

# **Design Document**

**ID: 620154561**

**Name: Aisha Forrester**

## **Program Design**

This program is designed to facilitate the interactions between a client and a server via a TCP connection. In the main method of the server, a welcoming socket and a socket dedicated to client communication are created and listening for a client request. When it receives the initial client message, I created a dictionary to not only store the input of the user (prime number  $p$  and generator  $g$ ) but also the random numbers ( $r, x$ ) and the server's internal calculations ( $t, V$ ). The control is then moved to `processMsgs(s, msg, state)` where it will send an appropriate message based on the first string of the `msg` argument. For example, if it receives 100 Hello, it will do its internal calculations and send an appropriate message through the socket (105 Generator + Commitment  $g, p, t, V$ ). The client is designed in the same manner, however, it first sends 100 Hello in the main method. It then receives the 105 Generator + Commitment  $g, p, t, V$  message and splits it up, grabbing the integers  $g, p, t, V$  and creating a random integer  $c$  (between 1 and  $p$ ). These values are then stored in the client-side dictionary for use in `processMsgs(s, msg, state)`, which will work as described above.

## **How it works**

After I binded the TCP server socket to the port, I set the socket into a listening mode where it will be 'waiting' for a client request. A new socket, `conn`, is created to listen to these requests. So the server has a welcoming socket as well as a socket dedicated to communication from the client. The client will send over '100 Hello'. Upon receiving this message, the server will ask the user for a prime number  $p$  between 127 and 7919. It will also ask for a suitable generator  $g$ . It will generate two random numbers of its own ( $x, r$ ) and do internal calculations for variables  $t$  and  $V$  using the  $\text{expMod}(g, r, p)$ . These values are then stored in a dictionary and sent as an argument in the `processMsgs(s, msg, state)`, where it takes the client socket, the '100 Hello' message, and the dictionary. At the `processMsgs(s, msg, state)` function, an if-else statement is used to conclude what actions must be taken depending on the `msg` received. Deciding it is the 100 Hello message, the program then extracts the values  $g, p, x, r$  from the dictionary and prepares a message to send to the client via the `s` socket argument. It processes these integers with the use of the `clientHello` function. The client will then receive this message (105 Generator + Commitment  $g, p, t, V$ ), and use the function `challengeMsg` to generate a value  $c$  and send it over to the server for challenging. When receives this the server will run a function `genChallengeResponse(r, x, c)`, which will do internal computation on  $c$ . The server will then send this message to the client (112 Response B). The client will then run a comparison between the values attained from 2 separate computations and send a 220 Verified message to the server if they match, or a 400 Error to the server if they do not. Both server and client will then close the connection. Otherwise, a 500 Bad

request is sent to either party if something that is outside the constraint is entered into the program.

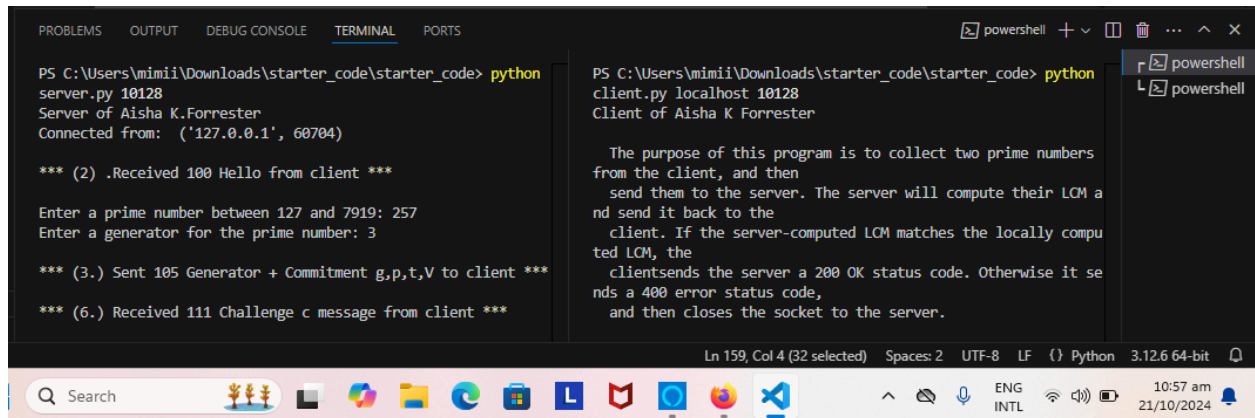
### **Design Trade-offs**

- Security Vs Performance: While security is crucial while handling data from users and protecting the data inside a program, simpler methods were used, however, which does not necessarily add the protection that this program needs.
- Simplicity Vs Flexibility: This program made use of flexibility in terms of allowing the user to choose what prime numbers and generators they would like to employ. The hardcoded values for these variables was not implemented as it would make the program rigid and less reusable

### **Improvements and Extensions**

- I chose to split the message 105 Generator + Commitment  $g,p,t,V$  in the main method before sending it off to the `processMsgs(s,m,dict)` instead of creating a function to do that and then calling that function when needed. This would have made the code “cleaner” and demonstrated good programming but would not affect the functionality of the program.
- When there is a ‘500 Bad Request, the connection should be closed. There is nothing in the Assignment document that alludes to this so my implementation does not reflect it.
- The code does not check the prime number to see if it is truly a prime number, and so it will perform calculations on invalid prime numbers. I have added a function that does this in the tests folder of the document. However, it is not implemented in the program
- In my implementation, I do not use a while loop. It is designed in a one-way approach, seeking to compute just one prime, generator pair before closing the socket. In the future, it could be extended to accommodate not only multiple clients but also have ongoing communication with them until the user decides to close the connection.

### **Screenshots of informative Messages**



```
PS C:\Users\mimii\Downloads\starter_code\starter_code> python server.py 10128
Server of Aisha K.Forrester
Connected from: ('127.0.0.1', 60704)

*** (2) .Received 100 Hello from client ***

Enter a prime number between 127 and 7919: 257
Enter a generator for the prime number: 3

*** (3.) Sent 105 Generator + Commitment g,p,t,V to client ***

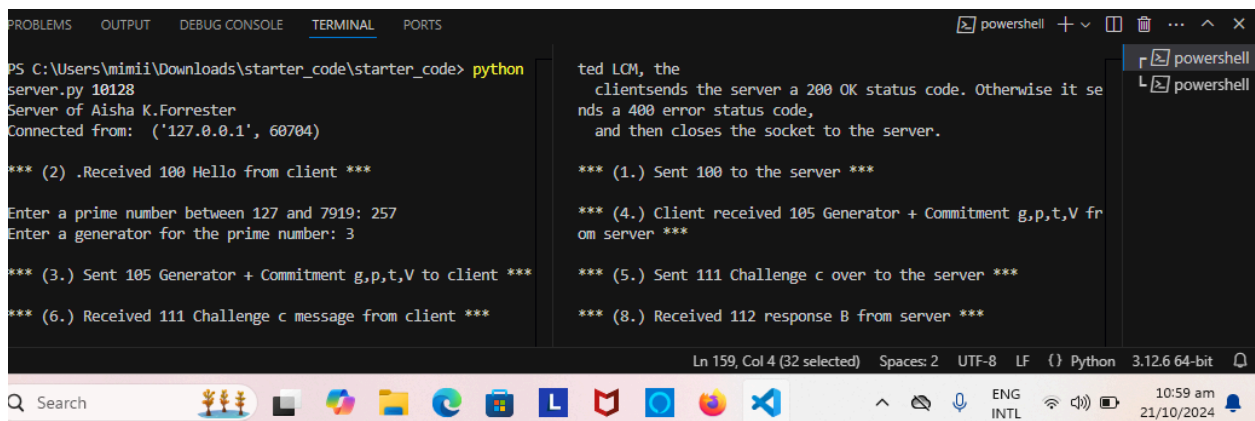
*** (6.) Received 111 Challenge c message from client ***

PS C:\Users\mimii\Downloads\starter_code\starter_code> python client.py localhost 10128
Client of Aisha K Forrester

The purpose of this program is to collect two prime numbers from the client, and then send them to the server. The server will compute their LCM and send it back to the client. If the server-computed LCM matches the locally computed LCM, the client sends the server a 200 OK status code. Otherwise it sends a 400 error status code, and then closes the socket to the server.
```

Generator = 3 Prime = 257. As the server and client receive and send information, it is printed on the screen so the grader can trace what is happening.

The grader can proceed to follow the communication as each process is numbered after it has taken place. Below is a screenshot of the messages using  $p=257$  and  $g=3$



```
ted LCM, the
client sends the server a 200 OK status code. Otherwise it sends a 400 error status code, and then closes the socket to the server.

*** (1.) Sent 100 to the server ***

*** (4.) Client received 105 Generator + Commitment g,p,t,V from server ***

*** (5.) Sent 111 Challenge c over to the server ***

*** (8.) Received 112 response B from server ***

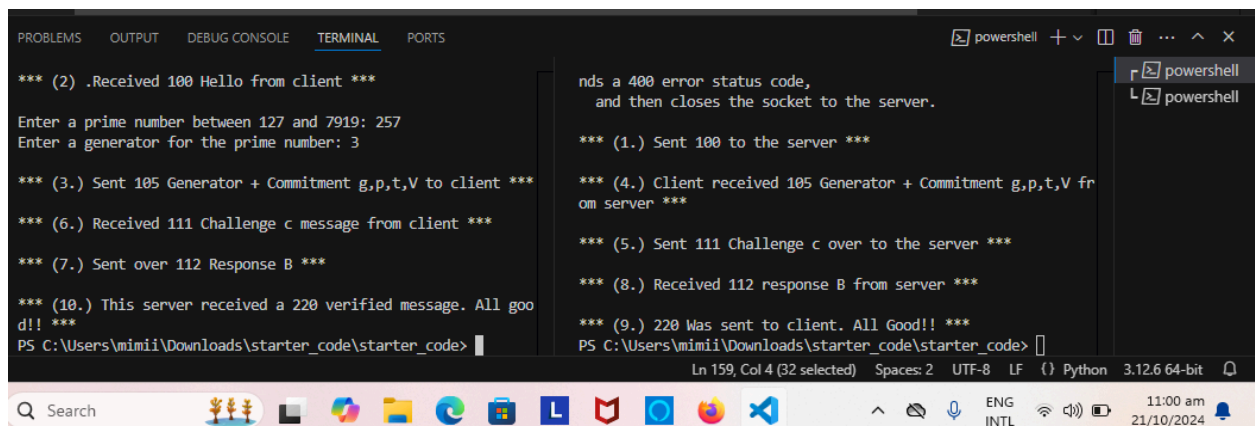
PS C:\Users\mimii\Downloads\starter_code\starter_code> python server.py 10128
Server of Aisha K.Forrester
Connected from: ('127.0.0.1', 60704)

*** (2) .Received 100 Hello from client ***

Enter a prime number between 127 and 7919: 257
Enter a generator for the prime number: 3

*** (3.) Sent 105 Generator + Commitment g,p,t,V to client ***

*** (6.) Received 111 Challenge c message from client ***
```



```
nds a 400 error status code, and then closes the socket to the server.

*** (1.) Sent 100 to the server ***

*** (4.) Client received 105 Generator + Commitment g,p,t,V from server ***

*** (5.) Sent 111 Challenge c over to the server ***

*** (8.) Received 112 response B from server ***

*** (9.) 220 Was sent to client. All Good!! ***

PS C:\Users\mimii\Downloads\starter_code\starter_code>

*** (2) .Received 100 Hello from client ***

Enter a prime number between 127 and 7919: 257
Enter a generator for the prime number: 3

*** (3.) Sent 105 Generator + Commitment g,p,t,V to client ***

*** (6.) Received 111 Challenge c message from client ***

*** (7.) Sent over 112 Response B ***

*** (10.) This server received a 220 verified message. All good!! ***

PS C:\Users\mimii\Downloads\starter_code\starter_code>
```

## Screenshots of interoperability tests:

1. I run the server on my pc and the client is run on a different pc

```
PS C:\Users\mimii\Downloads\starter_code\starter_code>
PS C:\Users\mimii\Downloads\starter_code\starter_code> python server.py 10128
Server of Aisha K.Forrester
Connected from: ('192.168.0.21', 53827)

*** (2.) Received 100 Hello from client ***

Enter a prime number between 127 and 7919: █
```

The client received the message and the program proceed to get the prime number and generator

```
159 | main()
160
161
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

The purpose of this program is to collect two prime numbers from the client,
send them to the server. The server will compute their LCM and send it back t
client. If the server-computed LCM matches the locally computed LCM, the
clientsends the server a 200 OK status code. Otherwise it sends a 400 error st
and then closes the socket to the server.

Sent 100 to the server

Client received: 105 Generator + Commitment a n t V from server
```

The client on the second Pc had sent over the message

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

*** (2.) Received 100 Hello from client ***

Enter a prime number between 127 and 7919: 257
Enter a generator for the prime number: 5

*** (3.) Sent 105 Generator + Commitment g,p,t,V to client ***

*** (6.) Received 111 Challenge c message from client ***

*** (7.) Sent over 112 Response B ***

*** (10.) This server received a 220 verified message. All good!! ***
PS C:\Users\mimii\Downloads\starter_code\starter_code>
```

The processes continue on my end(server)

```
155 #Handle the data that is read through the socket by using processMsgs(s, msg, state)
156
157 #Close the socket
158 if __name__ == "__main__":
159     main()
160
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

and then closes the socket to the server.

Sent 100 to the server

Client received: 105 Generator + Commitment g,p,t,V from server

11 Challenge c sent\*\*\*

Received 112 response B from server

220 Was sent

The process on the other end also continued (client)

## 2. I run the client on my pc and the server is run on a different pc

```
*** (1.) Sent 100 to the server ***
```

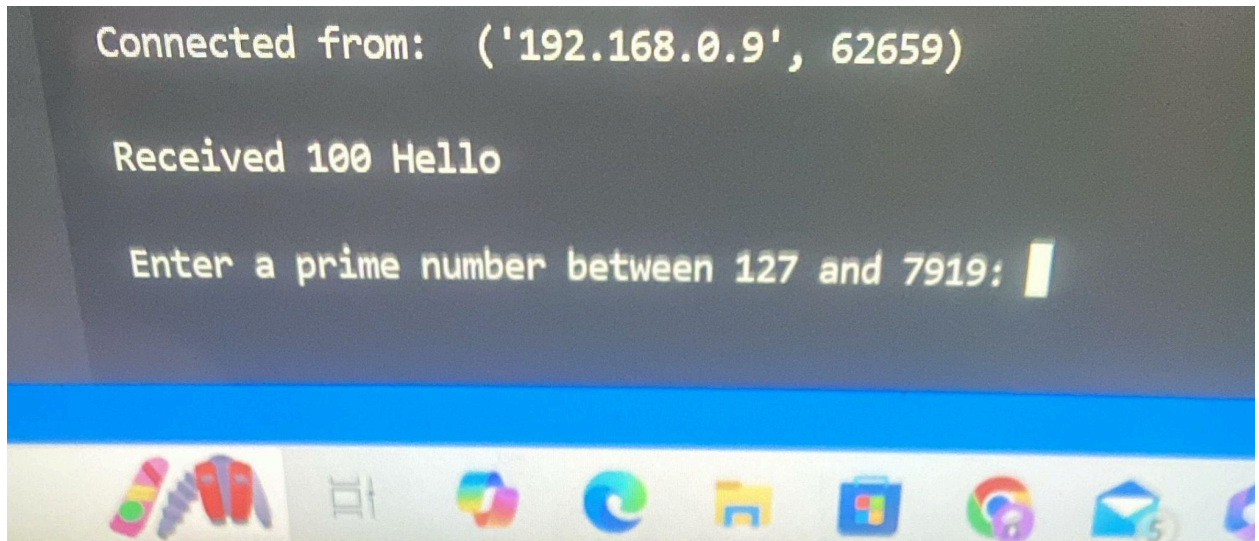
Ln 22, Col 22 Spaces: 2 UTF-8 LF {} Python 3.12.6 64-bit

Search

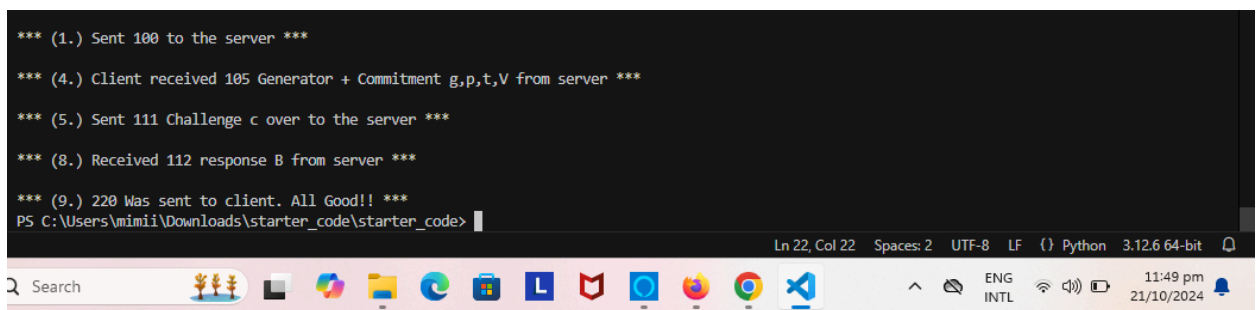
11:46 pm 21/10/2024

As I am the client, I sent over a Hello message to the server

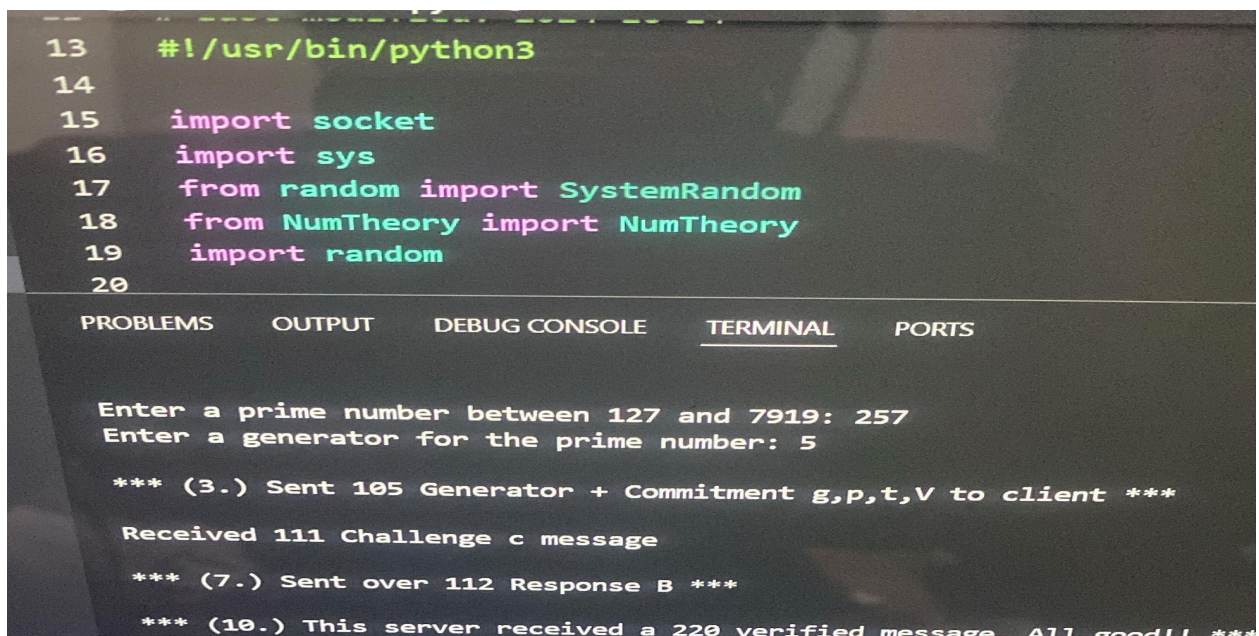




The server from the second pc prompted the user for the prime and generator

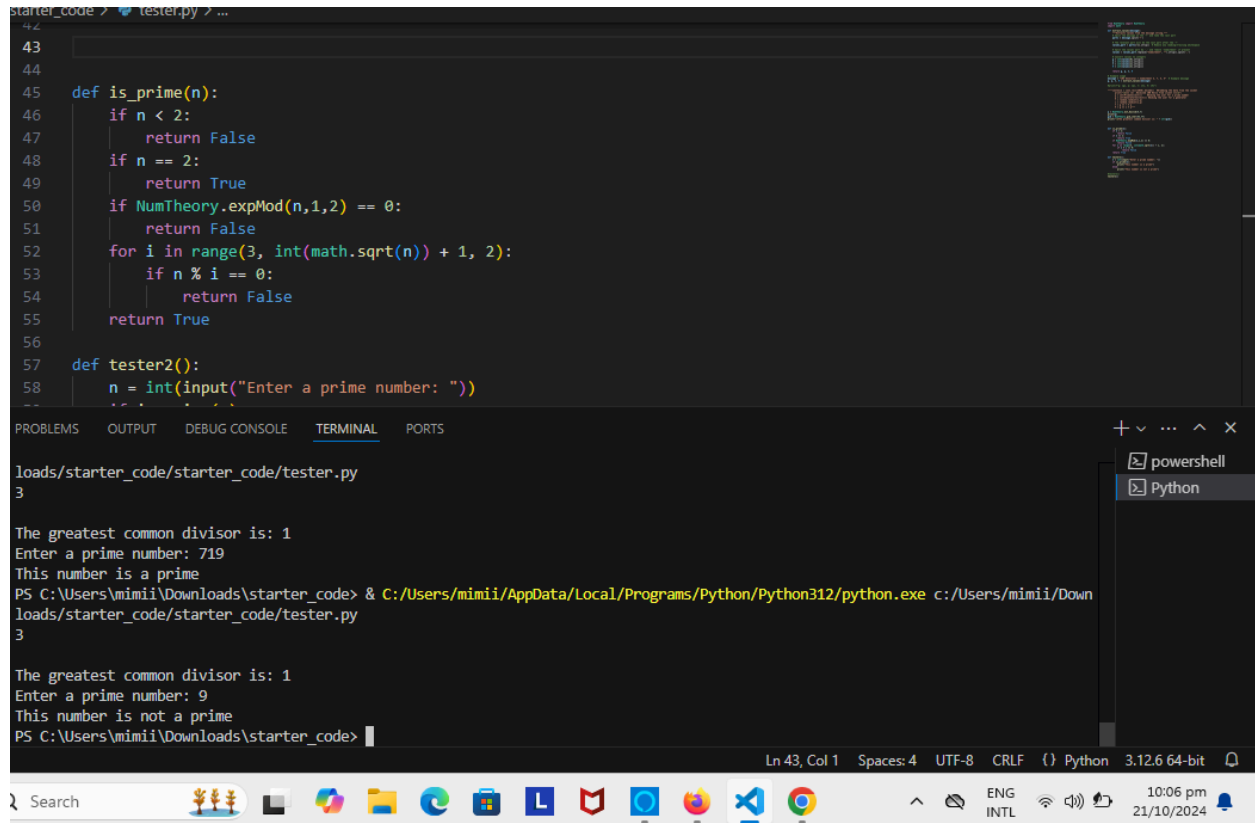


The process on my end continued (client)



## Implementing a function that checks if the number is prime

We know that for a number to be prime, the only factors that it must have are the integer 1 and itself. We can further conclude that the greatest common divisor of 2 integers must be 1 if they are both said to be prime numbers. We also know that any number less than 2 is not prime, and 2 is a prime number. Hence:



```
43
44
45 def is_prime(n):
46     if n < 2:
47         return False
48     if n == 2:
49         return True
50     if NumTheory.expMod(n,1,2) == 0:
51         return False
52     for i in range(3, int(math.sqrt(n)) + 1, 2):
53         if n % i == 0:
54             return False
55     return True
56
57 def tester2():
58     n = int(input("Enter a prime number: "))
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

loads/starter\_code/starter\_code/tester.py  
3

The greatest common divisor is: 1  
Enter a prime number: 719  
This number is a prime

PS C:\Users\mimii\Downloads\starter\_code> & C:/Users/mimii/AppData/Local/Programs/Python/Python312/python.exe c:/Users/mimii/Down  
loads/starter\_code/starter\_code/tester.py  
3

The greatest common divisor is: 1  
Enter a prime number: 9  
This number is not a prime

PS C:\Users\mimii\Downloads\starter\_code>

Ln 43, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.12.6 64-bit

## Part B: Using a Large Language Model (Optional: up to 10 extra-credit points)

### Prompt:

 ChatGPT 4o mini ▾

[Sign up](#)

[Log in](#)

Socket Programming: Say you have a server and a client, and they communicate over a TCP connection. The client will print out: "Client of Joan A. Smith", and a TCP socket to the server will open up. It will send: "100 Hello". The server's response will contain the string "105 Generator + Commitment g, p, t, V", where g, p, t, and V represent integers. It will arrive at the g and p (generator and prime respectively) by prompting the user for it. However, the t and V will be arrived at by the computation:  $t = g ** r \% p$  and  $V = g ** x \% p$ . It will also generate two random integers x and r that are between 1 and p. There is also a function in another class known as NumTheory. It goes as follows: class NumTheory:

```
@staticmethod
def expMod(b,n,m):
    """Computes the modular exponent of a number"""
```

Message ChatGPT



 ChatGPT 4o mini ▾

[Sign up](#)

[Log in](#)

```
def expMod(b,n,m):
    """Computes the modular exponent of a number"""
    """returns (b^n mod m)"""
    if n==0:
        return 1
    elif n%2==0:
        return NumTheory.expMod((b*b)%m, n/2, m)
    else:
        return(b*NumTheory.expMod(b,n-1,m))%m and can be used in
the calculation of t and V. After the server sent over the 105
Generator... message, the client will create their own random integer
less than p. It will then send to the client "111 Challenge C" where c
is that random integer less than p. When the server receives this, it
will compute:  $z = r + x$  where r and x would have been known by the
server as it had previously generated it. The server will then send to
```

Message ChatGPT





server as it had previously generated it. The server will then send to the client "112 Response B" where B is the result of that computation, in other words, it is z. The client, upon receiving this, will compare  $gz \bmod p$  to  $(t \times yc) \bmod p$ . If they match, a "220 Verified" message will be sent back to the server. If not, a "400 Error" will be sent. Otherwise, if any other message that is unexpected is sent to the server, it will send a "500 Bad Request" message to the client. Important functions includes:

```
def serverHello():  
    """Generates server hello message"""  
    status = "100 Hello"  
    return status
```

```
def AllGood():  
    """Generates 220 Verified"""  
    status = "220 OK"
```

Message ChatGPT



ChatGPT 4o mini

Sign

```
status = "220 OK"  
return status
```

```
def ErrorCondition():  
    """Generates 400 Error"""  
    status = "400 Error"  
    return status
```

```
def ChallengeMsg(c):  
    """Generates 111 Challenge """  
    status = "111 Challenge " + str(c)  
    return status (on the client side) and: def PrimeCollect():  
    """Accepts a prime number to send to the client"""  
    primeNbr = input("Enter a prime number between 127 and 7919: ")  
    return primeNbr
```

```
def GeneratorCollect():
    """Accepts a generator for the prime"""
    generator = input("Enter a generator for the prime number: ")
    return generator

def clientHello(g, p, x, r):
    """Generates an acknowledgement for the client
    heGeneratoressage"""
    msg = "105 Generator + Commitment " + str(g) + ", " + str(p) + ", " + \
    str(NumTheory.expMod(g,r,p)) + ", " + str(NumTheory.expMod(g,x,p))
    return msg

# r is the nonce chosen by the server
```

Message ChatGPT



```
# r is the nonce chosen by the server
# x is the server's secret integer
# c is the client's challenge integer
def genChallengeResponse(r, x, c):
    """Generates the 107 LCM string"""
    z = r + c*x
    msg = "112 Response " + str(z)
    return msg (on the server side) also: @staticmethod
def IsValidGenerator(g, p):
    """Validation of generator and prime"""
    x = set()
    for i in range(1,p): #to iterate on the powers of the generator
        modulo p
        x.add(NumTheory.expMod(g,i,p))
    if (len(x) == (p-1)) and (g < p):
```

Message ChatGPT



```

if (len(x) == (p-1)) and (g < p):
    return True
else:
    return False (from the NumTheory class and : @staticmethod
def expMod(b,n,m):
    """Computes the modular exponent of a number"""
    """returns (b^n mod m)"""
    if n==0:
        return 1
    elif n%2==0:
        return NumTheory.expMod((b*b)%m, n/2, m)
    else:
        return(b*NumTheory.expMod(b,n-1,m))%m ...NOTE: On both
client and server ↓ most of the code should be taken place in a def
processMsgs(s, msg, state): function where the s=socket, msg=any

```

gPT



```

elif n%2==0:
    return NumTheory.expMod((b*b)%m, n/2, m)
else:
    return(b*NumTheory.expMod(b,n-1,m))%m ...NOTE: On both
client and server side most of the code should be taken place in a def
processMsgs(s, msg, state): function where the s=socket, msg=any
incoming message from either party to be processed and state= the
state of the important variables such as g,p,t,V and others you might
find suitable. Note, you must still accept the client and server
parameters from the command line. If you need any other
clarifications, please inquire further

```

Code when run:

```
PS C:\Users\mimii\Downloads\starter_code\starter_code> cd test_and_documents
PS C:\Users\mimii\Downloads\starter_code\starter_code\tests_and_documents> python ChatGPIServer.py 65432
Enter the generator for the prime number: 3
Enter a prime number between 127 and 7919: 257
Server listening on localhost:65432...
Connected by ('127.0.0.1', 65429)

PS C:\Users\mimii\Downloads\starter_code\starter_code\tests_and_documents> python ChatGPTClient.py localhost 65432
Received: 100 Hello
Enter the generator sent by the server: 3
Enter the prime number sent by the server: 257
Received: 105 Generator + Commitment 3, 257, 118, 109
Sending: 111 Challenge 59
Received: 112 Response 215231
Bad Request: 500 Bad Request
```

Differences:

- The server starts and immediately prompts for the generator and prime number. This must be given before the client connection is even made
- The client is asking for the prime and generator that was previously entered on the server side instead of sending the challenge over
- The client side and server side ran as normal except it generated a Bad Request message for a valid generator and prime, suggesting that either the prompt was insufficiently written or the LLM did not implement the code properly on both ends

