**PRACTICAL 4**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name:** | Harsh Shah | **Semester:** | VII | **Division:** | 6 |
| **Roll No.:** | 21BCP359 | **Date:** | 22-08-24 | **Batch:** | G11 |
| **Aim:** | To Implement Digital Signature in any programming language | | | | |

**Digital Signature**

A digital signature is a mathematical scheme that is used to verify the integrity and authenticity of digital messages and documents. It may be considered as a digital version of the handwritten signature or stamped seal. The digital signatures use asymmetric cryptography i.e., also known as public key cryptography.

**Program**

import hashlib

import os

from cryptography.hazmat.primitives import serialization, hashes

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

import base64

PRIVATE\_KEY\_PATH = "example-rsa.pem"

PUBLIC\_KEY\_PATH = "example-rsa.pub"

PRIVATE\_KEY\_PASS = b"my$ecretp@$$word"

def generate\_private\_key():

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

pem\_private\_key = private\_key.private\_bytes(

encoding=serialization.Encoding.PEM,

format=serialization.PrivateFormat.PKCS8,

encryption\_algorithm=serialization.BestAvailableEncryption(PRIVATE\_KEY\_PASS),

)

with open(PRIVATE\_KEY\_PATH, "wb") as private\_key\_file:

private\_key\_file.write(pem\_private\_key)

return private\_key

def generate\_public\_key(private\_key):

pem\_public\_key = private\_key.public\_key().public\_bytes(

encoding=serialization.Encoding.PEM,

format=serialization.PublicFormat.SubjectPublicKeyInfo,

)

with open(PUBLIC\_KEY\_PATH, "wb") as public\_key\_file:

public\_key\_file.write(pem\_public\_key)

return pem\_public\_key

def get\_key\_pairs():

if os.path.exists(PRIVATE\_KEY\_PATH) and os.path.exists(PUBLIC\_KEY\_PATH):

with open(PRIVATE\_KEY\_PATH, "rb") as private\_key\_file:

keydata = private\_key\_file.read()

private\_key = serialization.load\_pem\_private\_key(

keydata, password=PRIVATE\_KEY\_PASS

)

with open(PUBLIC\_KEY\_PATH, "rb") as public\_key\_file:

keydata = public\_key\_file.read()

public\_key = serialization.load\_pem\_public\_key(keydata)

return private\_key, public\_key

else:

private\_key = generate\_private\_key()

generate\_public\_key(private\_key)

return private\_key

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

signature = private\_key.sign(message, padding.PKCS1v15(), hashes.SHA512())

return base64.b64encode(signature).decode("utf-8")

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

decoded\_signature = base64.b64decode(signature)

try:

public\_key.verify(

decoded\_signature, message, padding.PKCS1v15(), hashes.SHA512()

)

return True

except:

return False

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

private\_key, public\_key = get\_key\_pairs()

print("What do you want to do?\n 1-Sign\n 2-Verify")

choice = int(input("Enter your choice [1/2]: "))

if choice == 1:

message\_input = input("Enter the message to sign: ").encode("utf-8")

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

print("Signature:", signature)

elif choice == 2:

message\_input = input("Enter the message to verify: ").encode("utf-8")

signature\_input = input("Enter the signature: ")

is\_valid = verify\_signature(public\_key, message\_input, signature\_input)

print(f"Signature is {'valid' if is\_valid else 'invalid'}")

else:

print("Enter correct choice!")

**Output**

