Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Assignment #1

[Course title: COMP 492]

[For partial fulfillment of 4th year/2nd Semester in Computer Engineering]

Submitted by:

Gaurab Shrestha(43)

Submitted to:

Mr. Suresh Gautam

Department of Computer Science & Engineering(DoCSE)

Submission Date:

February 3, 2023

Task : To Implement AES or RSA encryption algorithm in Python that will be used to encrypt and decrypt messages sent between the client and the server.

➔ **"RSA algorithm"** is an asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. Public Key and Private Key. As the name describes, the Public Key is given to everyone and the Private key is kept private.

An example of asymmetric cryptography:

1. A client (for example browser) sends its public key to the server and requests some data.
2. The server encrypts the data using the client's public key and sends the encrypted data.
3. The client receives this data and decrypts it.

Since this is asymmetric, nobody else except the browser can decrypt the data even if a third party has the public key of the browser.

Algorithm:

1. Key Generation:
   a. Select two large prime numbers, p and q, and compute n = p * q.
   b. Compute the totient of n, φ(n) = (p-1) * (q-1).
   c. Select a public key exponent, e, such that 1 < e < φ(n) and e is coprime to φ(n).
   d. Compute the private key exponent, d, such that d * e ≡ 1 (mod φ(n)).
   e. The public key is the pair (n, e) and the private key is (n, d).

2. Encryption:
   a. To encrypt a message m, the sender computes the ciphertext c ≡ m^e (mod n).

3. Decryption:
   a. To decrypt the ciphertext, the receiver computes the original message m ≡ c^d (mod n).

The security of RSA is based on the difficulty of factoring large numbers and computing the private key d from the public key (n, e).

Implementation of client-server communication design using rsa:

1. At first, a simple socket program was implemented between client and server using the socket library.
2. Secondly, the keys(public and private key) for both client and server were generated randomly and securely using the rsa library and were stored in a separate folder named as keys.
3. The message sent from server and client was encrypted and decrypted using rsa algorithm, exchanging the generated public and private key for both server to client message program and vice versa.
4. The communication between server and client using rsa encryption algorithm was successfully implemented.
5. Lastly, a test plan and test case to show that the implementation is working correctly was integrated using the python unit testing module.

Output:

The implementation of the RSA encryption algorithm in Python for the communication with the feature of encryption and decryption of messages sent between the client and the server is linked here. Below screenshot depicts the basic working of the program.

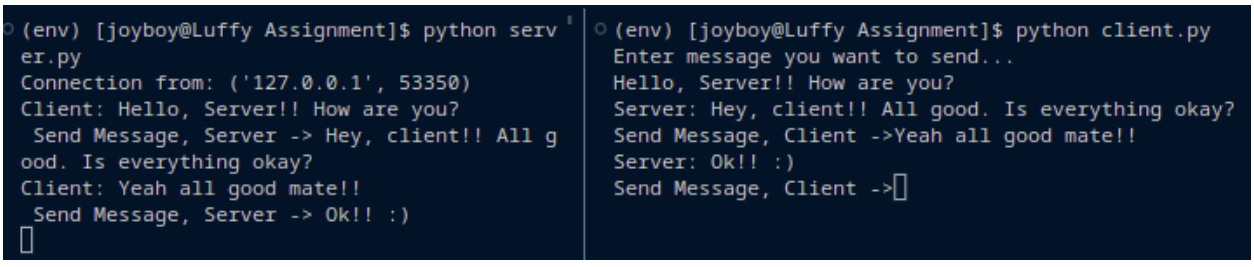

Fig 1: Output of communication after encryption and decryption between server and client

Here, the Python unittest module is used to test a unit of source code and the output can be seen as follows.

```
(env) [joyboy@Luffy Assignment]$ python testcase.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.030s

OK
```

Fig 2: Output of test case

Conclusion

Thus, in this way, the successful implementation of rsa encryption algorithm was done using python as compatible programming language for the client server communication and some test cases were also implemented to check whether the encryption and decryption of the passed messages passes the test successfully or not.

Appendix

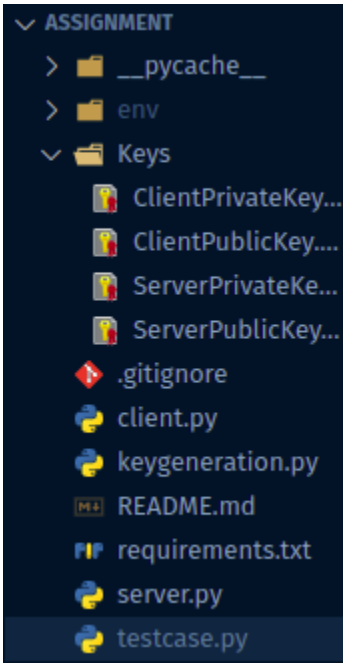

Fig 3: Folder Structure

keygeneration.py > ...

You, yesterday | 1 author (You)

```python
import rsa      You, yesterday • Two Way Communication using RSA …

def generateKeysForServerToClientMsg():
    public_key_Server, private_key_Client = rsa.newkeys(1024)
    with open("Keys/ServerPublicKey.pem","wb") as f:
        f.write(public_key_Server.save_pkcs1("PEM"))

    with open("Keys/ClientPrivateKey.pem","wb") as f:
        f.write(private_key_Client.save_pkcs1("PEM"))

def generateKeysForClientToServerMsg():
    public_key_Client, private_key_Server = rsa.newkeys(1024)
    with open("Keys/ClientPublicKey.pem","wb") as f:
        f.write(public_key_Client.save_pkcs1("PEM"))

    with open("Keys/ServerPrivateKey.pem","wb") as f:
        f.write(private_key_Server.save_pkcs1("PEM"))

generateKeysForServerToClientMsg()
generateKeysForClientToServerMsg()
```

Fig 4 : Keygeneration.py

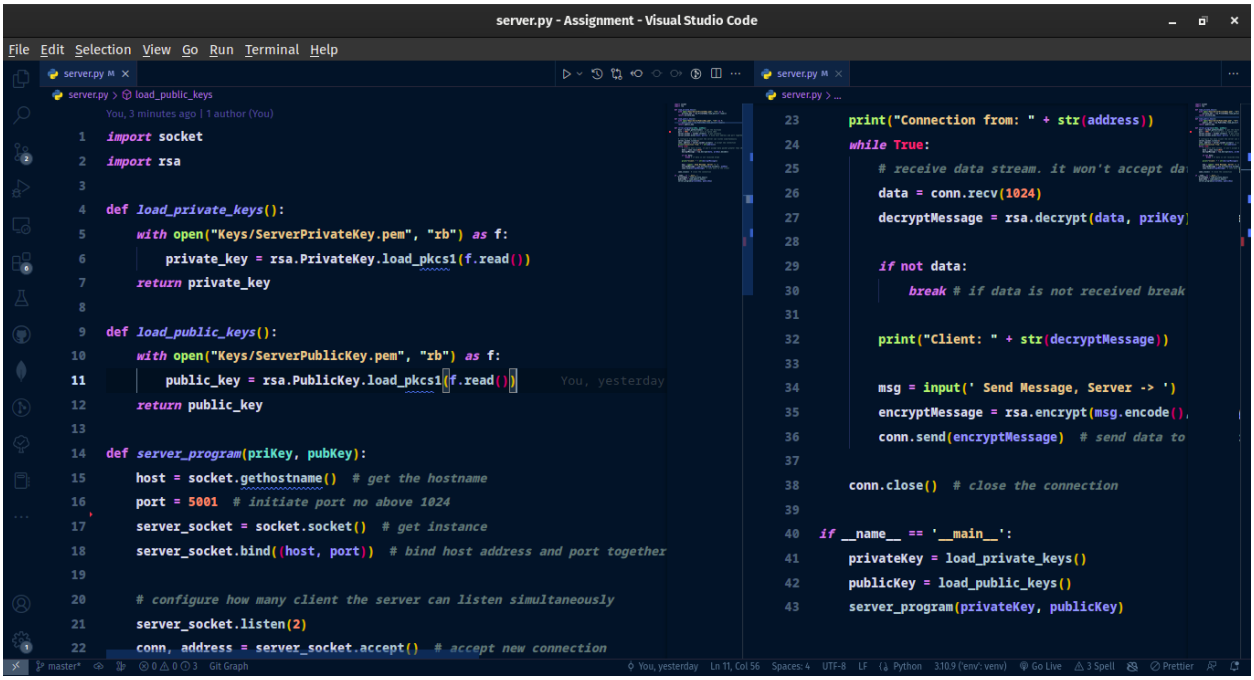

Fig 5 : Server.py

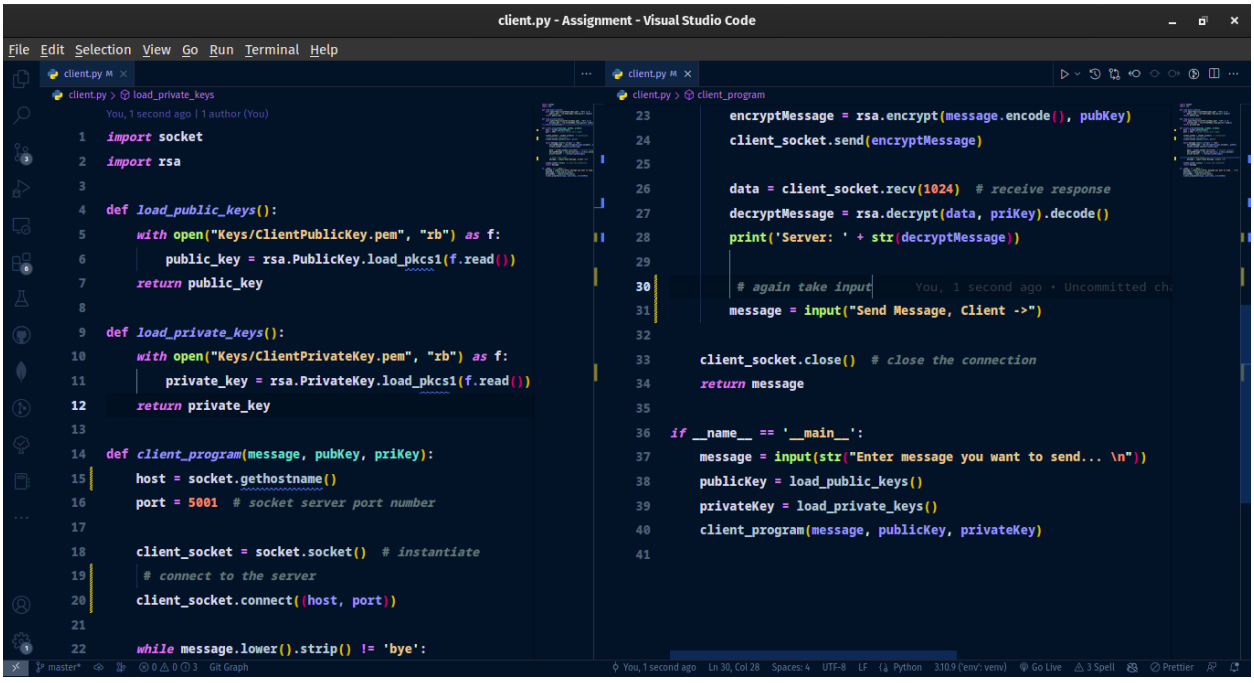

Fig 6: Client.py

```python
import unittest
from client import load_private_keys as clientPriKey, load_public_keys as clientPubKey
from server import load_private_keys as serverPriKey, load_public_keys as serverPubKey
import rsa      You, 14 minutes ago • Testcases added …
You, 1 second ago | 1 author (You)
class TestClientServerCommunication(unittest.TestCase):
    def test_client(self):
        clientMessage = 'Hey, Server'
        # clientMessage = 'Something wrong'
        PubKey = clientPubKey()
        PriKey = serverPriKey()
        encryptMessage = rsa.encrypt(clientMessage.encode(), PubKey)
        decryptMessage = rsa.decrypt(encryptMessage, PriKey).decode()
        self.assertEqual(decryptMessage, 'Hey, Server')

    def test_server(self):
        serverMessage = 'Hey, Sai'
        # serverMessage = 'New me'
        PubKey = serverPubKey()
        PriKey = clientPriKey()
        encryptMessage = rsa.encrypt(serverMessage.encode(), PubKey)
        decryptMessage = rsa.decrypt(encryptMessage, PriKey).decode()
        self.assertEqual(decryptMessage, 'Hey, Client')

if __name__ == '__main__':
    unittest.main()
```

Fig 7 : testcase.py