Name: Mansi Doshi
Enrollment Number: 20230905090518
Batch: B.Tech I.T. (Genius)
Subject: Cryptography And Network Security
ALA-1

Digital Signature Verifier

Title: Digital Signature Verifier using Python

Objective:

- Understand the concept of digital signatures and public-key cryptography.
- Implement a system to sign a message using a private key and verify it using the corresponding public key.
- Ensure message integrity using SHA-256 hashing.

Outcome:

- Learn how digital signatures ensure authenticity, integrity, and non-repudiation of messages.
- Hands-on experience with Python's cryptography library.

Python Installation of Required Libraries:

pip install cryptography

Key Concepts

1. Public-Key Cryptography:
   - Two keys: Private Key (kept secret) and Public Key (shared).
   - Private key is used for signing; public key is used for verifying.
2. Digital Signature:
   - A hash of the message is created using SHA-256.
   - This hash is encrypted using the private key → forms the digital signature.

    - Verification is done by decrypting the signature with the public key and comparing the hash.
3. SHA-256 Hashing:
    - Produces a fixed-length 256-bit hash.
    - Any tiny change in the message changes the hash completely → ensures integrity.

Source Code:

```
# Digital Signature Verifier in Python

from cryptography.hazmat.primitives import hashes

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

from cryptography.hazmat.primitives import serialization

# -----------------------------

# Step 1: Generate RSA Key Pair

# -----------------------------

private_key = rsa.generate_private_key(

    public_exponent=65537,

    key_size=2048

)

public_key = private_key.public_key()

# Save keys to files (optional)

with open("private_key.pem", "wb") as f:

    f.write(private_key.private_bytes(

        encoding=serialization.Encoding.PEM,

        format=serialization.PrivateFormat.PKCS8,

        encryption_algorithm=serialization.NoEncryption()

    ))

with open("public_key.pem", "wb") as f:
```

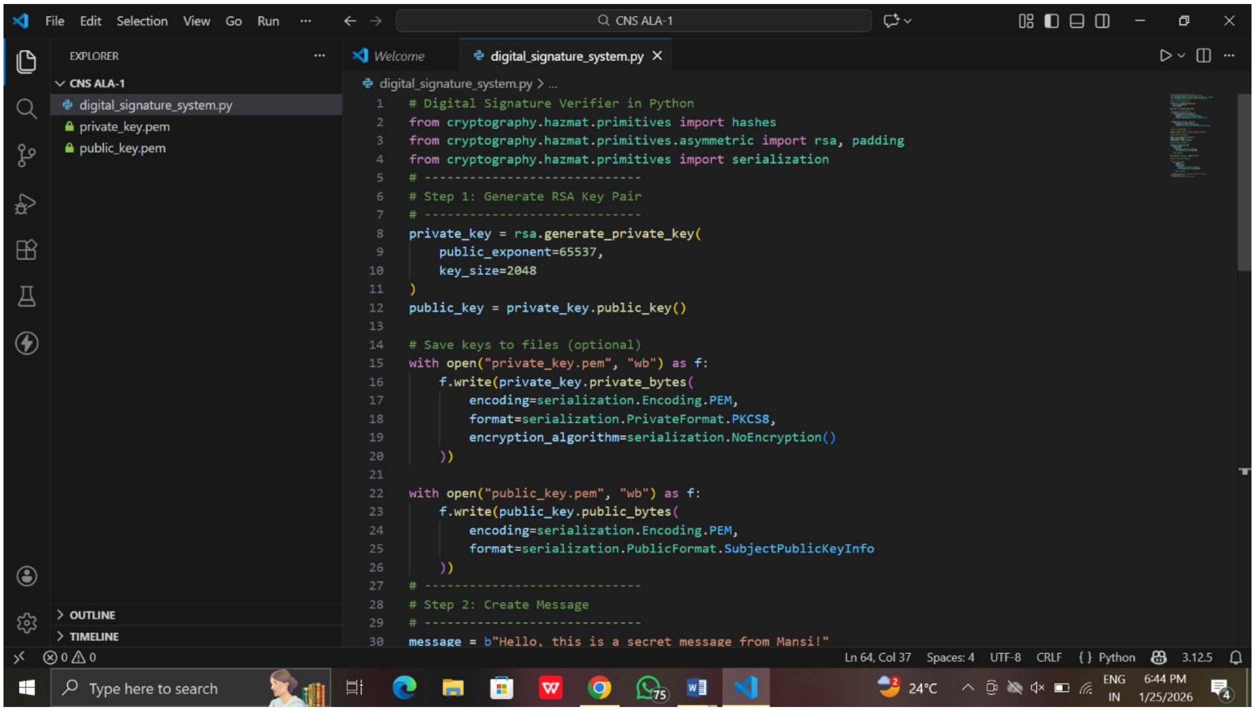
```
    f.write(public_key.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo
    ))
# -----------------------------
# Step 2: Create Message
# -----------------------------
message = b"Hello, this is a secret message from Mansi!"
# -----------------------------
# Step 3: Hash the Message
# -----------------------------
digest = hashes.Hash(hashes.SHA256())
digest.update(message)
hashed_message = digest.finalize()
# -----------------------------
# Step 4: Sign the Message
# -----------------------------
signature = private_key.sign(
    hashed_message,
    padding.PSS(
        mgf=padding.MGF1(hashes.SHA256()),
        salt_length=padding.PSS.MAX_LENGTH
    ),
    hashes.SHA256()
)
```

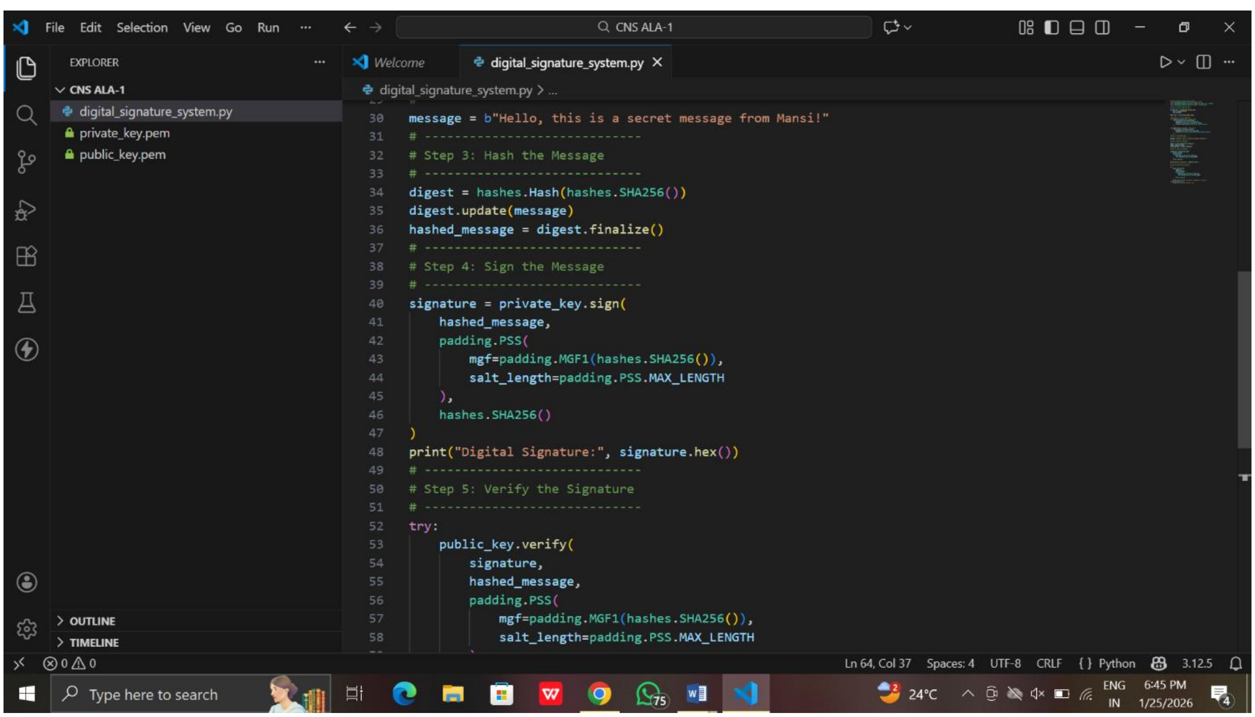
```
print("Digital Signature:", signature.hex())
# -----------------------------
# Step 5: Verify the Signature
# -----------------------------
try:
    public_key.verify(
        signature,
        hashed_message,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
    )
    print("Verification Successful! Signature is valid.")
except Exception as e:
    print("Verification Failed!", e)
```

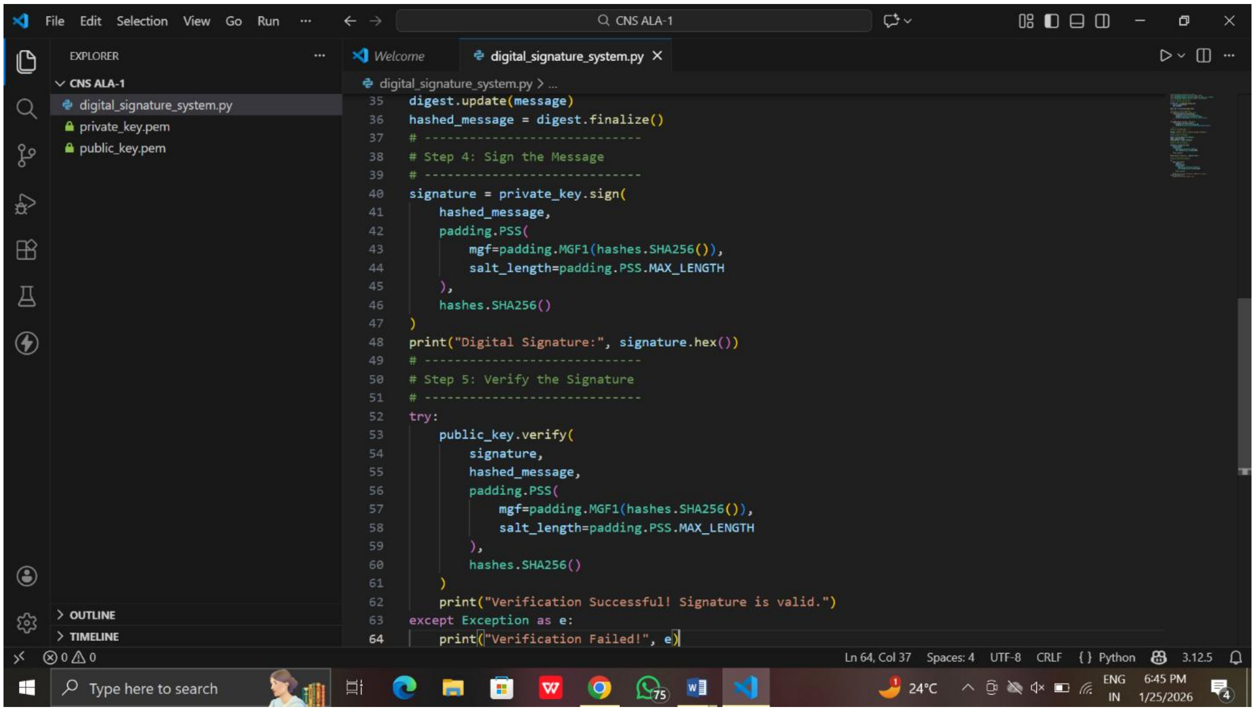
Explanation of Code:

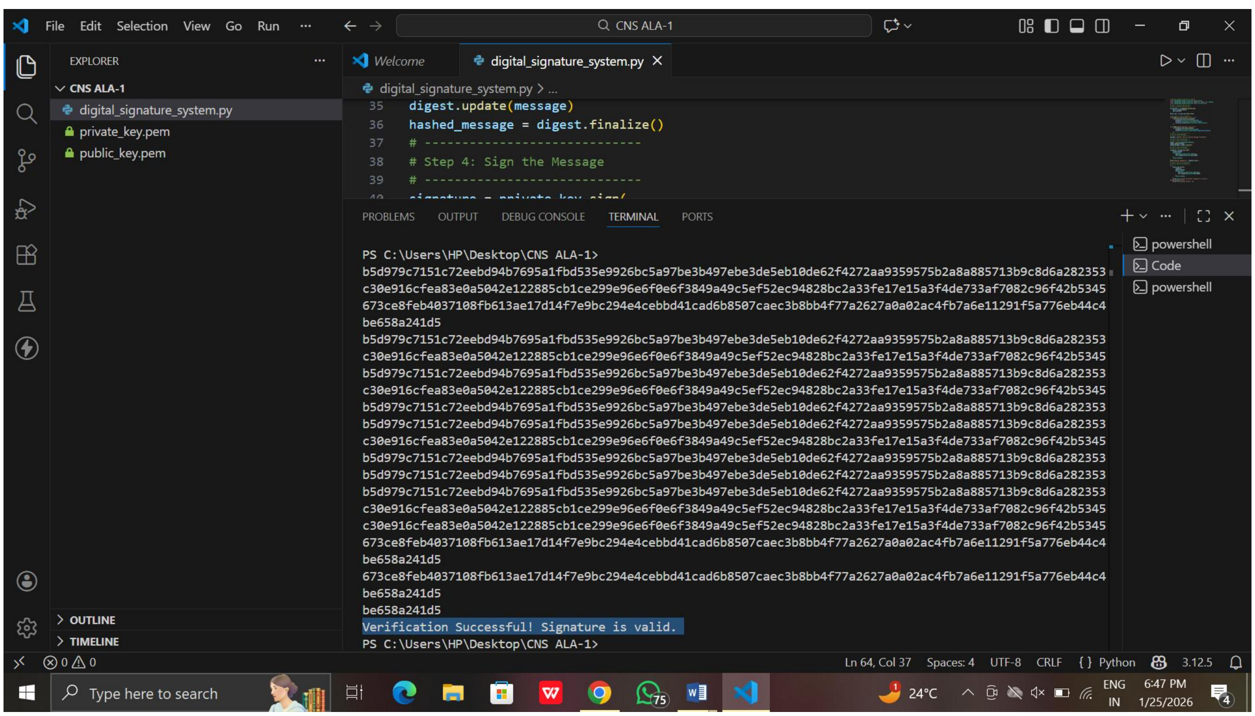
- rsa.generate_private_key() → Generates RSA private key (2048 bits).
- digest.update(message) → Hash the message using SHA-256.
- private_key.sign() → Sign the hash → Digital Signature.
- public_key.verify() → Verify the signature.

Screenshots:






digital_signature_system.py
digest.update(message)
hashed_message = digest.finalize()
# ---------------------------
# Step 4: Sign the Message
# ---------------------------
signature = private_key.sign(
    hashed_message,
    padding.PSS(
        mgf=padding.MGF1(hashes.SHA256()),
        salt_length=padding.PSS.MAX_LENGTH
    ),
    hashes.SHA256()
)
print("Digital Signature:", signature.hex())
# ---------------------------
# Step 5: Verify the Signature
# ---------------------------
try:
    public_key.verify(
        signature,
        hashed_message,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
)
    print("Verification Successful! Signature is valid.")
except Exception as e:
    print("Verification Failed!", e)


PS C:\Users\HP\Desktop\CNS ALA-1>
b5d979c7151c72eebd94b7695a1fbd535e9926bc5a97be3b497ebe3de5eb10de62f4272aa9359575b2a8a885713b9c8d6a282353
c30e916cfea83e0a5042e122885cb1ce299e96e6f0e6f3849a49c5ef52ec94828bc2a33fe17e15a3f4de733af7082c96f42b5345
673ce8feb4037108fb613ae17d14f7e9bc294e4cebbd41cad6b8507caec3b8bb4f77a2627a0a02ac4fb7a6e11291f5a776eb44c4
be658a241d5
b5d979c7151c72eebd94b7695a1fbd535e9926bc5a97be3b497ebe3de5eb10de62f4272aa9359575b2a8a885713b9c8d6a282353
c30e916cfea83e0a5042e122885cb1ce299e96e6f0e6f3849a49c5ef52ec94828bc2a33fe17e15a3f4de733af7082c96f42b5345
b5d979c7151c72eebd94b7695a1fbd535e9926bc5a97be3b497ebe3de5eb10de62f4272aa9359575b2a8a885713b9c8d6a282353
c30e916cfea83e0a5042e122885cb1ce299e96e6f0e6f3849a49c5ef52ec94828bc2a33fe17e15a3f4de733af7082c96f42b5345
b5d979c7151c72eebd94b7695a1fbd535e9926bc5a97be3b497ebe3de5eb10de62f4272aa9359575b2a8a885713b9c8d6a282353
b5d979c7151c72eebd94b7695a1fbd535e9926bc5a97be3b497ebe3de5eb10de62f4272aa9359575b2a8a885713b9c8d6a282353
c30e916cfea83e0a5042e122885cb1ce299e96e6f0e6f3849a49c5ef52ec94828bc2a33fe17e15a3f4de733af7082c96f42b5345
b5d979c7151c72eebd94b7695a1fbd535e9926bc5a97be3b497ebe3de5eb10de62f4272aa9359575b2a8a885713b9c8d6a282353
b5d979c7151c72eebd94b7695a1fbd535e9926bc5a97be3b497ebe3de5eb10de62f4272aa9359575b2a8a885713b9c8d6a282353
b5d979c7151c72eebd94b7695a1fbd535e9926bc5a97be3b497ebe3de5eb10de62f4272aa9359575b2a8a885713b9c8d6a282353
c30e916cfea83e0a5042e122885cb1ce299e96e6f0e6f3849a49c5ef52ec94828bc2a33fe17e15a3f4de733af7082c96f42b5345
c30e916cfea83e0a5042e122885cb1ce299e96e6f0e6f3849a49c5ef52ec94828bc2a33fe17e15a3f4de733af7082c96f42b5345
673ce8feb4037108fb613ae17d14f7e9bc294e4cebbd41cad6b8507caec3b8bb4f77a2627a0a02ac4fb7a6e11291f5a776eb44c4
be658a241d5
673ce8feb4037108fb613ae17d14f7e9bc294e4cebbd41cad6b8507caec3b8bb4f77a2627a0a02ac4fb7a6e11291f5a776eb44c4
be658a241d5
be658a241d5
Verification Successful! Signature is valid.
PS C:\Users\HP\Desktop\CNS ALA-1>