended can read and process it.

### 2. what is symmetric key cryptography?

Symmetric Key Cryptography also known as Symmetric Encryption is when a secret key is leveraged for both encryption and decryption functions. This method is the opposite of Asymmetric Encryption where one key is used to encrypt and another is used to decrypt.

#### ○ Examples:

Some examples of where symmetric cryptography is used are:

- ✦ Payment applications, such as card transactions where PII needs to be protected to prevent identity theft or fraudulent charges.
- ✦ Validations to confirm that the sender of a message is who he claims to be.
- ✦ Random number generation or hashing.

### ⚔ Attacks on symmetric key cryptography

reference:

Attacks against encrypted information fall into three main categories. They are:

1. Key search, brute force attacks

- 
- 2. Cryptanalysis
- 3. Systems-based attacks

#### Key Search (Brute Force) Attacks

The most straight-forward attack on an encrypted message is simply to attempt to decrypt the message with every possible key. Most of these attempts will fail. But one might work. At which point you can decrypt the message and any others that that key is used on. There is no way to defend against a key search attack because there is no way to keep an attacker from trying to decrypt your message with every possible key. Key searches are not very efficient. If the chosen key is long enough, a key search attack is not even feasible. For example, with a 128 bit key and any conceivable computing technology, life on Earth will cease to exist long before even a single key is likely to be cracked.

- Cryptanalysis

Most encryption algorithms can be defeated by using a combination of sophisticated mathematics and computing power. The results are that many encrypted messages can be deciphered without knowing the key. A skilled cryptanalyst can sometimes decipher encrypted text without even knowing the encryption algorithm.

A cryptanalytic attack can have two possible goals. The cryptanalyst might have ciphertext and want to discover the plaintext, or the cryptanalyst might have ciphertext and want to discover the encryption key that was used to encrypt the message. The following attacks are commonly used when the encryption algorithm is known. These may be applied to encrypted files or Internet traffic:

- Known Plaintext Attack

In this type of attack the cryptanalyst has a block of plaintext and a corresponding block of ciphertext. The goal of a known plaintext attack is to determine the cryptographic key and possibly the algorithm which can then be used to decrypt other messages.

#### Chosen Plaintext Attack

- The cryptanalyst has the subject of the attack unknowingly encrypt chosen blocks of data creating a result that the cryptanalyst can then analyze. The goal of a chosen plaintext attack is to determine the cryptographic key which can then be used to decrypt other messages.
- Differential Cryptanalysis  
This attack, which is a form of chosen plaintext attack, involves encrypting many texts that are only slightly different from one another and comparing the results.
- Differential Fault Analysis  
The attack works against cryptographic systems that are built in hardware. The device is subjected to environmental factors (heat, stress, radiation, etc) designed to coax the device into making mistakes during the encryption or decryption operation. These faults can be analyzed, and from them the device's internal state, including the encryption key or algorithm, can possibly be discovered.
- Differential Power Analysis  
This is another attack against cryptographic hardware, in particular smart cards. By observing the power that a smart card uses to encrypt a chosen block of data, it is possible to learn a little bit of information about the structure of the secret key. By subjecting the smart card to a number of specially chosen data blocks and carefully monitoring the power used, it is possible to determine the secret key.
- Differential Timing Analysis  
This attack is similar to differential power analysis except that the attacker carefully monitors the time that the smart card takes to perform the requested encryption operations.

### Systems Based Attack

Another way of breaking a code is to attack the cryptographic system that uses the cryptographic algorithm without actually attacking the algorithm itself.

- 

- Early TV Satellite System

One of the most spectacular cases of a systems-based attack was the VC-I video encryption algorithm that was used in early satellite TV systems. Satellite signals were encrypted with VC-I encryption as part of the satellite subscription for premium channels. For many years, video pirates sold decoder boxes that could intercept the transmission of keys and use them to decrypt the satellite TV signals. The video pirates would capture the encryption keys and update the receiver boxes of the satellite customers. This way the satellite customers could get the premium channels for free.

- Netscape's SSL Implementation

Many of the early attacks against Netscape's implementation of SSL were actually attacks on Netscape Navigator's implementation, rather than on the SSL protocol itself. Researchers David Wagner and Ian Goldberg at the University of California at Berkeley discovered that Navigator's random number generator was not really random. It was possible for attackers to closely monitor the computer that Navigator was running on, predict the random number generator's starting configuration, and determine the randomly chosen key using a fairly straightforward configuration.

- Covert Channels

Covert channels are defined as any communication channel that can be exploited by a process to transfer information in a manner that violates the system's security policy. If an attacker cannot decrypt email messages, she may be able to gain information by examining the message sender,

## **Practical 2**

❖ **Aim:** Implementation of Caesar cipher program in c.

### **Program:**

➤ Part 1:- Encryption

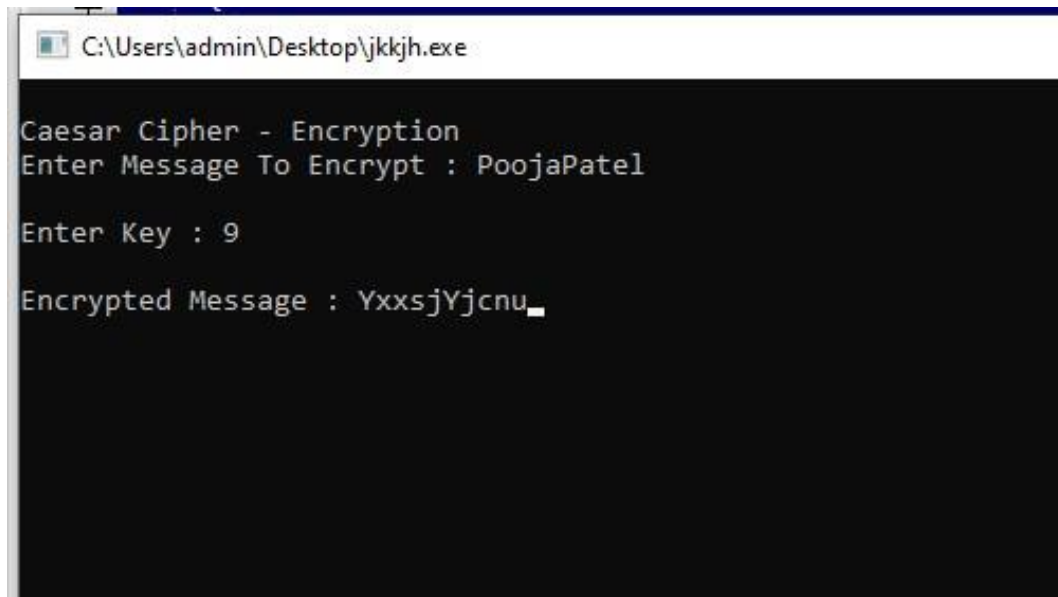
```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
void main() {
int i, key;      char
text[100],c;

    printf("\nCaesar Cipher - Encryption");
printf("\nEnter Message To Encrypt : ");
gets(text);      printf("\nEnter Key : ");
scanf("%d", &key);

    for(i=0;text[i]!='\0';++i)
    {
        c=text[i];
if(c>='a'&&c<='z')
    {
        c=c+key;
if(c>'z')
        {
c=c-'z'+'a'-1;
```

```
}
text[i]=c;
    }
    else if(c>='A'&&c<='Z')
    {
c=c+key;
if(c>'Z')      {
c=c-'Z'+'A'-1;
    }
text[i]=c;
    }
    }
printf("\nEncrypted Message : %s", text);
getch();
}
```

**Output:**



```
C:\Users\admin\Desktop\jkkjh.exe

Caesar Cipher - Encryption
Enter Message To Encrypt : PoojaPatel

Enter Key : 9

Encrypted Message : YxxsjYjcnu_
```

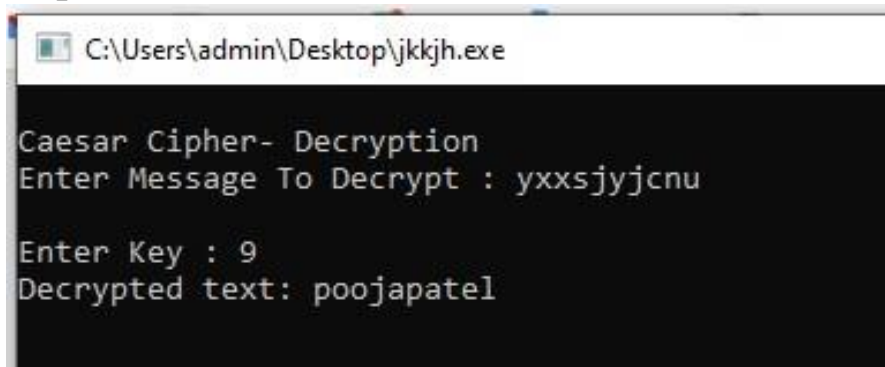
➤ Part 2:- Decryption

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
void main() {
    int i, key;
    char text[100],c;

    printf("\nCaesar Cipher- Decryption");
    printf("\nEnter Message To Decrypt : ");
    gets(text);    printf("\nEnter Key : ");
    scanf("%d", &key);
    for(i = 0; text[i] != '\0'; ++i)
    {
        c =
        text[i];
        if(c >= 'a' &&
        c <= 'z')
```

```
    {      c = c
- key;    if(c
< 'a')    {
    c = c + 'z' - 'a' + 1;
    }
text[i] = c;
    }
    else if(c >= 'A' && c <= 'Z')
    {      c = c
- key;    if(c
< 'A')    {
    c = c + 'Z' - 'A' + 1;
    }
text[i] = c;
    }
    }
    printf("Decrypted text: %s", text);
    getch();
}
```

Output:



```
C:\Users\admin\Desktop\jkkjh.exe

Caesar Cipher- Decryption
Enter Message To Decrypt : yxxsjycnu

Enter Key : 9
Decrypted text: poojapatel
```



### **Practical 3**

❖ **Aim:** Implementation of Playfair cipher program in c.

#### **Program:**

➤ Part 1:- Encryption

// C program to implement Playfair Cipher

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

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

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

```
#define SIZE 30
```

```
// Function to convert the string to lowercase void
```

```
toLowerCase(char plain[], int ps)
```

```
{
```

```
    int i;
```

```
    for (i = 0; i < ps; i++) {                if
```

```
(plain[i] > 64 && plain[i] < 91)
```

```
        plain[i] += 32;
```

```
    }
```

```
}
```

```
// Function to remove all spaces in a string 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;
```

```
}
```

```
// Function to generate the 5x5 key square
void generateKeyTable(char key[], int ks, char keyT[5][5])
{
    int i, j, k, flag = 0, *dicty;

    // a 26 character hashmap // to
    store count of the alphabet 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;
        }
    }
}

// Function to search for the characters of a digraph
// in the key square and return their position 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; }
        }
    }
}
```

```
}

// Function to find the modulus with 5 int
mod5(int a)
{
    return (a % 5);
}

// Function to make the plain text length to be even
int prepare(char str[], int ptrs)
{
    if (ptrs % 2 != 0) {
        str[ptrs++] = 'z';
        str[ptrs] = '\0';
    }
    return ptrs;
}

// Function for performing the encryption 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]];
        }
    }
}

// Function to encrypt using Playfair Cipher void
encryptByPlayfairCipher(char str[], char key[])
{
    char ps, ks, keyT[5][5];

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

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

    ps = prepare(str, ps);

    generateKeyTable(key, ks, keyT);

    encrypt(str, keyT, ps);
}

// Driver code int
main()
{
    char str[SIZE], key[SIZE];
```

```
// Key to be encrypted
strcpy(key, "Monarchy");
printf("Key text: %s\n", key);

// Plaintext to be encrypted
strcpy(str, "instruments");
printf("Plain text: %s\n", str);

// encrypt using Playfair Cipher
encryptByPlayfairCipher(str, key);

printf("Cipher text: %s\n", str);

return 0;
}
```

Output:

C:\Users\admin\Desktop\jkkjh.exe

```
Key text: Monarchy
Plain text: instruments
Cipher text: gatlmzclrqtx

-----
Process exited after 0.03368 seconds with return value 0
Press any key to continue . . .
```

➤ Part 2:- Decryption

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define SIZE 30
```

```
// Convert all the characters // of a
string to lowercase void
toLowerCase(char plain[], int ps)
{
    int i;
    for (i = 0; i < ps; i++) {
        if (plain[i] > 64 && plain[i] < 91)
            plain[i] += 32;
    }
}
```

```
// Remove all spaces in a string // can be
extended to remove punctuation 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;
}
```

```
// generates the 5x5 key square
void generateKeyTable(char key[], int ks,
                    char keyT[5][5])
{
    int i, j, k, flag = 0, *dicty;

    // a 26 character hashmap //
    to store count of the alphabet
    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;
                }
            }
        }
    }
```

// Search for the characters of a digraph //  
in the key square and return their position



```
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;
    }
    }
    }
}

// Function to find the modulus with 5 int
mod5(int a)
{
    return (a % 5);
}

// Function to decrypt
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]];
        }
    }
}
```

// Function to call decrypt

```
void decryptByPlayfairCipher(char str[], char key[])
```

```
{
    char ps, ks, keyT[5][5];

    // Key
    ks = strlen(key);    ks =
removeSpaces(key,      ks);
toLowerCase(key, ks);
    // ciphertext    ps =
strlen(str);
toLowerCase(str, ps);
    ps = removeSpaces(str, ps);

    generateKeyTable(key, ks, keyT);

    decrypt(str, keyT, ps);
}
```

```
}

// Driver code
int
main()
{
    char str[SIZE], key[SIZE];

    // Key to be encrypted
    strcpy(key, "Monarchy");
    printf("Key text: %s\n", key);

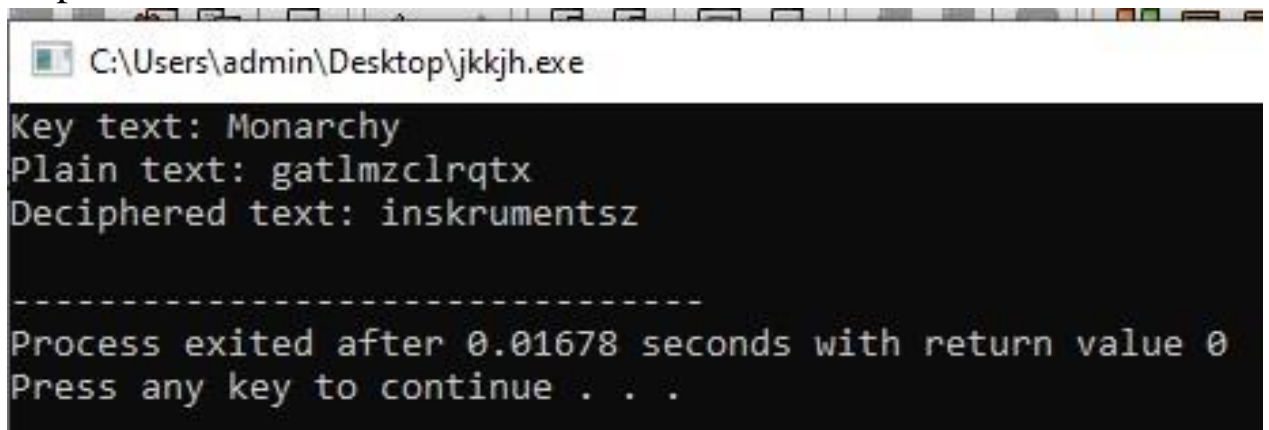
    // Ciphertext to be decrypted
    strcpy(str, "gatlmzclrqtz");
    printf("Plain text: %s\n", str);

    // encrypt using Playfair Cipher
    decryptByPlayfairCipher(str, key);

    printf("Deciphered text: %s\n", str);

    return 0;
}
```

Output:



```
C:\Users\admin\Desktop\jkkjh.exe
Key text: Monarchy
Plain text: gatlmzclrqtz
Deciphered text: inskrumentsz

-----
Process exited after 0.01678 seconds with return value 0
Press any key to continue . . .
```

## Practical 4

❖ **Aim:** Implementation of Hill cipher program in c.

Program:

```
#include<stdio.h>
int check(int x)
{ if(x%3==0)
return 0;

int a=x/3; int
b=3*(a+1);
int c=b-x;

return c;
}

void main()
{
int l,i,j;
int temp1; int
k[3][3]; int
p[3][1]; int
c[3][1]; char
ch;
printf("\nThe cipher has a key of length 9. ie. a 3*3 matrix.\nEnter
the 9 character key. ");

for(i=0;i<3;++i)
{
for(j=0;j<3;++j)
{
scanf("%c",&ch);
if(65<=ch && ch<=91)
k[i][j]=(int)ch%65; else
```

```
k[i][j]=(int)ch%97;  
}  
}
```

```
for(i=0;i<3;++i)  
{  
for(j=0;j<3;++j)  
{  
printf("%d ",k[i][j]);  
}  
printf("\n");  
}
```

```
printf("\nEnter the length of string to be encoded(without spaces).  
");
```

```
scanf("%d",&l);
```

```
temp1=check(l);
```

```
if(temp1>0)
```

```
printf("You have to enter %d bogus characters.",temp1);
```

```
char pi[l+temp1]; printf("\nEnter  
the string. "); for(i=-  
1;i<l+temp1;++i)  
{  
scanf("%c",&pi[i]);  
}
```

```
int temp2=l; int  
n=(l+temp1)/3; int
```

```
temp3; int flag=0;
int count;
printf("\n\nThe encoded cipher is : ");

while(n>0) { count=0;
for(i=flag;i<flag+3;++i
)
{ if(65<=pi[i] &&
pi[i]<=91)
temp3=(int)pi[i]%65; else
temp3=(int)pi[i]%97;

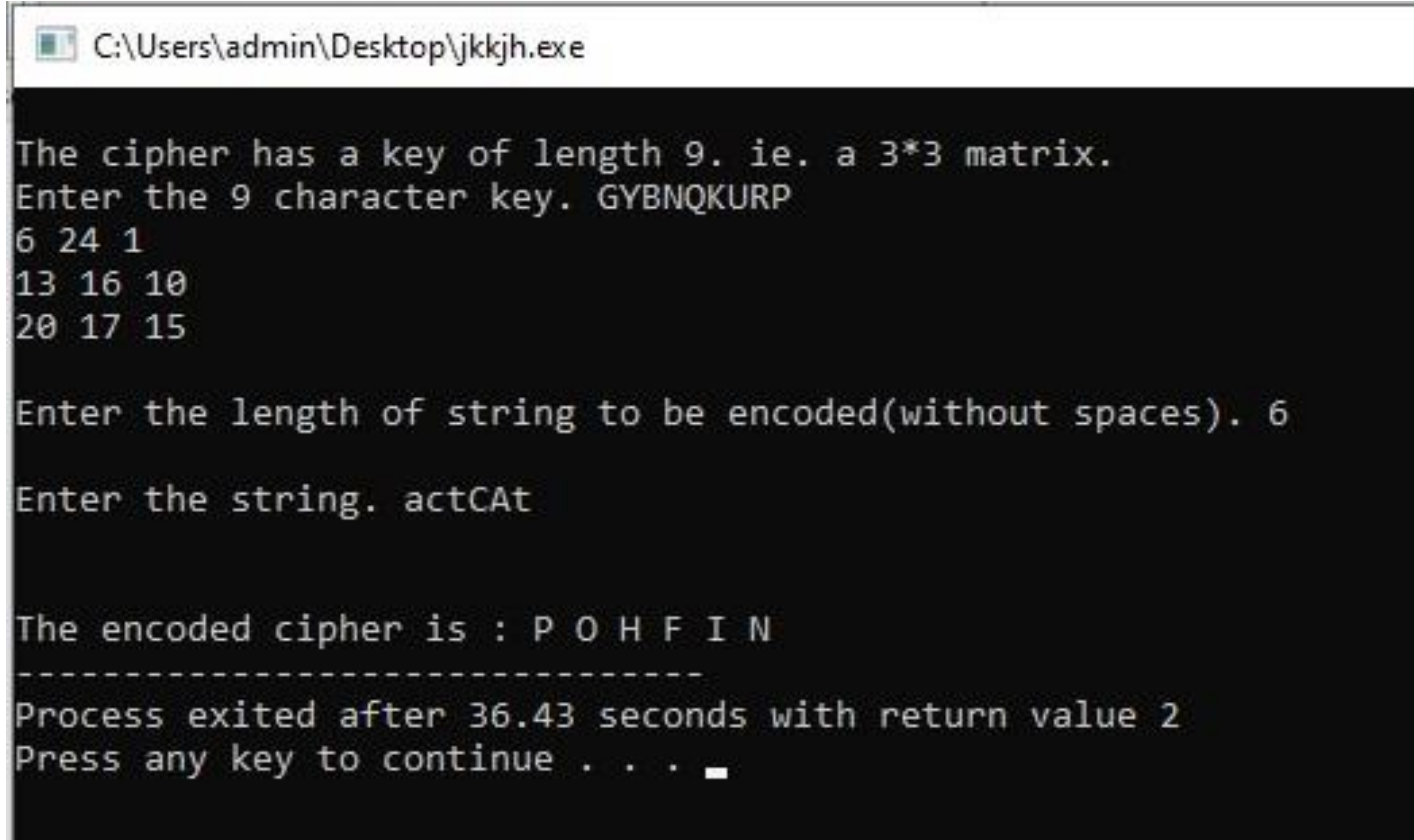
p[count][0]=temp3; count=count+1;
}

int k1; for(i=0;i<3;++i)
c[i][0]=0;

for(i=0;i<3;++i) {
for(j=0;j<1;++j)
{ for(k1=0;k1<3;++k1)
c[i][j]+=k[i][k1]*p[k1][j];
}
}

for(i=0;i<3;++i)
{
c[i][0]=c[i][0]%26;
printf("%c ",(char)(c[i][0]+65));
}

n=n-1; flag=flag+3;
}
}
```

**Output:**

```
C:\Users\admin\Desktop\jkkjh.exe

The cipher has a key of length 9. ie. a 3*3 matrix.
Enter the 9 character key. GYBNQKURP
6 24 1
13 16 10
20 17 15

Enter the length of string to be encoded(without spaces). 6
Enter the string. actCAt

The encoded cipher is : P O H F I N
-----
Process exited after 36.43 seconds with return value 2
Press any key to continue . . .
```

## **Practical 5**

❖ **Aim:** To implement Simple DES or AES.

**Program:**

```
#include<stdio.h>

int main()
{
    int i,
        cnt=0,
        p8[8]={6,7,8,9,1,2,3,4};
    int
p10[10]={6,7,8,9,10,1,2,3,4,5};

    char input[11], k1[10], k2[10], temp[11];
    char LS1[5], LS2[5];
    //k1, k2 are for storing interim keys
    //p8 and p10 are for storing permutation key

    //Read 10 bits from user...
    printf("Enter 10 bits input:");
    scanf("%s",input);  input[10]='\0';
```



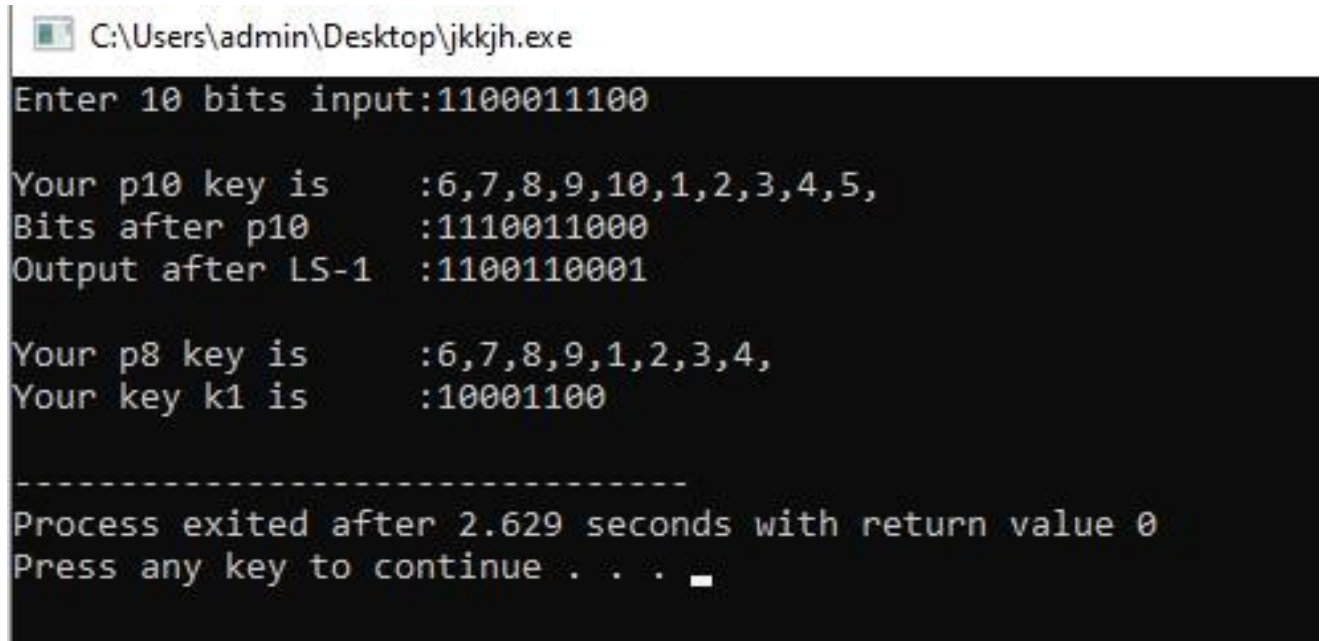
```
//Applying          p10...
for(i=0; i<10; i++)
{
    cnt = p10[i];
    temp[i] = input[cnt-1];
}
temp[i]='\0';
printf("\nYour p10 key is  :");
for(i=0; i<10; i++)
{
    printf("%d,",p10[i]);
}

printf("\nBits after p10  :");
puts(temp);
//Performing LS-1 on first half of temp
for(i=0; i<5; i++)
{
    if(i==4)
temp[i]=temp[0];
else
temp[i]=temp[i+1];
```

```
}  
//Performing LS-1 on second half of temp  
for(i=5; i<10; i++)  
{  
    if(i==9)  
temp[i]=temp[5];  
else  
temp[i]=temp[i+1];  
}  
printf("Output after LS-1 :");  
puts(temp);  
  
printf("\nYour p8 key is :");  
for(i=0; i<8; i++)  
{  
printf("%d,",p8[i]);  
}  
  
//Applying p8...  
for(i=0; i<8; i++)  
{
```

```
    cnt = p8[i]; k1[i] =  
temp[cnt-1];  
}  
printf("\nYour key k1 is    :");  
puts(k1);  
//This program can be extended to generate k2 as per DES algorithm.  
}
```

Output:



```
C:\Users\admin\Desktop\jkkjh.exe  
Enter 10 bits input:1100011100  
  
Your p10 key is      :6,7,8,9,10,1,2,3,4,5,  
Bits after p10      :1110011000  
Output after LS-1   :1100110001  
  
Your p8 key is      :6,7,8,9,1,2,3,4,  
Your key k1 is      :10001100  
  
-----  
Process exited after 2.629 seconds with return value 0  
Press any key to continue . . .
```

## **Practical 6**

❖ **Aim:** Implement Diffie-Hellman Key exchange Method.

**Program:**

```
/* This program calculates the Key for two persons using
the Diffie-Hellman Key exchange algorithm */
#include<stdio.h>
#include<math.h>

// Power function to return value of a ^ b mod P
long long int power(long long int a, long long int b,
                    long long int P)
{
    if (b == 1)
        return a;

    else
        return (((long long int)pow(a, b)) % P);
}

//Driver program
int main()
```

```
{
    long long int P, G, x, a, y, b, ka, kb;

    // Both the persons will be agreed upon the
        // public keys G and P
    P = 23; // A prime number P is taken
    printf("The value of P : %lld\n", P);

    G = 9; // A primitive root for P, G is taken
    printf("The value of G : %lld\n\n", G);

    // Alice will choose the private key a    a = 4; //
    a is the chosen private key  printf("The private key a
    for Alice : %lld\n", a);      x = power(G, a, P); //
    gets the generated key

    // Bob will choose the private key b    b = 3; //
    b is the chosen private key  printf("The private key b
    for Bob : %lld\n\n", b);     y = power(G, b, P); // gets
    the generated key

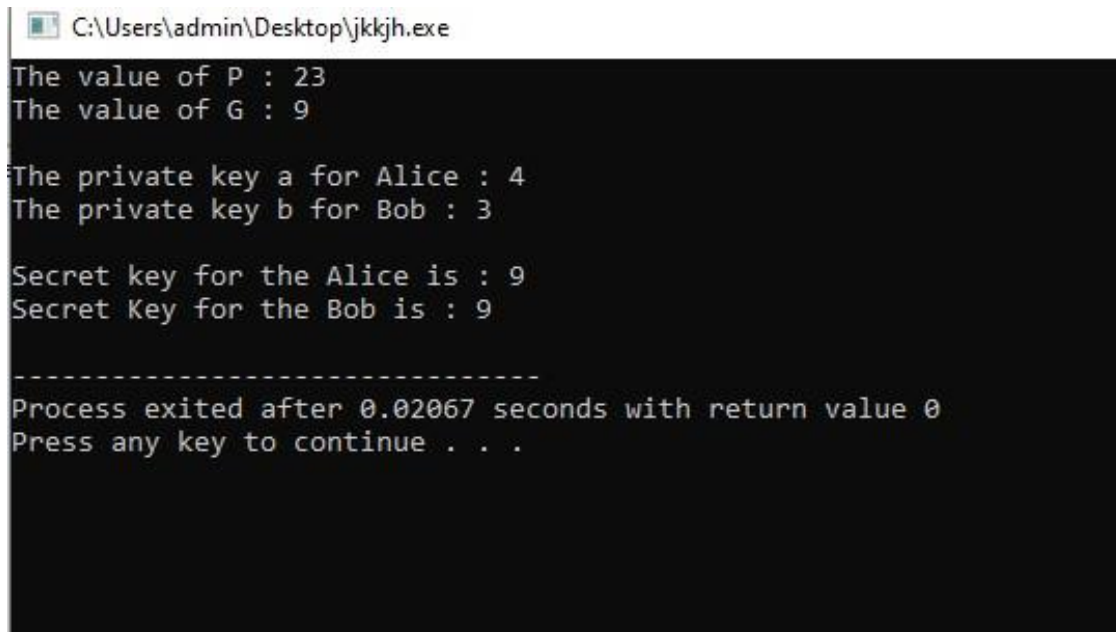
    // Generating the secret key after the exchange
        // of keys
    ka = power(y, a, P); // Secret key for Alice
```

```
kb = power(x, b, P); // Secret key for Bob

printf("Secret key for the Alice is : %lld\n", ka);
printf("Secret Key for the Bob is : %lld\n", kb);

return 0;
}
```

Output:



```
C:\Users\admin\Desktop\jkkjh.exe
The value of P : 23
The value of G : 9

The private key a for Alice : 4
The private key b for Bob : 3

Secret key for the Alice is : 9
Secret Key for the Bob is : 9

-----
Process exited after 0.02067 seconds with return value 0
Press any key to continue . . .
```

## Practical 7

❖ **Aim: Implement RSA encryption-decryption algorithm.**

**Program:**

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

//to find gcd int
gcd(int a, int h)
{
    int temp;
    while(1)
    {
        temp = a%h;
        if(temp==0)
            return h;
        a = h;
        h = temp;
    }
}

int main()
{
    //2 random prime numbers
    double p = 3;
    double q = 7;
    double n=p*q;
    double count;
    double totient = (p-1)*(q-1);

    //public key
    //e stands for encrypt
    double e=2;
```

```
//for checking co-prime which satisfies  $e > 1$ 
while( $e < \text{totient}$ )
{
    count = gcd(e,totient);
    if(count==1)
        break;
    else
        e++;
}

//private key
//d stands for decrypt
double d;

//k can be any arbitrary value
double k = 2;

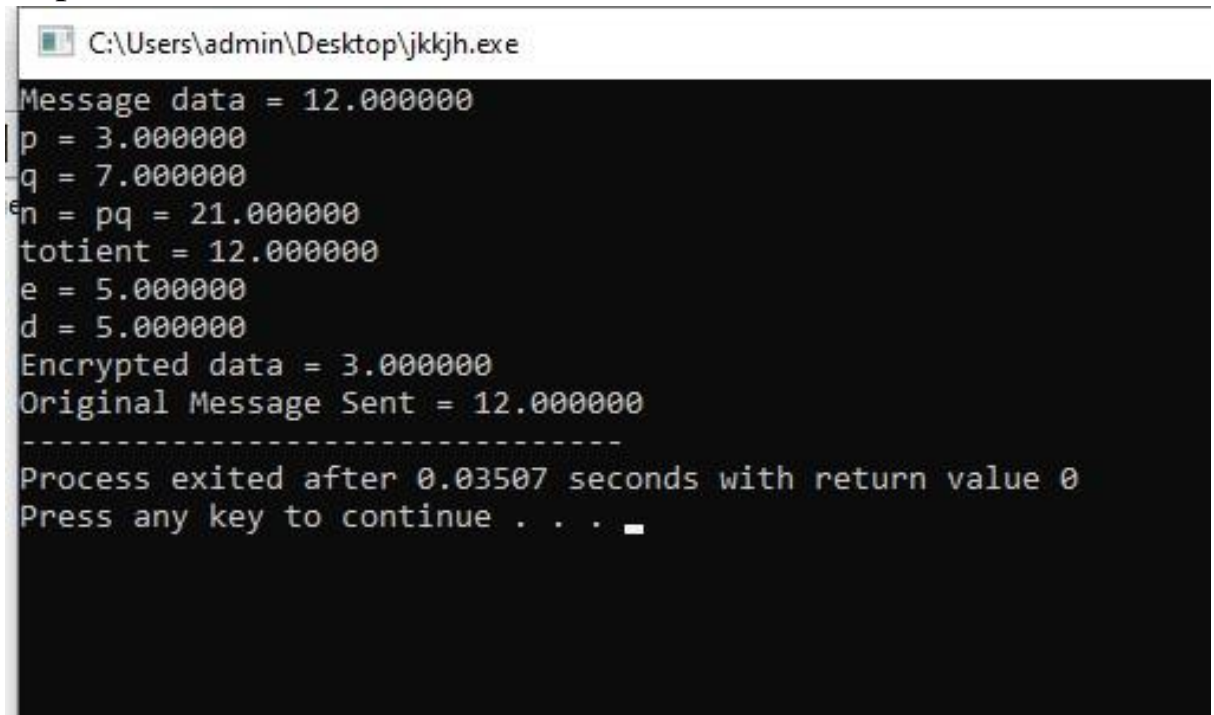
//choosing d such that it satisfies  $d * e = 1 + k * \text{totient}$ 
 $d = (1 + (k * \text{totient})) / e$ ;
double msg = 12;
double c = pow(msg,e);
double m = pow(c,d);
c=fmod(c,n);
m=fmod(m,n);

printf("Message data = %lf",msg);
printf("\np = %lf",p);
printf("\nq = %lf",q);
printf("\nn = pq = %lf",n);
printf("\ntotient = %lf",totient);
printf("\ne = %lf",e);
```



```
    printf("\nd = %lf",d);  
    printf("\nEncrypted data =%lf",c);  
    printf("\nOriginal Message Sent=  
%lf",m);  
  
    return 0;  
}
```

Output:



```
C:\Users\admin\Desktop\jkkjh.exe  
Message data = 12.000000  
p = 3.000000  
q = 7.000000  
n = pq = 21.000000  
totient = 12.000000  
e = 5.000000  
d = 5.000000  
Encrypted data = 3.000000  
Original Message Sent = 12.000000  
-----  
Process exited after 0.03507 seconds with return value 0  
Press any key to continue . . .
```

## Practical 8

❖ **Aim:** Implement a digital signature algorithm.

□ **Program:**

```
/* subject: Elliptic curve digital
```

```
signature algorithm,      toy version
```

```
for small modulus N.
```

```
tested : gcc 4.6.3, tcc 0.9.27
```

```
*/
```

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

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

```
#include <time.h>
```

```
// 64-bit integer type
```

```
typedef long long int
```

```
dlong; // rational ec
```

```
point typedef struct {
```

```
    dlong x, y;
```

```
} epnt;
```

```
//    elliptic    curve
```

```
parameters    typedef
```

```
struct
```

```
{
```

```
    long a, b;
```

```
    dlong N;
```

```
epnt G;
    dlong r;
} curve; //
signature
pair typedef
struct
{
    long a, b;
} pair;

// dlong for holding intermediate results,
// long variables in exgcd() for
efficiency, // maximum parameter
size 2 * p.y (line 129)
// limits the modulus size to 30 bits.

// maximum modulus
const long mxN =
1073741789; // max
order G = mxN + 65536
const long mxr =
1073807325; //
symbolic infinity const
long inf = -2147483647;
```

```
// single global curve
curve e; // point at
infinity zero epnt
zero; // impossible
inverse mod N int
inverr;
```

```
// return mod( $v^{-1}$ , u)
long exgcd (long v,
long u)
{
register long
q, t;
long r = 0,
s=1;
if (v < 0)
v += u;
while (v) {
q = u / v;
t = u - q * v;
u = v;
v = t;
```

```
t = r - q * s;
r = s;
s = t;
}
if (u != 1)
{
    printf (" impossible inverse mod N, gcd = %d\n", u);
    inverr = 1;
}
return r;
}
```

```
// return mod(a, N) static
inline dlong modn (dlong
a)
{
    a %= e.N;
    if (a < 0) a +=
e.N; return a;
}
```

```
// return mod(a, r)
dlong modr
(dlonga)
```

```
{  
    a %= e.r;  
    if (a < 0) a +=  
    e.r; return a;  
}
```

```
// return the discriminant of  
E long disc (void)  
{  
    dlong c, a = e.a, b = e.b;  
    c = 4 * modn(a * modn(a * a));  
    return modn(-16 * (c + 27 *  
    modn(b * b)));  
}
```

```
// return 1 if P =  
zerO int isO (epnt  
p)  
{  
    return (p.x == inf) && (p.y == 0);  
}
```

```

// return 1 if P is on
curve E int ison (epnt p)
{ long r, s; if (! isO (p)) {   r =
modn(e.b + p.x * modn(e.a + p.x *
p.x));   s = modn(p.y * p.y);
}
return (r == s);
}

```

```

// full ec point addition
void padd (epnt *r, epnt p, epnt q)
{
dlong la, t;

if (isO(p)) { *r = q;
return;} if (isO(q))
{ *r = p; return;}

```

```

if (p.x != q.x) {           //
R:= P + Q   t = p.y - q.y;
    la = modn(t * exgcd(p.x - q.x, e.N));
} else           // P
= Q, R := 2P

```

```

    if ((p.y == q.y) && (p.y !=
0)) {      t = modn(3 *
modn(p.x * p.x) + e.a);      la
= modn(t * exgcd (2 * p.y,
e.N));
    }
    else
        { *r = zerO; return;}      // P = -Q, R := O

```

```

t = modn(la * la - p.x -
q.x); r->y = modn(la *
(p.x - t) - p.y); r->x = t;
if (inverr) *r = zerO;
}

```

```

// R:= multiple kP void
pmul (epnt *r, epnt p, long
k)
{
    epnt s = zerO, q = p;

```

```

    for (; k; k >>= 1) {
    if (k & 1) padd(&s, s,
q);      if (inverr) {s =

```



```
zerO; break;}
```

```
padd(&q, q, q);
```

```
}
```

```
*r = s;
```

```
}
```

```
// print point P with
```

```
prefix f void pprint (char
```

```
*f, epnt p)
```

```
{
```

```
  dlong y = p.y;
```

```
    if (isO (p))
```

```
      printf("%s(0)\n",f)
```

```
;
```

```
    else {
```

```
      if (y > e.N - y) y -= e.N;
```

```
      printf ("%s (%lld, %lld)\n", f,
```

```
      p.x, y);
```

```
    }
```

```
}
```

```
// initialize elliptic
curve int ellinit (long
i[])
{
long a = i[0], b = i[1];
    e.N = i[2]; inverr = 0;

if ((e.N < 5) || (e.N > mxN)) return 0;

    e.a = modn(a);
    e.b = modn(b);
    e.G.x = modn(i[3]);
    e.G.y = modn(i[4]);
    e.r = i[5];

if ((e.r < 5) || (e.r > mxr)) return 0;

printf ("\nE: y^2 = x^3 + %dx +
%d", a, b); printf (" (mod
%lld)\n", e.N); pprint ("base
point G", e.G); printf ("order(G,
E) = %lld\n", e.r);

return 1;
}
```

```
// pseudorandom number [0..1)
double rnd(void)
{
return rand() / ((double)RAND_MAX + 1);
}
```

```
// signature primitive
pair signature (dlong s,
long f)
{
long c, d, u, u1;
pair sg;

epnt V;
```

```
printf ("\nsignature
computation\n"); do { do {
u = 1 + (long)(rnd() * (e.r -
1));    pmul (&V, e.G, u);
c = modr(V.x);
}
while (c == 0);
```

```

    u1 = exgcd (u, e.r);  d =
modr(u1 * (f + modr(s *
c)));
}
while (d == 0); printf
("one-time u = %d\n",
u); pprint ("V = uG", V);

```

```

sg.a = c; sg.b = d;
return sg;
}

```

```

// verification primitive
int verify (epnt W, long f,
pair sg)
{
long c = sg.a, d =
sg.b; long t, c1,
h1, h2; dlong h;
epnt V, V2;

```

```

    // domain check
t = (c > 0) && (c <
e.r);  t &= (d > 0)

```

```
&& (d < e.r);  if (!
t) return 0;

printf("\nsignatureverificatio
n\n");

h = exgcd (d, e.r);  h1 =
modr(f * h);

h2 = modr(c * h);

printf ("h1,h2 = %d, %d\n",
h1,h2);

pmul (&V, e.G, h1);
pmul (&V2, W, h2);
pprint ("h1G", V);
pprint ("h2W", V2);
padd (&V, V, V2);
pprint ("+=" , V);
if (isO (V))
return 0;

c1 = modr(V.x);
printf ("c' = %d\n", c1);

return (c1 == c);
}
```

```
// digital signature on message hash f,  
error bit d void ec_dsa (long f, long d)  
{ long i,  
s, t; pair  
sg; epnt  
W;
```

```
    // parameter check  
    t = (disc() ==  
0);  
    t |= isO (e.G);  
    pmul (&W, e.G,  
e.r);  
    t |= ! isO (W);  
    t |= ! ison (e.G);  
  
    if (t) goto  
errmsg;
```

```
printf ("\nkey generation\n");  
= 1 + (long)(rnd() * (e.r -  
1));
```

```
    pmul (&W, e.G, s);
printf ("private key s =
%d\n", s);

    pprint ("public key W =
sG", W);

    // next highest power of 2 - 1
    = e.r;
    for (i = 1; i < 32; i <= 1)
    t |= t >> i;
    while (f > t)
    f >>= 1;
    printf ("\naligned hash
%x\n", f);

    sg = signature (s, f);
    if (inverr)
goto
errmsg;
printf ("signature c,d = %d, %d\n",
sg.a, sg.b);

    if (d > 0) {
while (d > t) d
```

```
>>= 1;    f ^= d;
```

```
printf
```

```
("\\ncorrupted hash
```

```
%x\\n", f);
```

```
}
```

```
t = verify (W, f,
```

```
sg);
```

```
if (inverr)
```

```
goto errmsg;
```

```
if (t)
```

```
printf
```

```
("Valid\\n____\\n");
```

```
else
```

```
printf ("invalid\\n____\\n");
```

```
return;
```

```
errmsg:
```

```
printf ("invalid parameter
```

```
set\\n");
```

```
printf
```

```
("_____\\n")
```

```
;
```



```
}
```

```
void main (void)
```

```
{
```

```
typedef long eparm[6];
```

```
long d, f;
```

```
    zerO.x = inf;
```

```
    zerO.y = 0;
```

```
    srand(time(NULL
```

```
));
```

```
// Test vectors: elliptic curve domain
```

```
parameters, // short Weierstrass model  $y^2$ 
```

```
=  $x^3 + ax + b \pmod{N}$  eparm *sp,
```

```
sets[10] = {
```

```
//  a,  b, modulus N, base point G, order(G, E), cofactor
```

```
    {355, 671, 1073741789, 13693, 10088, 1073807281},
```

```
    { 0,  7, 67096021, 6580,  779, 16769911}, // 4
```

```
    {-3,  1, 877073,  0,  1, 878159},
```

```
    { 0, 14, 22651, 63, 30, 151}, // 151
```

```
    { 3, 2, 5, 2, 1, 5},
```

```
// ecdsa may fail if...
```

```

// the base point is of composite order
{ 0, 7, 67096021, 2402, 6067, 33539822}, // 2
// the given order is a multiple of the true order
{ 0, 7, 67096021, 6580, 779, 67079644}, // 1
// the modulus is not prime (deceptive example)
{ 0, 7, 877069, 3, 97123, 877069},
// fails if the modulus divides the discriminant
{ 39, 387, 22651, 95, 27, 22651},
};


// Digital signature on message
hash f, // set d > 0 to simulate
corrupted data f =
0x789abcde; d = 0;

for (sp = sets; ;
sp++)
{
    if (ellinit (*sp))
ec_dsa (f, d);

    else
break;
}
}

```

## Output:

 C:\Users\admin\Desktop\jkkjh.exe

```
E:  $y^2 = x^3 + 355x + 671 \pmod{1073741789}$ 
base point G (13693, 10088)
order(G, E) = 1073807281

key generation
private key s = 496957013
public key W = sG (44156345, -282189600)

aligned hash 789abcde

signature computation
one-time u = 596971046
V = uG (449125641, 275204213)
signature c,d = 449125641, 266412353

signature verification
h1,h2 = 517873459, 155779364
h1G (686326399, 491420973)
h2W (995403469, -242746724)
+ = (449125641, 275204213)
c' = 449125641
Valid
```

---

```
E:  $y^2 = x^3 + -3x + 1 \pmod{877073}$ 
base point G (0, 1)
order(G, E) = 878159
```

```
key generation
private key s = 135685
public key W = sG (552318, -281470)
```

```
aligned hash f1357
```

```
signature computation
one-time u = 448727
V = uG (725398, 372037)
signature c,d = 725398, 791814
```

```
signature verification
h1,h2 = 208674, 286765
h1G (759869, 124084)
h2W (498749, -138392)
+ = (725398, 372037)
c' = 725398
Valid
```

---

```
E:  $y^2 = x^3 + 0x + 14 \pmod{22651}$ 
base point G (63, 30)
order(G, E) = 151
```

```
key generation
private key s = 133
public key W = sG (17112, 197)
```

```
aligned hash f1
```

```
signature computation
one-time u = 57
V = uG (12480, -505)
signature c,d = 98, 66
```

```
signature verification
h1,h2 = 70, 93
h1G (15255, 7228)
h2W (12611, -3365)
+ = (12480, -505)
c' = 98
Valid
```

---

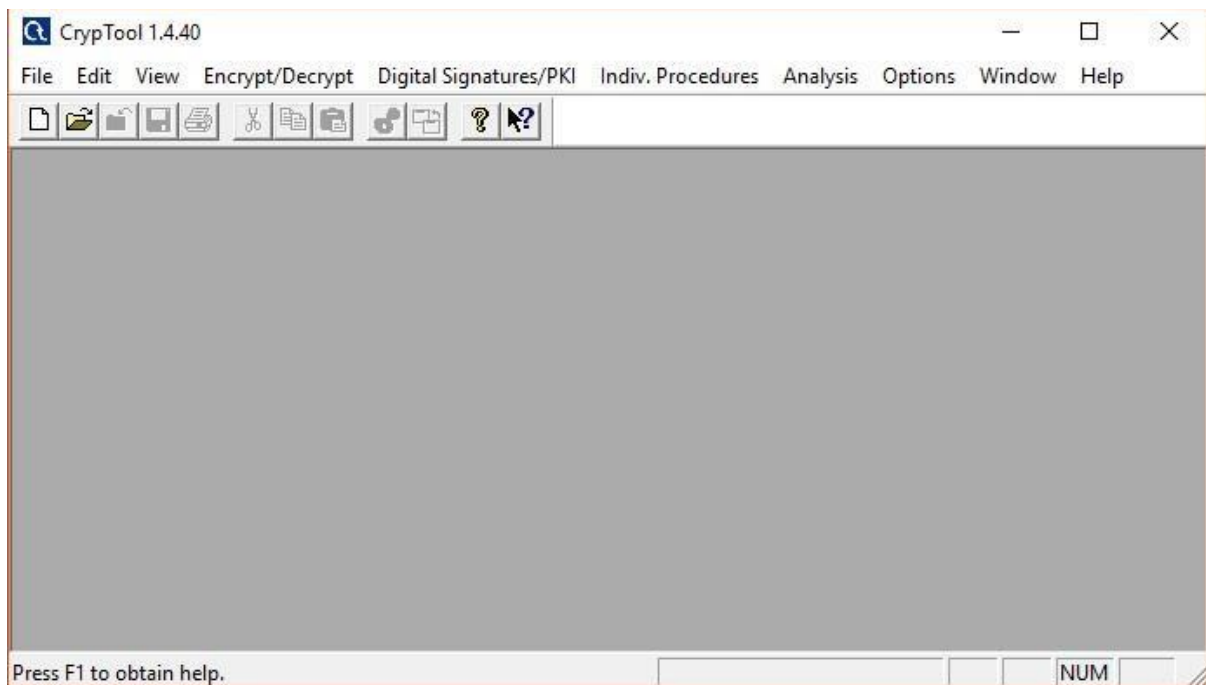
## Practical 9



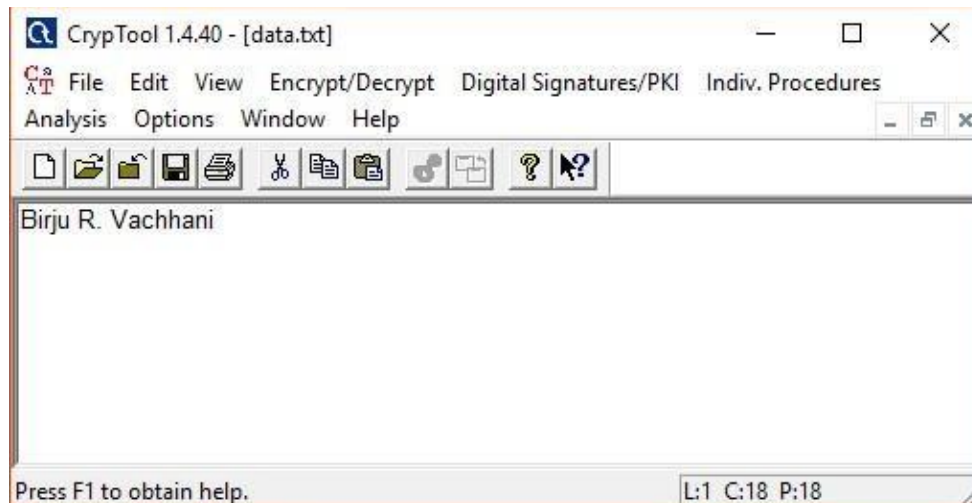
**Aim:** Write a program to generate SHA-1 hash.

**Steps:**

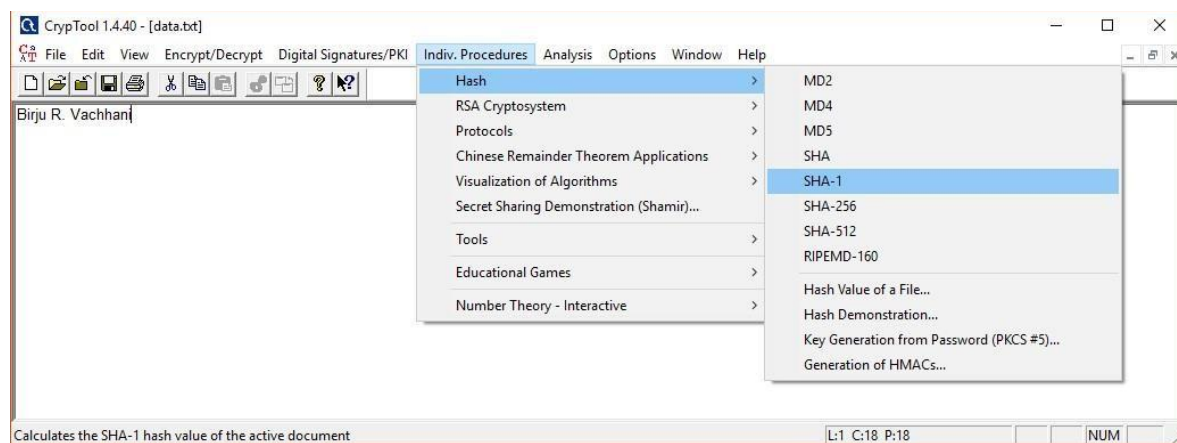
- i. Open Cyptool.



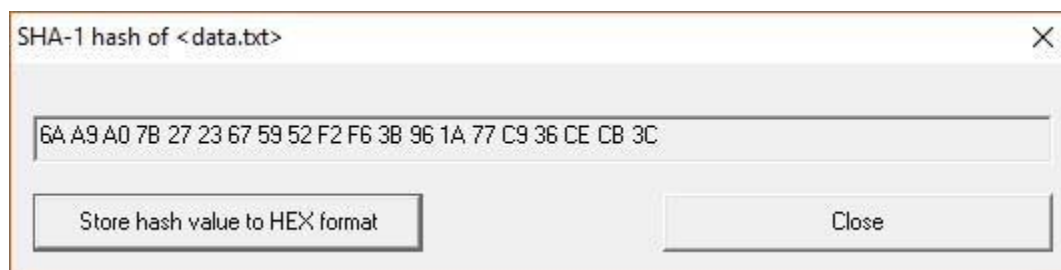
- ii. Go to File>Open and open a txt file you want to generate hash. After opening the file, the content of file will be displayed in cryptool window as shown in figure below.



- iii. Now, Go to Indiv. Procedures>Hash and select “SHA-1” as shown in figure below.



- iv. After selecting SHA-1, a window will be popped which will display the generated hash for the given file as shown in figure below.



## Practical 10

❖ **Aim:** Study and use the Wireshark for the various network protocols.

### **Description:**

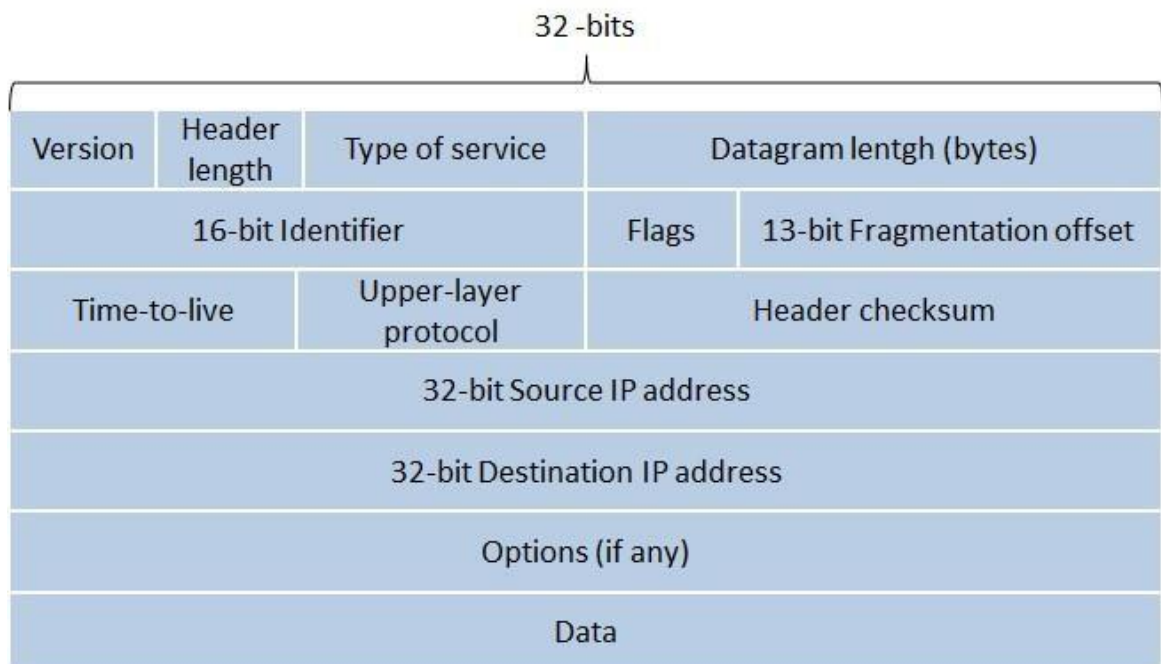
Wireshark is a free and open source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education. Originally named Ethereal, the project was renamed Wireshark in May 2006 due to trademark issues. Wireshark is cross-platform, using the Qt widget toolkit in current releases to implement its user interface, and using pcap to capture packets; it runs on Linux, macOS, BSD, Solaris, some other Unix-like operating systems, and Microsoft Windows.

There is also a terminal-based (non-GUI) version called TShark. Wireshark, and the other programs distributed with it such as TShark, are free software, released under the terms of the GNU General Public License.

### **STUDING TCP/UDP USING WIRESHARK**

#### **□ Internet Protocol**

The Internet Protocol (IP) is the method or protocol by which data is sent from one computer to another on the Internet. Each computer (known as a host) on the Internet has at least one IP address that uniquely identifies it from all other computers on the Internet.



Packet Analysing:

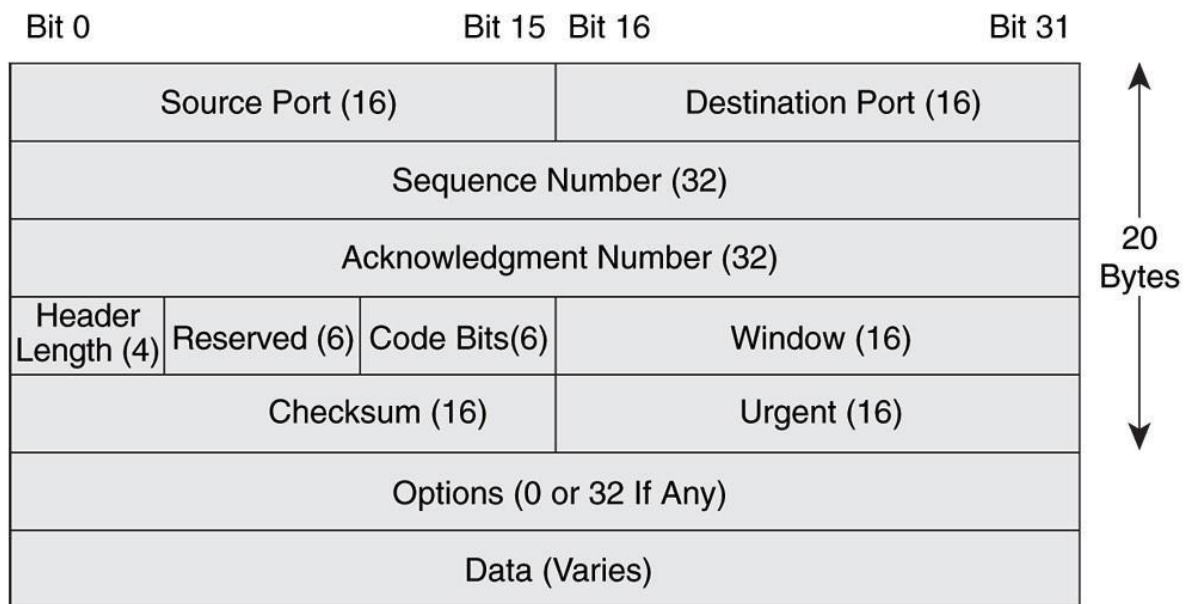
```

+ Frame 580: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
+ Ethernet II, Src: HewlettP_90:42:d1 (10:60:4b:90:42:d1), Dst: D-LinkIn_6f:b0:3e (cc:b2:55:6f:b0:3e)
+ Internet Protocol Version 4, Src: 10.0.62.144 (10.0.62.144), Dst: c-0001.c-msedge.net (13.107.4.50)
  Version: 4
  Header length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
  Total Length: 40
  Identification: 0x0727 (1831)
  Flags: 0x02 (Don't Fragment)
  Fragment offset: 0
  Time to live: 128
  Protocol: TCP (6)
  Header checksum: 0x997c [correct]
  Source: 10.0.62.144 (10.0.62.144)
  Destination: c-0001.c-msedge.net (13.107.4.50)
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
+ Transmission Control Protocol, Src Port: menandmice-lpm (1231), Dst Port: http (80), Seq: 266, Ack: 274, L
0000  cc b2 55 6f b0 3e 10 60 4b 90 42 d1 08 00 45 00  ..Uo.>.`K.B...E.
0010  00 28 07 27 40 00 80 06 99 7c 0a 00 3e 90 0d 6b  .(.@...|...>..k
0020  04 32 04 cf 00 50 d8 0c 4c 4a 3e a0 60 2a 50 10  .2...P..LJ>.*P.
0030  00 00 8d 67 00 00  ...g..

```

### TCP Frame format:

The Transmission Control Protocol (**TCP**) is a core protocol of the Internet protocol suite. It originated in the initial network implementation in which it complemented the Internet Protocol (IP). Therefore, the entire suite is commonly referred to as **TCP/IP**.



❖ Packet Analyzing:



```

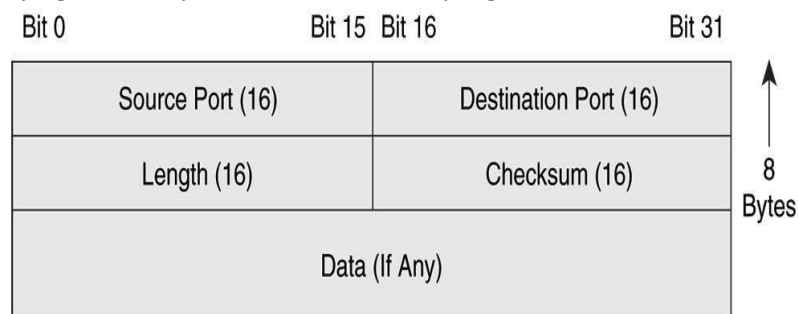
Transmission Control Protocol, Src Port: 443 (443), Dst Port: 49815 (49815), Seq: 1, Ack: 2, Len: 0
  Source Port: 443
  Destination Port: 49815
  [Stream index: 45]
  [TCP Segment Len: 0]
  Sequence number: 1 (relative sequence number)
  Acknowledgment number: 2 (relative ack number)
  Header Length: 32 bytes
  > Flags: 0x010 (ACK)
  Window size value: 988
  [Calculated window size: 988]
  [Window size scaling factor: -1 (unknown)]
  > Checksum: 0x0110 [validation disabled]
  Urgent pointer: 0
  > Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), SACK
  > [SEQ/ACK analysis]

```

|      |   |                    |
|------|---|--------------------|
| 0000 | 48 d2 24 63 0b bf 30 b5 c2 90 a9 40 08 00 45 00 | H.\$c..0. ...@..E. |
| 0010 | 00 34 3e 2b 40 00 39 06 e7 9c 17 34 43 b8 c0 a8 | .4>+@.9. ...4C...  |
| 0020 | 00 68 01 b6 c2 97 01 70 6e 3e 91 e8 cf b3 80 10 | .h...p n>.....     |
| 0030 | 03 dc 01 10 00 00 01 01 05 0a 91 e8 cf b2 91 e8 | .....              |
| 0040 | cf b3 4d 7c                                     | ..M                |

### UDP frame format:

UDP uses a simple connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes any unreliability of the underlying network protocol to the user's program.



### No Sequence Or Acknowledgment Fields

```

Protocol: UDP (17)
  Header checksum: 0x5207 [correct]
  Source: ubuntu-25.local (10.0.62.92)
  Destination: 224.0.0.251 (224.0.0.251)
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
User Datagram Protocol, Src Port: mdns (5353), Dst Port: mdns (5353)
Domain Name System (query)

```

|      |   |                   |
|------|---|-------------------|
| 0020 | 00 fb 14 e9 14 e9 00 7a 68 eb 00 00 00 00 00 01 | ...z h.....       |
| 0030 | 00 00 00 02 00 00 1d 75 62 75 6e 74 75 2d 32 35 | .....u buntu-25   |
| 0040 | 20 5b 31 30 3a 36 30 3a 34 62 3a 39 30 3a 34 34 | [10:60: 4b:90:44  |
| 0050 | 3a 31 35 5d 0c 5f 77 6f 72 6b 73 74 61 74 69 6f | :15]._wo rkstatio |
| 0060 | 6e 04 5f 74 63 70 05 6c 6f 63 61 6c 00 00 ff 00 | n tcp l ocal      |

Packet Analyzing:

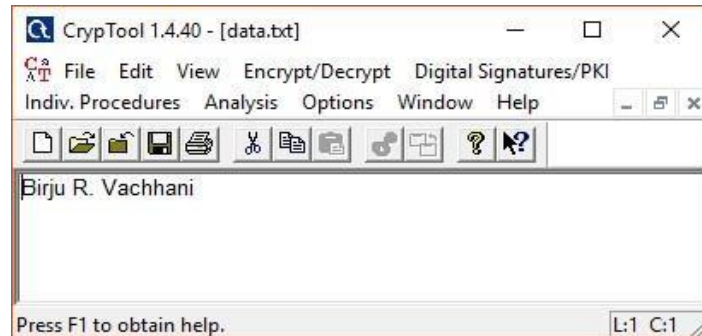
### Address Resolution Protocol (ARP)

Address Resolution Protocol (ARP) is one of the major protocol in the TCP/IP suit and the purpose of Address Resolution Protocol (ARP) is to resolve an IPv4 address (32 bit Logical

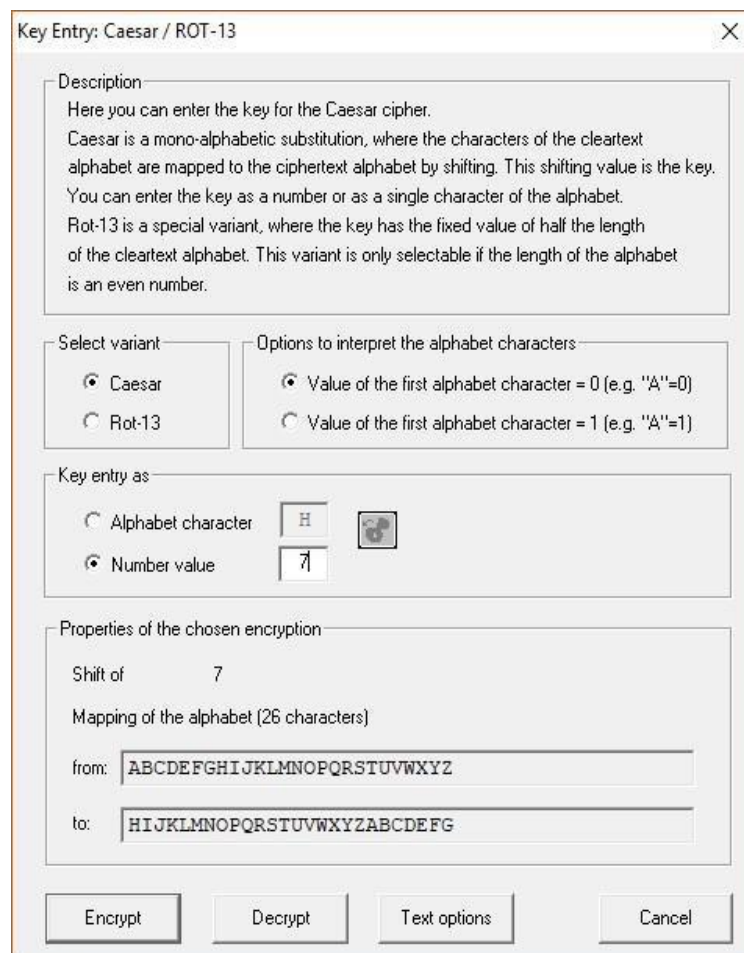


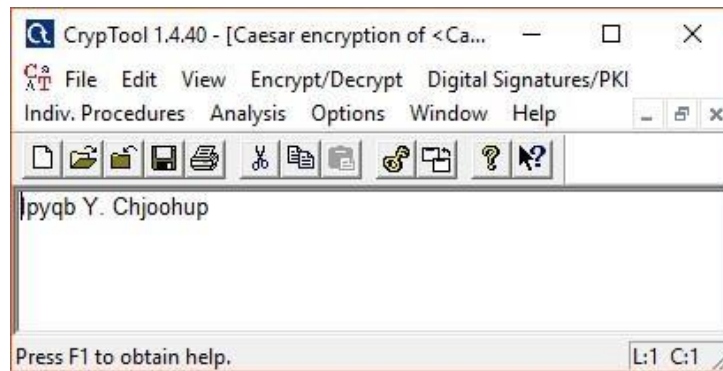
## Practical 11

- ❖ **Aim:** Perform various encryption-decryption techniques with cryptool.

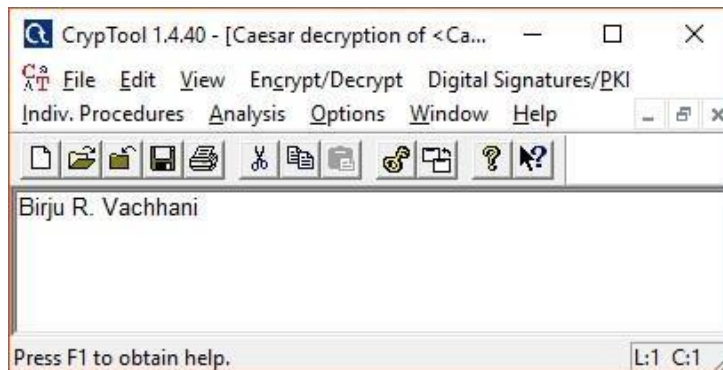


### 1. Caesar cipher



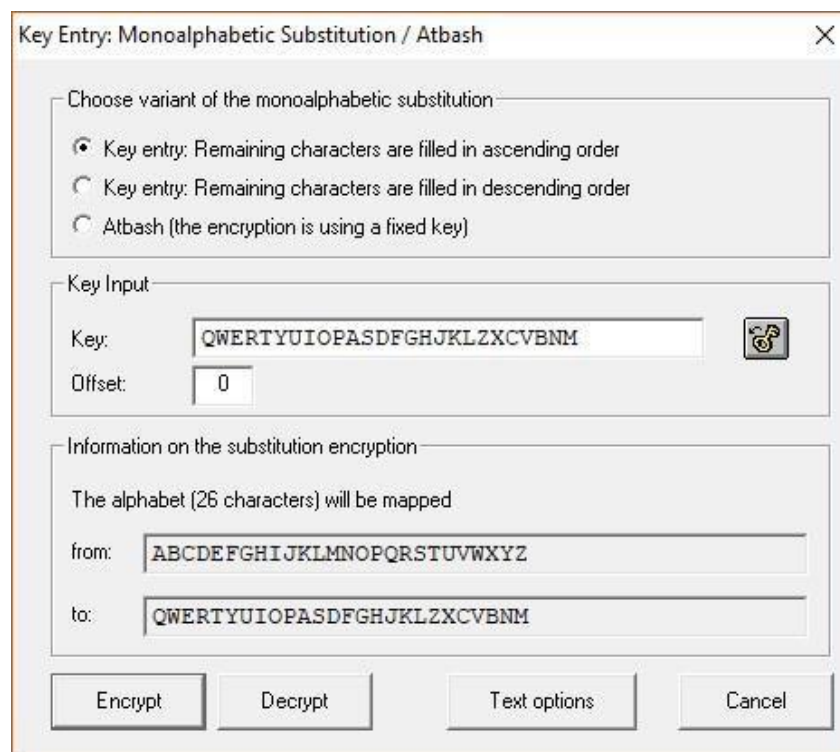


## Encryption



## Decryption

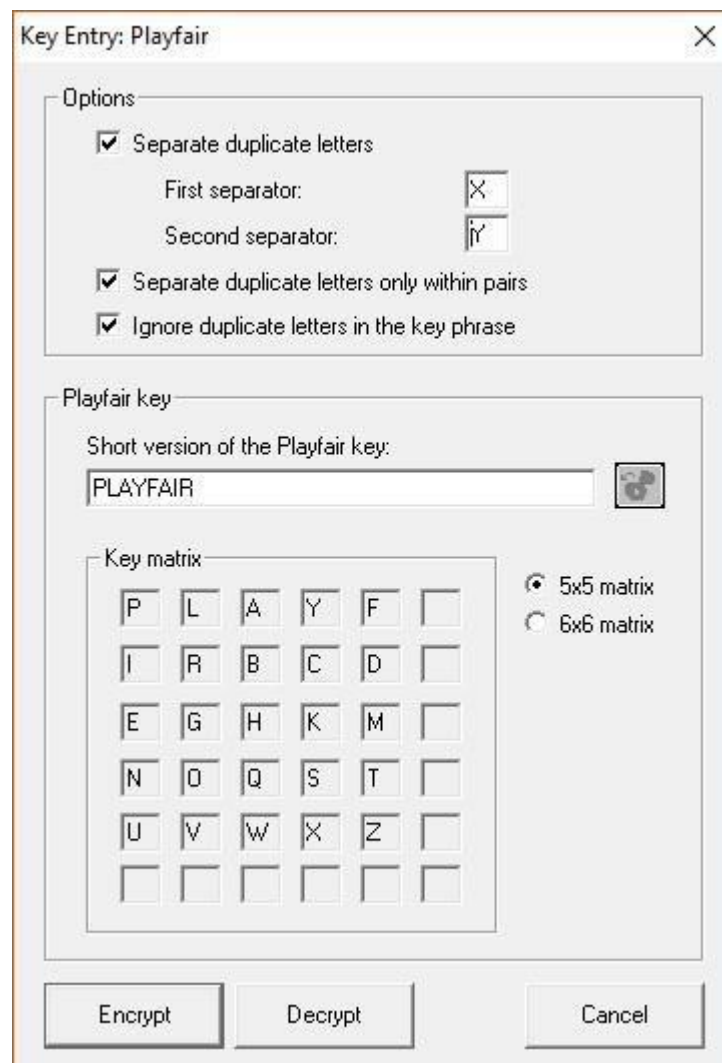
## 2. Monoalphabetic cipher





## Encryption

### 3. Playfair cipher







## Encryption

### 4. Hill cipher

**Key Entry: Hill**

**Description**

The Hill cipher is a polygraphic substitution cipher based on linear algebra. This was the first polygraphic cipher in which it was practical to operate on groups of more than three letters (blocks) at once. The key is a quadratic matrix. Its dimension is the length of the group of letters.

**Selected alphabet (26 characters)**

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Value of the first alphabet character: 0

**Hill key matrix**

☐ Alphabet characters

☒ Number values

**Alphabet characters**

|   |   |   |  |  |
|---|---|---|--|--|
| B | A | C |  |  |
| K | U | P |  |  |
| A | B | C |  |  |
|   |   |   |  |  |
|   |   |   |  |  |

**Number values**

|    |    |    |  |  |
|----|----|----|--|--|
| 01 | 00 | 02 |  |  |
| 10 | 20 | 15 |  |  |
| 00 | 01 | 02 |  |  |
|    |    |    |  |  |
|    |    |    |  |  |

**Multiplication variant**

☐ (row vector) \* (matrix)

☒ (matrix) \* (column vector)

**Size of matrix**

☐ 1 x 1

☐ 2 x 2

☒ 3 x 3

☐ 4 x 4

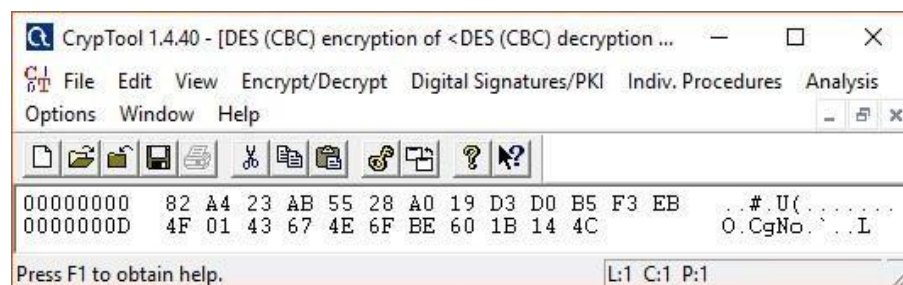
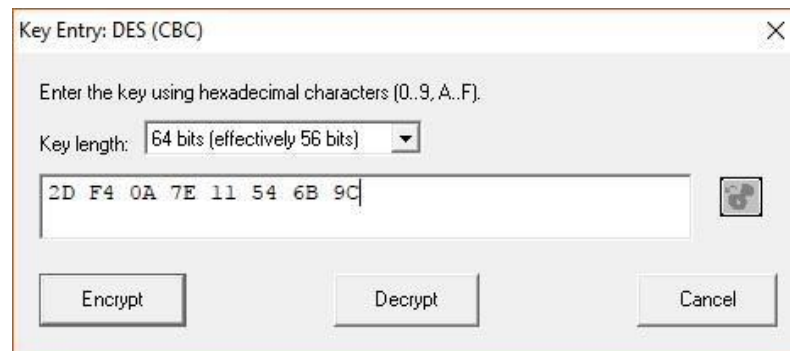
☐ 5 x 5

☐ Show details and single steps of the Hill cipher



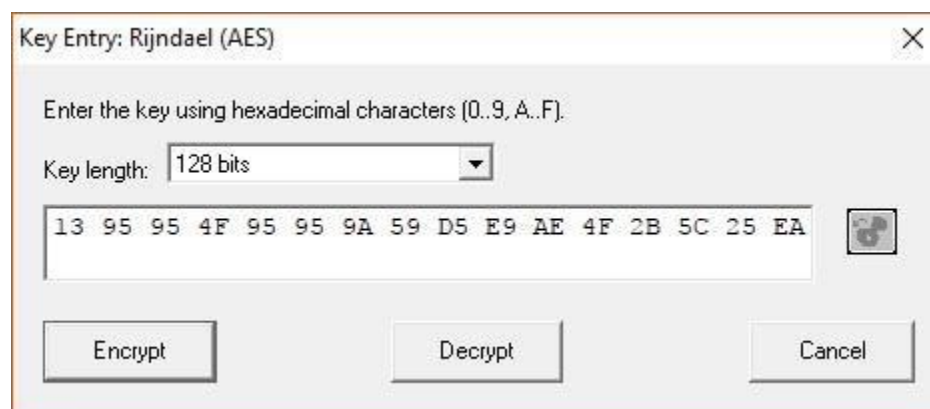
## Encryption

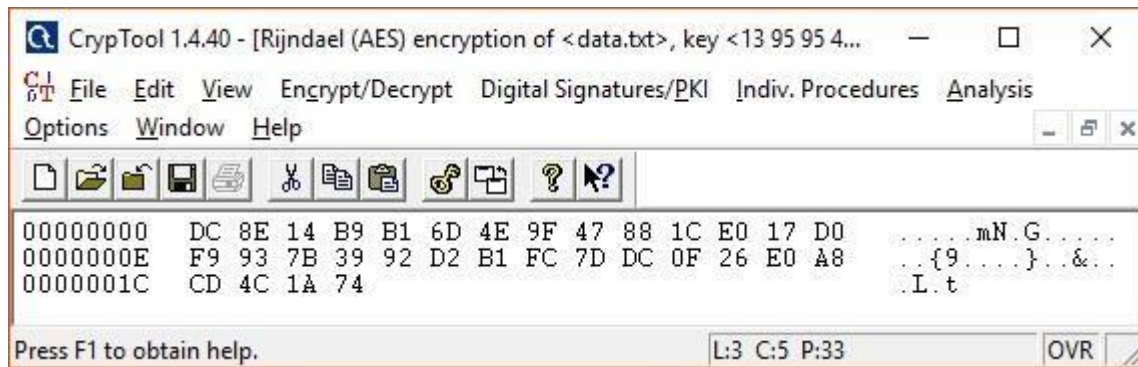
### 5. DES



## Encryption

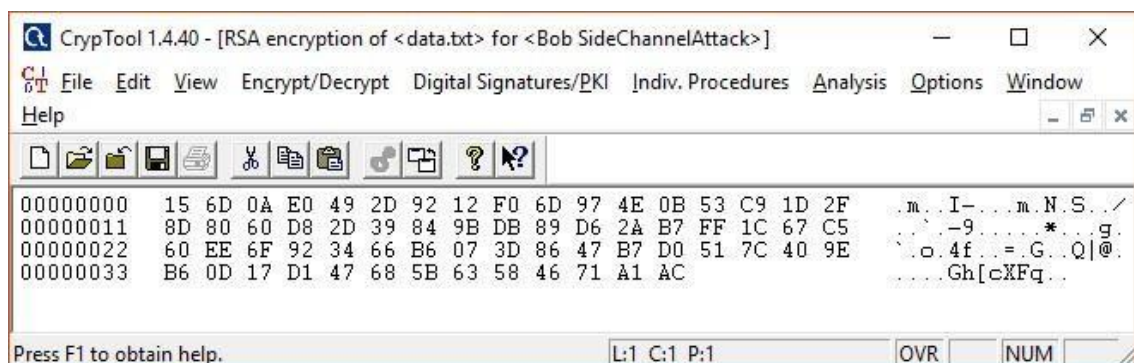
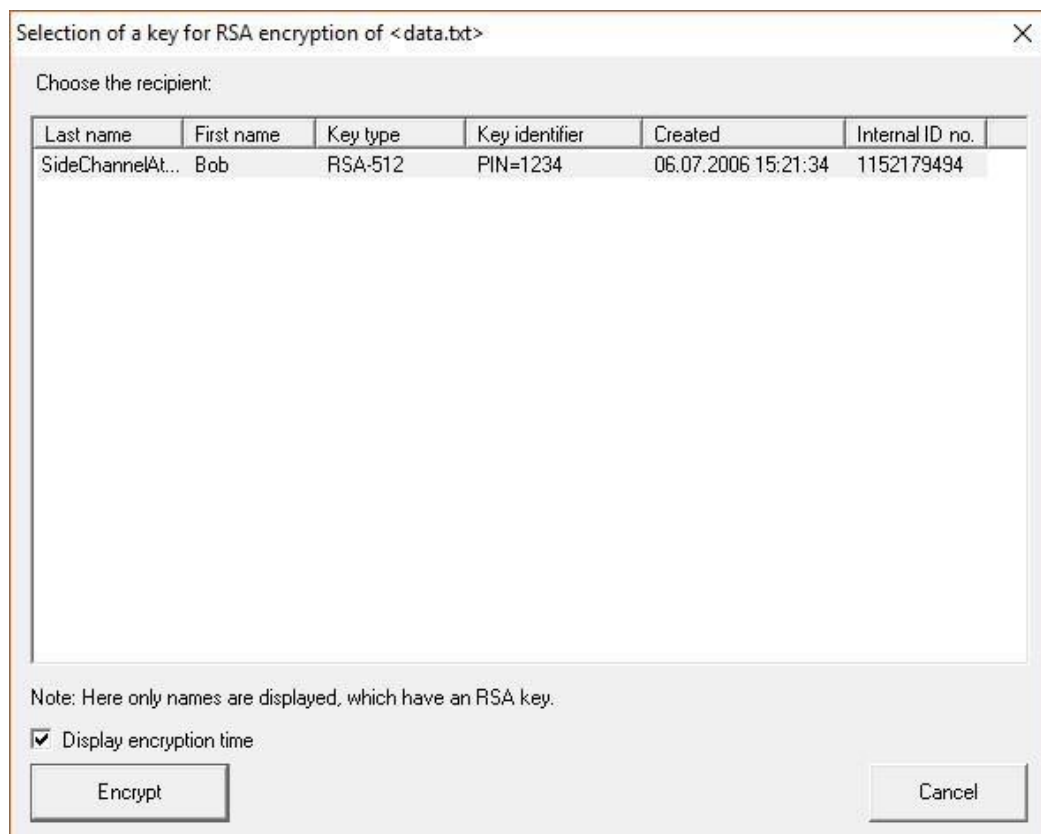
### 6. AES





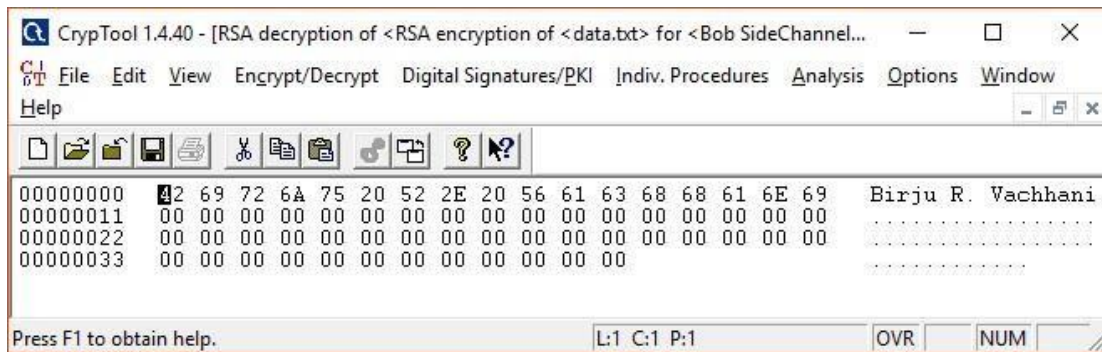
## Encryption

### 7. RSA



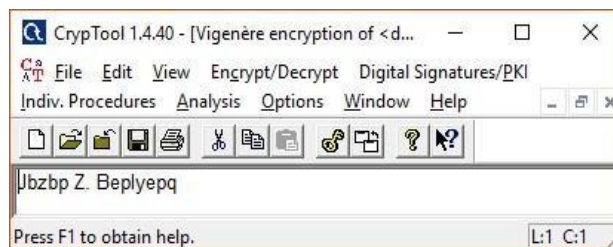
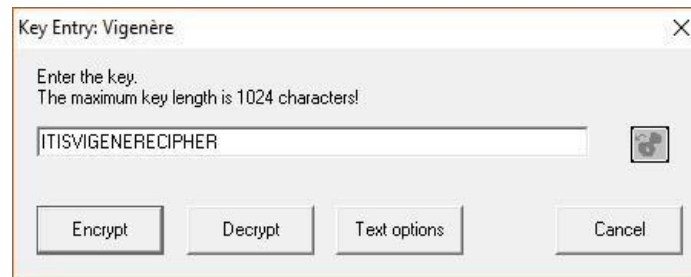


## Encryption



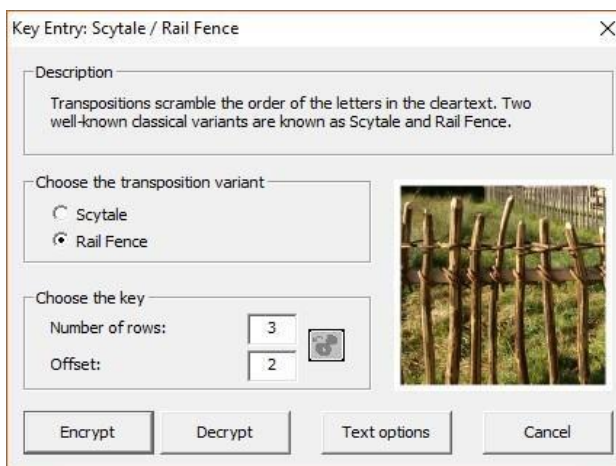
## Decryption

### 8. Vigenere cipher



## Encryption

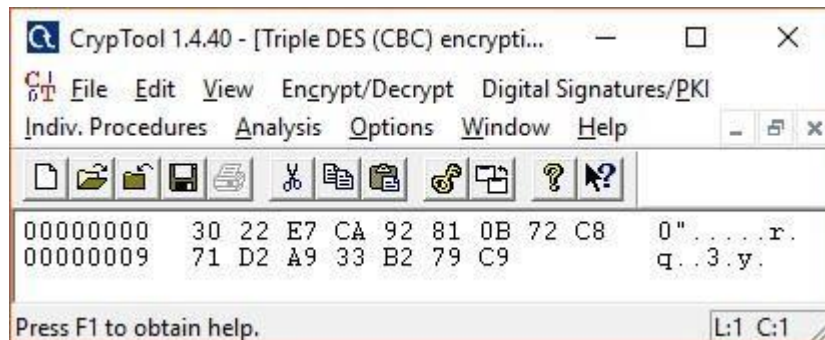
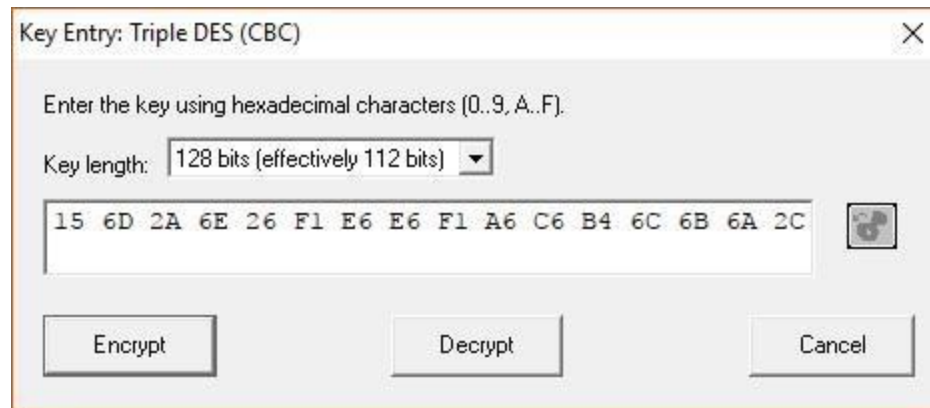
### 9. Rail Fence cipher





## Encryption

### 10. Triple DES



## Encryption