**Digital Signature**

**Source code**

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

from cryptography.hazmat.primitives import hashes

from cryptography.hazmat.backends import default\_backend

def generate\_key\_pair():

    private\_key = rsa.generate\_private\_key(public\_exponent=65537, key\_size=2048, backend=default\_backend())

    return private\_key, private\_key.public\_key()

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

    return private\_key.sign(message, padding.PSS(mgf=padding.MGF1(hashes.SHA256()), salt\_length=padding.PSS.MAX\_LENGTH), hashes.SHA256())

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

    try:

        public\_key.verify(signature, message, padding.PSS(mgf=padding.MGF1(hashes.SHA256()), salt\_length=padding.PSS.MAX\_LENGTH), hashes.SHA256())

        return True

    except:

        return False

# Example Usage:

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

# Message to be signed

message1 = b"This is the first message ."

message2 = b"This is the second message."

# Sign the first message

signature1 = sign\_message(private\_key, message1)

# Verify the signature for the first message

is\_valid1 = verify\_signature(public\_key, message1, signature1)

# Sign the second message

signature2 = sign\_message(private\_key, message2)

# Verify the signature for the second message

is\_valid2 = verify\_signature(public\_key, message2, signature2)

# Print results

print("Original Message 1:", message1.decode())

print("Signature Valid 1:", is\_valid1)

print("\nOriginal Message 2:", message2.decode())

print("Signature Valid 2:", is\_valid2)

**Output**

Original Message 1: This is the first.

Signature Valid 1: True

Original Message 2: This is the second message.

Signature Valid 2: True