Assignment-1 Problem: Secure communication between a client and a server

Submitted by :
Janak Sitaula
CE 2020

Submitted to:
Mr. Suresh Gautam
Instructor – COMP 492 Blockchain Technology

Date: 16/03/2025

Use Case:

A company is developing a new messaging service that allows clients to communicate securely with a server. The company wants to ensure that all messages sent between the client and the server are encrypted and secure.

Assignment:

Your task is to write an implementation of the AES or RSA encryption algorithm in Python that will be used to encrypt and decrypt messages sent between the client and the server. The implementation should include the ability to generate a key for encryption/decryption and should be able to handle large amounts of data. The code should also include a mechanism for securely exchanging the encryption key between the client and the server.

Implementation details:

- The program should be able to encrypt and decrypt data using AES or RSA algorithm.
- The encryption key should be generated randomly and securely.
- The key should be exchanged between the client and the server securely (e.g. by using asymmetric encryption).
- The program should be able to handle large amounts of data.
- The program should be well-documented and easy to understand.
- The program should be compatible with Python 3.

Implementation:

The program has been developed in three component:
1. Server component represented by server.py
2. Client component represented by client.py
3. Test component represented by test.py

My implementation uses the ‘rsa’ library in python to create a client-server application with asymmetric encryption. This server generates and stores RSA key pairs, exposes the public key via an endpoint and provides endpoints for encrypted communication. The client fetches the public key, encrypts the message locally and sends them to the server for decryption.

Key components:

- Key storage in PEM format files
- Public key distribution via HTTP
- Message encryption with rsa.encrypt()
- Message decryption with rsa.decrypt()

Design choices:

1. **Key Size (2048-bit):** provides strong security for general uses however with computational overhead
2. **Key storage in files:** offers persistence across server restarts but requires file system access and proper permissions
3. **HTTP for Key Distribution:** Simple implementation using standard web protocols however vulnerable to man-in-the-middle attacks without HTTPS
4. **Using** *jsonpickle* **for serialization:** Handles complex Python objects but less efficient than pure JSON for simple data

Test plan:

Unit test strategy included testing full encryption and decryption cycle with key generation verification, storage, and retrieval. It also checks error handling for invalid inputs. Integration testing is achieved via end-to-end communication between client and server. Performance testing can be achieved via measurement of encryption and decryption speed with varying message sizes.

Code:

server.py

```
from flask import Flask, request
import rsa
import jsonpickle
import os

app = Flask(__name__)

# Generate RSA keys when server starts
def generate_keys_if_needed():
try:
with open("private.pem", "r") as private_file:
with open("public.pem", "r") as public_file:
print("Keys already exist.")
return
except FileNotFoundError:
print("Generating new keys...")
publicKey, privateKey = rsa.newkeys(2048)
publicKey_PEM = publicKey.save_pkcs1().decode('utf8')
privateKey_PEM = privateKey.save_pkcs1().decode('utf8')
with open("private.pem", "w") as private_file:
private_file.write(privateKey_PEM)
with open("public.pem", "w") as public_file:
public_file.write(publicKey_PEM)
print("Keys generated successfully.")

# Generate keys at startup
generate_keys_if_needed()

@app.route("/")
def home():
return "<p>OK!</p>"

# Serve the public key
@app.route("/getkey")
def serve_public_key():
with open("public.pem", "r") as public_file:
response = public_file.read()
return jsonpickle.encode(response) # Return as json format

# Decrypt and process incoming messages
@app.route("/message", methods=['POST'])
def process_message():
with open("private.pem", "r") as private_file:
private_key = rsa.PrivateKey.load_pkcs1(private_file.read())
encrypted_data = request.data
print(encrypted_data)
```

```
decrypted_message = rsa.decrypt(encrypted_data, private_key).decode()
print(f"Decrypted Message: {decrypted_message}")
return "Message received by server"

# Test endpoint for debugging
@app.route("/test", methods=['POST'])
def test_endpoint():
with open("private.pem", "r") as private_file:
private_key = rsa.PrivateKey.load_pkcs1(private_file.read())
encrypted_data = request.data
print(encrypted_data)
decrypted_message = rsa.decrypt(encrypted_data, private_key).decode()
print(f"Test Decryption: {decrypted_message}")
return decrypted_message

if __name__ == "__main__":
app.run(debug=True, port=8080, use_reloader=False)
```

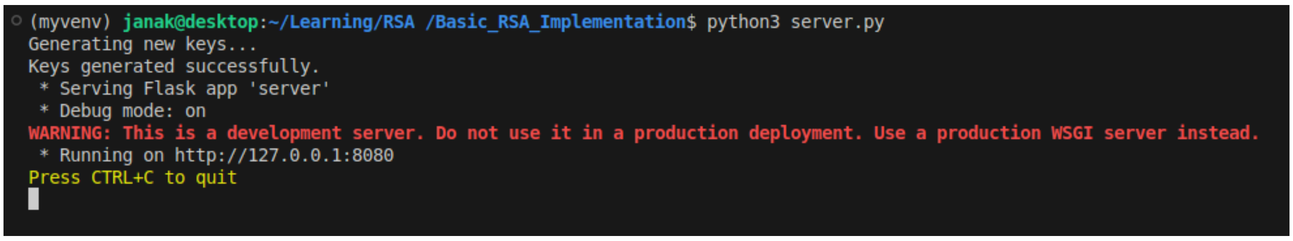
client.py

```
import requests
import os
import rsa
import json

# Disable proxy for localhost
os.environ['NO_PROXY'] = '127.0.0.1'

# Fetch public key from server
try:
response = requests.get('http://127.0.0.1:8080/getkey') # Changed port to 8080 to match server
# Parse the jsonpickle response correctly
decoded_response = json.loads(response.content)
public_key_raw = decoded_response
# Load the public key properly
public_key = rsa.PublicKey.load_pkcs1(public_key_raw)
print("Successfully loaded the public key from server")
except Exception as e:
print(f"Error fetching or parsing the public key: {e}")
exit(1)

# Loop to send encrypted messages
while True:
try:
user_input = input("Enter message (or type 'exit' to quit):\n")
if user_input.lower() == 'exit':
print("Exiting...")
break
# Encrypt the message using the server's public key
encrypted_msg = rsa.encrypt(user_input.encode('utf-8'), public_key)
print(f"Message encrypted successfully")
# Send the encrypted message to the server
server_response = requests.post('http://127.0.0.1:8080/message', data=encrypted_msg)
# Handle the server response
if server_response.status_code == 200:
print(f"Server Response: {server_response.content.decode()}\n")
else:
print(f"Error: Server returned status code {server_response.status_code}")
except Exception as e:
print(f"Error: {e}")
```

test.py

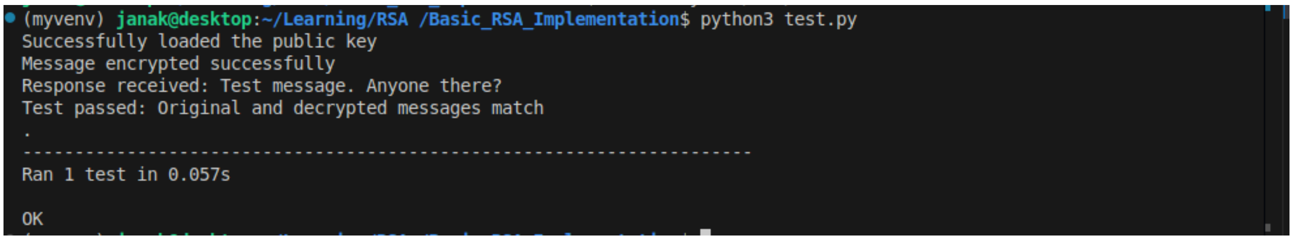
```
import requests
import os
import rsa
import json

# Disable proxy for localhost
os.environ['NO_PROXY'] = '127.0.0.1'

# Fetch public key from server
try:
response = requests.get('http://127.0.0.1:8080/getkey') # Changed port to 8080 to match server
# Parse the jsonpickle response correctly
decoded_response = json.loads(response.content)
public_key_raw = decoded_response
# Load the public key properly
public_key = rsa.PublicKey.load_pkcs1(public_key_raw)
print("Successfully loaded the public key from server")
except Exception as e:
print(f"Error fetching or parsing the public key: {e}")
exit(1)

# Loop to send encrypted messages
while True:
try:
user_input = input("Enter message (or type 'exit' to quit):\n")
if user_input.lower() == 'exit':
print("Exiting...")
break
# Encrypt the message using the server's public key
encrypted_msg = rsa.encrypt(user_input.encode('utf-8'), public_key)
print(f"Message encrypted successfully")
# Send the encrypted message to the server
server_response = requests.post('http://127.0.0.1:8080/message', data=encrypted_msg)
# Handle the server response
if server_response.status_code == 200:
print(f"Server Response: {server_response.content.decode()}\n")
else:
print(f"Error: Server returned status code {server_response.status_code}")
except Exception as e:
print(f"Error: {e}")
```

Output:



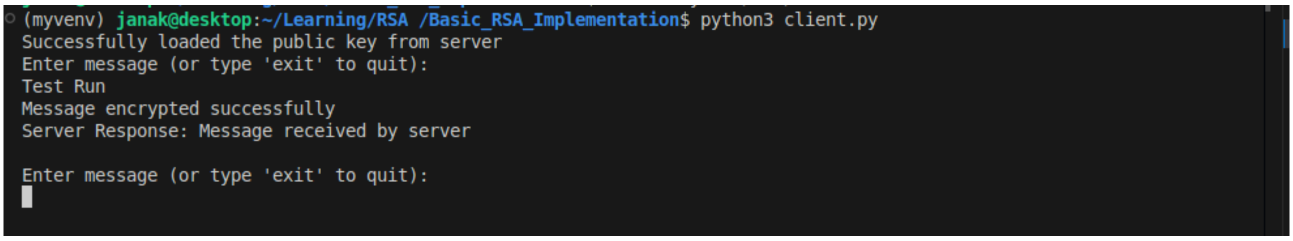
```
○ (myvenv) janak@desktop:~/Learning/RSA /Basic_RSA_Implementation$ python3 server.py
  Generating new keys...
  Keys generated successfully.
   * Serving Flask app 'server'
   * Debug mode: on
  WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
   * Running on http://127.0.0.1:8080
  Press CTRL+C to quit
```

server initialization – server running – key generation successful



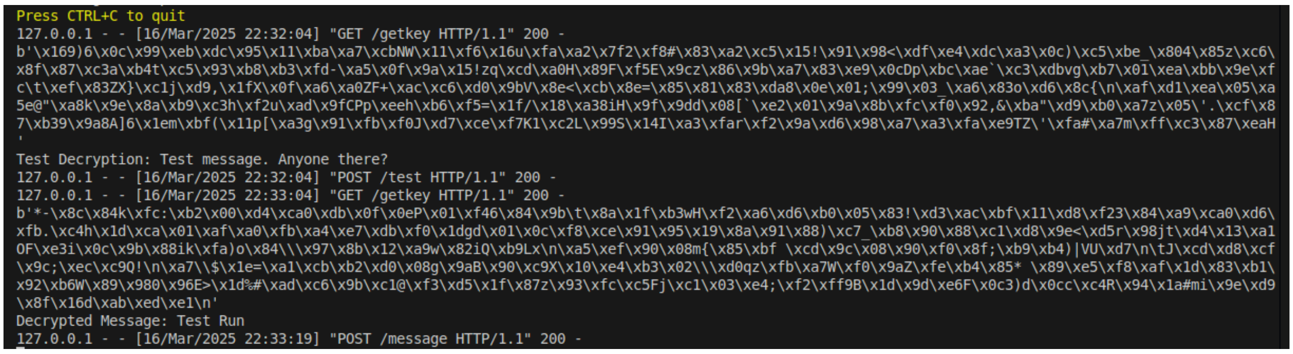
```
● (myvenv) janak@desktop:~/Learning/RSA /Basic_RSA_Implementation$ python3 test.py
  Successfully loaded the public key
  Message encrypted successfully
  Response received: Test message. Anyone there?
  Test passed: Original and decrypted messages match
  .
  ----------------------------------------------------------------------
  Ran 1 test in 0.057s

  OK
```

successful completion of testing with required test cases



```
○ (myvenv) janak@desktop:~/Learning/RSA /Basic_RSA_Implementation$ python3 client.py
  Successfully loaded the public key from server
  Enter message (or type 'exit' to quit):
  Test Run
  Message encrypted successfully
  Server Response: Message received by server

  Enter message (or type 'exit' to quit):
```

client.py functioning with message being encrypted and sent to the server



```
Press CTRL+C to quit
127.0.0.1 - - [16/Mar/2025 22:32:04] "GET /getkey HTTP/1.1" 200 -
b'\x169)6\x0c\x99\xeb\xdc\x95\x11\xba\xa7\xcbNW\x11\xf6\x16u\xfa\xa2\x7f2\xf8#\x83\xa2\xc5\x15!\x91\x98<\xdf\xe4\xdc\xa3\x0c)\xc5\xbe_\x804\x85z\xc6\
x8f\x87\xc3a\xb4t\xc5\x93\xb8\xb3\xfd-\xa5\x0f\x9a\x15!zq\xcd\xa0H\x89F\xf5E\x9cz\x86\x9b\xa7\x83\xe9\x0cDp\xbc\xae`\xc3\xdbvg\xb7\x01\xea\xbb\x9e\xf
c\t\xef\x83ZX}\xc1j\xd9,\x1fX\x0f\xa6\xa0ZF+\xac\xc6\xd0\x9bV\x8e<\xcb\x8e=\x85\x81\x83\xda8\x0e\x01;\x99\x03_\xa6\x83o\xd6\x8c{\n\xaf\xd1\xea\x05\xa
5e@"\xa8k\x9e\x8a\xb9\xc3h\xf2u\xad\x9fCPp\xeeh\xb6\xf5=\x1f/\x18\xa38iH\x9f\x9dd\x08[`\xe2\x01\x9a\x8b\xfc\xf0\x92,&\xba"\xd9\xb0\xa7z\x05\'.\xcf\x8
7\xb39\x9a8A]6\x1em\xbf(\x11p[\xa3g\x91\xfb\xf0J\xd7\xce\xf7K1\xc2L\x99S\x14I\xa3\xfar\xf2\x9a\xd6\x98\xa7\xa3\xfa\xe9TZ\'\xfa#\xa7m\xff\xc3\x87\xeaH
'
Test Decryption: Test message. Anyone there?
127.0.0.1 - - [16/Mar/2025 22:32:04] "POST /test HTTP/1.1" 200 -
127.0.0.1 - - [16/Mar/2025 22:33:04] "GET /getkey HTTP/1.1" 200 -
b'*-\x8c\x84k\xfc:\xb2\x00\xd4\xca0\xdb\x0f\x0eP\x01\xf46\x84\x9b\t\x8a\x1f\xb3wH\xf2\xa6\xd6\xb0\x05\x83!\xd3\xac\xbf\x11\xd8\xf23\x84\xa9\xca0\xd6\
xfb.\xc4h\x1d\xca\x01\xaf\xa0\xfb\xa4\xe7\xdb\xf0\x1dgd\x01\x0c\xf8\xce\x91\x95\x19\x8a\x91\x88)\xc7_\xb8\x90\x88\xc1\xd8\x9e<\xd5r\x98jt\xd4\x13\xa1
OF\xe3i\x0c\x9b\x88ik\xfa)o\x84\\\x97\x8b\x12\xa9w\x82iQ\xb9Lx\n\xa5\xef\x90\x08m{\x85\xbf \xcd\x9c\x08\x90\xf0\x8f;\xb9\xb4)|VU\xd7\n\tJ\xcd\xd8\xcf
\x9c;\xec\xc9Q!\n\xa7\\$\x1e=\xa1\xcb\xb2\xd0\x08g\x9aB\x90\xc9X\x10\xe4\xb3\x02\\\xd0qz\xfb\xa7W\xf0\x9aZ\xfe\xb4\x85* \x89\xe5\xf8\xaf\x1d\x83\xb1\
x92\xb6W\x89\x980\x96E>\x1d%#\xad\xc6\x9b\xc1@\xf3\xd5\x1f\x87z\x93\xfc\xc5Fj\xc1\x03\xe4;\xf2\xff9B\x1d\x9d\xe6F\x0c3)d\x0cc\xc4R\x94\x1a#mi\x9e\xd9
\x8f\x16d\xab\xed\xe1\n'
Decrypted Message: Test Run
127.0.0.1 - - [16/Mar/2025 22:33:19] "POST /message HTTP/1.1" 200 -
```

successful functioning of encryption algorithm

Github: https://github.com/sitjan/Basic_RSA_Implementation

Conclusion:

The RSA implementation presented demonstrates a functional representation for secure message exchange. We consider balancing the security, usability and performance issues. We generate cryptographically secure keys, safely distributing the public key, and encrypt the message on the client side, and decrypting them securely on the server.