Sno:8	**Experiment name:** The Digital Signature Standard Algorithm	**Date:**

Aim: Implement the Digital Signature Standard Algorithm.

Description: Digital Signature Standard (DSS) is a Federal Information Processing Standard(FIPS) which defines algorithms that are used to generate digital signatures with the help of Secure Hash Algorithm(SHA) for the authentication of electronic documents.The DSA algorithm works in the framework of public-key cryptosystems and is based on the algebraic properties of modular exponentiation, together with the discrete logarithm problem, which is considered to be computationally intractable. The algorithm uses a key pair consisting of a public key and a private key. The private key is used to generate a digital signature for a message, and such a signature can be verified by using the signer's corresponding public key. The digital signature provides message authentication (the receiver can verify the origin of the message), integrity (the receiver can verify that the message has not been modified since it was signed) and non-repudiation (the sender cannot falsely claim that they have not signed the message).

Algorithm:

- The DSA is based on the difficulty of computing discrete logarithms and is based on schemes originally presented by ElGamal and Schnorr.
- There are three parameters that are public and can be common to a group of users. A 160-bit prime number is chosen.
- Next, P a prime number is selected with a length between 512 and 1024 bits such that P divides (q - 1). Finally, g is chosen to be of the form h(p-1)/qmod p, where h is an integer between 1 and (q-1)with the restriction that must be greater than 12.
- Thus, the global public-key components of DSA have the same for as in the Schnorr signature scheme.With these numbers in hand, each user selects a private key and generates a public key.
- The private key must be a number from 1 to (q-1)and should be chosen randomly or pseudorandomly. The public key is calculated from the private key as y=gxmodp.
- The calculation of given is relatively straightforward. However, given the public key, it is believed to be computationally infeasible to determine ,which is the discrete logarithm of y to the base g,modp
- To create a signature, a user calculates two quantities, and that are functions of the public key components(p,q,g) , the user’s private key x , the hash code of the message H(M) , and an additional integer k that should be generated randomly or pseudo randomly and be unique for each signing.

- At the receiving end, verification is performed using the formulas shown in Figure .The receiver generates a quantity that is a function of the public key components, the sender's public key, and the hash code of the incoming message.
- If this quantity matches the component of the signature, then the signature is validated.

Source Code:

```
package java_cryptography;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.SecureRandom;
import java.security.Signature;
import java.util.Scanner;
import javax.xml.bind.DatatypeConverter;
public class Digital  Signature  GeeksforGeeks {
    private static final String
        SIGNING_ALGORITHM
        = "SHA256withRSA";
    private static final String RSA = "RSA";
    private static Scanner sc;
    public static byte[] Create_Digital_Signature(
        byte[] input,
        PrivateKey Key)
        throws Exception
    {
        Signature signature
            = Signature.getInstance(
                SIGNING_ALGORITHM);
        signature.initSign(Key);
        signature.update(input);
        return signature.sign();
    }
    public static KeyPair Generate_RSA_KeyPair()
        throws Exception
    {
        SecureRandom secureRandom
            = new SecureRandom();
        KeyPairGenerator keyPairGenerator
            = KeyPairGenerator
                .getInstance(RSA);
        keyPairGenerator
            .initialize(2048, secureRandom);
        return keyPairGenerator
            .generateKeyPair();
    }
```

```
public static boolean
    Verify_Digital_Signature(
        byte[] input,
        byte[] signatureToVerify,
        PublicKey key)
        throws Exception
    {
        Signature signature
            = Signature.getInstance(
                SIGNING  ALGORITHM);
        signature.initVerify(key);
        signature.update(input);
        return signature
            .verify(signatureToVerify);
    }

    public static void main(String args[])
        throws Exception
    {
        String input
            = "GEEKSFORGEEKS IS A"
              + " COMPUTER SCIENCE PORTAL";
        KeyPair keyPair
            = Generate_RSA_KeyPair();

        byte[] signature
            = Create_Digital_Signature(
                input.getBytes(),
                keyPair.getPrivate());

        System.out.println(
            "Signature Value:\n "
            + DatatypeConverter
                .printHexBinary(signature));

        System.out.println(
            "Verification: "
            + Verify_Digital_Signature(
                input.getBytes(),
                signature, keyPair.getPublic()));
    }
}
```

Execution Results : All test cases have succeeded

```
Signature Value:
2492035AE7782EEB75E18C1C76651384FDE30178DBE806A67DA4C884D52BF15A35CB8D1F
Verification: true
```

Result: Successfully completed Digital Signature Standard algorithm.