## Overview of the Program Design

1. **Key Features**:
    - I. Secure key exchange using RSA encryption.
    - II. Symmetric encryption of messages and nonce verification with AES.
    - III. Modular functions for encryption, decryption, and socket communication.
2. **Flow**:
    - I. The **server** generates RSA public-private key pairs based on user-supplied prime numbers (p, q) and sends the public key (n, e) and a random nonce to the client.
    - II. The **client**:
        - ▪ Generates a random symmetric key.
        - ▪ Encrypts the symmetric key using the RSA public key and sends it to the server.
        - ▪ Encrypts the nonce using the symmetric key and sends it for verification.
    - o The **server** verifies the nonce and sends confirmation back to the client.

**How It Works**

1. **Server**:
    - I. Validates user-supplied prime numbers and computes RSA keys:
        - ▪ $n = p \times q$
        - ▪ Public key: (n, e)
        - ▪ Private key: d (calculated as the modular inverse of e mod $\phi(n)$).
    - II. Sends public key and a random nonce to the client.
2. **Client**:
    - I. Establishes a socket connection and receives the server's public key and nonce.
    - II. Encrypts the symmetric key using RSA: Ciphertext = $(\text{Symmetric Key})^e \bmod n$
    - III. Encrypts the nonce using the symmetric key and sends both encrypted values to the server.
3. **Server Verification**:
    - I. Decrypts the symmetric key using its private RSA key.
    - II. Decrypts the nonce using the symmetric key and verifies its correctness.

**Design Tradeoffs**

1. **Simplicity vs. Security**:
    - I. **Choice**: Simplified RSA and AES implementations were used for educational purposes.
    - II. **Tradeoff**: Reduces complexity but is not secure for real-world implementaion as part c shows.
2. **Error Handling**:
    - I. **Choice**: Basic error handling for invalid inputs or connection issues.
    - II. **Tradeoff**: Focuses on core functionality but risks failure in edge cases.
3. **Message Parsing**:
    - I. **Choice**: Hardcoded parsing of server messages.
    - II. **Tradeoff**: Simplifies communication but limits flexibility for format changes.

**Improvements and Extensions**

1. **Enhanced Security**:
    - I. Replace simplified RSA and AES with standard libraries (e.g., pycryptodome).
    - II. Use larger key sizes for real-world cryptographic strength.
2. **Scalability**:
    - I. Add threading or asynchronous I/O to handle multiple clients simultaneously.
3. **Protocol Robustness**:
    - I. Implement mutual authentication where both client and server verify each other's identity.
    - II. Extend the protocol to encrypt subsequent communication using the exchanged symmetric key.
4. **Detailed Error Handling**:
    - I. Improve error messages and implement retry mechanisms for failed connections or mismatched data.

## Testing and Validation

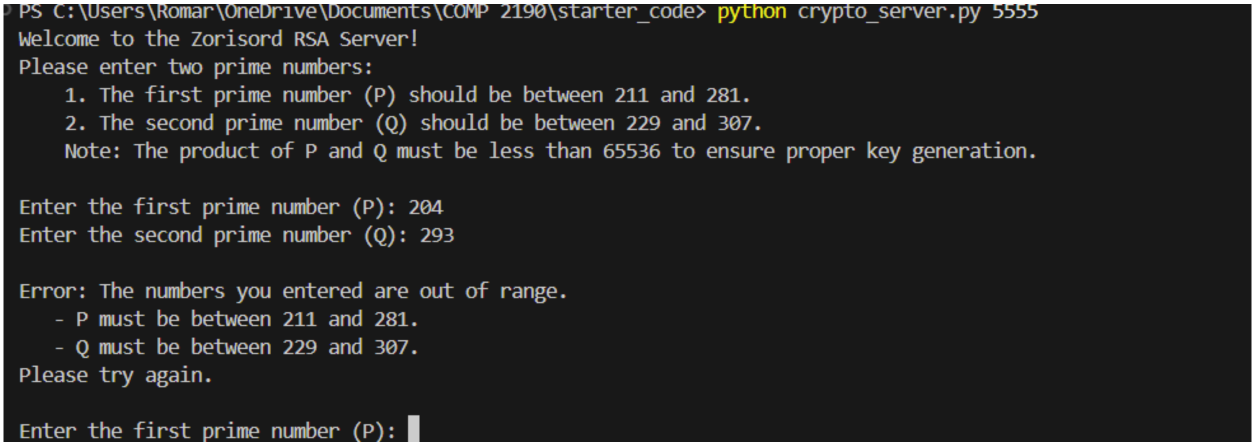
### Tests Conducted

1. **Prime Validation**:
    - ➢ Checked the server correctly accepts valid primes and rejects invalid or out-of-range inputs.
2. **Key Exchange**:
    - ➢ Verified the client correctly encrypts the symmetric key using RSA.
    - ➢ Tested that the server successfully decrypts and retrieves the symmetric key.
3. **Nonce Verification**:
    - ➢ Ensured the server correctly decrypts and verifies the nonce using the symmetric key.
    - ➢ Tested incorrect nonce decryption and observed appropriate error responses.
4. **Connection Handling**:
    - ➢ Simulated disconnections and invalid inputs to verify error handling.

### Known Limitations

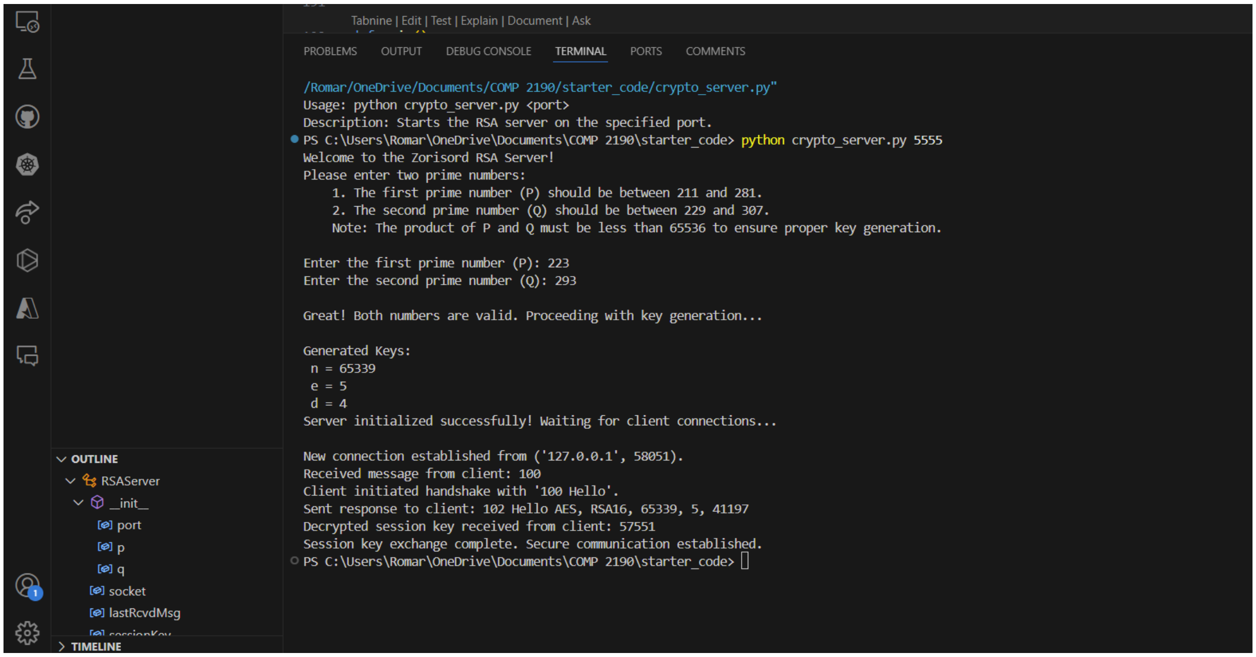
1. **Security Weakness**:
    - ➢ Simplified RSA and small key sizes are vulnerable to attacks.
2. **Edge Cases**:
    - ➢ Does not handle malformed messages or extremely large inputs gracefully.
3. **Static Message Parsing**:
    - ➢ Client relies on hardcoded formats, making it less adaptable to protocol changes.

Different Errors handled by the server:



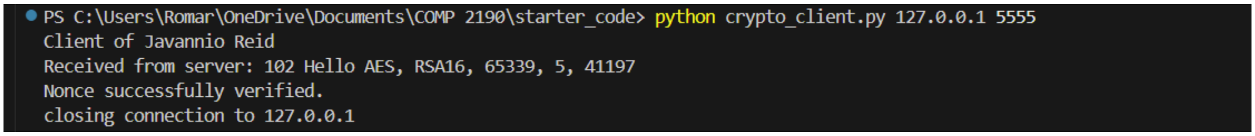
```
PS C:\Users\Romar\OneDrive\Documents\COMP 2190\starter_code> python crypto_server.py 5555
Welcome to the Zorisord RSA Server!
Please enter two prime numbers:
    1. The first prime number (P) should be between 211 and 281.
    2. The second prime number (Q) should be between 229 and 307.
    Note: The product of P and Q must be less than 65536 to ensure proper key generation.

Enter the first prime number (P): 204
Enter the second prime number (Q): 293

Error: The numbers you entered are out of range.
   - P must be between 211 and 281.
   - Q must be between 229 and 307.
Please try again.

Enter the first prime number (P): 
```

Server operating as expected:



```
/Romar/OneDrive/Documents/COMP 2190/starter_code/crypto_server.py"
Usage: python crypto_server.py <port>
Description: Starts the RSA server on the specified port.
PS C:\Users\Romar\OneDrive\Documents\COMP 2190\starter_code> python crypto_server.py 5555
Welcome to the Zorisord RSA Server!
Please enter two prime numbers:
    1. The first prime number (P) should be between 211 and 281.
    2. The second prime number (Q) should be between 229 and 307.
    Note: The product of P and Q must be less than 65536 to ensure proper key generation.

Enter the first prime number (P): 223
Enter the second prime number (Q): 293

Great! Both numbers are valid. Proceeding with key generation...

Generated Keys:
 n = 65339
 e = 5
 d = 4
Server initialized successfully! Waiting for client connections...

New connection established from ('127.0.0.1', 58051).
Received message from client: 100
Client initiated handshake with '100 Hello'.
Sent response to client: 102 Hello AES, RSA16, 65339, 5, 41197
Decrypted session key received from client: 57551
Session key exchange complete. Secure communication established.
PS C:\Users\Romar\OneDrive\Documents\COMP 2190\starter_code> 
```

Response on client-side:



```
PS C:\Users\Romar\OneDrive\Documents\COMP 2190\starter_code> python crypto_client.py 127.0.0.1 5555
Client of Javannio Reid
Received from server: 102 Hello AES, RSA16, 65339, 5, 41197
Nonce successfully verified.
closing connection to 127.0.0.1
```