Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

LAB 1

[Code No: COMP 492]

Submitted by

Paribartan Timalsina

Roll no: 56

Group: Computer Engineering

Level: IV year/II sem

Submitted to

Mr. Suresh Gautam

Department of Computer Science and Engineering

Assignment-1

Problem: Secure communication between a client and a server

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.

Deliverables:
- The implementation of the AES or RSA encryption algorithm in Python.
- A brief report explaining the design choices made in the implementation and the trade-offs between different options.
- A test plan and test cases to show that the implementation is working correctly.

1 Introduction

The report presents the implementation of a secure communication system between client and server using the **AES(Advanced Encryption Standard)** and **RSA(Rivest-Shamir-Adleman)** encryption. The aim of this lab work is to show the process of secure message transmission between the client and server using the public and private keys and how the information is exchanged between two parties. This lab work includes the process of encryption, decryption and key exchange.

AES (Advanced Encryption Standard) is a symmetric encryption algorithm that encrypts and decrypts data using the same key. It is a block cipher that processes data in fixed-size blocks of 128 bits and supports key sizes of 128, 192, and 256 bits, with larger keys providing stronger security. AES is widely used for securing bulk data in applications such as file encryption, VPNs, and TLS/SSL due to its high speed and resistance to brute-force attacks. The encryption process involves multiple rounds of substitution, permutation, mixing, and key addition, making AES both efficient and secure for high-throughput environments.

On the other hand, **RSA (Rivest-Shamir-Adleman)** is an asymmetric encryption algorithm that uses a pair of mathematically related keys: a public key for encryption and a private key for decryption. Unlike AES, RSA relies on the computational difficulty of factoring large prime numbers, making it secure but significantly slower than symmetric encryption. RSA is commonly used for digital signatures, secure key exchange in TLS/SSL, and email encryption, ensuring data integrity and authentication. While RSA is not typically used for bulk encryption due to its computational overhead, it plays a crucial role in securing communications by facilitating the secure exchange of symmetric encryption keys like those used in AES.

2 Implementation of Details

2.1 Technologies Used

Programming Language: Python 3

2.2 Libraries:

Flask- For server-side communication
Pycryptodome- For RSA and AES encryption
Jsonpickle- For object serialization

3 Server Implementation

The server is implemented using the flask server. During the server implementation at first we have generated RSA keys i.e both public and private keys for asymmetric encryption. The private key remains secure in the server whereas the public key is secured with clients. Then the client generates random AES keys of 128 bits that are encrypted using RSA before sending it to the server. The client then sends both the encrypted message(encrypted using AES implementation) and the encrypted AES(encrypted using RSA) to the server. The server decrypts the encrypted AES key using the private RSA key. After getting the decrypted form of AES keys that is then further used to decrypt the message using the help of decrypted AES key. Doing this work ensures that the server is working correctly and all the works are performed securely. The code implementation for the server side algorithm is is given as:

```
def generate_keys():
    """Generate RSA key pair if not already present."""
    if not os.path.exists('private_key.pem') or not os.path.exists('public_key.pem'):
        key = RSA.generate(2048)
        private_key = key.export_key()
        public_key = key.publickey().export_key()
        with open('private_key.pem', 'wb') as file:
            file.write(private_key)
        with open('public_key.pem', 'wb') as file:
            file.write(public_key)
        print("Keys generated")
    else:
        print("Keys already exist")

generate_keys()



@app.route("/decrypt_message", methods=['POST'])
def decrypt_message_api():
    """Decrypt AES-encrypted message using the stored private key."""
    with open('private_key.pem', 'rb') as file:
        private_key = RSA.import_key(file.read())

    data = request.get_json()
    encrypted_message = data['encrypted_message']
    encrypted_aes_key = data['encrypted_aes_key']
```

```
    aes_key = decrypt_aes_key(encrypted_aes_key, private_key)
    decrypted_message = decrypt_message(encrypted_message, aes_key)

    return jsonify({"decrypted_message": decrypted_message})

def decrypt_aes_key(encrypted_key, rsa_private_key):
    """Decrypt AES key using RSA private key."""
    cipher_rsa = PKCS1_OAEP.new(rsa_private_key)
    decrypted_key = cipher_rsa.decrypt(bytes.fromhex(encrypted_key))
    return decrypted_key

def decrypt_message(encrypted_message, aes_key):
    """Decrypt message using AES."""
    encrypted_data = bytes.fromhex(encrypted_message)
    nonce, tag, ciphertext = encrypted_data[:16], encrypted_data[16:32], encrypted_data[32:]
    cipher_aes = AES.new(aes_key, AES.MODE_EAX, nonce=nonce)
    decrypted_message = cipher_aes.decrypt_and_verify(ciphertext, tag)
    return decrypted_message.decode('utf-8')

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

4 Client implementation

In the client side implementation the message of the client is taken and then client generates a random AES key for secure symmetric encryption..The AES key is encrypted using the server's obtained public RSA key before transmission and the messages are encrypted using AES before sending it to the server. The client side code is given as:

```
def encrypt_aes_key(aes_key, public_key_pem):
    """Encrypt AES key using the server's RSA public key."""
    public_key = RSA.import_key(public_key_pem)
    cipher_rsa = PKCS1_OAEP.new(public_key)
    encrypted_key = cipher_rsa.encrypt(aes_key)
    return encrypted_key.hex()
```

```
def encrypt_message(message, aes_key):
    """Encrypt a message using AES encryption."""
    cipher_aes = AES.new(aes_key, AES.MODE_EAX)
    ciphertext, tag = cipher_aes.encrypt_and_digest(message.encode('utf-8'))
    encrypted_data = cipher_aes.nonce + tag + ciphertext
    return encrypted_data.hex()

def send_encrypted_data(encrypted_message, encrypted_aes_key):
    """Send encrypted AES key and message to the server for decryption."""
    payload = {
        "encrypted_message": encrypted_message,
        "encrypted_aes_key": encrypted_aes_key
    }
    response = requests.post(f"{SERVER_URL}/decrypt_message", json=payload)
    return response.json()
```

5 Testing the Implementation

The test cases were written for three conditions. The first test is to look if the keys are exchanged properly or not and the second test case is written to ensure the correct message is obtained when passing the correct encrypted key and message to the server and the third one is to find out if the server correctly handles the invalid keys. From test cases it was obtained that sever successfully retrieves the public key. The message encrypts and decrypts correctly and the server handles invalid AES keys appropriately.

The code for test implementation is:

```
class SecureCommunicationTests(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        """Fetch server's public key once before running tests."""
        response = requests.get(f"{SERVER_URL}/send_key")
        cls.server_public_key = jsonpickle.decode(response.text).encode("utf-8")
        cls.aes_key = get_random_bytes(16)  # Generate AES key (128-bit)

    def encrypt_aes_key(self, aes_key, public_key_pem):
        """Helper method to encrypt the AES key using RSA public key."""
```

```
    public_key = RSA.import_key(public_key_pem)
    cipher_rsa = PKCS1_OAEP.new(public_key)
    return cipher_rsa.encrypt(aes_key).hex()

def encrypt_message(self, message, aes_key):
    """Helper method to encrypt messages using AES."""
    cipher_aes = AES.new(aes_key, AES.MODE_EAX)
    ciphertext, tag = cipher_aes.encrypt_and_digest(message.encode("utf-8"))
    encrypted_data = cipher_aes.nonce + tag + ciphertext
    return encrypted_data.hex()

def test_1_server_key_retrieval(self):
    """Test if the server's public key is retrievable and valid."""
    self.assertTrue(self.server_public_key.startswith(b"-----BEGIN PUBLIC KEY-----"))

def test_2_encryption_decryption(self):
    """Test if an encrypted message is correctly decrypted by the server."""
    original_message = "Hello, This is the secure server!"
    encrypted_aes_key = self.encrypt_aes_key(self.aes_key, self.server_public_key)
    encrypted_message = self.encrypt_message(original_message, self.aes_key)

    payload = {
        "encrypted_message": encrypted_message,
        "encrypted_aes_key": encrypted_aes_key
    }
    response = requests.post(f"{SERVER_URL}/decrypt_message", json=payload)

    self.assertEqual(response.status_code, 200)
    decrypted_message = response.json().get("decrypted_message", "")
    self.assertEqual(decrypted_message, original_message)

def test_3_invalid_aes_key(self):
    """Test server's response to an invalid AES key (wrong decryption)."""
    wrong_aes_key = get_random_bytes(16)  # Generate a random wrong key
    encrypted_message = self.encrypt_message("Hello, Secure Server!", wrong_aes_key)
    encrypted_aes_key = self.encrypt_aes_key(self.aes_key, self.server_public_key)

    payload = {
        "encrypted_message": encrypted_message,
```

```
    "encrypted_aes_key": encrypted_aes_key
}
response = requests.post(f"{SERVER_URL}/decrypt_message", json=payload)

self.assertEqual(response.status_code, 500)  # Expect a failure due to invalid decryption
```

6 Results:

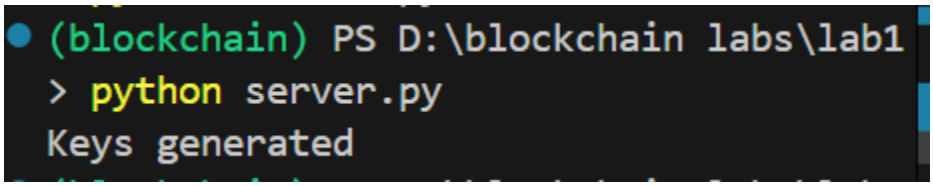

Figure: Generating the RSA keys

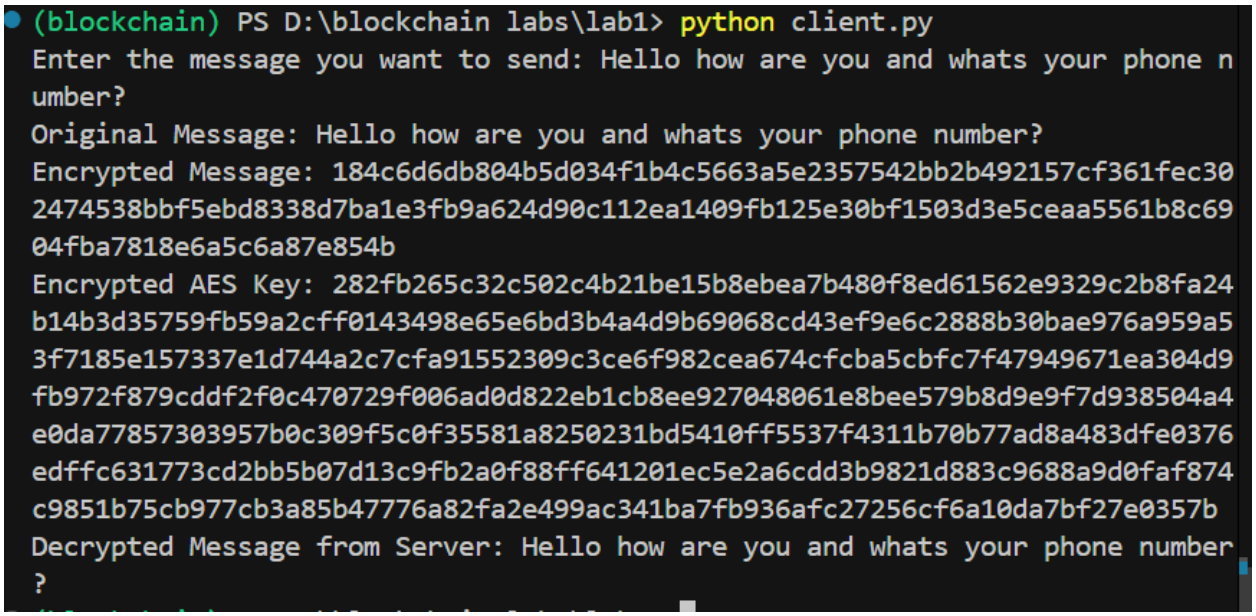

Figure : Passing the encrypted key and message and getting the decrypted message from server

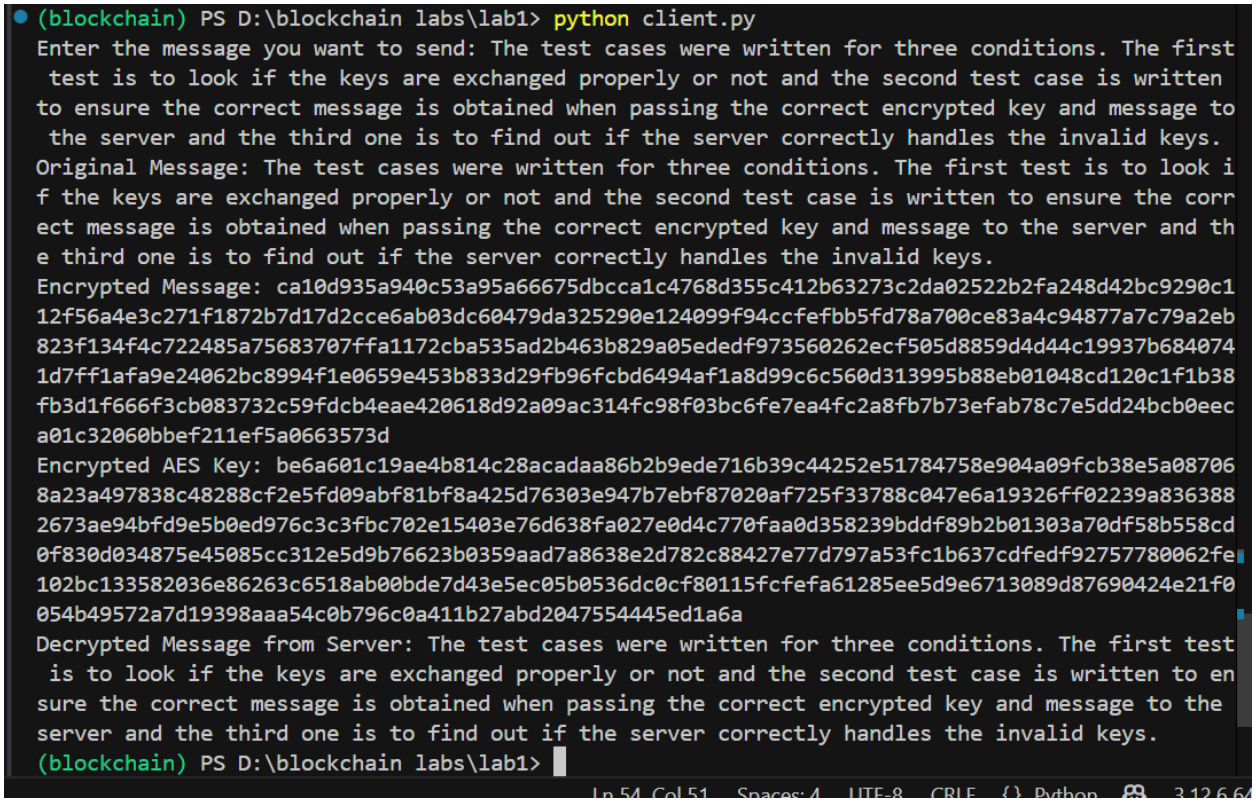

Figure: Passing the encrypted key and long message and getting the decrypted message from server

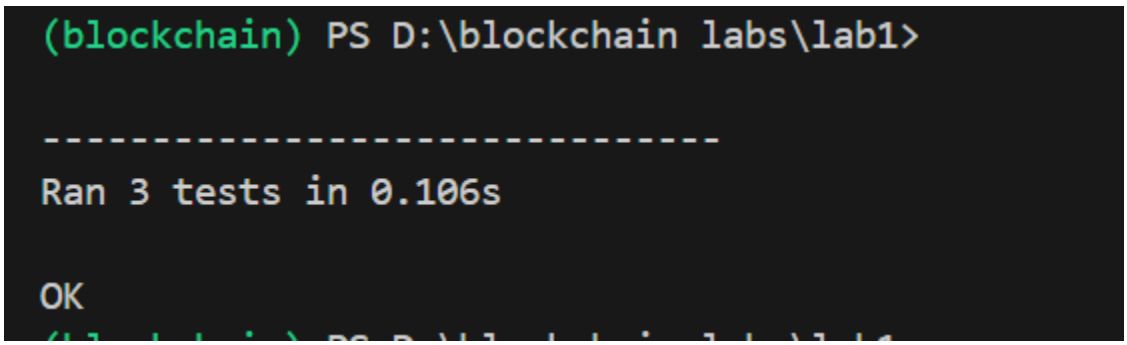

Figure: Running 3 test cases successfully

7 Trade-offs and Design Choices

AES vs RSA for message encryption
AES is preferred for encrypting large messages due to its speed and efficiency, whereas RSA is primarily used for securely exchanging the AES key, as its computational cost makes it impractical for encrypting large amounts of data.

Key Size comparison
In terms of key size, AES-128 is chosen as it provides a balance between security and performance, while RSA-2048 ensures strong security for key exchange.

Mode of operation for AES
For the mode of operation, AES is used in EAX mode, which was selected for its built-in integrity checks, offering both encryption and authentication to ensure data confidentiality and integrity.

8 Conclusion

The implementation of code provided a secure and efficient way to exchange the message between client and server using the combination of RSA for key exchange and AES for message encryption. The test cases validated the correctness of encryption and decryption ensuring reliability or work. Hence by combining the RSA and AES the system ensures secure, scalable and efficient communication between a client and a server.

The complete code is available in the repo:
https://github.com/Paribartan-Timalsina/Blockchain-Labs