Karnataka Law Society's

GOGTE INSTITUTE OF TECHNOLOGY

Udyambag Belagavi -590008

Karnataka, India.

Department of Computer Science Engineering

Course Project Report

Submitted in the partial fulfillment for the academic requirement of

6th semester B.E.

In

Information and Network Security

Submitted by

Sl.no	Name	USN
01	Prathamesh Kesti	2GI19CS096

Under the Guidance of

Prof. Naitik Suryavanshi

2021-22

Department of Computer Science and Engineering

Team Members Details:

Sl.no	Name	USN
01	Prathamesh Kesti	2GI19CS096

Marks Allocation:

	Batch No.:					
1.	Seminar Title:	Marks Range	**USN**			
			2GI19CS096			
2.	Abstract (PO2)	0-2				
3.	Application of the topic to the course (PO2)	0-3				
4.	Literature survey and its findings (PO2)	0-4				
5.	Methodology, Results and Conclusion (PO1, PO3, PO4)	0-6				
6.	Report and Oral presentation skill (PO9, PO10)	0-5				
	Total	20				

Signature of Staff

Karnataka Law Society's

GOGTE INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the course-based project entitled "**Information Network and Security**" is a Bonafide work done by Prathamesh Kesti(2GI19CS096 in partial fulfilment of the requirement forthe award of degree in "BACHELOR OF ENGINEERING in Computer Science Engineering"during the academic year 2021-2022.

Faculty In Charge
Prof. Naitik Suryavanshi

Head of the Department
Dr. Prof. V.S. Rajpurohit

Question 1:

Write a program that can encrypt and decrypt using the general Caesar cipher, also known as an additive cipher.

The Caesar Cipher technique is one of the earliest and simplest methods of encryption technique. It's simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter with a 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 a shift which indicates the number of positions each letter of the text has been moved down.

The encryption can be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A = 0, B = 1,…, Z = 25.

Example:

Text: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Shift: 23
Cipher: XYZABCDEFGHIJKLMNOPQRSTUVW

Text: ATTACKATONCE
Shift: 4
Cipher: EXXEGOEXSRGI

Algorithm for Caesar Cipher:
Input:

1. A String of lower-case letters, called Text.
2. An Integer between 0-25 denoting the required shift.

Procedure:

- Traverse the given text one character at a time .
- For each character, transform the given character as per the rule, depending on whether we're encrypting or decrypting the text.
- Return the new string generated.

SOURCE CODE:

Encryption:

```
#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: ATTACKATONCE

Enter key: 4

Encrypted message: EXXEGOEXSRGI

Decryption

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

OUTPUT:

Enter a message to decrypt: EXXEGOEXSRGI

Enter key: 4

decrypted message: ATTACKATONCE

Question 2:

Create software that can encrypt and decrypt using a 2 × 2 Hill cipher.

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 × 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 × n matrices (modulo 26).

Example:

Encryption

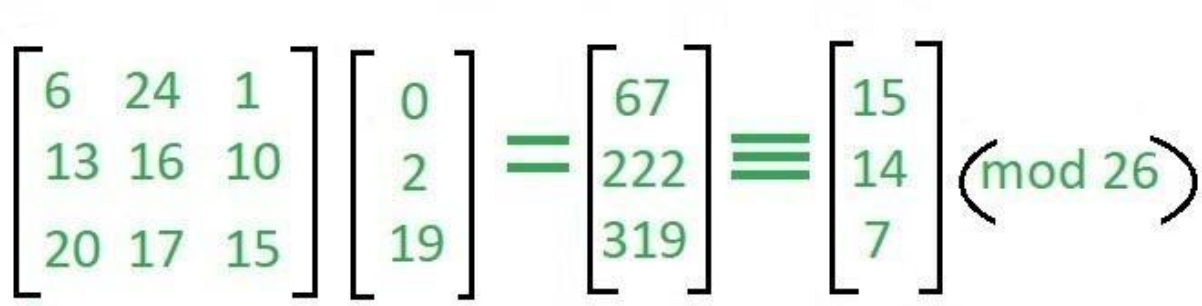
We have to encrypt the message ‘ACT’ (n=3). The key is ‘GYBNQKURP’ which can be written as the n x n matrix:

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix}$$

The message ‘ACT’ is written as vector:

$$\begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix}$$

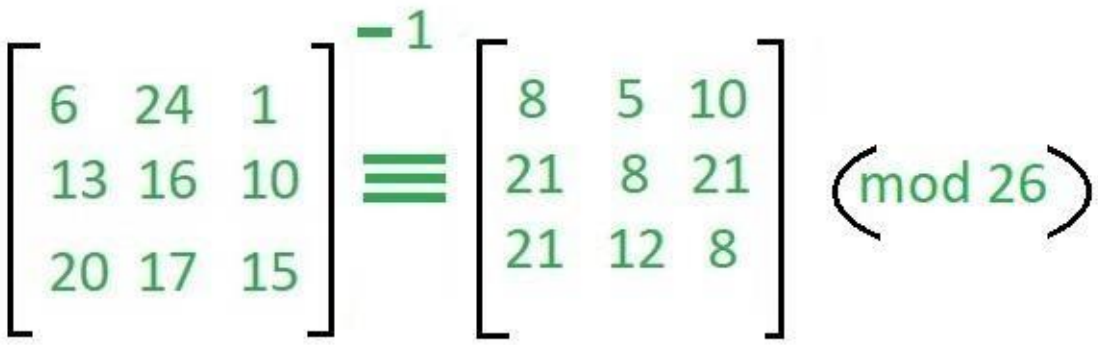
The enciphered vector is given as:



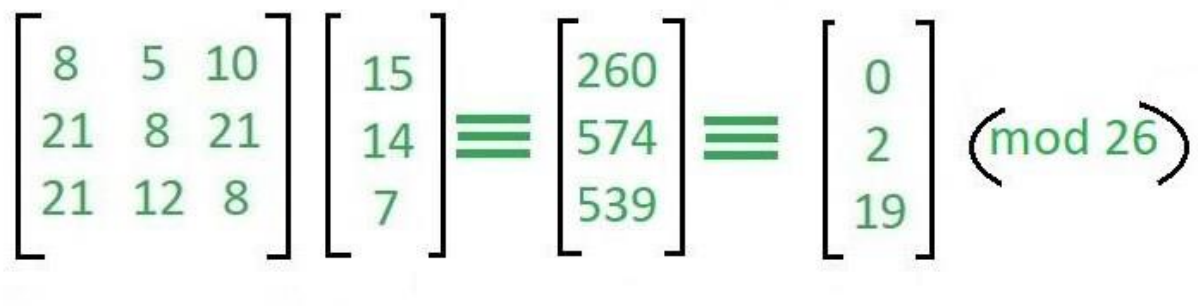
which corresponds to ciphertext of ‘POH’.

Decryption:

To decrypt the message, we turn the ciphertext back into a vector, then simply multiply by the inverse matrix of the key matrix (IFKVIVVMI in letters). The inverse of the matrix used in the previous example is:



For the previous Ciphertext ‘POH’:



SOURCE CODE:

Encryption:

```
#include<iostream>
#include<vector>
using namespace std;
int main(){
int x,y,i,j,k,n;
cout<<"Enter the size of key matrix\n";
cin>>n;
cout<<"Enter the key matrix\n";
int a[n][n];
for(i=0;i<n;i++){
for(j=0;j<n;j++){
cin>>a[i][j];
}
}
cout<<"Enter the message to encrypt\n";
string s;
cin>>s;
int temp = (n-s.size()%n)%n;
for(i=0;i<temp;i++){
s+='x';
}
k=0;
string ans="";
while(k<s.size()){
for(i=0;i<n;i++){
int sum = 0;
int temp = k;
for(j=0;j<n;j++){
sum += (a[i][j]%26*(s[temp++]-'a')%26)%26;
sum = sum%26;
}
ans+=(sum+'a');
}
k+=n;
}
```

```
cout<<ans<<'\n';
return 0;
}
```

OUTPUT:

```
Enter the size of key matrix 2
Enter the key matrix
4 1
3 2
Enter the message to encrypt: helloworld
Encrypted message: gdddaivyvn
```

Decryption:

```
#include<iostream>
#include<vector>
using namespace std;
int modInverse(int a, int m){
a=a%m;
for(int x=-m;x<m;x++)
if((a*x)%m==1)
return x;
}
void getCofactor(vector<vector<int> > &a, vector<vector<int> > &temp, int p, int q,
int n){
int i=0,j=0;
for(int row=0;row<n;row++){
for(int col=0;col<n;col++){
if(row!=p&&col!=q){
temp[i][j++] = a[row][col];
if (j==n-1){
j=0;
i++;
}
}
}
}
}
```

```
int determinant(vector<vector<int> > &a, int n, int N){
int D = 0;
if(n==1)
return a[0][0];
vector<vector<int> > temp(N, vector<int>(N));
int sign = 1;
for(int f=0;f<n;f++){
getCofactor(a, temp, 0, f, n);
D += sign * a[0][f] * determinant(temp, n - 1, N);
sign = -sign;
}
return D;
}
void adjoint(vector<vector<int> > &a,vector<vector<int> > &adj,int N){
if(N == 1){
adj[0][0] = 1;
return;
}
int sign = 1;
vector<vector<int> > temp(N, vector<int>(N));
for(int i=0;i<N;i++){
for(int j=0;j<N;j++){
getCofactor(a, temp, i, j, N);
sign = ((i+j)%2==0)? 1: -1;
adj[j][i] = (sign)*(determinant(temp, N-1 , N));
}
}
}
bool inverse(vector<vector<int> > &a, vector<vector<int> > &inv, int N){
int det = determinant(a, N, N);
if(det == 0){
cout << "Inverse does not exist";
return false;
}
int invDet = modInverse(det,26);
cout<<det%26<<' '<<invDet<<'\n';
vector<vector<int> > adj(N, vector<int>(N));
adjoint(a, adj, N);
for(int i=0;i<N;i++)
for(int j=0;j<N;j++)
inv[i][j] = (adj[i][j]*invDet)%26;
return true;
}
```

```
int main(){
int x,y,i,j,k,n;
cout<<"Enter the size of key matrix\n";
cin>>n;
cout<<"Enter the key matrix\n";
vector<vector<int> > a(n, vector<int>(n));
vector<vector<int> > adj(n, vector<int>(n));
vector<vector<int> > inv(n, vector<int>(n));
for(i=0;i<n;i++){
for(j=0;j<n;j++){
cin>>a[i][j];
}
}
if(inverse(a,inv,n)){
cout<<"Inverse exist\n";
}
cout<<"Enter the message to decrypt\n";
string s;
cin>>s;
k=0;
string ans;
while(k<s.size()){
for(i=0;i<n;i++){
int sum = 0;
int temp = k;
for(j=0;j<n;j++){
sum += ((inv[i][j] + 26)%26*(s[temp++]-'a')%26)%26;
sum = sum%26;
}
ans+=(sum+'a');
}
k+=n;
}
//ans+='\0';
int f=ans.size()-1;
while(ans[f]=='x'){
f--;
}
for(i=0;i<=f;i++){
cout<<ans[i];
}
cout<<'\n';
return 0; }
```

OUTPUT:

```
Enter the size of key matrix: 2
Enter the key matrix
4 1
3 2
5 21
Inverse exist
Enter the message to decrypt: gdddaivyvn
Decrypted message: helloworld
```

Question 3:

In the RSA public-key encryption scheme, each user has a public key, e, and a private key, d. Suppose Bob leaks his private key. Rather than generating a new modulus, he decides to generate a new public and a new private key. Is this safe?

Solution:

No, it is not safe. Once Bob leaks his private key, Alice can use this to factor his modulus, N. Then Alice can crack any message that Bob sends.

Here is one way to factor the modulus:

Let **k= ed – 1**. Then k is congruent to 0 mod φ(N) (where 'φ' is the Euler totient function). Select a random x in the multiplicative group Z(N). Then $x_k \equiv 1 \bmod N$, which implies that $x_{k/2}$ is a square root of 1 mod N. With 50% probability, this is a nontrivial square root of N, so that $\gcd(x_{k/2} - 1, N)$ will yield a prime factor of N. If $x^{k/2} = 1 \bmod N$, then try $x^{k/4}$, $x^{k/8}$, etc...

This will fail if and only if $x^{k/2i} \equiv -1$ for some i.

If it fails, then choose a new x. This will factor N in expected polynomial time.

Question 4:

The principal objective of the IBM Cryptographic Subsystem is to protect transmissions between a terminal and the processing system. Devise a procedure, perhaps adding instructions, which will allow the processor to generate a session key KS and distribute it to Terminal i and Terminal j without having to store a key-equivalent variable in the host.

Solution:

One solution is to add an instruction similar to RFMK of the form KEYGEN[RN, KMT,, KMT]

which will interpret RN as E(KMH0, KS) and return both E(KMH,, KS) and E(KMH, KS), which are sent to the terminals i and j, respectively. RN need not be maintained at the host.