**Final Year B.Tech. (CSE) – VII [ 2024-25]**

**6CS451: Cryptography and Network Security Lab (C&NS Lab)**

**Date: 08/10/2024**

**Assignment 10**

**PRN:** 21510017  **Name:** Onkar Anand Yemul

**10. Implement the SIGNATURE SCHEME – Digital Signature Standard**

**Ans:**

To implement the **Digital Signature Standard (DSS)**, we need to understand its process, which involves the **Digital Signature Algorithm (DSA)**. The DSS is a Federal Information Processing Standard (FIPS) for digital signatures, and it involves three main stages:

1. **Key Generation**: Generate a public and private key pair.
2. **Signature Generation**: Use the private key to sign a message.
3. **Signature Verification**: Use the public key to verify the authenticity of the message.

**Overview of the DSA Algorithm:**

* DSA involves a pair of keys: **private key** (used for signing) and **public key** (used for verification).
* The signature is generated by applying a **hashing algorithm** (such as SHA-1 or SHA-256) to the message, which is then signed using the private key.
* The signature is verified using the public key.

**Python Implementation of the DSA (Digital Signature Algorithm) using Python's *cryptography* library for cryptographic operations:**- Using a library here simplifies the task

**Steps:**

1. **Key Generation**: Generate the DSA keys.

* A DSA private key is generated using dsa.generate\_private\_key(key\_size=2048). The key size can be adjusted (1024, 2048, or 3072 bits), but 2048 is commonly used.
* The corresponding public key is derived from the private key and returned.

1. **Signing**: Use the private key to sign a message.

* The private\_key.sign function is used to sign the message using a specified hash function (SHA-256 here).
* This produces a signature that can later be verified.

1. **Verification**: Use the public key to verify the signature.

* The public\_key.verify function checks whether the signature matches the original message using the public key.
* If the verification fails, an InvalidSignature exception is raised, indicating that the signature is not valid.

**Python Code: DSA-based Digital Signature Implementation:**

from cryptography.hazmat.backends import default\_backend

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

from cryptography.hazmat.primitives import hashes, serialization

from cryptography.exceptions import InvalidSignature

import hashlib

# Function to generate RSA private and public keys

def generate\_keys():

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

        public\_exponent=65537,

        key\_size=2048,

        backend=default\_backend()

    )

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

    return private\_key, public\_key

# Function to sign a message and print its hash

def sign\_message(private\_key, message):

    # Hash the message using SHA-256

    message\_hash = hashlib.sha256(message.encode()).hexdigest()

    # Sign the hashed message using RSA private key

    signature = private\_key.sign(

        bytes.fromhex(message\_hash),  # Convert the hex hash to bytes

        padding.PSS(

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

            salt\_length=padding.PSS.MAX\_LENGTH

        ),

        hashes.SHA256()

    )

    print(f"Hash of the message: {message\_hash}")  # Print the message hash

    return signature, message\_hash

# Function to verify the signature using the provided hash

def verify\_signature(public\_key, message\_hash, signature):

    try:

        # Verify the signature using RSA public key

        public\_key.verify(

            signature,

            bytes.fromhex(message\_hash),  # Convert the hex hash back to bytes

            padding.PSS(

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

                salt\_length=padding.PSS.MAX\_LENGTH

            ),

            hashes.SHA256()

        )

        return True

    except InvalidSignature:

        return False

# Function to save keys to files

def save\_keys\_to\_file(private\_key, public\_key):

    # Save private key

    with open("private\_key.pem", "wb") as private\_file:

        private\_file.write(

            private\_key.private\_bytes(

                encoding=serialization.Encoding.PEM,

                format=serialization.PrivateFormat.PKCS8,

                encryption\_algorithm=serialization.NoEncryption()

            )

        )

    # Save public key

    with open("public\_key.pem", "wb") as public\_file:

        public\_file.write(

            public\_key.public\_bytes(

                encoding=serialization.Encoding.PEM,

                format=serialization.PublicFormat.SubjectPublicKeyInfo

            )

        )

    print("Keys saved to files: private\_key.pem and public\_key.pem")

# Menu for the digital signature system

def menu():

    private\_key, public\_key = None, None

    signature = None

    message\_hash = None

    while True:

        print("\n===== Digital Signature System =====")

        print("1. Generate RSA Keys")

        print("2. Sign a Message")

        print("3. Verify Signature")

        print("4. Save Keys to Files")

        print("5. Exit")

        choice = input("Enter your choice (1/2/3/4/5): ")

        if choice == '1':

            # Generate RSA private and public keys

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

            print("\nRSA Keys Generated!")

        elif choice == '2':

            # Sign a message

            if private\_key is None:

                print("You need to generate RSA keys first.")

            else:

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

                signature, message\_hash = sign\_message(private\_key, message)

                print("\nMessage signed successfully!")

        elif choice == '3':

            # Verify the signature

            if public\_key is None or message\_hash is None or signature is None:

                print("You need to sign a message first.")

            else:

                input\_hash = input("Enter the hash of the message to verify: ")

                verification\_result = verify\_signature(public\_key, input\_hash, signature)

                if verification\_result:

                    print("\nSignature verified successfully! The message is authentic.")

                else:

                    print("\nSignature verification failed! The message is not authentic.")

        elif choice == '4':

            # Save RSA keys to files

            if private\_key is None or public\_key is None:

                print("You need to generate RSA keys first.")

            else:

                save\_keys\_to\_file(private\_key, public\_key)

        elif choice == '5':

            print("Exiting the program...")

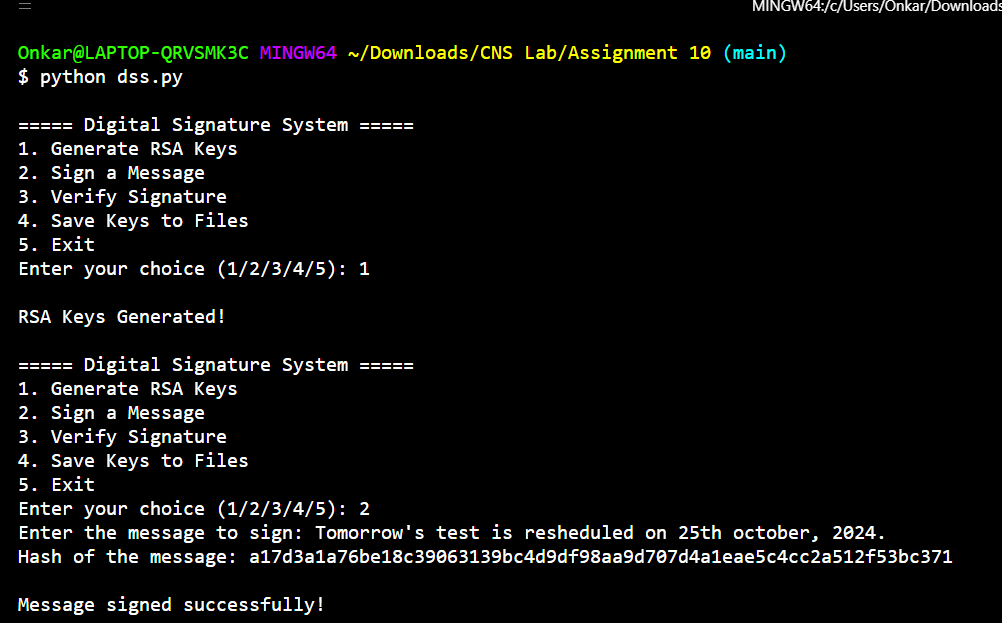
            break

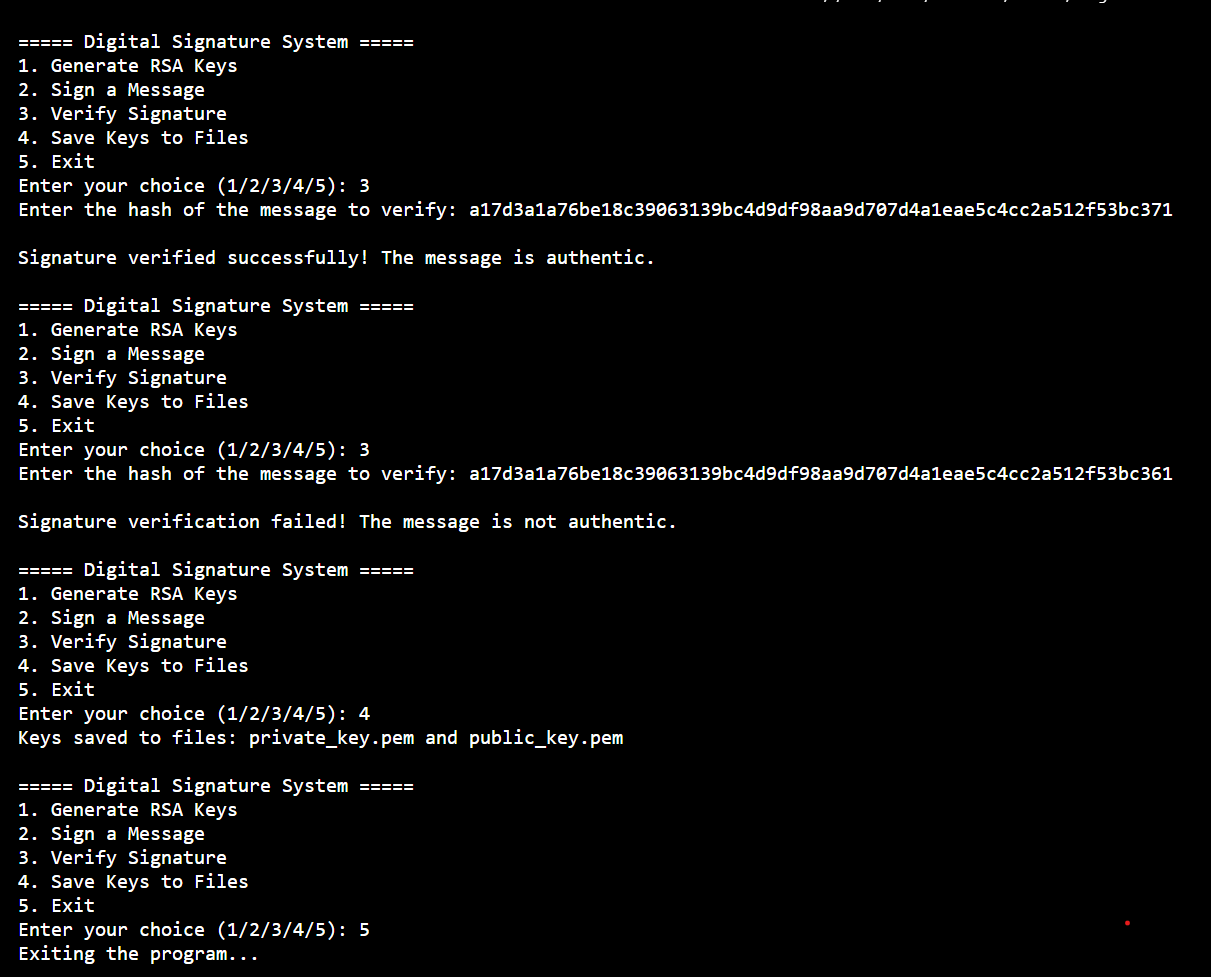
        else:

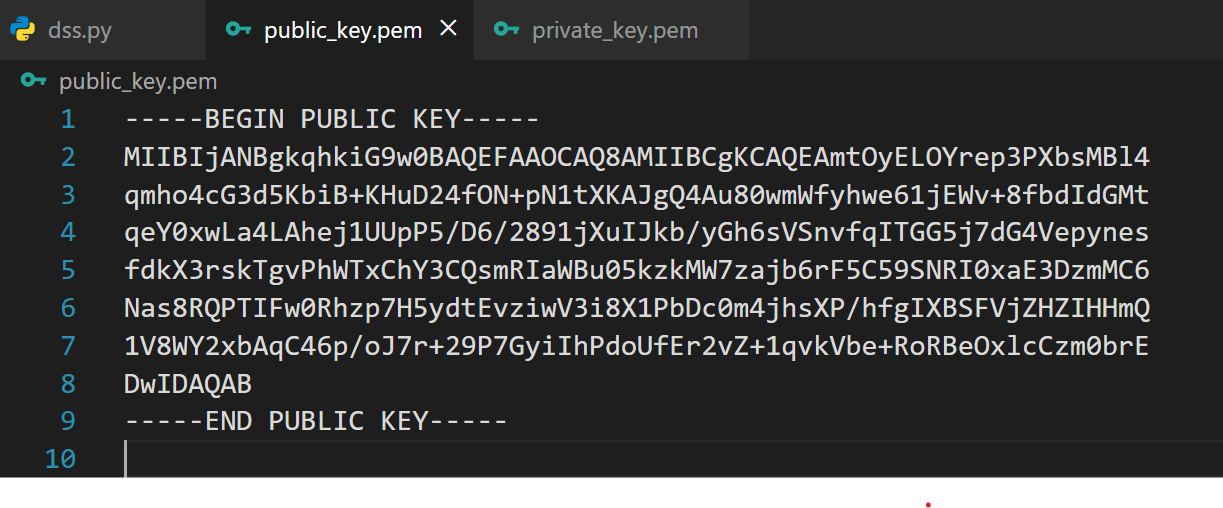
            print("Invalid choice. Please try again.")

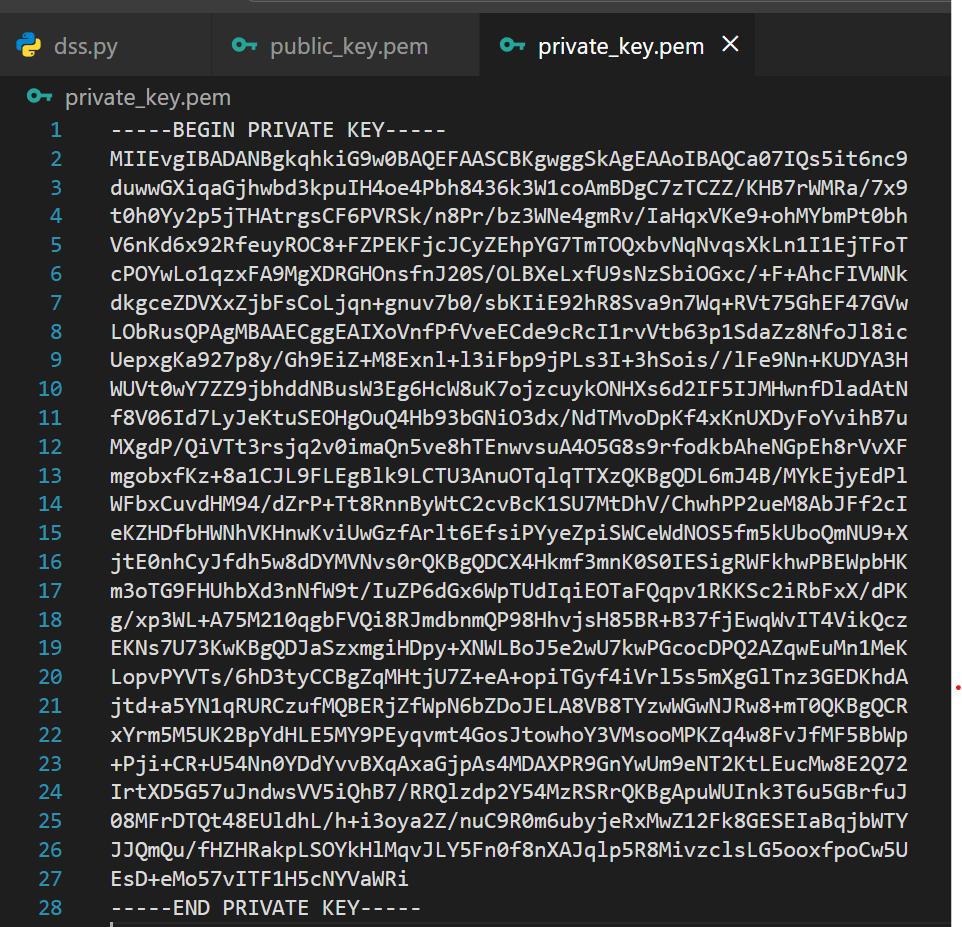
if \_\_name\_\_ == "\_\_main\_\_":

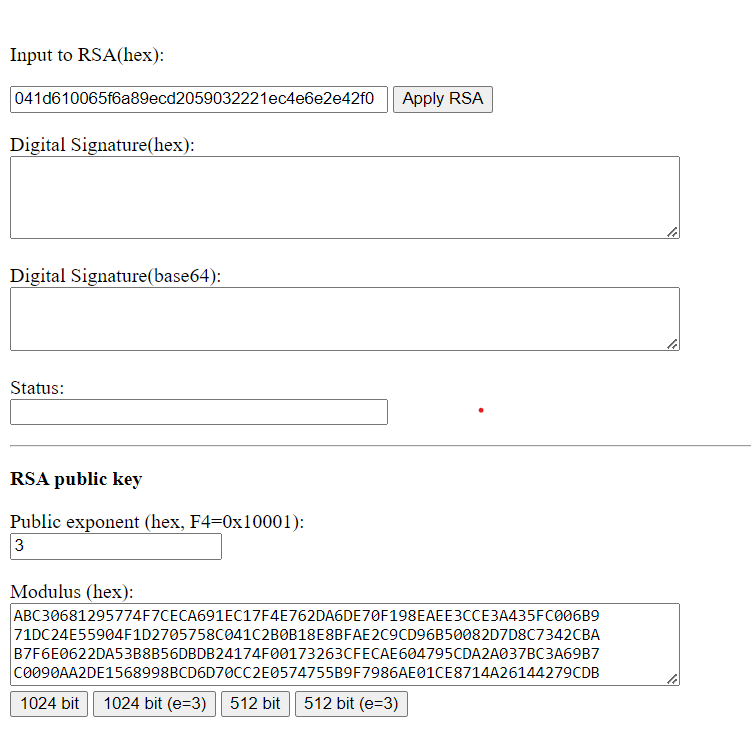
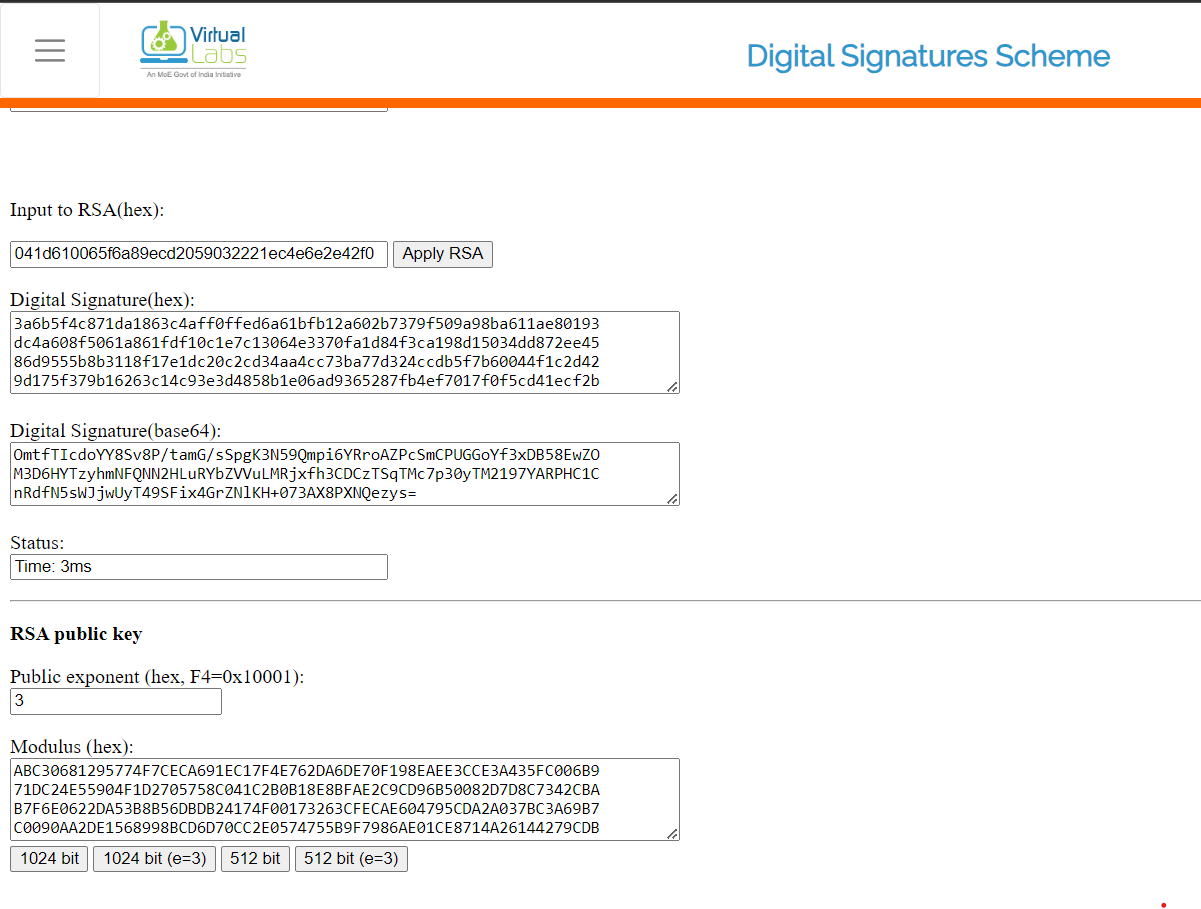
    menu()

**Output:  
  
**

****

****

****

**Virtual labs:  
  
  
  
  
**

**How It Works:**

* The **private key** signs the message, creating a unique signature.
* The **public key** is used by anyone to verify that the message has not been tampered with and that it was signed by the owner of the private key.

**Practical Applications:**

* **Digital Signatures** ensure the authenticity and integrity of a message or document.
* **Message Authentication**: In secure communication, the sender can sign the message, and the receiver can verify the signature to ensure the message is authentic.

**Advantages of DSS (Digital Signature Standard, based on DSA):**

* **Authentication**: Ensures the identity of the sender, as only the sender's private key can create a valid signature.
* **Integrity**: Guarantees that the message hasn't been altered, as any change would invalidate the signature.
* **Non-repudiation**: The sender cannot deny having signed the message, as the signature is unique to the private key.
* **Efficiency**: DSA is optimized for creating signatures, making it relatively fast for signing compared to some other algorithms.
* **Security**: Provides a high level of security, especially with modern key sizes (2048 bits or more).

**Disadvantages of DSS:**

* **Slower Verification**: DSA is slower at verifying signatures compared to some alternatives like RSA, making it less suitable for environments that require frequent verification.
* **Key Management**: Requires secure management of private keys; if the private key is compromised, the entire security system is at risk.
* **Message Size Limitations**: DSA only signs the hash of a message, so very large messages require hashing before signature generation.
* **Limited Use Cases**: DSA is primarily designed for digital signatures and not for encryption, unlike algorithms like RSA.

**Importance of DSS:**

* **Government Standard**: DSS (and DSA) is an official standard used by governments and organizations worldwide for secure digital signatures.
* **Legal Recognition**: Digital signatures created using DSS are often legally recognized, making them suitable for contracts and other legal documents.
* **Widely Used in Secure Communications**: DSS is crucial in protocols like SSL/TLS, ensuring secure web communications, digital certificates, and more.

In summary, DSS is important for ensuring the authenticity, integrity, and non-repudiation of digital communications, despite some performance-related limitations.