**Blockchain Technology Lab**

**Lab – 4**

**Aim : Simulating Digital Signature Generation and Verification.**

**Theory:**

**Process of Signing a Message**

The process of signing a message involves the following steps:

1. **Hashing the Message**: The original message is hashed using SHA-256 to create a message digest.
2. **Concatenating the Message and Digest**: The original message and the hash (digest) are concatenated together, separated by a delimiter
3. **Signing the Data**: The concatenated string is then encrypted using the sender’s private RSA key. This encryption serves as the digital signature, which can be verified by anyone possessing the sender’s public key.

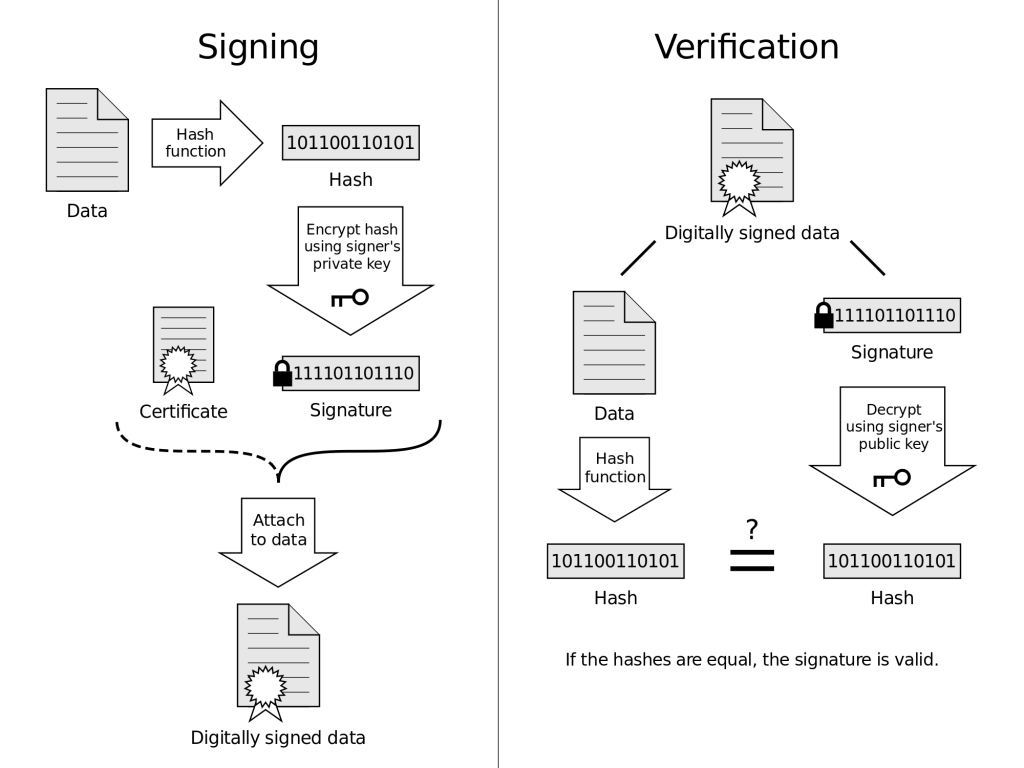
**Saving the Signature**

The signed data (concatenated message and digest) along with the digital signature is saved to a binary file. This binary file contains both the original data and the signature, which is used later for verification.

**Verification Process**

The verification of the digital signature follows these steps:

1. **Reading the Signature File**: The binary file containing the concatenated message, digest, and digital signature is read.
2. **Decrypting the Signature**: The signature is decrypted using the sender’s public key to retrieve the concatenated string (original message + hash).
3. **Splitting the Concatenated Data**: The concatenated data is split into the original message and the original hash.
4. **Re-Hashing the Message**: The original message is hashed again using the same SHA-256 algorithm.
5. **Comparing Hashes**: The newly computed hash is compared with the original hash. If they match, the message is verified as authentic and unaltered; otherwise, the verification fails, indicating that the message might have been tampered with.



**Code:**

*from* cryptography.hazmat.primitives.asymmetric *import* rsa, padding

*from* cryptography.hazmat.primitives *import* hashes

*import* hashlib

*import* os

*import* uuid

*def* generate\_keys():

    private\_key = rsa.generate\_private\_key(

        public\_exponent=65537,

        key\_size=2048

    )

    public\_key = private\_key.public\_key()

*return* public\_key, private\_key

*def* sign\_data(data, private\_key):

    message\_digest = hashlib.sha256(data.encode()).hexdigest()

    concatenated\_data = data + message\_digest

    signature = private\_key.sign(

        concatenated\_data.encode('utf-8'),

        padding.PSS(

            mgf=padding.MGF1(hashes.SHA256()),

            salt\_length=padding.PSS.MAX\_LENGTH

        ),

        hashes.SHA256()

    )

    file\_name = *f*"{str(uuid.uuid4().fields[-1])[:8]}.bin"

*with* open(file\_name, 'wb') *as* signature\_file:

        signature\_file.write(concatenated\_data.encode('utf-8') + signature)

    print(*f*"Signature saved to {file\_name}")

*def* verify\_data(signature\_file\_path, public\_key):

*try*:

*with* open(signature\_file\_path, 'rb') *as* signature\_file:

            content = signature\_file.read()

        key\_size = 2048 // 8  *# For a 2048-bit key, the signature size will be 256 bytes*

        concatenated\_data = content[:-key\_size]

        digest = content[-key\_size:]

        public\_key.verify(

            digest,

            concatenated\_data,

            padding.PSS(

                mgf=padding.MGF1(hashes.SHA256()),

                salt\_length=padding.PSS.MAX\_LENGTH

            ),

            hashes.SHA256()

        )

        print("Verification successful: The signature is valid.")

*except* Exception *as* e:

        print(*f*"Verification failed: {str(e)}")

*if* \_\_name\_\_ == "\_\_main\_\_":

    public\_key, private\_key = generate\_keys()

*while* True:

        print("\nMenu:")

        print("1. Sign Data")

        print("2. Verify Data")

        print("3. Exit")

        choice = input("Enter your choice: ")

*if* choice == '1':

            message = input("Enter the message to sign: ")

            sign\_data(message, private\_key)

*elif* choice == '2':

            signature\_file\_path = input("Enter the path of the signature file: ")

*if* *not* os.path.exists(signature\_file\_path):

                print("File does not exist.")

*continue*

            verify\_data(signature\_file\_path, public\_key)

*elif* choice == '3':

            print("Exiting.")

*break*

*else*:

            print("Invalid choice.")

**Output:**

**A computer screen with white text

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**