



**MIT-ADT**  
**UNIVERSITY**  
PUNE, INDIA

A leap towards World Class Education

(Established by MIT Art, Design and Technology University Act, 2015  
(Maharashtra Act No. XXXIX of 2015))

**MIT Art Design and Technology University**  
**MIT School of Computing, Pune**

**Department of Computer Science and  
Engineering**

# **Lab Manual**

**Practical - Programming Laboratory –V**  
**( CC+ IS )**

**Class - L.Y. (SEM-I), <Core-II>**

**A.Y. 2023 - 2024**

# INDEX

Sr. No.	Title	Page No.
1	Write a C program that contains a string (char pointer) with a value \Hello World'. The program should AND or and XOR 2 each character in this string with 127 and display the result.	03
2	Write a Java program to perform encryption and decryption using Caesar Cipher algorithm	07
3	Write a Program to Implement RSA Algorithm.	14
4	Write a Program to Simulate Diffie-Hellman Key Exchange	21
5	Implementation of Columnar Transposition Technique	27

## Experiment No:1

**Title:** Write a C program that contains a string (char pointer) with a value 'Hello World'. The program should AND or and XOR 2 each character in this string with 127 and display the result.

After completion of this experiment students will be able to:

Learn AND, OR and XOR operations on string

**Aim:** Program for AND OR and XOR each character in string operation

**Theory:**

### XOR Operation

There are two inputs and one output in binary XOR (exclusive OR) operation. It is similar to ADD operation which takes two inputs and produces one result i.e. one output.

The inputs and result to a binary XOR operation can only be 0 or 1. The binary XOR operation will always produce a 1 output if either of its inputs is 1 and will produce a 0 output if both of its inputs are 0 or 1.

XOR Truth Table

Input		Output
X	Y	
0	0	0
0	1	1
1	0	1
1	1	0

**Algorithm/Pseudocode:**

1. Start
2. Declare array for storing string
3. Take input from user as string

4. Read the string
5. Find the length of string using built in function
6. Perform AND operation display result character by character
7. Perform XOR operation display result character by character
8. Perform OR operation display result character by character
9. Stop

**Program:**

```
#include <stdio.h>

#include<string.h>

int main()

{

char str[24],c[4],a[5],b[5];

int i,x;

printf("Enter the string:");

scanf("%s",str);

x=strlen(str);

printf("\n AND operation: ");

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

{

a[i]=str[i]&127;

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

```
}  
  
printf("\n XOR operation: ");  
for(i=0;i<=x;i++)  
{  
    b[i]=str[i]^127;  
    printf("%c",b[i]);  
}  
  
printf("\n OR operation: ");  
for(i=0;i<=x;i++)  
{  
    c[i]=str[i]|127;  
    printf("%c",c[i]);  
}  
}
```

### **OUTPUT:**

Enter the string:

HelloWorld

AND operation: HelloWorld

XOR operation: 7

OR operation:

**Conclusion:**

Hence we learn to take input from user perform AND, XOR, OR operations on sting.

**References:**

1. William Stallings, "Cryptography and Network Security", Sixth Edition Pearson Education.
2. Atul Kahate, "Cryptography and Network Security", Tata McGraw-Hill.

## Experiment No:2

**Title:** Write a Java program to perform encryption and decryption using Caesar Cipher algorithm

After completion of this experiment students will be able to:

Learn caesar cipher substitution technique

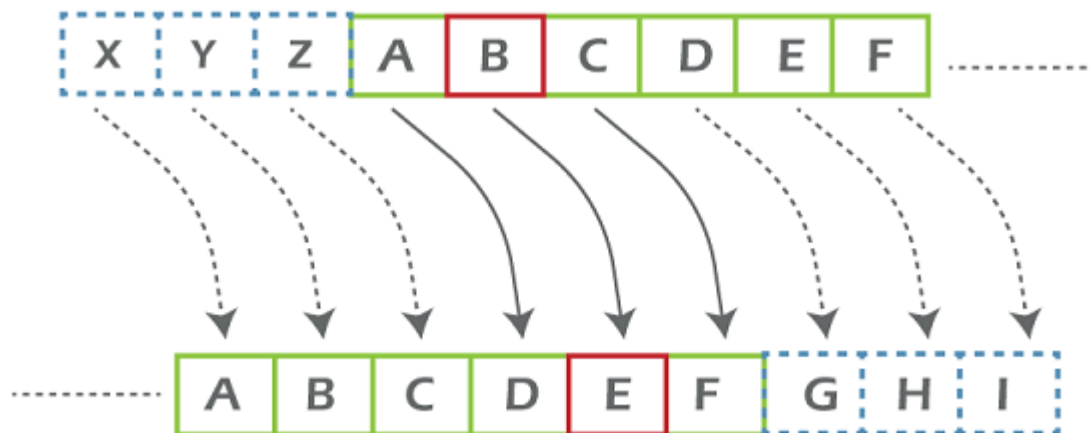
**Aim:** Program for Caesar cipher.

### Theory:

It is one of the simplest and most used encryption techniques. In this technique, each letter of the given text is replaced by a letter of some fixed number of positions down the alphabet.

For example, with a shift of 1, X would be replaced by Y, Y would become Z, and so on. Julius Caesar was the first one who used it for communicating with his officials. Based on his name, this technique was named as Caesar Cipher technique.

An integer value is required to cipher a given text. The integer value is known as shift, which indicates the number of positions each letter of the text has been moved down.



We can mathematically represent the encryption of a letter by a shift  $n$  in the following way:

Encryption phase with shift  $n = E_n(x) = (x+n) \bmod 26$

Decryption phase with shift  $n = D_n(x) = (x-n) \bmod 26$

### Examples

**Text :** ABCDEFGHIJKLMNOPQRSTUVWXYZ

**Shift :** 23

**Cipher :** XYZABCDEFGHIJKLMNOPQRSTUVW

**Text :** ATTACKATONCE

**Shift :** 4

**Cipher :** EXXEGOEXSRGI

### Algorithm/Pseudocode:

1. Take an input string from the user to encrypt it using the Caesar Cipher technique.
2. Take an input integer from the user for shifting characters. The input integer should be between 0-25.
3. Traverse input string one character at a time.
4. Depending on the encryption and decryption, we transform each character as per the rule.
5. Returns the newly generated string.

### Program:

```
// import required classes and package, if any  
  
import java.util.Scanner;
```



```
// create class CaesarCipherExample for encryption and decryption

public class CaesarCipherExample
{
    // ALPHABET string denotes alphabet from a-z

    public static final String ALPHABET = "abcdefghijklmnopqrstuvwxyz";

    // create encryptData() method for encrypting user input string with given shift key

    public static String encryptData(String inputStr, int shiftKey)
    {
        // convert inputStr into lower case

        inputStr = inputStr.toLowerCase();

        // encryptStr to store encrypted data

        String encryptStr = "";

        // use for loop for traversing each character of the input string

        for (int i = 0; i < inputStr.length(); i++)
        {
            // get position of each character of inputStr in ALPHABET

            int pos = ALPHABET.indexOf(inputStr.charAt(i));

            // get encrypted char for each char of inputStr

            int encryptPos = (shiftKey + pos) % 26;
```

```

        char encryptChar = ALPHABET.charAt(encryptPos);

        // add encrypted char to encrypted string
        encryptStr += encryptChar;
    }

    // return encrypted string
    return encryptStr;
}

// create decryptData() method for decrypting user input string with given shift key
public static String decryptData(String inputStr, int shiftKey)
{
    // convert inputStr into lower case
    inputStr = inputStr.toLowerCase();

    // decryptStr to store decrypted data
    String decryptStr = "";

    // use for loop for traversing each character of the input string
    for (int i = 0; i < inputStr.length(); i++)
    {

```

```

// get position of each character of inputStr in ALPHABET
int pos = ALPHABET.indexOf(inputStr.charAt(i));

// get decrypted char for each char of inputStr
int decryptPos = (pos - shiftKey) % 26;

// if decryptPos is negative
if (decryptPos < 0){
    decryptPos = ALPHABET.length() + decryptPos;
}
char decryptChar = ALPHABET.charAt(decryptPos);

// add decrypted char to decrypted string
decryptStr += decryptChar;
}

// return decrypted string
return decryptStr;
}

// main() method start
public static void main(String[] args)

```

```
{  
    // create an instance of Scanner class  
  
    Scanner sc = new Scanner(System.in);  
  
    // take input from the user  
  
    System.out.println("Enter a string for encryption using Caesar Cipher: ");  
  
    String inputStr = sc.nextLine();  
  
    System.out.println("Enter the value by which each character in the plaintext  
message gets shifted: ");  
  
    int shiftKey = Integer.valueOf(sc.nextLine());  
  
    System.out.println("Encrypted Data ==> "+encryptData(inputStr, shiftKey));  
  
    System.out.println("Decrypted Data ==> "+decryptData(encryptData(inputStr,  
shiftKey), shiftKey));  
  
    // close Scanner class object  
  
    sc.close();  
}  
}
```

**OUTPUT:**

Enter a string for encryption using Caesar Cipher:

MITADT

Enter the value by which each character in the plaintext message gets shifted:

3

Encrypted Data ==> plwdgw

Decrypted Data ==> mitadt

**Conclusion:****References:**

1. William Stallings, "Cryptography and Network Security", Sixth Edition Pearson Education.
2. Atul Kahate, "Cryptography and Network Security", Tata McGraw-Hill.

## Experiment No: 3

**Title:** Write a Program to Implement RSA Algorithm.

**Aim:** To develop a program for encryption and decryption of the message using RSA algorithm in C Language.

### Theory:

An original message is known as the plaintext, while the coded message is called the ciphertext. The process of converting from plaintext to ciphertext is known as enciphering or encryption; restoring the plaintext from the ciphertext is deciphering or decryption.

- **Plaintext:** This is the original intelligible message or data that is fed into the algorithm as input.
- **Encryption algorithm:** The encryption algorithm performs various substitutions and transformations on the plaintext.
- **Secret key:** The secret key is also input to the encryption algorithm. The key is a value independent of the plaintext and of the algorithm. The algorithm will produce a different output depending on the specific key being used at the time. The exact substitutions and transformations performed by the algorithm depend on the key.
- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different ciphertexts. The ciphertext is an apparently random stream of data and, as it stands, is unintelligible.
- **Decryption algorithm:** This is essentially the encryption algorithm run in reverse. It takes the ciphertext and the secret key and produces the original plaintext.

### **RSA Algorithm:**

RSA algorithms have been proposed for public-key cryptography. RSA was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and first published in 1978. Public-key cryptography, also known as asymmetric cryptography, uses two different but mathematically linked keys, one public and one private. The public key can be shared with everyone, whereas the private key must be kept secret. In RSA cryptography, both the public and the private keys can encrypt a message; the opposite key from the one used to encrypt a message is used to decrypt it. This attribute is one reason why RSA has become the most widely used asymmetric.

RSA is more secure because of factoring large integers that are the product of two large prime numbers. Multiplying these two numbers is easy, but determining the original prime numbers from the total factoring is considered infeasible due to the time it would take even using today's super computers.

The RSA scheme is a block cipher in which the plaintext and ciphertext are integers between 0 and  $n - 1$  for some  $n$ . A typical size for  $n$  is 1024 bits. That is,  $n$  is less than  $2^{1024}$ . In RSA naming convention are Plaintext block  $M$  and ciphertext block  $C$ .

Some terms related to RSA are as below

$p, q$ , two prime numbers

$$n = p * q$$

$e$ , with  $\gcd(\phi(n), e) = 1$ ;  $1 < e < \phi(n)$

$$de = 1 \pmod{\phi(n)}$$

The private key consists of  $\{d, n\}$  and the public key consists of  $\{e, n\}$ . Suppose that user A has published its public key and that user B wishes to send the message  $M$  to A. Then B calculates  $C = M^e \pmod{n}$  and transmits  $C$ . On receipt

of this ciphertext, user A decrypts by calculating  $M = C^d \bmod n$ .

**Algorithm:**

1. Select two prime numbers,  $p = 17$  and  $q = 11$ .
2. Calculate  $n = pq = 17 \times 11 = 187$ .
3. Calculate  $\phi(n) = (p - 1)(q - 1) = 16 \times 10 = 160$ .
4. Select  $e$  such that  $e$  is relatively prime to  $\phi(n) = 160$  and less than  $\phi(n)$ ; we choose  $e = 7$ .
5. Determine  $d$  such that  $de = 1 \pmod{n}$  and  $d < 160$ . The correct value is  $d = 23$ , because  $23 \times 7 = 161 = (1 \times 160) + 1$ ;

The resulting keys are public key  $PU = \{7, 187\}$  and private key  $PR = \{23, 187\}$ .

The example shows the use of these keys for a plaintext input of  $M = 88$ . For encryption, we need to calculate  $C = 88^7 \bmod 187$ . we can evaluate as follows.

$$88^7 \bmod 187 = [(88^4 \bmod 187) \times (88^2 \bmod 187) \times (88^1 \bmod 187)] \bmod 187$$

$$88^1 \bmod 187 = 88$$

$$88^2 \bmod 187 = 7744 \bmod 187 = 77$$

$$88^4 \bmod 187 = 59,969,536 \bmod 187 = 132$$

$$88^7 \bmod 187 = (88 \times 77 \times 132) \bmod 187 = 894,432 \bmod 187 = 11$$

For decryption, we calculate  $M = 11^{23} \bmod 187$ :

$$11^{23} \bmod 187 = [(11^1 \bmod 187) * (11^2 \bmod 187) \times (11^4 \bmod 187) * (11^8 \bmod 187) * (11^8 \bmod 187)] \bmod 187$$

$$11^1 \bmod 187 = 11$$

$$11^2 \bmod 187 = 121$$

$$11^4 \bmod 187 = 14,641 \bmod 187 = 55$$

$$11^8 \bmod 187 = 214,358,881 \bmod 187 = 33$$

$$11^{23} \bmod 187 = (11 \times 121 \times 55 \times 33 \times 33) \bmod 187 = 79,720,245 \bmod 187 = 88$$



## Program:

```
/* Aim: Program for encryption and decryption of message */
/* Title: Program for encryption and decryption using RSA algorithm */
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<math.h>

void main()
{
    unsigned long p,q,n,k,e,d;
    unsigned long cipher,plain;
    unsigned long C,M;
    char message[80];
    char encrypt[80],decrypt[80];
    unsigned int len,i,j;
    int t;
    clrscr();
    printf("\nEnter P and Q : "); // P & Q are prime number
    scanf("%ld%ld",&p,&q);
    n = p * q;
    k = (p-1) * (q-1);
    printf("\nEnter the value of public key : ");
    scanf("%ld",&e); //e relative prime to k and 1 < e < k
    d = 3;
    while(1)
    {
        if ((d * e) == (1+k)) // de= 1(mod k)
```

```

        break;
    d++;
}
printf("\nPublic Key = %ld, Private key = %ld",e,d);

printf("\nEnter the message : "); // take message (plain text as input)
fflush(stdin);
gets(message);
len = strlen(message);
for(i=0;i<len;i++)
{
    M = message[i];
    C = 1;
    for(j=0;j<e;j++)
    {
        C= C * M % n; // calculate the vlaue of cipher
        C = C % n;
    }
    encrypt[i] = C;
}

printf("\nEncrypted Message is : ");
for(i=0;i<len;i++)
    printf("%c",encrypt[i]);
printf("\n");
for(i=0;i<len;i++)
{
    M = 1;
    C = encrypt[i];
    if(C>127)        // char whose value is more than 127

```

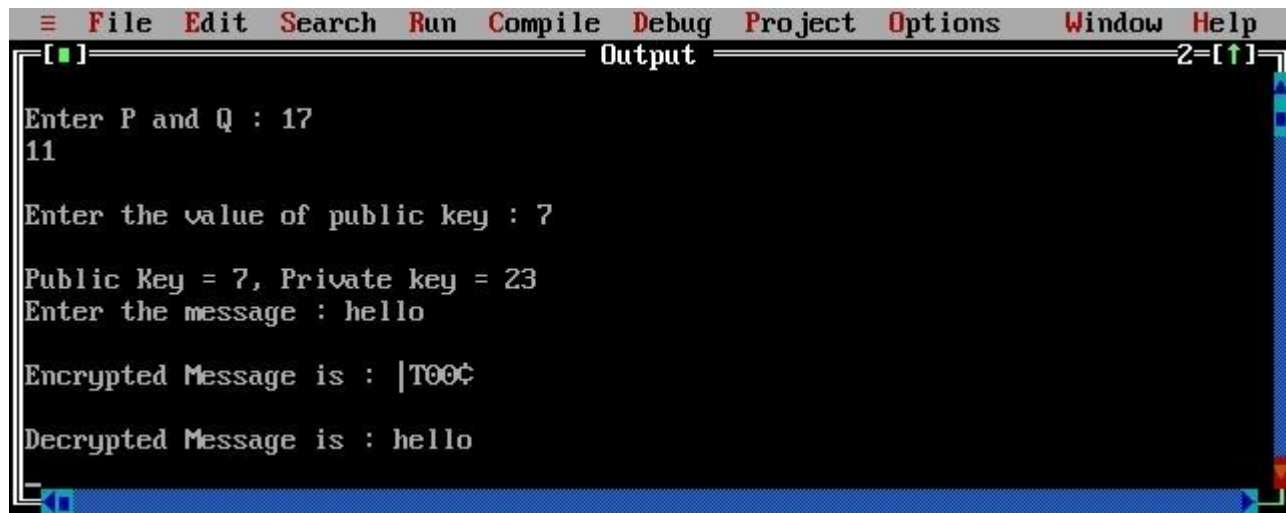
```

        C = 256 + C; // more than 127 MSB bit set to 1
    for(j=0;j<d;j++)
    {
        M = M * C % n; // Calculate plain text messgae
        M = M % n;
    }
    decrypt[i] = M;
}
printf("\nDecrypted Message is : ");
for(i=0;i<len;i++)
    printf("%c",decrypt[i]);
printf("\n");

getch();
}

```

### Output:



```

File Edit Search Run Compile Debug Project Options Window Help
Output
Enter P and Q : 17
11
Enter the value of public key : 7
Public Key = 7, Private key = 23
Enter the message : hello
Encrypted Message is : |T00¢
Decrypted Message is : hello

```

## **Conclusion:**

## **References:**

1. William Stallings, "Cryptography and Network and Network security-Principals and practices", Pearson Education.
2. Atul Kahate, "Cryptography and Network Security", Tata McGraw-Hill.

## Experiment No: 4

**Title:** Write a Program to Simulate Diffie-Hellman Key Exchange.

**Aim:** To develop a Program for Diffie-Hellman Key Exchange Algorithm using C Language.

### Theory:

Diffie-Hellman was the first public-key algorithm ever invented, way back in 1976. It gets its security from the difficulty of calculating discrete logarithms in a finite field, as compared with the ease of calculating exponentiation in the same field. Diffie-Hellman can be used for key distribution—Alice and Bob can use this algorithm to generate a secret key—but it cannot be used to encrypt and decrypt messages. The math is simple. First, Alice and Bob agree on a large prime,  $n$  and  $g$ , such that  $g$  is primitive mod  $n$ . These two integers don't have to be secret; Alice and Bob can agree to them over some insecure channel. They can even be common among a group of users. It doesn't matter.

Then, the protocol goes as follows:

(1) Alice chooses a random large integer  $x$  and sends Bob

$$X = g^x \bmod n$$

(2) Bob chooses a random large integer  $y$  and sends Alice

$$Y = g^y \bmod n$$

(3) Alice computes

$$k = Y^x \bmod n$$

(4) Bob computes

$$k' = X^y \bmod n$$

Both  $k$  and  $k'$  are equal to  $g^{xy} \bmod n$ . No one listening on the channel can compute that value; they only know  $n$ ,  $g$ ,  $X$ , and  $Y$ . unless they can compute the discrete

logarithm and recover  $x$  or  $y$ , they do not solve the problem. So,  $k$  is the secret key that both Alice and Bob computed independently. The choice of  $g$  and  $n$  can have a substantial impact on the security of this system. The number  $(n - 1)/2$  should also be a prime. And most important,  $n$  should be large: The security of the system is based on the difficulty of factoring numbers the same size as  $n$ . You can choose any  $g$ , such that  $g$  is primitive mod  $n$ ; there's no reason not to choose the smallest  $g$  you can—generally a one-digit number. (And actually,  $g$  does not have to be primitive; it just has to generate a large subgroup of the multiplicative group mod  $n$ .)

## DIFFIE-HELLMAN WITH THREE OR MORE PARTIES

The Diffie-Hellman key-exchange protocol can easily be extended to work with three or more people. In this example, Alice, Bob, and Carol together generate a secret key.

- (1) Alice chooses a random large integer  $x$  and sends Bob

$$X = g^x \bmod n$$

- (2) Bob chooses a random large integer  $y$  and sends Carol

$$Y = g^y \bmod n$$

- (3) Carol chooses a random large integer  $z$  and sends Alice

$$Z = g^z \bmod n$$

- (4) Alice sends Bob

$$Z' = Z^x \bmod n$$

- (5) Bob sends  
Carol

$$X' = X^y \bmod n$$

- (6) Carol sends Alice

$$Y' = Y^z \bmod n$$

- (7) Alice computes

$$k = Y^x \bmod n$$

(8) Bob computes

$$k = Z^y \bmod n$$

(9) Carol computes

$$k = X^z \bmod n$$

The secret key,  $k$ , is equal to  $g^{xyz} \bmod n$ , and no one else listening in on the communications can compute that value. The protocol can be easily extended to four or more people; just add more people and more rounds of computation.

### Program:

```
#include<stdio.h>

#include<math.h>

int alice(int,int,int);

int bob(int,int,int);


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

    printf("\nEnter the value of n & g:");

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

    printf("\n Enter the value of x & y:");

    scanf("%d %d",&x,&y);

    a=alice(n,g,x);

    printf("\nAlice generate end value is %d:",a);
```

```
b=bob(n,g,y);  
printf("\nBob generate end value is %d:",b);
```

```
k1=alice(n,b,x);  
printf("\nValue of k1 is %d :",k1);
```

```
k2=bob(n,a,y);  
printf("\nValue of k2 is %d:",k2);
```

```
if(k1==k2)  
{  
    printf("\nMessage is shared secretely between Alice and Bob");  
}  
else  
{  
    printf("\nMessage is not shared");  
}  
return 0;  
}
```

```
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;
    b1=pow(g,y);
    b=b1%n;
    return(b);
}

```

### **Output:**

Enter the value of n & g:13 6

Enter the value of x & y:3 10

Alice generate end value is 8:

Bob generate end value is 4:

Value of k1 is 12 :

Value of k2 is 12:

Message is shared secretly between Alice and Bob

**Conclusion:****References:**

1. William Stallings, "Cryptography and Network and Network security Principals and practices", Pearson Education
2. Bruce Schneider, "Applied cryptography: Protocols, Algorithms and sources code in C", Second edition, Willey, 2008.

## Experiment No: 5

**Title:** Write a Program to Implement Columnar Cipher Text

**Aim:** Implementation of Columnar Transposition Technique.

### Theory:

A kind of mapping is achieved by performing some sort of permutation on the plaintext letters. This technique is referred to as a transposition cipher.

A complex scheme is to write the message in a rectangle, row by row, and read the message off, column by column, but permute the order of the columns. The order of the columns then becomes the key to the algorithm. For example

Key:           4 3 1 2 5 6 7  
Plaintext:     a t t a c k p  
                o s t p o n e  
                d u n t i l t  
                w o a m x y z

Ciphertext: TTNAAPTMTSUOAODWCOIXKNLYPETZ

Thus, in this example, the key is 4312567. To encrypt, start with the column that is labeled 1, in this case column 3. Write down all the letters in that column. Proceed to column 4, which is labeled 2, then column 2, then column 1, then columns 5, 6, and 7.

A pure transposition cipher is easily recognized because it has the same letter frequencies as the original plaintext. For the type of columnar transposition just shown, cryptanalysis is fairly straightforward and involves laying out the ciphertext in a matrix and playing around with column positions.

The transposition cipher can be made significantly more secure by performing more than one stage of transposition. The result is a more complex permutation that is not easily reconstructed. Thus, if the foregoing message is reencrypted using the same algorithm,

Key: 4 3 1 2 5 6 7

Input: t t n a a p t

m t s u o a o

d w c o i x k

n l y p e t z

Output: NSCYAUOPTTWLTMDNAOIEPAXTTOKZ

To visualize the result of this double transposition, designate the letters in the original plaintext message by the numbers designating their position. Thus, with 28 letters in the message, the original sequence of letters is

01 02 03 04 05 06 07 08 09 10 11 12 13 14

15 16 17 18 19 20 21 22 23 24 25 26 27 28

After the first transposition, we have

03 10 17 24 04 11 18 25 02 09 16 23 01 08

15 22 05 12 19 26 06 13 20 27 07 14 21 28

which has a somewhat regular structure. But after the second

transposition, we have 17 09 05 27 24 16 12 07 10 02 22

20 03 25

15 13 04 23 19 14 11 01 26 21 18 08 06 28

This is a much less structured permutation and is much more difficult to cryptanalyze.

**Program:**

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

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

```
void cipher(int i, int c);
```

```
int findMin();
```

```
void makeArray(int, int);
```

```
char arr[22][22], darr[22][22], emessage[111], retmessage[111], key[55];
```

```
char temp[55], temp2[55];
```

```
int k = 0;
```

```
int main() {
```

```
    char *message, *dmessage;
```

```
    int i, j, klen, emlen, flag = 0;
```

```
    int r, c, index, min, rows;
```

```
    printf("Enetr the key\n");
```

```
    fflush(stdin);
```

```
    gets(key);
```

```
    printf("\nEnter message to be ciphered\n");
```

```
    fflush(stdin);
```

```
    gets(message);
```

```
    strcpy(temp, key);
```

```
    klen = strlen(key);
```

```
    k = 0;
```

```

for (i = 0;; i++) {
    if (flag == 1)
        break;

    for (j = 0; key[j] != NULL; j++) {
        if (message[k] == NULL) {
            flag = 1;
            arr[i][j] = '-';
        } else {
            arr[i][j] = message[k++];
        }
    }
}
r = i;
c = j;

```

```

for (i = 0; i < r; i++) {
    for (j = 0; j < c; j++) {
        printf("%c ", arr[i][j]);
    }
    printf("\n");
}

```

```

k = 0;

```

```

for (i = 0; i < klen; i++) {
    index = findMin();
    cipher(index, r);
}

```

```

emessage[k] = '\0';
printf("\nEncrypted message is\n");
for (i = 0; emessage[i] != NULL; i++)
    printf("%c", emessage[i]);

printf("\n\n");
//deciphering

emlen = strlen(emessage);
//emlen is length of encrypted message

strcpy(temp, key);

rows = emlen / klen;
//rows is no of row of the array to made from ciphered message
rows;
j = 0;

for (i = 0, k = 1; emessage[i] != NULL; i++, k++) {
    //printf("\nEmlen=%d",emlen);
    temp2[j++] = emessage[i];
    if ((k % rows) == 0) {
        temp2[j] = '\0';
        index = findMin();
        makeArray(index, rows);
        j = 0;
    }
}

printf("\nArray Retrieved is\n");

```

```

k = 0;
for (i = 0; i < r; i++) {
    for (j = 0; j < c; j++) {
        printf("%c ", darr[i][j]);
        //retrieving message
        retmessage[k++] = darr[i][j];

    }
    printf("\n");
}
retmessage[k] = '\0';

printf("\nMessage retrieved is\n");

for (i = 0; retmessage[i] != NULL; i++)
    printf("%c", retmessage[i]);

getch();
return (0);
}

void cipher(int i, int r) {
    int j;
    for (j = 0; j < r; j++) {
        {
            emessage[k++] = arr[j][i];
        }
    }
}
// emessage[k]='\0';

```



```
}
```

```
void makeArray(int col, int row) {
```

```
    int i, j;
```

```
    for (i = 0; i < row; i++) {
```

```
        darr[i][col] = temp2[i];
```

```
    }
```

```
}
```

```
int findMin() {
```

```
    int i, j, min, index;
```

```
    min = temp[0];
```

```
    index = 0;
```

```
    for (j = 0; temp[j] != NULL; j++) {
```

```
        if (temp[j] < min) {
```

```
            min = temp[j];
```

```
            index = j;
```

```
        }
```

```
    }
```

```
    temp[index] = 123;
```

```
    return (index);
```

```
}
```

### **Output:**

Enter the key

hello

Enter the message to be ciphered

how are you

h o w a

r e y o

u - - - -

Encrypted message is

oe-hruw - y-ao-

Array Retrieved is

h o w a

r e y o

u - - - -

Message retrieved is

how are you----

## CONCLUSION

### References:

1. William Stalling, "Cryptography and Network and Network security-Principals and practices", Pearson Education
2. Atul Kahate, "Cryptography and Network Security", Tata McGraw-Hill.