**Lab Assignment 4**

**AP21110010302**

**Krish Srivastava**

**CSE – E  
Network Security – CSE 315L**

1. **Write a program to ensure sender authentication, integrity and confidentiality in client server communication using asymmetric key based mechanism.**

**Hint:**

**a. Generate public private key pair**

**b. Use protocol in slide 19 (include mac in it)**

**SERVER.py**

import socket

from Crypto.PublicKey import RSA

from Crypto.Cipher import PKCS1\_OAEP

from Crypto.Signature import PKCS1\_v1\_5

from Crypto.Hash import SHA256

def create\_key\_pair():

    private\_key = RSA.generate(2048)

    public\_key = private\_key.publickey()

    return private\_key, public\_key

def load\_public\_key\_from\_pem(pem):

    return RSA.import\_key(pem)

def serialize\_public\_key(public\_key):

    return public\_key.export\_key().decode()

def encrypt\_message(public\_key, message):

    cipher = PKCS1\_OAEP.new(public\_key)

    return cipher.encrypt(message.encode()).hex()

def decrypt\_message(private\_key, encrypted\_message):

    cipher = PKCS1\_OAEP.new(private\_key)

    return cipher.decrypt(bytes.fromhex(encrypted\_message)).decode()

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

    h = SHA256.new(message.encode())

    signer = PKCS1\_v1\_5.new(private\_key)

    return signer.sign(h).hex()

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

    h = SHA256.new(message.encode())

    verifier = PKCS1\_v1\_5.new(public\_key)

    return verifier.verify(h, bytes.fromhex(signature))

def send\_data(data, connection):

    connection.send(data.encode())

def receive\_data(connection):

    return connection.recv(1024).decode()

def exchange\_messages(connection, private\_key, other\_public\_key):

    while True:

        message = receive\_data(connection)

        print("Received (Encrypted):", message, "\n")

        signature = receive\_data(connection)

        print("Received (Signature):", signature, "\n")

        if verify\_signature(other\_public\_key, message, signature):

            message = decrypt\_message(private\_key, message)

            print("Decrypted:", message, "\n")

            print("Signature Verification: Successful", "\n")

        else:

            print("Signature Verification: Failed", "\n")

            continue

        outgoing\_message = input("You: ")

        outgoing\_message = encrypt\_message(other\_public\_key, outgoing\_message)

        print("Sent (Encrypted):", outgoing\_message, "\n")

        signature = sign\_message(private\_key, outgoing\_message)

        print("Sent (Signature):", signature, "\n")

        send\_data(outgoing\_message, connection)

        send\_data(signature, connection)

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind(('localhost', 12345))

server\_socket.listen(1)

print("Server is listening...")

connection, address = server\_socket.accept()

print("Connection from:", address)

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

send\_data(serialize\_public\_key(public\_key), connection)

other\_public\_key = load\_public\_key\_from\_pem(receive\_data(connection))

exchange\_messages(connection, private\_key, other\_public\_key)

**CLIENT.py**

import socket

from Crypto.PublicKey import RSA

from Crypto.Cipher import PKCS1\_OAEP

from Crypto.Signature import PKCS1\_v1\_5

from Crypto.Hash import SHA256

def create\_key\_pair():

    private\_key = RSA.generate(2048)

    public\_key = private\_key.publickey()

    return private\_key, public\_key

def load\_public\_key\_from\_pem(pem):

    return RSA.import\_key(pem)

def serialize\_public\_key(public\_key):

    return public\_key.export\_key().decode()

def encrypt\_message(public\_key, message):

    cipher = PKCS1\_OAEP.new(public\_key)

    return cipher.encrypt(message.encode()).hex()

def decrypt\_message(private\_key, encrypted\_message):

    cipher = PKCS1\_OAEP.new(private\_key)

    return cipher.decrypt(bytes.fromhex(encrypted\_message)).decode()

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

    h = SHA256.new(message.encode())

    signer = PKCS1\_v1\_5.new(private\_key)

    return signer.sign(h).hex()

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

    h = SHA256.new(message.encode())

    verifier = PKCS1\_v1\_5.new(public\_key)

    return verifier.verify(h, bytes.fromhex(signature))

def send\_data(data, connection):

    connection.send(data.encode())

def receive\_data(connection):

    return connection.recv(1024).decode()

def exchange\_messages(connection, private\_key, other\_public\_key):

    while True:

        outgoing\_message = input("You: ")

        outgoing\_message = encrypt\_message(other\_public\_key, outgoing\_message)

        print("Sent (Encrypted):", outgoing\_message, "\n")

        signature = sign\_message(private\_key, outgoing\_message)

        print("Sent (Signature):", signature, "\n")

        send\_data(outgoing\_message, connection)

        send\_data(signature, connection)

        message = receive\_data(connection)

        print("Received (Encrypted):", message, "\n")

        signature = receive\_data(connection)

        print("Received (Signature):", signature, "\n")

        if verify\_signature(other\_public\_key, message, signature):

            message = decrypt\_message(private\_key, message)

            print("Decrypted:", message, "\n")

            print("Signature Verification: Successful", "\n")

        else:

            print("Signature Verification: Failed", "\n")

            continue

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect(('localhost', 12345))

server\_public\_key\_pem = receive\_data(client\_socket)

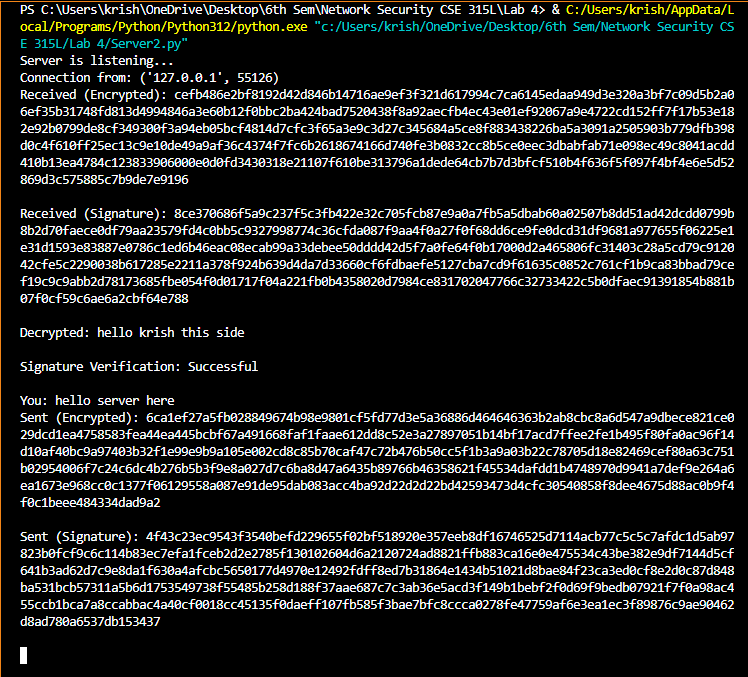
server\_public\_key = load\_public\_key\_from\_pem(server\_public\_key\_pem)

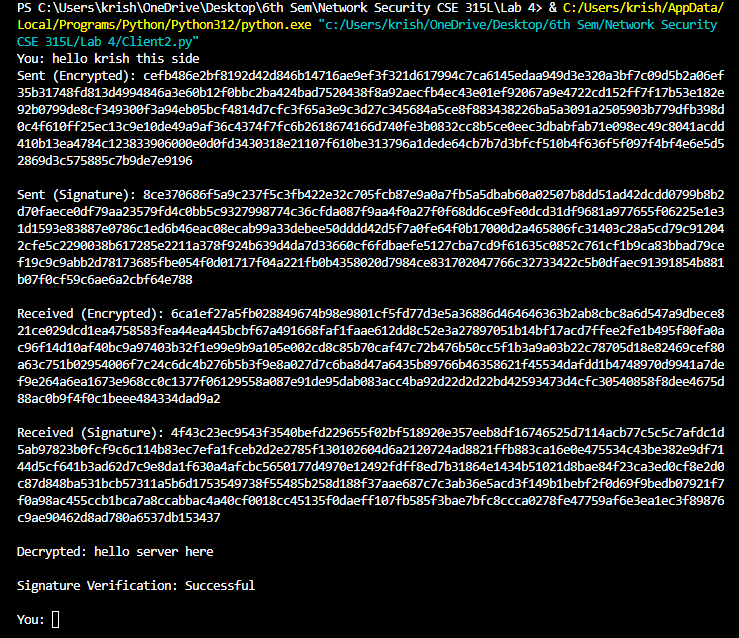
client\_private\_key, client\_public\_key = create\_key\_pair()

send\_data(serialize\_public\_key(client\_public\_key), client\_socket)

exchange\_messages(client\_socket, client\_private\_key, server\_public\_key)

**OUTPUT**



****