|  |  |  |  |
| --- | --- | --- | --- |
| **Ex. No. 06** | **Digital Signature Algorithm** | | |
| Date of Exercise | 24 – 02 - 2015 | Date of Output Verification | 24 – 02 - 2015 |

**Question**

Develop an application where the transactions are verified using Digital Signature Algorithm

**Aim**

To Develop an Application implementing the concept of Digital Signature Algorithm for Verification in java

**Procedure**

The first part of the DSA algorithm is the public key and private key generation, which can be described as:

* Choose a prime number q, which is called the prime divisor.
* Choose another primer number p, such that p-1 mod q = 0. p is called the prime modulus.
* Choose an integer g, such that 1 < g < p, g\*\*q mod p = 1 and g = h\*\*((p–1)/q) mod p. q is also called g's multiplicative order modulo p.
* Choose an integer, such that 0 < x < q.
* Compute y as g\*\*x mod p.
* Package the public key as {p,q,g,y}.
* Package the private key as {p,q,g,x}.

The second part of the DSA algorithm is the signature generation and signature verification, which can be described as:

To generate a message signature, the sender can follow these steps:

* Generate the message digest h, using a hash algorithm like SHA1.
* Generate a random number k, such that 0 < k < q.
* Compute r as (g\*\*k mod p) mod q. If r = 0, select a different k.
* Compute i, such that k\*i mod q = 1. i is called the modular multiplicative inverse of k modulo q.
* Compute s = i\*(h+r\*x) mod q. If s = 0, select a different k.
* Package the digital signature as {r,s}.

To verify a message signature, the receiver of the message and the digital signature can follow these steps:

* Generate the message digest h, using the same hash algorithm.
* Compute w, such that s\*w mod q = 1. w is called the modular multiplicative inverse of s modulo q.
* Compute u1 = h\*w mod q.
* Compute u2 = r\*w mod q.
* Compute v = (((g\*\*u1)\*(y\*\*u2)) mod p) mod q.
* If v == r, the digital signature is valid.

**Program**

*DSA Sender*

import java.io.\*;

import java.math.\*;

import java.net.\*;

import java.util.\*;

//@author William Scott

public class NetworkSecurityLabExp6DigitalSignatureAlgorithmSender {

//Network Security Lab - Exp 6 - Digital Signature Algorithm - Sender

public static BufferedReader br;

public static String input, hmt = "", opr;

public static BigInteger p, q, h, g, x, y, k, r, s, w, v, u1, u2, p1q, hm;

public static int ti = 0;

public static Random rand = new Random();

public static ArrayList al = new ArrayList();

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

System.out.println("UR12CS135 - DSA Sender");

ServerSocket ss = new ServerSocket(3000);

System.out.println("\nWaiting for Receiver to Connect");

Socket so = ss.accept();

System.out.println("\nReceiver Connected!");

br = new BufferedReader(new FileReader("..\\Exp 6 - Input File.doc"));

input = br.readLine().toUpperCase();

System.out.println("\nInput: " + input);

InputStream is = so.getInputStream();

OutputStream os = so.getOutputStream();

PrintWriter pw = new PrintWriter(os, true);

for (char c : input.toCharArray()) {

ti = (int) c % 26;

if (ti < 10) {

hmt += "0" + ti;

} else {

hmt += ti;

}

}

hm = new BigInteger(hmt);

System.out.println("Hashed String: " + hm);

hm = hm.mod(new BigInteger("10"));

System.out.println("Hashed Input : " + hm);

System.out.println("\n----Global Public-Key Components----");

getpq();

p1q = p.subtract(BigInteger.ONE).divide(q);

do {

h = getrandom(1, p.subtract(BigInteger.ONE));

g = h.modPow(p1q, p);

} while (g.compareTo(BigInteger.ONE) != 1);

System.out.println("h: " + h);

System.out.println("g: " + g);

opr = p + ":" + q + ":" + g;

System.out.println("\nSending p:q:g:h(m): " + opr);

pw.println(opr);

System.out.println("\n---------User’s Private Key---------");

x = getrandom(0, q);

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

System.out.println("\n---------User’s Public Key----------");

y = g.modPow(x, p);

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

System.out.println("\nSending y: " + y);

pw.println(y);

System.out.println("\n--User's Per-Message Secret Number--");

k = getrandom(0, q);

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

System.out.println("\n--------------Signing---------------");

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

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

s = getinverse(q, k).multiply(hm.add(x.multiply(r))).mod(q);

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

opr = hm + ":" + s + ":" + r;

System.out.println("Sending h(m):s:r: " + opr);

pw.println(opr);

}

public static BigInteger getrandom(int a, BigInteger b) {

BigInteger c;

int ca, cb;

do {

c = new BigInteger(b.bitLength(), rand);

ca = c.compareTo(BigInteger.valueOf(a));

cb = c.compareTo(b);

System.out.println(" \*f: " + c);

} while (ca != 1 || cb != -1);

return c;

}

public static BigInteger getinverse(BigInteger a, BigInteger b) throws IOException {

long i = 0, count = 0;

BigInteger t, ra;

al.clear();

if (b.intValue() == 0) {

System.out.println("Problem Finding the invesrse, as the denominator is 0, Please run Again!");

System.exit(0);

}

do {

t = a.multiply(BigInteger.valueOf((int) i)).add(BigInteger.ONE);

ra = t.mod(b);

if (!al.contains(q)) {

al.add(q);

count++;

}

i++;

System.out.println("--i: " + count + " t : " + t + " ra: " + ra);

} while (ra.compareTo(BigInteger.ZERO) != 0);

System.out.println(" inv: " + t.divide(b));

return t.divide(b);

}

public static void getpq() {

p = BigInteger.probablePrime(15, rand);

System.out.println("p: " + p);

System.out.println("p.long: " + p.longValue());

al.clear();

BigInteger bt;

long t = 0, tt;

do {

do {

tt = (long) (2 + (Math.random() \* (p.longValue() - 2)));

System.out.println(" \*q: " + tt);

} while (!checkprime(tt));

q = BigInteger.valueOf(tt);

bt = p.subtract(BigInteger.ONE).mod(q);

if (!al.contains(q)) {

al.add(q);

t++;

}

System.out.println(" t : " + t + " q: " + q + " bt : " + bt);

} while (bt != BigInteger.ZERO);

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

}

public static boolean checkprime(long a) {

if (a <= 1) {

return false;

}

for (long i = 2; i < a; i++) {

if (a % i == 0) {

return false;

}

}

return true;

}

}

*DSA Receiver*

import java.io.\*;

import java.math.BigInteger;

import java.net.\*;

import java.util.ArrayList;

//@author William Scott

public class NetworkSecurityLabExp6DigitalSignatureAlgorithmReceiver {

//Network Security Lab - Exp 6 - Digital Signature Algorithm - Receiver

public static String t1, t2[];

public static BigInteger p, q, g, k, r, y, s, w, v, u1, u2, p1q, hm;

public static ArrayList al=new ArrayList();

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

System.out.println("UR12CS135 - DSA Receiver");

Socket so = new Socket("127.0.0.1", 3000);

System.out.println("\nConnected to Sender\n");

InputStream is = so.getInputStream();

OutputStream os = so.getOutputStream();

BufferedReader br = new BufferedReader(new InputStreamReader(is));

System.out.println("----------Receiving Values----------");

t1 = br.readLine();

t2 = t1.split(":");

p = new BigInteger(t2[0]);

q = new BigInteger(t2[1]);

g = new BigInteger(t2[2]);

System.out.println("Received p: " + p);

System.out.println("Received q: " + q);

System.out.println("Received g: " + g);

y = new BigInteger(br.readLine());

System.out.println("\nReceived y: " + y);

t1 = br.readLine();

t2 = t1.split(":");

hm = new BigInteger(t2[0]);

s = new BigInteger(t2[1]);

r = new BigInteger(t2[2]);

System.out.println("\nReceived h(m): " + hm);

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

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

System.out.println("\n--------------Verifying-------------");

w = getinverse(q, s).mod(q);

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

u1 = (hm.multiply(w)).mod(q);

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

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

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

v = (((g.pow(u1.intValue())).multiply(y.pow(u2.intValue()))).mod(p)).mod(q);

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

System.out.println("\n------------Testing v & r-----------");

if (v.equals(r)) {

System.out.println("v and r are equal, hence Authenticated");

} else {

System.out.println("v and r are not equal, hence not Authenticated");

}

}

public static BigInteger getinverse(BigInteger a, BigInteger b) {

long i = 0, count=0;

BigInteger t, ra;

al.clear();

if (b.intValue() == 0) {

System.out.println("Problem Finding the invesrse, as the denominator is 0, Please run Again!");

System.exit(0);

}

do {

t = a.multiply(BigInteger.valueOf((int) i)).add(BigInteger.ONE);

ra = t.mod(b);

if (!al.contains(q)) {

al.add(q);

count++;

}

i++;

System.out.println("--i: " + count + " t : " + t + " ra: " + ra);

} while (ra.compareTo(BigInteger.ZERO) != 0);

System.out.println(" inv: " + t.divide(b));

return t.divide(b);

}

}

**Input**

JELLOFAD

**Output**

*Sender*

UR12CS135 - DSA Sender

Waiting for Receiver to Connect

Receiver Connected!

Input: JELLOFAD

Hashed String: 2217242401181316

Hashed Input : 6

----Global Public-Key Components----

p: 18097

p.long: 18097

q: 3

\*f: 22659

\*f: 15854

h: 15854

g: 12109

Sending p:q:g:h(m): 18097:3:12109

---------User’s Private Key---------

\*f: 0

\*f: 3

\*f: 2

x: 2

---------User’s Public Key----------

y: 5987

Sending y: 5987

--User's Per-Message Secret Number--

\*f: 1

k: 1

--------------Signing---------------

r: 1

inv: 1

s: 2

Sending h(m):s:r: 6:2:1

BUILD SUCCESSFUL (total time: 2 seconds)

*Receiver*

UR12CS135 - DSA Receiver

Connected to Sender

----------Receiving Values----------

Received p: 18097

Received q: 3

Received g: 12109

Received y: 5987

Received h(m): 6

Received s: 2

Received r: 1

--------------Verifying-------------

--i: 1 t : 1 ra: 1

--i: 1 t : 4 ra: 0

inv: 2

w: 2

u1: 0

u2: 2

v: 1

------------Testing v & r-----------

v and r are equal, hence Authenticated

BUILD SUCCESSFUL (total time: 0 seconds)

**Result**

The implementation of DSA is successfully done in java.

[Signature of the Staff In-charge]

Name of the Staff In – charge: Mr. Manoj Kumar

Date: 24 – 03 - 2015