Ex. No.: 4 **Date:**

RSA

Aim:

To implement RSA asymmetric key cryptosystem using C.

Algorithm:

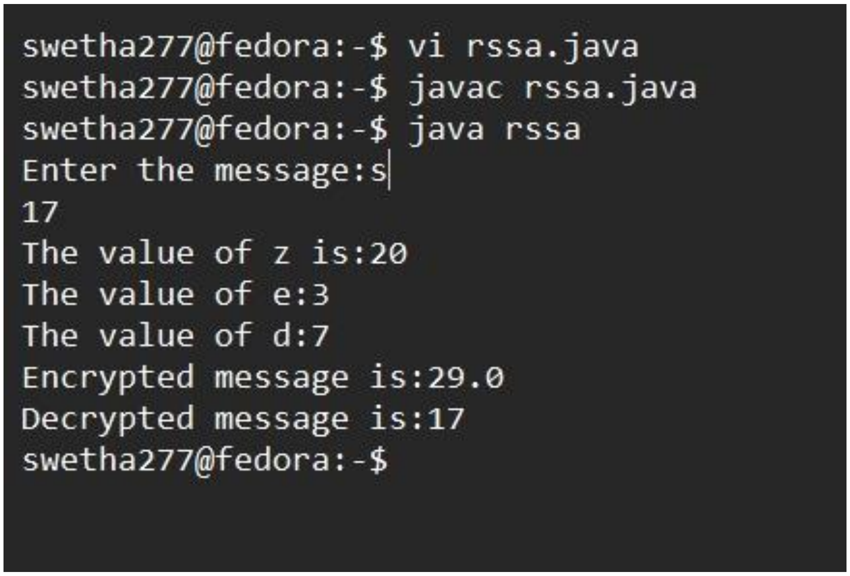
1. Select two large prime numbers p and q
2. Compute n=pxq
3. Choose system modulus: Ø(n)=(p-1)x(q-1)
4. Select a random encryption key e such that gcd(e,Ø(n)=1
5. Decrypt by computing d=1 mod Ø(n)
6. Print the public key{e,n}
7. Print the private key{d,n}

Program Code:

```
#include <stdio.h>
#include <math.h>
int power(int,unsigned int,int);
int gcd(int,int);
int multiplicativeInverse(int,int,int);
int main()
{
  int p,q,n,e,d,phi,M,C;

  printf("\nEnter two prime numbers p and q that are not equal : ");
  scanf("%d %d",&p,&q);
  n = p * q;
  phi = (p - 1)*(q - 1);
  printf("Phi(%d) = %d",n,phi);
  printf("\nEnter the integer e : ");
  scanf("%d",&e);
  if(e >= 1 && e < phi)
  {
          if(gcd(phi,e)!=1)
          {
                  printf("\nChoose proper value for e !!!\n");
                  return 1;
          }
  }

  //Key Generation
  d = multiplicativeInverse(e,phi,n);

  printf("\nPublic Key PU = {%d,%d}",e,n);
  printf("\nPrivate Key PR = {%d,%d}",d,n);

  //Encryption
  printf("\nMessage M = ");
  scanf("%d",&M);
  C = power(M,e,n);
  printf("\nCiphertext C = %d \n",C);

  //Decryption
  M = power(C,d,n);
  printf("\nDecrypted Message M = %d \n",M);


  return 0;
}


int power(int x, unsigned int y, int p)
{
    int res = 1;      // Initialize result

    x = x % p; // Update x if it is more than or equal to p

    while (y > 0)
    {
        // If y is odd, multiply x with
        result if (y & 1)
            res = (res*x) % p;


        // y must be even now
        y = y>>1; // y = y/2 x
        = (x*x) % p;
    }
    return res;
}


int gcd ( int a, int b )
{
  int c;
  while ( a != 0 )
  {
            c = a;
            a = b % a;
            b = c;
  }
  return b;
}
```

```
int multiplicativeInverse(int a, int b, int n)
{
        int sum,x,y;

      for(y=0;y<n;y++)
      {
            for(x=0;x<n;x++)
            {
                  sum=a*x + b*(-y);
                  if(sum==1)
                        return x;
            }
      }
}
```

Output:



```
swetha277@fedora:-$ vi rssa.java
swetha277@fedora:-$ javac rssa.java
swetha277@fedora:-$ java rssa
Enter the message:s
17
The value of z is:20
The value of e:3
The value of d:7
Encrypted message is:29.0
Decrypted message is:17
swetha277@fedora:-$
```

Result:

Ex. No.: 5 **Date:**

DIFFIE-HELLMAN KEY EXCHANGE

The simplest and the original implementation of the protocol uses the multiplicative group of integers modulo *p*, where *p* is prime, and *g* is a primitive root modulo *p*. Here is an example of the protocol, with non-secret values in blue, and secret values in **red**.

1. Alice and Bob agree to use a prime number $p = 23$ and base $g = 5$ (which is a primitive root modulo 23).
2. Alice chooses a secret integer $\boldsymbol{a} = \mathbf{6}$, then sends Bob $A = g^{\boldsymbol{a}} \bmod p$
   - $A = 5^{\mathbf{6}} \bmod 23 = 8$
3. Bob chooses a secret integer $\boldsymbol{b} = \mathbf{15}$, then sends Alice $B = g^{\boldsymbol{b}} \bmod p$
   - $B = 5^{\mathbf{15}} \bmod 23 = 19$
4. Alice computes $\boldsymbol{s} = B^{\boldsymbol{a}} \bmod p$
   - $\boldsymbol{s} = 19^{\mathbf{6}} \bmod 23 = \mathbf{2}$
5. Bob computes $\boldsymbol{s} = A^{\boldsymbol{b}} \bmod p$
   - $\boldsymbol{s} = 8^{\mathbf{15}} \bmod 23 = \mathbf{2}$
6. Alice and Bob now share a secret (the number **2**).

Aim:

To implement Diffie-Hellman key exchange using C.

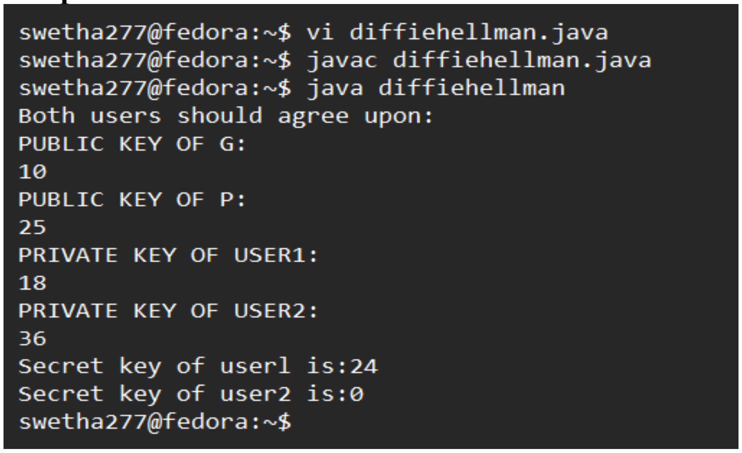
Algorithm:

1. Get a prime number q as input from the user.
2. Get a value xa and xb which is less than q.
3. Calculate primitive root α
4. For each user A, generate a key Xa < q
5. Compute public key, α pow(Xa) mod q
6. Each user computes Ya
7. Print the values of exchanged keys.

Program Code:

```
//This program uses fast exponentiation function power instead of pow library function
#include <stdio.h>
#include <math.h>
int power( int,unsigned int,int);
int main()
{
  int x,y,z,count,ai[20][20];
  int alpha,xa,xb,ya,yb,ka,kb,q;
  printf("\nEnter a Prime Number \"q\":");
  scanf("%d",&q);
  printf("\nEnter a No \"xa\" which is less than value of
  q:"); scanf("%d",&xa);
  printf("\nEnter a No \"xb\" which is less than value of q:");
  scanf("%d",&xb);
  printf("\nEnter alpha:");
  scanf("%d",&alpha);
  ya = power(alpha,xa,q);
  yb = power(alpha,xb,q);
  ka = power(yb,xa,q);
  kb = power(ya,xb,q);
  printf("\nya = %d \nyb = %d \nka = %d \nkb = %d \n",ya,yb,ka,kb);
  if(ka == kb)
          printf("\nThe secret keys generated by User A and User B are same\n");
  else
      printf("\nThe secret keys generated by User A and User B are not same\n");
  return 0;
}

int power(int x, unsigned int y, int p)
{
    int res = 1;      // Initialize result

    x = x % p; // Update x if it is more than or equal to p

    while (y > 0)
    {
        // If y is odd, multiply x with
        result if (y & 1)
            res = (res*x) % p;


        // y must be even now
        y = y>>1; // y = y/2 x
        = (x*x) % p;
    }
    return res;
}
```

Output:



```
swetha277@fedora:~$ vi diffiehellman.java
swetha277@fedora:~$ javac diffiehellman.java
swetha277@fedora:~$ java diffiehellman
Both users should agree upon:
PUBLIC KEY OF G:
10
PUBLIC KEY OF P:
25
PRIVATE KEY OF USER1:
18
PRIVATE KEY OF USER2:
36
Secret key of userl is:24
Secret key of user2 is:0
swetha277@fedora:~$
```

Result: