**AIM: To write a C program to implement the signature scheme named digital signature standard (Euclidean Algorithm).**

**ALGORITHM:**

STEP-1: Alice and Bob are investigating a forgery case of x and y.

STEP-2: X had document signed by him but he says he did not sign that document digitally.

STEP-3: Alice reads the two prime numbers p and a.

STEP-4: He chooses a random co-primes alpha and beta and the x’s original signature x.

STEP-5: With these values, he applies it to the elliptic curve cryptographic equation to obtain y.

STEP-6: Comparing this ‘y’ with actual y’s document, Alice concludes that y is a forgery.

**PROGRAM: (Digital Signature Standard)**

import java.util.\*;

import java.math.BigInteger;

class dsaAlg {

final static BigInteger one = new BigInteger("1");

final static BigInteger zero = new BigInteger("0");

public static BigInteger getNextPrime(String ans)

{

BigInteger test = new BigInteger(ans);

while (!test.isProbablePrime(99))

e:

{

test = test.add(one);

}

return test;

}

public static BigInteger findQ(BigInteger n)

{

BigInteger start = new BigInteger("2");

while (!n.isProbablePrime(99))

{

while (!((n.mod(start)).equals(zero)))

{

start = start.add(one);

}

n = n.divide(start);

}

return n;

}

public static BigInteger getGen(BigInteger p, BigInteger q,

Random r)

{

BigInteger h = new BigInteger(p.bitLength(), r);

h = h.mod(p);

return h.modPow((p.subtract(one)).divide(q), p);

}

public static void main (String[] args) throws

java.lang.Exception

{

Random randObj = new Random();

BigInteger p = getNextPrime("10600"); /\* approximate

prime \*/

BigInteger q = findQ(p.subtract(one));

BigInteger g = getGen(p,q,randObj);

System.out.println(" \n simulation of Digital Signature

Algorithm \n");

System.out.println(" \n global public key components

are:\n");

System.out.println("\np is: " + p);

System.out.println("\nq is: " + q);

System.out.println("\ng is: " + g);

BigInteger x = new BigInteger(q.bitLength(), randObj);

x = x.mod(q);

BigInteger y = g.modPow(x,p);

BigInteger k = new BigInteger(q.bitLength(), randObj);

k = k.mod(q);

BigInteger r = (g.modPow(k,p)).mod(q);

BigInteger hashVal = new BigInteger(p.bitLength(),

randObj);

BigInteger kInv = k.modInverse(q);

BigInteger s = kInv.multiply(hashVal.add(x.multiply(r)));

s = s.mod(q);

System.out.println("\nsecret information are:\n");

System.out.println("x (private) is:" + x);

System.out.println("k (secret) is: " + k);

System.out.println("y (public) is: " + y);

System.out.println("h (rndhash) is: " + hashVal);

System.out.println("\n generating digital signature:\n");

System.out.println("r is : " + r);

System.out.println("s is : " + s);

BigInteger w = s.modInverse(q);

BigInteger u1 = (hashVal.multiply(w)).mod(q);

BigInteger u2 = (r.multiply(w)).mod(q);

BigInteger v = (g.modPow(u1,p)).multiply(y.modPow(u2,p));

v = (v.mod(p)).mod(q);

System.out.println("\nverifying digital signature

(checkpoints)\n:");

System.out.println("w is : " + w);

System.out.println("u1 is : " + u1);

System.out.println("u2 is : " + u2);

System.out.println("v is : " + v);

if (v.equals(r))

{

System.out.println("\nsuccess: digital signature is

verified!\n " + r);

}

else

{

System.out.println("\n error: incorrect digital

signature\n ");

}

}

}

**OUTPUT:**

