LABORATORY MANUAL

INFORMATION AND NETWORK SECURITY

SUBJECT CODE: 2170709

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

B.E. 7th SEMESTER

NAME:____

ENROLLMENT NO: ________________________________

BATCH NO: ____________________________________

YEAR: ______________________________________

Amiraj College of Engineering and Technology,

Nr.Tata Nano Plant, Khoraj, Sanand, Ahmedabad.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Amiraj College of Engineering and Technology,

Nr.Tata Nano Plant, Khoraj, Sanand, Ahmedabad.

CERTIFICATE

This is to certify that Mr. / Ms. ____

Of class____________________ Enrolment No ___________________________has

Satisfactorily completed the course in ____________________________________as

by the Gujarat Technological University for ____ Year (B.E.) semester___ of Computer Science and Engineering in the Academic year________.

Date of Submission:-

Faculty Name and Signature

Paras Narkhede

Head of Department

Foram Patel

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

B.E. 7th SEMESTER

SUBJECT: INOFRMATION AND NETWORK SECURITY

SUBJECT CODE: 2170709

List Of Experiments

Sr. No.	Experiments	Date of Performance	Date of submission	Sign	Remarks
1	Implement Caesar cipher encryption-decryption.				
2	Implement Playfair cipher encryption-decryption.				
3	Implement Polyalphabetic cipher encryption-decryption.				
4	Implement Hill cipher encryption-decryption.				
5	Implementation of Transposition cipher.				

6	To implement Simple DES or AES.				
7	Implement Diffi-Hellmen Key exchange Method.				
8	Implement RSA encryption-decryption algorithm.				
9	Perform various encryption-decryption techniques with cryptool.				
10	Study and use the Wireshark for the various network protocols.				

PRACTICAL- 1

OBJECTIVE:
Implement CAESER cipher encryption- decryption

THEORY:
The Caesar cipher is one of the earliest known and simplest ciphers. It is a type of substitution cipher in which each letter in the plaintext is 'shifted' a certain number of places down the alphabet. For example, with a shift of 1, A would be replaced by B, B would become C, and so on. The method is named after Julius Caesar, who apparently used it to communicate with his generals. The Caesar cipher involves replacing each letter of the alphabet with the letter standing three places further down the alphabet.

For example,

```
plain: meet me after the toga party
cipher: PHHW PH DIWHU WKH WRJD SDUWB
```

Note that the alphabet is wrapped around, so that the letter following Z is A. We can define the transformation by listing all possibilities, as follows:

```
plain: a b c d e f g h i j k l m n o p q r s t u v w x y z
cipher: D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
```

Let us assign a numerical equivalent to each letter:

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12

n	o	p	q	r	s	t	u	v	w	x	y	z
13	14	15	16	17	18	19	20	21	22	23	24	25

Then the algorithm can be expressed as follows. For each plaintext letter, substitute the cipher text
letter C_2:

$$C = E(3, p) = (p + 3) \bmod 26$$

A shift may be of any amount, so that the general Caesar algorithm is

$$C = E(k, p) = (p + k) \bmod 26$$

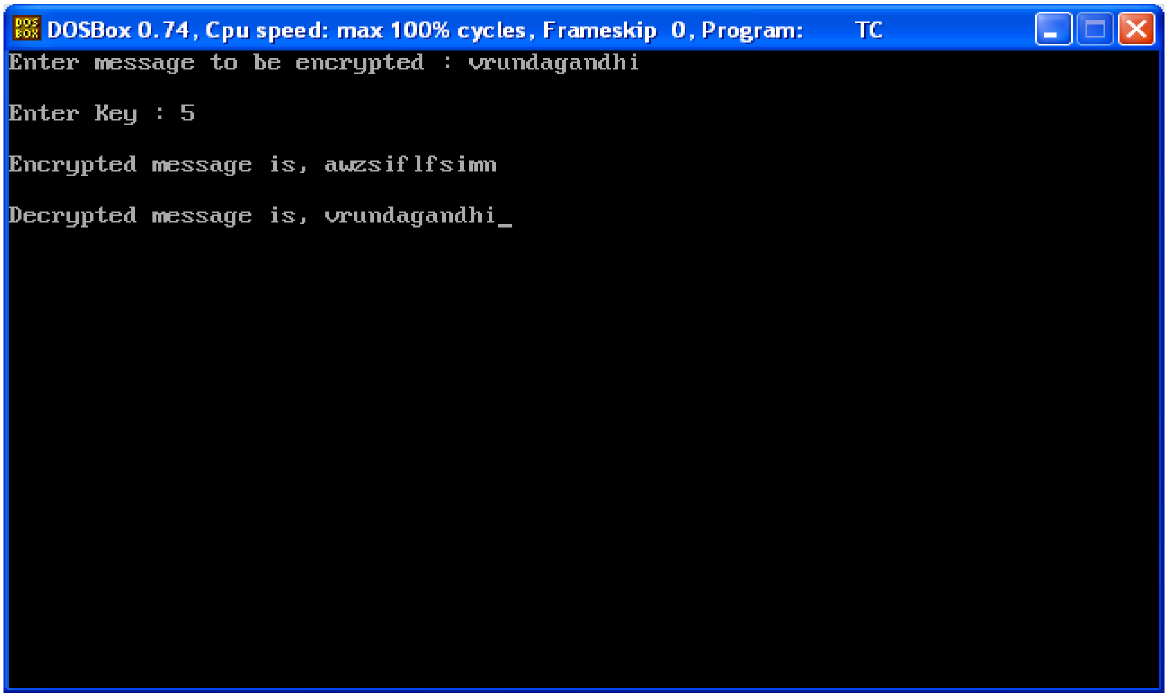
Where takes on a value in the range 1 to 25.The decryption algorithm is simply

$$p = D(k, C) = (C - k) \bmod 26$$

RSULTS / OBSERVATIONS:

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

void main(){
	char pt[50], ct[50], dt[50];
	int key, len, i;
	clrscr();
	printf("Enter message to be encrypted : ");
	scanf("%s", &pt);
	len = strlen(pt);
	again :
	printf("\nEnter Key : ");
	scanf("%d", &key);
	if(key > 26){
		printf("\nInvalid key. Please enter valid key.\n");
		goto again;
	}
	for(i = 0; i < len; i++){
		ct[i] = pt[i] + key;
		if(ct[i] > 122)
			ct[i] = ct[i] - 26;
	}
	ct[i] = '\0';
	printf("\nEncrypted message is, %s", ct);
	for(i = 0; i < len; i++) {
		dt[i] = ct[i] - key;
		if(dt[i] < 97)
			dt[i] = dt[i] + 26;
	}
	dt[i] = '\0';
	printf("\n\nDecrypted message is, %s", dt);
	getch();
}
```

Output:-


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter message to be encrypted : vrundagandhi
Enter Key : 5
Encrypted message is, awzsiflfsimn
Decrypted message is, vrundagandhi_

PRACTICAL- 2

OBJECTIVE :

Implement Playfair cipher encryption-decryption.

THEORY:

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.

The technique encrypts pairs of letters (digraphs), instead of single letters as in the simple substitution cipher. The Playfair is significantly harder to break since the frequency analysis used for simple substitution ciphers does not work with it. Frequency analysis can still be undertaken, but on the 25*25=625 possible digraphs rather than the 25 possible monographs. Frequency analysis thus requires much more ciphertext in order to work.

The best-known multiple-letter encryption cipher is the Playfair, which treats digrams in the plaintext as single units and translates these units into ciphertext digrams. The Playfair algorithm is based on the use of a 5 × 5 matrix of letters constructed using a keyword. Here is an example, solved by Lord Peter Wimsey in Dorothy Sayers's *Have His Carcase:*

M	O	N	A	R
C	H	Y	B	D
E	F	G	I/J	K
L	P	Q	S	T
U	V	W	X	Z

In this case, the keyword is *monarchy*. The matrix is constructed by filling in the letters of the keyword (minus duplicates) from left to right and from top to bottom, and then filling in the remainder of the matrix with the remaining letters in alphabetic order. The letters I and J count as one letter. Plaintext is encrypted two letters at a time, according to the following rules:

1. Repeating plaintext letters that are in the same pair are separated with a filler letter, such as x, so that balloon would be treated as ba lx lo on.

2. Two plaintext letters that fall in the same row of the matrix are each replaced by the letter to the right, with the first element of the row circularly following the last. For example, ar is encrypted as RM.
3. Two plaintext letters that fall in the same column are each replaced by the letter beneath, with the top element of the column circularly following the last. For example, mu is encrypted as CM.
4. Otherwise, each plaintext letter in a pair is replaced by the letter that lies in its own row and the column occupied by the other plaintext letter. Thus, hs becomes BP and ea becomes IM (or JM, as the encipherer wishes).

Assume one wants to encrypt the digraph OR. There are three general cases:

```
m * * a *
* * * * *
* * * * *
l * * s *
* * * * *
```

Hence, al -> ms

```
* * * * *
* h y b d
* * * * *
* * * * *
* * * * *
```

Hence, hb -> yd

```
* * n * *
* * y * *
* * * * *
* * q * *
* * w * *
```

Hence, nq -> yw

An example encryption, "we are discovered, save yourself" using the key square shown at the beginning of this section:

```
plaintext: wearediscoveredsaveyourselfx
ciphertext: ugrmkcsxhmufmkbtoxgcmvatluiv
```

RSULTS / OBSERVATIONS:

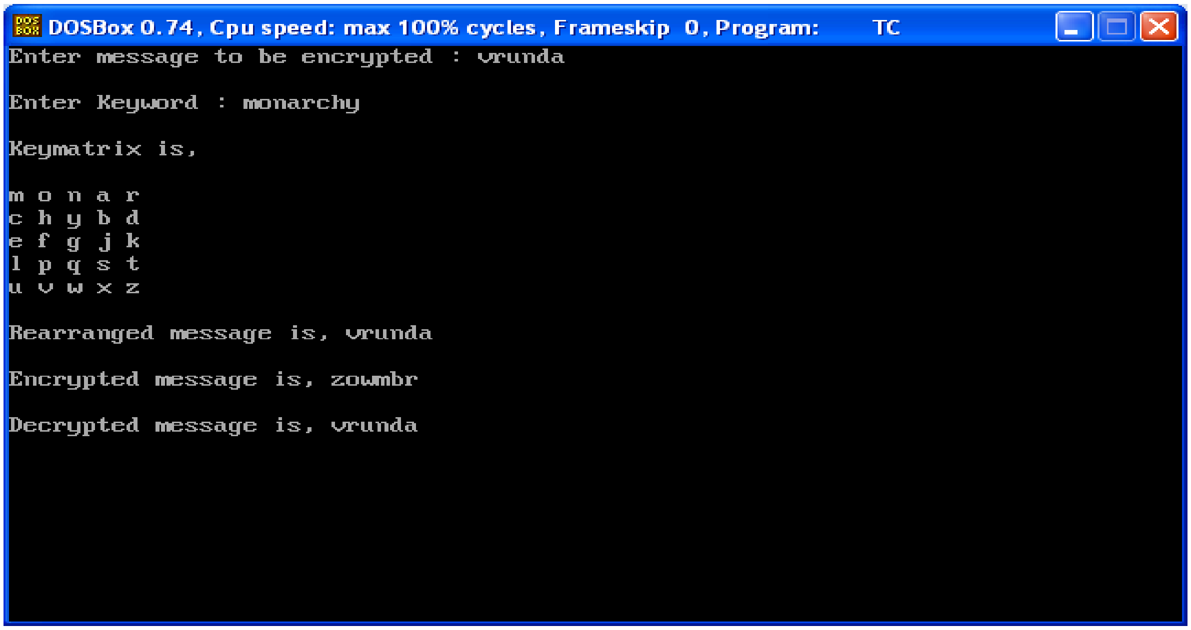
```
#include<stdio.h>
#include<conio.h>

void main(){
        int n,i,j,k,temp,len,r1,r2,c1,c2, flag, index = 0, defaultValue = 97, count = 0;
        char pt[50], ct[50], dt[50], key[10], keyMatrix[5][5], comparedValue, tempText[26];
        clrscr();
        printf("Enter message to be encrypted : ");
        scanf("%s", &pt);
        n=strlen(pt);
        len=n;
        printf("\nEnter Keyword : ");
        scanf("%s", &key);
        for(i = 0; i < strlen(key); i++){
                if(key[i] == 'i'){
                        key[i] = 'j';
                }
        }
        for(i = 0; i < 5; i++){
                for(j = 0; j < 5; j++){
                        if(index < strlen(key)){
                                comparedValue = key[index];
                                index++;
                        }else{
                                if(defaultValue == 105){
                                        defaultValue = 106;
                                }
                                comparedValue = defaultValue;
                                defaultValue++;
                        }
                        flag = 0;
                        for(k = 0; k < count; k++){
                                if(comparedValue == tempText[k]){
                                        flag = 1;
                                        break;
                                }
                        }
                        if (flag != 0){
                          j--;
                          if(j < 0){
                                i--;
```

```
                    j = 4;
                  }
                }else{
                  keyMatrix[i][j] = comparedValue;
                  tempText[count] = keyMatrix[i][j];
                  count++;
                }
            }
        }
        printf("\nKeymatrix is, \n\n");
        for(i = 0; i < 5; i++){
            for(j = 0; j < 5; j++){
                printf("%c ", keyMatrix[i][j]);
            }
            printf("\n");
        }
        for(i = 0; i < n; i = (i+2)){
            if(pt[i] == 'i')
                pt[i] = 'j';
            if(pt[i+1] == 'i')
                pt[i+1] = 'j';
            if(pt[i] == pt[i+1]){
                temp = pt[i+1];
                pt[i+1] = 'x';
                len++;
                for(j =(n-1); j > (i+1); j--)
                    pt[j+1] = pt[j];
                pt[j+1] = temp;
            }
        }
        if((len % 2) != 0){
            pt[len] = 'z';
            len++;
        }
        for(i = 0; i < len; i = (i+2)){
            for(j = 0; j < 5; j++){
                for(k = 0; k < 5; k++){
                    if(pt[i] == keyMatrix[j][k]){
                        r1 = j;
                        c1 = k;
                    }
                    if(pt[i+1] == keyMatrix[j][k]){
                        r2 = j;
                        c2 = k;
                    }
                }
```

```
        }
        if(r1 == r2){
            c1++;
            c2++;
            if(c1 > 4)
                c1 = 0;
            if(c2 > 4)
                c2 = 0;
            ct[i] = keyMatrix[r1][c1];
            ct[i+1] = keyMatrix[r2][c2];
        }else if(c1 == c2){
            r1++;
            r2++;
            if(r1 > 4)
                r1 = 0;
            if(r2 > 4)
                r2 = 0;
            ct[i] = keyMatrix[r1][c1];
            ct[i+1] = keyMatrix[r2][c2];
        }else{
            ct[i] = keyMatrix[r1][c2];
            ct[i+1] = keyMatrix[r2][c1];
        }
    }
    printf("\nRearranged message is, ");
    for(i = 0; i < len; i++){
        printf("%c",pt[i]);
    }
    printf("\n\nEncrypted message is, ");
    for(i = 0; i < len; i++){
        printf("%c",ct[i]);
    }
    for(i = 0; i < len; i = (i+2)){
        for(j = 0; j < 5; j++){
            for(k = 0; k < 5; k++){
                if(ct[i] == keyMatrix[j][k]){
                    r1 = j;
                    c1 = k;
                }
                if(ct[i+1] == keyMatrix[j][k]){
                    r2 = j;
                    c2 = k;
                }
            }
        }
        if(r1 == r2){
```

```
                    c1--;
                    c2--;
                    if(c1 < 0)
                        c1 = 4;
                    if(c2 < 0)
                        c2 = 4;
                    dt[i] = keyMatrix[r1][c1];
                    dt[i+1] = keyMatrix[r2][c2];
                }else if(c1 == c2){
                    r1--;
                    r2--;
                    if(r1 < 0)
                        r1 = 4;
                    if(r2 < 0)
                        r2 = 4;
                    dt[i] = keyMatrix[r1][c1];
                    dt[i+1] = keyMatrix[r2][c2];
                }else{
                    dt[i] = keyMatrix[r1][c2];
                    dt[i+1] = keyMatrix[r2][c1];
                }
        }
        printf("\n\nDecrypted message is, ");
        for(i = 0; i < len; i++){
                printf("%c",dt[i]);
        }
        getch();
}
```

Output:-



PRACTICAL- 3

OBJECTIVE :

Implement Polyalphabetic (Vigenere) cipher encryption-decryption.

THEORY:

The Vigenère cipher, was invented by a Frenchman, Blaise de Vigenère in the 16th century. It is a polyalphabetic cipher because it uses two or more cipher alphabets to encrypt the data. In other words, the letters in the Vigenère cipher are shifted by different amounts, normally done using a word or phrase as the encryption key.

Unlike the mono-alphabetic ciphers, polyalphabetic ciphers are not susceptible to frequency analysis, as more than one letter in the plaintext can be represented by a single letter in the encryption. One of the main problems with simple substitution ciphers is that they are so vulnerable to *frequency analysis*.

Given a sufficiently large cipher text, it can easily be broken by mapping the frequency of its letters to the know frequencies of, say, English text. Therefore, to make ciphers more secure, cryptographers have long been interested in developing enciphering techniques that are immune to frequency analysis. One of the most common approaches is to suppress the normal frequency data by using more than one alphabet to encrypt the message.

A *polyalphabetic substitution cipher* involves the use of two or more cipher alphabets. Instead of there being a one-to-one relationship between each letter and its substitute, there is a one-to-many relationship between each letter and its substitutes.

THE VIGENÈRE SQUARE:

Blaise de Vigenère developed a square to help encode messages. Reading along each row, you can see that it is a really a series of Caesar ciphers the first has a shift of 1, the second a shift of 2 and so.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	D
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	C
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	B
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

The Vigenère cipher uses this table in conjunction with a key to encipher a message. So, if we were to encode a message using the key COUNTON, we write it as many times as necessary above our message. To find the encryption, we take the letter from the intersection of the Key letter row, and the Plaintext letter column.

Key	C	O	U	N	T	O	N	C	O	U	N	T	O	N
Plaintext	V	I	G	E	N	E	R	E	C	I	P	H	E	R
Encryption	X	W	A	R	G	S	E	G	Q	C	C	A	S	E

To decipher the message, the recipient needs to write out the key above the cipher text and reverse the process. The maths behind the Vigenère cipher can be written as follows:

To encrypt a message: Ca = Ma + Kb (mod 26)

To decrypt a message: Ma = Ca – Kb (mod 26)

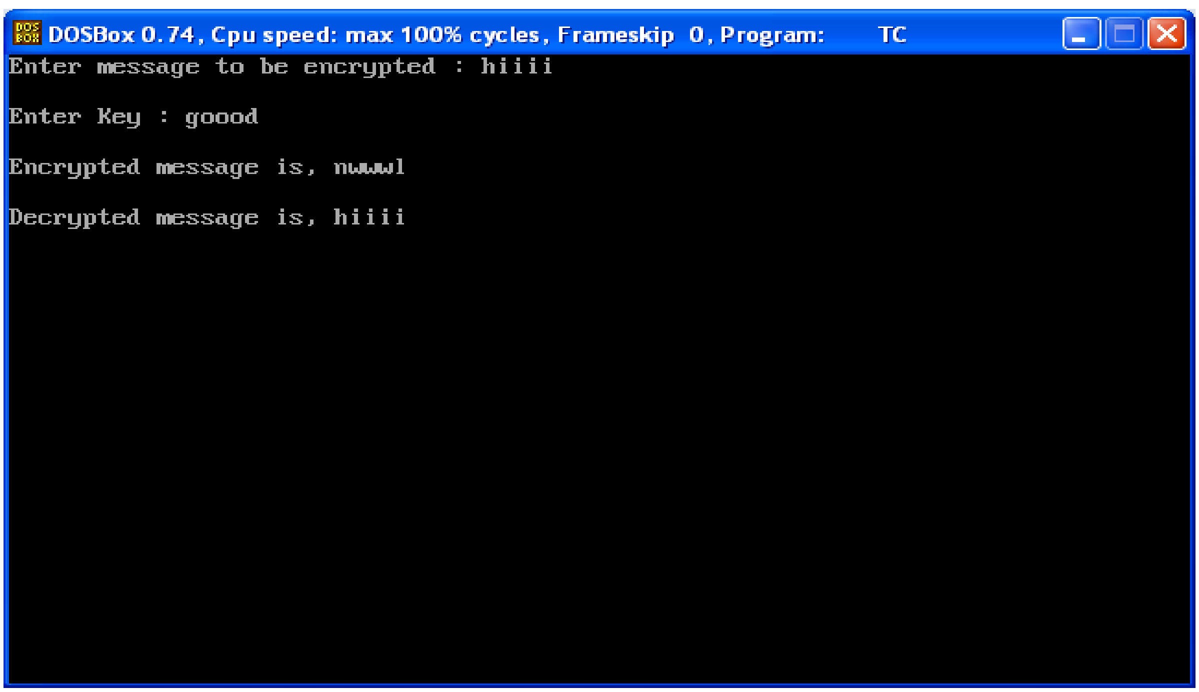
(Where C = Code, M = Message, K = Key, and where a = the ath character of the message bounded by the message, and b is the bth character of the Key bounded by the length of the key.)

RSULTS / OBSERVATIONS:

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

void main(){
	char pt[50], o[26], ct[50], dt[50], k, key[15], inversekey[15];
	int i, j = 0, ptlen, keylen;
	clrscr();
	for(i = 0; i < 26; i++)
		o[i] = i + 97;
	printf("Enter message to be encrypted : ");
	scanf("%s", &pt);
	ptlen = strlen(pt);
	printf("\nEnter Key : ");
	scanf("%s", &key);
	keylen = strlen(key);
	for(i = 0; i < ptlen; i++){
		for(k = 0; k < 26; k++){
			if(pt[i] == o[k])
			break;
		}
		ct[i] = key[j] + k;
		if(ct[i] > 122)
			ct[i] = ct[i] - 26;
		if(j > (keylen - 2))
			j = 0;
		else
			j++;
	}
	ct[i] = '\0';
	printf("\nEncrypted message is, %s", ct);
	j = 0;
	for(i = 0; i < keylen; i++){
		for(k = 0; k < 26; k++){
			if(key[i] == o[k])
			break;
		}
		inversekey[i] = 123 - k;
		if(inversekey[i] > 122)
			inversekey[i] = inversekey[i] - 26;
	}
```

```
        inversekey[i] = '\0';
        for(i = 0; i < ptlen; i++){
                for(k = 0; k < 26; k++){
                        if(ct[i] == o[k])
                        break;
                }
                if((inversekey[j] + k) > 122)
                        dt[i] = (inversekey[j] + k) - 26;
                else
                        dt[i] = inversekey[j] + k;
                if(j > (keylen - 2))
                         j = 0;
                else
                         j++;
        }
        dt[i] = '\0';
        printf("\n\nDecrypted message is, %s", dt);
        getch();
}
```

<u>Output</u>:-



PRACTICAL- 4

OBJECTIVE :

Implement Hill cipher encryption-decryption.

THEORY:

THE HILL Cipher is an encryption algorithm takes **m** successive plaintext letters and substitutes for them **m** cipher text letters. The substitution is determined by **m** linear equations in which each character is assigned a numerical value **(a = 0, b = 1… z = 25)**. For **m =3**, the system can be described as;

$$c_1 = (k_{11}p_1 + k_{12}p_2 + k_{13}p_3) \bmod 26$$
$$c_2 = (k_{21}p_1 + k_{22}p_2 + k_{23}p_3) \bmod 26$$
$$c_3 = (k_{31}p_1 + k_{32}p_2 + k_{33}p_3) \bmod 26$$

This can be expressed in terms of row vectors and matrices:

$$(c_1\ c_2\ c_3) = (p\ \ p_2\ \ p_3)\begin{pmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{pmatrix} \bmod 26$$

C = **PK** mod 26

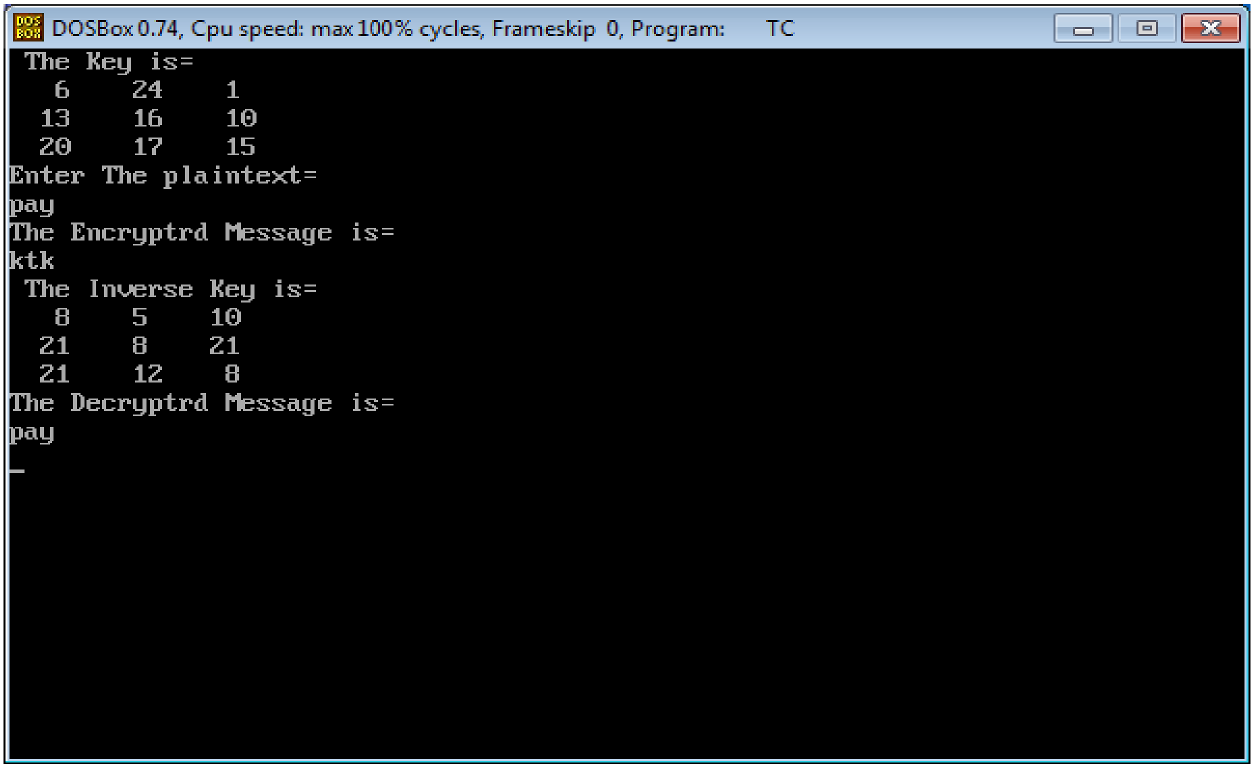
Where **C** and **P** are row vectors of length 3 representing the plaintext and cipher text, and **K** is a 3 x 3 matrix representing the encryption key. Operations are performed mod 26.

RESULTS / OBSERVATIONS:

```
#include<stdio.h>
#include<conio.h>
void main()
{
	char s[26]={'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z'};
	char p[100],ency[100],decy[100];
	int
i,j,a[10][10]={{6,24,1},{13,16,10},{20,17,15}},length,num[100],c[100],b[10][10]={{8,5,10},{2
1,8,21},{21,12,8}};
	clrscr();
	printf(" The Key is=\n ");
	for (i=0;i<3;i++)
	{
		for(j=0;j<3;j++)
			printf("  %d  ",a[i][j]);
		 printf("\n");
	}
	printf("Enter The plaintext=\n");
	scanf("%s",p);
	length=strlen(p);
	for (i=0;i<length;i++)
	{
		for (j=0;j<26;j++)
		{
			if (s[j]==p[i])
				num[i]=j;
		}
	}
	for (i=0;i<3;i++)
	{
		c[i]=0;
		for (j=0;j<3;j++)
		  c[i]=c[i]+(a[i][j]*num[j]);
	}
	c[i]='\0';
	printf("The Encryptrd Message is=\n");
	for (i=0;i<3;i++)
		ency[i]=s[c[i]%26];
	ency[i]='\0';
	puts(ency);
	printf(" The Inverse Key is=\n ");
```

```
        for (i=0;i<3;i++)
        {
                for(j=0;j<3;j++)
                        printf("  %d  ",b[i][j]);
                 printf("\n");
        }
        for (i=0;i<length;i++)
        {
                for (j=0;j<26;j++)
                {
                        if (s[j]==ency[i])
                                num[i]=j;
                }
        }
        for (i=0;i<3;i++)
        {
                c[i]=0;
                for (j=0;j<3;j++)
                  c[i]=c[i]+(b[i][j]*num[j]);
        }
        c[i]='\0';
        printf("The Decryptrd Message is=\n");
        for (i=0;i<3;i++)
                decy[i]=s[c[i]%26];
        decy[i]='\0';
        puts(decy);
        getch();
}
```

Output :-



PRACTICAL- 5

OBJECTIVE :

Implementation of Transposition cipher.

THEORY:

Transposition cipher, simple data encryption scheme in which plaintext characters are shifted in some regular pattern to form cipher text.In manual systems transpositions are generally carried out with the aid of an easily remembered mnemonic. For example, a popular schoolboy cipher is the “rail fence,” in which letters of the plaintext are written alternating between rows and the rows are then read sequentially to give the cipher. In a depth-two rail fence (two rows) the message WE ARE DISCOVERED SAVE YOURSELF would be written;

W A E I C V R D A E O R E F
E R D S O E E S V Y U S L

or

W A E I C V R D A E O R E F E R D S O E E S V Y U S L .

Simple frequency counts on the cipher text would reveal to the cryptanalyst that letters occur with precisely the same frequency in the cipher as in an average plaintext and, hence, that a simple rearrangement of the letters is probable.

The rail fence is the simplest example of a class of transposition ciphers, known as route ciphers, which enjoyed considerable popularity in the early history of cryptology. In general, the elements of the plaintext (usually single letters) are written in a prearranged order (route) into a geometric array (matrix)—typically a rectangle—agreed upon in advance by the transmitter and receiver and then read off by following another prescribed route through the matrix to produce the cipher. The key in a route cipher consists of keeping secret the geometric array, the starting point, and the routes. Clearly both the matrix and the routes can be much more complex than in this example; but even so, they provide little security. One form of transposition (permutation) that was widely used depends on an easily remembered key word for identifying the route in which the columns of a rectangular matrix are to be read. For example, using the key word AUTHOR and ordering the columns by the lexicographic order of the letters in the key word

A	U	T	H	O	R
1	6	5	2	3	4
W	E	A	R	E	D
I	S	C	O	V	E
R	E	D	S	A	V
E	Y	O	U	R	S
E	L	F	A	B	C

yields the cipher

W I R E E R O S U A E V A R B D E V S C A C D O F E S E Y L .

In decrypting a route cipher, the receiver enters the cipher text symbols into the agreed-upon matrix according to the encryption route and then reads the plaintext according to the original order of entry. A significant improvement in crypto-security can be achieved by re-encrypting the cipher obtained from one transposition with another transposition. Because the result (product) of two transpositions is also a transposition, the effect of multiple transpositions is to define a complex route in the matrix, which in itself would be difficult to describe by any simple mnemonic. In the same class also fall systems that make use of perforated cardboard matrices called grilles;

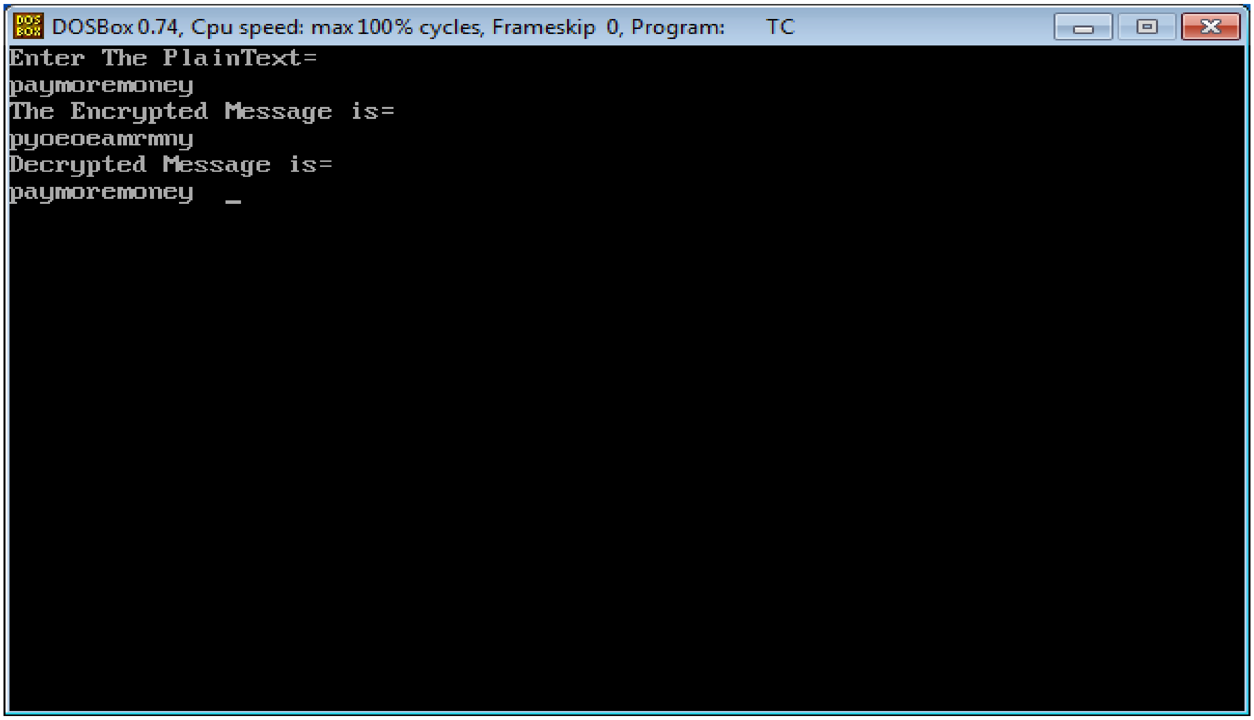
descriptions of such systems can be found in older books on cryptography. In contemporary cryptography, transpositions serve principally as one of several encryption steps in forming a compound or product cipher.

RSULTS / OBSERVATIONS:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
	int i,length,j,k,l,len1,len2,temp1,temp2,q;
	char
a[100],text1[100],text2[100],encrypt[100],decrypttext1[100],decrypttext2[100],decrypt[100];
	clrscr();
	printf("Enter The PlainText=\n");
	gets(a);
	length=strlen(a);
	j=0;
	k=0;
	for (i=0;i<length;i++)
	{
		if (i%2==0)
		{
			text1[j]=a[i];
			j++;
		}
		else
		{
			text2[k]=a[i];
			k++;
		}
	}
	for (i=0;i<j;i++)
	{
		encrypt[i]=text1[i];
	}
	len1=i;
	for (l=0;l<k;l++)
	{
		encrypt[i]=text2[l];
		i++;
	}
	len2=l;
	encrypt[i]='\0';
	printf("The Encrypted Message is=\n");
```

```
        printf("%s\n",encrypt);
                for (i=0;i<len1;i++)
                {
                        decrypttext1[i]=encrypt[i];
                }
                decrypttext1[i]='\0';
                for (j=0;j<len2;j++)
                {
                        decrypttext2[j]=encrypt[i];
                        i++;
                }
                decrypttext2[j]='\0';
                q=0;
                printf("Decrypted Message is=\n");
                for (i=0;i<strlen(encrypt);i++)
                {
                    if (i<=len1)
                    {
                    printf("%c",decrypttext1[i]);
                    }
                     if (i<=len2)
                    {
                    printf("%c",decrypttext2[i]);
                    }
                }
                decrypt[q]='\0';
        getch();
}
```

Output :-



PRACTICAL- 6

OBJECTIVE :

To implement Simple DES.

THEORY:

The **Data Encryption Standard** is a previously predominant symmetric-key algorithm for the encryption of electronic data. It was highly influential in the advancement of modern cryptography in the academic world. Developed in the early 1970s at IBM and based on an earlier design by Horst Feistel, the algorithm was submitted to the National Bureau of Standards (NBS) following the agency's invitation to propose a candidate for the protection of sensitive, unclassified electronic government data. In 1976, after consultation with the National Security Agency (NSA), the NBS eventually selected a slightly modified version, which was published as an official Federal Information Processing Standard (FIPS) for the United States in 1977. The publication of an NSA-approved encryption standard simultaneously resulted in its quick international adoption and widespread academic scrutiny. Controversies arose out of classified design elements, a relatively short key length of the symmetric-key block cipher design, and the involvement of the NSA, nourishing suspicions about a backdoor. The intense academic scrutiny the algorithm received over time led to the modern understanding of block ciphers and their cryptanalysis.

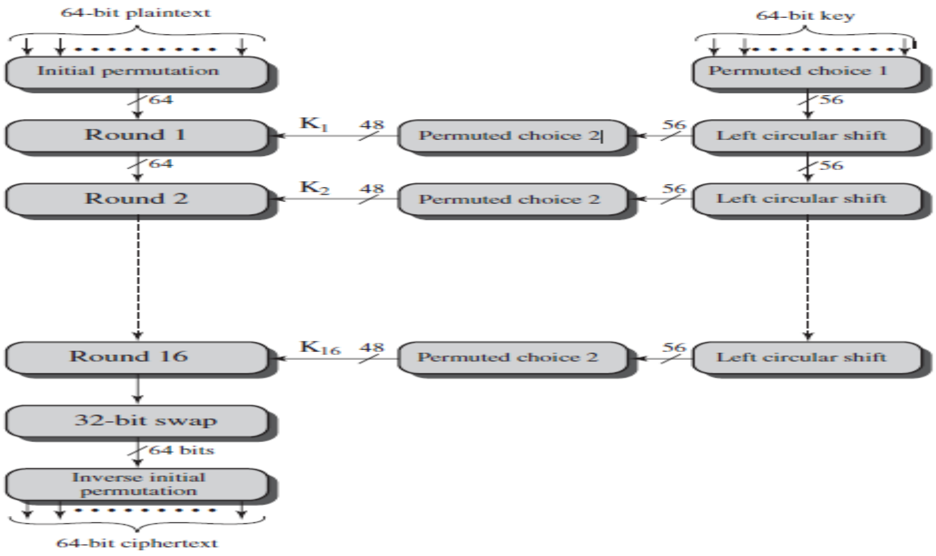
DES FEATURES:

- Block size = 64 bits
- Key size = 56 bits (in reality, 64 bits, but 8 are used as parity-check bits for error control)
- Number of rounds = 16
- 16 intermediary keys, each 48 bits

KEY LENGTH IN DES:

- In the DES specification, the key length is 64 bit.
- 8 bytes; in each byte, the 8th bit is a parity-check bit.

Each parity-check bit is the XOR of the previous 7 bits.



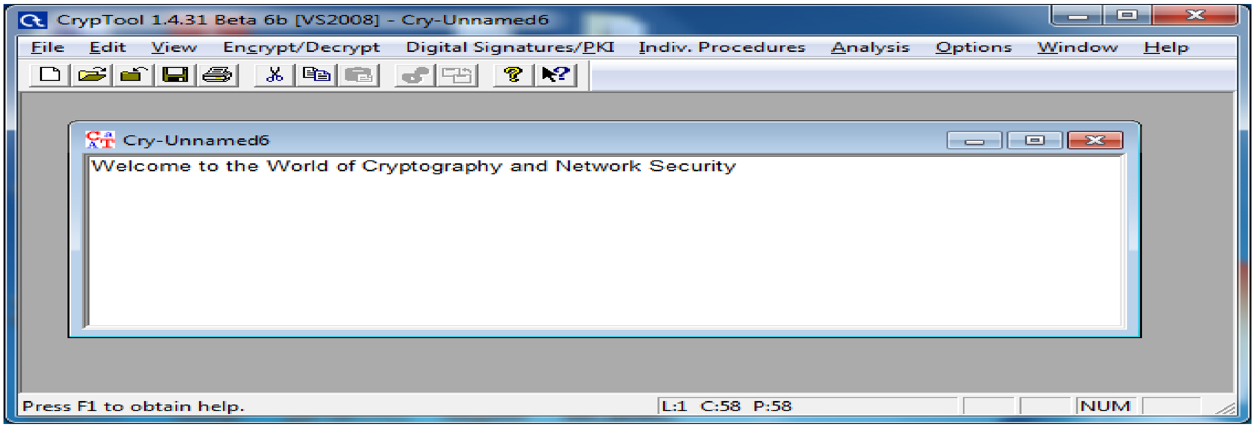
PROCEDURE:

In this excercize you can use the **DES**-component to encrypt an arbitrary text entered in the **Input message**-component on the left side. The resulting encrypted text is displayed in the **Output message**component on the right side after hitting the Play-button. The **DES**-component works on binary values, i.e. bytes. Thus, the inputed text is first converted to bytes with the **Message decoder**-component. With the current settings, it is interpreted as ASCII. The resulting bytes are then encrypted with **DES**, yielding another sequence of bytes. These bytes are then simply printed as hexadecimal values with the help of the **Message encoder**-component. Note that you can also decrypt messages with this template. To do so, you first copy the encrypted hexadecimal values to the **Input message**. Then you have to change the following:

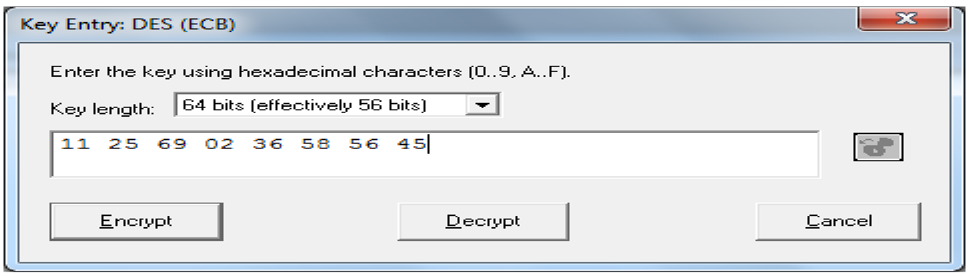
- Set *Input format* of the **Message decoder** to *Hexadecimal*
- Set *Action* of **DES** to *Decrpyt*;
- Set *Format* of the **Message encoder** to *Text* and the *encoding* to *ASCII*.

RESULTS / OBSERVATIONS:

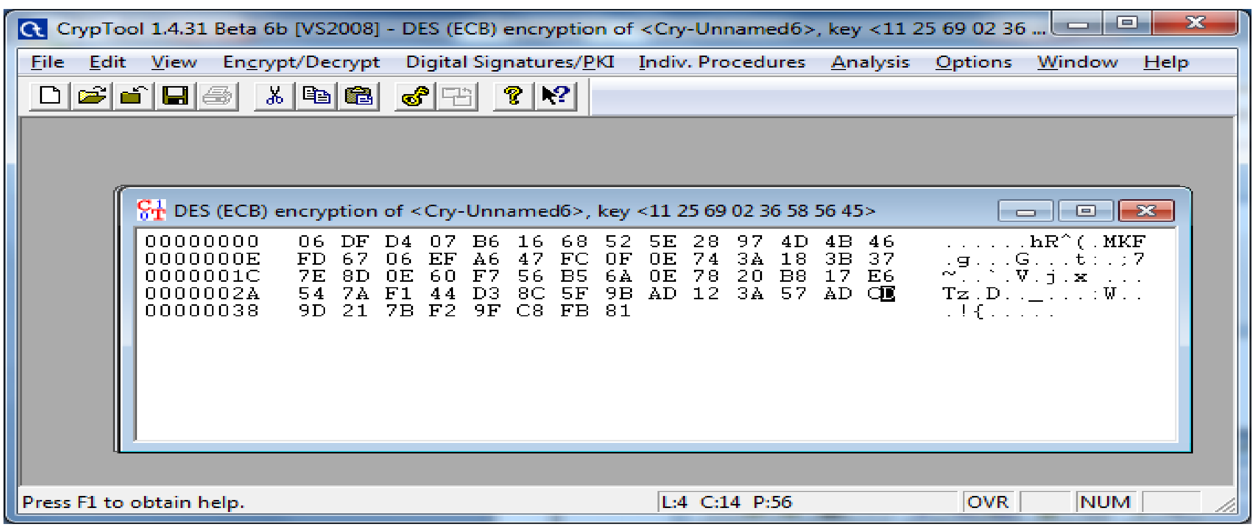
Step 1: Enter the Plaintext



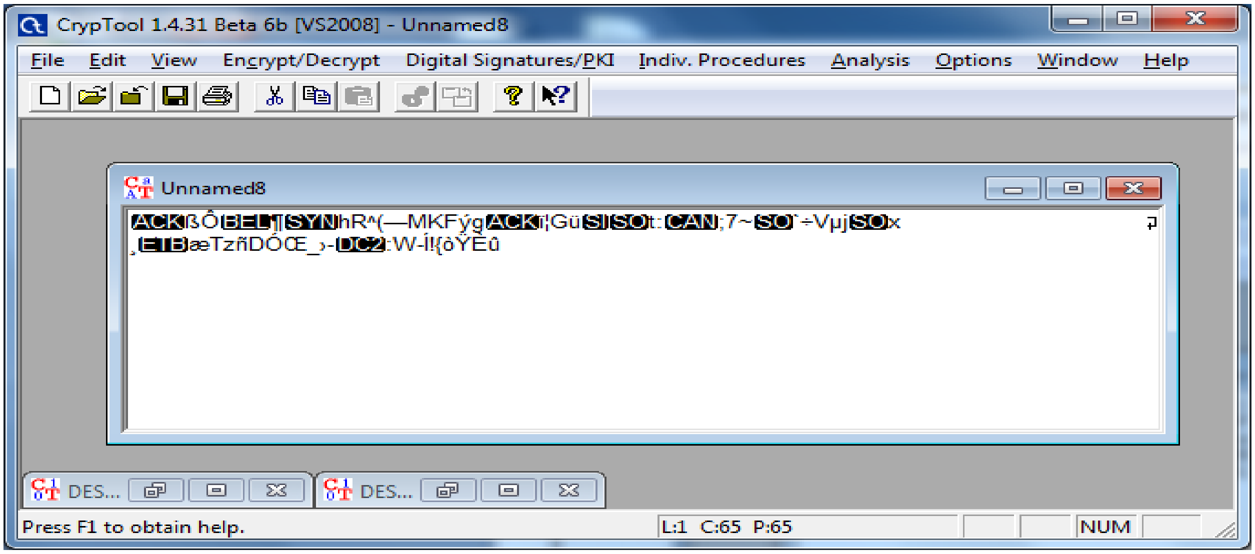
Step 2 : Enter the Key Value

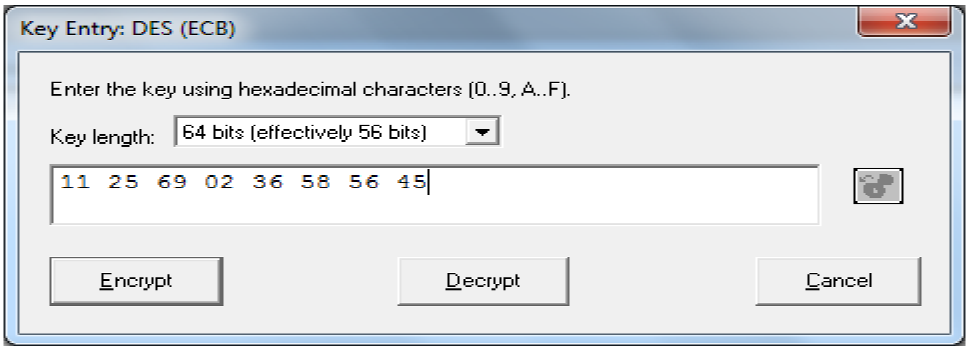


Step 3 : Generated Encrypted Text (Cipher Text)

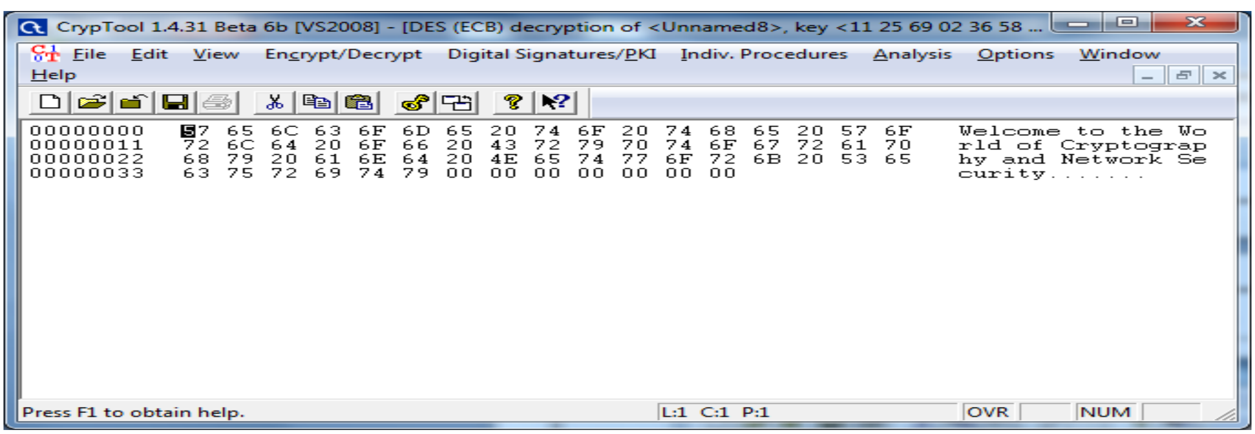


Step 4: For Decryption Process Enter Cipher Text and the same key value





Step 5 : Generated Plain Text



PRACTICAL- 7

OBJECTIVE :

Implement Diffi-Hellmen Key exchange Method.

THEORY:

first public-key type scheme proposed

- By Diffie & Hellman in 1976 along with the exposition of public key concepts
- Note: now know that Williamson (UK CESG) secretly proposed the concept in 1970
  is a practical method for public exchange of a secret key
- Used in a number of commercial products
- Public-key distribution scheme cannot be used to exchange an arbitrary message Rather it can establish a common key
  known only to the two participants
- Value of key depends on the participants (and their private and public key information)
  based on exponentiation in a finite (Galois) field (modulo a prime or a polynomial)

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 primitve 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;
}
```

Result:

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

PRACTICAL- 8

OBJECTIVE :

Implement RSA encryption-decryption algorithm.

THEORY:

by Rivest, Shamir & Adleman of MIT in 1977

- best known & widely used public-key scheme
- based on exponentiation in a finite (Galois) field
- over integers modulo a prime
- exponentiation takes O((log n)3) operations (easy)
- uses large integers (eg. 1024 bits)
- security due to cost of factoring large numbers
- factorization takes O(e log n log log n) operations (hard)to encrypt a message M the sender:

RSA Example - Key Setup

1. Select primes: p=17 & q=11

2. Compute $n = pq$ =17 x 11=187

3. Compute ø(n)=(p–1)(q-1)=16 x 10=160

4. Select e: gcd(e,160)=1; choose e=7

5. Determine d: de=1 mod 160 and d < 160

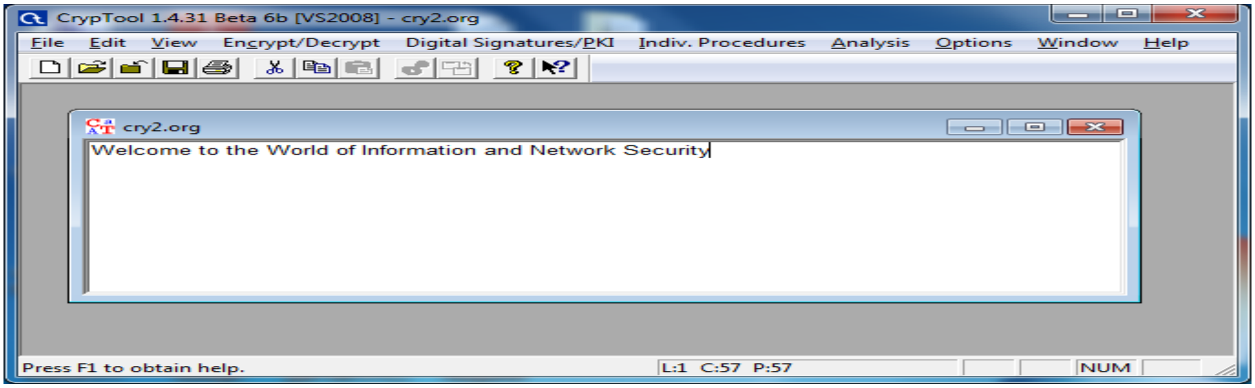
Value is d=23 since 23x7=161= 10x160+1

6. Publish public key PU={7,187}

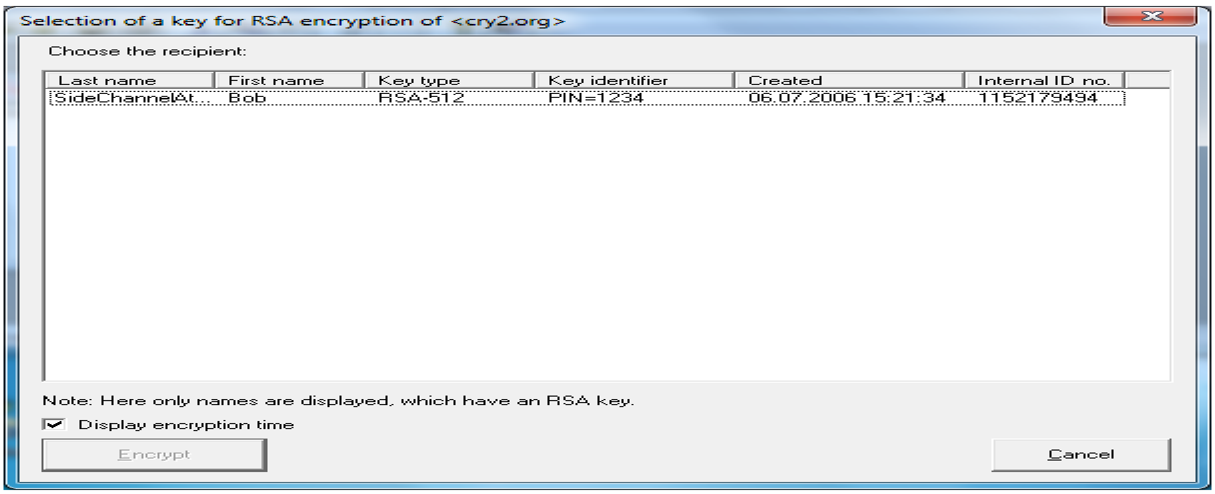
7. Keep secret private key PR={23,187}

RESULTS / OBSERVATIONS:

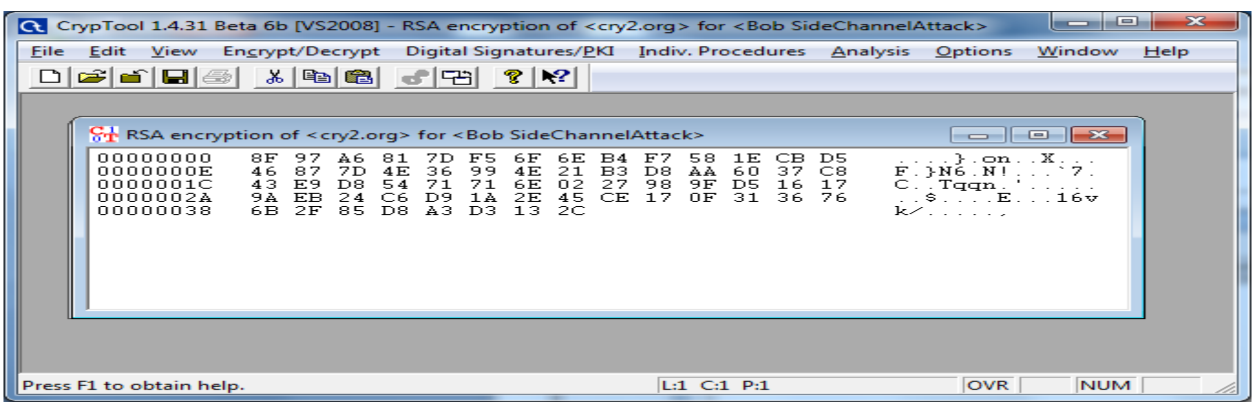
Step 1: Enter Plain Text to Decrypt



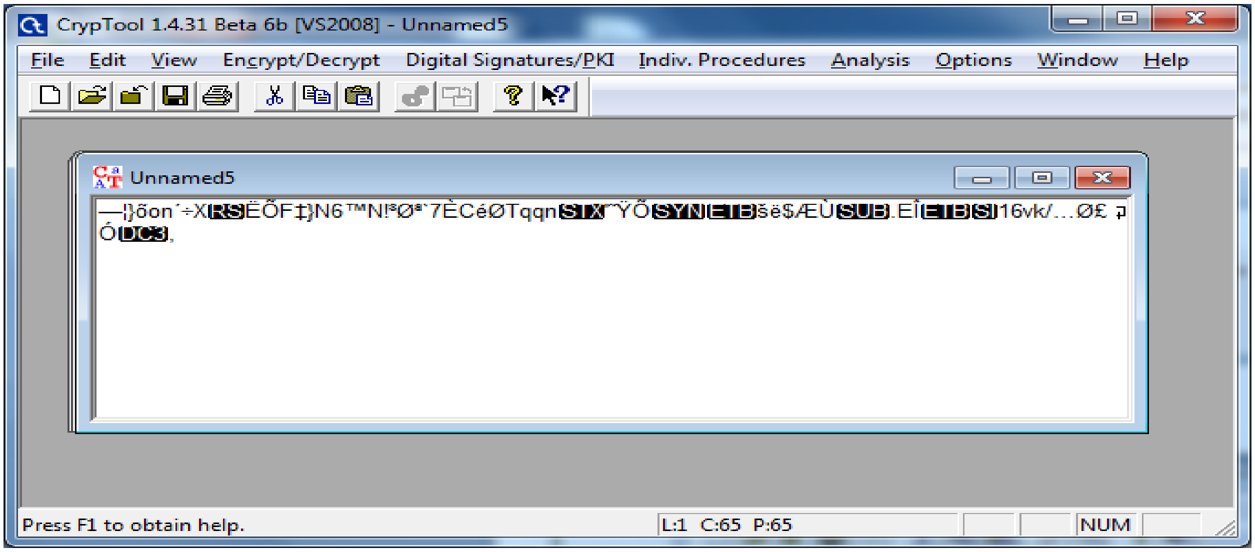
Step 2 : Selection of Key



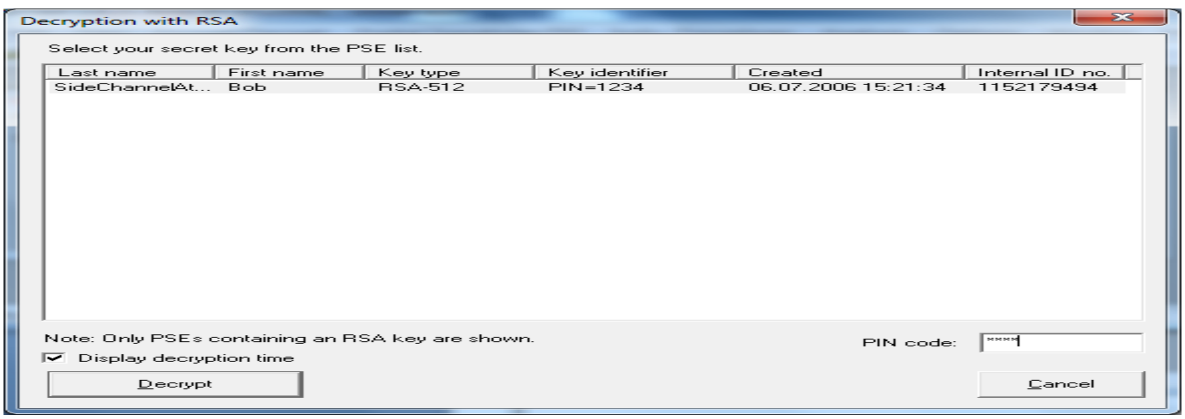
Step 3: Encrypted Text (Cipher Text)



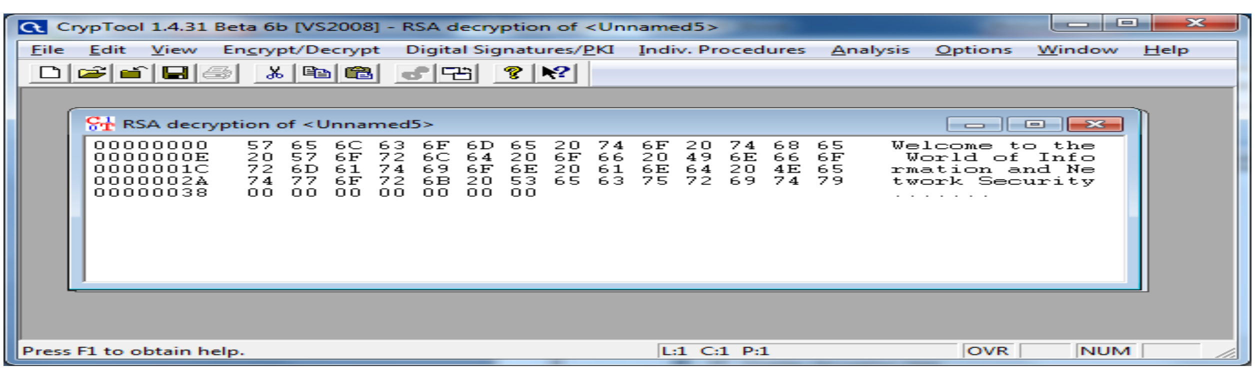
Step 4: Decrypt Cipher Text to Plain Text



Step 5 : Decrypt Text Using Key



Step 6: Generated Plain Text



PRACTICAL- 9

OBJECTIVE:

Perform various encryption-decryption techniques with Cryptool.

INTRODUCTION:

The Cryptool 2 is supplied free program with which each person installing the computer to experiment with encryption. Create their own provisions, to test the algorithms and study the results.

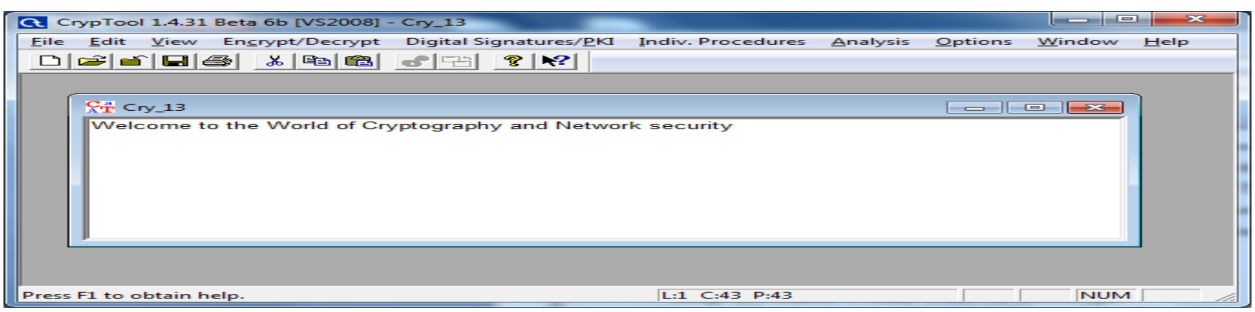
BASIC DESCRIPTION:

More specifically, a description of the basic steps are the following: Select an algorithm determines what kind of work will be performed (encryption - decryption), place the input data (text - file, picture , etc) appropriate type keys according to the algorithm , we form connections in the interface and the output connectors where the result will look and if everything is as it should be - if not, the program does not allow the completion of connections and not running - and then displayed after the execution result. The results obtained are thereafter studied, and compared or used in new encryption or decryption.

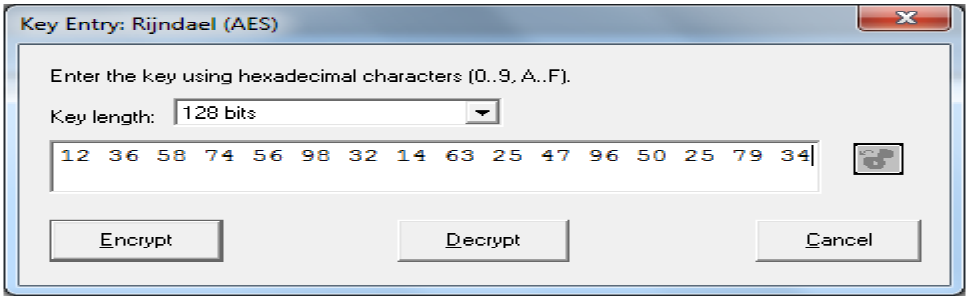
RESULTS / OBSERVATIONS:

A. AES Encryption Technique

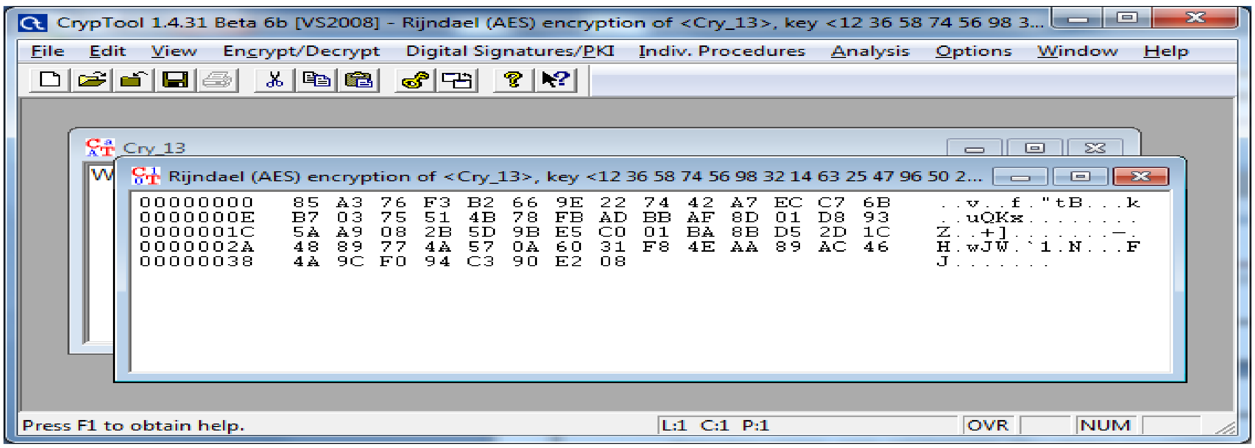
Step 1 : Enter Plain Text to Decrypt



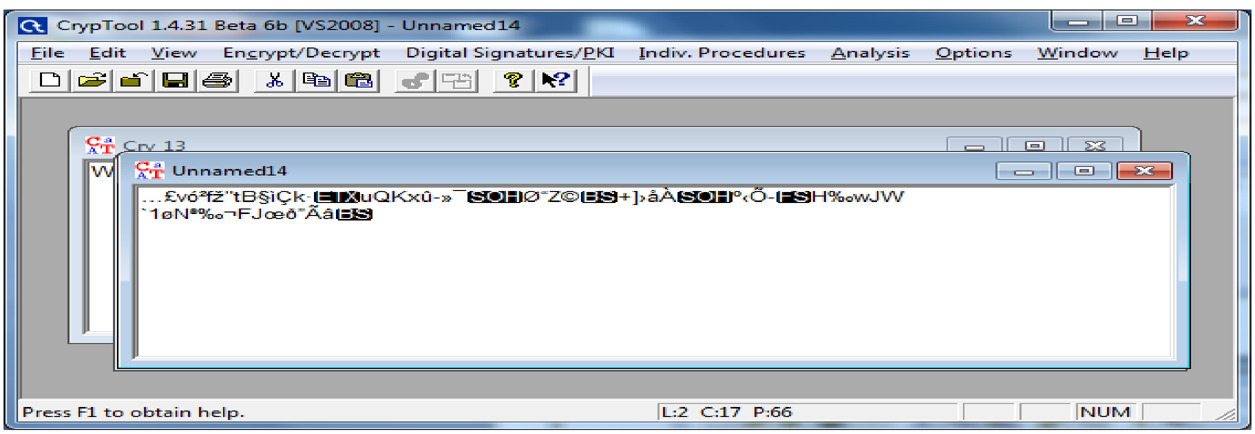
Step 2 : Enter the Key Value

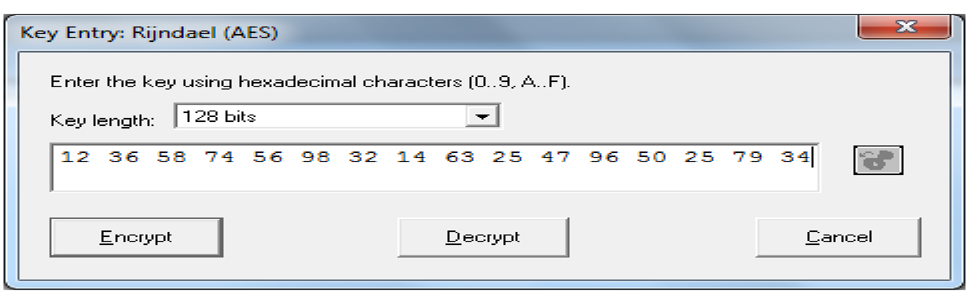


Step 3 : Generated Cipher Text

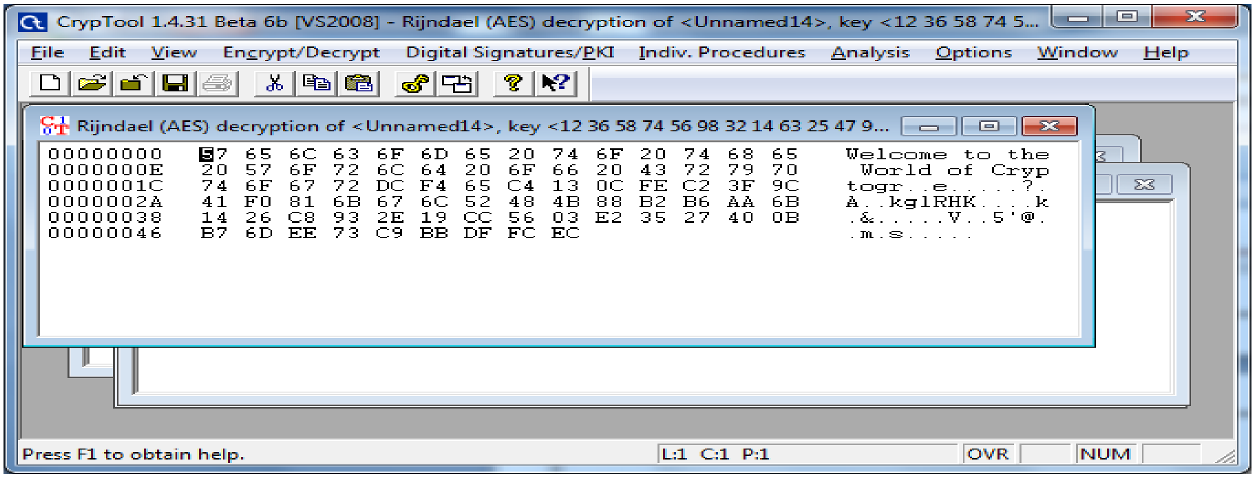


Step 4 : For Decryption use Cipher Text and Key



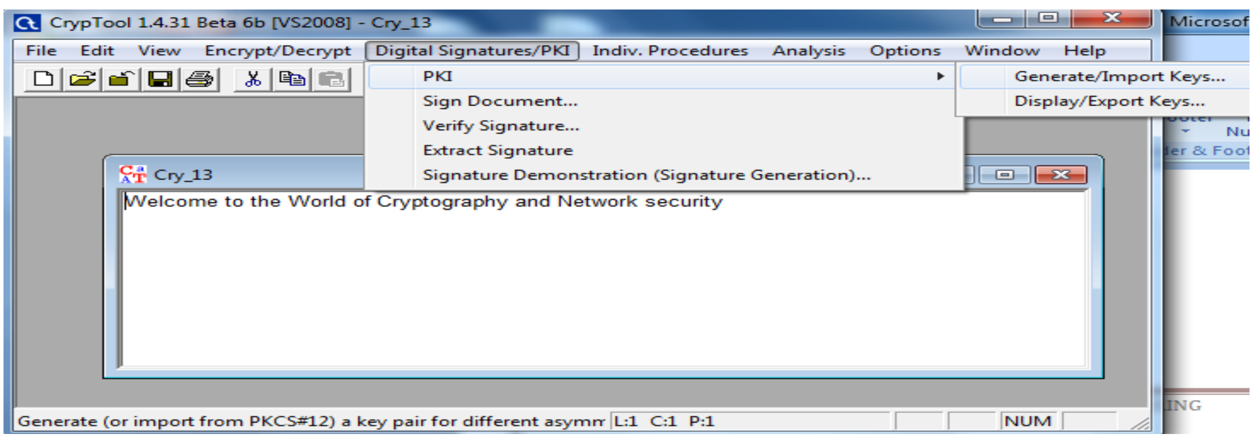


Step 5 : Generated Plain Text

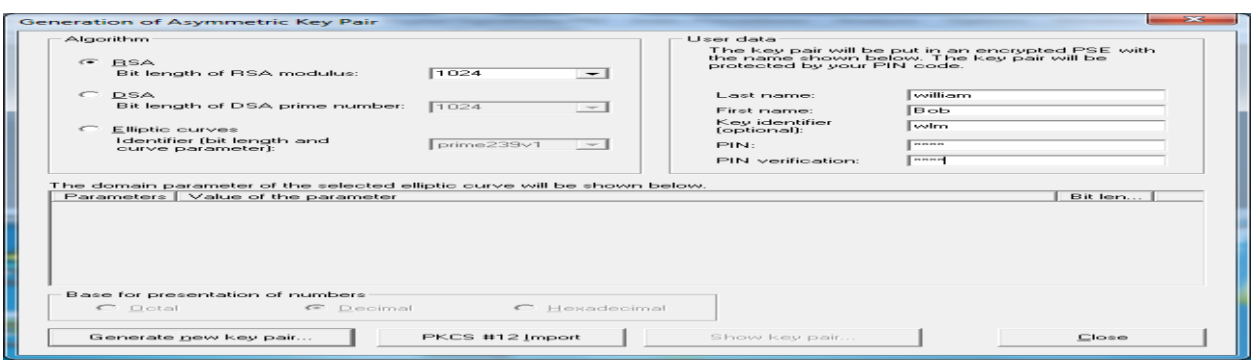


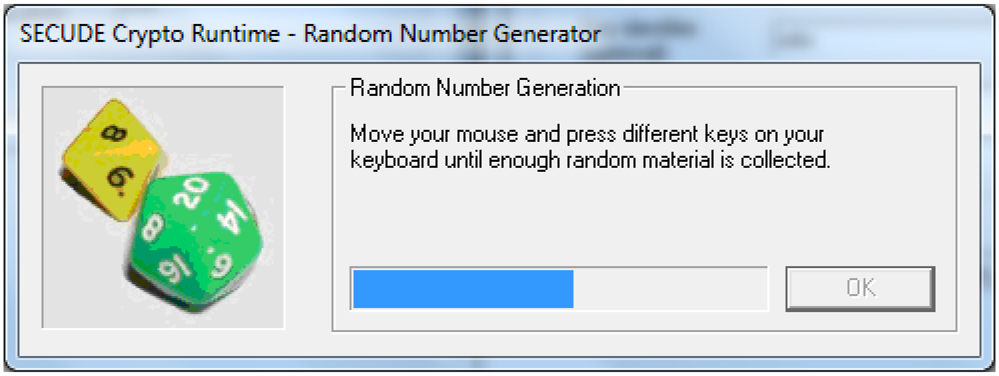
B. Public Key Infrastructure

Step 1: Open PKI for Key Generation

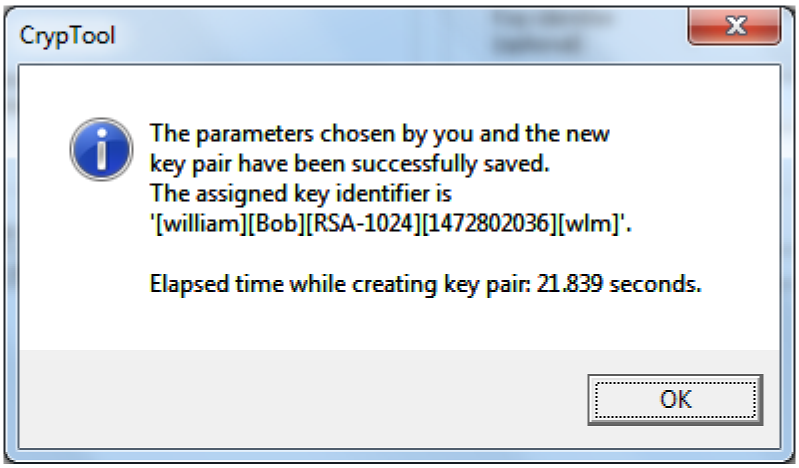


Step 2: Filled up User data

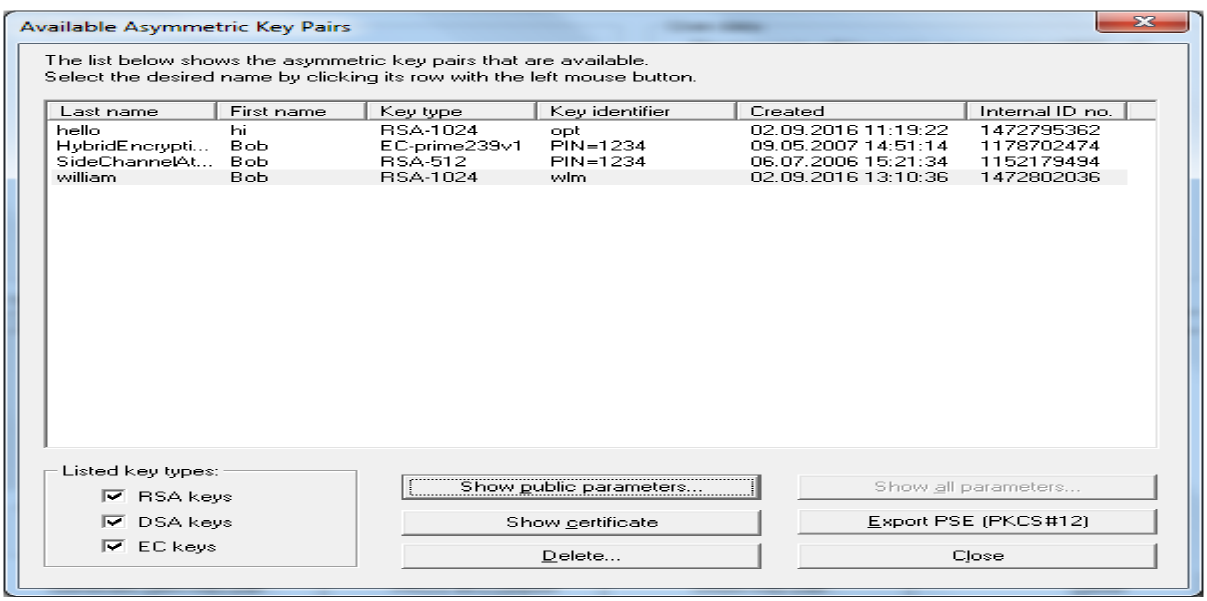




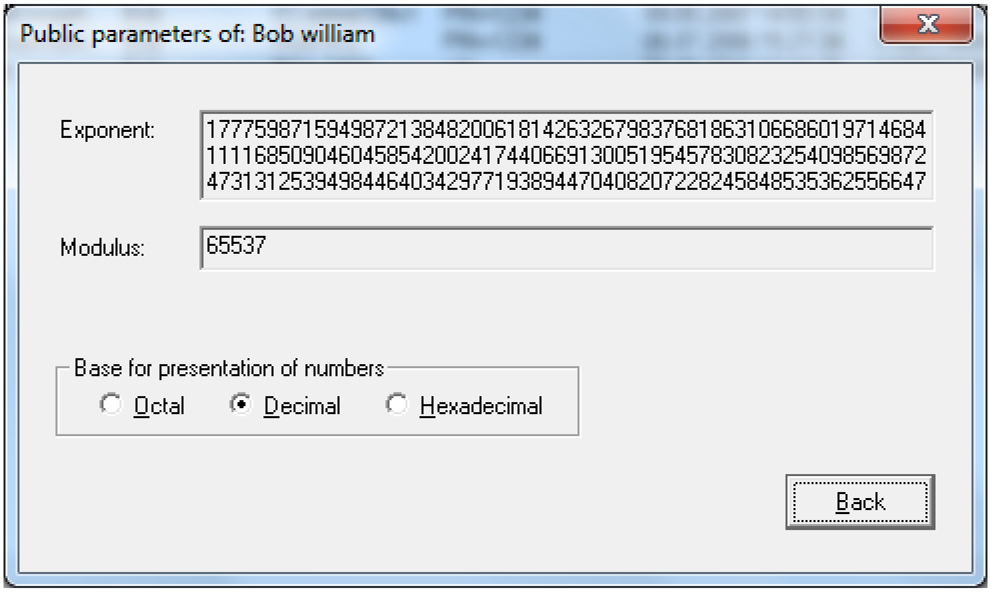
Step 3: Generating New Key Pair



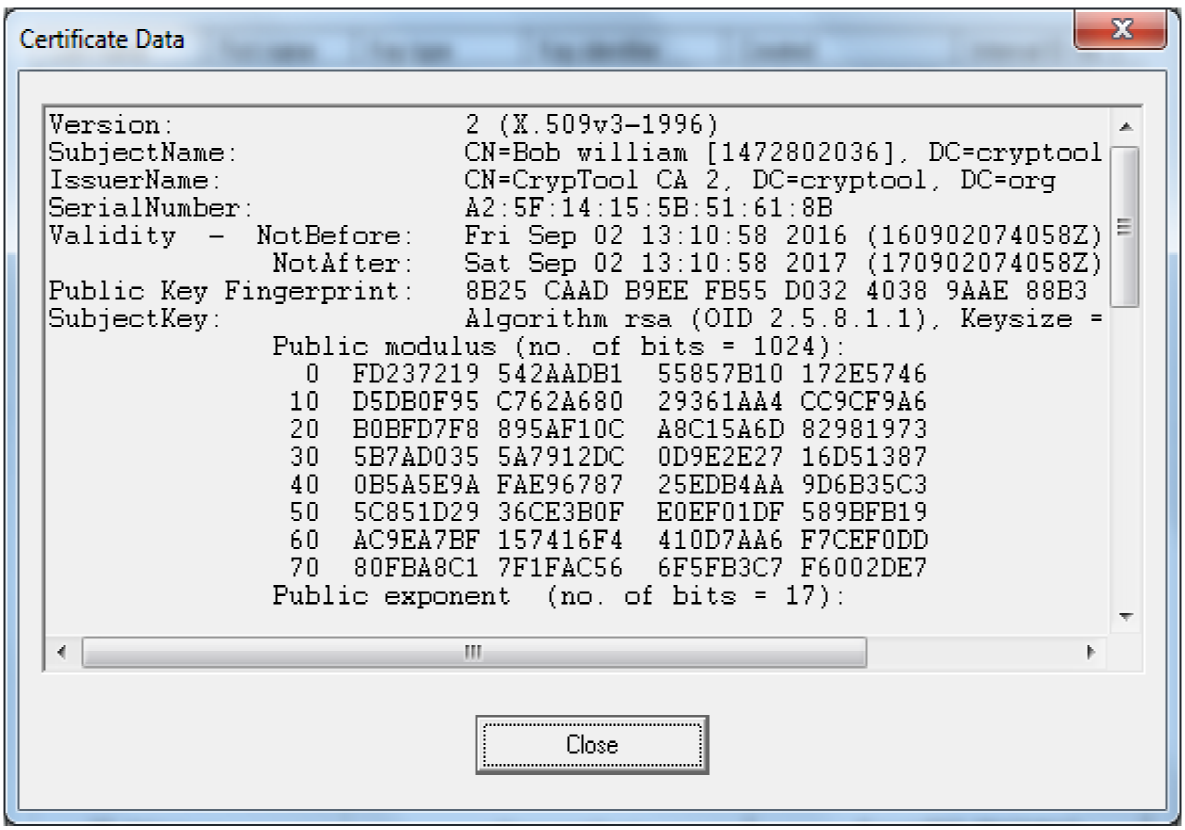
Step 4 : Show Available Asymmetric Key Pairs



Step 5 : Show the Public Parameters of Key Value



Step 6 : Certificate Generation



PRACTICAL- 10

OBJECTIVE:

Study and use the Wireshark for the various network protocols.

THEORY:

A packet sniffer, sometimes referred to as a network monitor or network analyzer, can be used by a network or system administrator to monitor and troubleshoot network traffic. Using the information captured by the packet sniffer an administrator can identify erroneous packets and use the data to pinpoint bottlenecks and help maintain efficient network data transmission.

In its simple form a packet sniffer simply captures all of the packets of data that pass through a given network interface. By placing a packet sniffer on a network in promiscuous mode, a Malicious intruder can capture and analyze all of the network traffic. Wireshark is a network packet analyzer. A network packet analyzer will try to capture network packets and tries to display that packet data as detailed as possible. Download and install wireshark network analyzer.