

PRACTICAL-1

Aim: Implement Caesar Cipher Encryption-Decryption Algorithm.

Explanation:

- The Caesar Cipher technique is one of the earliest and simplest method of encryption technique.
- It's simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter some fixed number of positions 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 apparently named after Julius Caesar, who apparently used it to communicate with his officials.
- Thus to cipher a given text we need an integer value, known as shift which indicates the number of position each letter of the text has been moved down.

Expression:

$$E_n(X)=(X+N) \bmod 26$$

$$D_n(X)=(X-N) \bmod 26$$

Where n=key and x=text.

Code:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void encryption(char [],int);
void decryption(char [],int);
void encryption(char msg[],int key){
    printf("\nCipher Text is:");
    int i;
    for(i=0;i<strlen(msg);i++) {
        if(isupper(msg[i]))
            {msg[i]=((msg[i]+key-65)%26)+65;}
        else
            {msg[i]=((msg[i]+key-97)%26)+97;} }
    puts(msg);
}
void decryption(char cipher_text[],int key){
    printf("\nDecrypted Text is:");
```

```
int i;
for(i=0;i<strlen(cipher_text);i++) {
    if(isupper(cipher_text[i]))
        { cipher_text[i]=((cipher_text[i]-key-65)%26)+65;}
    else
        { cipher_text[i]=((cipher_text[i]-key-97)%26)+97;} }
puts(cipher_text); }
int main(){
    char msg[30];
    int key;
    clrscr();
    printf("Enter plain text:");
    gets(msg);
    printf("Enter key:");
    scanf("%d",&key);
    encryption(msg,key);
    decryption(msg,key);
    getch();
    return 0;
}
```

Output:

```
Enter plain text:demo
Enter key:3

Cipher text is:ghpr
Decrypted text is:demo
```

PRACTICAL-2

Aim: Implement Monoalphabetic Cipher Encryption And Decryption Algorithm.

Explanation:

- Monoalphabetic cipher is a substitution cipher in which for a given key, the cipher alphabet for each plain alphabet is fixed throughout the encryption process.
- For example, if 'A' is encrypted as 'D', for any number of occurrence in that plain text, 'A' will always get encrypted to 'D'.
- There are many different monoalphabetic substitution ciphers, in fact infinitely many, as each letter can be encrypted to any symbol, not just another letter.

Code:

```
#include<stdio.h>

#include<string.h>

#include<conio.h>

char pt[30],c[27],ct[30];

int i,j,index;

void encrypt(char ct[],char c[]);

void decrypt(char pt[],char c[]);

int main(){

printf("enter your plaintext:");

gets(pt);

printf("enter your key:");

for(i=0;i<26;i++)

{

printf("%c-",i+97);

c[i]=getch();

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

}

for(i=0;i<strlen(pt);i++)

{

index=pt[i]-97;

ct[i]=c[index];

}

encrypt(ct,c);
```

```

decrypt(pt,c);

return 0;

}

void encrypt(char ct[], char c[]){

printf("\n\ncipher Text is : ");

for(i=0;i<strlen(pt);i++)

{ printf("%c",ct[i]);}

for(i=0;i<strlen(pt);i++)

{ ct[i]=pt[i];}

}

void decrypt(char pt[], char c[]) {

printf("\n\nPlain Text is : ");

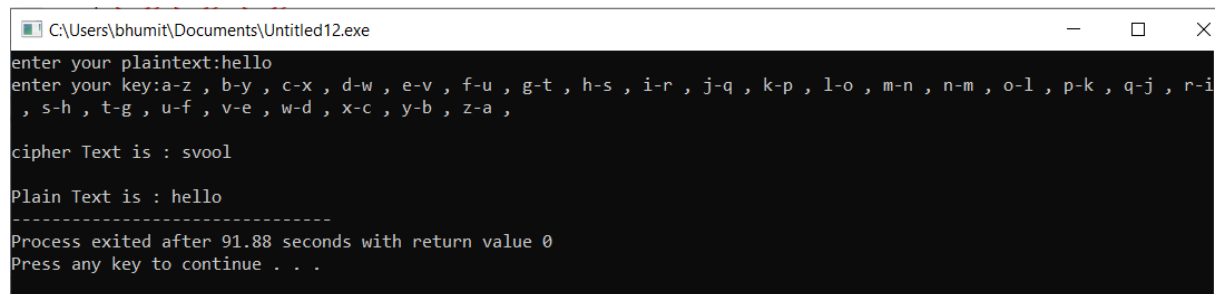
for(i=0;i<strlen(pt);i++)

{ printf("%c",ct[i]);}

}

```

Output:

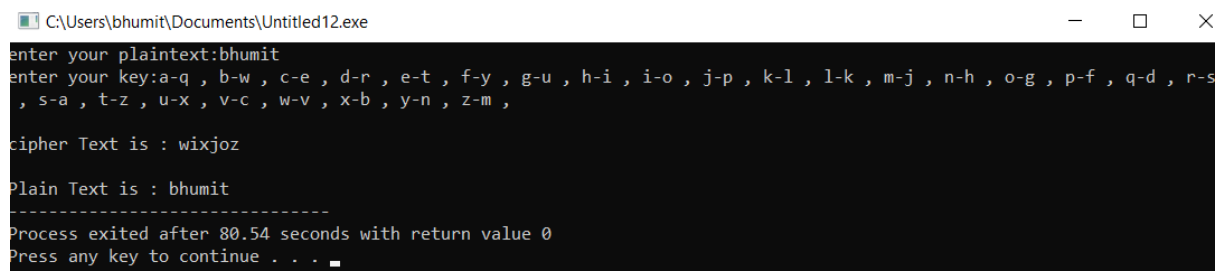


```

C:\Users\bhumit\Documents\Untitled12.exe
enter your plaintext:hello
enter your key:a-z , b-y , c-x , d-w , e-v , f-u , g-t , h-s , i-r , j-q , k-p , l-o , m-n , n-m , o-l , p-k , q-j , r-i
, s-h , t-g , u-f , v-e , w-d , x-c , y-b , z-a ,

cipher Text is : svoid
Plain Text is : hello
-----
Process exited after 91.88 seconds with return value 0
Press any key to continue . . .

```



```

C:\Users\bhumit\Documents\Untitled12.exe
enter your plaintext:bhumit
enter your key:a-q , b-w , c-e , d-r , e-t , f-y , g-u , h-i , i-o , j-p , k-l , l-k , m-j , n-h , o-g , p-f , q-d , r-s
, s-a , t-z , u-x , v-c , w-v , x-b , y-n , z-m ,

cipher Text is : wixjoz
Plain Text is : bhumit
-----
Process exited after 80.54 seconds with return value 0
Press any key to continue . . .

```

PRACTICAL-3

Aim : Write a C Program For Encryption And Decryption Of Playfair Cipher.

Introduction:

- 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.
- Consider ways to reduce the “spikyness” of natural language text, since if just map one letter always to another, the frequency distribution is just shuffled. One approach is to encrypt more than 1 letter at once. The playfair cipher is an example of doing this, treats digrams in the plaintext as single units and translates these units into cipher digrams.

Encryption and decryption:

- Plaintext is encrypted two letters at a time.
 - If a pair is repeated letter, insert filler like ‘X’.
 - If both letters fall in the same row, replace each with letter to right.
 - If both letters fall in the same column, replace each with letter below it.
 - Otherwise each letter is replaced by the letter in the same row and in the column of the other letter of the pair.
- Decryption of course works exactly in reverse.

Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define SIZE 30

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);
}

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);
}

int main() {
    char str[SIZE], key[SIZE], cipher[SIZE];
    printf("Enter plain text:");gets(str);
    printf("Enter key:");gets(key);
    encryptByPlayfairCipher(str, key);
    printf("Cipher text: %s\n", str);
    decryptByPlayfairCipher(str,key);
    printf("Deciphered text: %s\n",str);
    return 0;
}
```

Output:

```
C:\Users\bhumi\Desktop\playfair_gfg.exe
Enter plain text:instruments
Enter key:monarchy
Cipher text: gatlmzclrqtx
Deciphered text: inskrumentsz

-----
Process exited after 19.48 seconds with return value 0
Press any key to continue . . .
```

PRACTICAL-4

Aim: Write a C program to implement encryption and decryption of Polyalphabetic Cipher(Vigenere).

Introduction:

- The best known and one of the simplest such algorithms is referred to as the Vigenere cipher, where the set of related monoalphabetic substitution rules consists of the 26 Caesar ciphers, with shifts of 0 through 25. Each cipher is denoted by a key letter, which is the ciphertext letter that substitutes for the plaintext letter 'a' and which are used in turn, as shown next.
- It makes cryptanalysis harder with more alphabets to guess and flatter frequency distribution. But the Vigenere cipher still do not completely obscure the underlying language characteristics.

Code:

```
#include<stdio.h>

#include<string.h>

int main(){

    char msg[100], key[20];

    fflush(stdin);

    printf("Enter the plain text : ");

    gets(msg);

    printf("Enter Key : ");

    gets(key);

    int msgLen=0, keyLen=0, i=0, j=0;

    msgLen = strlen(msg);

    keyLen = strlen(key);

    printf("%d \n", msgLen);

    char newKey[msgLen], encryptedMsg[msgLen], decryptedMsg[msgLen];

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

        if(j == keyLen)

            j = 0;

        newKey[i] = key[j];

    }

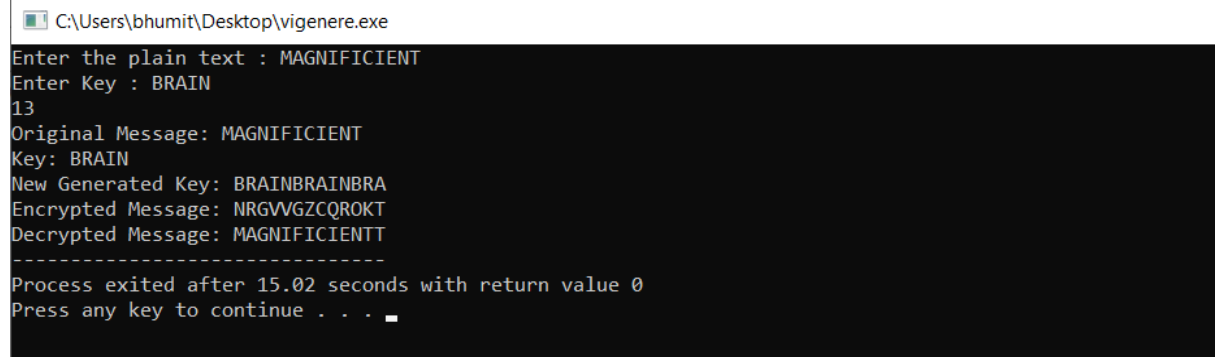
    newKey[i] = '\0';
```

```
//encryption
for(i = 0; i < msgLen; ++i){
    encryptedMsg[i] = ((msg[i] + newKey[i]) % 26) + 'A';
}
encryptedMsg[i] = '\0';

//decryption
for(i = 0; i < msgLen; ++i){
    decryptedMsg[i] = (((encryptedMsg[i] - newKey[i]) + 26) % 26) + 'A';
}
decryptedMsg[i] = '\0';
printf("Original Message: %s", msg);
printf("\nKey: %s", key);
printf("\nNew Generated Key: %s", newKey);
printf("\nEncrypted Message: %s", encryptedMsg);
printf("\nDecrypted Message: %s", decryptedMsg);

return 0;
}
```

Output:



C:\Users\bhumit\Desktop\vigenere.exe

```
Enter the plain text : MAGNIFICIENT
Enter Key : BRAIN
13
Original Message: MAGNIFICIENT
Key: BRAIN
New Generated Key: BRAINBRAINBRA
Encrypted Message: NRGVVGZCQROKT
Decrypted Message: MAGNIFICIENTT
-----
Process exited after 15.02 seconds with return value 0
Press any key to continue . . .
```

PRACTICAL-5

Aim: Write a C program for encryption and decryption of Hill Cipher.

Introduction:

- Hill cipher is a polygraphic substitution cipher based on linear algebra. Each letter is represented by a number modulo 26. Often the simple scheme A = 0, B = 1, ..., Z = 25 is used, but this is not an essential feature of the cipher. To encrypt a message, each block of n letters (considered as an n-component vector) is multiplied by an invertible $n \times n$ matrix, against modulus 26. To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption.
- The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible $n \times n$ matrices (modulo 26).

Code:

```
#include<stdio.h>

#include<math.h>

float encrypt[3][1], decrypt[3][1], a[3][3], b[3][3], mes[3][1], c[3][3];

void encryption();    //encrypts the message

void decryption();    //decrypts the message

void getKeyMessage();    //gets key and message from user

void inverse();        //finds inverse of key matrix

int main() {

    getKeyMessage();

    encryption();

    decryption();  }

void encryption() {

    int i, j, k;

    for(i = 0; i < 3; i++)

        for(j = 0; j < 1; j++)

            for(k = 0; k < 3; k++)

                encrypt[i][j] = encrypt[i][j] + a[i][k] * mes[k][j];
```

```
printf("\nEncrypted string is: ");

for(i = 0; i < 3; i++)

    printf("%c", (char)(fmod(encrypt[i][0], 26) + 97));  }

void decryption() {

    int i, j, k;

    inverse();

    for(i = 0; i < 3; i++)

        for(j = 0; j < 1; j++)

            for(k = 0; k < 3; k++)

                decrypt[i][j] = decrypt[i][j] + b[i][k] * encrypt[k][j];

    printf("\nDecrypted string is: ");

    for(i = 0; i < 3; i++)

        printf("%c", (char)(fmod(decrypt[i][0], 26) + 97));

    printf("\n");

}

void getKeyMessage() {

    int i, j;

    char msg[3];

    printf("Enter 3x3 matrix for key (It should be inversible):\n");

    for(i = 0; i < 3; i++)

        for(j = 0; j < 3; j++) {

            scanf("%f", &a[i][j]);

            c[i][j] = a[i][j];

        }

    printf("\nEnter a 3 letter string: ");

    scanf("%s", msg);

    for(i = 0; i < 3; i++)
```

```
        mes[i][0] = msg[i] - 97;

    }

void inverse() {

    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 = c[i][k];

            q = c[k][k];

            for(j = 0; j < 3; j++) {

                if(i != k) {

                    c[i][j] = c[i][j]*q - p*c[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] / c[i][i];

    printf("\n\nInverse Matrix is:\n");

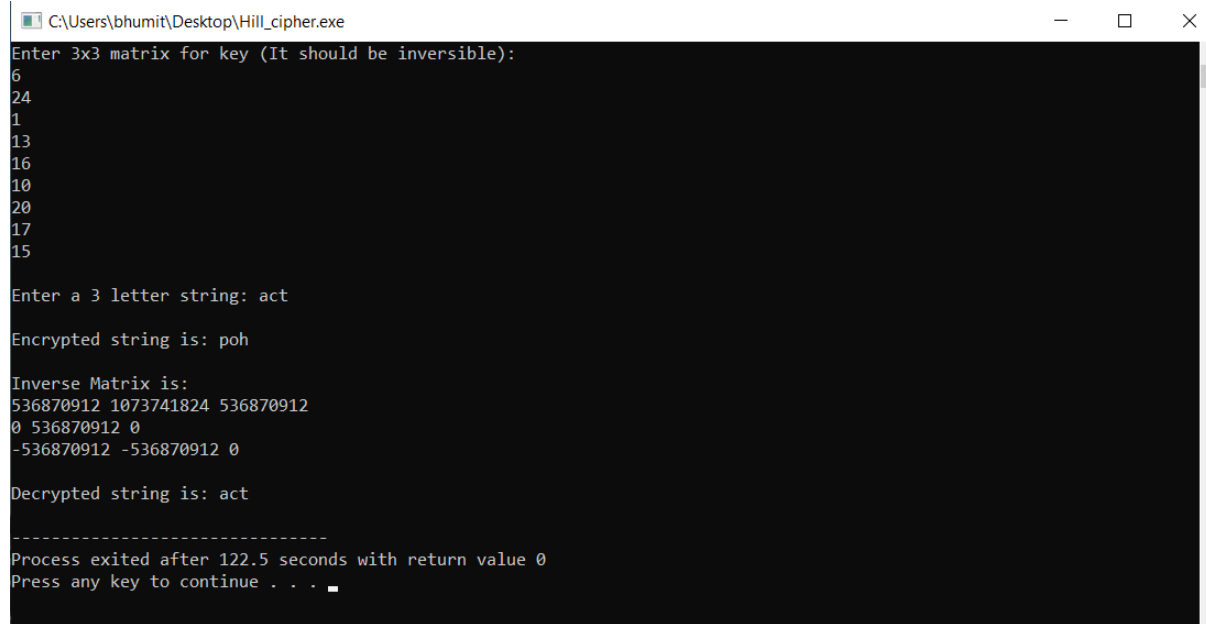
    for(i = 0; i < 3; i++) {

        for(j = 0; j < 3; j++)

            printf("%d ", b[i][j]);
```

```
printf("\n"); } }
```

Output:



```
C:\Users\bhumit\Desktop\Hill_cipher.exe
Enter 3x3 matrix for key (It should be inversible):
6
24
1
13
16
10
20
17
15

Enter a 3 letter string: act

Encrypted string is: poh

Inverse Matrix is:
536870912 1073741824 536870912
0 536870912 0
-536870912 -536870912 0

Decrypted string is: act

-----
Process exited after 122.5 seconds with return value 0
Press any key to continue . . .
```


PRACTICAL-6

Aim: Write a C program to implement simple DES.

Introduction:

- The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST).
- DES is an implementation of a Feistel Cipher. It uses 16 round Feistel structure. The block size is 64-bit. Though, key length is 64-bit, DES has an effective key length of 56 bits, since 8 of the 64 bits of the key are not used by the encryption algorithm (function as check bits only).
- Since DES is based on the Feistel Cipher, all that is required to specify DES is :
 - Round function
 - Key schedule
 - Any additional processing – Initial and final permutation
- The DES satisfies both the desired properties of block cipher. These two properties make cipher very strong.
 - **Avalanche effect** – A small change in plaintext results in the very great change in the ciphertext.
 - **Completeness** – Each bit of ciphertext depends on many bits of plaintext.
- During the last few years, cryptanalysis have found some weaknesses in DES when key selected are weak keys. These keys shall be avoided.
- DES has proved to be a very well designed block cipher. There have been no significant cryptanalytic attacks on DES other than exhaustive key search.

Code:

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

```
int Original_key [64] = { // you can change key if required
```

```
0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0,
```

```
0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1,
```

```
1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0,
```

```
1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1 };
```

```
int Permuted_Choice1[56] = {
```

```
57, 49, 41, 33, 25, 17, 9,
```

```
1, 58, 50, 42, 34, 26, 18,
```

```
10, 2, 59, 51, 43, 35, 27,
```

```
19, 11, 3, 60, 52, 44, 36,
```

```
63, 55, 47, 39, 31, 23, 15,
```

```
7, 62, 54, 46, 38, 30, 22,
```

```
14, 6, 61, 53, 45, 37, 29,
```

```
21, 13, 5, 28, 20, 12, 4};  
  
int Permuted_Choice2[48] = {  
    14, 17, 11, 24, 1, 5,  
    3, 28, 15, 6, 21, 10,  
    23, 19, 12, 4, 26, 8,  
    16, 7, 27, 20, 13, 2,  
    41, 52, 31, 37, 47, 55,  
    30, 40, 51, 45, 33, 48,  
    44, 49, 39, 56, 34, 53,  
    46, 42, 50, 36, 29, 32 };  
  
int Initial_Permutation [64] = {  
    58, 50, 42, 34, 26, 18, 10, 2,  
    60, 52, 44, 36, 28, 20, 12, 4,  
    62, 54, 46, 38, 30, 22, 14, 6,  
    64, 56, 48, 40, 32, 24, 16, 8,  
    57, 49, 41, 33, 25, 17, 9, 1,  
    59, 51, 43, 35, 27, 19, 11, 3,  
    61, 53, 45, 37, 29, 21, 13, 5,  
    63, 55, 47, 39, 31, 23, 15, 7 };  
  
int Final_Permutation[] = {  
    40, 8, 48, 16, 56, 24, 64, 32,  
    39, 7, 47, 15, 55, 23, 63, 31,  
    38, 6, 46, 14, 54, 22, 62, 30,  
    37, 5, 45, 13, 53, 21, 61, 29,  
    36, 4, 44, 12, 52, 20, 60, 28,  
    35, 3, 43, 11, 51, 19, 59, 27,  
    34, 2, 42, 10, 50, 18, 58, 26,  
    33, 1, 41, 9, 49, 17, 57, 25 };  
  
int P[] = {  
    16, 7, 20, 21,  
    29, 12, 28, 17,  
    1, 15, 23, 26,
```

```
5, 18, 31, 10,  
2, 8, 24, 14,  
32, 27, 3, 9,  
19, 13, 30, 6,  
22, 11, 4, 25 };
```

```
int E[] = {  
32, 1, 2, 3, 4, 5,  
4, 5, 6, 7, 8, 9,  
8, 9, 10, 11, 12, 13,  
12, 13, 14, 15, 16, 17,  
16, 17, 18, 19, 20, 21,  
20, 21, 22, 23, 24, 25,  
24, 25, 26, 27, 28, 29,  
28, 29, 30, 31, 32, 1 };
```

```
int S1[4][16] = {  
14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,  
0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,  
4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,  
15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13};
```

```
int S2[4][16] = {  
15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,  
3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,  
0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,  
13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9};
```

```
int S3[4][16] = {  
10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,  
13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,  
13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,  
1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12};
```

```
int S4[4][16] = {  
7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,  
13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,
```

```
10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,
3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14};

int S5[4][16] = {
    2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,
    14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
    4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
    11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3};

int S6[4][16] = {
    12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,
    10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,
    9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,
    4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13};

int S7[4][16]= {
    4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,
    13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,
    1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,
    6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12};

int S8[4][16]= {
    13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,
    1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,
    7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
    2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11};

int shifts_for_each_round[16] = { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 };

int _56bit_key[56];
int _48bit_key[17][48];
int text_to_bits[99999], bits_size=0;
int Left32[17][32], Right32[17][32];
int EXPtext[48];
int XORtext[48];
int X[8][6];
int X2[32];
int R[32];
```

```
int chiper_text[64];
int encrypted_text[64];
int XOR(int a, int b) { return (a ^ b);}
void Dec_to_Binary(int n) {
    int binaryNum[1000];
    int i = 0;
    while (n > 0) {
        binaryNum[i] = n % 2;
        n = n / 2;
        i++; }
    for (int j = i - 1; j >= 0; j--) {
        text_to_bits[bits_size++] = binaryNum[j];
    }
}
int F1(int i){
    int r, c, b[6];
    for (int j = 0; j < 6; j++)
        b[j] = X[i][j];
    r = b[0] * 2 + b[5];
    c = 8 * b[1] + 4 * b[2] + 2 * b[3] + b[4];
    if (i == 0)
        return S1[r][c];
    else if (i == 1)
        return S2[r][c];
    else if (i == 2)
        return S3[r][c];
    else if (i == 3)
        return S4[r][c];
    else if (i == 4)
        return S5[r][c];
    else if (i == 5)
        return S6[r][c];
}
```

```
        else if (i == 6)
            return S7[r][c];
        else if (i == 7)
            return S8[r][c];}
int PBox(int pos, int bit){
    int i;
    for (i = 0; i < 32; i++)
        if (P[i] == pos + 1)
            break;

    R[i] = bit;}
int ToBits(int value){
    int k, j, m;
    static int i;
    if (i % 32 == 0)
        i = 0;
    for (j = 3; j >= 0; j--)
    {
        m = 1 << j;
        k = value & m;
        if (k == 0)
            X2[3 - j + i] = '0' - 48;
        else
            X2[3 - j + i] = '1' - 48;
    }
    i = i + 4;
}
int SBox(int XORtext[]){
    int k = 0;
    for (int i = 0; i < 8; i++)
        for (int j = 0; j < 6; j++)
            X[i][j] = XORtext[k++];

    int value;
```

```

        for (int i = 0; i < 8; i++)
        {
            value = F1(i);
            ToBits(value);} }

void expansion_function(int pos, int bit){
    for (int i = 0; i < 48; i++)
        if (E[i] == pos + 1)
            EXPtext[i] = bit; }

void cipher(int Round, int mode){
    for (int i = 0; i < 32; i++)
        expansion_function(i, Right32[Round - 1][i]);
    for (int i = 0; i < 48; i++)
    {
        if (mode == 0)
            XORtext[i] = XOR(EXPtext[i], _48bit_key[Round][i]);
        else
            XORtext[i] = XOR(EXPtext[i], _48bit_key[17 - Round][i]);
    }
    SBox(XORtext);
    for (int i = 0; i < 32; i++)
        PBox(i, X2[i]);
    for (int i = 0; i < 32; i++)
        Right32[Round][i] = XOR(Left32[Round - 1][i], R[i]);
}

void finalPermutation(int pos, int bit){
    int i;
    for (i = 0; i < 64; i++)
        if (Final_Permutation[i] == pos + 1)
            break;
    encrypted_text[i] = bit;}

void Encrypt_each_64_bit (int plain_bits []){
    int IP_result [64] , index=0;

```

```

    for (int i = 0; i < 64; i++) {
        IP_result[i] = plain_bits[ Initial_Permutation[i] ];
    }
    for (int i = 0; i < 32; i++)
        Left32[0][i] = IP_result[i];
    for (int i = 32; i < 64; i++)
        Right32[0][i - 32] = IP_result[i];
    for (int k = 1; k < 17; k++)
    { // processing through all 16 rounds
        cipher(k, 0);
        for (int i = 0; i < 32; i++)
            Left32[k][i] = Right32[k - 1][i]; // right part comes as it is to next round left
part}
    for (int i = 0; i < 64; i++)
    { // 32bit swap as well as Final Inverse Permutation
        if (i < 32)
            cipher_text[i] = Right32[16][i];
        else
            cipher_text[i] = Left32[16][i - 32];
        finalPermutation(i, cipher_text[i]);
    }
    for (int i = 0; i < 64; i++)
        printf("%d", encrypted_text[i]);}
void convert_Text_to_bits(char *plain_text){
    for(int i=0;plain_text[i];i++){
        int asci = plain_text[i];
        Dec_to_Binary(asci);
    } }
void key56to48(int round, int pos, int bit)
{
    int i;
    for (i = 0; i < 56; i++)
        if (Permuted_Choice2[i] == pos + 1)

```



```
        break;

    _48bit_key[round][i] = bit;
}

int key64to56(int pos, int bit)
{
    int i;
    for (i = 0; i < 56; i++)
        if (Permuted_Choice1[i] == pos + 1)
            break;
    _56bit_key[i] = bit;
}

void key64to48(int key[])
{
    int k, backup[17][2];
    int CD[17][56];
    int C[17][28], D[17][28];
    for (int i = 0; i < 64; i++)
        key64to56(i, key[i]);
    for (int i = 0; i < 56; i++)
        if (i < 28)
            C[0][i] = _56bit_key[i];
        else
            D[0][i - 28] = _56bit_key[i];
    for (int x = 1; x < 17; x++)
    {
        int shift = shifts_for_each_round[x - 1];
        for (int i = 0; i < shift; i++)
            backup[x - 1][i] = C[x - 1][i];
        for (int i = 0; i < (28 - shift); i++)
            C[x][i] = C[x - 1][i + shift];
        k = 0;
        for (int i = 28 - shift; i < 28; i++)
```

```

        C[x][i] = backup[x - 1][k++];
    for (int i = 0; i < shift; i++)
        backup[x - 1][i] = D[x - 1][i];
    for (int i = 0; i < (28 - shift); i++)
        D[x][i] = D[x - 1][i + shift];
    k = 0;
    for (int i = 28 - shift; i < 28; i++)
        D[x][i] = backup[x - 1][k++]; }
for (int j = 0; j < 17; j++) {
    for (int i = 0; i < 28; i++)
        CD[j][i] = C[j][i];
    for (int i = 28; i < 56; i++)
        CD[j][i] = D[j][i - 28];}
for (int j = 1; j < 17; j++)
    for (int i = 0; i < 56; i++)
        key56to48(j, i, CD[j][i]);}

int main(){
    char plain_text[100];
    printf("Enter plain text:");gets(plain_text);
    convert_Text_to_bits(plain_text);
    key64to48(Original_key); // it creates all keys for all rounds
    int _64bit_sets = bits_size/64;
    printf("Decrypted output is\n");
    for(int i=0;i<= _64bit_sets ;i++) {
        Encrypt_each_64_bit (text_to_bits + 64*i);}
    return 0; }

```

Output:

```

C:\Users\bhumit\Desktop\DES.exe
Enter plain text:This is demo of cipher
Decrypted output is
1000001011111000110011001100111011111000001100110000110110000001101101010010000101100111111100100101011000010001000111
110001111011100111111111001100110011000011110000100000101001010010110000011
-----
Process exited after 12.19 seconds with return value 0
Press any key to continue . . .

```

PRACTICAL-7

Aim: Write a C program to implement Diffie Hellman Key Exchange Algorithm.

Introduction:

- Diffie Hellman (DH) key exchange algorithm is a method for securely exchanging cryptographic keys over a public communications channel. Keys are not actually exchanged – they are jointly derived. It is named after their inventors Whitfield Diffie and Martin Hellman.
- If Alice and Bob wish to communicate with each other, they first agree between them a large prime number p , and a generator (or base) g (where $0 < g < p$).
- Alice chooses a secret integer a (her private key) and then calculates $g^a \bmod p$ (which is her public key). Bob chooses his private key b , and calculates his public key in the same way.
- Bob knows b and g^a , so he can calculate $(g^a)^b \bmod p = g^{ab} \bmod p$. Therefore both Alice and Bob know a shared secret $g^{ab} \bmod p$. An eavesdropper Eve who was listening in on the communication knows p , g , Alice's public key ($g^a \bmod p$) and Bob's public key ($g^b \bmod p$). She is unable to calculate the shared secret from these values.
- In static-static mode, both Alice and Bob retain their private/public keys over multiple communications. Therefore the resulting shared secret will be the same every time. In ephemeral-static mode one party will generate a new private/public key every time, thus a new shared secret will be generated.

Code:

```
#include<stdio.h>

long int power(int a,int b,int mod) {

    long long int t;

    if(b==1)

        return a;

    t=power(a,b/2,mod);

    if(b%2==0)

        return (t*t)%mod;

    else

        return (((t*t)%mod)*a)%mod; }

long long int calculateKey(int a,int x,int n) {

    return power(a,x,n); }
```

```
int main(){

    int n,g,x,a,y,b;

    // both the persons will be agreed upon the common n and g

    printf("Enter the value of n and g : ");

    scanf("%d%d",&n,&g);

    // first person will choose the x

    printf("Enter the value of x for the first person : ");

    scanf("%d",&x); a=power(g,x,n);

    // second person will choose the y

    printf("Enter the value of y for the second person : ");

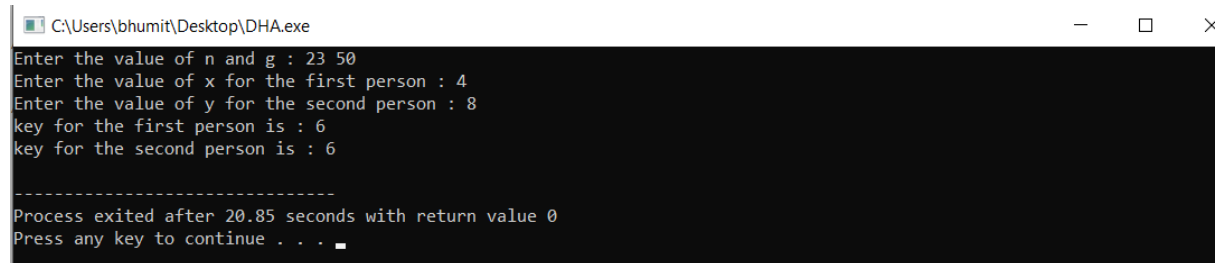
    scanf("%d",&y); b=power(g,y,n);

    printf("key for the first person is : %lld\n",power(b,x,n));

    printf("key for the second person is : %lld\n",power(a,y,n));

    return 0; }
```

Output:



```
C:\Users\bhumit\Desktop\DHA.exe
Enter the value of n and g : 23 50
Enter the value of x for the first person : 4
Enter the value of y for the second person : 8
key for the first person is : 6
key for the second person is : 6

-----
Process exited after 20.85 seconds with return value 0
Press any key to continue . . .
```

PRACTICAL-8

Aim: Write a C program to implement RSA encryption and decryption algorithm.

Introduction:

- RSA algorithm is a public key encryption technique and is considered as the most secure way of encryption. It was invented by Rivest, Shamir and Adleman in year 1978 and hence name **RSA** algorithm.
- The RSA algorithm holds the following features –
 - RSA algorithm is a popular exponentiation in a finite field over integers including prime numbers.
 - The integers used by this method are sufficiently large making it difficult to solve.
 - There are two sets of keys in this algorithm: private key and public key.
- You will have to go through the following steps to work on RSA algorithm –

Step 1: Generate the RSA modulus

- The initial procedure begins with selection of two prime numbers namely p and q , and then calculating their product N , as shown –
- $N = p * q$
- Here, let N be the specified large number.

Step 2: Derived Number (e)

- Consider number e as a derived number which should be greater than 1 and less than $(p-1)$ and $(q-1)$. The primary condition will be that there should be no common factor of $(p-1)$ and $(q-1)$ except 1.

Step 3: Public key

- The specified pair of numbers n and e forms the RSA public key and it is made public.

Step 4: Private Key

- Private Key d is calculated from the numbers p , q and e . The mathematical relationship between the numbers is as follows –
- $ed = 1 \text{ mod } (p-1)(q-1)$
- The above formula is the basic formula for Extended Euclidean Algorithm, which takes p and q as the input parameters.

• Encryption Formula

- Consider a sender who sends the plain text message to someone whose public key is (n,e) . To encrypt the plain text message in the given scenario, use the following syntax –
- $C = P^e \text{ mod } n$

• Decryption Formula

- The decryption process is very straightforward and includes analytics for calculation in a systematic approach. Considering receiver **C** has the private key **d**, the result modulus will be calculated as –
- $\text{Plaintext} = C^d \bmod n$

Code:

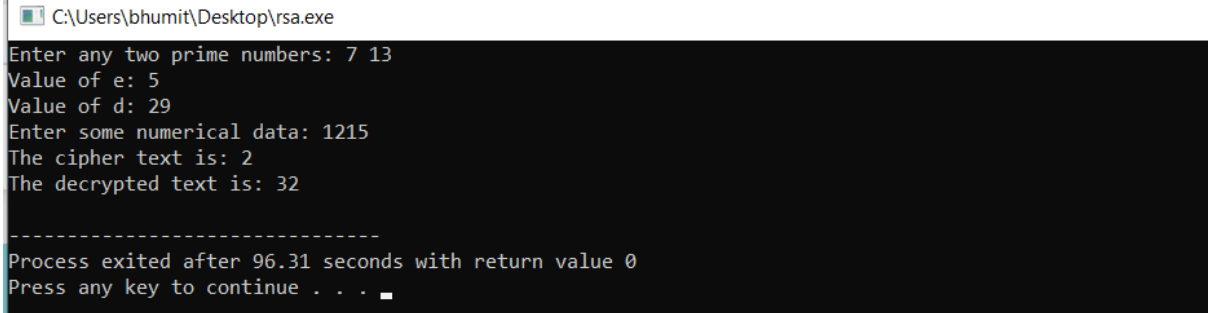
```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int checkPrime(int n) {
    int i;
    int m = n / 2;

    for (i = 2; i <= m; i++) {
        if (n % i == 0) {
            return 0; // Not Prime
        }
    }
    return 1; // Prime }
int findGCD(int n1, int n2) {
    int i, gcd;
    for(i = 1; i <= n1 && i <= n2; ++i) {
        if(n1 % i == 0 && n2 % i == 0)
            gcd = i;
    }
    return gcd; }
int powMod(int a, int b, int n) {
    long long x = 1, y = a;
    while (b > 0) {
        if (b % 2 == 1)
            x = (x * y) % n;
        y = (y * y) % n; // Squaring the base
        b /= 2;
    }

    return x % n; }
int main(int argc, char* argv[]) {
    int p, q;
    int n, phin;
    int data, cipher, decrypt;
    while (1) {
        printf("Enter any two prime numbers: ");
        scanf("%d %d", &p, &q);
        if (!(checkPrime(p) && checkPrime(q)))
            printf("Both numbers are not prime. Please enter prime numbers
only...\n");
        else if (!checkPrime(p))
            printf("The first prime number you entered is not prime, please try
again...\n");
        else if (!checkPrime(q))
            printf("The second prime number you entered is not prime, please try
again...\n");
        else
```

```
                break; }
n = p * q;
phin = (p - 1) * (q - 1);
int e = 0;
for (e = 5; e <= 100; e++) {
    if (findGCD(phin, e) == 1)
        break; }
int d = 0;
for (d = e + 1; d <= 100; d++) {
    if ( ((d * e) % phin) == 1)
        break; }
printf("Value of e: %d\nValue of d: %d\n", e, d);
printf("Enter some numerical data: ");
scanf("%d", &data);
cipher = powMod(data, e, n);
printf("The cipher text is: %d\n", cipher);
decrypt = powMod(cipher, d, n);
printf("The decrypted text is: %d\n", decrypt);
return 0; }
```

Output:



```
C:\Users\bhumit\Desktop\rsa.exe
Enter any two prime numbers: 7 13
Value of e: 5
Value of d: 29
Enter some numerical data: 1215
The cipher text is: 2
The decrypted text is: 32

-----
Process exited after 96.31 seconds with return value 0
Press any key to continue . . .
```

PRACTICAL-9

Aim: Write a program to implement SHA1 Algorithm.

Introduction:

- SHA-1 or Secure Hash Algorithm 1 is a cryptographic hash function which takes an input and produces a 160-bit (20-byte) hash value. This hash value is known as a message digest. This message digest is usually then rendered as a hexadecimal number which is 40 digits long. It is a U.S. Federal Information Processing Standard and was designed by the United States National Security Agency.
- SHA-1 is now considered insecure since 2005. Major tech giants browsers like Microsoft, Google, Apple and Mozilla have stopped accepting SHA-1 SSL certificates by 2017.
- The different SHA hash functions are explained below.
 - **SHA256** : This hash function belong to hash class SHA-2, the internal block size of it is 32 bits.
 - **SHA384** : This hash function belong to hash class SHA-2, the internal block size of it is 32 bits. This is one of the truncated version.
 - **SHA224** : This hash function belong to hash class SHA-2, the internal block size of it is 32 bits. This is one of the truncated version.
 - **SHA512** : This hash function belong to hash class SHA-2, the internal block size of it is 64 bits.
 - **SHA1** : The 160 bit hash function that resembles MD5 hash in working and was discontinued to be used seeing its security vulnerabilities.
- Hashing transforms a string of characters into a shorter fixed length value which is sufficient to represent the original string. The hash obtained at the end of hashing process is combined with the original message and is used to ensure the integrity of the message. If the hash calculated at receiver end does not matches with the attached hash, the integrity of message has been compromised. Also, a layer of encryption over the packet (which has both Message + HMAC) can help to ensure message confidentiality, because if adversary somehow manages to get his hands on this packet, he still cannot make any sense out of it.

Code:

```
#include <iostream>

#include <string>

#include <sstream>

#include <iomanip>

#include <fstream>

#define SHA1_ROL(value, bits) (((value) << (bits)) | (((value) & 0xffffffff) >> (32 - (bits))))

#define SHA1_BLK(i) (block[i&15] = SHA1_ROL(block[(i+13)&15] ^ block[(i+8)&15] ^ block[(i+2)&15] ^ block[i&15],1))

#define SHA1_R0(v,w,x,y,z,i) z += ((w&(x^y))^y) + block[i] + 0x5a827999 + SHA1_ROL(v,5); w=SHA1_ROL(w,30);

#define SHA1_R1(v,w,x,y,z,i) z += ((w&(x^y))^y) + SHA1_BLK(i) + 0x5a827999 + SHA1_ROL(v,5); w=SHA1_ROL(w,30);
```



```

#define SHA1_R2(v,w,x,y,z,i) z += (w^x^y)          + SHA1_BLK(i) + 0x6ed9eba1 +
SHA1_ROL(v,5); w=SHA1_ROL(w,30);

#define SHA1_R3(v,w,x,y,z,i) z += (((w|x)&y)|(w&x)) + SHA1_BLK(i) + 0x8f1bbcdc +
SHA1_ROL(v,5); w=SHA1_ROL(w,30);

#define SHA1_R4(v,w,x,y,z,i) z += (w^x^y)          + SHA1_BLK(i) + 0xca62c1d6 +
SHA1_ROL(v,5); w=SHA1_ROL(w,30);

class SHA1{
public:
    SHA1();

    void update(const std::string &s);
    void update(std::istream &is);

    std::string final();

    static std::string from_file(const std::string &filename);

private:
    typedef unsigned long int uint32;
    typedef unsigned long long uint64;

    static const unsigned int DIGEST_INTS = 5;
    static const unsigned int BLOCK_INTS = 16;
    static const unsigned int BLOCK_BYTES = BLOCK_INTS * 4;

    uint32 digest[DIGEST_INTS];

    std::string buffer;
    uint64 transforms;

    void reset();

    void transform(uint32 block[BLOCK_BYTES]);

    static void buffer_to_block(const std::string &buffer, uint32 block[BLOCK_BYTES]);

    static void read(std::istream &is, std::string &s, int max);
};

std::string sha1(const std::string &string);

using namespace std;

int main(int argc, char *argv[])
{
    string str;

    cout<<"Enter word: ";

```

```
    cin>>str;

    cout << sha1(str) << endl;

    return 0;
}

SHA1::SHA1()
{
    reset();
}

void SHA1::update(const std::string &s)
{
    std::istringstream is(s);
    update(is);
}

void SHA1::update(std::istream &is)
{
    std::string rest_of_buffer;
    read(is, rest_of_buffer, BLOCK_BYTES - buffer.size());
    buffer += rest_of_buffer;
    while (is)
    {
        uint32 block[BLOCK_INTS];
        buffer_to_block(buffer, block);
        transform(block);
        read(is, buffer, BLOCK_BYTES);
    }
}

std::string SHA1::final()
{
    uint64 total_bits = (transforms*BLOCK_BYTES + buffer.size()) * 8;
    buffer += 0x80;
    unsigned int orig_size = buffer.size();
    while (buffer.size() < BLOCK_BYTES)
```

```

{
    buffer += (char)0x00;
}

uint32 block[BLOCK_INTS];
buffer_to_block(buffer, block);

if (orig_size > BLOCK_BYTES - 8)
{
    transform(block);
    for (unsigned int i = 0; i < BLOCK_INTS - 2; i++)
    {
        block[i] = 0; }
    }

    block[BLOCK_INTS - 1] = total_bits;
    block[BLOCK_INTS - 2] = (total_bits >> 32);
    transform(block);

    std::ostringstream result;
    for (unsigned int i = 0; i < DIGEST_INTS; i++)
    {
        result << std::hex << std::setfill('0') << std::setw(8);
        result << (digest[i] & 0xffffffff);
    }

    reset();

    return result.str();
}

std::string SHA1::from_file(const std::string &filename)
{
    std::ifstream stream(filename.c_str(), std::ios::binary);
    SHA1 checksum;
    checksum.update(stream);
    return checksum.final();
}

void SHA1::reset()

```

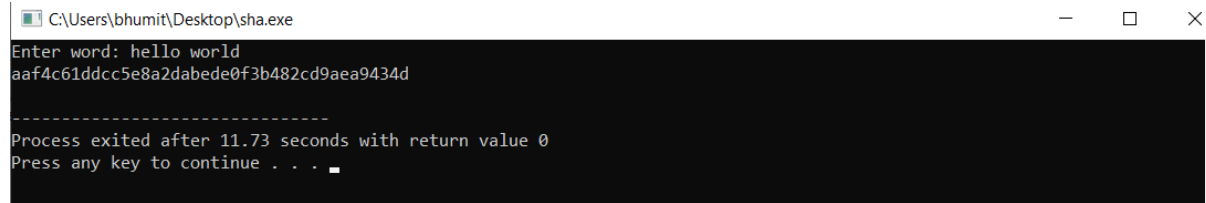
```
{
    digest[0] = 0x67452301;
    digest[1] = 0xefcdab89;
    digest[2] = 0x98badcfe;
    digest[3] = 0x10325476;
    digest[4] = 0xc3d2e1f0;
    transforms = 0;
    buffer = "";
}

void SHA1::transform(uint32 block[BLOCK_BYTES])
{
    uint32 a = digest[0];
    uint32 b = digest[1];
    uint32 c = digest[2];
    uint32 d = digest[3];
    uint32 e = digest[4];
    SHA1_R0(a,b,c,d,e, 0);
    SHA1_R0(e,a,b,c,d, 1);
    SHA1_R0(d,e,a,b,c, 2);
    SHA1_R0(c,d,e,a,b, 3);
    SHA1_R0(b,c,d,e,a, 4);
    SHA1_R0(a,b,c,d,e, 5);
    SHA1_R0(e,a,b,c,d, 6);
    SHA1_R0(d,e,a,b,c, 7);
    SHA1_R0(c,d,e,a,b, 8);
    SHA1_R0(b,c,d,e,a, 9);
    SHA1_R0(a,b,c,d,e,10);
    SHA1_R0(e,a,b,c,d,11);
    SHA1_R0(d,e,a,b,c,12);
    SHA1_R0(c,d,e,a,b,13);
    SHA1_R0(b,c,d,e,a,14);
    SHA1_R0(a,b,c,d,e,15);
```

SHA1_R1(e,a,b,c,d,16);
SHA1_R1(d,e,a,b,c,17);
SHA1_R1(c,d,e,a,b,18);
SHA1_R1(b,c,d,e,a,19);
SHA1_R2(a,b,c,d,e,20);
SHA1_R2(e,a,b,c,d,21);
SHA1_R2(d,e,a,b,c,22);
SHA1_R2(c,d,e,a,b,23);
SHA1_R2(b,c,d,e,a,24);
SHA1_R2(a,b,c,d,e,25);
SHA1_R2(e,a,b,c,d,26);
SHA1_R2(d,e,a,b,c,27);
SHA1_R2(c,d,e,a,b,28);
SHA1_R2(b,c,d,e,a,29);
SHA1_R2(a,b,c,d,e,30);
SHA1_R2(e,a,b,c,d,31);
SHA1_R2(d,e,a,b,c,32);
SHA1_R2(c,d,e,a,b,33);
SHA1_R2(b,c,d,e,a,34);
SHA1_R2(a,b,c,d,e,35);
SHA1_R2(e,a,b,c,d,36);
SHA1_R2(d,e,a,b,c,37);
SHA1_R2(c,d,e,a,b,38);
SHA1_R2(b,c,d,e,a,39);
SHA1_R3(a,b,c,d,e,40);
SHA1_R3(e,a,b,c,d,41);
SHA1_R3(d,e,a,b,c,42);
SHA1_R3(c,d,e,a,b,43);
SHA1_R3(b,c,d,e,a,44);
SHA1_R3(a,b,c,d,e,45);
SHA1_R3(e,a,b,c,d,46);
SHA1_R3(d,e,a,b,c,47);

SHA1_R3(c,d,e,a,b,48);
SHA1_R3(b,c,d,e,a,49);
SHA1_R3(a,b,c,d,e,50);
SHA1_R3(e,a,b,c,d,51);
SHA1_R3(d,e,a,b,c,52);
SHA1_R3(c,d,e,a,b,53);
SHA1_R3(b,c,d,e,a,54);
SHA1_R3(a,b,c,d,e,55);
SHA1_R3(e,a,b,c,d,56);
SHA1_R3(d,e,a,b,c,57);
SHA1_R3(c,d,e,a,b,58);
SHA1_R3(b,c,d,e,a,59);
SHA1_R4(a,b,c,d,e,60);
SHA1_R4(e,a,b,c,d,61);
SHA1_R4(d,e,a,b,c,62);
SHA1_R4(c,d,e,a,b,63);
SHA1_R4(b,c,d,e,a,64);
SHA1_R4(a,b,c,d,e,65);
SHA1_R4(e,a,b,c,d,66);
SHA1_R4(d,e,a,b,c,67);
SHA1_R4(c,d,e,a,b,68);
SHA1_R4(b,c,d,e,a,69);
SHA1_R4(a,b,c,d,e,70);
SHA1_R4(e,a,b,c,d,71);
SHA1_R4(d,e,a,b,c,72);
SHA1_R4(c,d,e,a,b,73);
SHA1_R4(b,c,d,e,a,74);
SHA1_R4(a,b,c,d,e,75);
SHA1_R4(e,a,b,c,d,76);
SHA1_R4(d,e,a,b,c,77);
SHA1_R4(c,d,e,a,b,78);
SHA1_R4(b,c,d,e,a,79);

```
        digest[0] += a;
        digest[1] += b;
        digest[2] += c;
        digest[3] += d;
        digest[4] += e;
        transforms++;
    }
void SHA1::buffer_to_block(const std::string &buffer, uint32 block[BLOCK_BYTES])
{
    for (unsigned int i = 0; i < BLOCK_INTS; i++)
    {
        block[i] = (buffer[4*i+3] & 0xff)
            | (buffer[4*i+2] & 0xff)<<8
            | (buffer[4*i+1] & 0xff)<<16
            | (buffer[4*i+0] & 0xff)<<24;
    }
}
void SHA1::read(std::istream &is, std::string &s, int max)
{
    char sbuf[max];
    is.read(sbuf, max);
    s.assign(sbuf, is.gcount());
}
std::string sha1(const std::string &string)
{
    SHA1 checksum;
    checksum.update(string);
    return checksum.final();
}
```

Output:

```
C:\Users\bhumit\Desktop\sha.exe
Enter word: hello world
aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d
-----
Process exited after 11.73 seconds with return value 0
Press any key to continue . . .
```


PRACTICAL-10

Aim : Write a program to Implement a Digital Signature Algorithm.

Introduction:

- Digital Signatures are an asymmetrically encrypted hash of a digital message (data). It is a value that can provide a guarantee of authenticity, non-repudiation, and integrity.
- In other terms, it means you can verify the sender, date & time and message content have not been revealed or compromised.

Digital Signature Flow:

- Let “A” and “B” be the fictional actors in the cryptography system for better understanding.
- “A” is the sender and calculates the hash of the message and attaches signature which he wants to send using his private key.
- The other side “B” hashes the message and then decrypts the signature with A’s public key and compares the two hashes
- If “B” finds the hashes matching then the message has not been altered or compromised.

Code:

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.SecureRandom;
import java.security.Signature;
import java.util.Scanner;
import javax.xml.bind.DatatypeConverter;

public class Digital_Signature
{

    private static final String SIGNING_ALGORITHM = "SHA256withRSA";
    private static final String RSA = "RSA";
    private static Scanner sc;

    public static byte[] Create_Digital_Signature( byte[] input, PrivateKey Key) throws Exception {
        Signature signature = Signature.getInstance(SIGNING_ALGORITHM);
        signature.initSign(Key);
        signature.update(input);
        return signature.sign();
    }

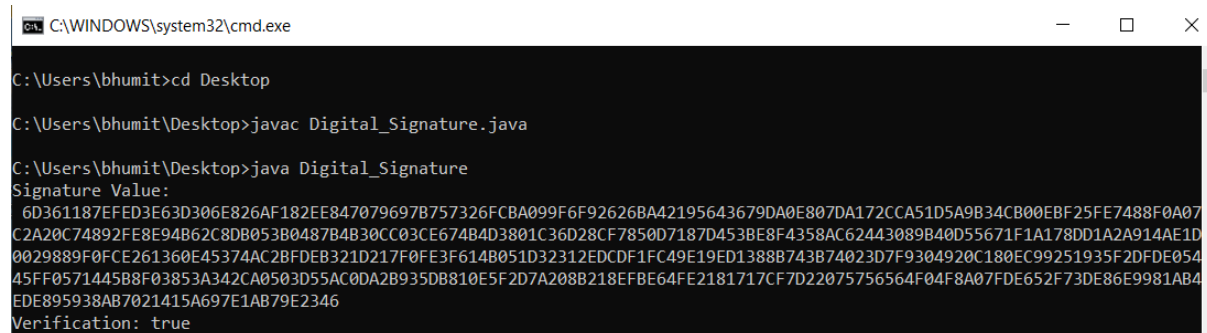
    public static KeyPair Generate_RSA_KeyPair() throws Exception
    {
        SecureRandom secureRandom = new SecureRandom();
        KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance(RSA);
```

```

        keyPairGenerator.initialize(2048, secureRandom);
        return keyPairGenerator.generateKeyPair();
    }
    public static boolean Verify_Digital_Signature(byte[] input,byte[]
signatureToVerify,PublicKey key)throws Exception
    {
        Signature signature = Signature.getInstance(SIGNING_ALGORITHM);
        signature.initVerify(key);
        signature.update(input);
        return signature.verify(signatureToVerify);
    }
    public static void main(String args[])throws Exception
    {
        String input = "bhumit";
        KeyPair keyPair = Generate_RSA_KeyPair();
        byte[] signature = Create_Digital_Signature(input.getBytes(),keyPair.getPrivate());
        System.out.println("Signature Value:\n "+
DatatypeConverter.printHexBinary(signature));
        System.out.println( "Verification:
"+Verify_Digital_Signature(input.getBytes(),signature,
        keyPair.getPublic()));
    }
}

```

Output:



```

C:\WINDOWS\system32\cmd.exe
C:\Users\bhumit>cd Desktop
C:\Users\bhumit\Desktop>javac Digital_Signature.java
C:\Users\bhumit\Desktop>java Digital_Signature
Signature Value:
 6D361187EFED3E63D306E826AF182EE847079697B757326FCBA099F6F92626BA42195643679DA0E807DA172CCA51D5A9B34CB00EBF25FE7488F0A07
C2A20C74892FE8E94B62C8DB053B0487B4B30CC03CE674B4D3801C36D28CF7850D7187D453BE8F4358AC62443089B40D55671F1A178DD1A2A914AE1D
0029889F0FE261360E45374AC2BFDEB321D217F0FE3F614B051D32312EDCDF1FC49E19ED1388B743B74023D7F9304920C180EC99251935F2DFDE054
45FF0571445B8F03853A342CA0503D55AC0DA2B935DB810E5F2D7A208B218EFBE64FE2181717CF7D22075756564F04F8A07FDE652F73DE86E9981AB4
EDE895938AB7021415A697E1AB79E2346
Verification: true

```

PRACTICAL: 11

Aim: Perform various Encryption-Decryption techniques with the Cryptool.

Introduction:

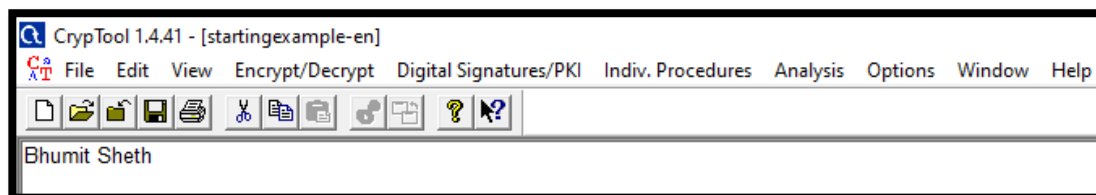
- Cryptool is an open source e-learning tool illustrating cryptographic and cryptanalytic concepts.
- Cryptool implements more than 300 algorithms. Users can adjust these with own parameters.
- The graphical interface, online documentation, analytic tools and algorithms of Cryptool introduce users to the field of cryptography.
- Classical ciphers are available alongside asymmetric cryptography including RSA, elliptic curve cryptography, digital signatures, homomorphic encryption, and Diffie–Hellman key exchange, many of which are visualized by animations.
- Cryptool also contains some didactical games, and an animated tutorial about primes and elementary number theory.

Encryption – decryption :

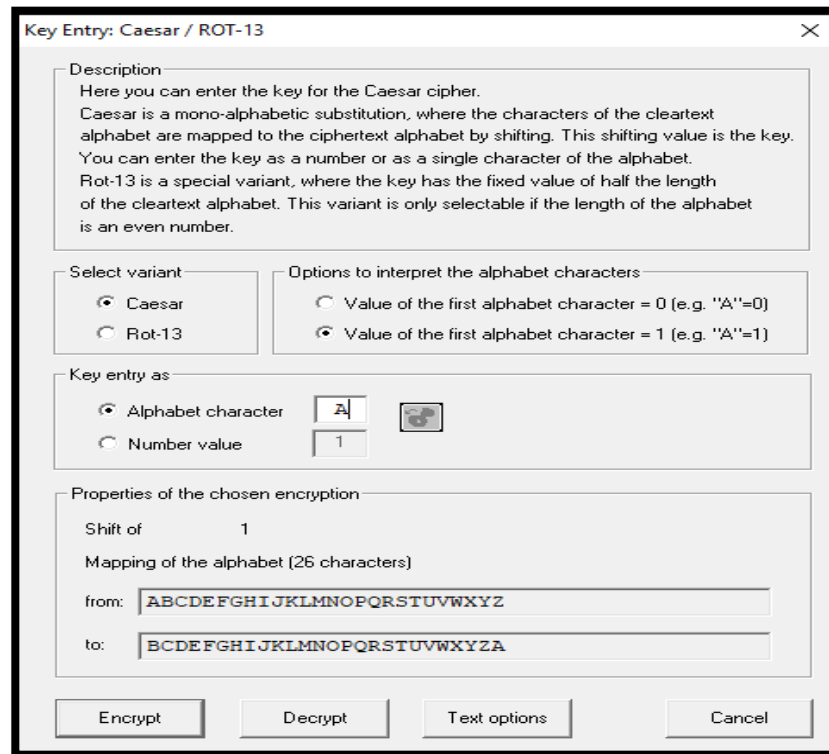
1. Caesar Cipher

- The Caesar cipher is named so, as it was first implemented by Julius Caesar. It also goes by the name of shift cipher and belongs to the family of substitution ciphers. With this cipher each character is represented by another character in the alphabet, more specifically the whole alphabet is shifted by a set amount to encrypt the data.
- The set amount that the alphabet is shifted by becomes the key, anyone with knowledge of this key should be able to reveal the true contents of the ciphertext and Cryptographic Techniques for Network Security -25- conversely anyone without knowledge of the key should not be able to interpret the ciphertext as the original plaintext.

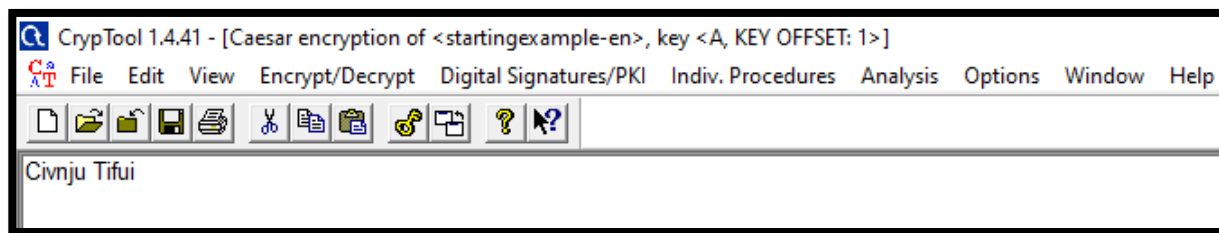
Encryption:



- Open a new cryptool file for the plaintext, Choose Encrypt/Decrypt/Symmetric (classic)/Caesar/Rot-13/



- Then apply the type of encryption to perform on plaintext. And click the Encrypt button.



Decryption:


- Open a new cryptool file for the ciphertext, Choose Encrypt/Decrypt/Symmetric (classic)/Caesar/Rot-13/

Key Entry: Caesar / ROT-13

Description
 Here you can enter the key for the Caesar cipher.
 Caesar is a mono-alphabetic substitution, where the characters of the cleartext alphabet are mapped to the ciphertext alphabet by shifting. This shifting value is the key. You can enter the key as a number or as a single character of the alphabet.
 Rot-13 is a special variant, where the key has the fixed value of half the length of the cleartext alphabet. This variant is only selectable if the length of the alphabet is an even number.

Select variant
☒ Caesar
☐ Rot-13

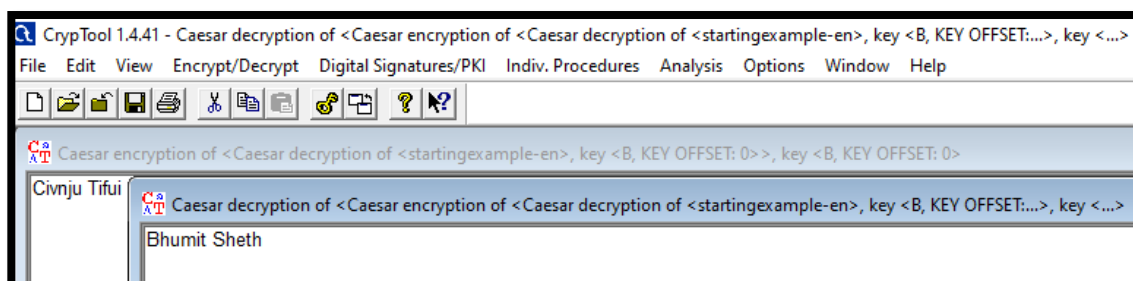
Options to interpret the alphabet characters
☐ Value of the first alphabet character = 0 (e.g. "A"=0)
☒ Value of the first alphabet character = 1 (e.g. "A"=1)

Key entry as
☒ Alphabet character 
☐ Number value

Properties of the chosen encryption
 Shift of 1
 Mapping of the alphabet (26 characters)
 from: ABCDEFGHIJKLMNOPQRSTUVWXYZ
 to: BCDEFGHIJKLMNOPQRSTUVWXYZA

Encrypt Decrypt Text options Cancel

- After applying the type of decryption, click the Decrypt button.



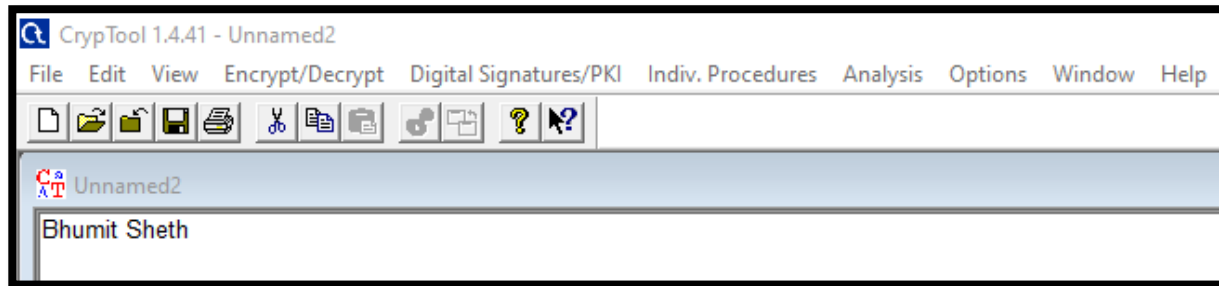
2. Vigenere Cipher

- The Vigenère cipher bears much resemblance to the Modified Caesar cipher, discussed above, but using a key phrase instead of the numbers in the shift vector. The encryption process for this cipher takes each individual letter of a plaintext message and combines it with a letter from the key, together these two letters act as coordinates for a position on the Vigenère square. To decipher a message, reverse the process, starting by finding the character

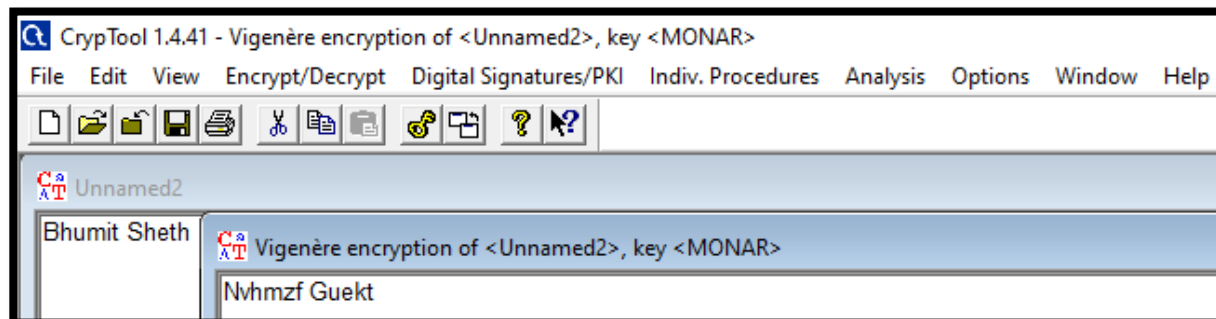
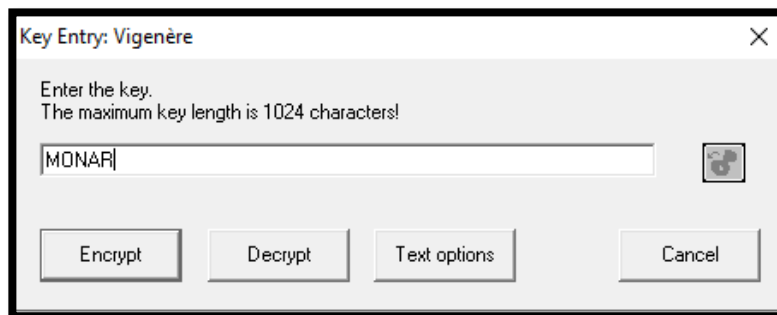
of the key, and then find the ciphertext letter along the same row and then find what column it is on to reveal the plaintext.

Encryption:

- Open a new cryptool file for the plaintext, Choose Encrypt/Decrypt/Symmetric (classic)/Vignere/

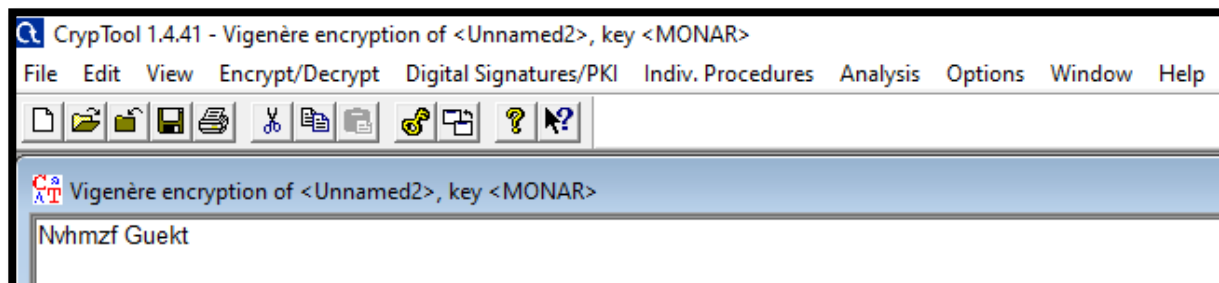


- Enter the key for the encryption, Then click the Encrypt button.

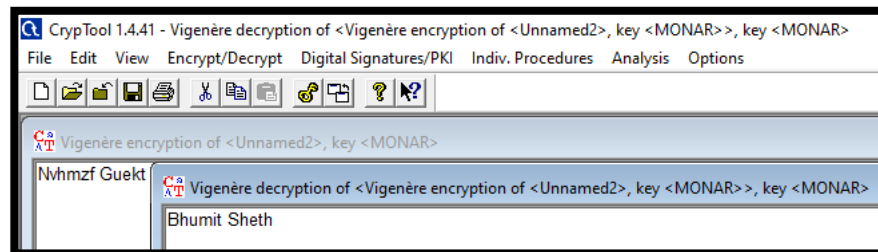
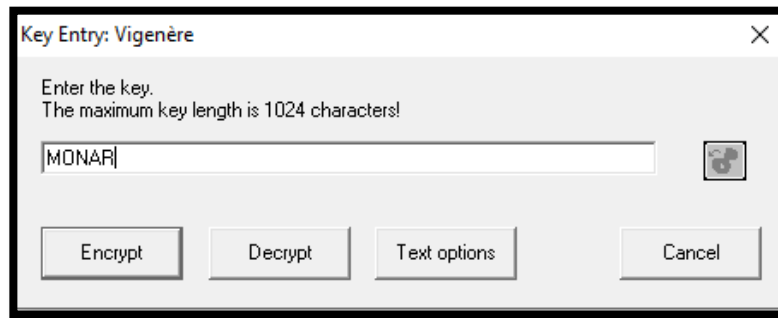


Decryption:

- Open a new cryptool file for the cipherntext, Choose Encrypt/Decrypt/Symmetric (classic)/Vignere/



- Enter the key for the decryption, Then click the Decrypt button.

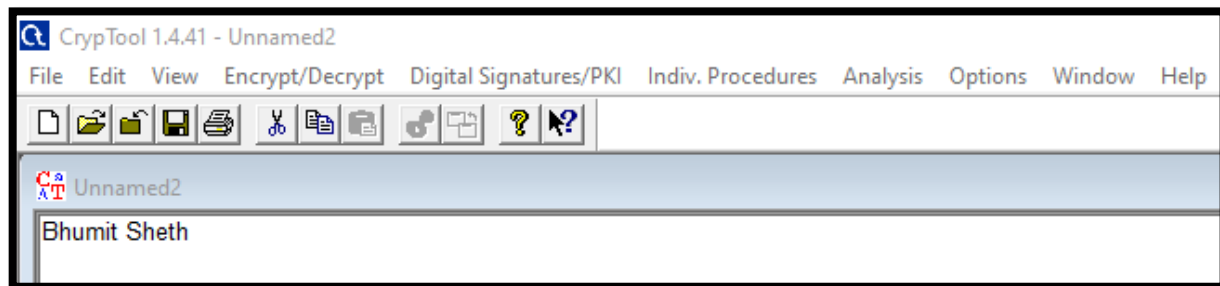


3. Playfair Cipher

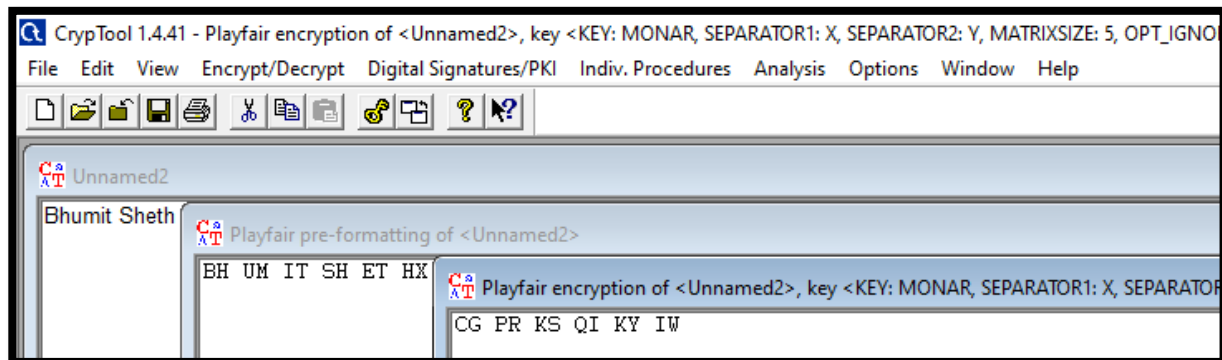
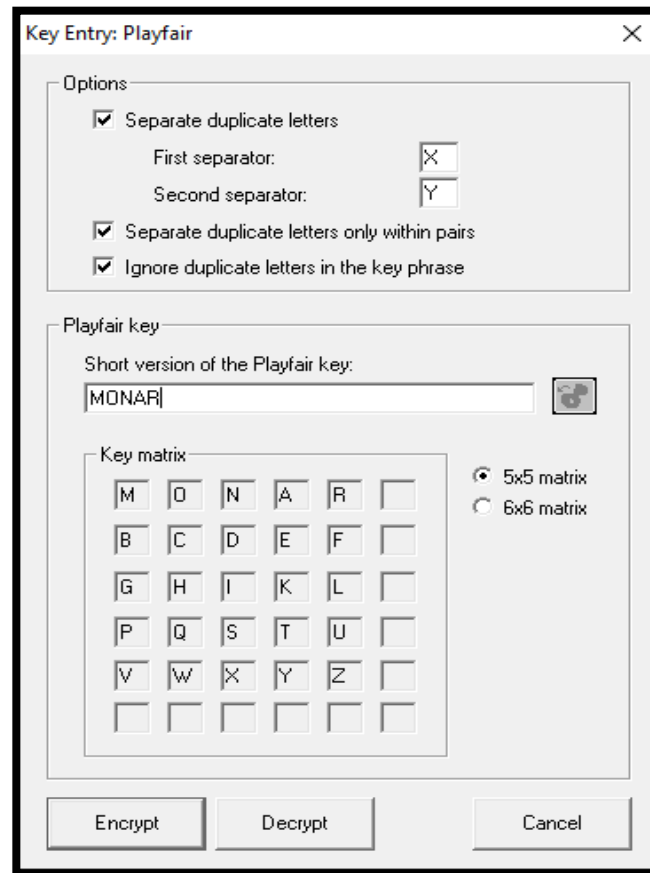
- The Playfair cipher was the first practical digraph substitution cipher. The scheme was invented in 1854 by Charles Wheatstone but was named after Lord Playfair who promoted the use of the cipher. In playfair cipher unlike traditional cipher we encrypt a pair of alphabets(digraphs) instead of a single alphabet.
- It was used for tactical purposes by British forces in the Second Boer War and in World War I and for the same purpose by the Australians during World War II. This was because Playfair is reasonably fast to use and requires no special equipment.

Encryption:

- Open a new cryptool file for the plaintext, Choose Encrypt/Decrypt/Symmetric (classic)/Playfair

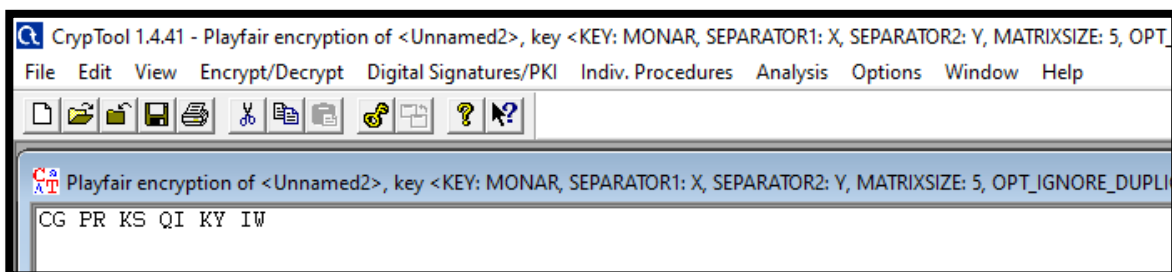


- Enter the key for the encryption, Then click the Encrypt button.



Decryption:

- Open a new cryptool file for the cipherntext, Choose Encrypt/Decrypt/Symmetric (classic)/Playfair




- Enter the key for the decryption, Then click the Decrypt button.

Key Entry: Playfair

Options

- ☒ Separate duplicate letters
 - First separator:
 - Second separator:
- ☒ Separate duplicate letters only within pairs
- ☒ Ignore duplicate letters in the key phrase

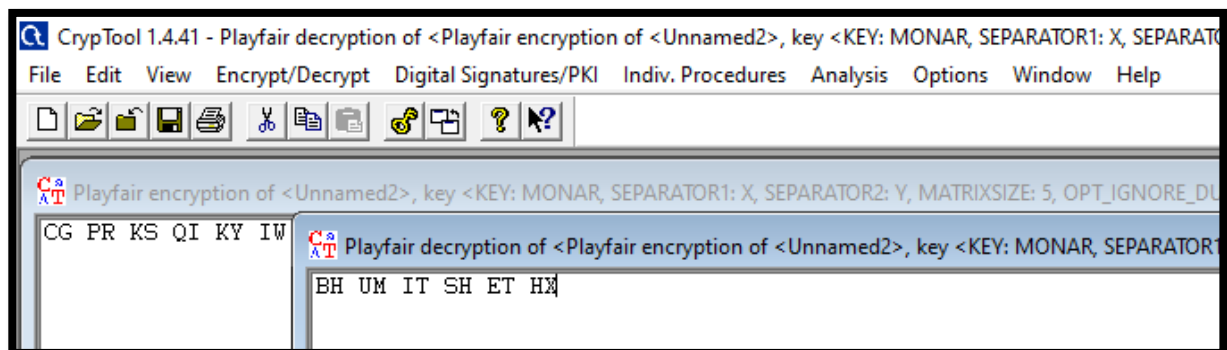
Playfair key

Short version of the Playfair key: 

Key matrix

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

☒ 5x5 matrix
☐ 6x6 matrix



PRACTICAL-12

Aim: Study use the Wireshark for the various network protocols.

Introduction:

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. One could think of a network packet analyzer as a measuring device for examining what's happening inside a network cable, just like an electrician uses a voltmeter for examining what's happening inside an electric cable. Wireshark is available for free, is open source and is one of the best packet analyzer available today. Wireshark is available for windows, linux and mac operating system.

Packet Sniffer:

The basic tool for observing the messages exchanged between executing protocol entities is called a packet sniffer. As the name suggests, a packet sniffer captures ("sniffs") messages being sent/received from/by your computer; it will also typically store and/or display the contents of the various protocol fields in these captured messages. A packet sniffer itself is passive. It observes messages being sent and received by applications and protocols running on your computer, but never sends packets itself. Similarly, received packets are never explicitly addressed to the packet sniffer. Instead, a packet sniffer receives a copy of packets that are sent / received from/by application and protocols executing on your machine.

Purposes:

Here are some reasons people use Wireshark:

- Network administrators use it to troubleshoot network problems
- Network security engineers use it to examine security problems
- QA engineers use it to verify network applications
- Developers use it to debug protocol implementations
- People use it to learn network protocols.

Features:

The following are some of the many features Wireshark provides:

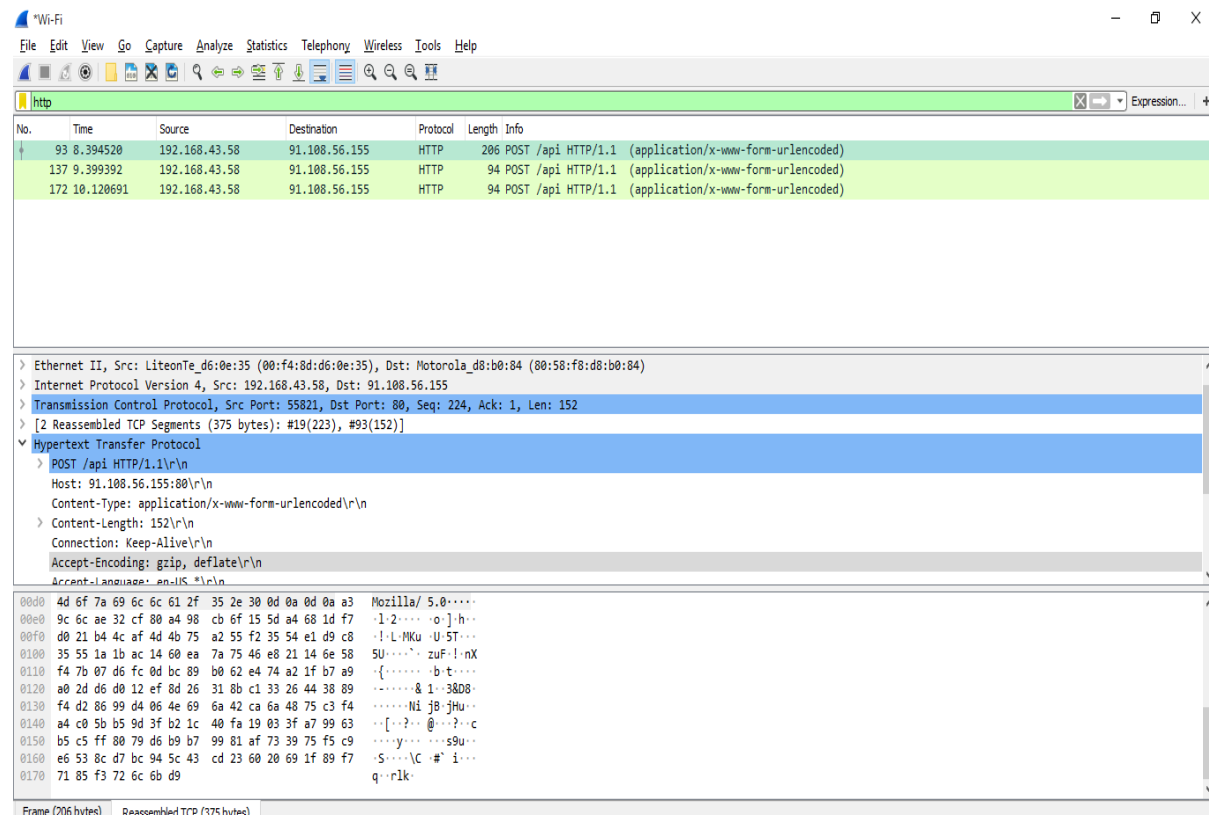
- Available for UNIX and Windows.
- Capture live packet data from a network interface.
- Open files containing packet data captured with tcpdump/WinDump, Wireshark, and many other packet capture programs.
- Import packets from text files containing hex dumps of packet data.
- Display packets with very detailed protocol information.
- Save packet data captured.
- Export some or all packets in a number of capture file formats.
- Filter packets on many criteria.
- Search for packets on many criteria.
- Colorize packet display based on filters.

- Create various statistics.
- ...and a lot more!

Wireshark can be downloaded from following link: <https://wireshark.org>

HTTP Protocol :

The **Hypertext Transfer Protocol (HTTP)** is an application layer protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web, where hypertext documents include hyperlink to other resources that the user can easily access.



TCP Protocol :

TCP (Transmission Control Protocol) is connection-oriented, and a connection between client and server is established before data can be sent. The server must be listening (passive open) for connection requests from clients before a connection is established. Three-way handshake (active open), retransmission, and error-detection adds to reliability but lengthens latency. Applications that do not require reliable data stream service may use the User Datagram Protocol (UDP), which provides a connectionless datagram service that prioritizes time over reliability. TCP employs network congestion avoidance. However, there are vulnerabilities to TCP including denial of service, connection hijacking, TCP veto, and reset attack. For network security, monitoring, and debugging, TCP traffic can be intercepted and logged with a packet sniffer.

The screenshot shows a Wireshark capture of a TCP session. The packet list pane displays several packets, with packet 133 highlighted. The packet details pane shows the structure of the selected packet, including Ethernet II, Internet Protocol Version 4, and Transmission Control Protocol. The packet bytes pane shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
127	9.210750	192.168.43.58	91.108.56.155	TCP	54	56644 → 443 [ACK] Seq=1 Ack=1 Win=65536 Len=0
128	9.210818	192.168.43.58	91.108.56.155	TCP	54	56645 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
129	9.211034	91.108.56.155	192.168.43.58	TCP	54	[TCP Out-Of-Order] 80 → 55797 [FIN, ACK] Seq=1 Ack=2 Win=11052 Len=0
130	9.211086	192.168.43.58	91.108.56.155	TCP	54	[TCP ZeroWindow] 55797 → 80 [ACK] Seq=2 Ack=2 Win=0 Len=0
131	9.211377	192.168.43.58	91.108.56.155	TCP	276	56645 → 80 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=222 [TCP segment of a reassembled PDU]
132	9.214456	192.168.43.58	91.108.56.155	SSL	159	Continuation Data
133	9.270866	91.108.56.155	192.168.43.58	TCP	54	80 → 55797 [ACK] Seq=2 Ack=2 Win=0 Len=0
136	9.397630	192.168.43.58	91.108.56.155	TCP	54	56644 → 443 [FIN, ACK] Seq=106 Ack=1 Win=65536 Len=0
137	9.399392	192.168.43.58	91.108.56.155	HTTP	94	POST /api HTTP/1.1 (application/x-www-form-urlencoded)
138	9.402995	192.168.43.58	91.108.56.155	TCP	66	56646 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1

Frame 75: 55 bytes on wire (440 bits), 55 bytes captured (440 bits) on interface 0
 Ethernet II, Src: LiteonTe_d6:0e:35 (00:f4:8d:d6:0e:35), Dst: Motorola_d8:b0:84 (80:58:f8:d8:b0:84)
 Internet Protocol Version 4, Src: 192.168.43.58, Dst: 151.101.154.114
 Transmission Control Protocol, Src Port: 55938, Dst Port: 443, Seq: 1, Ack: 1, Len: 1

0000 80 58 f8 d8 b0 84 00 f4 8d d6 0e 35 08 00 45 00 X.....5..E.
 0010 00 29 57 da 40 00 06 85 3a c0 a8 2b 3a 97 65 W@...:..e
 0020 9a 72 da 82 01 bb 91 2c 23 dd 80 1d 68 50 10 P.....,hP
 0030 00 fe d4 3f 00 00 00?..

UDP Protocol :

UDP(User Datagram Protocol) uses a simple connectionless communication model with a minimum of protocol mechanisms. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram. It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network; there is no guarantee of delivery, ordering, or duplicate protection.

The screenshot shows a Wireshark capture of a UDP session. The packet list pane displays several packets, with packet 87 highlighted. The packet details pane shows the structure of the selected packet, including Ethernet II, Internet Protocol Version 6, and User Datagram Protocol. The packet bytes pane shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
82	8.036023	2404:6800:4007:807::	2405:205:c963:26b6::	UDP	1392	443 → 51769 Len=1330
83	8.062065	2405:205:c963:26b6::	2404:6800:4007:807::	UDP	99	51769 → 443 Len=37
84	8.064924	2404:6800:4007:807::	2405:205:c963:26b6::	UDP	1392	443 → 51769 Len=1330
85	8.090631	2405:205:c963:26b6::	2404:6800:4007:807::	UDP	99	51769 → 443 Len=37
86	8.171025	2404:6800:4007:807::	2405:205:c963:26b6::	UDP	1392	443 → 51769 Len=1330
87	8.196729	2405:205:c963:26b6::	2404:6800:4007:807::	UDP	99	51769 → 443 Len=37
88	8.257800	2404:6800:4007:807::	2405:205:c963:26b6::	UDP	1392	443 → 51769 Len=1330
89	8.283532	2405:205:c963:26b6::	2404:6800:4007:807::	UDP	99	51769 → 443 Len=37
90	8.360897	2404:6800:4007:807::	2405:205:c963:26b6::	UDP	1392	443 → 51769 Len=1330
91	8.386343	2405:205:c963:26b6::	2404:6800:4007:807::	UDP	99	51769 → 443 Len=37

Frame 87: 99 bytes on wire (792 bits), 99 bytes captured (792 bits) on interface 0
 Ethernet II, Src: LiteonTe_d6:0e:35 (00:f4:8d:d6:0e:35), Dst: Motorola_d8:b0:84 (80:58:f8:d8:b0:84)
 Internet Protocol Version 6, Src: 2405:205:c963:26b6::c1be, Dst: 2404:6800:4007:807::2004
 User Datagram Protocol, Src Port: 51769, Dst Port: 443
 Data (37 bytes)

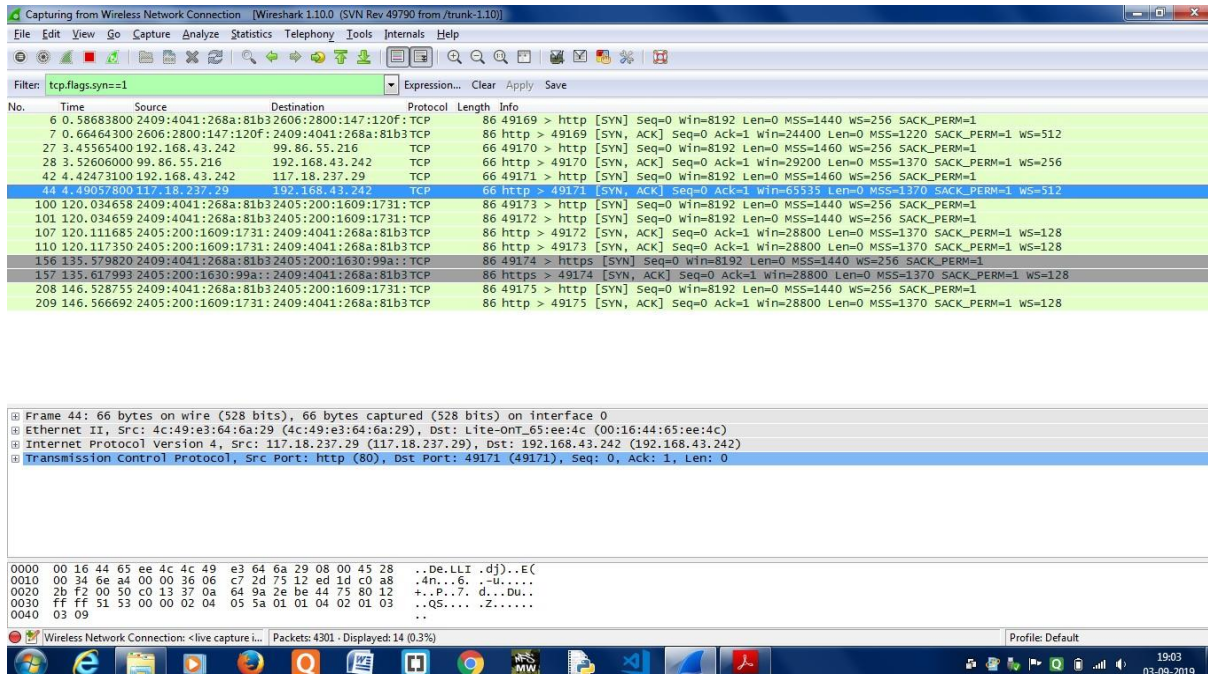
0000 80 58 f8 d8 b0 84 00 f4 8d d6 0e 35 86 dd 60 00 X.....5..
 0010 00 00 00 2d 11 40 24 05 02 05 c9 63 26 b6 c1 be @S...c&..
 0020 cc 93 40 3c a3 db 24 04 68 00 40 07 08 07 00 00 @c..\$.h@..
 0030 00 00 00 20 04 ca 39 01 bb 00 2d 05 6f 57 78 :..9...eo..
 0040 f5 04 f43 c3 c2 18 75 0a 64 03 93 20 71 6a e4 :..:..b...0..
 0050 3d 7c 9d 8f 12 09 ae 2b ae d4 22 c5 ae a7 33 2d :.....:..
 0060 c9 ca 52 ..R

Commands for Wireshark

1) Analyzing TCP session using wireshark:

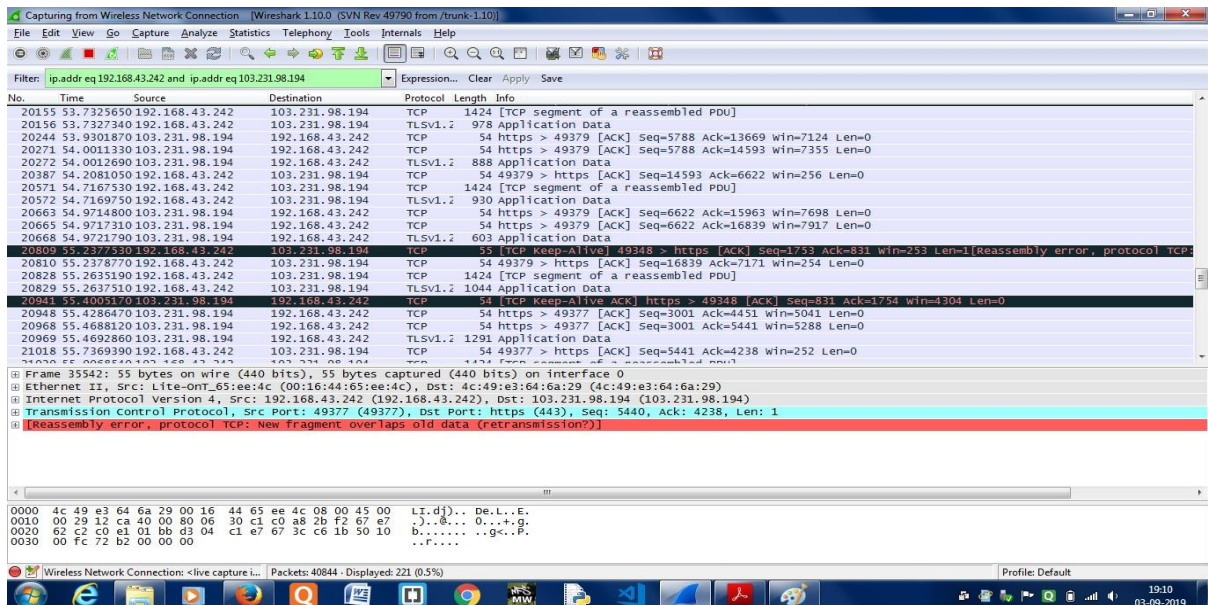
Find first SYN packet, sent from your PC to the web server. This signifies the start of a TCP 3-way

handshake. Choose the correct flag and add==1.Hit the find button and first SYN packet in the trace should be highlighted.



2) Filtering particular IP address from given IP addresses:

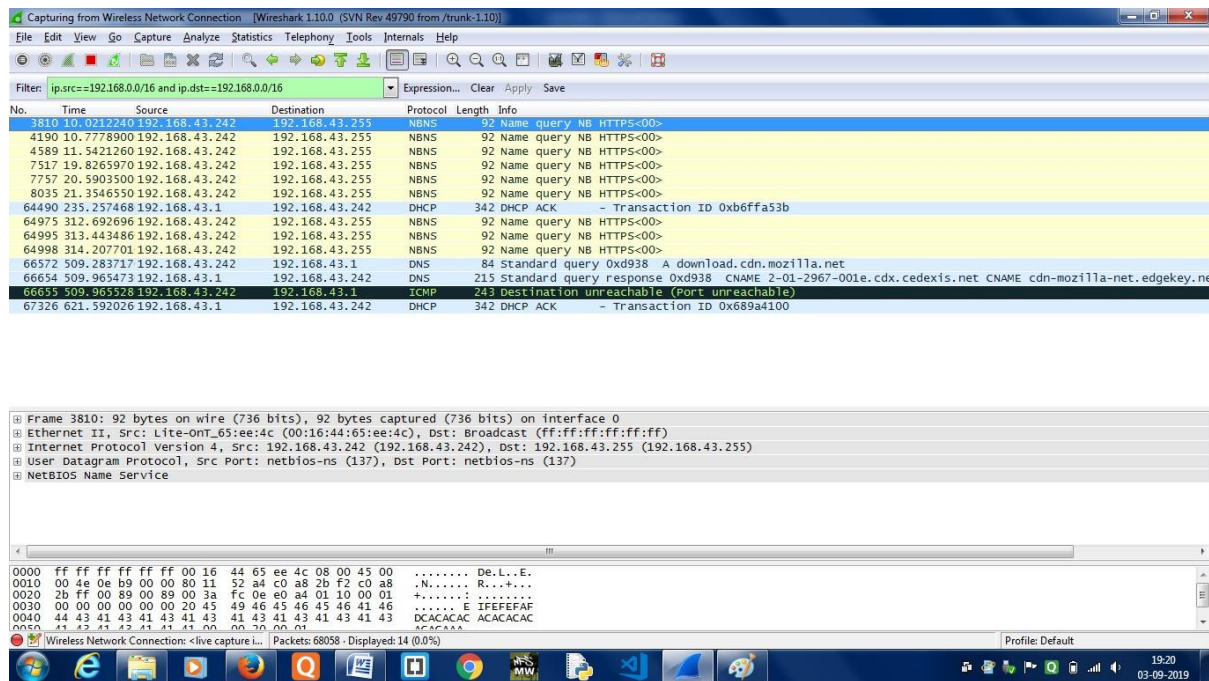
This command helps a programmer to filter packets according to its requirement from available packets which are being travelling from source port to destination port. Thus, by using Wireshark we can analyse packets travelling from source to destination.
ip.addr eq 192.168.43.242 and ip.addr eq 103.231.98.194



3) Analysing packets travelling from a particular source to particular destination:

The given command helps a programmer to trace those packets which are travelling from given source IP address to given destination IP address.

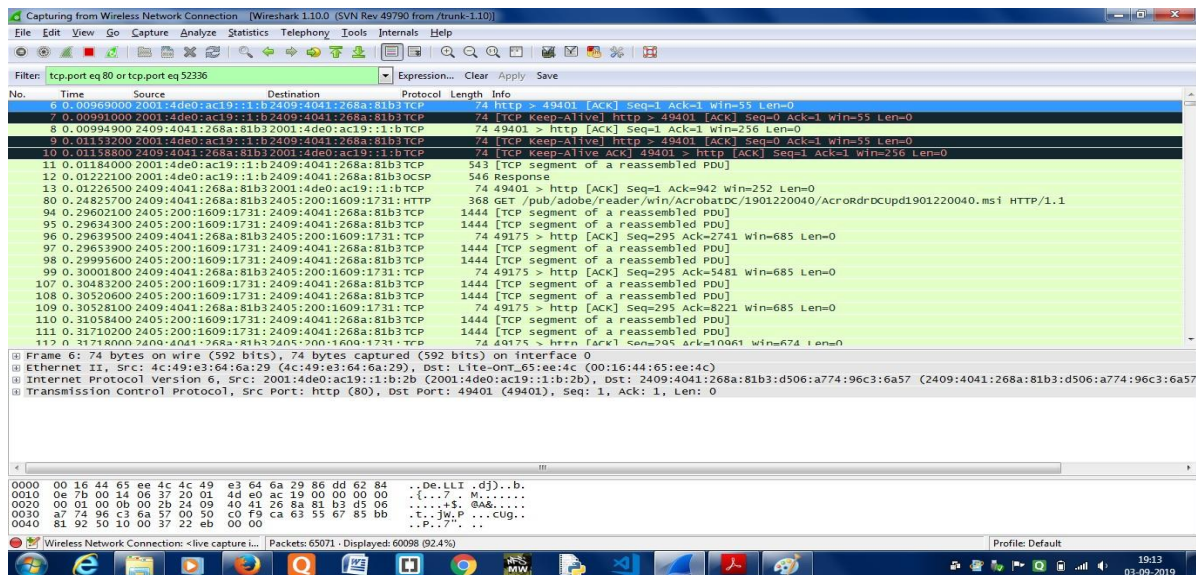
ip.src==192.168.0.0/16 and ip.dst==192.168.0.0/16



4)Monitoring traffic on more than two ports:

Sometimes you will be interested in inspecting traffic that matches either (or both) conditions whatsoever. For example if user wants to monitor tcp ports and other ports simultaneously then or relation can be used between them.

Tcp.port eq 80 or tcp.port eq 52336



Conclusion:

We have concluded that wireshark is a program that is used to capture data packets to allow a more precise analysis. Such a tool allows the user to examine his/her own computer for protocol errors and problems within the network architecture.

Virtual Lab Practical

DES to 3-DES

Aim: In this experiment, you are asked to design the triple DES cryptosystem provided that you are given an implementation of DES

Manual:

Step 1 : Generate Plaintext m, keyA and keyB by clicking on respective buttons PART I of the simulation page.

PART I

Message:

Key Part A:

Key Part B:

Step 2 : Enter generated Plaintext m from PART I to PART II in "Your text to be encrypted/decrypted:" block.

PART II

Your text to be encrypted/decrypted:

Key to be used:

Output:

Step 3 : Enter generated keyA from PART I to PART II "Key to be used:" block and click on DES encrypt button to output ciphertext c1. This is First Encryption.

PART II

Your text to be encrypted/decrypted:

Key to be used:

Output:

Step 4 : Enter generated ciphertext c1 from PART II "Output:" Block to PART II in "Your text to be encrypted/decrypted:" block.

PART II

Your text to be encrypted/decrypted:

Key to be used:

Output:

Step 5 : Enter generated keyB from PART I to PART II in "Key to be used:" block and click on

DES decrypt button to output ciphertext c2. This is Second Encryption.

PART II

Your text to be encrypted/decrypted: 00111110 11010100 11010111 01101101 10000110 11100111 00010001 01111110

Key to be used: 922fb510c71f436e

DES Encrypt DES Decrypt

Output: 10101011 10101110 01111110 01111111 01111000 10000100 10011100 10010101

Step 6 : Enter generated ciphertext c2 from PART II "Output:" block to PART II in "Your text to be encrypted/decrypted:" block.

PART II

Your text to be encrypted/decrypted: 10101011 10101110 01111110 01111111 01111000 10000100 10011100 10010101

Key to be used: 922fb510c71f436e

DES Encrypt DES Decrypt

Output: 10101011 10101110 01111110 01111111 01111000 10000100 10011100 10010101

Step 7 : Enter generated keyA from PART I to PART II "Key to be used:" block and click on DES encrypt button to output ciphertext c3. This is Third Encryption. As Encryption is done thrice. This Scheme is called triple DES.

PART II

Your text to be encrypted/decrypted: 10101011 10101110 01111110 01111111 01111000 10000100 10011100 10010101

Key to be used: 3b3898371520f75e

DES Encrypt DES Decrypt

Output: 00011101 11100100 10001000 01101111 11010001 00011011 00110000 110000

Step 8 : Enter generated ciphertext c3 from PART II "Output:" Block to PART III "Enter your answer here:" block in order to verify your Triple DES.

PART III

Enter your answer here:

00011101 11100100 10001000 01101111 11010001 00011011 00110000 110000

Check Answer!

CORRECT!

Reference-

<http://cse29-iiith.vlabs.ac.in/exp6/Introduction.html?domain=Computer%20Science&lab=Cryptography%20Lab>

Breaking the shift cipher

Aim: In this experiment, we work with a well-known historical encryption scheme, namely the shift cipher, that has a very small key space.

Your task is to break the shift cipher. Specifically, given (only) the ciphertext in some instance of a shift cipher, you need to find the plaintext and the secret key.

Manual:

STEP 1 : For the given ciphertext in the PART I of the simulation page, the first step is to decrypt it using each of the twenty-six different keys, $k=0,1,\dots,25$ and obtain the corresponding plaintexts. For decryption, you may use the tool given in the PART III of the simulation page.

PART I

Ciphertext to be decrypted:

ymnx nx ymj ktwjxy uwnrjafq

Next Ciphertext

PART III

Plaintext:

the quick brown fox

shift: 5 ▾

v Encrypt v ^ Decrypt ^

Ciphertext

ymj vznhp gwtbs ktc

STEP 2 : After each decryption, you may cut-and-paste the resultant plaintext in the scratch-pad in the (PART II) of the simulation page, if you need to remember it.

STEP 3 : Finally, observe the plaintexts and choose the most appropriate one (the one that is a meaningful English text) as the recovered plaintext and cut-and-paste it in the text-field named PART IV "Solution Plaintext". Also select the corresponding key in the text-field named "Key" and click on "Check My answer" Button.

PART III

Plaintext:

this is the forest primeval

shift: 5 ▾

v Encrypt v ^ Decrypt ^

Ciphertext

ymnx nx ymj ktwjxy uwnrjafq

STEP 4 [OPTIONAL] : Verify that your answer is correct, by encrypting the solution plaintext with your key.

PART IV

Enter your solution Plaintext and shift key here:

this is the forest primeval

Key 5 ▾

Check my answer!

CORRECT!!

Reference-

<http://cse29-iiith.vlabs.ac.in/exp1/Introduction.html?domain=Computer%20Science&lab=Cryptography%20Lab>

Digital signature scheme

Aim: In Public key setting, it becomes difficult to verify for a receiver whether message is originated from claimed source.

In this experiment, we show how a receiver can verify integrity of the message in public key setting.

Your task is to verify, whether digital signature scheme really works and why it works?

About:

A Digital Signature is an authentication mechanism that enables the creator of the message to attach a code that acts as a signature. The signature is formed by taking the hash of the message and encrypting the message with the creator's private key. The signature guarantees the source and integrity of the message.

Manual:

Step 1 : Enter the input text to be encrypted in the 'Plaintext' area and generate hash value for message by clicking on the SHA-1 button

Digitally sign the plaintext with Hashed RSA.

Plaintext (string):

how are you

SHA-1

Hash output(hex):

4bfe46a79d38aba1480cd5d9466bebg911895bf

Step 2 : Copy content of Hash Output(hex) field and paste it in Input to RSA(hex) field.

Input to RSA(hex):

4bfe46a79d38aba1480cd5d9466bebg111895bf

Step 3 : Select keysize of public key from RSA Public key section by clicking on any key button

RSA public key

Public exponent (hex, F4=0x10001):

3

Modulus (hex):

BC86E3DC782C446EE756B874ACECF2A115E613021EAF1ED5EF295BEC2BED89
9D
26FE2EC896BF9DE84FE381AF67A7B7CBB48D85235E72AB595ABF8FE840D5F8
DB

Here, 512 bit (e=3) is chosen

Step 4 : Click on Apply RSA button to generate a digital signature.

Input to RSA(hex):

4bfe46a79d38aba1480cd5d9466bebg111895bf

Digital Signature(hex):

428fco8828533c593e7d4eebo5ecfa954549713e715fb2b6005575e6bef8c5b4
d792892b21c0487629511c923438750ca38fd0oc2ad82b2ef833dda2c9381a51

Digital Signature(base64):

Qo/AiChTPFk+fU7rBez6lUVJcT5xX7K2AFV15r74xbTXkokrlcBdiLRHJIoOHUM
o4/QDCrYKy74M9ziyTgaUQ==

Status:

Time: 13ms

Reference:

<http://cse29-iiith.vlabs.ac.in/exp10/Introduction.html?domain=Computer%20Science&lab=Cryptography%20Lab>

OPEN ENDED PROBLEM

Problem statement: Implement two cylinder rotor machine in C/C++. First cylinder is faster than second one. Internal mapping of cylinder must be random. It encrypts as many as strings until user opt to exit. Also it prints states of cylinder before and after encrypting a string, and ciphertext in between this two states of cylinder.

Introduction:

Electric rotor machines were mechanical devices that allowed to use encryption algorithms that were much more complex than ciphers, which were used manually. They were developed in the middle of the second decade of the 20th century. They became one of the most important cryptographic solutions in the world for the next tens of years.

Usage

Electro-mechanical machines fitted with movable rotors are used to produce long random keystreams, thus allowing to encrypt messages by using complicated polyalphabetic substitution ciphers.

Description

The main idea that lies behind rotor machines is relatively simple. One can imagine a simple device, similar to a typewriter, with a number of keys used to input text. The number of keys may differ, however usually there are 26 to 32 characters.

Simple substitution cipher

Each keystroke produces an output character, depending of the internal construction of the machine. In the simplest case, if only wires are used (without any rotors), each input key will be mapped to one specific output character.

For example, if someone pressed K in the keyboard, the machine would always produce C. As a result, the machine would encrypt the messages by using a simple substitution cipher.

Adding a rotor

Having a simple substitution machine, one can imagine adding an additional internal rotor with an internal wiring. The rotor will rotate with a gear, each time after a keystroke. As a result, after pressing the same letter twice, it will be encoded differently due to different internal wiring.

For example, if someone pressed KK in the keyboard, the machine would produce CB (because the wiring changed after the first keystroke, due to the rotor movement).

The internal wiring of the rotor should be kept secret, however we may expect that over time the enemy will discover its design. It will make it easier for them to break the cipher but it won't compromise the security altogether.

To decode a ciphertext the receiver would need a machine with the same rotor. Adding the rotor caused the encryption to become a stronger polyalphabetic substitution cipher.

Make it difficult

To improve the security, one could add more rotors. The output of one rotor would be connected to the input of the second rotor. Similarly, the second rotor output would be connected to the third one, and so on. The strength of the encryption depends on several factors:

- the number of rotors inside the machine.
- the size of each rotor.

- the number of rotor types (with different internal wirings).

Each rotor would contain a different internal wiring. The substitution performed by each rotor should be unknown for the enemy. To make cryptanalysis more difficult and to ensure that the wiring inside each rotor changes with different frequency, the discs should rotate with different speeds.

Additionally, depending on the design of the machine, some additional features may be added to the machine, to ensure that the produced substitution is as random as possible (for example, an additional fixed substitution that does not depend on the rotors).

Some rotor machines (most notably Enigma) were designed to be symmetrical. That means that encrypting the same message twice (with the same settings), would produce the original message.

Cryptographic rotor machine

Hebern Rotor Machine

Hebern rotor machines were one of the first cryptographic machines, designed for encrypting messages automatically. It was first produced in 1917 in the USA.

Lorenz Rotor Machine

Lorenz cryptographic rotor machines were used by the Germans during World War II for encrypting strategic communication between major cities in occupied Europe.

Enigma

Enigma was one of the most famous and most popular cryptographic rotor machines, used mainly by the Germans during World War II.

Source code:

```
#include<conio.h>
#include<stdio.h>
char rotor1[26]={ 'f','u','e','t','r','g','z','d','s','h','y','l','k','w','j','i','b','x','m','p','c','n','a','q','v','o'};
char rotor2[26]={ 'q','w','e','r','t','y','u','i','o','p','a','s','d','f','g','h','j','k','l','z','x','c','v','b','n','m'};

int main()
{
    char input;
    int i,j;
    printf("Enter the string \n\n"); scanf("%c",&input);
    printf("Initial internal mapping of cylinder 1 is:\n");
    for(i=0;i<26;i++)
    {
        printf("%c ",rotor1[i]);
    }
    printf("\nInitial internal mapping of cylinder 2 is:\n");
    for(i=0;i<26;i++)
    {
        printf("%c ",rotor2[i]);
    }
}
```

```

printf("\n\n\tycylinder 1\tycylinder 2\n");
while(input!=' ')
{
i=input-97; printf("%c",input); printf("\t%c\t",rotor1[i]); j=rotor1[i]-97; printf("%c\n",rotor2[j]);

{
rotor1[i]=((rotor1[i]-97+2)%26)+'a';
rotor2[i]=((rotor2[i]-97+1)%26)+'a';
}

scanf("%c",&input);
}

printf("Final internal mapping of cylinder 1 is:\n");
for(i=0;i<26;i++)
{
printf("%c ",rotor1[i]);
}
printf("\nFinal internal mapping of cylinder 2 is:\n");
for(i=0;i<26;i++)
{
printf("%c ",rotor2[i]);
}
return 0;
}

```

Output :

```

C:\Users\bhumit\Desktop\rotor.exe
Enter the string
hello
Initial internal mapping of cylinder 1 is:
f u e t r g z d s h y l k w j i b x m p c n a q v o
Initial internal mapping of cylinder 2 is:
q w e r t y u i o p a s d f g h j k l z x c v b n m

      cylinder 1      cylinder 2
h      d              r
e      r              k
l      l              s
l      n              f
o      j              p
Final internal mapping of cylinder 1 is:
f u e t t g z f s h y p k w l i b x m p c n a q v o
Final internal mapping of cylinder 2 is:
q w e r u y u j o p a u d f h h j k l z x c v b n m
-----
Process exited after 6.268 seconds with return value 0
Press any key to continue . . .

```

Reference:

- <http://www.crypto-it.net/eng/simple/rotor-machines.html>
- William Stallings

Conclusion(Entire Subject):

Hence, the entire Information & Network Security subject can be concluded as that there are various techniques and methods available from protecting our confidential data from intruders who may try to steal this information.