

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:

This screenshot shows the Visual Studio Code interface with the following details:

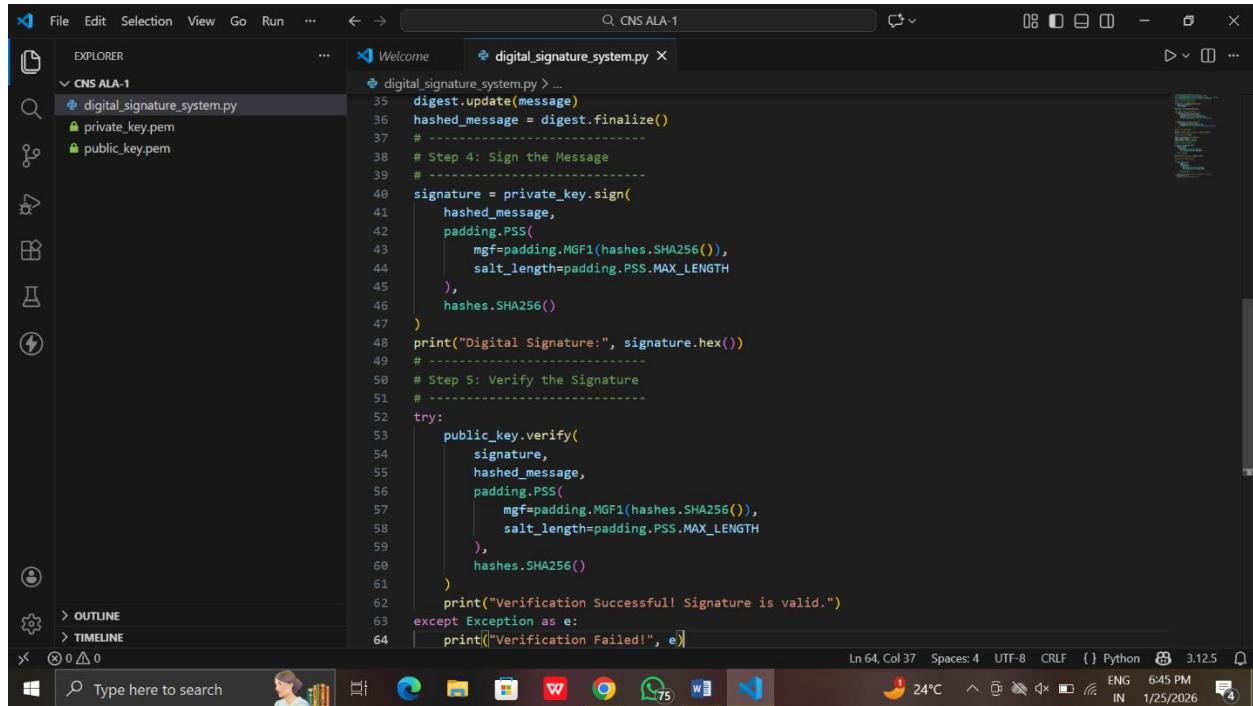
- File Explorer:** Shows a folder named "CNS ALA-1" containing "digital_signature_system.py", "private_key.pem", and "public_key.pem".
- Editor:** Displays the first 30 lines of a Python script named "digital_signature_system.py". The code initializes RSA keys and saves them to files.
- Bottom Status Bar:** Shows the file path "CNS ALA-1", line 64, column 37, spaces: 4, encoding: UTF-8, CRLF, Python 3.12.5, and the date/time "1/25/2026".

```
1 # Digital Signature Verifier in Python
2 from cryptography.hazmat.primitives import hashes
3 from cryptography.hazmat.primitives.asymmetric import rsa, padding
4 from cryptography.hazmat.primitives import serialization
5 #
6 # Step 1: Generate RSA Key Pair
7 #
8 private_key = rsa.generate_private_key(
9     public_exponent=65537,
10    key_size=2048
11 )
12 public_key = private_key.public_key()
13
14 # Save keys to files (optional)
15 with open("private_key.pem", "wb") as f:
16     f.write(private_key.private_bytes(
17         encoding=serialization.Encoding.PEM,
18         format=serialization.PrivateFormat.PKCS8,
19         encryption_algorithm=serialization.NoEncryption()
20     ))
21
22 with open("public_key.pem", "wb") as f:
23     f.write(public_key.public_bytes(
24         encoding=serialization.Encoding.PEM,
25         format=serialization.PublicFormat.SubjectPublicKeyInfo
26     ))
27 #
28 # Step 2: Create Message
29 #
30 message = b"Hello, this is a secret message from Mansi!"
```

This screenshot shows the Visual Studio Code interface with the following details:

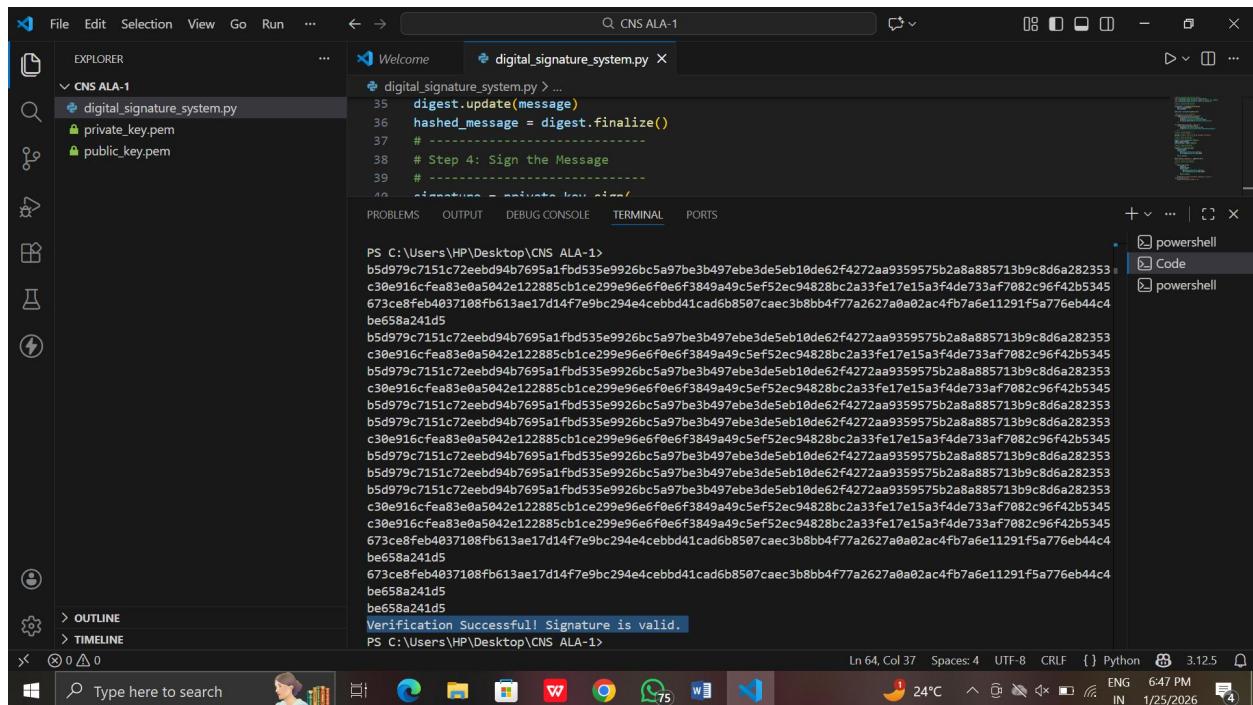
- File Explorer:** Shows a folder named "CNS ALA-1" containing "digital_signature_system.py", "private_key.pem", and "public_key.pem".
- Editor:** Displays the full Python script "digital_signature_system.py". It includes steps for hashing the message, signing it with the private key, and verifying the signature with the public key.
- Bottom Status Bar:** Shows the file path "CNS ALA-1", line 64, column 37, spaces: 4, encoding: UTF-8, CRLF, Python 3.12.5, and the date/time "1/25/2026".

```
30 message = b"Hello, this is a secret message from Mansi!"
31 #
32 # Step 3: Hash the Message
33 #
34 digest = hashes.Hash(hashes.SHA256())
35 digest.update(message)
36 hashed_message = digest.finalize()
37 #
38 # Step 4: Sign the Message
39 #
40 signature = private_key.sign(
41     hashed_message,
42     padding.PSS(
43         mgf=padding.MGF1(hashes.SHA256()),
44         salt_length=padding.PSS.MAX_LENGTH
45     ),
46     hashes.SHA256()
47 )
48 print("Digital Signature:", signature.hex())
49 #
50 # Step 5: Verify the Signature
51 #
52 try:
53     public_key.verify(
54         signature,
55         hashed_message,
56         padding.PSS(
57             mgf=padding.MGF1(hashes.SHA256()),
58             salt_length=padding.PSS.MAX_LENGTH
59     ))
60 except:
61     print("Signature verification failed!")
```



```
File Edit Selection View Go Run ... ← → Q CNS ALA-1
EXPLORER ... Welcome digital.signature_system.py
digital.signature_system.py > ...
35 digest.update(message)
36 hashed_message = digest.finalize()
37 #
38 # Step 4: Sign the Message
39 #
40 signature = private_key.sign(
41     hashed_message,
42     padding.PSS(
43         mgf=padding.MGF1(hashes.SHA256()),
44         salt_length=padding.PSS.MAX_LENGTH
45     ),
46     hashes.SHA256()
47 )
48 print("Digital Signature:", signature.hex())
49 #
50 # Step 5: Verify the Signature
51 #
52 try:
53     public_key.verify(
54         signature,
55         hashed_message,
56         padding.PSS(
57             mgf=padding.MGF1(hashes.SHA256()),
58             salt_length=padding.PSS.MAX_LENGTH
59         ),
60         hashes.SHA256()
61     )
62     print("Verification Successful! Signature is valid.")
63 except Exception as e:
64     print("Verification Failed!", e)

Ln 64, Col 37 Spaces: 4 UTF-8 CRLF {} Python 3.12.5 Q
24°C ⌂ ⌃ ⌄ ⌅ ⌆ ⌇ ENG 645 PM IN 1/25/2026 4
```



```
File Edit Selection View Go Run ... ← → Q CNS ALA-1
EXPLORER ... Welcome digital.signature_system.py
digital.signature_system.py > ...
35 digest.update(message)
36 hashed_message = digest.finalize()
37 #
38 # Step 4: Sign the Message
39 #
40 signature = private_key.sign(
41     hashed_message,
42     padding.PSS(
43         mgf=padding.MGF1(hashes.SHA256()),
44         salt_length=padding.PSS.MAX_LENGTH
45     ),
46     hashes.SHA256()
47 )
48 print("Digital Signature:", signature.hex())
49 #
50 # Step 5: Verify the Signature
51 #
52 try:
53     public_key.verify(
54         signature,
55         hashed_message,
56         padding.PSS(
57             mgf=padding.MGF1(hashes.SHA256()),
58             salt_length=padding.PSS.MAX_LENGTH
59         ),
60         hashes.SHA256()
61     )
62     print("Verification Successful! Signature is valid.")
63 except Exception as e:
64     print("Verification Failed!", e)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ⌂ ⌃ ⌄ ⌅ ⌆ ⌇
PS C:\Users\HP\Desktop\CNS ALA-1>
b5d979c7151c72eebd94b7695a1fb0d35e9926bc5a97be3b497ebe3de5eb10de62f4272aa9359575b2a8a885713b9c8d6a282353
c30e916cfea83e0a5942e122885cb1ce299e96e6f0e6f3849a49c5ef52ec94828bc2a33fe17e15a3f4de733af7082c96f42b5345
673cefeb4037108fb613ae17d14f7e9bc294e4cebbd41cad6b8507caec3b8bb4f77a0a2ac4fb7a6e11291f5a776eb44c4
be658a241d5
b5d979c7151c72eebd94b7695a1fb0d35e9926bc5a97be3b497ebe3de5eb10de62f4272aa9359575b2a8a885713b9c8d6a282353
c30e916cfea83e0a5942e122885cb1ce299e96e6f0e6f3849a49c5ef52ec94828bc2a33fe17e15a3f4de733af7082c96f42b5345
b5d979c7151c72eebd94b7695a1fb0d35e9926bc5a97be3b497ebe3de5eb10de62f4272aa9359575b2a8a885713b9c8d6a282353
c30e916cfea83e0a5942e122885cb1ce299e96e6f0e6f3849a49c5ef52ec94828bc2a33fe17e15a3f4de733af7082c96f42b5345
b5d979c7151c72eebd94b7695a1fb0d35e9926bc5a97be3b497ebe3de5eb10de62f4272aa9359575b2a8a885713b9c8d6a282353
b5d979c7151c72eebd94b7695a1fb0d35e9926bc5a97be3b497ebe3de5eb10de62f4272aa9359575b2a8a885713b9c8d6a282353
c30e916cfea83e0a5942e122885cb1ce299e96e6f0e6f3849a49c5ef52ec94828bc2a33fe17e15a3f4de733af7082c96f42b5345
c30e916cfea83e0a5942e122885cb1ce299e96e6f0e6f3849a49c5ef52ec94828bc2a33fe17e15a3f4de733af7082c96f42b5345
673cefeb4037108fb613ae17d14f7e9bc294e4cebbd41cad6b8507caec3b8bb4f77a0a2ac4fb7a6e11291f5a776eb44c4
be658a241d5
673cefeb4037108fb613ae17d14f7e9bc294e4cebbd41cad6b8507caec3b8bb4f77a0a2ac4fb7a6e11291f5a776eb44c4
be658a241d5
be658a241d5
Verification Successful! Signature is valid.
PS C:\Users\HP\Desktop\CNS ALA-1>

Ln 64, Col 37 Spaces: 4 UTF-8 CRLF {} Python 3.12.5 Q
24°C ⌂ ⌃ ⌄ ⌅ ⌆ ⌇ ENG 647 PM IN 1/25/2026 4
```