

**Kathmandu University**  
**Department of Computer Science and Engineering**  
**Dhulikhel, Kavre**



**Assignment #1**  
**[ Course title: COMP 492]**

[For partial fulfillment of 4<sup>th</sup> year/2<sup>nd</sup> 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  $n$ ,  $\phi(n) = (p-1) * (q-1)$ .
  - c. Select a public key exponent,  $e$ , such that  $1 < e < \phi(n)$  and  $e$  is coprime to  $\phi(n)$ .
  - d. Compute the private key exponent,  $d$ , such that  $d * e \equiv 1 \pmod{\phi(n)}$ .
  - 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  $c \equiv m^e \pmod{n}$ .
3. Decryption:
  - a. To decrypt the ciphertext, the receiver computes the original message  $m \equiv c^d \pmod{n}$ .

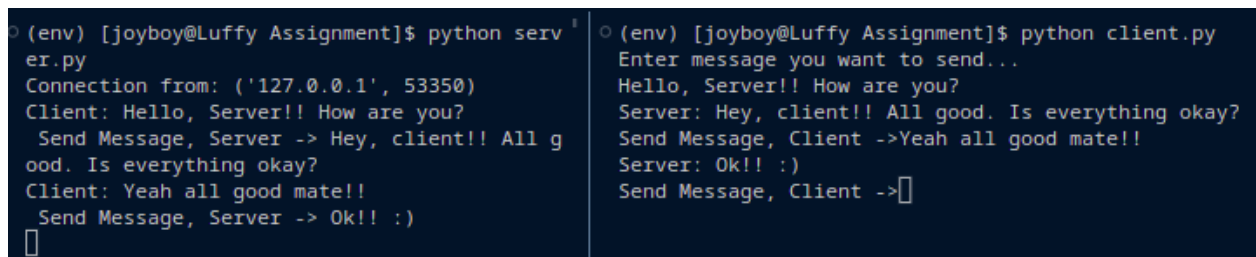
The security of RSA is based on the difficulty of factoring large numbers and computing the private key  $d$  from the public key  $(n, e)$ .

## 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.



```
(env) [joyboy@Luffy Assignment]$ python server.py
Connection from: ('127.0.0.1', 53350)
Client: Hello, Server!! How are you?
Send Message, Server -> Hey, client!! All good. Is everything okay?
Client: Yeah all good mate!!
Send Message, Server -> Ok!! :)
^

(env) [joyboy@Luffy Assignment]$ python client.py
Enter message you want to send...
Hello, Server!! How are you?
Server: Hey, client!! All good. Is everything okay?
Send Message, Client ->Yeah all good mate!!
Server: Ok!! :)
Send Message, Client ->^
```

*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

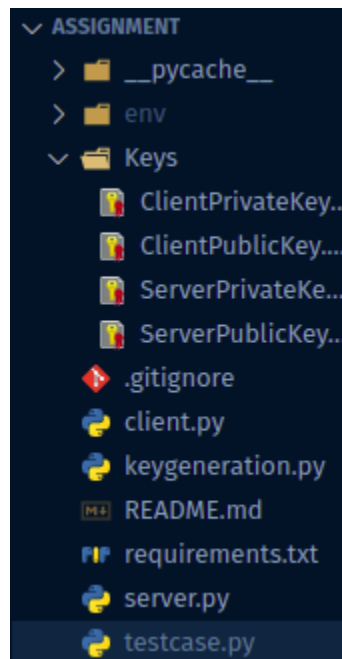
OK
```

*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)

1  import rsa          You, yesterday • Two Way Communication using RSA ...
2
3  def generateKeysForServerToClientMsg():
4      public_key_Server, private_key_Client = rsa.newkeys(1024)
5      with open("Keys/ServerPublicKey.pem","wb") as f:
6          f.write(public_key_Server.save_pkcs1("PEM"))
7
8      with open("Keys/ClientPrivateKey.pem","wb") as f:
9          f.write(private_key_Client.save_pkcs1("PEM"))
10
11 def generateKeysForClientToServerMsg():
12     public_key_Client, private_key_Server = rsa.newkeys(1024)
13     with open("Keys/ClientPublicKey.pem","wb") as f:
14         f.write(public_key_Client.save_pkcs1("PEM"))
15
16     with open("Keys/ServerPrivateKey.pem","wb") as f:
17         f.write(private_key_Server.save_pkcs1("PEM"))
18
19 generateKeysForServerToClientMsg()
20 generateKeysForClientToServerMsg()

```

*Fig 4 : Keygeneration.py*

```

server.py - Assignment - Visual Studio Code
File Edit Selection View Go Run Terminal Help

server.py M x
server.py > load_public_keys
You, 3 minutes ago | 1 author (You)
1 import socket
2 import rsa
3
4 def load_private_keys():
5     with open("Keys/ServerPrivateKey.pem", "rb") as f:
6         private_key = rsa.PrivateKey.load_pkcs1(f.read())
7     return private_key
8
9 def load_public_keys():
10    with open("Keys/ServerPublicKey.pem", "rb") as f:
11        public_key = rsa.PublicKey.load_pkcs1(f.read())
12    return public_key
13
14 def server_program(privateKey, pubKey):
15     host = socket.gethostname() # get the hostname
16     port = 5001 # initiate port no above 1024
17     server_socket = socket.socket() # get instance
18     server_socket.bind((host, port)) # bind host address and port together
19
20     # configure how many client the server can listen simultaneously
21     server_socket.listen(2)
22     conn, address = server_socket.accept() # accept new connection
23
24     print("Connection from: " + str(address))
25     while True:
26         # receive data stream. it won't accept data from the client until it receives the full message.
27         data = conn.recv(1024)
28         decryptMessage = rsa.decrypt(data, privateKey)
29
30         if not data:
31             break # if data is not received break
32
33         print("Client: " + str(decryptMessage))
34
35         msg = input(' Send Message, Server -> ')
36         encryptMessage = rsa.encrypt(msg.encode(), pubKey)
37         conn.send(encryptMessage) # send data to
38
39     conn.close() # close the connection
40
41 if __name__ == '__main__':
42     privateKey = load_private_keys()
43     publicKey = load_public_keys()
44     server_program(privateKey, publicKey)

```

Fig 5 : Server.py

```

client.py - Assignment - Visual Studio Code
File Edit Selection View Go Run Terminal Help

client.py M x
client.py > load_public_keys
You, 1 second ago | 1 author (You)
1 import socket
2 import rsa
3
4 def load_public_keys():
5     with open("Keys/ClientPublicKey.pem", "rb") as f:
6         public_key = rsa.PublicKey.load_pkcs1(f.read())
7     return public_key
8
9 def load_private_keys():
10    with open("Keys/ClientPrivateKey.pem", "rb") as f:
11        private_key = rsa.PrivateKey.load_pkcs1(f.read())
12    return private_key
13
14 def client_program(message, pubKey, priKey):
15     host = socket.gethostname()
16     port = 5001 # socket server port number
17
18     client_socket = socket.socket() # instantiate
19     # connect to the server
20     client_socket.connect((host, port))
21
22     while message.lower().strip() != 'bye':
23         encryptMessage = rsa.encrypt(message.encode(), pubKey)
24         client_socket.send(encryptMessage)
25
26         data = client_socket.recv(1024) # receive response
27         decryptMessage = rsa.decrypt(data, priKey).decode()
28         print('Server: ' + str(decryptMessage))
29
30         # again take input
31         message = input("Send Message, Client -> ")
32
33     client_socket.close() # close the connection
34     return message
35
36 if __name__ == '__main__':
37     message = input(str("Enter message you want to send... \n"))
38     publicKey = load_public_keys()
39     privateKey = load_private_keys()
40     client_program(message, publicKey, privateKey)
41

```

Fig 6: Client.py

```

1  import unittest
2  from client import load_private_keys as clientPriKey, load_public_keys as clientPubKey
3  from server import load_private_keys as serverPriKey, load_public_keys as serverPubKey
4  import rsa
5
6  class TestClientServerCommunication(unittest.TestCase):
7
8      def test_client(self):
9          clientMessage = 'Hey, Server'
10         # clientMessage = 'Something wrong'
11         PubKey = clientPubKey()
12         PriKey = serverPriKey()
13         encryptMessage = rsa.encrypt(clientMessage.encode(), PubKey)
14         decryptMessage = rsa.decrypt(encryptMessage, PriKey).decode()
15         self.assertEqual(decryptMessage, 'Hey, Server')
16
17     def test_server(self):
18         serverMessage = 'Hey, Sai'
19         # serverMessage = 'New me'
20         PubKey = serverPubKey()
21         PriKey = clientPriKey()
22         encryptMessage = rsa.encrypt(serverMessage.encode(), PubKey)
23         decryptMessage = rsa.decrypt(encryptMessage, PriKey).decode()
24         self.assertEqual(decryptMessage, 'Hey, Client')
25
26 if __name__ == '__main__':
27     unittest.main()

```

Fig 7 : testcase.py