CEASER CIPHER

Practical No: 1 Date:

Aim: Understand CEASER CIPHER.

Exercises: Write a C program to implement Caesar cipher encryption-decryption.

Encryption:

Program:

```
#include<stdio.h>

int main()
{
char message[100], ch;
 int i, key;
printf("Enter a message to encrypt: "); gets(message);
printf("Enter key: ");
 scanf("%d", &key);
for(i = 0; message[i] != '\0'; ++i){ ch = message[i];
if(ch >= 'a' && ch <= 'z'){ ch = ch + key;
if(ch > 'z'){
```

```
ch = ch - 'z' + 'a' - 1;
}
message[i] = ch;
}
else if(ch >= 'A' && ch <= 'Z'){ ch = ch + key;
if(ch > 'Z'){
ch = ch - 'Z' + 'A' - 1;
}
message[i] = ch;
}
}
printf("Encrypted message: %s", message); return 0;
}
```

Output: -

```
Enter a message to encrypt: Hello world
Enter key: 3
Encrypted message: Khoor zruog

...Program finished with exit code 0
Press ENTER to exit console.
```

Decryption:

Program:

```
#include<stdio.h>
int main()
{
char message[100], ch;
int i, key;
printf("Enter a message to decrypt: ");
gets(message);
printf("Enter key: ");
scanf("%d", &key);
for(i = 0; message[i] != '\0'; ++i){ ch = message[i] ;
if(ch >= 'a' && ch <= 'z'){ ch = ch - key;
if(ch < 'a'){
ch = ch + 'z' - 'a' + 1;
}
message[i] = ch;
}
else if(ch >= 'A' && ch <= 'Z'){ ch = ch - key;
if(ch < 'A'){
ch = ch + 'Z' - 'A' + 1;
}
message[i] = ch;
```

190120116078

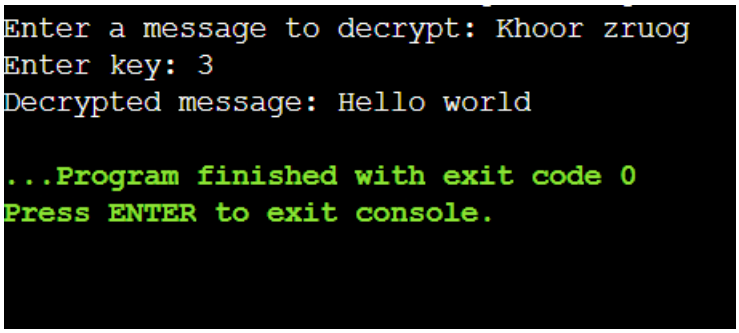
}

}

printf("Decrypted message: %s", message); return 0;

}

Output :-



MONOALPHABETIC CIPHER

Practical No: 2 Date:

Aim: Introduction to MONOALPHABETIC CIPHER.

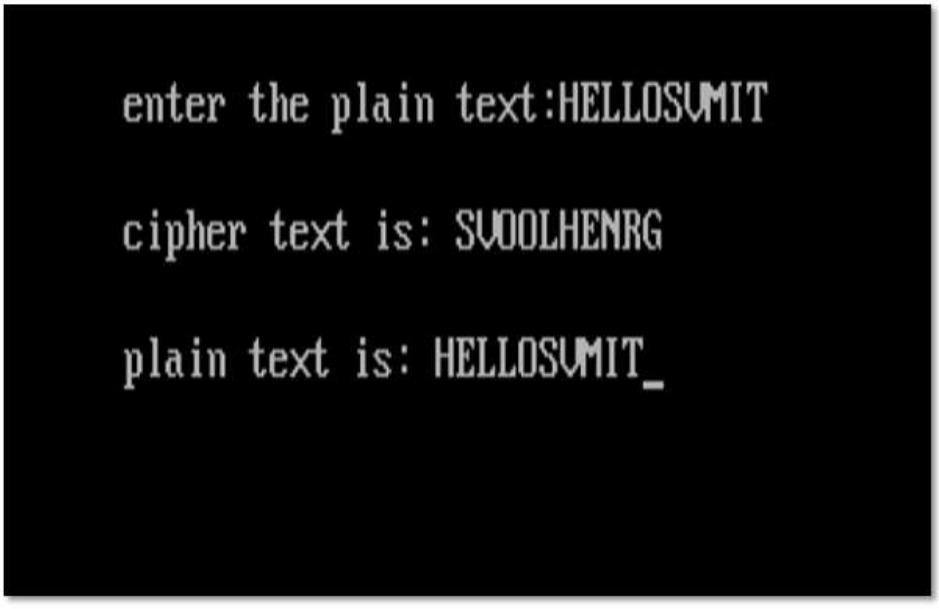
Exercises: Write a C program to implement Monoalphabetic cipher encryption-decryption.

Program:

```
//MONOALPHABATIC CIPHER
#include<stdio.h>
 #include<conio.h>
 #include<string.h>
void main()
{
char pt[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 ct[26]={'Z','Y','X','W','V','U','T','S','R','Q','P','O','N','M','L','K','J','I','H','G','F','E','D','C','B','A'};
char p[20]={'\0'},c[20]={'\0'},r[20]={'\0'};
int i,j; clrscr();
printf("\n enter the plain text:"); gets(p);
//converting plain text into cipher text (encryption) for(i=0;i<strlen(p);i++)
{
```

```
for(j=0;j<26;j++)
{
if(pt[j]==p[i])
{
c[i]=ct[j];
}}}
printf("\n cipher text is: %s",c);
//converting cipher text into plain text (decryption) for(i=0;i<strlen(c);i++)
{ for(j=0;j<26;j++){
if(ct[j]==c[i]) {
r[i]=pt[j];
}}
printf("\n \n plain text is: %s",r); getch();}
```

Output: -



PLAYFAIR CIPHER

Practical No: 3 Date:

Aim: Introduction to PLAYFAIR CIPHER**.**

Exercises: Write a C program to implement Playfair cipher encryption-decryption.

Program:

```
//PLAYFAIR CIPHER
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SIZE 30

// Function to convert the string to lowercase
void toLowerCase(char plain[], int ps)
{
   int i;
   for (i = 0; i < ps; i++) {
```

```
        if (plain[i] > 64 && plain[i] < 91)
            plain[i] += 32;
    }
}

// Function to remove all spaces in a string
int removeSpaces(char* plain, int ps)
{
    int i, count = 0;
    for (i = 0; i < ps; i++)
        if (plain[i] != ' ')
            plain[count++] = plain[i];
    plain[count] = '\0';
    return count;
}

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

    // a 26 character hashmap
    // to store count of the alphabet
    dicty = (int*)calloc(26, sizeof(int));
    for (i = 0; i < ks; i++) {
        if (key[i] != 'j')
```

```
        dicty[key[i] - 97] = 2;
    }

    dicty['j' - 97] = 1;

    i = 0;
    j = 0;

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

    for (k = 0; k < 26; k++) {
        if (dicty[k] == 0) {
            keyT[i][j] = (char)(k + 97);
            j++;
            if (j == 5) {
                i++;
```

```
                j = 0;
            }
        }
    }
}


// Function to search for the characters of a digraph
// in the key square and return their position
void search(char keyT[5][5], char a, char b, int arr[])
{
    int i, j;

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

    for (i = 0; i < 5; i++) {

        for (j = 0; j < 5; j++) {

            if (keyT[i][j] == a) {
                arr[0] = i;
                arr[1] = j;
            }
```

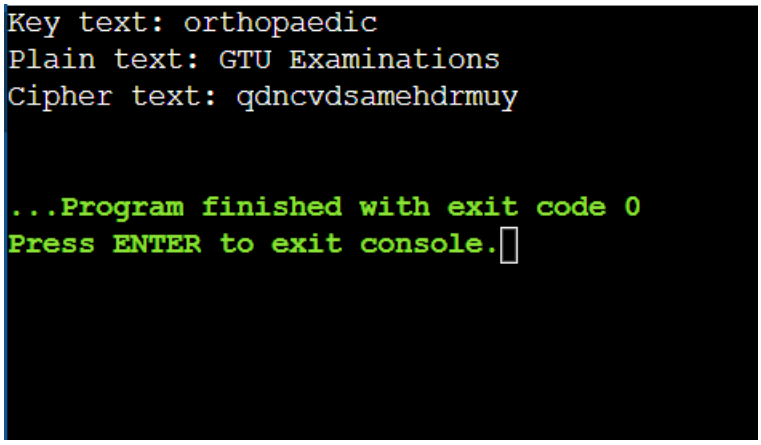
```
            else if (keyT[i][j] == b) {
                arr[2] = i;
                arr[3] = j;
            }
        }
    }
}

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

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

// Function for performing the encryption
void encrypt(char str[], char keyT[5][5], int ps)
{
    int i, a[4];
```

```
    for (i = 0; i < ps; i += 2) {

        search(keyT, str[i], str[i + 1], a);

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

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

    // Key
    ks = strlen(key);
```

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

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

    ps = prepare(str, ps);

    generateKeyTable(key, ks, keyT);

    encrypt(str, keyT, ps);
}

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

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

    // Plaintext to be encrypted
    strcpy(str, "GTU Examinations");
```

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

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

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

    return 0;
}
```

Output :-



POLYALPHABETIC

Practical No: 4 Date:

Aim: Introduction to POLYALPHABETIC.

Exercises: Write a C program to implement Polyalphabetic cipher encryption-decryption.

Program:

```
//POLYALPHABETIC
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
    char msg[30],key[30],k[20],ct[20],pt[20];
    int lenm,lenk,i,j;
    clrscr();

    printf("Enter Message : ");
    gets(msg);
    printf("Enter Key : ");
```

```
    gets(key);
    lenm=strlen(msg);
    lenk=strlen(key);
    for(i=0;i<lenm;i++,j++)
    {
        if(j==lenk)
        {
            j=0;
        }
        k[i]=key[j];
    }
    for(i=0;i<lenm;i++)
    {
        ct[i]=((msg[i]+k[i])%26)+'A';
    }
    ct[i]='\0';
    for(i=0;i<lenm;i++)
    {
        pt[i]=(((ct[i]-k[i])+26)%26)+'A';
    }
    pt[i]='\0';
    printf("\nEncrypted Message : %s", ct);
    printf("\nDecrypted Message : %s", pt);
    getch();
}
```

Output :-

```
Enter Message : WEAREDISCOVERED
Enter Key : DECEPTIVEDECEPT

Encrypted Message : ZICVTWQNGRZGVTW
Decrypted Message : WEAREDISCOVERED
```

190120116078

HILL CIPHER

Practical No: 5 Date:

Aim: Introduction to HILL CIPHER**.**

Exercises: Write a C program to implement Hill cipher encryption-decryption.

Program:

```
//HILL CIPHER

#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


void 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");
}}
```

190120116078

Output :-

```
Enter 3x3 matrix for key (It should be inversible):
6 24 1
13 16 10
20 17 15

Enter a 3 letter string: gtu

Encrypted string is: skp

Inverse Matrix is:
-551103840 -576171868 -576171868
-551103840 -576171868 -576171868
-551103840 -576171868 -576171868

Decrypted string is: gtu


...Program finished with exit code 0
Press ENTER to exit console.
```

DES Algorithm

Practical No: 6 Date:

Aim: Introduction to simple DES Algorithm.

Exercises: Write a C program to implement simple DES.

DES Program:

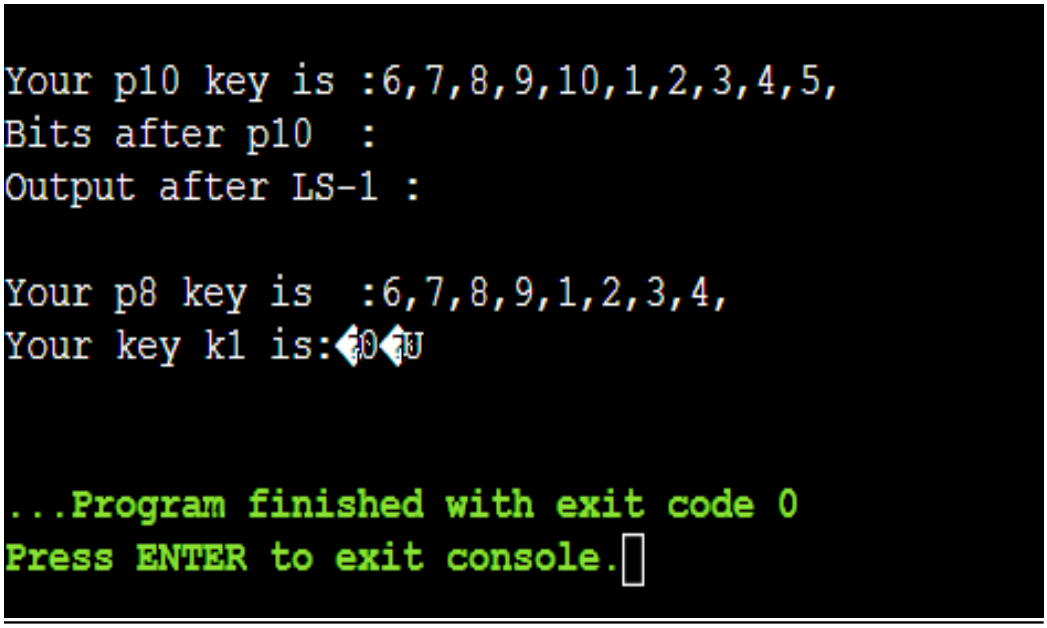
```
#include<stdio.h>
 void main()
{
int i, cnt=0, p8[8]={6,7,8,9,1,2,3,4};
int p10[10]={6,7,8,9,10,1,2,3,4,5};
char input[11], k1[10], k2[10], temp[11]; char LS1[5], LS2[5];
//k1, k2 are for storing interim keys
//p8 and p10 are for storing permutation key


//Read 10 bits from user... printf("Enter 10 bits input:"); scanf("%s",input);
input[10]='\0';
//Applying p10... for(i=0; i<10; i++)
{
```

```
cnt = p10[i];
temp[i] = input[cnt-1];
}
temp[i]='\0';
printf("\nYour p10 key is :"); for(i=0; i<10; i++)
{ printf("%d,",p10[i]); }
 printf("\nBits after p10:"); puts(temp);
//Performing LS-1 on first half of temp for(i=0; i<5; i++)
{
if(i==4) temp[i]=temp[0]; else temp[i]=temp[i+1];
}
//Performing LS-1 on second half of temp for(i=5; i<10; i++)
{
if(i==9) temp[i]=temp[5]; else temp[i]=temp[i+1];
}
printf("Output after LS-1 :"); puts(temp);

printf("\nYour p8 key is:"); for(i=0; i<8; i++)
{ printf("%d,",p8[i]); }
//Applying p8... for(i=0; i<8; i++)
{
cnt = p8[i];
k1[i] = temp[cnt-1];
 }
printf("\nYour key k1 is:"); puts(k1);
//This program can be extended to generate k2 as per DES algorithm.
```

190120116078

}

Output: -



190120116078

Diffie-Hellman Key Exchange

Practical No: 7 Date:

Aim: Introduction to Diffie-Hellman Key Exchange.

Exercises: Write a C program to implement Diffie-Hellman Key Exchange Method.

Program:

```
#include <conio.h>
#include <iostream>
#include <math.h>

int alice(int,int,int);
int bob(int,int,int);

using namespace std;

int main()
{
 long int g,x,y,a,b,k1,k2,n;

 cout<<"\n\t Enter value of n & g: ";
 cin>>n>>g;
```

```
 cout<<"\n\t Enter value of x & y: ";
 cin>>x>>y;

 a=alice(n,g,x);
 cout<<"\n\t alice end value:"<<a;

 b=bob(n,g,y);
 cout<<"\n\t bob end value:"<<b;

 k1=alice(n,b,x);
 cout<<"\n\t valueof k1 :"<<k1;

 k2=alice(n,a,y);
 cout<<"\n\t valueof k2 :"<<k2;

 getch();
}

int alice(int n,int g,int x)
{
 long int a,a1;
 a1=pow(g,x);
 a=a1%n;
 return(a);
}

int bob(int n,int g,int y)
{
 long int b,b1,k2,t2;
 b1=pow(g,y);
```

```
 b=b1%n;
 return(b);
}
```

Output:

```
        Enter value of n & g: 6 9

        Enter value of x & y: 9 8

        alice end value:3
        bob end value:3
        valueof k1 :3
        valueof k2 :3

...Program finished with exit code 0
Press ENTER to exit console.
```

190120116078

RSA Algorithm

Practical No: 8 **Date:**

Aim: Introduction to RSA encryption decryption algorithm**.**

Exercises: Write a C program to implement RSA encryption-decryption algorithm.

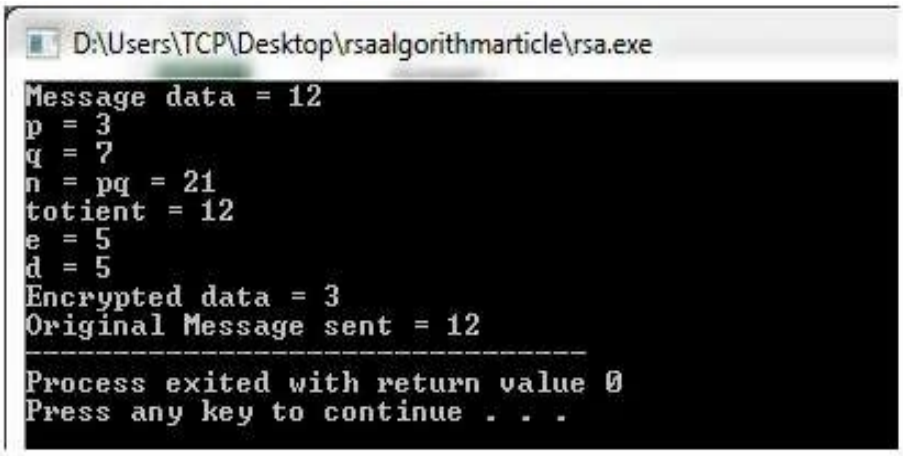
Program:

```
//Program for RSA asymmetric cryptographic algorithm
//for demonstration values are relatively small compared to practical application

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

//to find gcd
int gcd(int a, int h)
{
int temp; while(1)
```

```
{
temp = a%h; if(temp==0) return h;
a = h;
h = temp;
}
}

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

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

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

```
else
e++;
}

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

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

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

printf("Message data = %lf",msg); printf("\np = %lf",p);
printf("\nq = %lf",q); printf("\nn = pq = %lf",n); printf("\ntotient =
%lf",totient); printf("\ne = %lf",e); printf("\nd = %lf",d);
printf("\nEncrypted data = %lf",c); printf("\nOriginal Message Sent =
%lf",m);

return 0;
}
```

Output:


D:\Users\TCP\Desktop\rsaalgorithmarticle\rsa.exe
Message data = 12
p = 3
q = 7
n = pq = 21
totient = 12
e = 5
d = 5
Encrypted data = 3
Original Message sent = 12
Process exited with return value 0
Press any key to continue . . .

Digital Signature Algorithm

Practical No: 9 **Date:**

Aim: Introduction to Digital signature algorithm.

Exercises: Write a C program to implement digital signature algorithm.

Program:

```
#include <stdio.h>
 #include <conio.h>
#include <tchar.h>
 #include <windows.h>
#include <wincrypt.h>

// Link with the Crypt32.lib file. #pragma comment (lib, "Crypt32")
#define MY_ENCODING_TYPE (PKCS_7_ASN_ENCODING | X509_ASN_ENCODING)
// Define the name of a certificate subject.
// To use this program, the definition of SIGNER_NAME
```

```
// must be changed to the name of the subject of
// a certificate that has access to a private key. That certificate
// must have either the CERT_KEY_PROV_INFO_PROP_ID or
// CERT_KEY_CONTEXT_PROP_ID property set for the context to
// provide access to the private signature key.
// You can use a command similar to the following to create a
//certificate that can be used with this example:
//makecert -n "cn=Test" -sk Test -ss my

//#define SIGNER_NAME L"test"
#define SIGNER_NAME L"Insert_signer_name_here"
//                  Define the name of the store where the needed certificate
//                  can be found.

#define CERT_STORE_NAME L"MY"
// Local function prototypes. void MyHandleError(LPTSTR psz);
bool SignMessage(CRYPT_DATA_BLOB *pSignedMessageBlob); bool
VerifySignedMessage(
CRYPT_DATA_BLOB *pSignedMessageBlob, CRYPT_DATA_BLOB
*pDecodedMessageBlob);

int _tmain(int argc, _TCHAR* argv[])
```

```
{
UNREFERENCED_PARAMETER(argc);
UNREFERENCED_PARAMETER(argv);
CRYPT_DATA_BLOB SignedMessage; if(SignMessage(&SignedMessage))
{
CRYPT_DATA_BLOB DecodedMessage;

if(VerifySignedMessage(&SignedMessage, &DecodedMessage))
{
free(DecodedMessage.pbData);
}

free(SignedMessage.pbData);
}

_tprintf(TEXT("Press any key to exit."));
_getch();
}
//                  MyHandleError
void MyHandleError(LPTSTR psz)
{
_ftprintf(stderr, TEXT("An error occurred in the program. \n"));
```

```
_ftprintf(stderr, TEXT("%s\n"), psz);
_ftprintf(stderr, TEXT("Error number %x.\n"), GetLastError());
_ftprintf(stderr, TEXT("Program terminating. \n"));
} // End of MyHandleError
//                  SignMessage
bool SignMessage(CRYPT_DATA_BLOB *pSignedMessageBlob)
{
bool fReturn = false; BYTE* pbMessage; DWORD cbMessage;
HCERTSTORE hCertStore = NULL;
PCCERT_CONTEXT pSignerCert; CRYPT_SIGN_MESSAGE_PARA
SigParams; DWORD cbSignedMessageBlob;
BYTE *pbSignedMessageBlob = NULL;

// Initialize the output pointer. pSignedMessageBlob->cbData = 0;
pSignedMessageBlob->pbData = NULL;

// The message to be signed.
// Usually, the message exists somewhere and a pointer is
// passed to the application. pbMessage =
(BYTE*)TEXT("CryptoAPI is a good way to handle security");

// Calculate the size of message. To include the
```

```
// terminating null character, the length is one more byte
// than the length returned by the strlen function.
cbMessage = (lstrlen((TCHAR*) pbMessage) + 1) * sizeof(TCHAR);

// Create the MessageArray and the MessageSizeArray. const BYTE*
MessageArray[] = {pbMessage}; DWORD_PTR MessageSizeArray[1];
MessageSizeArray[0] = cbMessage;

// Begin processing.
_tprintf(TEXT("The message to be signed is \"%s\".\n"), pbMessage);

// Open the certificate store.
if ( !( hCertStore = CertOpenStore( CERT_STORE_PROV_SYSTEM,
0, NULL,
CERT_SYSTEM_STORE_CURRENT_USER, CERT_STORE_NAME)))
{
MyHandleError(TEXT("The MY store could not be opened.")); goto
exit_SignMessage;
}

// Get a pointer to the signer's certificate.
// This certificate must have access to the signer's private key. if(pSignerCert
= CertFindCertificateInStore(
```

```
hCertStore, MY_ENCODING_TYPE,
0, CERT_FIND_SUBJECT_STR, SIGNER_NAME,
NULL))
{
_tprintf(TEXT("The signer's certificate was found.\n"));
}
else
{
MyHandleError( TEXT("Signer certificate not found.")); goto
exit_SignMessage;
}

// Initialize the signature structure.
SigParams.cbSize = sizeof(CRYPT_SIGN_MESSAGE_PARA);
SigParams.dwMsgEncodingType = MY_ENCODING_TYPE;
SigParams.pSigningCert = pSignerCert; SigParams.HashAlgorithm.pszObjId
= szOID_RSA_SHA1RSA; SigParams.HashAlgorithm.Parameters.cbData =
NULL; SigParams.cMsgCert = 1;
SigParams.rgpMsgCert = &pSignerCert; SigParams.cAuthAttr = 0;
SigParams.dwInnerContentType = 0;
SigParams.cMsgCrl = 0;
SigParams.cUnauthAttr = 0;
```

```
SigParams.dwFlags = 0; SigParams.pvHashAuxInfo = NULL;
SigParams.rgAuthAttr = NULL;


// First, get the size of the signed BLOB. if(CryptSignMessage(
&SigParams, FALSE,
1,
MessageArray, MessageSizeArray, NULL,
&cbSignedMessageBlob))
{
_tprintf(TEXT("%d bytes needed for the encoded BLOB.\n"),
cbSignedMessageBlob);
}
else
{
MyHandleError(TEXT("Getting signed BLOB size failed")); goto
exit_SignMessage;
}


// Allocate memory for the signed BLOB. if(!(pbSignedMessageBlob =
(BYTE*)malloc(cbSignedMessageBlob)))
{
MyHandleError(
```

```
TEXT("Memory allocation error while signing.")); goto exit_SignMessage;
}

// Get the signed message BLOB. if(CryptSignMessage(
&SigParams, FALSE,
1,
MessageArray, MessageSizeArray, pbSignedMessageBlob,
&cbSignedMessageBlob))
{
_tprintf(TEXT("The message was signed successfully. \n"));

// pbSignedMessageBlob now contains the signed BLOB. fReturn = true;
}
else
{
MyHandleError(TEXT("Error getting signed BLOB")); goto
exit_SignMessage;
}

exit_SignMessage:

// Clean up and free memory as needed. if(pSignerCert)
```

```
{
CertFreeCertificateContext(pSignerCert);
}

if(hCertStore)
{
CertCloseStore(hCertStore, CERT_CLOSE_STORE_CHECK_FLAG);
hCertStore = NULL;
}
// Only free the signed message if a failure occurred. if(!fReturn)
{
if(pbSignedMessageBlob)
{
free(pbSignedMessageBlob); pbSignedMessageBlob = NULL;
}
}

if(pbSignedMessageBlob)
{
pSignedMessageBlob->cbData = cbSignedMessageBlob;
pSignedMessageBlob->pbData = pbSignedMessageBlob;
}
```

```
return fReturn;
}
//                VerifySignedMessage
//
//                Verify the message signature. Usually, this would be done in
//                a separate program. bool VerifySignedMessage(
CRYPT_DATA_BLOB *pSignedMessageBlob, CRYPT_DATA_BLOB
*pDecodedMessageBlob)
{
bool fReturn = false;
DWORD cbDecodedMessageBlob;
BYTE *pbDecodedMessageBlob = NULL;
CRYPT_VERIFY_MESSAGE_PARA VerifyParams;

// Initialize the output. pDecodedMessageBlob->cbData = 0;
pDecodedMessageBlob->pbData = NULL;

// Initialize the VerifyParams data structure. VerifyParams.cbSize =
sizeof(CRYPT_VERIFY_MESSAGE_PARA);
VerifyParams.dwMsgAndCertEncodingType = MY_ENCODING_TYPE;
VerifyParams.hCryptProv = 0;
```

```
VerifyParams.pfnGetSignerCertificate = NULL; VerifyParams.pvGetArg =
NULL;


// First, call CryptVerifyMessageSignature to get the length
// of the buffer needed to hold the decoded message.
if(CryptVerifyMessageSignature( &VerifyParams,
0,
pSignedMessageBlob->pbData, pSignedMessageBlob->cbData, NULL,
&cbDecodedMessageBlob, NULL))
{
_tprintf(TEXT("%d bytes needed for the decoded message.\n"),
cbDecodedMessageBlob);


}
else
{
_tprintf(TEXT("Verification message failed. \n")); goto
exit_VerifySignedMessage;
}
// Allocate memory for the decoded message. if(!(pbDecodedMessageBlob =
(BYTE*)malloc(cbDecodedMessageBlob)))
{
```

```
MyHandleError(
TEXT("Memory allocation error allocating decode BLOB.")); goto
exit_VerifySignedMessage;
}
// Call CryptVerifyMessageSignature again to verify the signature
// and, if successful, copy the decoded message into the buffer.
// This will validate the signature against the certificate in
// the local store. if(CryptVerifyMessageSignature(
&VerifyParams, 0,
pSignedMessageBlob->pbData, pSignedMessageBlob->cbData,
pbDecodedMessageBlob, &cbDecodedMessageBlob, NULL))
{
_tprintf(TEXT("The verified message is \"%s\".\n"),
pbDecodedMessageBlob);
fReturn = true;
}
else
{
_tprintf(TEXT("Verification message failed. \n"));
}

exit_VerifySignedMessage:
```

```
// If something failed and the decoded message buffer was
// allocated, free it. if(!fReturn)
{
if(pbDecodedMessageBlob)
{
free(pbDecodedMessageBlob); pbDecodedMessageBlob = NULL;
}
}

// If the decoded message buffer is still around, it means the
// function was successful. Copy the pointer and size into the
// output parameter. if(pbDecodedMessageBlob)
{
pDecodedMessageBlob->cbData = cbDecodedMessageBlob;
pDecodedMessageBlob->pbData = pbDecodedMessageBlob;
}

return fReturn;
```

190120116078

Cryptool

Practical No: 10 **Date:**

Aim: Introduction to cryptool.

Exercises: Perform any one of the above encryption technique with cryptool.

Program:

Encryption and Decryption of Caesar Cipher

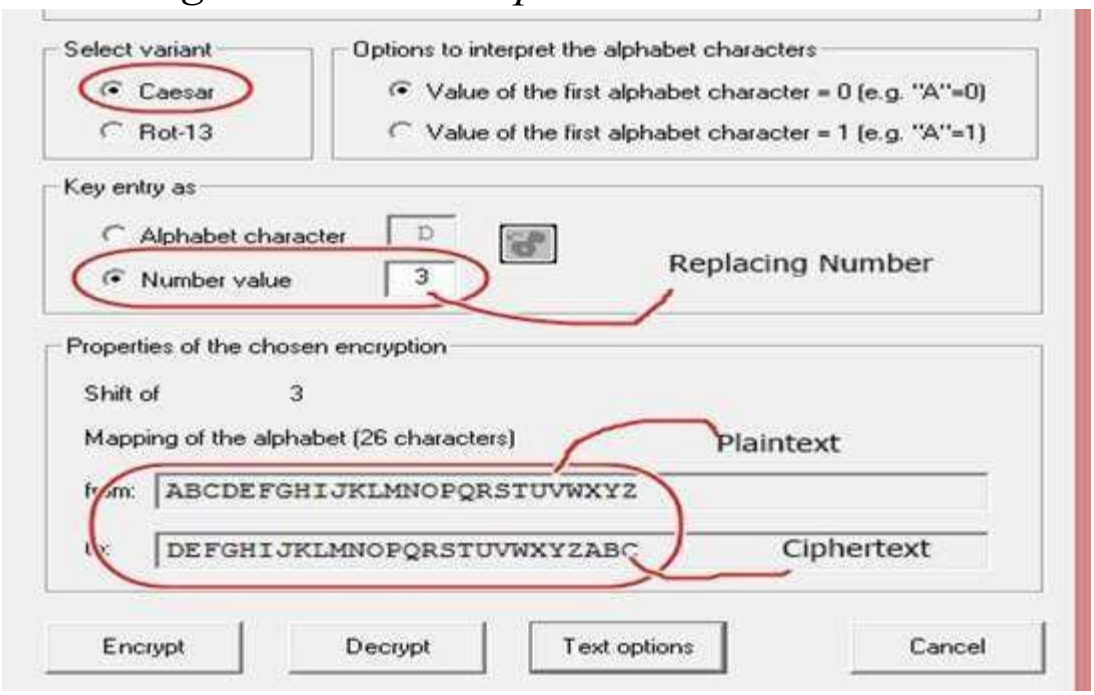
Here, we will implement an encryption and decryption of Caesar Cipher, which is actually a substitution method of cryptography. The Caesar Cipher involves replacing each letter of the alphabet with a letter – placed down or up according to the key given.

To start with the process you have to move to the Encrypt/Decrypt tab of the program. There, you will find Symmetric (Classic) tab - Choose Caesar Cipher. For further information, you can get guided by the image below.

Figure1: Encrypt/Decrypt of Cryptool

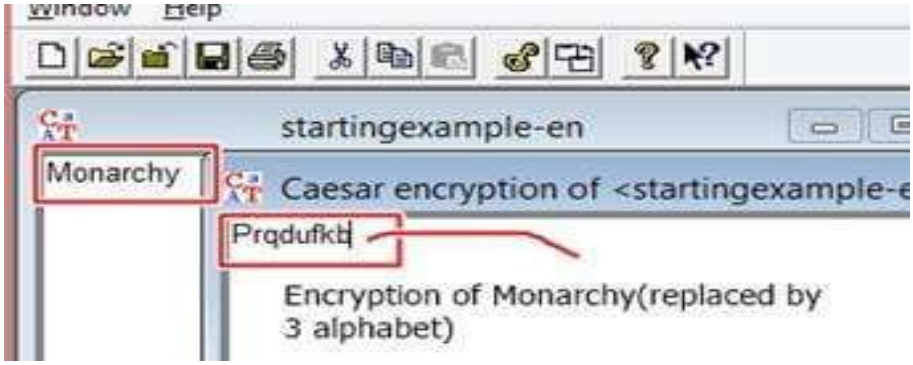
In encryption, we are replacing the plaintext letter with the 3rd letter of the alphabet that is if "A" is our plaintext character, then the Ciphertext will be "D".

Figure2: Caesar Cipher



So, if I give "Monarchy" as plaintext in Caesar Cipher, it will show me the encryption, as shown in the below image.

Figure3: Caesar Cipher Encryption



Encryption and Decryption of Playfair

Again, we have to move to Encrypt/Decrypt - Symmetric - Playfair Cipher and perform the encryption part. We are putting the same plaintext – MONARCHY.

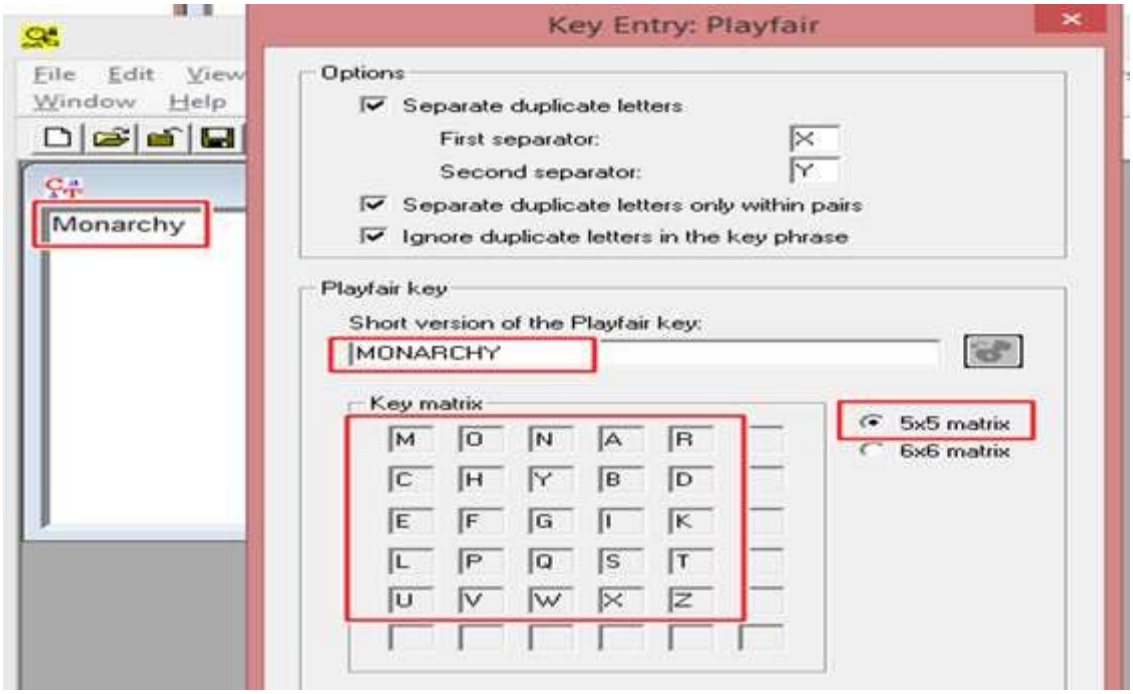


Figure4: Playfair Cipher

So, when we press the encrypt button, we will get the Ciphertext – “ONARMDYB”.

Figure5: Playfair Encryption

Encryption and Decryption of Hill Cipher

Again, we have to move to Encrypt/Decrypt - Symmetric - Hill Cipher and perform the encryption part. We are putting the plaintext as – DRGREERROCKS and assuming that the program gives us the Ciphertext as – FZIFTOTBXGPO.

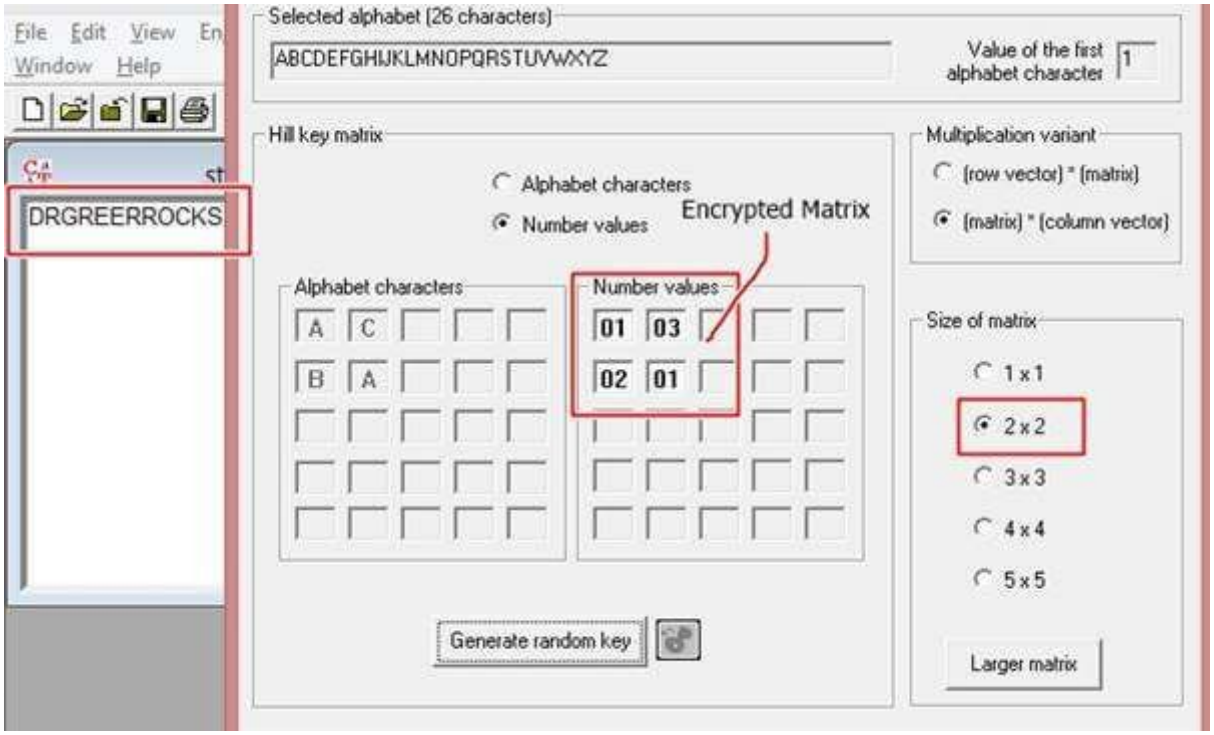


Figure6: Hill Cipher

So, when we press the encrypt button, we will get the Ciphertext – "FZIFTOTBXGPO".

Figure7: Hill Cipher Encryption

Encryption and Decryption of Vigener Cipher

Again, we have to move to Encrypt/Decrypt - Symmetric - Vigener Cipher and perform the encryption part. We are putting the plaintext as – MICHIGANTECHNOLOGICALUNIVERSITY and assuming that the program gives us the Ciphertext as – TWWNPZOAAS…..,with the help of key as – HOUGHTON.

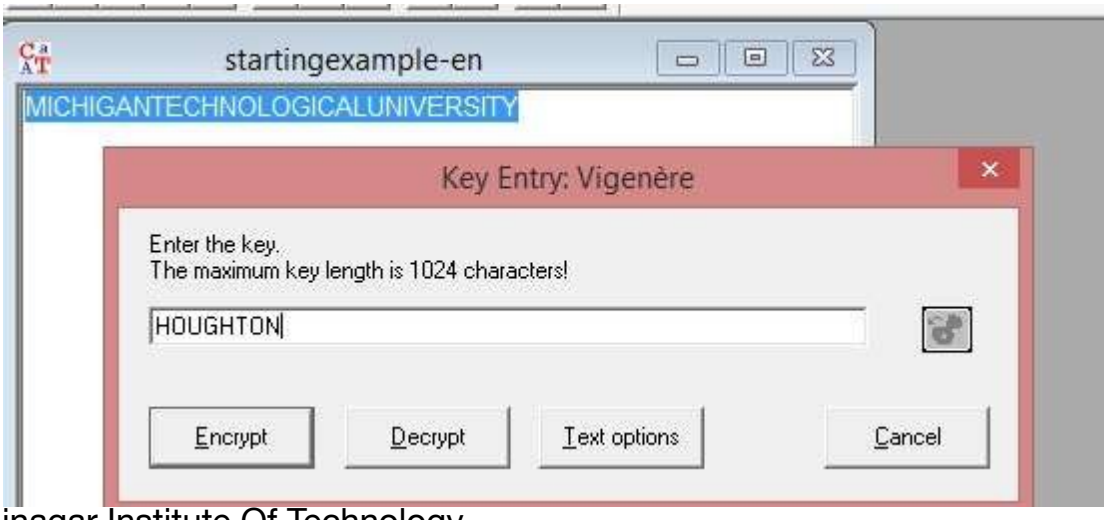
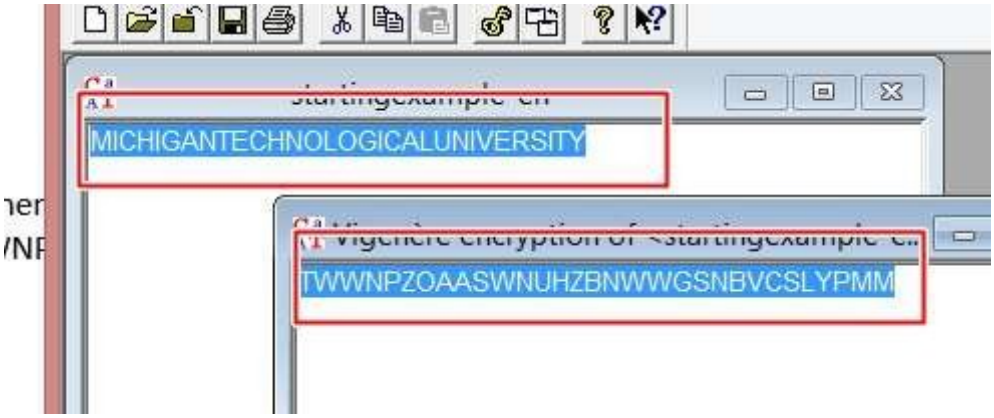


Figure8: Vigener Cipher

So, when we press the encrypt button, we will get the Ciphertext somewhat like – “TWWNPZOAASWNUHZBNWWGSNBVCSLYPMM”.

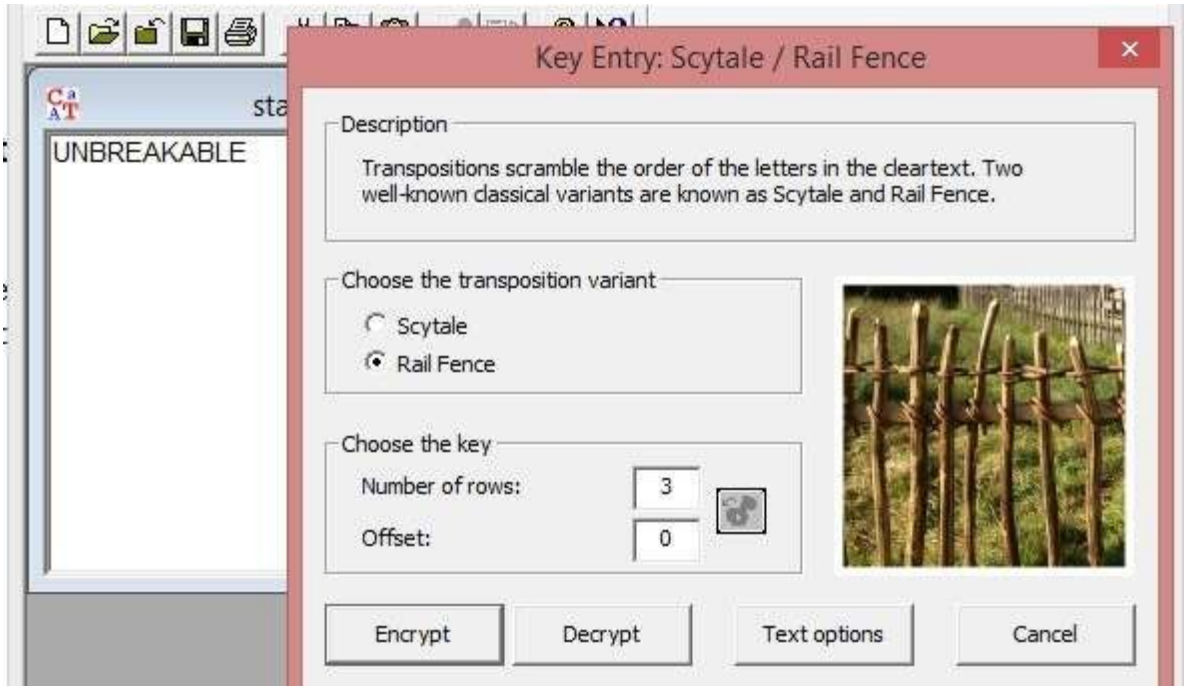
Figure9: Vigener Cipher Encryption



Encryption and Decryption of Railfence Cipher

Again, we have to move to Encrypt/Decrypt - Symmetric - Railfence Cipher and perform the encryption part. We are putting the plaintext as – UNBREAKABLE and assuming that the program gives us the Ciphertext as – UEBNRAALBKE…..,with the help of key as – 3.

Figure10: Railfence Cipher



So, when we press the encrypt button, we will get the Ciphertext like – “UEBNRAALBKE”.

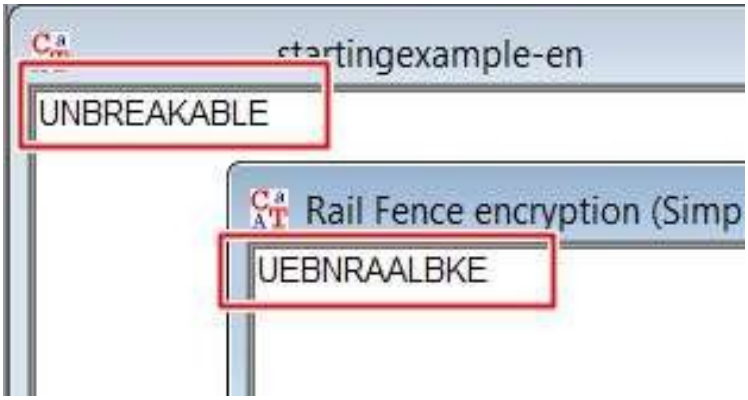


Figure11: Railfence Cipher Encryption

190120116078

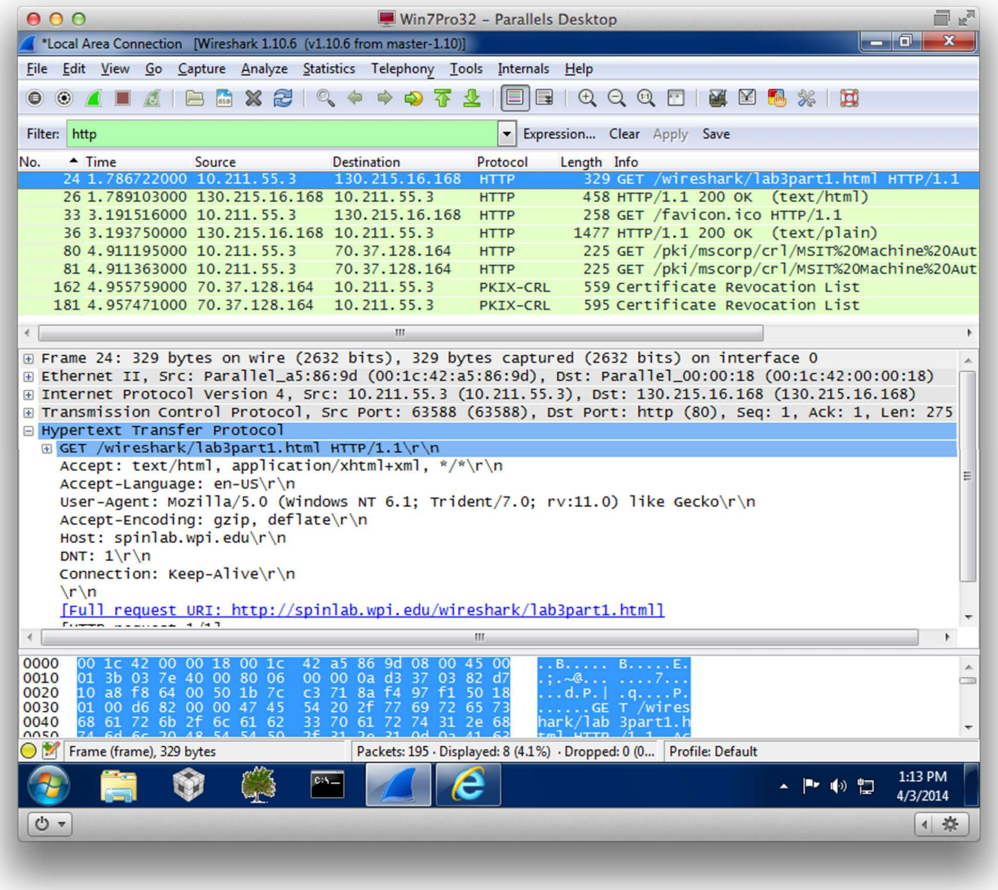
Wireshark

Practical No: 11 Date:

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

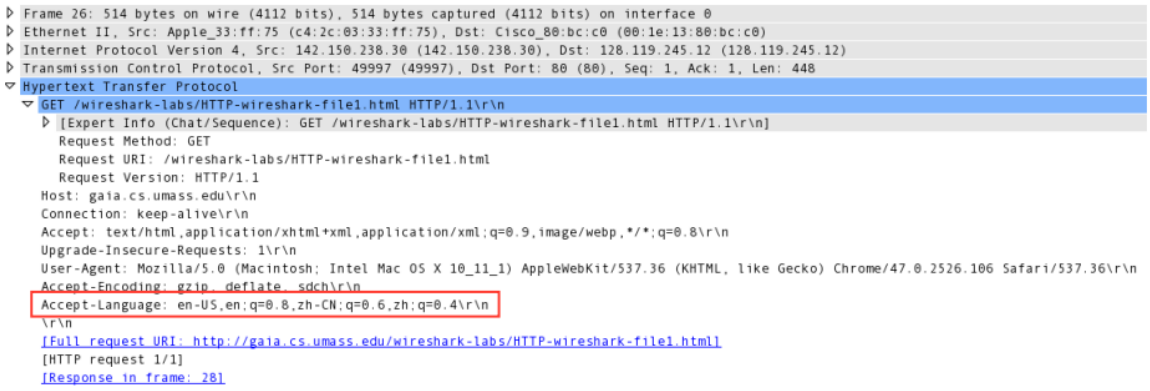
Exercises: Perform Below Exercises using Wireshark

1. Is your browser running HTTP version 1.0 or 1.1? What version of HTTP is the server running?



2. What languages (if any) does your browser indicate that it can accept to the server?

   en-US and zh-CN (languages information is listed in the item ‘Accept-Language’ in the HTTP GET message)



3. What is the IP address of your computer? Of the gaia.cs.umass.edu server?

   my computer: xxx.xxx.xxx.xxx

   gaia.cs.umass.edu: 128.119.245.12

```
▷ Frame 26: 514 bytes on wire (4112 bits), 514 bytes captured (4112 bits) on interface 0
▷ Ethernet II, Src: Apple_33:ff:75 (c4:2c:03:33:ff:75), Dst: Cisco_80:bc:c0 (00:1e:13:80:bc:c0)
▷ Internet Protocol Version 4, Src: 142.150.238.30 (142.150.238.30), Dst: 128.119.245.12 (128.119.245.12)
▷ Transmission Control Protocol, Src Port: 49997 (49997), Dst Port: 80 (80), Seq: 1, Ack: 1, Len: 448
▽ Hypertext Transfer Protocol
  ▽ GET /wireshark-labs/HTTP-wireshark-file1.html HTTP/1.1\r\n
    ▷ [Expert Info (Chat/Sequence): GET /wireshark-labs/HTTP-wireshark-file1.html HTTP/1.1\r\n]
      Request Method: GET
      Request URI: /wireshark-labs/HTTP-wireshark-file1.html
      Request Version: HTTP/1.1
    Host: gaia.cs.umass.edu\r\n
    Connection: keep-alive\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
    Upgrade-Insecure-Requests: 1\r\n
    User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36\r\n
    Accept-Encoding: gzip, deflate, sdch\r\n
    Accept-Language: en-US,en;q=0.8,zh-CN;q=0.6,zh;q=0.4\r\n
    \r\n
    [Full request URI: http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html]
    [HTTP request 1/1]
    [Response in frame: 28]
```

4. What is the status code returned from the server to your browser?

status code:200

(status code information is listed in the HTTP OK message)

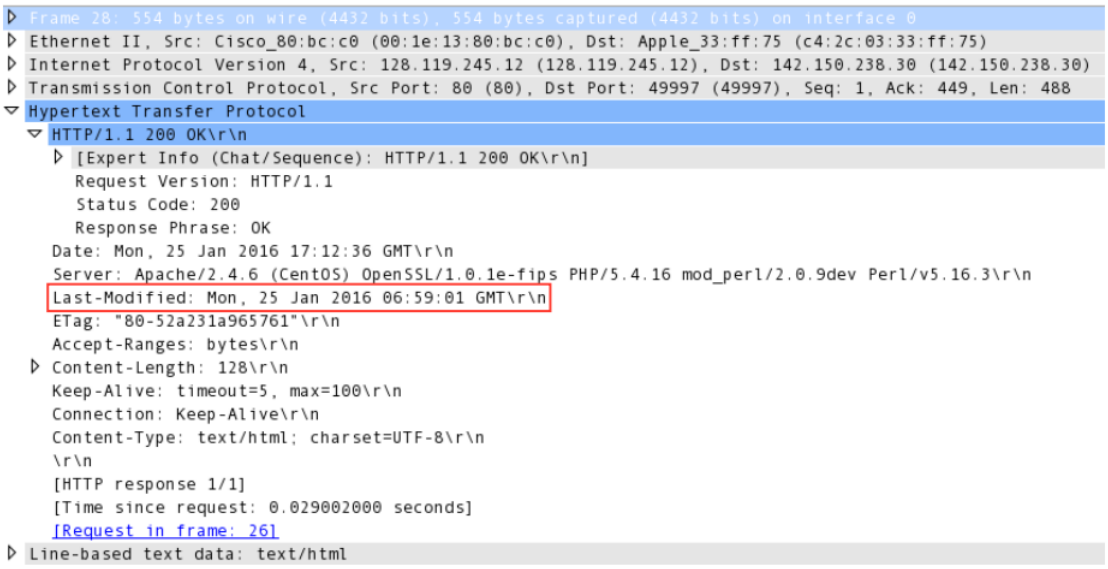
```
▷ Frame 28: 554 bytes on wire (4432 bits), 554 bytes captured (4432 bits) on interface 0
▷ Ethernet II, Src: Cisco_80:bc:c0 (00:1e:13:80:bc:c0), Dst: Apple_33:ff:75 (c4:2c:03:33:ff:75)
▷ Internet Protocol Version 4, Src: 128.119.245.12 (128.119.245.12), Dst: 142.150.238.30 (142.150.238.30)
▷ Transmission Control Protocol, Src Port: 80 (80), Dst Port: 49997 (49997), Seq: 1, Ack: 449, Len: 488
▽ Hypertext Transfer Protocol
  ▽ HTTP/1.1 200 OK\r\n
    ▷ [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
      Request Version: HTTP/1.1
      Status Code: 200
      Response Phrase: OK
    Date: Mon, 25 Jan 2016 17:12:36 GMT\r\n
    Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.1e-fips PHP/5.4.16 mod_perl/2.0.9dev Perl/v5.16.3\r\n
    Last-Modified: Mon, 25 Jan 2016 06:59:01 GMT\r\n
    ETag: "80-52a231a965761"\r\n
    Accept-Ranges: bytes\r\n
  ▷ Content-Length: 128\r\n
    Keep-Alive: timeout=5, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html; charset=UTF-8\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.029002000 seconds]
    [Request in frame: 26]
▷ Line-based text data: text/html
```

5. When was the HTML file that you are retrieving last modified at the server?

   Mon, 25 Jan 2016

   (last modified information is listed in the item ‘Last-Modified’ in the HTTP OK

   message)

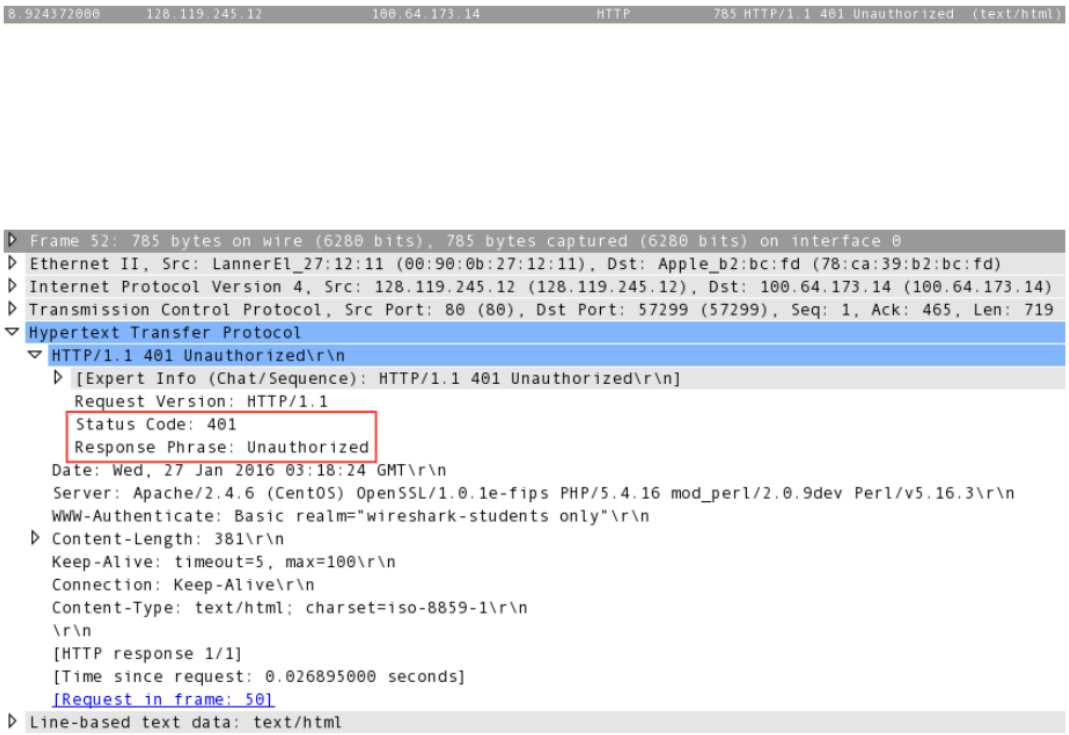


6. How many bytes of content are being returned to your browser?

   Content length: 128

   (Content length information is listed in the item ‘Content-Length’ in the HTTP OK

   message)



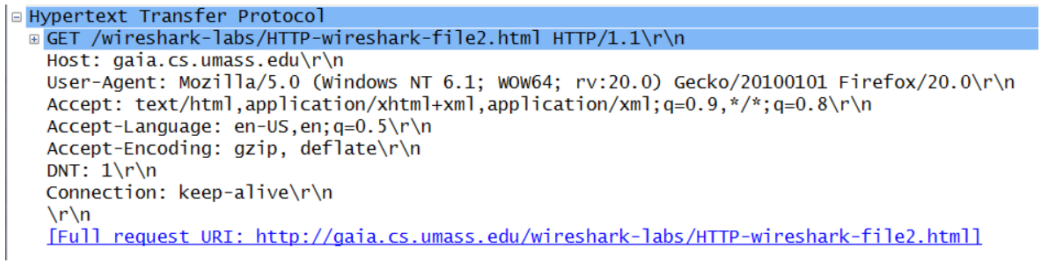
7. By inspecting the raw data in the packet content window, do you see any headers within the data that are not displayed in the packet-listing window? If so, name one.

   We got a response that said ‘HTTP/1.1 401 Unauthorized’.

   Status code: 401

   Response phrase: Unauthorized

8. Inspect the contents of the first HTTP GET request from your browser to the server. Do you see an ―IF-MODIFIED-SINCE‖ line in the HTTP GET?



```
Hypertext Transfer Protocol
  GET /wireshark-labs/HTTP-wireshark-file2.html HTTP/1.1\r\n
  Host: gaia.cs.umass.edu\r\n
  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:20.0) Gecko/20100101 Firefox/20.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
  Accept-Language: en-US,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate\r\n
  DNT: 1\r\n
  Connection: keep-alive\r\n
  \r\n
  [Full request URI: http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file2.html]
```

No, there is no “IF-MODIFIED-SINCE” line in the HTTP GET.

9. Inspect the contents of the server response. Did the server explicitly return the contents of the file? How can you tell?

```
Line-based text data: text/html
  \n
  <html>\n
  \n
  Congratulations again!  Now you've downloaded the file lab2-2.html. <br>\n
  This file's last modification date will not change.  <p>\n
  Thus  if you download this multiple times on your browser, a complete copy <br>\n
  will only be sent once by the server due to the inclusion of the IN-MODIFIED-SINCE<br>\n
  field in your browser's HTTP GET request to the server.\n
  \n
  </html>\n
```

Yes, the server explicitly returned the contents of the file. I am able to tell this because of the Line-based text data in the OK response to the GET.

10. Now inspect the contents of the second HTTP GET request from your browser to the server.

Do you see an ―IF-MODIFIED-SINCE:‖ line in the HTTP GET? If so, what information

follows the ―IF-MODIFIED-SINCE:‖ header?

```
Hypertext Transfer Protocol
  GET /wireshark-labs/HTTP-wireshark-file2.html HTTP/1.1\r\n
  Host: gaia.cs.umass.edu\r\n
  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:20.0) Gecko/20100101 Firefox/20.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
  Accept-Language: en-US,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate\r\n
  DNT: 1\r\n
  Connection: keep-alive\r\n
  If-Modified-Since: Sun, 05 May 2013 23:46:01 GMT\r\n
  If-None-Match: "d6c96-173-2fbf8040"\r\n
  Cache-Control: max-age=0\r\n
  \r\n
  [Full request URI: http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file2.html]
```

An "IF-MODIFIED-SINCE:" line in the HTTP GET was present. The information that followed the "IF-MODIFIED-SINCE:" header that was not present in the other HTTP GET was the date it was modified, a match query, and a cache-control (I suppose this is part of why the cache had to be cleared first).

11. What is the HTTP status code and phrase returned from the server in response to this second HTTP GET? Did the server explicitly return the contents of the file? Explain.

```
41 19:47:47.160881000 10.33.47.177   128.119.245.12   HTTP   506 GET /wireshark-labs/HTTP-wireshark-file2.html HTTP/1.1
42 19:47:47.195817000 128.119.245.12 10.33.47.177     HTTP   701 HTTP/1.1 200 OK  (text/html)
```

The HTTP status code that the server responded with was a 200 OK, surprisingly. This means that the server explicitly returned the contents of the file. After a Google search, it appears this was not supposed to happen.