SEMESTER PRACTICALS.

SECURITY LAB.

Name: ARAVIND.K

3122171040013.

Date: 17.12.2020

Aim:

To Develop a Java Program to implement the Diffie-Hellman Key exchange algorithm. Implement the Miller Rabin Primality checking algorithm to check the primality of the input prime number for DH key exchange algorithm.

Algorithm:

(1) Diffie-Hellman Key exchange:

1.1) consider a prime 'q'.

1.2) select 'α' (Primitive root). such that ($\alpha < q$)

{α & q} is Public Known quantities.

1.3)

let X → denotes Private.

Y → denotes Public

→ now assume X_A (Private key for user A). such that $X_A < q$

→ similarly assume X_B (Private key for User B). such that $X_B < q$

1.4) now, Public key of A is given by

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

& Public key of B is given by

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

] - Public known quantities.

1.5)

now the secret key is

$(Y_B)^{X_A} \bmod q$ - for user A &

$(Y_A)^{X_B} \bmod q$ - for user B

note: $(Y_B)^{X_A} \bmod q = (Y_A)^{X_B} \bmod q$.

Thus. The secket key is generated.

(2) Miller - Rabin Primality checking algorithm.

- suppose 'n' is the numbe we want to check

step 1: find integers k, m such that

$(n-1) = 2^k \cdot m$

step 2: select a random integer a

(s.t). $1 < a < n-1$

step 3: If $a^m \bmod n = 1$, Then return ("~~maybe~~ prime")

step 4: for $j = 0$ to $k-1$ do.

step 5: if $a^{2^j m} \bmod n = n-1$, Then return ("prime")

step 6: Else return composite.

note: ~~The~~ If algorithm returns prime - it's not 100%. true, (can also give false positive result)
but if algorithm returns composite - its 100%. true (or) correct

~~NOTE~~

Name:Aravind.k
Roll no:312217104013
Date:17.12.2020

AIM:
Develop a java program to implement the Diffie-Hellman Key exchange algorithm. Implement the Miller Rabin primality checking algorithm to check the primality of the input prime number for DH key exchange algorithm.

Code:

```
import java.math.BigInteger;
import java.util.*;
public class Main {
   public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        // int P = BigInteger.probablePrime(30, new Random()).intValue();
        // int P = 29;
        System.out.println("enter value of P:");
        int P = in.nextInt();
        int G = primitiveRoot(P);
        System.out.println("P = "+P+" G = "+G);
        BigInteger g = new BigInteger(""+G);
        BigInteger p = new BigInteger(""+P);

        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("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 A_sh = yb.modPow(xa, p);
        BigInteger B_sh = ya.modPow(xb, p);
```

```
        System.out.println("A's shared secret = "+A_sh);
        System.out.println("B's shared secret = "+B_sh);


        in.close();

    }

    public static int primitiveRoot(int p)
    {
        if (!(new BigInteger(""+p).isProbablePrime(1))) return -1;
        // if(!(isPrime(p))) return -1;
        int phi = p-1;
        ArrayList<Integer> fact = primeFactors(phi);
        for (int res=2; res<=p; ++res) {
            Boolean ok = true;
            for (int i=0; i<fact.size() && ok; ++i)
                ok &= powmod (res, phi / fact.get(i), p) != 1;
            if (ok)  return res;
        }
        return -1;
    }

    public static ArrayList<Integer> primeFactors(int n)
    {
        ArrayList<Integer> facts = new ArrayList<>();
        for (int i=2; i*i<=n; ++i)
            if (n % i == 0) {
                facts.add(i);
                while (n % i == 0)
                    n /= i;
            }

        return facts;

    }
    public static int powmod(int a,int b,int mod)
    {
        if (b==0)
```

```
            return 1;
        if (b==1)
            return a%mod;
        int temp = powmod(a,b/2,mod);
        temp = (temp*temp);
        if (b%2!=0)
            temp*= a;
        return temp%mod;
    }

    static boolean millerTest(int d, int n) {
        int a = 2 + (int)(Math.random() % (n - 4));
        int x = powmod(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) {
        if (n <= 1 || n == 4)
            return false;
        if (n == 3)
            return true;
        int d = n - 1;
        int k=n-2;
        while (d % 2 == 0)
        {
            d /= 2;
            // k++;
        }
        for (int i = 0; i < k; i++)
```

```
            if (!millerTest(d, n))
                return false;

        return true;
    }
}
```

OUTPUT: