18/12/2020

12.

Janet Reshma J

312217104062

DIFFIE Hellman Key Exchange
with
Miller Rabin Algorithm check.

AIM:

To develop a java program to implement the Diffie Hellman Key exchange algorithm. To implement the Miller Rabin primality checking algorithm to check the primality of the input prime number of DH Key exchange Algorithm.

ALGORITHM:

DH Key Exchange:

Sender: (A)

1. The sender and the receiver share a prime number q and an integer α such that $\alpha < q$

   α is the primitive root of q.

2. sender generates a private key X_A, $X_A < q$
3. sender calculates a public key $Y_A = \alpha^{X_A} \bmod q$.
4. sender receives the receiver's public key.
5. The shared key is calculated using the formula

$$K = (Y_B)^{X_A} \bmod q.$$

Receiver:

1. Sender and receiver share a prime number q and primitive root α, $\alpha < q$.

2. Receiver generates a private key X_B such that $X_B < q$.

3. Receiver calculates a public key $Y_B = \alpha^{X_B} \bmod q$.

4. Receiver receives sender's public key Y_A.

5. The shared key is calculated by.

$$K = (Y_A)^{X_B} \bmod q.$$

Miller Rabin Algorithm:

Odd integers

1. Get a number as input say n.

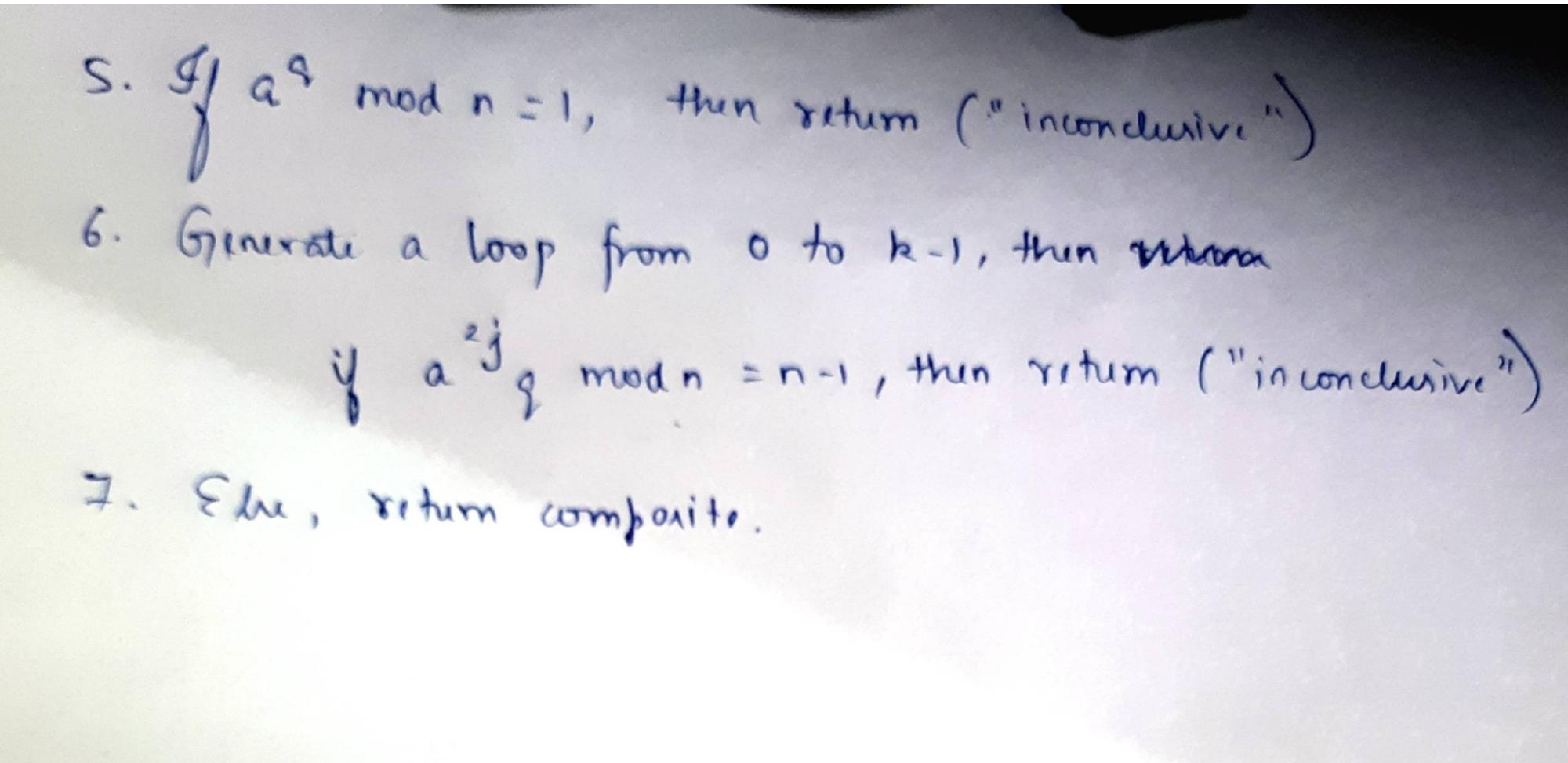
2. Calculate $n-1$

3. Represent the number $n-1$ as $2^k q$.

where k is the division, $k > 0$.

q = the odd number

4. Select a random integer a, $1 < a < n-1$



```
5. If $a^q$ mod n = 1, then return ("inconclusive")
6. Generate a loop from 0 to k-1, then ~~return~~
       if $a^{2^j q}$ mod n = n-1, then return ("inconclusive")
7. Else, return composite.
```

Janet Reshma J
312217104062

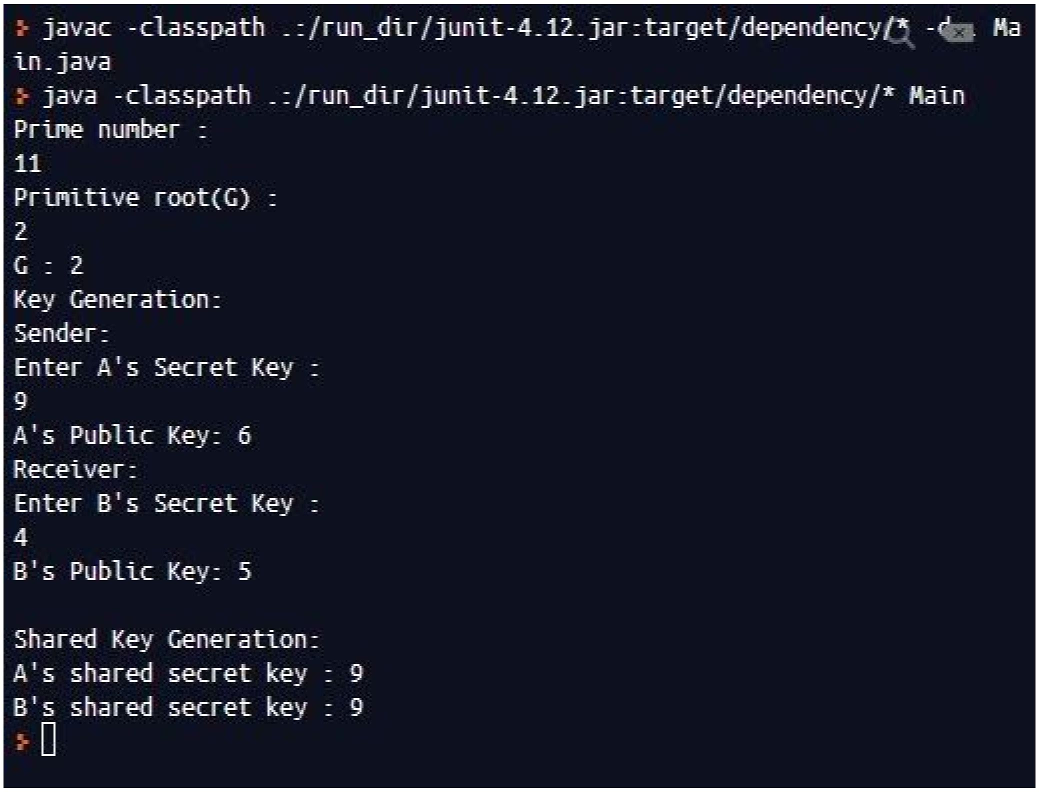
PROGRAM:

```
import java.math.BigInteger;
import java.util.*;

class Main {
 public static void main(String[] args) {
   Scanner in=new Scanner(System.in);
   System.out.println("Prime number : ");
   int P=in.nextInt();
   while(!isPrime(P,1)){
      System.out.println("Prime number (P) : ");
      P=in.nextInt();
   }
   System.out.println("Primitive root(G) : ");
   int G=in.nextInt();
   //int P=BigInteger.probablePrime(15, new Random()).intValue();

   //System.out.println("Shared prime number"+ P);

   System.out.println("G : "+ G);
   BigInteger g= new BigInteger(""+G);
   BigInteger p= new BigInteger(""+P);
   System.out.println("Key Generation:\nSender:");
   System.out.println("Enter A's Secret Key :");
   BigInteger xa=new BigInteger(in.next());
   BigInteger ya=g.modPow(xa,p);
   System.out.println("A's Public Key: "+ ya);
   System.out.println("Receiver:");
   System.out.println("Enter B's Secret Key :");
   BigInteger xb=new BigInteger(in.next());
   BigInteger yb=g.modPow(xb,p);
   System.out.println("B's Public Key: "+ yb);
   BigInteger shA=yb.modPow(xa,p);
   BigInteger shB=ya.modPow(xb,p);
   System.out.println("\nShared Key Generation:");
   System.out.println("A's shared secret key : "+shA);
   System.out.println("B's shared secret key : "+shB);
   in.close();
}
static boolean 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;

}
static int power(int x,int y,int p){
  int res=1;
  while(y>0){
    if(y%2 ==1)
      res=(res*x)%p;
    y=y>>1;
    x=(x*x)%p;
  }
  return res;
}
static void findPrimefactors(HashSet<Integer> s,int n){
  while(n%2==0){
    s.add(2);
    n=n/2;

  }
  for (int i=3;i<=Math.sqrt(n);i=i+2){
    while(n%i ==0){
      s.add(i);
      n=n/i;
    }
  }
  if(n>2)
    s.add(n);
}
```

```
static int findPrimitive(int n){
  HashSet <Integer> s=new HashSet <Integer>();
  if(isPrime(n)==false)
    return -1;
  int phi =n-1;
  findPrimefactors(s,phi);
  for(int r=2;r<=phi;r++){
    boolean flag=false;
    for(Integer a:s){
      if(power(r,phi/(a),n)==1){
        flag=true;
        break;
      }
    }
    if(flag==false)
      return r;
  }
  return -1;
}
static boolean millerTest(int d,int n){
  int a=2+(int)(Math.random()%(n-4));
  int x=power(a,d,n);
  if(x==1 || x==n-1)
     return true;
  while(d!=n-1){
    x=(x*x)%n;
    d*=2;
    if(x==1)
      return false;
    if(x==n-1)
      return true;
  }
  return false;

}
static boolean isPrime(int n,int k){
  if(n<=1 || n==4)
    return false;
  if(n<=3)
    return true;
```

```
   int d =n-1;
   while(d%2==0)
     d/=2;
   for(int i=0;i<k;i++){
     if(!millerTest(d,n))
       return false;
   }
   return true;
 }
}
```

OUTPUT:



```
› javac -classpath .:/run_dir/junit-4.12.jar:target/dependency/* -d   Ma
in.java
› java -classpath .:/run_dir/junit-4.12.jar:target/dependency/* Main
Prime number :
11
Primitive root(G) :
2
G : 2
Key Generation:
Sender:
Enter A's Secret Key :
9
A's Public Key: 6
Receiver:
Enter B's Secret Key :
4
B's Public Key: 5

Shared Key Generation:
A's shared secret key : 9
B's shared secret key : 9
› 
```