Lab Report: Implementation of RSA Algorithm with TCP Client/Server Sockets in Python

Purpose

The purpose of this lab is to:

1. Write a program to implement the RSA algorithm with a key size of 1024 bits.
2. Demonstrate the encryption and decryption process using a Client/Server application over TCP. The server generates the public and private keys and sends the public key to the client via a TCP socket. The client uses the public key to encrypt a message and send the ciphertext back to the server for decryption.

Methods

The implementation consists of two main classes: Server and Client.

- **Server**:
  - Generates RSA public and private keys.
  - Sends the public key to the client through a TCP socket.
  - Receives ciphertext from the client and decrypts it using the private key.
- **Client**:
  - Receives the public key from the server through a TCP socket.
  - Encrypts a message using the received public key and sends the ciphertext to the server.

Steps:

1. **Key Generation**:
   - The server generates two large prime numbers `p` and `q`.
   - Computes `n = p * q` and the Euler's totient function `φ(n) = (p-1)*(q-1)`.
   - Chooses a public exponent `e = 65537` and calculates the private exponent `d` such that `e * d ≡ 1 (mod φ(n))`.
2. **Encryption and Decryption**:
   - To encrypt a message `M` with the public key, compute the ciphertext `C` using `C = M^e mod n`.
   - To decrypt the ciphertext `C` with the private key, compute the plaintext `M` using `M = C^d mod n`.

Implementation

The code below demonstrates the Server and Client implementation in Python using sockets and the RSA algorithm.

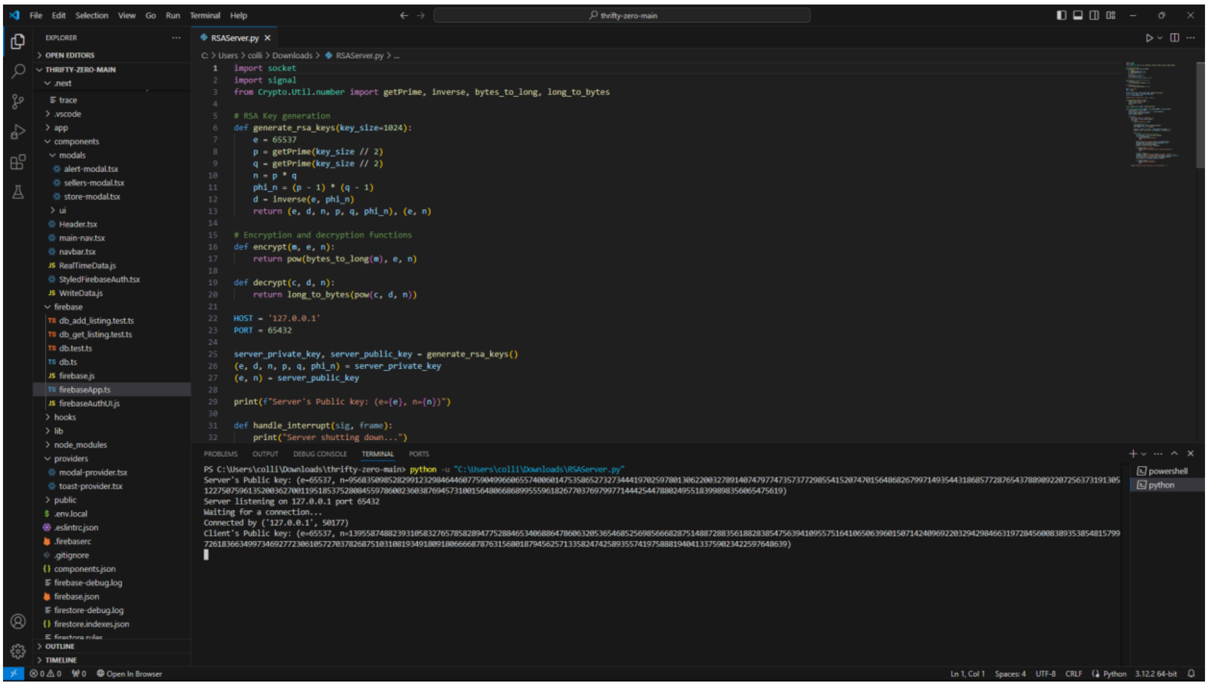
Server Code(Alice)

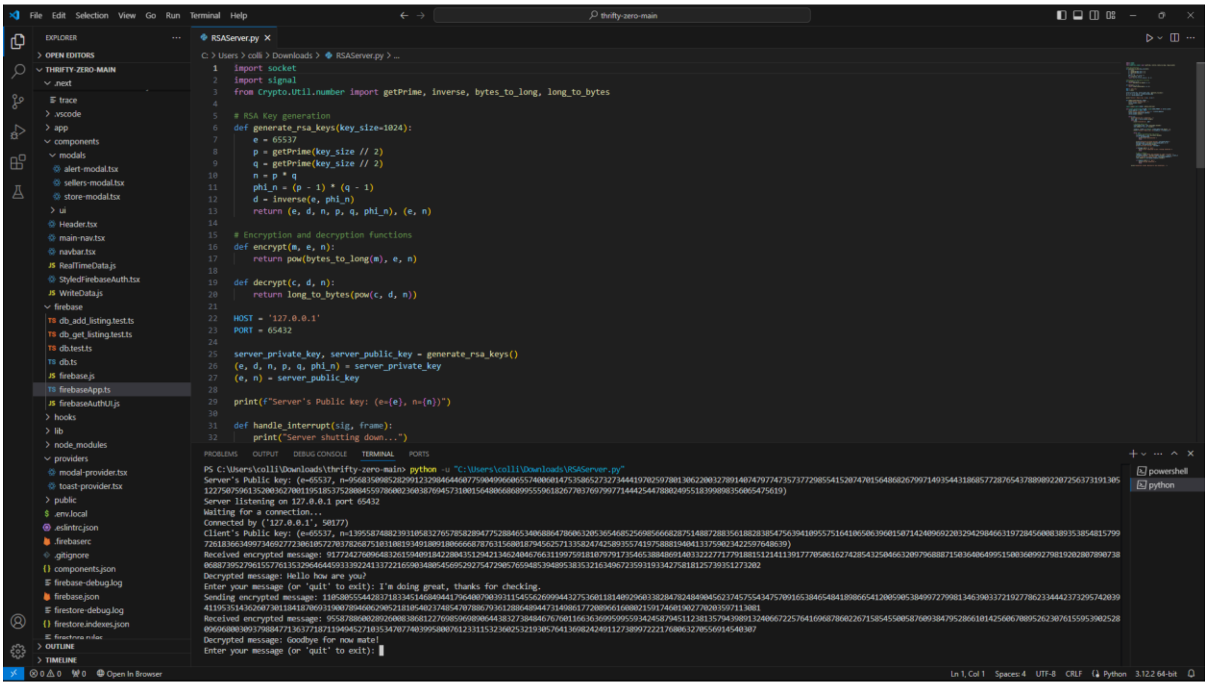
```
python
Copy code
import socket
import signal
```

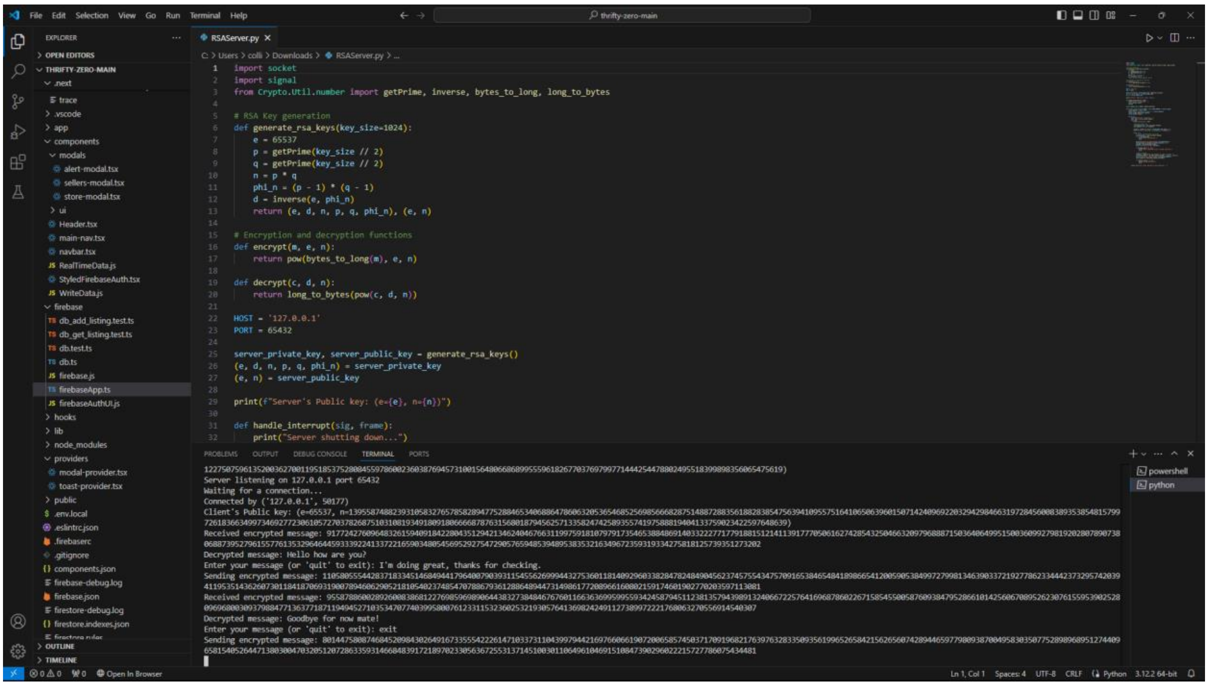
```python
from Crypto.Util.number import getPrime, inverse, bytes_to_long,
long_to_bytes

# RSA Key generation
def generate_rsa_keys(key_size=1024):
    e = 65537
    p = getPrime(key_size // 2)
    q = getPrime(key_size // 2)
    n = p * q
    phi_n = (p - 1) * (q - 1)
    d = inverse(e, phi_n)
    return (e, d, n, p, q, phi_n), (e, n)

# Encryption and decryption functions
def encrypt(m, e, n):
    return pow(bytes_to_long(m), e, n)

def decrypt(c, d, n):
    return long_to_bytes(pow(c, d, n))

HOST = '127.0.0.1'
PORT = 65432

server_private_key, server_public_key = generate_rsa_keys()
(e, d, n, p, q, phi_n) = server_private_key
(e, n) = server_public_key

print(f"Server's Public key: (e={e}, n={n})")

def handle_interrupt(sig, frame):
    print("Server shutting down...")
    server_socket.close()
    exit(0)

signal.signal(signal.SIGINT, handle_interrupt)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
    server_address = (HOST, PORT)
    print('Server listening on %s port %s' % server_address)
    server_socket.bind(server_address)
    server_socket.listen()

    while True:
        print("Waiting for a connection...")
        conn, addr = server_socket.accept()
        with conn:
            print('Connected by', addr)

            # Exchange public keys
            client_public_key = conn.recv(1024).decode()
            conn.sendall(f"{e},{n}".encode())

            client_e, client_n = map(int, client_public_key.split(','))
            print(f"Client's Public key: (e={client_e}, n={client_n})")

            while True:
                # Receive encrypted data from client
                encrypted_data = conn.recv(1024).decode()
                if not encrypted_data:
                    print("Client disconnected.")
                    break
```

```
            print(f"Received encrypted message: {encrypted_data}")
            decrypted_data = decrypt(int(encrypted_data), d, n)
            message = decrypted_data.decode()
            print(f"Decrypted message: {message}")

            if message.lower() == 'quit':
                print("Client requested to quit. Closing connection.")
                break

            # Send a response
            response = input("Enter your message (or 'quit' to exit):
")
            encrypted_response = encrypt(response.encode(), client_e,
client_n)
            print(f"Sending encrypted message: {encrypted_response}")
            conn.sendall(str(encrypted_response).encode())

            if response.lower() == 'quit':
                print("Closing connection.")
                break

        print("Connection closed. Waiting for new connections...")
```

Server Output







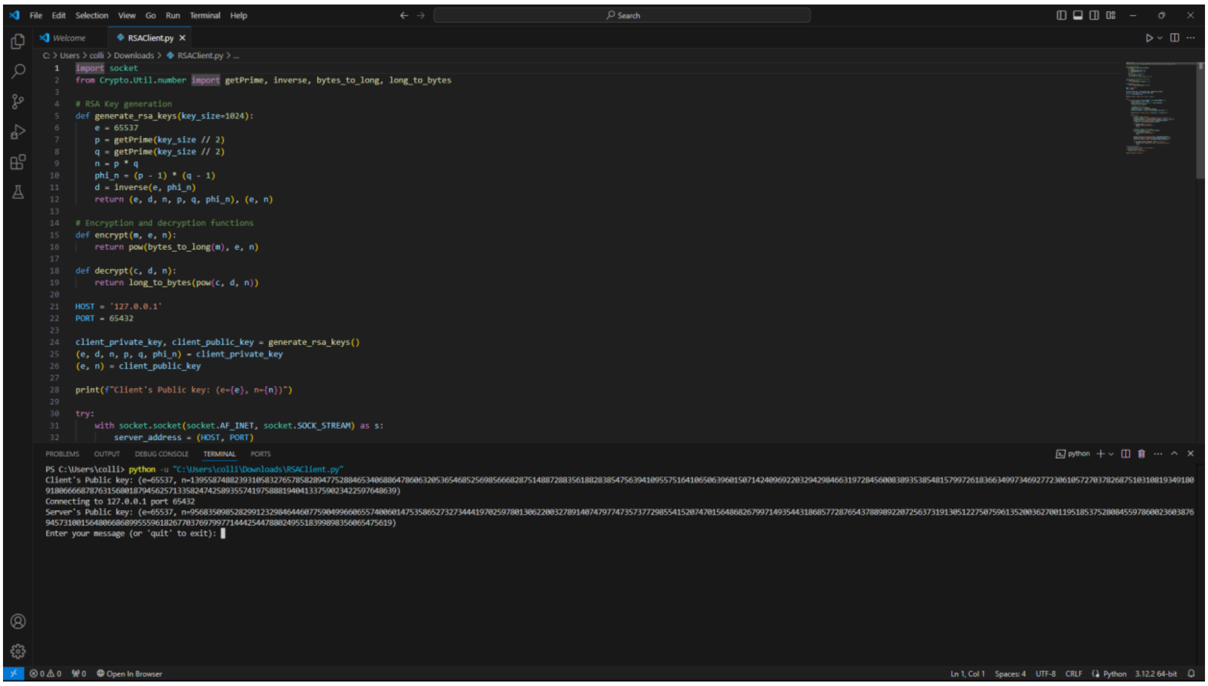
Client Code(Bob)

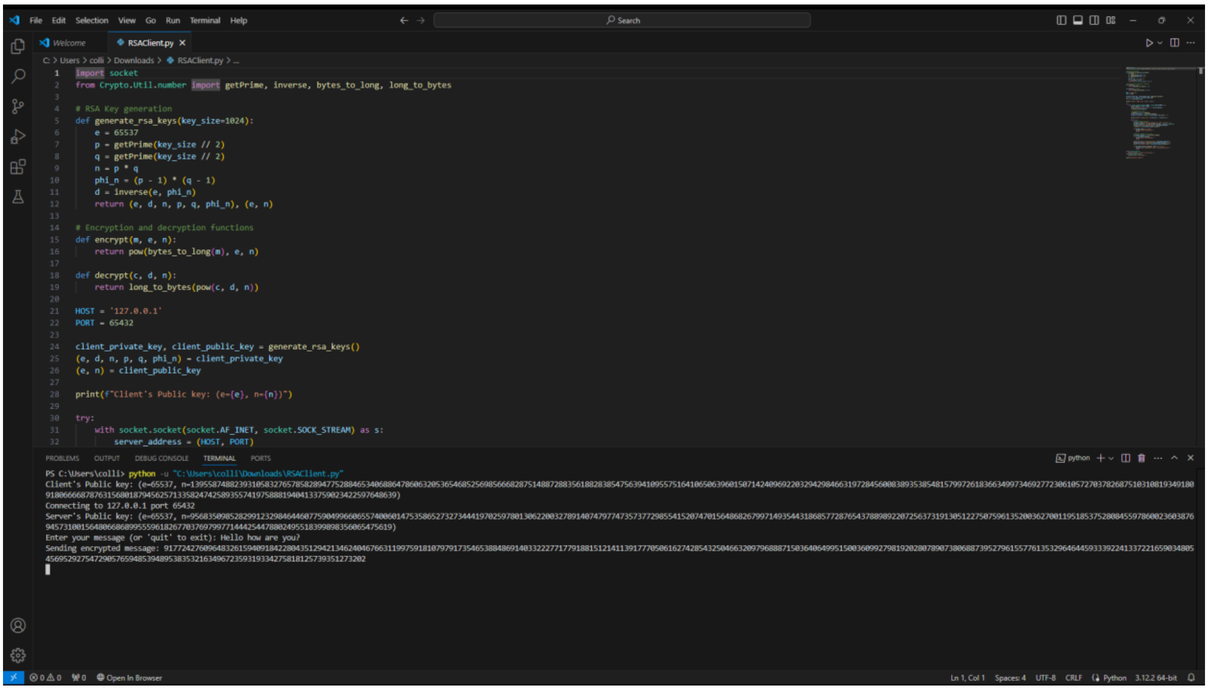
```
python
Copy code
import socket
from Crypto.Util.number import getPrime, inverse, bytes_to_long,
long_to_bytes

# RSA Key generation
def generate_rsa_keys(key_size=1024):
    e = 65537
    p = getPrime(key_size // 2)
```

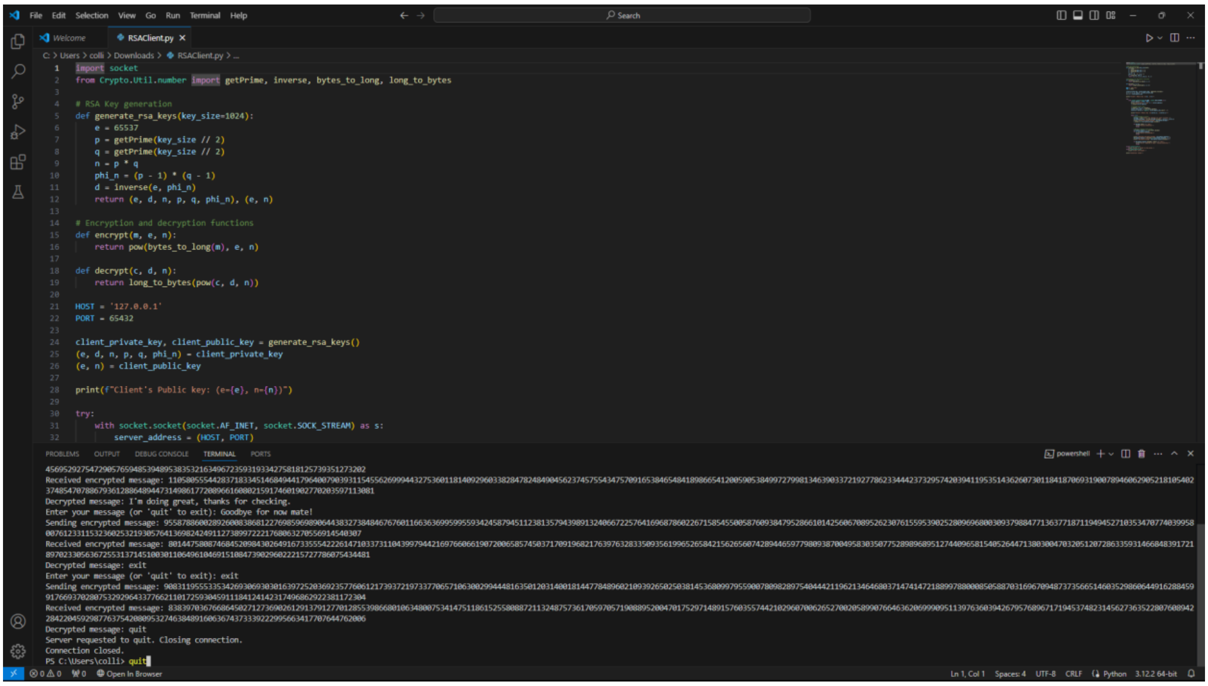
```
    q = getPrime(key_size // 2)
    n = p * q
    phi_n = (p - 1) * (q - 1)
    d = inverse(e, phi_n)
    return (e, d, n, p, q, phi_n), (e, n)

# Encryption and decryption functions
def encrypt(m, e, n):
    return pow(bytes_to_long(m), e, n)

def decrypt(c, d, n):
    return long_to_bytes(pow(c, d, n))

HOST = '127.0.0.1'
PORT = 65432

client_private_key, client_public_key = generate_rsa_keys()
(e, d, n, p, q, phi_n) = client_private_key
(e, n) = client_public_key

print(f"Client's Public key: (e={e}, n={n})")

try:
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        server_address = (HOST, PORT)
        print('Connecting to %s port %s' % server_address)
        s.connect(server_address)

        # Exchange public keys
        s.sendall(f"{e},{n}".encode())
        server_public_key = s.recv(1024).decode()
        server_e, server_n = map(int, server_public_key.split(','))

        print(f"Server's Public key: (e={server_e}, n={server_n})")

        while True:
            # Send message to server
            message = input("Enter your message (or 'quit' to exit): ")
            encrypted_message = encrypt(message.encode(), server_e,
server_n)
            print(f"Sending encrypted message: {encrypted_message}")
            s.sendall(str(encrypted_message).encode())

            if message.lower() == 'quit':
                print("Closing connection.")
                break

            # Receive response from server
            encrypted_response = s.recv(1024).decode()
            if not encrypted_response:
                print("Server disconnected.")
                break

            print(f"Received encrypted message: {encrypted_response}")
            decrypted_response = decrypt(int(encrypted_response), d, n)
            print(f"Decrypted message: {decrypted_response.decode()}")

            if decrypted_response.decode().lower() == 'quit':
                print("Server requested to quit. Closing connection.")
                break
```

```
except ConnectionError:
    print('Failed to connect to the server.')
except KeyboardInterrupt:
    print('Client interrupted.')

print("Connection closed.")
```

Client Output







Conclusion

This lab demonstrated the implementation of the RSA algorithm and the establishment of a TCP Client/Server communication for encrypted message exchange. The server generates RSA keys, sends the public key to the client, and decrypts messages received from the client. The client uses the server's public key to encrypt messages and sends the ciphertext to the server. This approach ensures secure communication over the network using RSA encryption.