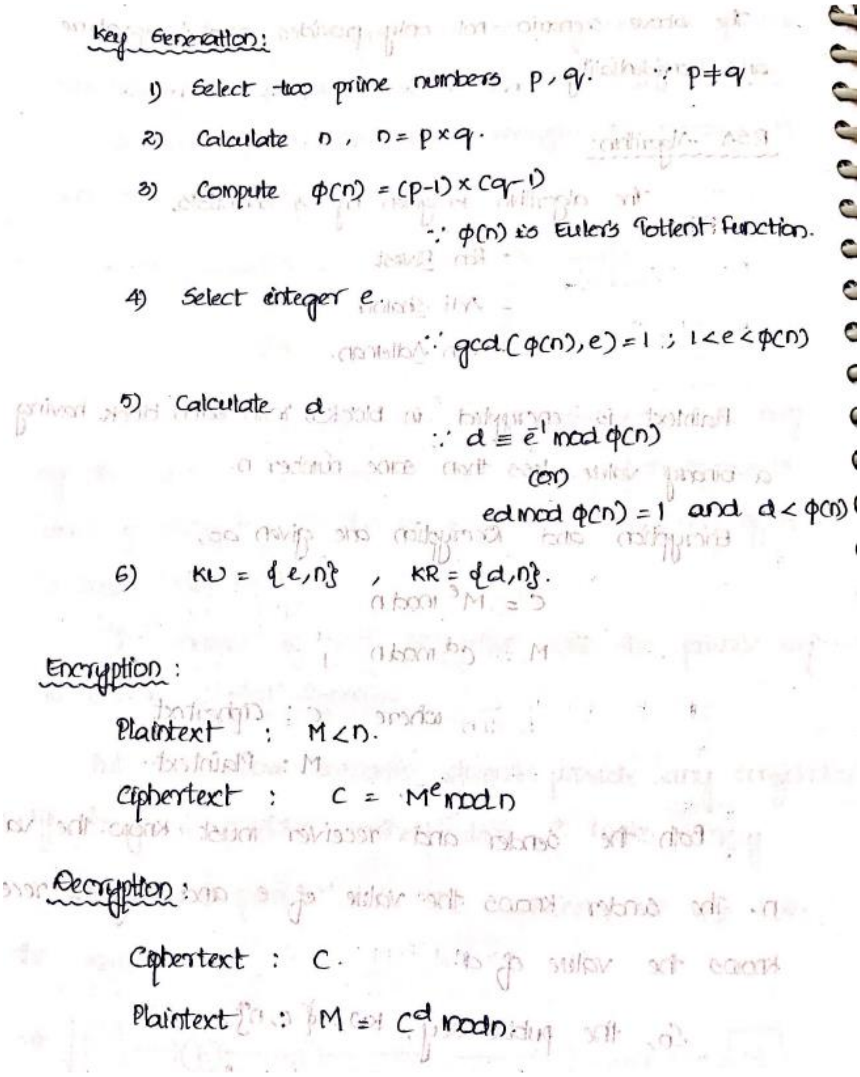
RSA

ALGORITHM



Key Generation:

1) Select two prime numbers p, q. ∵ $p \neq q$

2) Calculate n, $n = p \times q$.

3) Compute $\phi(n) = (p-1) \times (q-1)$

∵ $\phi(n)$ is Euler's Totient Function.

4) Select integer e.

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

5) Calculate d

∵ $d \equiv e^{-1} \bmod \phi(n)$

(or)

$ed \bmod \phi(n) = 1$ and $d < \phi(n)$

6) $KU = \{e, n\}$, $KR = \{d, n\}$.

Encryption:

Plaintext : $M < n$.

Ciphertext : $C = M^e \bmod n$

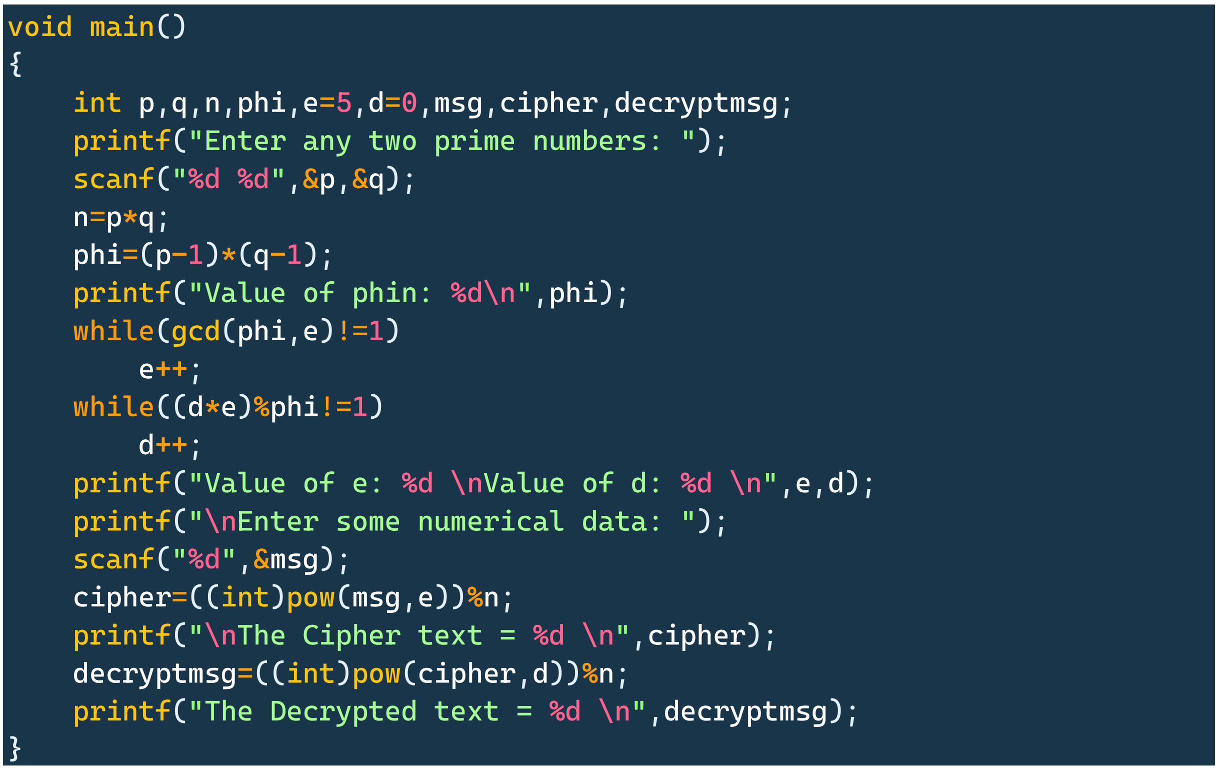
Decryption:

Ciphertext : C.

Plaintext : $M = C^d \bmod n$

PROGRAM

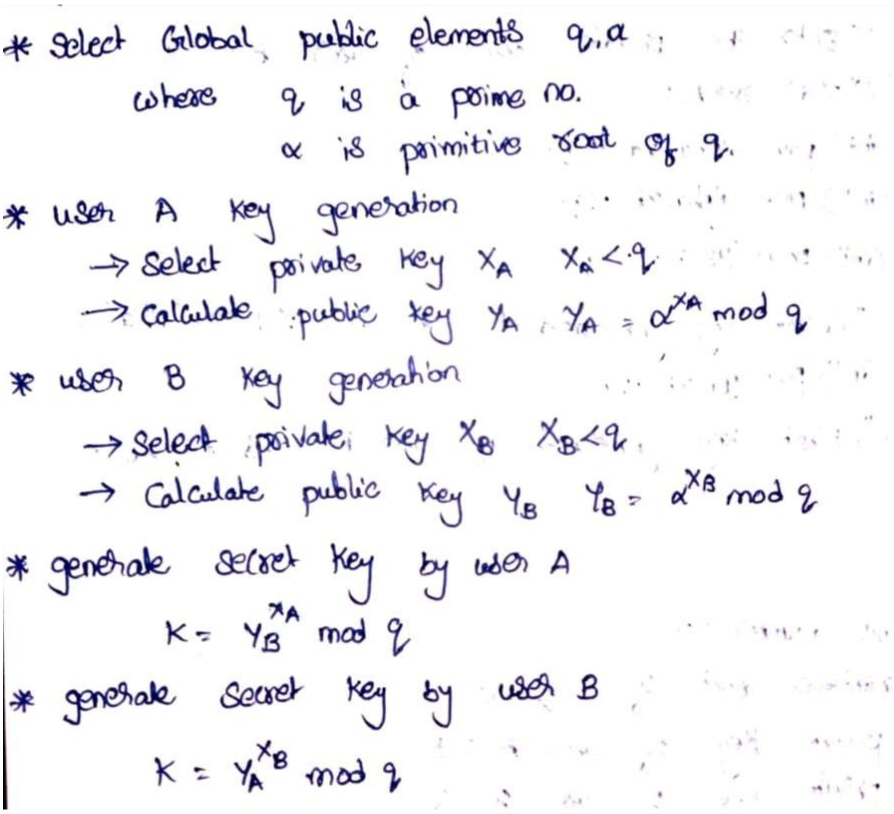
```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int gcd(int a, int b){
    int i,gcd;
    for(i=1;i<=a && i<=b; i++)
    {
        if(a%i==0 && b%i==0)
            gcd=i;
    }
    return gcd;
}
```



```c
void main()
{
    int p,q,n,phi,e=5,d=0,msg,cipher,decryptmsg;
    printf("Enter any two prime numbers: ");
    scanf("%d %d",&p,&q);
    n=p*q;
    phi=(p-1)*(q-1);
    printf("Value of phin: %d\n",phi);
    while(gcd(phi,e)!=1)
        e++;
    while((d*e)%phi!=1)
        d++;
    printf("Value of e: %d \nValue of d: %d \n",e,d);
    printf("\nEnter some numerical data: ");
    scanf("%d",&msg);
    cipher=((int)pow(msg,e))%n;
    printf("\nThe Cipher text = %d \n",cipher);
    decryptmsg=((int)pow(cipher,d))%n;
    printf("The Decrypted text = %d \n",decryptmsg);
}
```

Diffie – Hellman Key Exchange

ALGORITHM



* Select Global public elements q, α

where q is a prime no.

α is primitive root of q.

* user A key generation

→ Select private key X_A $X_A < q$

→ Calculate public key Y_A $Y_A = \alpha^{X_A} \bmod q$

* user B key generation

→ Select private key X_B $X_B < q$

→ Calculate public key Y_B $Y_B = \alpha^{X_B} \bmod q$

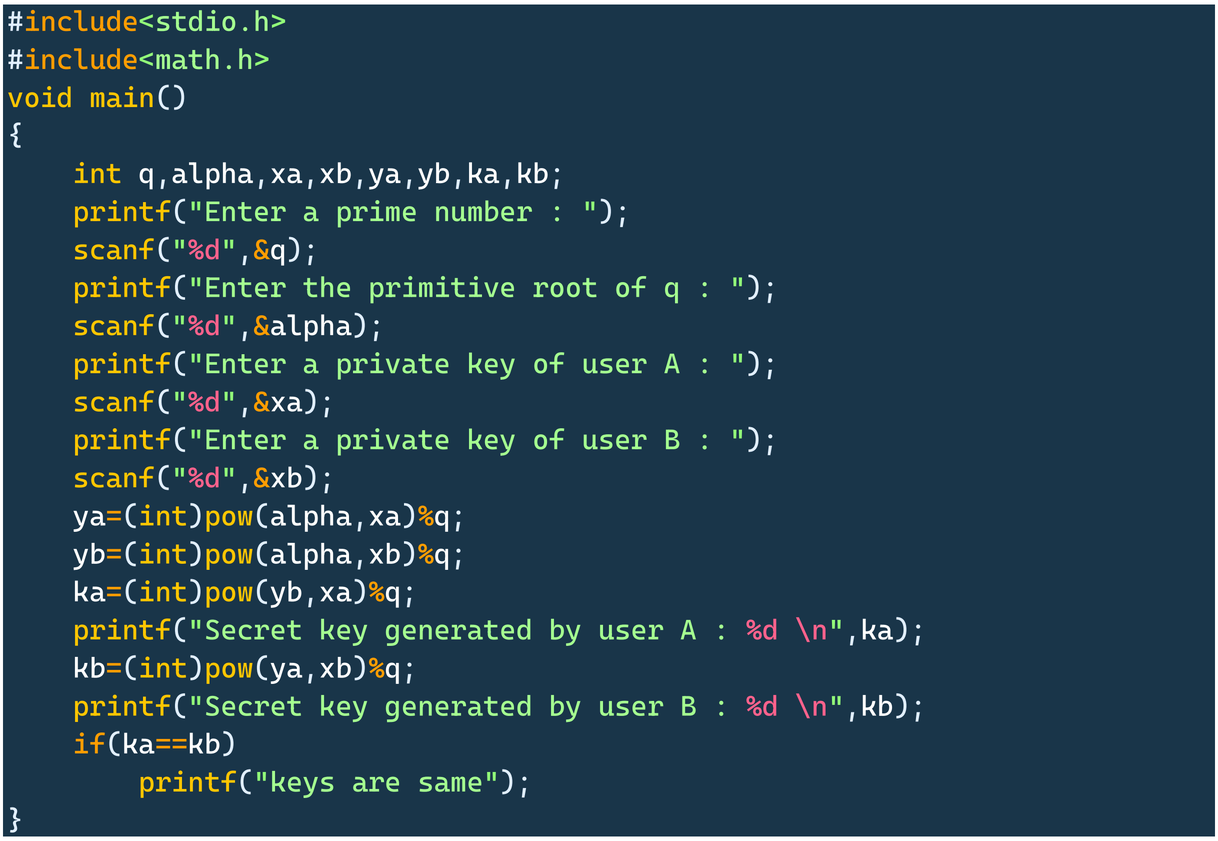
* generate secret key by user A

$K = Y_B^{X_A} \bmod q$

* generate secret key by user B

$K = Y_A^{X_B} \bmod q$

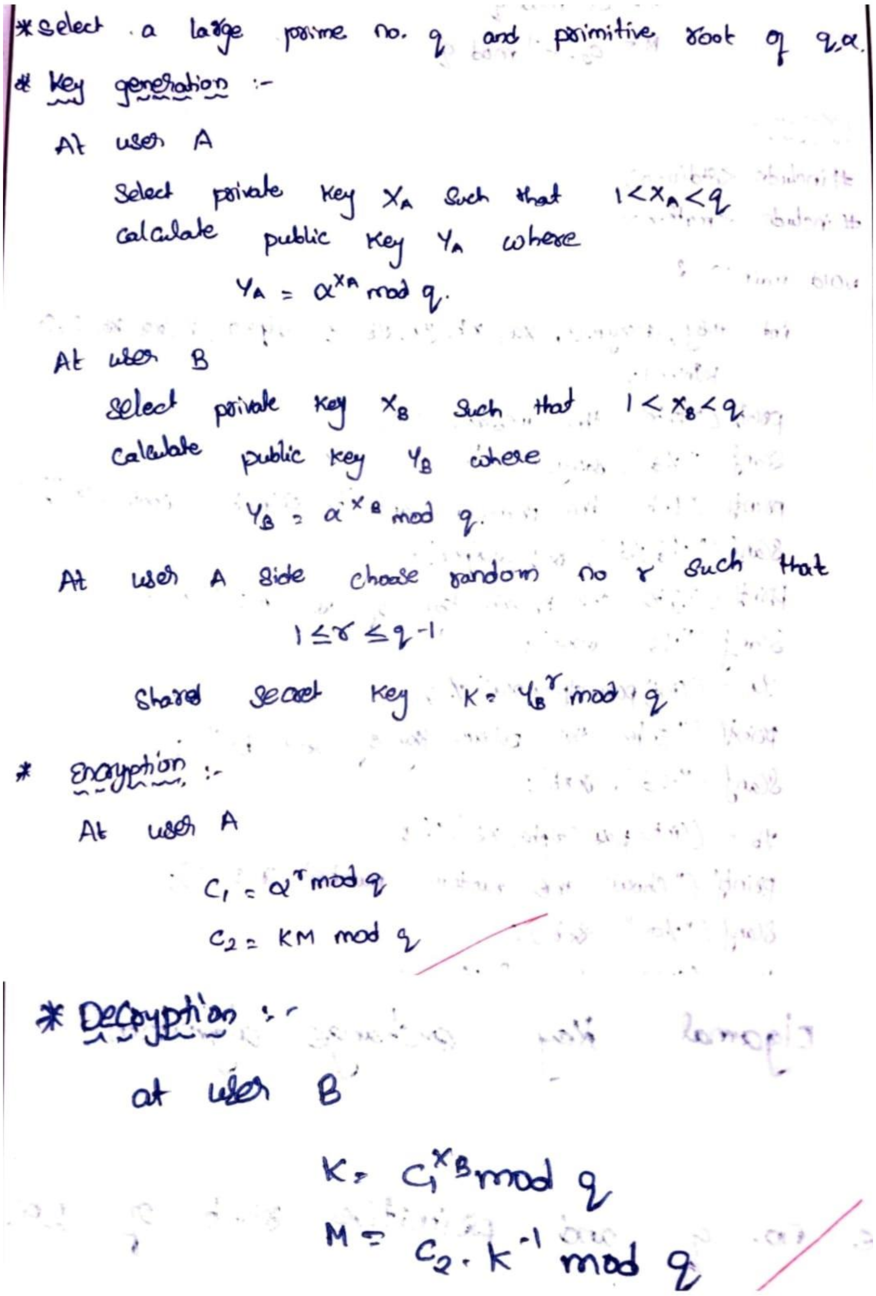
PROGRAM



```
#include<stdio.h>
#include<math.h>
void main()
{
    int q,alpha,xa,xb,ya,yb,ka,kb;
    printf("Enter a prime number : ");
    scanf("%d",&q);
    printf("Enter the primitive root of q : ");
    scanf("%d",&alpha);
    printf("Enter a private key of user A : ");
    scanf("%d",&xa);
    printf("Enter a private key of user B : ");
    scanf("%d",&xb);
    ya=(int)pow(alpha,xa)%q;
    yb=(int)pow(alpha,xb)%q;
    ka=(int)pow(yb,xa)%q;
    printf("Secret key generated by user A : %d \n",ka);
    kb=(int)pow(ya,xb)%q;
    printf("Secret key generated by user B : %d \n",kb);
    if(ka==kb)
        printf("keys are same");
}
```

ElGamal Key Exchange

ALGORITHM



*select a large prime no. q and primitive root of q, α.

* Key generation :-

At user A

Select private key X_A such that $1 < X_A < q$

calculate public key Y_A where

$$Y_A = \alpha^{X_A} \bmod q.$$

At user B

select private key X_B such that $1 < X_B < q$

calculate public key Y_B where

$$Y_B = \alpha^{X_B} \bmod q.$$

At user A side choose random no r such that

$$1 \leq r \leq q-1$$

Shared secret key $K = Y_B^{r} \bmod q$

* Encryption :-

At user A

$$C_1 = \alpha^{r} \bmod q$$

$$C_2 = KM \bmod q$$

* Decryption :-

at user B

$$K = C_1^{X_B} \bmod q$$

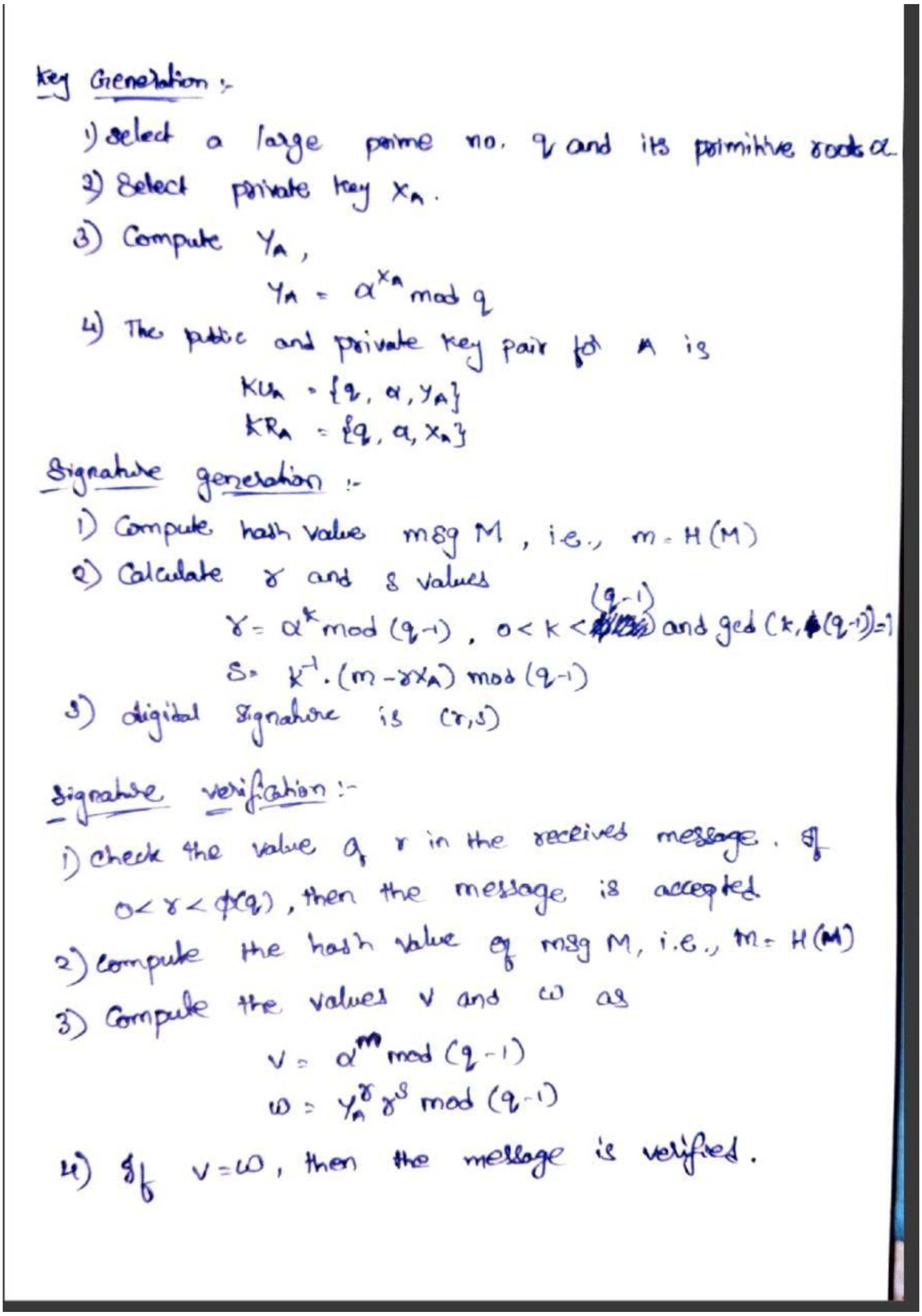
$$M = C_2 \cdot K^{-1} \bmod q$$

PROGRAM

```
#include<stdio.h>
#include<math.h>
void main()
{
    int q,alpha,xa,xb,ya,yb,k1,k,kInv=1,r,c1,c2,msg,decrymsg;
    printf("Enter a prime number : ");
    scanf("%d",&q);
    printf("Enter the primitive root of q : ");
    scanf("%d",&alpha);
    printf("Enter a private key of user A : ");
    scanf("%d",&xa);
    printf("Enter a private key of user B : ");
    scanf("%d",&xb);
    printf("Choose a random number r : ");
    scanf("%d",&r);
    ya=(int)pow(alpha,xa)%q;
    yb=(int)pow(alpha,xb)%q;
    k1=(int)pow(yb,r)%q;
    printf("Enter plain text: ");
    scanf("%d",&msg);
    c1=(int)pow(alpha,r)%q;
    c2=(k1*msg)%q;
    printf("Encrypted message is (%d,%d)\n",c1,c2);
    k=(int)pow(c1,xb)% q;
    while((kInv*k)%q != 1)
        kInv++;
    decrymsg=(c2*kInv)%q;
    printf("Decrypted message is %d\n",decrymsg);
}
```

ElGamal Digital Signature

ALGORITHM



Key Generation :-

1) select a large prime no. q and its primitive roots α.
2) Select private key X_A.
3) Compute Y_A,

$$Y_A = \alpha^{X_A} \bmod q$$

4) The public and private key pair for A is

$$KU_A = \{q, \alpha, Y_A\}$$
$$KR_A = \{q, \alpha, X_A\}$$

Signature generation :-

1) Compute hash value msg M, i.e., $m = H(M)$
2) Calculate r and s values

$$r = \alpha^k \bmod (q-1), \quad 0 < k < (q-1) \text{ and } \gcd(k, \phi(q-1)) = 1$$
$$S = k^{-1} \cdot (m - r X_A) \bmod (q-1)$$

3) digital signature is (r, s)

Signature verification :-

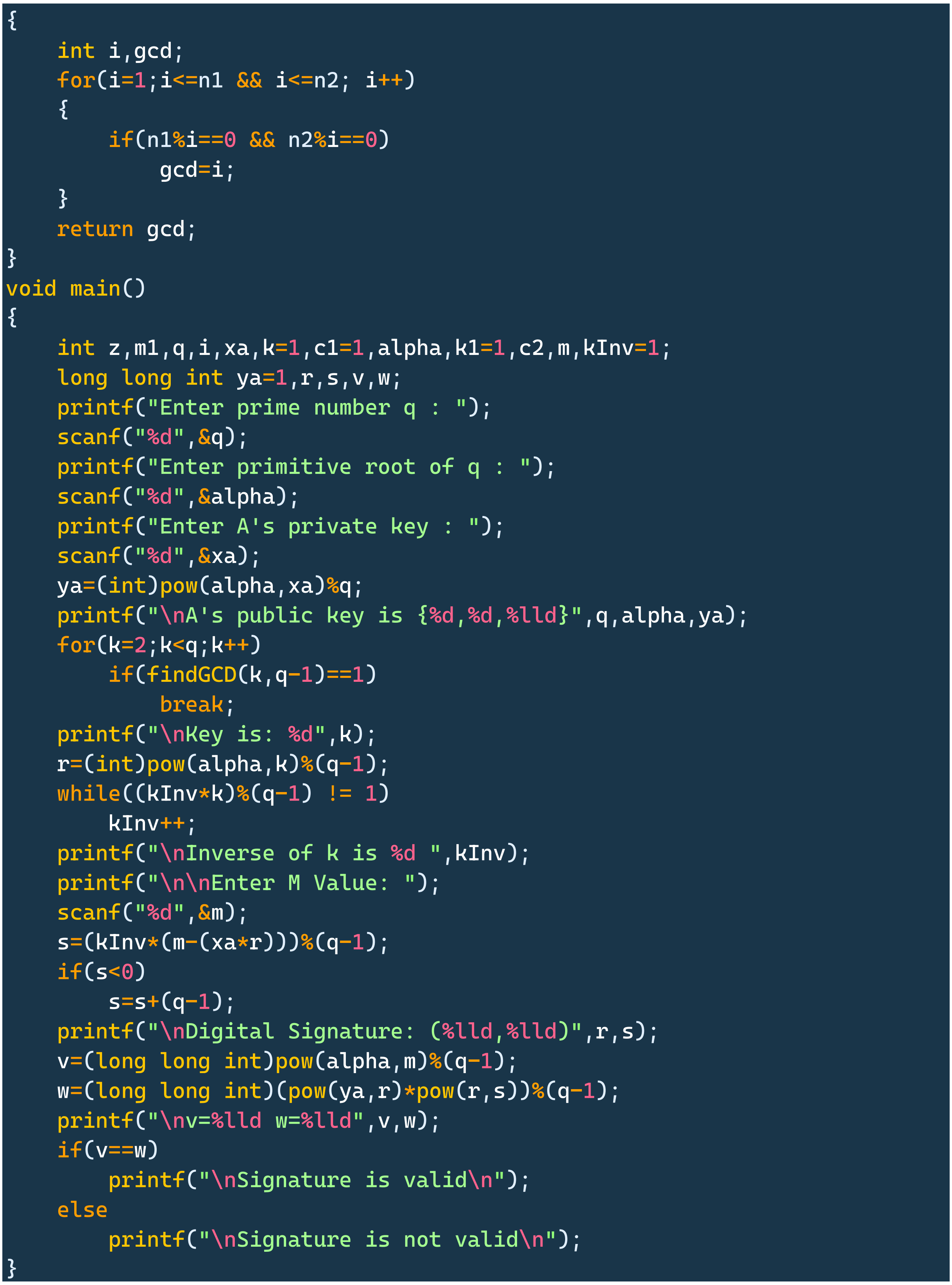
1) check the value of r in the received message. If $0 < r < \phi(q)$, then the message is accepted
2) compute the hash value of msg M, i.e., $m = H(M)$
3) Compute the values V and W as

$$V = \alpha^m \bmod (q-1)$$
$$W = Y_A^r r^s \bmod (q-1)$$

4) If $V = W$, then the message is verified.

PROGRAM

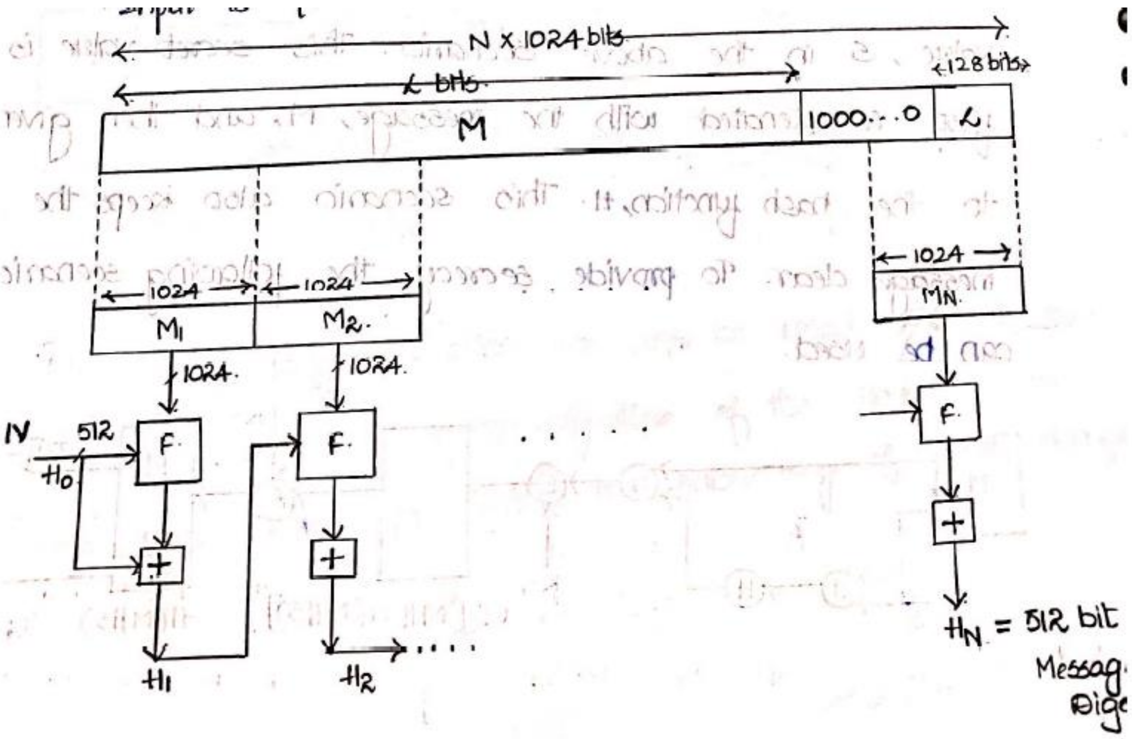
```
#include<stdio.h>
#include<math.h>
int findGCD(int n1,int n2)
```



```
{
    int i,gcd;
    for(i=1;i<=n1 && i<=n2; i++)
    {
        if(n1%i==0 && n2%i==0)
            gcd=i;
    }
    return gcd;
}
void main()
{
    int z,m1,q,i,xa,k=1,c1=1,alpha,k1=1,c2,m,kInv=1;
    long long int ya=1,r,s,v,w;
    printf("Enter prime number q : ");
    scanf("%d",&q);
    printf("Enter primitive root of q : ");
    scanf("%d",&alpha);
    printf("Enter A's private key : ");
    scanf("%d",&xa);
    ya=(int)pow(alpha,xa)%q;
    printf("\nA's public key is {%d,%d,%lld}",q,alpha,ya);
    for(k=2;k<q;k++)
        if(findGCD(k,q-1)==1)
            break;
    printf("\nKey is: %d",k);
    r=(int)pow(alpha,k)%(q-1);
    while((kInv*k)%(q-1) != 1)
        kInv++;
    printf("\nInverse of k is %d ",kInv);
    printf("\n\nEnter M Value: ");
    scanf("%d",&m);
    s=(kInv*(m-(xa*r)))%(q-1);
    if(s<0)
        s=s+(q-1);
    printf("\nDigital Signature: (%lld,%lld)",r,s);
    v=(long long int)pow(alpha,m)%(q-1);
    w=(long long int)(pow(ya,r)*pow(r,s))%(q-1);
    printf("\nv=%lld w=%lld",v,w);
    if(v==w)
        printf("\nSignature is valid\n");
    else
        printf("\nSignature is not valid\n");
}
```

SHA-1

BLOCK DIAGRAM



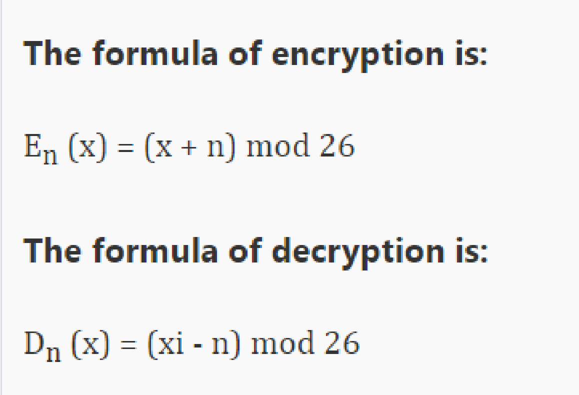
PROGRAM

```
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Scanner;

public class SHA1
{
    public static String encryptThisString(String input)
    {
        try{
            MessageDigest md = MessageDigest.getInstance("SHA-1");
            byte[] messageDigest = md.digest(input.getBytes());
            BigInteger no= new BigInteger(1,messageDigest);
            String hashtext= no.toString(16);
            while(hashtext.length()<32){
                hashtext="0"+hashtext;
            }
            return hashtext;
        }
        catch(NoSuchAlgorithmException e){
            throw new RuntimeException(e);
        }
    }
    public static void main(String[] args )
    {
        System.out.println("HashCode generated by SHA-1 for:");
        Scanner sc=new Scanner(System.in);
        String s1=sc.nextLine();
        System.out.println(":" +encryptThisString(s1));
        String s2=sc.nextLine();
        System.out.println(":" +encryptThisString(s2));
        sc.close();
    }
}
```

Caesar Cipher

ALGORITHM



The formula of encryption is:

$E_n(x) = (x + n) \bmod 26$

The formula of decryption is:

$D_n(x) = (xi - n) \bmod 26$

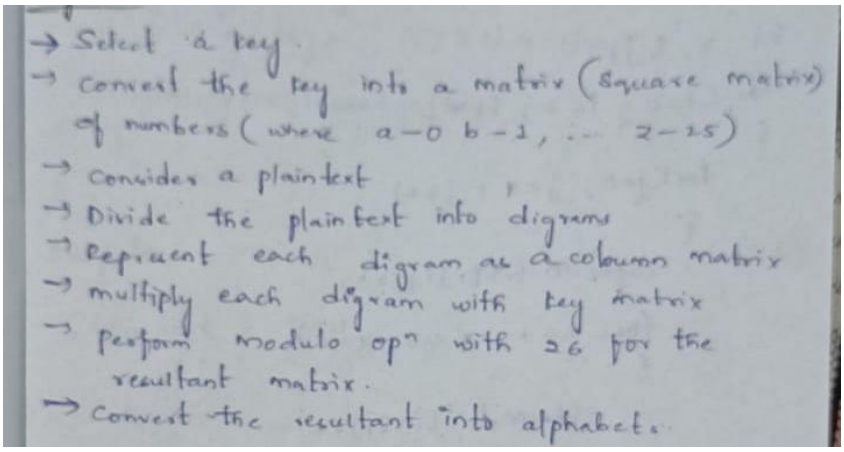
PROGRAM

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

void main()
{
    char msg[30];
    int key;
    printf("Enter the msg : ");
    scanf("%s", msg);
    printf("Enter the key : ");
    scanf("%d", &key);
    for (int i = 0; i < msg[i]!='\0'; i++) {
        if (isupper(msg[i]))
            msg[i] = (msg[i] + key - 'A') % 26 + 'A';
        else
            msg[i] = (msg[i] + key - 'a') % 26 + 'a';
    }
    printf("Encrypted Message : %s", msg);
}
```

Hill Cipher

ALGORITHM



→ Select a key.

→ Convert the key into a matrix (Square matrix) of numbers (where a – 0 b – 1, ... z – 25)

→ Consider a plaintext

→ Divide the plaintext into digrams

→ Represent each digram as a column matrix

→ multiply each digram with key matrix

→ Perform modulo op^n with 26 for the resultant matrix.

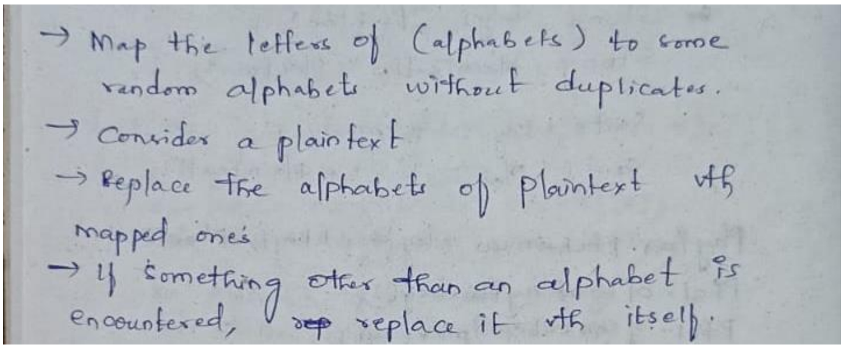
→ Convert the resultant into alphabets.

PROGRAM

```
#include <stdio.h>
#include <string.h>
void main()
{
    char msg[20];
    int key[20][20],c[20],cipher[20];
    int determinant=0,t=0;
    printf("Enter plain text : ");
    scanf("%s",msg);
    printf("Enter Key : \n");
    for(int i=0;i<strlen(msg);i++)
        for(int j=0;j<strlen(msg);j++)
            scanf("%d",&key[i][j]);
    printf("Message in Column matrix form : ");
    for(int i=0;i<strlen(msg);i++) {
        c[i]=msg[i]-65;
        printf("%d ",c[i]);
    }
    for(int i=0;i<strlen(msg);i++) {
        t=0;
        for(int j=0;j<strlen(msg);j++)
            t=t+(key[i][j]*c[j]);
        cipher[i]=t%26;
    }
    printf("\nEncrypted Cipher Text :");
    for(int i=0;i<strlen(msg);i++)
        printf("%c",cipher[i]+65);
}
```

Monoalphabetic Cipher

ALGORITHM



→ Map the letters of (alphabets) to some random alphabets without duplicates.

→ Consider a plaintext

→ Replace the alphabets of plaintext wth mapped ones

→ If something other than an alphabet is encountered, ~~sep~~ replace it wth itself.

PROGRAM

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

void main()
{
    char key[26], msg[26];
    printf("Enter key : ");
    for (int i = 0; i < 26; i++)
    {
        scanf("%c", &key[i]);
        for (int j = 0; j < i; j++)
        {
            if (key[i] == key[j])
            {
                printf("error : Invalid key");
                exit(0);
            }
        }
    }
    printf("Enter plain text :");
    scanf("%s", msg);
    char cipher[26];
    for (int i = 0; msg[i] != '\0'; i++)
    {
        if (isupper(msg[i]))
            cipher[i] = key[msg[i] - 65];
        else
            cipher[i] = key[msg[i] - 97];
    }
    printf("Cipher text is : %s", cipher);
}
```

Rail Fence

ALGORITHM

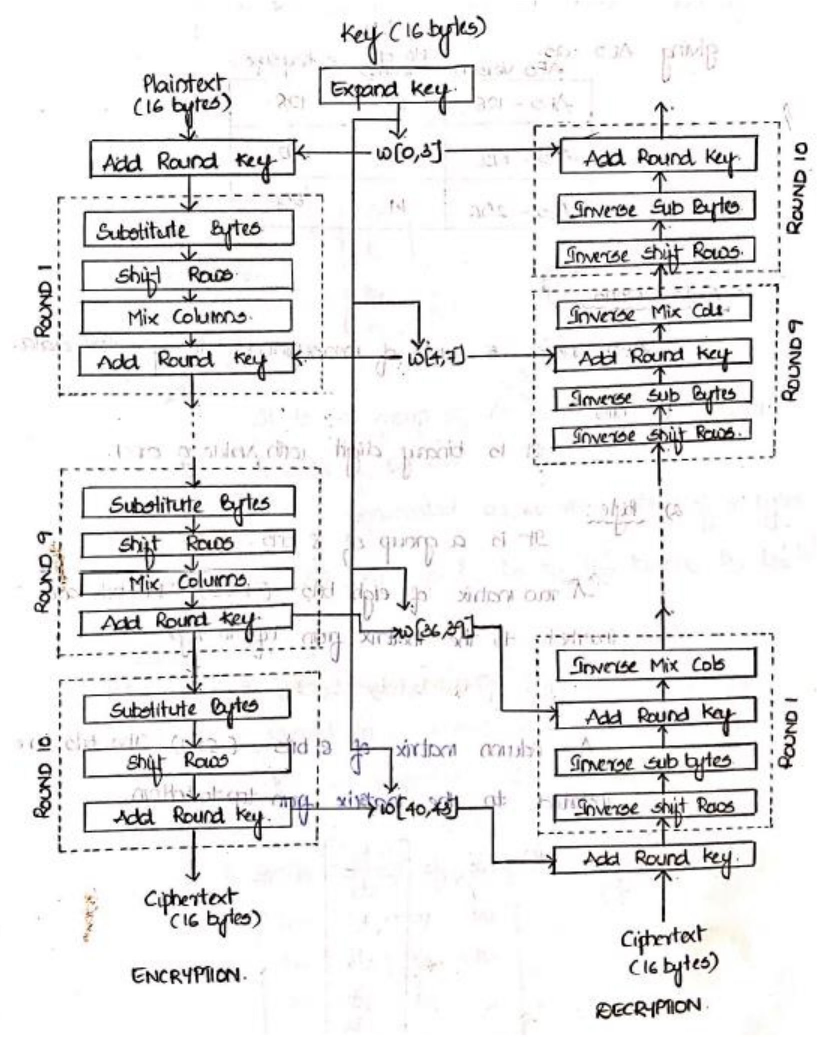
- Write the letters in a zigzag pattern, going downwards and upwards between the levels of the top and bottom imaginary rails.
- Next, all the letters are read off and concatenated, to produce one line of ciphertext.
- The letters should be read in rows, usually from the top row down to the bottom one.

PROGRAM

```
#include <stdio.h>
#include <string.h>
int main()
{
    char msg[25] = "Meet me in toga Party";
    int i, j = 0, k = 0, l = 0;
    char s1[20], s2[20];
    for (i = 0; i < strlen(msg); i++) {
        if (msg[i] != ' '){
            if (j % 2 == 0){
                s1[k++] = msg[i];
                j++;
            }
            else{
                s2[l++] = msg[i];
                j++;
            }
        }
    }
    s2[l]='\0';
    strcat(s1, s2);
    printf("cipher text : %s", s1);
}
```

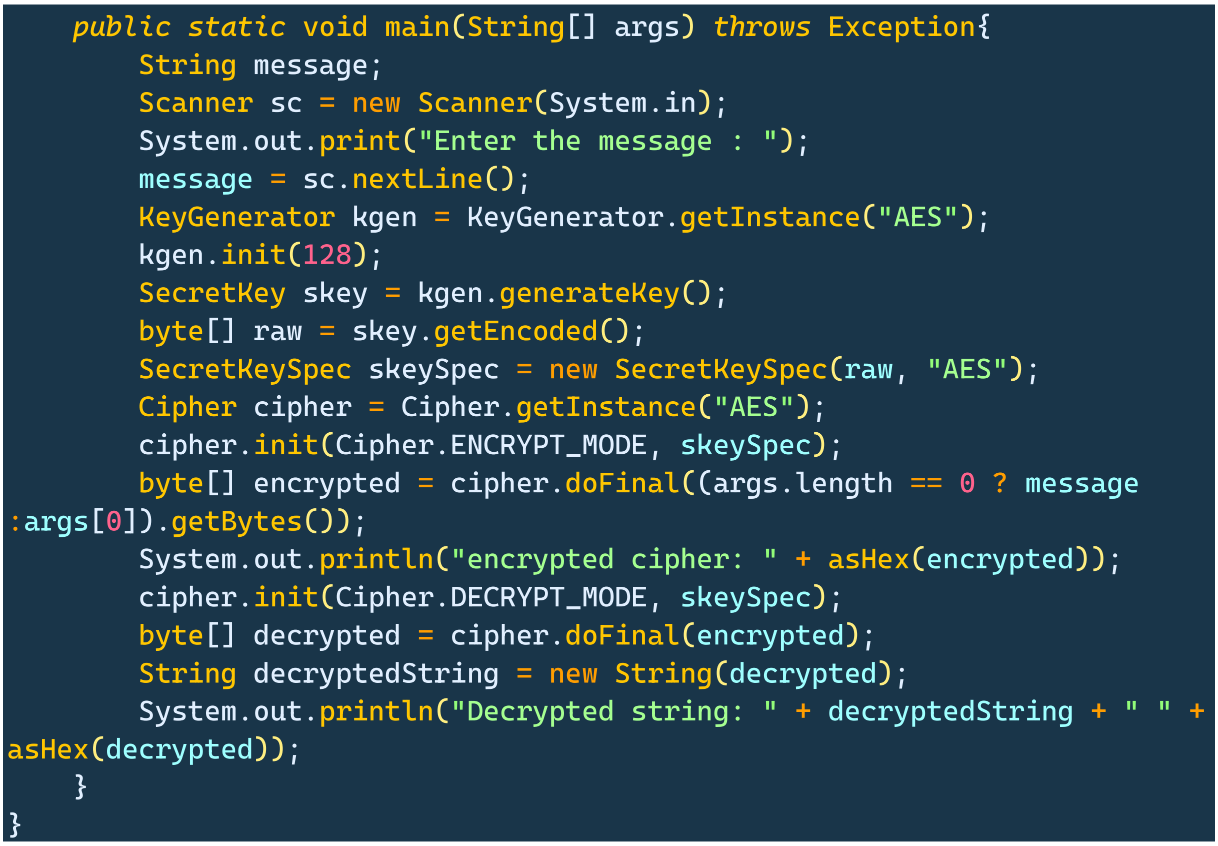
AES

BLOCK DIAGRAM



PROGRAM

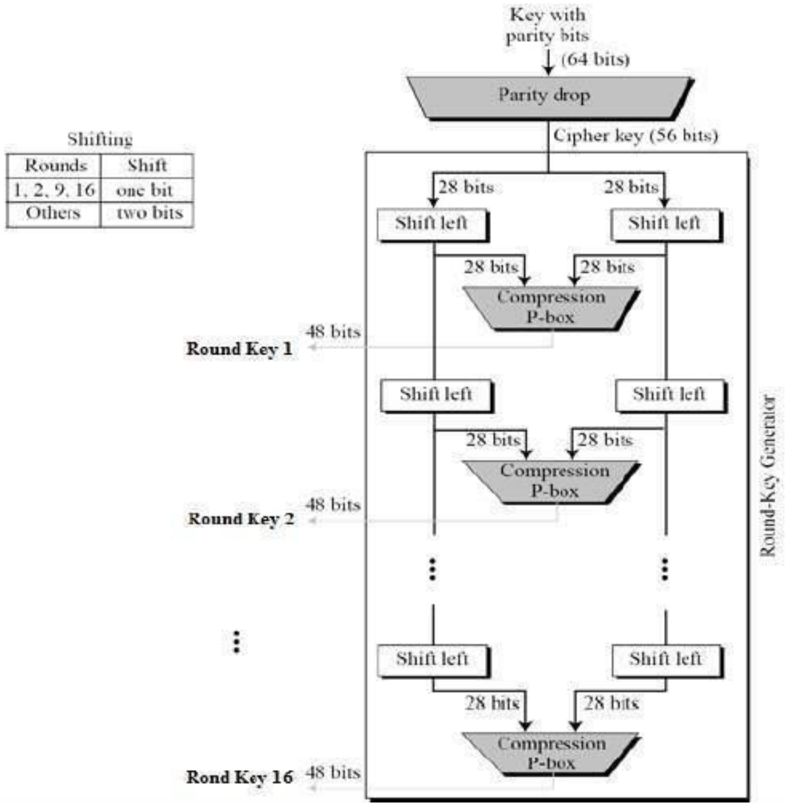
```
import java.util.Scanner;
import javax.crypto.*;
import javax.crypto.spec.*;
public class AES {
    public static String asHex (byte buf[]) {
        StringBuffer strbuf = new StringBuffer(buf.length * 2);
        int i;
        for (i = 0; i < buf.length; i++) {
            if (((int) buf[i] & 0xff) < 0x10)
                strbuf.append("0");
            strbuf.append(Long.toString((int) buf[i] & 0xff, 16));
        }
        return strbuf.toString();
    }
```



```java
    public static void main(String[] args) throws Exception{
        String message;
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the message : ");
        message = sc.nextLine();
        KeyGenerator kgen = KeyGenerator.getInstance("AES");
        kgen.init(128);
        SecretKey skey = kgen.generateKey();
        byte[] raw = skey.getEncoded();
        SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
        byte[] encrypted = cipher.doFinal((args.length == 0 ? message
:args[0]).getBytes());
        System.out.println("encrypted cipher: " + asHex(encrypted));
        cipher.init(Cipher.DECRYPT_MODE, skeySpec);
        byte[] decrypted = cipher.doFinal(encrypted);
        String decryptedString = new String(decrypted);
        System.out.println("Decrypted string: " + decryptedString + " " +
asHex(decrypted));
    }
}
```

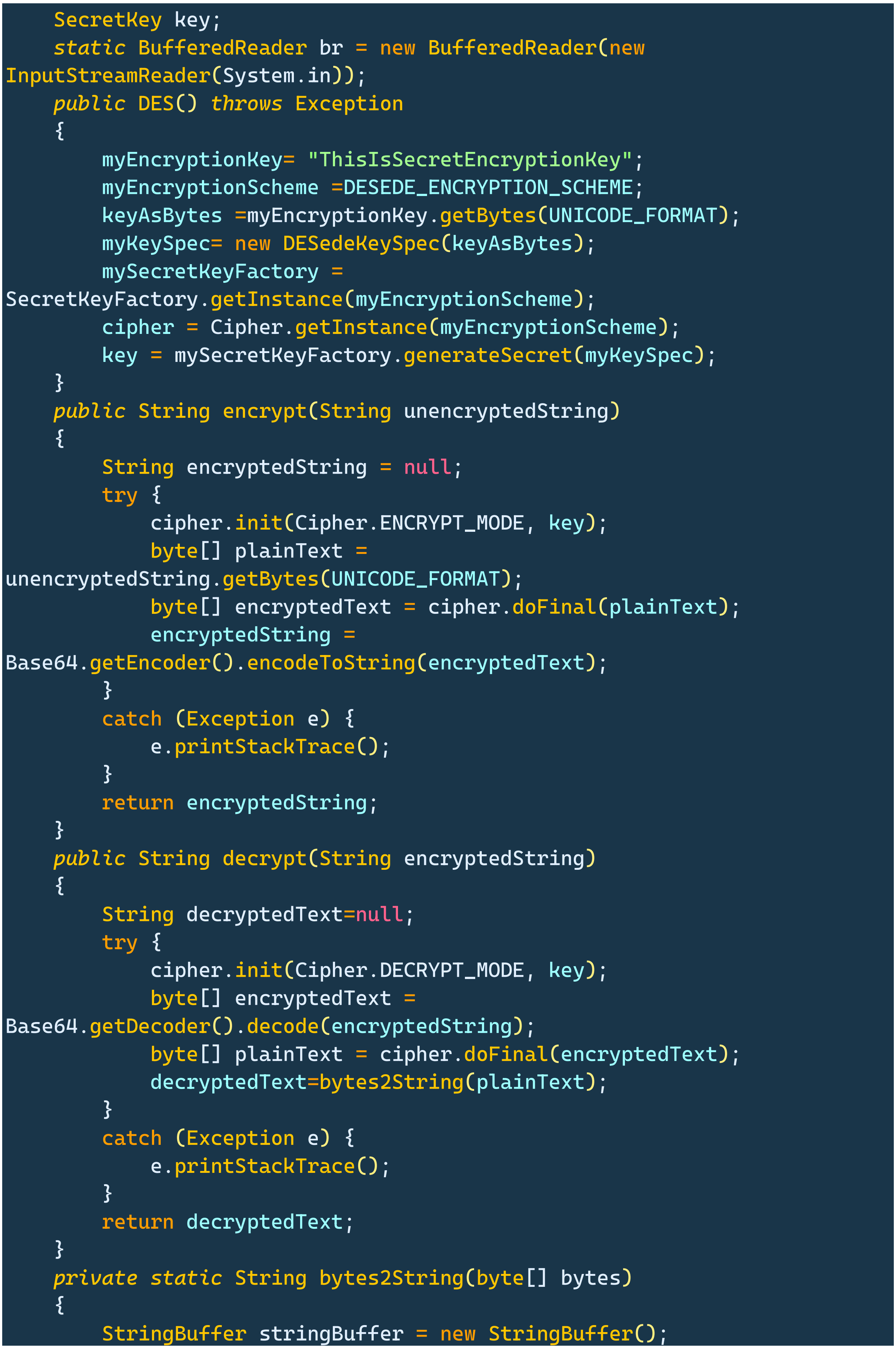
DES

BLOCK DIAGRAM



PROGRAM

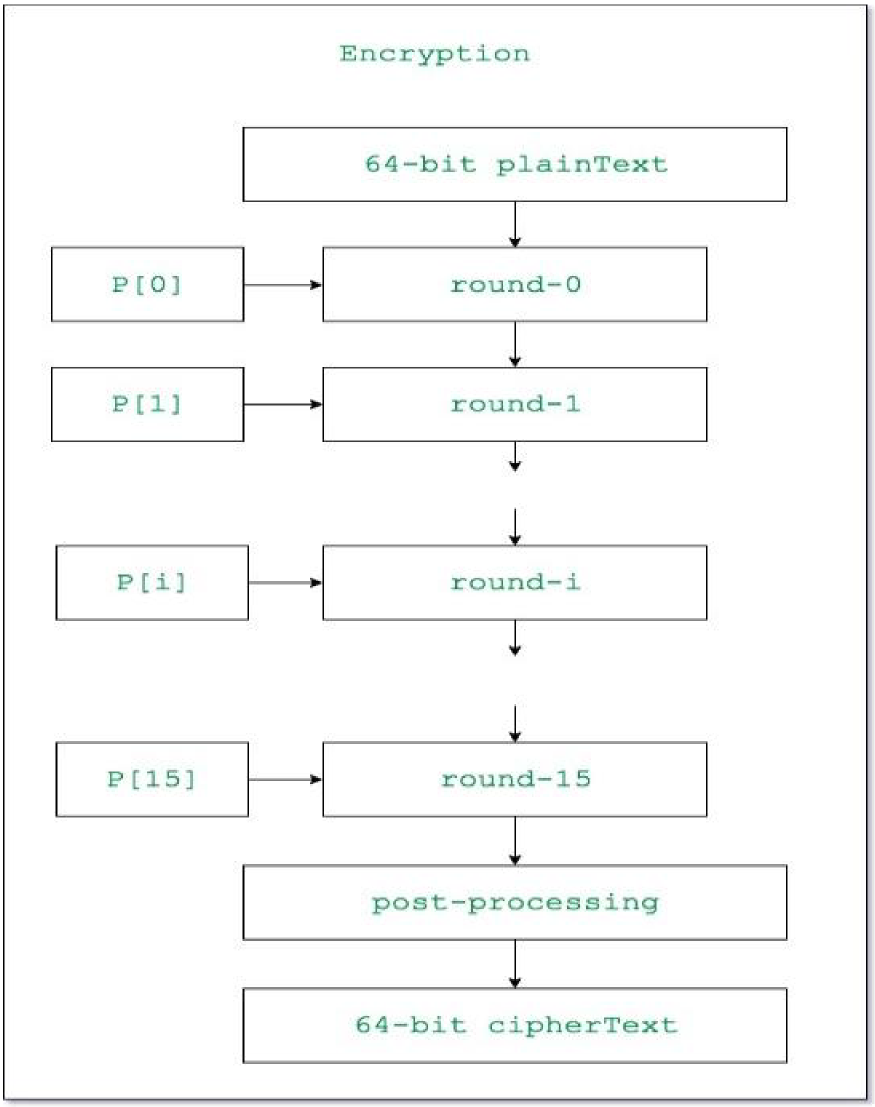
```
import java.util.*;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.security.spec.KeySpec;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.DESedeKeySpec;
import java.util.Base64.Decoder;
import java.util.Base64.Encoder;
public class DES
{
    private static final String UNICODE_FORMAT = "UTF8";
    public static final String DESEDE_ENCRYPTION_SCHEME = "DESEDE";
    private KeySpec myKeySpec;
    private SecretKeyFactory mySecretKeyFactory;
    private Cipher cipher;
    byte[] keyAsBytes;
    private String myEncryptionKey;
    private String myEncryptionScheme;
```



```
    SecretKey key;
    static BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
    public DES() throws Exception
    {
        myEncryptionKey= "ThisIsSecretEncryptionKey";
        myEncryptionScheme =DESEDE_ENCRYPTION_SCHEME;
        keyAsBytes =myEncryptionKey.getBytes(UNICODE_FORMAT);
        myKeySpec= new DESedeKeySpec(keyAsBytes);
        mySecretKeyFactory =
SecretKeyFactory.getInstance(myEncryptionScheme);
        cipher = Cipher.getInstance(myEncryptionScheme);
        key = mySecretKeyFactory.generateSecret(myKeySpec);
    }
    public String encrypt(String unencryptedString)
    {
        String encryptedString = null;
        try {
            cipher.init(Cipher.ENCRYPT_MODE, key);
            byte[] plainText =
unencryptedString.getBytes(UNICODE_FORMAT);
            byte[] encryptedText = cipher.doFinal(plainText);
            encryptedString =
Base64.getEncoder().encodeToString(encryptedText);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        return encryptedString;
    }
    public String decrypt(String encryptedString)
    {
        String decryptedText=null;
        try {
            cipher.init(Cipher.DECRYPT_MODE, key);
            byte[] encryptedText =
Base64.getDecoder().decode(encryptedString);
            byte[] plainText = cipher.doFinal(encryptedText);
            decryptedText=bytes2String(plainText);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        return decryptedText;
    }
    private static String bytes2String(byte[] bytes)
    {
        StringBuffer stringBuffer = new StringBuffer();
        for (int i = 0; i <bytes.length;i++)
            stringBuffer.append((char) bytes[i]);
        return stringBuffer.toString();
    }
    public static void main(String args []) throws Exception
    {
        System.out.print("Enter the string: ");
        DES myEncryptor= new DES();
        String stringToEncrypt = br.readLine();
        String encrypted = myEncryptor.encrypt(stringToEncrypt);
        String decrypted = myEncryptor.decrypt(encrypted);
        System.out.println("\nString To Encrypt: " +stringToEncrypt);
        System.out.println("\nEncrypted Value : " +encrypted);
        System.out.println("\nDecrypted Value : " +decrypted);
System.out.println("");
    }
}
```

Blowfish

BLOCK DIAGRAM



PROGRAM

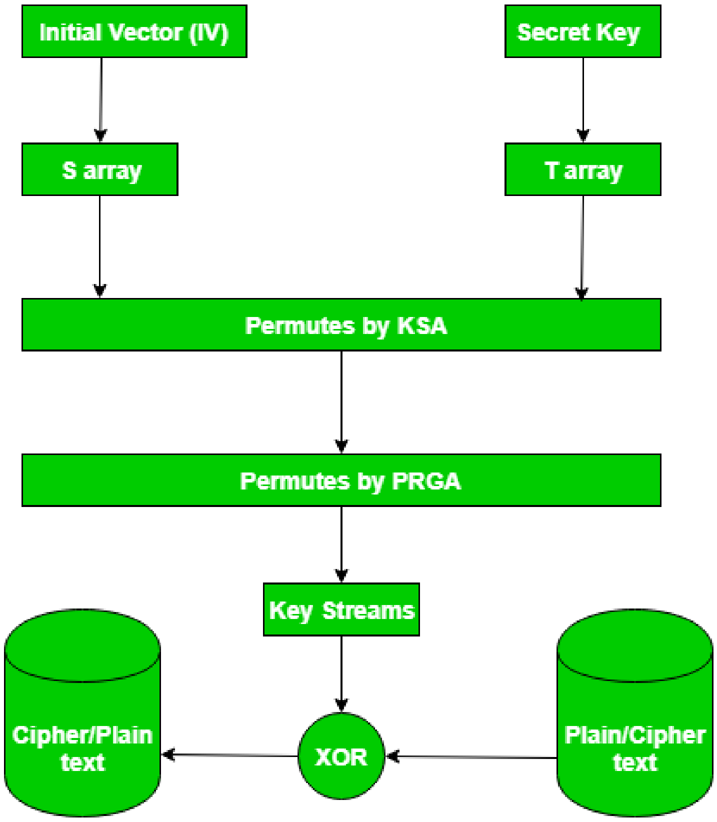
```
import java.io.*;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.*;
import javax.swing.JOptionPane;
import javax.crypto.spec.*;


public class Blowfish
{
    public static void main(String[] args) throws Exception {
```

```
    KeyGenerator kgen=KeyGenerator.getInstance("Blowfish");
    Cipher cipher = Cipher.getInstance("Blowfish");
    SecretKey skey = kgen.generateKey();
    byte[] raw = skey.getEncoded();
    SecretKeySpec KS = new SecretKeySpec(raw, "Blowfish");
    cipher.init(Cipher.ENCRYPT_MODE, skey);
    String inputText = JOptionPane.showInputDialog("Input your message:
");
    byte[] encrypted = cipher.doFinal(inputText.getBytes());
    cipher.init(Cipher.DECRYPT_MODE, skey);
    byte[] decrypted = cipher.doFinal(encrypted);
    JOptionPane.showMessageDialog(JOptionPane.getRootFrame(),"\nEncrpted
text: " + new String(encrypted)+ "\n" + "\nDecrpted text: " + new
String(decrypted));
    }
}
```

RC4

BLOCK DIAGRAM



PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main()
{
    int k[4],p[4],s[8],t[8];
    int i,j,k1,t1;
    int temp=0;
    printf("enter k:\n");
    for(i=0;i<4;i++)
        scanf("%d",&k[i]);
    printf("enter p:\n");
    for(i=0;i<4;i++)
        scanf("%d",&p[i]);
    for(i=0;i<8;i++)
        s[i]=i;
    for(i=0;i<8;i++) {
        if(i<4)
            t[i]=k[i];
        else
            t[i]=k[i-4];
    }
    j=0;
    for(i=0;i<=7;i++) {
        j=(j+s[i]+t[i])%8;
        temp=s[i];
        s[i]=s[j];
        s[j]=temp;
    }
    printf("after ksa state vector is:\n");
    for(i=0;i<8;i++)
        printf("%d",s[i]);
    printf("\nafter prga cipher text is:\n");
    i=0;
    j=0;
    while(i<4) {
        i=(i+1)%8;
        j=(j+s[i])%8;
        temp=s[i];
        s[i]=s[j];
        s[j]=temp;
        t1=(s[i]+s[j])%8;
        k1=s[t1];
        k1=k1^p[i-1];
        printf("%d",k1);
    }
    return 0;
}
```