CSE3501 INFORMATION SECURITY ANALYSIS AND AUDIT

EXERCISE – 11

NAME: P.NITYASREE

REGNO:17MIS1007

FACULTY: Dr.PARKAVI

DATE: 08.10.2020

IMPLEMENTATION OF CRYPTOGRAPHY ALGORITHMS

Perform the following tasks

1. Symmetric Key Encryption/Decryption

a. Perform Encryption and Decryption

b. In a separate word file, show the input, output of encryption and decryption algorithm. Show that you are retrieving the Plain Text at the end.

c. Show the algorithm-pseudo code, flow-chart diagram

d. Describe the algorithm

e. Upload the word file and program file

1. **Symmetric Key Encryption/Decryption (AES)**

Java Code:

```
import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;
import java.util.Base64;

import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;

public class AES {

    private static SecretKeySpec secretKey;
    private static byte[] key;
```

```
    public static void setKey(String myKey)
    {
        MessageDigest sha = null;
        try {
            key = myKey.getBytes("UTF-8");
            sha = MessageDigest.getInstance("SHA-1");
            key = sha.digest(key);
            key = Arrays.copyOf(key, 16);
            secretKey = new SecretKeySpec(key, "AES");
        }
        catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }

    public static String encrypt(String strToEncrypt, String secret)
    {
        try
        {
            setKey(secret);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            return
Base64.getEncoder().encodeToString(cipher.doFinal(strToEncrypt.getBytes("
UTF-8")));
        }
        catch (Exception e)
        {
            System.out.println("Error while encrypting: " + e.toString());
        }
        return null;
    }

    public static String decrypt(String strToDecrypt, String secret)
    {
        try
        {
            setKey(secret);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
```

```
            cipher.init(Cipher.DECRYPT_MODE, secretKey);
            return new
String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));
        }
        catch (Exception e)
        {
            System.out.println("Error while decrypting: " + e.toString());
        }
        return null;
    }
    public static void main(String[] args)
    {
        final String secretKey = "isaa";

        String originalString = "informationsecurity";
        String encryptedString = AES.encrypt(originalString, secretKey) ;
        String decryptedString = AES.decrypt(encryptedString, secretKey) ;

        System.out.println("Original String : "+originalString);
        System.out.println("Encrypted String: "+encryptedString);
        System.out.println("Decrypted String: "+decryptedString);
    }
}
```

Output:

Result

```
$javac AES.java

$java -Xmx128M -Xms16M AES
Original String : informationsecurity
Encrypted String: YxLSXt2PqX6GUO33fMzs8tjuLHGnwsi9VtaiW+T6UuQ=
Decrypted String: informationsecurity
```

Pseudo Code:
Encryption

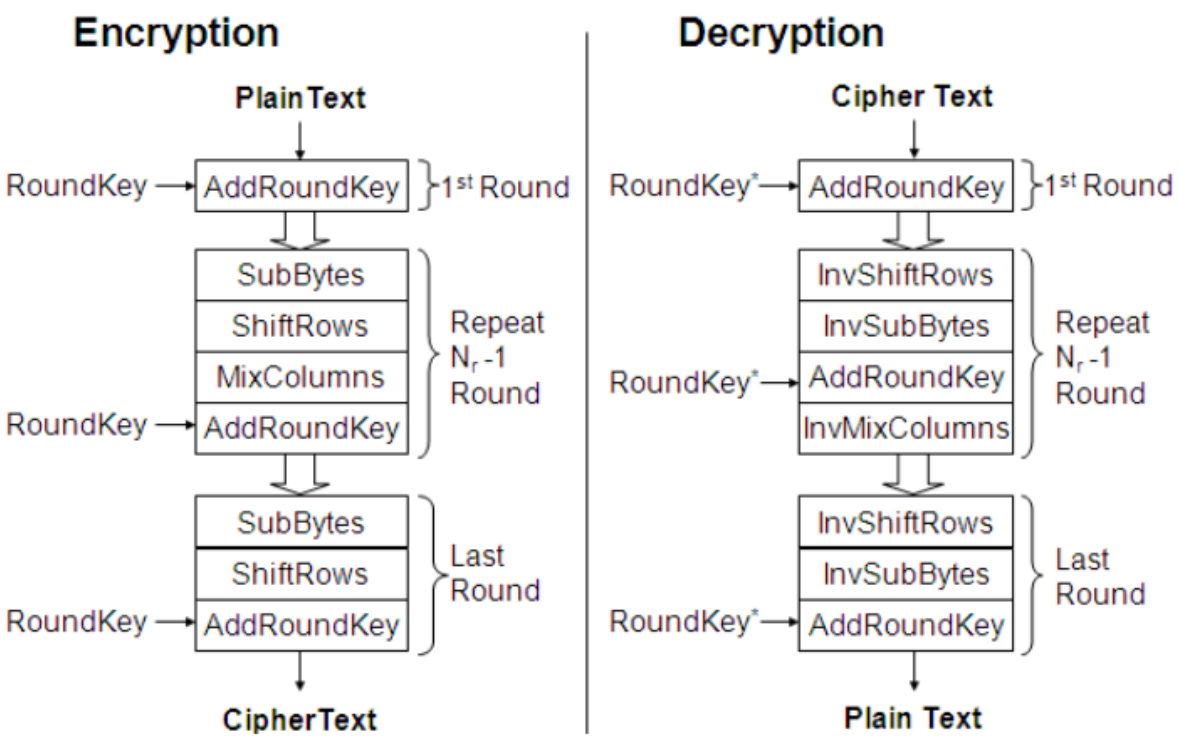
Begin
Byte state[4,nb]
State=in

```
AddRoundKey(state,w[0,nb-1])
for round =1 step 1 to nr-1
subbytes(state)
shiftrows(state)
Mixcolumns(state)
AddRoundKey(state,w[round*nb,(round+1)*nb-1])
End for
Subbytes(state)
Shiftrows(state)
AddRoundKey(state,w[nr*nb,(nr+1)*nb-1])
Out=state
End
```

Decrption

```
Begin
Byte state[4,nb]
State=in
AddRoundKey(state,w[nr*nb,(nr+1)*nb-1])
for round =1 step 1 to nr-1
Invshiftrows(state)
Invsubbytes(state)
Mixcolumns(state)
AddRoundKey(state,w[round*nb,(round+1)*nb-1])
End for
InvShiftrows(state)
InvSubbytes(state)
AddRoundKey(state,w[0,nb-1])
Out=state
End
```

Flow Chart:



AES Algorithm:
AES is an iterative rather than Feistel cipher. It is based on 'substitution–permutation network'. It comprises of a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations).

Interestingly, AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix −

Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.

The schematic of AES structure is given in the following illustration −

AES Structure
Encryption Process
Here, we restrict to description of a typical round of AES encryption. Each round comprise of four sub-processes. The first round process is depicted below −

First Round Process
Byte Substitution (SubBytes)
The 16 input bytes are substituted by looking up a fixed table (S-box) given in design. The result is in a matrix of four rows and four columns.

Shiftrows
Each of the four rows of the matrix is shifted to the left. Any entries that 'fall off' are re-inserted on the right side of row. Shift is carried out as follows −

First row is not shifted.

Second row is shifted one (byte) position to the left.

Third row is shifted two positions to the left.

Fourth row is shifted three positions to the left.

The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.

MixColumns
Each column of four bytes is now transformed using a special mathematical function. This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column. The result is another new matrix consisting of 16 new bytes. It should be noted that this step is not performed in the last round.

Addroundkey
The 16 bytes of the matrix are now considered as 128 bits and are XORed to the 128 bits of the round key. If this is the last round then the output is the ciphertext. Otherwise, the resulting 128 bits are interpreted as 16 bytes and we begin another similar round.

Decryption Process
The process of decryption of an AES ciphertext is similar to the encryption process in the reverse order. Each round consists of the four processes conducted in the reverse order −

Add round key
Mix columns
Shift rows

Byte substitution
Since sub-processes in each round are in reverse manner, unlike for a Feistel Cipher, the encryption and decryption algorithms needs to be separately implemented, although they are very closely related.

2. Asymmetric Key Encryption/Decryption

a. Perform Encryption and Decryption

b. In a separate word file, show the input, output of encryption and decryption algorithm. Show that you are retrieving the Plain Text at the end.

c. Show the algorithm-pseudo code, flow-chart diagram

d. Describe the algorithm

e. Upload the word file and program file

2. **Asymmetric Key Encryption/Decryption (RSA)**

Java Code:

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.util.Base64;

import javax.crypto.Cipher;

public class RSA_Encryption
{
   static String plainText = "Information security 123";

   public static void main(String[] args) throws Exception
   {
      KeyPairGenerator keyPairGenerator =
KeyPairGenerator.getInstance("RSA");
      keyPairGenerator.initialize(4096);
      KeyPair keyPair = keyPairGenerator.generateKeyPair();
      PublicKey publicKey = keyPair.getPublic();
      PrivateKey privateKey = keyPair.getPrivate();
      System.out.println("Original Text  : "+plainText);
      byte[] cipherTextArray = encrypt(plainText, publicKey);
```

```
        String encryptedText =
Base64.getEncoder().encodeToString(cipherTextArray);
        System.out.println("Encrypted Text : "+encryptedText);
        String decryptedText = decrypt(cipherTextArray, privateKey);
        System.out.println("DeCrypted Text : "+decryptedText);
    }

    public static byte[] encrypt (String plainText,PublicKey publicKey ) throws
Exception
    {
        Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPWITHSHA-
512ANDMGF1PADDING");
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);
        byte[] cipherText = cipher.doFinal(plainText.getBytes()) ;
        return cipherText;
    }

    public static String decrypt (byte[] cipherTextArray, PrivateKey privateKey)
throws Exception
    {
        Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPWITHSHA-
512ANDMGF1PADDING");
        cipher.init(Cipher.DECRYPT_MODE, privateKey);
        byte[] decryptedTextArray = cipher.doFinal(cipherTextArray);
        return new String(decryptedTextArray);
    }

}
```

Output:

```
$javac RSA_Encryption.java

$java -Xmx128M -Xms16M RSA_Encryption
Original Text  : Information security 123
Encrypted Text : i1gQtL+BWQVwbW23NSHQpOFdol/wVqbD7VSPS6jB3tXsbl0Xz5heYxm1cWmHTA216Sapxfllw5eDNe+WJEXR6BEwbe
DeCrypted Text : Information security 123
```

Pseudo Code:

Encryption

```
Begin
Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPWITHSHA-
512ANDMGF1PADDING");
        cipher.init(Cipher.DECRYPT_MODE, privateKey);
        byte[] decryptedTextArray = cipher.doFinal(cipherTextArray);
        return new String(decryptedTextArray);
```

End

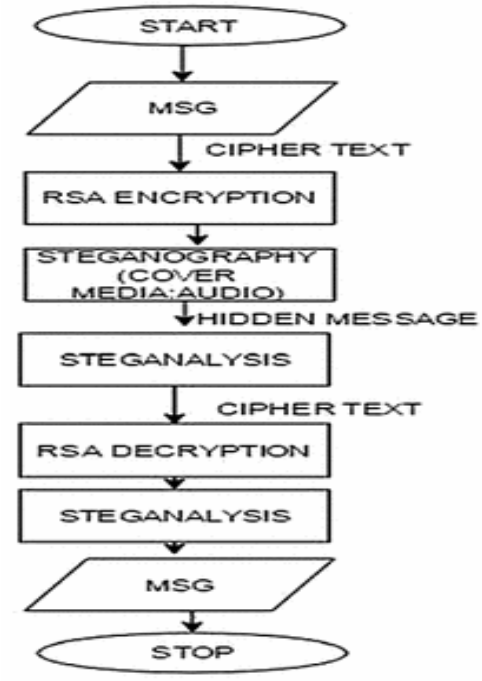
Decryption

Begin
Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPWITHSHA-512ANDMGF1PADDING");
cipher.init(Cipher.DECRYPT_MODE, privateKey);
byte[] decryptedTextArray = cipher.doFinal(cipherTextArray);
return new String(decryptedTextArray);
End

Key Generation:

Begin
KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
keyPairGenerator.initialize(4096);
KeyPair keyPair = keyPairGenerator.generateKeyPair();
PublicKey publicKey = keyPair.getPublic();
PrivateKey privateKey = keyPair.getPrivate();
End

Flow chart:



Algorithm:

The RSA algorithm is an asymmetric cryptography algorithm; this means that it uses a public key and a private key (i.e two different, mathematically linked keys). As their names suggest, a public key is shared publicly, while a private key is secret and must not be shared with anyone.

The RSA algorithm is named after those who invented it in 1978: Ron Rivest, Adi Shamir, and Leonard Adleman.

3. Hashing

a. Perform Hashing on the input (text & file)

b. Show the Message and its Hash.

c. Introduce a very small modification in the input. Show the change in the hash code.

d. Show the algorithm-pseudo code, flow-chart diagram

e. Describe the algorithm f. Upload the word file and program file

3. Hashing

Java Code:

```
public class hashing
{
   public static void main(String[] args)
   {
      String s1 = "Information security";
      System.out.println("Hash Code for "+s1+" "+s1.hashCode() );
      String s2 = "Information securit";
      System.out.println("Hash Code for "+s2+" "+ s2.hashCode() );
   }
}
```

Output:

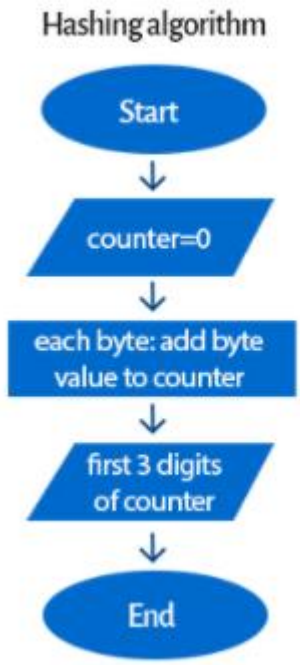
```
$javac hashing.java

$java -Xmx128M -Xms16M hashing
Hash Code for Information security -2052970540
Hash Code for Information securit 1042153797
```

Pseudo Code:

Begin
String s1;
s1.hashcode()
end

Flow chart:



Algorithm:
A hashing algorithm is a cryptographic hash function. It is a mathematical algorithm that maps data of arbitrary size to a hash of a fixed size.

A hash function algorithm is designed to be a one-way function, infeasible to invert. However, in recent years several hashing algorithms have been compromised. This happened to MD5, for example — a widely known hash function designed to be a cryptographic hash function, which is now so easy to reverse — that we could only use for verifying data against unintentional corruption.

It's easy to figure out what the ideal cryptographic hash function should be like:

It should be fast to compute the hash value for any kind of data;
It should be impossible to regenerate a message from its hash value (brute force attack as the only option);
It should be infeasible to find two messages with the same hash (a collision);

Every change to a message, even the smallest one, should change the hash value. It should be completely different. It's called the avalanche effect

4. Digital Signature

a. Affix digital signature to your input (Text & File)

b. In a separate word file, show the input, output of double time encryption and double time decryption algorithm. Show that you are retrieving the Plain Text at the end.

c. Show the algorithm-pseudo code, flow-chart diagram

d. Describe the algorithm

e. Upload the word file and program file

4.Digital Signature

Java Code:

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.Signature;
import java.util.Scanner;

public class CreatingDigitalSignature {
   public static void main(String args[]) throws Exception {
      String msg = "Information security";
      KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("DSA");
      keyPairGen.initialize(2048);
      KeyPair pair = keyPairGen.generateKeyPair();
      PrivateKey privKey = pair.getPrivate();
      Signature sign = Signature.getInstance("SHA256withDSA");
      sign.initSign(privKey);
      byte[] bytes = "msg".getBytes();
      sign.update(bytes);
      byte[] signature = sign.sign();
      System.out.println("Digital signature for "+msg+""+new String(signature,
"UTF8"));
   }
```

}

Output:

```
$javac CreatingDigitalSignature.java
$java -Xmx128M -Xms16M CreatingDigitalSignature
Digital signature for Information security0;□□<□#X□□Q□□6□□¹□[2□6□□#y□L&□□;□I□AM□□m>□T□□□□□3□□
```

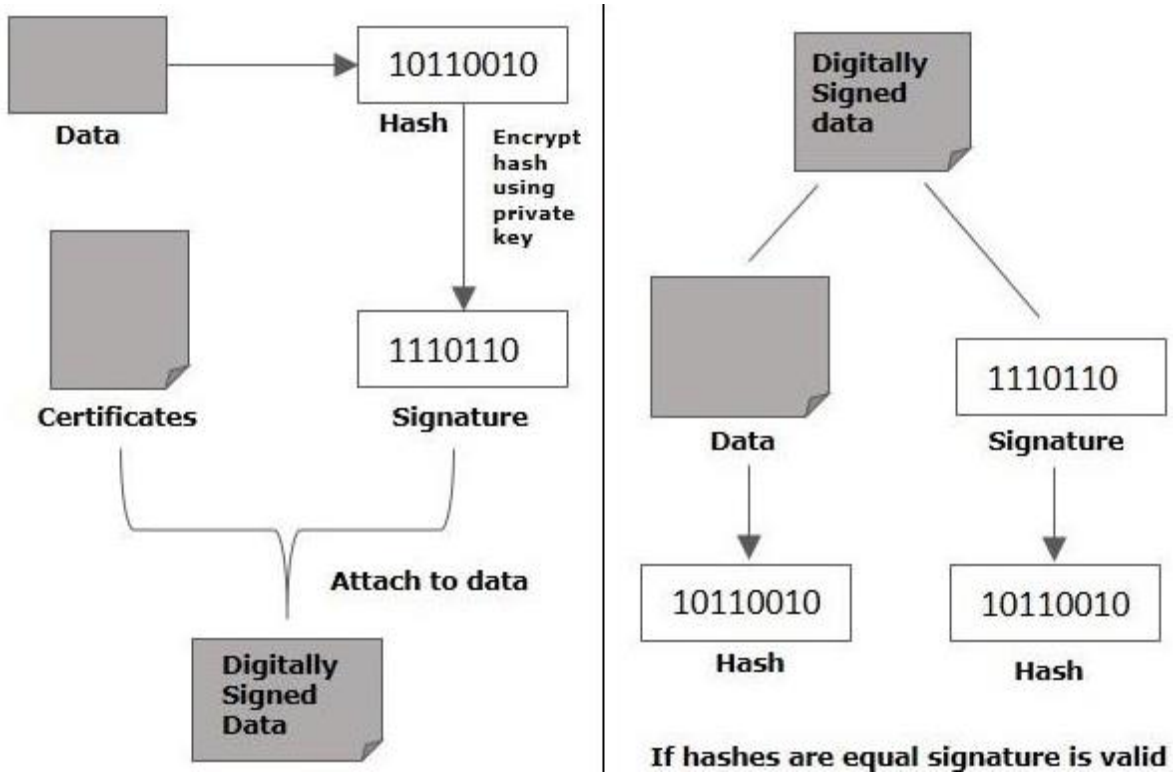
Pseudo Code:

Begin

```
KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("DSA");
    keyPairGen.initialize(2048);
    KeyPair pair = keyPairGen.generateKeyPair();
    PrivateKey privKey = pair.getPrivate();
    Signature sign = Signature.getInstance("SHA256withDSA");
    sign.initSign(privKey);
    byte[] bytes = "msg".getBytes();
    sign.update(bytes);
    byte[] signature = sign.sign();
```

End

Flow chart:



Algorithm:

A digital signature algorithm (DSA) refers to a standard for digital signatures. It was introduced in 1991 by the National Institute of Standards and Technology (NIST) as a better method of creating digital signatures. Along

with RSA, DSA is considered one of the most preferred digital signature algorithms used today.
Digital signatures allow us to verify the author, date and time of signatures, authenticate the message contents. It also includes authentication function for additional capabilities.
Step 1: Create a KeyPairGenerator object
The KeyPairGenerator class provides getInstance() method which accepts a String variable representing the required key-generating algorithm and returns a KeyPairGenerator object that generates keys.
Step 2: Initialize the KeyPairGenerator object
The KeyPairGenerator class provides a method named initialize() this method is used to initialize the key pair generator. This method accepts an integer value representing the key size.
Step 3: Generate the KeyPairGenerator
You can generate the KeyPair using the generateKeyPair() method.
Step 4: Get the private key from the pair
You can get the private key from the generated KeyPair object using the getPrivate() method.
Step 5: Create a signature object
The getInstance() method of the Signature class accepts a string parameter representing required signature algorithm and returns the respective Signature object.
Step 6: Initialize the Signature object
The initSign() method of the Signature class accepts a PrivateKey object and initializes the current Signature object.
Step 7: Add data to the Signature object
The update() method of the Signature class accepts a byte array representing the data to be signed or verified and updates the current object with the data given.
Step 8: Calculate the Signature
The sign() method of the Signature class returns the signature bytes of the updated data.

5. Ensure the presence of CIA (Confidentiality, Integrity, Authentication) triad

a. Perform Digital Signature and Hashing in a single program (Text 7 File input)

b. In a separate word file, show the input, output of your complete algorithm. Show that you are retrieving the Plain Text at the end and proving the hash code match also.

c. Show the algorithm-pseudo code, flow-chart diagram

d. Describe the algorithm

5.Ensure the presence of CIA (Confidentiality, Integrity, Authentication) triad

Java Code:

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.Signature;

public class SignatureVerification {
   public static void main(String args[]) throws Exception{
      KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("DSA");
      keyPairGen.initialize(2048);
      KeyPair pair = keyPairGen.generateKeyPair();
      PrivateKey privKey = pair.getPrivate();
      Signature sign = Signature.getInstance("SHA256withDSA");
      sign.initSign(privKey);
      byte[] bytes = "Information Security".getBytes();
      sign.update(bytes);
      byte[] signature = sign.sign();
      sign.initVerify(pair.getPublic());
      sign.update(bytes);
      boolean bool = sign.verify(signature);
      if(bool) {
         System.out.println("Signature verified");
      }else{
         System.out.println("Signature failed");
      }
   }
}
```

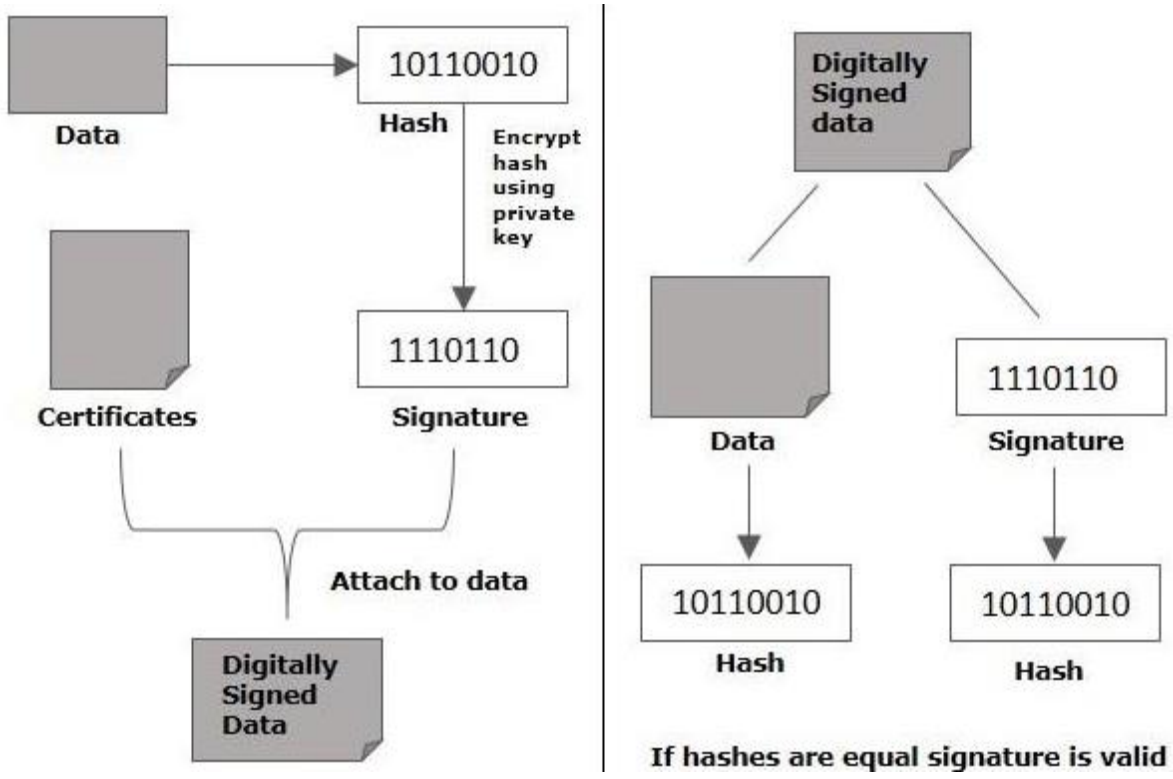
Output:

```
$javac SignatureVerification.java
$java -Xmx128M -Xms16M SignatureVerification
Signature verified
```

Pseudo Code:

```
Begin
KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("DSA");
    keyPairGen.initialize(2048);
    KeyPair pair = keyPairGen.generateKeyPair();
    PrivateKey privKey = pair.getPrivate();
    Signature sign = Signature.getInstance("SHA256withDSA");
    sign.initSign(privKey);
    byte[] bytes = "Information Security".getBytes();
    sign.update(bytes);
    byte[] signature = sign.sign();
    sign.initVerify(pair.getPublic());
    sign.update(bytes);
    boolean bool = sign.verify(signature);
End
```

Flow Chart:



Algorithm:

Step 1: Create a KeyPairGenerator object

The KeyPairGenerator class provides getInstance() method which accepts a String variable representing the required key-generating algorithm and returns a KeyPairGenerator object that generates keys.

Step 2: Initialize the KeyPairGenerator object

The KeyPairGenerator class provides a method named initialize() method. This method is used to initialize the key pair generator. This method accepts an integer value representing the key size.

Step 3: Generate the KeyPairGenerator

You can generate the KeyPair using the generateKeyPair() method.

Step 4: Get the private key from the pair

You can get the private key from the generated KeyPair object using the getPrivate() method.

Step 5: Create a signature object

The getInstance() method of the Signature class accepts a string parameter representing required signature algorithm and returns the respective Signature object.

Step 6: Initialize the Signature object

The initSign() method of the Signature class accepts a PrivateKey object and initializes the current Signature object.

Step 7: Add data to the Signature object

The update() method of the Signature class accepts a byte array representing the data to be signed or verified and updates the current object with the data given.

Step 8: Calculate the Signature

The sign() method of the Signature class returns the signature bytes of the updated data.

Step 9: Initialize the signature object for verification

To verify a Signature object you need to initialize it first using the initVerify() method it method accepts a PublicKey object.

Step 10: Update the data to be verified

Update the initialized (for verification) object with the data the data to be verified using the update method

Step 11: Verify the Signature

The verify() method of the Signature class accepts another signature object and verifies it with the current one. If a match occurs, it returns true else it returns false.