**EX.NO.7 DIFFIE HELLMAN KEY EXCHANGE ALGORITHM**

**CODE:**

import java.io.IOException;

import java.math.BigInteger;

import java.util.\*;

public class DHK2 {

private BigInteger p;

private BigInteger g;

private BigInteger phi;

private BigInteger Xa;

private BigInteger Xb;

private BigInteger Ya;

private BigInteger Yb;

private BigInteger Ka;

private BigInteger Kb;

private int bitlength = 32;

private int noOfIterations = 5;

private Random rand;

private static final BigInteger ZERO = BigInteger.ZERO;

private static final BigInteger ONE = BigInteger.ONE;

private static final BigInteger TWO = new BigInteger("2");

private static final BigInteger THREE = new BigInteger("3");

private static final BigInteger FOUR = new BigInteger("4");

public DHK2() {

rand = new Random();

generatePrime();

getPrimitiveRoot();

}

public void getPrimitiveRoot() {

phi = p.subtract(ONE);

HashSet<BigInteger> primeFactors = getPrimeFactors();

ArrayList<BigInteger> primitiveRoots = new ArrayList<>();

for (BigInteger r = BigInteger.TWO;r.compareTo(phi) < 0;r = r.add(BigInteger.ONE)) {

boolean flg = false;

for (BigInteger l : primeFactors) {

BigInteger phiBig = phi.divide(l);

BigInteger pRootBig = r.modPow(phiBig, p);

if (pRootBig.compareTo(BigInteger.valueOf(1)) == 0) {

flg = true;

break;

}

}

if (!flg) {

primitiveRoots.add(r);

}

}

g= primitiveRoots.get(new Random().nextInt(primitiveRoots.size()));

}

public HashSet<BigInteger> getPrimeFactors() {

HashSet<BigInteger> primesFactors = new HashSet<>();

while (phi.mod(TWO).signum() == 0) {

primesFactors.add(TWO);

phi = phi.divide(TWO);

}

for (BigInteger i = THREE; i.compareTo(phi.sqrt()) <= 0; i = i.add(TWO)) {

if (phi.mod(i).signum() == 0) {

primesFactors.add(i);

phi = phi.divide(i);

}

}

if (phi.compareTo(TWO) > 0) {

primesFactors.add(phi);

}

return primesFactors;

}

void generatePrime() {

byte[] b = new byte[bitlength / 8];

rand.nextBytes(b);

p = new BigInteger(b);

while (!isPrime(p, noOfIterations)) {

rand.nextBytes(b);

p = new BigInteger(b);

}

}

boolean miillerTest(BigInteger d, BigInteger n) {

BigInteger maxLimit = n.subtract(TWO);

BigInteger minLimit = TWO;

BigInteger bigInteger = maxLimit.subtract(minLimit);

int len = maxLimit.bitLength();

BigInteger a = new BigInteger(len, rand);

if (a.compareTo(minLimit) < 0) a = a.add(minLimit);

if (a.compareTo(bigInteger) >= 0) a = a.mod(bigInteger).add(minLimit);

BigInteger x = a.modPow(d, n);

if (x.compareTo(ONE) == 0 || x.compareTo(n.subtract(ONE)) == 0) return true;

while (d.compareTo(n.subtract(ONE)) != 0) {

x = x.multiply(x).mod(n);

d = d.multiply(TWO);

if (x.compareTo(ONE) == 0) return false;

if (x.compareTo(n.subtract(ONE)) == 0) return true;

}

return false;

}

boolean isPrime(BigInteger n, int k) {

if (n.compareTo(ONE) <= 0 || n.compareTo(FOUR) == 0) return false;

if (n.compareTo(THREE) <= 0) return true;

BigInteger d = n.subtract(ONE);

while (d.mod(TWO).signum() == 0) d = d.divide(TWO);

for (int i = 0; i < k; i++) if (!miillerTest(d, n)) return false;

return true;

}

void userAgen(){

BigInteger maxLimit = g;

BigInteger minLimit = ONE;

BigInteger bigInteger = maxLimit.subtract(minLimit);

int len = maxLimit.bitLength();

Xa = new BigInteger(len, rand);

if (Xa.compareTo(minLimit) < 0)

Xa = Xa.add(minLimit);

if (Xa.compareTo(bigInteger) >= 0)

Xa = Xa.mod(bigInteger).add(minLimit);

Ya=g.modPow(Xa,p);

}

void userBgen(){

BigInteger maxLimit = g;

BigInteger minLimit = ONE;

BigInteger bigInteger = maxLimit.subtract(minLimit);

int len = maxLimit.bitLength();

Xb = new BigInteger(len, rand);

if (Xb.compareTo(minLimit) < 0)

Xb = Xb.add(minLimit);

if (Xb.compareTo(bigInteger) >= 0)

Xb = Xb.mod(bigInteger).add(minLimit);

Yb=g.modPow(Xb,p);

}

void secretAgen(){

Ka=Yb.modPow(Xa,p);

}

void secretBgen(){

Kb=Ya.modPow(Xb,p);

}

public static void main(String[] args) throws IOException {

Scanner sc = new Scanner(System.in);

System.out.println("\nDHK ALGORITHM");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

DHK2 dhk = new DHK2();

System.out.println("\nKey Generation");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("\nPrime no, P is (in Big Integer)");

System.out.println("---------------------\n" + dhk.p);

System.out.println("\nPrimitive root, G is (in Big Integer)");

System.out.println("---------------------\n" + dhk.g);

dhk.userAgen();

System.out.println("\nA's private key is (in Big Integer)");

System.out.println("---------------------\n" + dhk.Xa);

System.out.println("\nA's public key is (in Big Integer)");

System.out.println("---------------------\n" + dhk.Ya);

dhk.userBgen();

System.out.println("\nB's private key is (in Big Integer)");

System.out.println("---------------------\n" + dhk.Xb);

System.out.println("\nB's public key is (in Big Integer)");

System.out.println("---------------------\n" + dhk.Yb);

dhk.secretAgen();

System.out.println("\nA's shared key is (in Big Integer)");

System.out.println("---------------------\n" + dhk.Ka);

dhk.secretBgen();

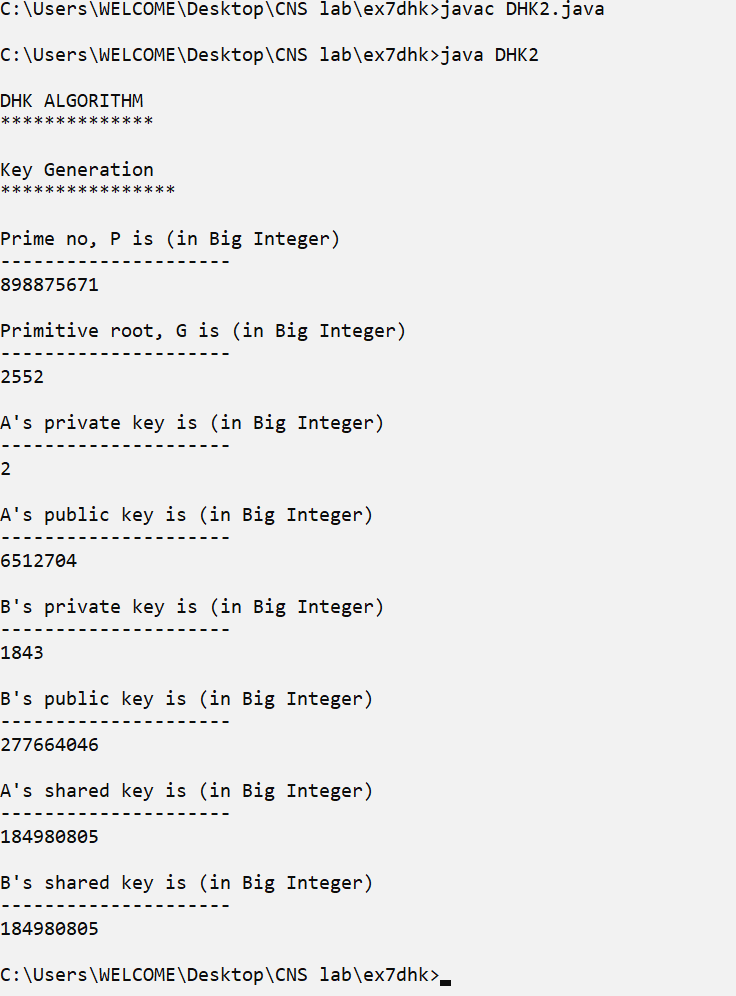
System.out.println("\nB's shared key is (in Big Integer)");

System.out.println("---------------------\n" + dhk.Kb);

}

}

**Example 1:**



**Example 2:**

