Experiment No.2
To Implement the concept of authentication of sender using Digital Signature
Date of Performance:17/8/23
Date of Submission:19/8/23

AIM: To Implement the concept of authentication of sender using Digital Signature

Objective: To develop a program to create a digital signature for the sample input and verify it

Theory:

A digital signature is a mathematical technique used to validate the authenticity and integrity of a message, software or digital document. It's the digital equivalent of a handwritten signature or stamped seal, but it offers far more inherent security. A digital signature is intended to solve the problem of tampering and impersonation in digital communications.

Digital signatures can provide evidence of origin, identity and status of electronic documents, transactions or digital messages. Signers can also use them to acknowledge informed consent.

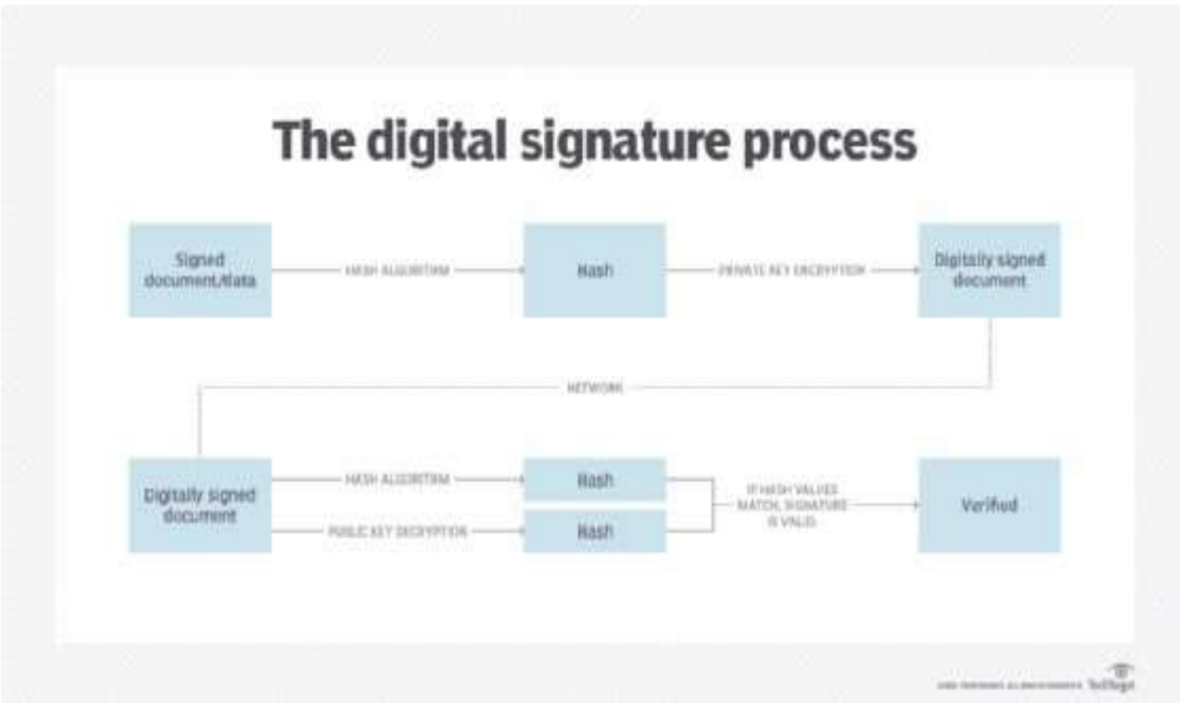

Fig. 2.1 Digital Signature Process

To create a digital signature, signing software, such as an email program, is used to provide a one-way hash of the electronic data to be signed.

A hash is a fixed-length string of letters and numbers generated by an algorithm. The digital signature creator's private key is then used to encrypt the hash. The encrypted hash -- along with other information, such as the hashing algorithm -- is the digital signature.

The reason for encrypting the hash instead of the entire message or document is a hash function can convert an arbitrary input into a fixed-length value, which is usually much shorter. This saves time as hashing is much faster than signing.

The value of a hash is unique to the hashed data. Any change in the data, even a change in a single character, will result in a different value. This attribute enables others to use the signer's public key to decrypt the hash to validate the integrity of the data.

If the decrypted hash matches a second computed hash of the same data, it proves that the data hasn't changed since it was signed. If the two hashes don't match, the data has either been tampered with in some way and is compromised or the signature was created with a private key that doesn't correspond to the public key presented by the signer -- an issue with authentication.

Role of Digital Signature in Blockchain: Digital signatures are a fundamental building block in blockchains, used mainly to authenticate transactions. When users submit transactions, they must prove to every node in the system that they are authorized to spend those funds, while preventing other users from spending them. Every node in the network will verify the submitted transaction and check all other nodes' work to agree on a correct state.

Process:

Step 1. Create a sample information on which digital signature is to be obtained

Step 2. Generate Private-public key pairs for the sender and recipients

Step 3. Create Hash of the sample information using SHA-256 algorithm

Step 4. Encrypt the Hash using private key of the sender to obtain Digital Signature

Step 5. Append Hash to the original sample information

Step 6. Encrypt the information obtained from step 5 using public key of recipient

Step 7. Send the information (Cipher text) obtained from step 6 to the recipient

Step 8. Decrypt the Cipher text using private key of the recipient

Step 9 . Decrypt Digital signature using public key of the sender to obtain original hash as obtained by step 3

Step 10. Recipient perform hashing of the decrypted sample information in step 8 using SHA-256 to obtain latest hash

Step 11 . The latest hash obtained is then compared with the hash obtained in step 9 to authenticate the sender

code :

```
package javaapplication1;

// Imports  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 JavaApplication1  {  //

Signing Algorithm  private static final

String

                    SIGNING_ALGORITHM
```

```
        =  "SHA256withRSA";

    private static final String RSA = "RSA";

    //private static Scanner sc;

     // Function to implement Digital signature

     // using SHA256 and RSA algorithm

    // by passing private key.

     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();

     }

     // Generating the asymmetric key pair

     // using SecureRandom class
```

```
// functions and RSA algorithm.
public static KeyPair Generate_RSA_KeyPair()
        throws Exception
{
        SecureRandom secureRandom
                = new SecureRandom();
        KeyPairGenerator keyPairGenerator
                = KeyPairGenerator
                        .getInstance(RSA);
        keyPairGenerator
                .initialize(
                        2048    , secureRandom);
        return keyPairGenerator
                .generateKeyPair();
}
// Function for Verification of the //
digital signature by using the public key
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);
    }
     // Driver Code  public static void
    main(String args[])  throws
    Exception
    {

             String input
                   =  "VCET"
                   +  "BlockChain";
```

```
String input1 = "mumbai";
        KeyPair keyPair
            = Generate_RSA_KeyPair();

        // Function Call
        byte[] signature
            = Create_Digital_Signature(
                input.getBytes(),
                keyPair.getPrivate());
System.out.println("The original message is " + input +"\n");

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

        System.out.println(
            "Verification: "
            + Verify_Digital_Signature(
                input1.getBytes(),
```

```
                    signature, keyPair.getPublic()));
    }
}
```

Output:

The original message is VCETBlockChain

Signature Value:

84 EB7C1C5AAE83B3D94E24E6613F81E559C75C4172A5690E9742E504F72C4E6132A4

FBA1E09DA57416932564249FA18F846383238C65339F0887DB013F6EEE0F1911AA98D
BC268D31A88D5E4F31500E15B8A6ADCDAD798DB6B2B57A011A095D37D514F42D1
B99638E9C44DEA8DFBA9FF173BF48200995D4BCBB52D2383609DAB41BC8A063D3
A78444EBB2B44C1E26BAA257BA01F93E837DD7EE1045A952F2EEFEC6BCCF288F51
054209D2F21336CB31598
FB93D54C3304B21B5B28C294AFB555E9B268D01A1CEF7B5
156726C8F8
B41E8E468AB761B96A2B164B637FF405B7130F3DE84032DCAB9CF22899
B3EE32DD840B8FEDB61210BC167A4BC4326BFCAFBE0

Verification: false

BUILD SUCCESSFUL (total time: 0 seconds)

Conclusion: In conclusion, the utilization of digital signatures for sender authentication offers a robust approach to validate the sender's identity and uphold data integrity during transmission. Through the implementation of digital signatures using asymmetric encryption, this method ensures both the validity of the sender's data and maintains a high level of security. This process provides a reliable means of verifying authenticity without compromising on safeguarding sensitive information