Lab 1: Write a program to implement Caesar Cipher.

- **Theory:**

The Caesar Cipher technique is 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.

- **Program Code:**

```
#include <stdio.h>
#include <string.h>
void encrypt(char *plaintext, int key)
{
   int i;
   char ch;
   for (i = 0; plaintext[i] != '\0'; ++i)
   {
      ch = plaintext[i];
      if (ch >= 'a' && ch <= 'z')
      {
         ch = 'a' + (ch - 'a' + key) % 26;
      }
      else if (ch >= 'A' && ch <= 'Z')
      {
         ch = 'A' + (ch - 'A' + key) % 26;
      }
      plaintext[i] = ch;
   }
}

void decrypt(char *ciphertext, int key)
{
   int i;
   char ch;
   for (i = 0; ciphertext[i] != '\0'; ++i)
```

```
    {
        ch = ciphertext[i];
        if (ch >= 'a' && ch <= 'z')
        {
            ch = 'a' + (ch - 'a' - key + 26) % 26;
        }
        else if (ch >= 'A' && ch <= 'Z')
        {
            ch = 'A' + (ch - 'A' - key + 26) % 26;
        }
        ciphertext[i] = ch;
    }
}
int main()
{
    char plaintext[100];
    int key;
    printf("Enter plaintext: \n");
    fgets(plaintext, sizeof(plaintext), stdin);
    printf("Enter the key (a number between 0 and 25): ");
    scanf("%d", &key);
    encrypt(plaintext, key);
    printf("Encrypted text: %s\n", plaintext);
    decrypt(plaintext, key);
    printf("Decrypted text: %s\n", plaintext);
    return 0;
}
```

- **Output:**

  Run 1-

Enter plaintext:

kul

Enter the key (a number between 0 and 25): 3

Encrypted text: nxo

Decrypted text: kul

Lab 2: Write a program to implement Rail Fence Cipher.

- **Theory:**

The Rail Fence cipher is a type of transposition cipher where the plaintext is written in a zigzag pattern (like a fence) and then read out row by row to create the ciphertext. Decryption involves reversing this process to obtain the original plaintext.

- **Program Code:**

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
void encryptRailFence(char *plaintext, int key) {
    int len = strlen(plaintext);
    char rail[key][len];
    for (int i = 0; i < key; ++i)
        for (int j = 0; j < len; ++j)
            rail[i][j] = '\0';
    int row = 0, dir = 0;
    for (int i = 0; i < len; ++i) {
        if (row == 0 || row == key - 1)
            dir = !dir;
        rail[row][i] = plaintext[i];
        dir ? ++row : --row;
    }
    printf("Encrypted text: \n");
    for (int i = 0; i < key; ++i)
```

```
        for (int j = 0; j < len; ++j)
            if (rail[i][j] != '\0')
                printf("%c", rail[i][j]);
}
void decryptRailFence(char *ciphertext, int key) {
    int len = strlen(ciphertext);
    char rail[key][len];
    for (int i = 0; i < key; ++i)
        for (int j = 0; j < len; ++j)
            rail[i][j] = '\0';
    int row = 0, dir = 0;
    for (int i = 0; i < len; ++i) {
        if (row == 0)
            dir = 1;
        else if (row == key - 1)
            dir = 0;
        rail[row][i] = '*'; //
        dir ? ++row : --row;
    }
    int index = 0;
    for (int i = 0; i < key; ++i)
        for (int j = 0; j < len; ++j)
            if (rail[i][j] == '*' && index < len)
                rail[i][j] = ciphertext[index++];
    row = 0;
    dir = 0;
    printf("\nDecrypted text: \n");
    for (int i = 0; i < len; ++i) {
        if (row == 0 || row == key - 1)
            dir = !dir;
        printf("%c", rail[row][i]);
        dir ? ++row : --row;
    }
}
int main() {
    char text[100];
```

```c
    int key;
    printf("Enter plaintext: ");
    fgets(text, sizeof(text), stdin);
    printf("Enter the key (number of rails): ");
    scanf("%d", &key);
    text[strcspn(text, "\n")] = '\0';
    encryptRailFence(text, key);
    decryptRailFence(text, key);
    return 0;
}
```

- **Output:**

Run 1-

Enter plaintext: kusal

Enter the key (number of rails): 3

Encrypted text:

Kslau

Decrypted text:

kluas

Lab 3: Write a program to implement Playfair Cipher.

- **Theory:**

Playfair cipher program in c is a manual symmetrical encryption technique that is used to encrypt or encode a message. As this technique uses the same key for encryption and decryption, so this technique falls under the category of symmetrical encryption technique. It was the first literal digraph substitution cipher

- **Program 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;
    }
}
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;
}
void generateKeyTable(char key[], int ks, char keyT[5][5])
{
    int i, j, k, flag = 0, *dicty;
    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;
            }
        }
    }
}
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;
            }
        }
    }
}
int mod5(int a) { return (a % 5); }
int prepare(char str[], int ptrs)
{
    if (ptrs % 2 != 0) {
        str[ptrs++] = 'z';
        str[ptrs] = '\0';
    }
    return ptrs;
}
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]];
        }
    }
}
void encryptByPlayfairCipher(char str[], char key[])
{
    char ps, ks, keyT[5][5];
```

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

    ps = strlen(str);
    toLowerCase(str, ps);
    ps = removeSpaces(str, ps);
    ps = prepare(str, ps);
    generateKeyTable(key, ks, keyT);
    encrypt(str, keyT, ps);
}
int main()
{
    char str[SIZE], key[SIZE];
    printf("Enter the key:");
    scanf("%s",&key);
    printf("Key text: %s\n", key);
    printf("Enter the plain text to encrypt:");
    scanf("%s",&str);
    printf("Plain text: %s\n", str);
    encryptByPlayfairCipher(str, key);
    printf("Cipher text: %s\n", str);
    return 0;
}
```

- **Output:**

Run 1-

Enter the key:CITY

Key text: CITY

Enter the plain text to encrypt:KUSAL

Plain text: KUSAL

Cipher text: HVZGMW

Lab 4: Write a program to implement Vernam Cipher.

- **Theory:**

Vernam Cipher is a method of encrypting alphabetic text. It is one of the Substitution techniques for converting plain text into cipher text. In this mechanism, we assign a number to each character of the Plain-Text, like (a = 0, b = 1, c = 2, … z = 25). In the Vernam cipher algorithm, we take a key to encrypt the plain text whose length should be equal to the length of the plain text.

- **Program Code:**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void vernamEncrypt(char *plaintext, char *key, char
*ciphertext) {
    int length = strlen(plaintext);
    for (int i = 0; i < length; i++) {
        ciphertext[i] = plaintext[i] ^ key[i];
    }
    ciphertext[length] = '\0';
}
void vernamDecrypt(char *ciphertext, char *key, char
*decryptedtext) {
    int length = strlen(ciphertext);
    for (int i = 0; i < length; i++) {
        decryptedtext[i] = ciphertext[i] ^ key[i];
    }
    decryptedtext[length] = '\0';
}
int main() {
```

```
    char plaintext[100], key[100], ciphertext[100],
decryptedtext[100];
    printf("Enter the plaintext: ");
    fgets(plaintext, sizeof(plaintext), stdin);
    plaintext[strcspn(plaintext, "\n")] = '\0';
    printf("Enter the key (same length as plaintext): ");
    fgets(key, sizeof(key), stdin);
    key[strcspn(key, "\n")] = '\0';
    vernamEncrypt(plaintext, key, ciphertext);
    printf("\nEncrypted Text: %s\n", ciphertext);

    vernamDecrypt(ciphertext, key, decryptedtext);
    printf("Decrypted Text: %s\n", decryptedtext);
    return 0;
}
```

Output:

Enter the plaintext: KUSAL

Enter the key (same length as plaintext): THAPA

Encrypted Text: DBSPL

Decrypted Text: KUSAL

Lab 5: Write a program to implement Euclidean Algorithm.

- **Theory:**

The Euclidean algorithm is a way to find the greatest common divisor of two positive integers. GCD of two numbers is the largest number that divides both of them. A simple way to find GCD is to factorize both numbers and multiply common prime factors.

- **Program Code:**

```c
#include <stdio.h>
void main() {
    int m, n;
    printf("Enter-two integer numbers: ");
    scanf ("%d %d", &m, &n);
    while (n > 0) {
        int r = m % n;
        m = n;
        n = r;
    }
    printf ("GCD = %d \n",m);
}
```

- **Output:**

Run 1-

Enter-two integer numbers: 5 15

GCD = 5

Run 2-

Enter-two integer numbers: 5 13

GCD = 1

Lab 6: Write a program to implement Extended Euclidean Algorithm.

- **Theory:**

Extended Euclidean algorithm also finds integer coefficients x and y such that: ax + by = gcd(a, b).The extended Euclidean algorithm updates the results of gcd(a, b) using the results calculated by the recursive call gcd(b%a, a).

- **Program Code:**

```c
#include <stdio.h>
int gcdExtended(int a, int b, int *x, int *y)
{
    if (a == 0)
    {
        *x = 0;
        *y = 1;
        return b;
    }
    int x1, y1;
    int gcd = gcdExtended(b%a, a, &x1, &y1);
    *x = y1 - (b/a) * x1;
    *y = x1;
    return gcd;
}
int main()
{
    int x, y,a,b;
    printf("Enter two integers:");
    scanf("%d %d",&a, &b);
    int g = gcdExtended(a, b, &x, &y);
    printf("gcd(%d, %d) = %d", a, b, g);
    return 0;
}
```

- **Output:**

Run 1-

Enter two integers:5 7

gcd(5, 7) = 1

Lab 7: Write a program to implement Extended Euclidean Algorithm to find multiplicative inverse of a number under modulo p.

- **Theory:**

Given two integers A and M, find the modular multiplicative inverse of A under modulo M. The modular multiplicative inverse is an integer X such that:

$$A\,X \cong 1 \pmod{M}$$

The value of X should be in the range {1, 2, ... M-1}, i.e., in the range of integer modulo M. (X cannot be 0 as A*0 mod M will never be 1). The multiplicative inverse of "A modulo M" exists if and only if A and M are relatively prime (i.e. if gcd(A, M) = 1).

- **Program Code:**

```c
#include <stdio.h>
int extendedEuclidean(int a, int b, int *x, int *y) {
    if (a == 0) {
        *x = 0;
        *y = 1;
        return b;
    }
    int x1, y1;
    int gcd = extendedEuclidean(b % a, a, &x1, &y1);
    *x = y1 - (b / a) * x1;
    *y = x1;
    return gcd;
}
int modInverse(int a, int p) {
    int x, y;
    int gcd = extendedEuclidean(a, p, &x, &y);
    if (gcd != 1) {
        printf("Inverse doesn't exist\n");
        return -1;
    } else {
        int result = (x % p + p) % p;
        return result;
    }
```

```
}
int main() {
    int a, p;
    printf("Enter a and p (a mod p): ");
    scanf("%d %d", &a, &p);
    int inverse = modInverse(a, p);
    if (inverse != -1) {
        printf("Multiplicative inverse of %d under modulo %d
is: %d\n", a, p, inverse);
    }
    return 0;
}
```

Output:

Run 1-

Enter a and p (a mod p): 7 26

Multiplicative inverse of 7 under modulo 26 is: 15

Lab 8: Write a program to find primitive root of a prime number.

Theory:

Given a prime number n, the task is to find its primitive root under modulo n. The primitive root of a prime number n is an integer r between[1, n-1] such that the values of r^x(mod n) where x is in the range[0, n-2] are different. Return -1 if n is a non-prime number.

Program Code:

```
#include <stdio.h>
#include <stdbool.h>
int power(int x, unsigned int y, int p) {
```

```
    int res = 1;
    x = x % p;
    while (y > 0) {
        if (y & 1)
            res = (res * x) % p;
        y = y >> 1;
        x = (x * x) % p;
    }
    return res;
}
bool isPrimitiveRoot(int n, int prime) {
    bool visited[prime];
    for (int i = 0; i < prime; i++) {
        visited[i] = false;
    }

    for (int i = 1; i < prime; i++) {
        int val = power(n, i, prime);
        if (visited[val])
            return false;
        visited[val] = true;
    }
    return true;
}
int findPrimitiveRoot(int prime) {
    for (int i = 2; i < prime; i++) {
        if (isPrimitiveRoot(i, prime))
            return i;
    }
    return -1;
}
int main() {
    int p;
    printf("Enter a prime number: ");
    scanf("%d", &p);
    if (p <= 1) {
```

```
        printf("Invalid input: Enter a prime number greater
than 1.\n");
        return 1;
    }
    int primitiveRoot = findPrimitiveRoot(p);
    if (primitiveRoot == -1) {
        printf("Primitive root does not exist for %d.\n", p);
    } else {
        printf("The primitive root of %d is: %d\n", p,
primitiveRoot);
    }
    return 0;
}
```

Output:

Enter a prime number: 23

The primitive root of 23 is: 5

Lab 9: Write a program to implement Diffie hellman key exchange algorithm.

- **Theory:**

The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secret communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

- **Program Code:**

```
#include <math.h>
#include <stdio.h>
long long int power(long long int a, long long int b,
                    long long int P)
```

```
{
    if (b == 1)
        return a;

    else
        return (((long long int)pow(a, b)) % P);
}
int main()
{
    long long int P, G, x, a, y, b, ka, kb;
    printf("Enter the value of p and g:");
    scanf("%lld %lld",&P,&G);
    printf("The value of P : %lld\n", P);
    printf("The value of G : %lld\n\n", G);
    printf("Enter the private key for alice:");
    scanf("%lld",&a);
    printf("The private key a for Alice : %lld\n", a);
    x = power(G, a, P); // gets the generated key
    printf("Enter the private key for Bob:");
    scanf("%lld",&b);
    printf("The private key b for Bob : %lld\n\n", b);
    y = power(G, b, P); // gets the generated key
    ka = power(y, a, P); // Secret key for Alice
    kb = power(x, b, P); // Secret key for Bob
    printf("Secret key for the Alice is : %lld\n", ka);
    printf("Secret Key for the Bob is : %lld\n", kb);
    return 0;
}
```

- **Output:**

Enter the value of p and g:23 9

The value of P : 23

The value of G : 9

Enter the private key for alice:3

The private key a for Alice : 3

Enter the private key for Bob:4

The private key b for Bob : 4

Secret key for the Alice is : 9

Secret Key for the Bob is : 9

Lab 10: Write a program to implement RSA algorithm.

- **Theory:**

RSA algorithm is an asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. Public Key and Private Key. As the name describes that the Public Key is given to everyone and the Private key is kept private.

An example of asymmetric cryptography:

A client (for example browser) sends its public key to the server and requests some data.

The server encrypts the data using the client's public key and sends the encrypted data.

The client receives this data and decrypts it.

- **Program Code:**

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>
long int
p,q,n,t,flag,e[100],d[100],temp[100],j,m[100],en[100],i;
```

```
char msg[100];
int prime(long int);
void ce();
long int cd(long int);
void encrypt();
void decrypt();
int main()
{
    printf("\nENTER FIRST PRIME NUMBER\n");
    scanf("%ld",&p);
    flag=prime(p);
    if(flag==0)
    {
        printf("\nWRONG INPUT\n");
        exit(1);
    }
    printf("\nENTER ANOTHER PRIME NUMBER\n");
    scanf("%ld",&q);
    flag=prime(q);
    if(flag==0||p==q)
    {
        printf("\nWRONG INPUT\n");
        exit(1);
    }
    printf("\nENTER MESSAGE\n");
    fflush(stdin);
    scanf("%s",msg);
    for(i=0;msg[i]!=NULL;i++)
        m[i]=msg[i];
    n=p*q;
    t=(p-1)*(q-1);
    ce();
    printf("\nPOSSIBLE VALUES OF e AND d ARE\n");
    for(i=0;i<j-1;i++)
        printf("\n%ld\t%ld",e[i],d[i]);
    encrypt();
```

```
    decrypt();
    return 0;
}
int prime(long int pr)
{
    int i;
    j=sqrt(pr);
    for(i=2;i<=j;i++)
    {
        if(pr%i==0)
            return 0;
    }
    return 1;
}
void ce()
{
    int k;
    k=0;
    for(i=2;i<t;i++)
        {
            if(t%i==0)
                continue;
                flag=prime(i);
            if(flag==1&&i!=p&&i!=q)
            {
                e[k]=i; flag=cd(e[k]);
                if(flag>0)
        {
            d[k]=flag;
            k++;
        }
        if(k==99)
            break;
        }
    }
}
```

```
long int cd(long int x)
{
    long int k=1;
    while(1)
    {
        k=k+t;
        if(k%x==0)
            return(k/x);
    }
}
void encrypt()
{
    long int pt,ct,key=e[0],k,len;
    i=0;
    len=strlen(msg);
    while(i!=len)
    {
        pt=m[i];
        pt=pt-96;
        k=1;
        for(j=0;j<key;j++)
        {
            k=k*pt;
            k=k%n;
        }
        temp[i]=k;
        ct=k+96;
        en[i]=ct;
        i++;
    }
    en[i]=-1;
    printf("\nTHE ENCRYPTED MESSAGE IS\n");
    for(i=0;en[i]!=-1;i++)
        printf("%c",en[i]);
}
void decrypt()
```

```
{
    long int pt,ct,key=d[0],k;
    i=0;
    while(en[i]!=-1)
    {
        ct=temp[i];
        k=1;
        for(j=0;j<key;j++)
        {
            k=k*ct;
            k=k%n;
        }
        pt=k+96;
        m[i]=pt;
        i++;
    }
    m[i]=-1;
    printf("\nTHE DECRYPTED MESSAGE IS\n");
    for(i=0;m[i]!=-1;i++)
        printf("%c",m[i]);
}
```

- **Output:**

ENTER FIRST PRIME NUMBER

5

ENTER ANOTHER PRIME NUMBER

7

ENTER MESSAGE

KUSAL

POSSIBLE VALUES OF e AND d ARE

11 11

13 13

17 17

THE ENCRYPTED MESSAGE IS

KPEFL

THE DECRYPTED MESSAGE IS

KUSAL

Lab 11: Write a program to implement Elgamal Cryptosystem.

- **Theory:**

ElGamal encryption is a public-key cryptosystem. It uses asymmetric key encryption for communicating between two parties and encrypting the message. This cryptosystem is based on the difficulty of finding discrete logarithm in a cyclic group that is even if we know g_a and g_k, it is extremely difficult to compute g_{ak}.

- **Program Code:**

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
bool isPrime(int n) {
    if (n <= 1) return false;
    if (n <= 3) return true;
    if (n % 2 == 0 || n % 3 == 0) return false;
    for (int i = 5; i * i <= n; i = i + 6)
        if (n % i == 0 || n % (i + 2) == 0) return false;
    return true;
}
unsigned long long modulo(unsigned long long a, unsigned long
long b, unsigned long long c) {
    unsigned long long result = 1;
    a = a % c;
```

```
    while (b > 0) {
        if (b % 2 == 1)
            result = (result * a) % c;
        b = b / 2;
        a = (a * a) % c;
    }
    return result;
}
int gcd(int a, int b) {
    if (b == 0)
        return a;
    return gcd(b, a % b);
}

int findPrimitiveRoot(int p) {
    int phi = p - 1;
    for (int i = 2; i < p; i++) {
        if (gcd(i, p) == 1) {
            bool found = true;
            for (int j = 1; j <= phi; j++) {
                if (modulo(i, j, p) == 1 && j < phi) {
                    found = false;
                    break;
                }
            }
            if (found)
                return i;
        }
    }
    return -1;
}

void generateKeys(int p, int *g, int *x, int *y) {
    *g = findPrimitiveRoot(p);
    *x = rand() % (p - 1) + 1;
    *y = modulo(*g, *x, p);
```

```
}

void encrypt(int p, int g, int y, int msg, int *a, int *b) {
    int k = rand() % (p - 2) + 1;
    *a = modulo(g, k, p);
    *b = (modulo(y, k, p) * msg) % p;
}

int decrypt(int p, int x, int a, int b) {
    int numerator = b % p;
    int denominator = modulo(a, x, p);
    int inv = 1;
    while ((denominator * inv) % p != 1) {
        inv++;
    }
    return (numerator * inv) % p;
}

int main() {
    int p, g, x, y, msg, encryptedMsgA, encryptedMsgB,
decryptedMsg;
    printf("Enter a prime number (p): ");
    scanf("%d", &p);
    if (!isPrime(p)) {
        printf("Not a prime number. Exiting...\n");
        return 1;
    }
    g = findPrimitiveRoot(p);
    generateKeys(p, &g, &x, &y);
    printf("Public Key (p, g, y): (%d, %d, %d)\n", p, g, y);
    printf("Private Key (x): %d\n", x);
    printf("Enter the message to be encrypted (an integer): ");
    scanf("%d", &msg);
    encrypt(p, g, y, msg, &encryptedMsgA, &encryptedMsgB);
    printf("Encrypted Message (a, b): (%d, %d)\n",
encryptedMsgA, encryptedMsgB);
```

```
    decryptedMsg = decrypt(p, x, encryptedMsgA, encryptedMsgB);
    printf("Decrypted Message: %d\n", decryptedMsg);
    return 0;
}
```

- **Output:**

Run 1-

Enter a prime number (p): 7

Public Key (p, g, y): (7, 3, 2)

Private Key (x): 2

Enter the message to be encrypted (an integer): 3

Encrypted Message (a, b): (2, 5)

Decrypted Message: 3