|  |  |
| --- | --- |
| Ex.No.10  30.09.24 | **IMPLEMENTATION OF DIGITAL SIGNATURE STANDARD** |

|  |
| --- |
| **AIM:** |

To implement key generation, signature generation and verification using Digital Signature Standard.

|  |
| --- |
| **ALGORITHM:** |

**Step 1: Key Generation:**

1. **Input the prime p** from the user.
2. **Compute q**:

* Use the formula q = (p - 1) / 2.
* If q is not prime, print an error and exit the process.

3. **Generate the generator g**:

* Use the formula g = h^((p - 1) / q) mod p.
* Ensure g > 1, and retry until a valid g is found.

4. **Input the private key x**:

* Ask the user to input x such that 0 < x < q.
* If x is not in this range, print an error and exit.

5. **Compute the public key y**:

* Use the formula y = g^x mod p.

6. **Output the generated keys**:

* Print the values of p, q, g, x, and y.

**Step 2: File Signing:**

1. **Input the file path** for the file to be signed.
2. **Read the file** and convert its content to a byte array.
3. **Hash the file data** using a simple hash function:

* Multiply each byte by 31 and sum them up.

4. **Generate a random integer k**:

* Ensure 0 < k < q.

5. **Compute the signature (r, s)**:

* Calculate r = (g^k mod p) mod q.
* Compute the hash h of the file.
* Calculate s = (k^-1 \* (h + x \* r)) mod q.

6. **Output the signature**:

* Print the values of r and s.

**Step 3: Signature Verification:**

1. **Input the file path** for the file to be verified.
2. **Input the signature (r, s)**:

* Use the signature generated during the signing process.

3. **Verify the signature values**:

* Ensure that 0 < r < q and 0 < s < q. If not, return invalid.

4. **Read the file** and convert its content to a byte array.

5. **Hash the file data** to get h.

6. **Compute the verification values**:

* Calculate w = s^-1 mod q.
* Calculate u1 = (h \* w) mod q and u2 = (r \* w) mod q.
* Compute v = ((g^u1 mod p) \* (y^u2 mod p) mod p) mod q.

7. **Compare v and r**:

* If v == r, the signature is valid.
* Otherwise, the signature is invalid.

8. **Output the verification result**:

* Print whether the signature is valid or invalid.

`

|  |
| --- |
| **CODING:** |

**DigitalSignature.java:**

import java.io.IOException;

import java.math.BigInteger;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.security.SecureRandom;

import java.util.Scanner;

public class DigitalSignature {

private static boolean isPrime(int num) {

if (num < 2)

return false;

for (int i = 2; i < num / 2; i++) {

if (num % 2 == 0)

return false;

}

return true;

}

private static BigInteger primeDivisor(BigInteger num) {

for (BigInteger i = new BigInteger("101"); i.compareTo(num.subtract(BigInteger.ONE)) < 0; i = i.add(BigInteger.ONE)) {

if (num.mod(i).equals(BigInteger.ZERO) && isPrime(i.intValue())) {

return i;

}

}

return null;

}

private static BigInteger[] keyGen() {

Scanner s = new Scanner(System.in);

System.out.print("Enter The Prime (p):");

BigInteger p = s.nextBigInteger();

if (!isPrime(p.intValue())) {

System.out.println("P is not Prime.!");

return null;

}

BigInteger q = primeDivisor(p.subtract(BigInteger.ONE));

BigInteger h = new BigInteger(p.bitLength(), new SecureRandom()).mod(p.subtract(BigInteger.ONE)).add(BigInteger.ONE);

BigInteger g = h.modPow(p.subtract(BigInteger.ONE).divide(q), p);

BigInteger x = new BigInteger(q.bitLength(), new SecureRandom()).mod(q.subtract(BigInteger.ONE)).add(BigInteger.ONE);

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

System.out.print("Generated Keys: p=" + p + ",q=" + q + ",h=" + h + ",g=" + g + ",x=" + x + ",y=" + y);

return new BigInteger[] { p, q, h, g, x, y };

}

private static BigInteger[] SignGen(BigInteger p, BigInteger q, BigInteger g, BigInteger x, BigInteger HashMsg) {

BigInteger k = new BigInteger(q.bitLength(), new SecureRandom()).mod(q.subtract(BigInteger.ONE)).add(BigInteger.ONE);

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

BigInteger s = k.modInverse(q).multiply(HashMsg.add(x.multiply(r))).mod(q);

return new BigInteger[] { r, s };

}

private static void SignVerification(BigInteger p, BigInteger q, BigInteger g, BigInteger r, BigInteger s,

BigInteger y, BigInteger hashMsg) {

BigInteger w = s.modInverse(q);

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

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

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

if (v.equals(r)) {

System.out.println("Signature Verified Successfully.");

} else {

System.out.println("Signature Verification Failed.");

}

}

private static BigInteger hashMsg(String msg) {

try {

MessageDigest md = MessageDigest.getInstance("SHA-256");

byte[] hashBytes = md.digest(msg.getBytes());

return new BigInteger(hashBytes);

} catch (NoSuchAlgorithmException e) {

throw new RuntimeException(e);

}

}

private static BigInteger HashReadFile(String filePath) {

try {

String content = new String(Files.readAllBytes(Paths.get(filePath)));

return hashMsg(content);

} catch (IOException e) {

System.out.println("File Not Found.!");

return null;

}

}

public static void main(String[] args) {

BigInteger p = null, q = null, h = null, g = null, x = null, y = null;

Scanner s = new Scanner(System.in);

int choice;

do {

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

System.out.println("1->Key Generation.");

System.out.println("2->Signature Generation (File).");

System.out.println("3->Signature Verification (File).");

System.out.println("4->Signature Generation (Message).");

System.out.println("5->Signature Verification (Message).");

System.out.println("6->Exit.");

System.out.print("Enter Your Choice:");

choice = s.nextInt();

switch (choice) {

case 1:

BigInteger[] keys = keyGen();

if (keys != null) {

p = keys[0];

q = keys[1];

h = keys[2];

g = keys[3];

x = keys[4];

y = keys[5];

}

break;

case 2:

if (p == null || q == null || h == null || g == null || x == null || y == null) {

System.out.println("Keys Are Not Generated Yet.Generate Keys First.!");

continue;

}

System.out.print("Enter The File Path For Signature Generation:");

s.nextLine();

String path = s.nextLine();

BigInteger filehash = HashReadFile(path);

if (filehash != null) {

BigInteger[] signs = SignGen(p, q, g, x, filehash);

System.out.println("Generated Signature: r=" + signs[0] + ",s=" + signs[1]);

}

break;

case 3:

if (p == null || q == null || h == null || g == null || x == null || y == null) {

System.out.println("Keys Are Not Generated Yet.Generate Keys First.!");

continue;

}

System.out.print("Enter The File Path For Signature Verification:");

s.nextLine();

String vpath = s.nextLine();

BigInteger vfilehash = HashReadFile(vpath);

if (vfilehash != null) {

System.out.print("Enter r:");

BigInteger r = s.nextBigInteger();

System.out.print("Enter s:");

BigInteger S = s.nextBigInteger();

SignVerification(p, q, g, r, S, y, vfilehash);

}

break;

case 4:

if (p == null || q == null || g == null || y == null || x == null) {

System.out.println("Keys not generated yet. Please generate keys first.");

continue;

}

System.out.print("Enter the message to sign: ");

s.nextLine(); // Consume the newline

String message = s.nextLine();

BigInteger messageHash = hashMsg(message);

BigInteger[] messageSignature = SignGen(p,q,g,x,messageHash);

System.out.println("Generated Signature for Message: r=" + messageSignature[0] + ", s=" + messageSignature[1]);

break;

case 5:

if (p == null || q == null || g == null || y == null) {

System.out.println("Keys not generated yet. Please generate keys first.");

continue;

}

System.out.print("Enter the message for verification: ");

s.nextLine(); // Consume the newline

String verifyMessage = s.nextLine();

BigInteger verifyMessageHash = hashMsg(verifyMessage);

System.out.print("Enter r: ");

BigInteger rMsg = s.nextBigInteger();

System.out.print("Enter s: ");

BigInteger sMsg = s.nextBigInteger();

SignVerification(p, q, g, rMsg, sMsg, y, verifyMessageHash);

break;

case 6:

System.out.println("Exiting...");

break;

default:

System.out.println("Invalid choice. Please select a valid option.");

}

} while (choice != 6);

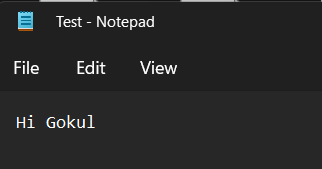
s.close();

}

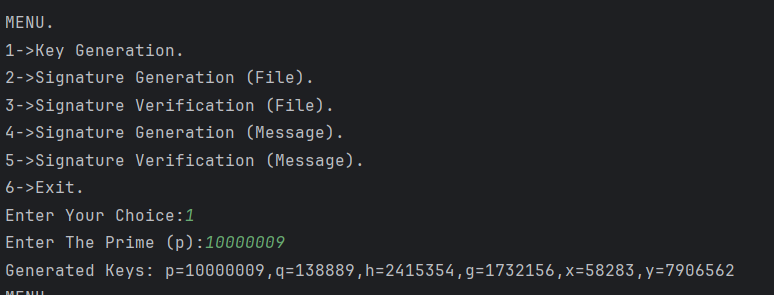
}

|  |
| --- |
| **SCREEN SHOTS:** |

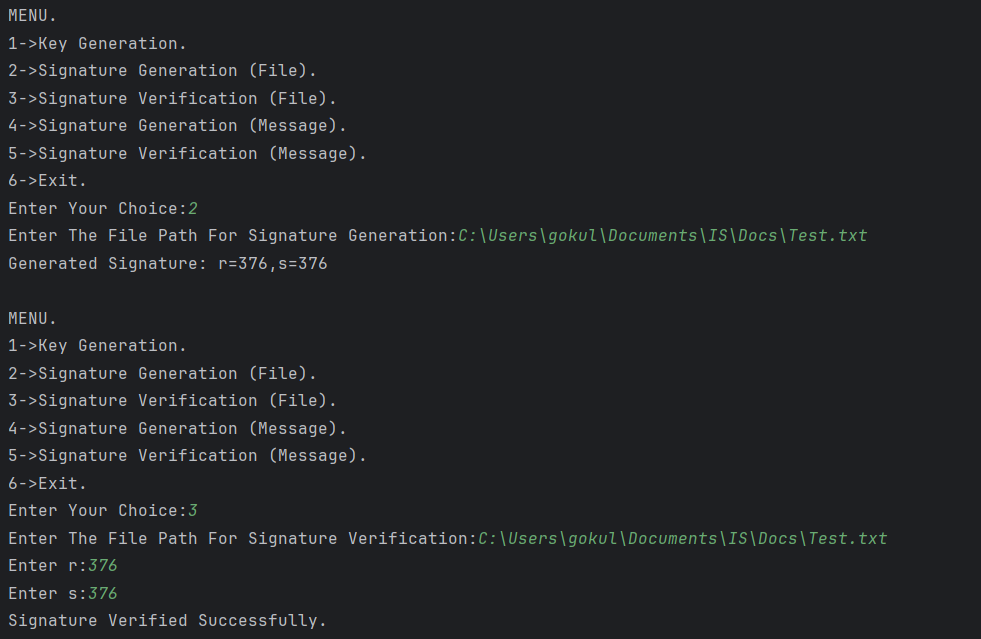
**Test.txt:**

****

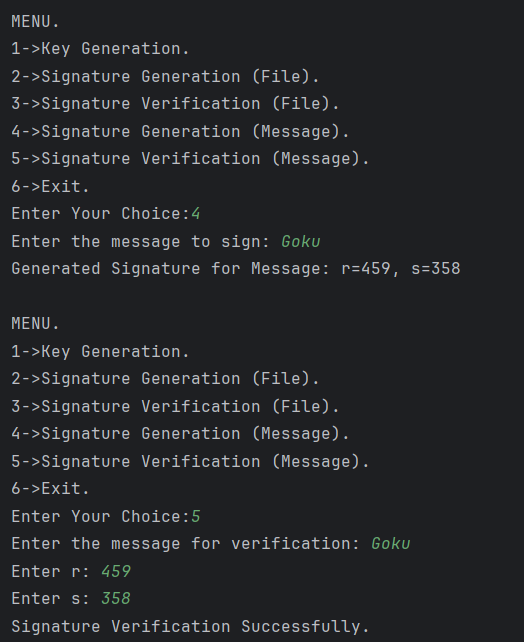
**Key Generation:**

****

**Signing And Verification of a File:**



**Signing And Verification of a Message:**

****

|  |
| --- |
| **RESULT:** |

Thus, implemented key generation, signature generation and verification using Digital Signature Standard.

|  |
| --- |
| **Evaluation** |

|  |  |  |
| --- | --- | --- |
| Parameter | Max Marks | Marks Obtained |
| Uniqueness of the Code | 50 |  |
| Completion of experiment on time | 10 |  |
| Documentation | 15 |  |
| Total | 75 |  |
| Signature of the faculty with Date |  |  |