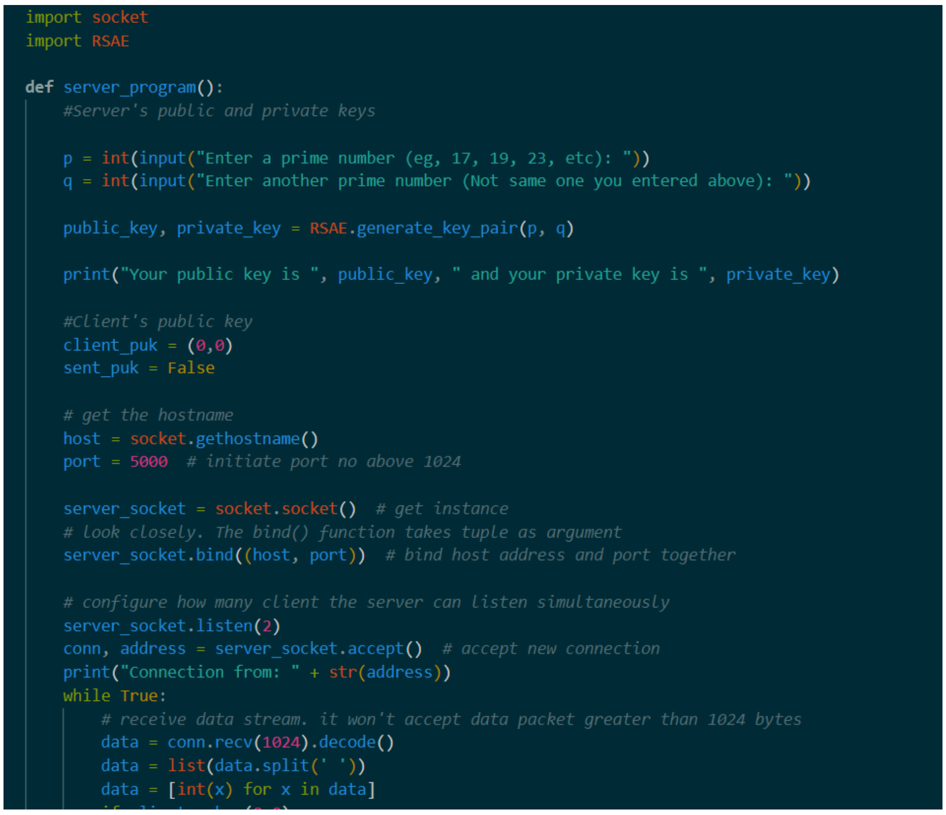
Homework 3

Code for RSA (RSAE.py)

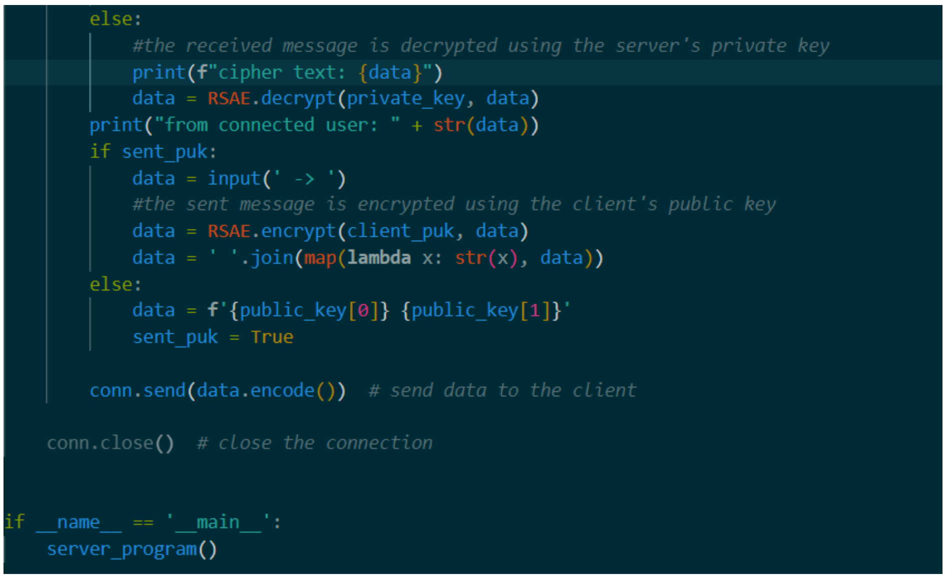
We were given the RSA code and I didn't make any changes to the code

RSA_socket_server.py

This is a screenshot from RSA_socket_server.py. First the server has to choose p and q to generate its public and private key. The public key from the client is received and then the server sends its own public key. After the public keys are shared, the messages are encrypted using the RSA cipher.



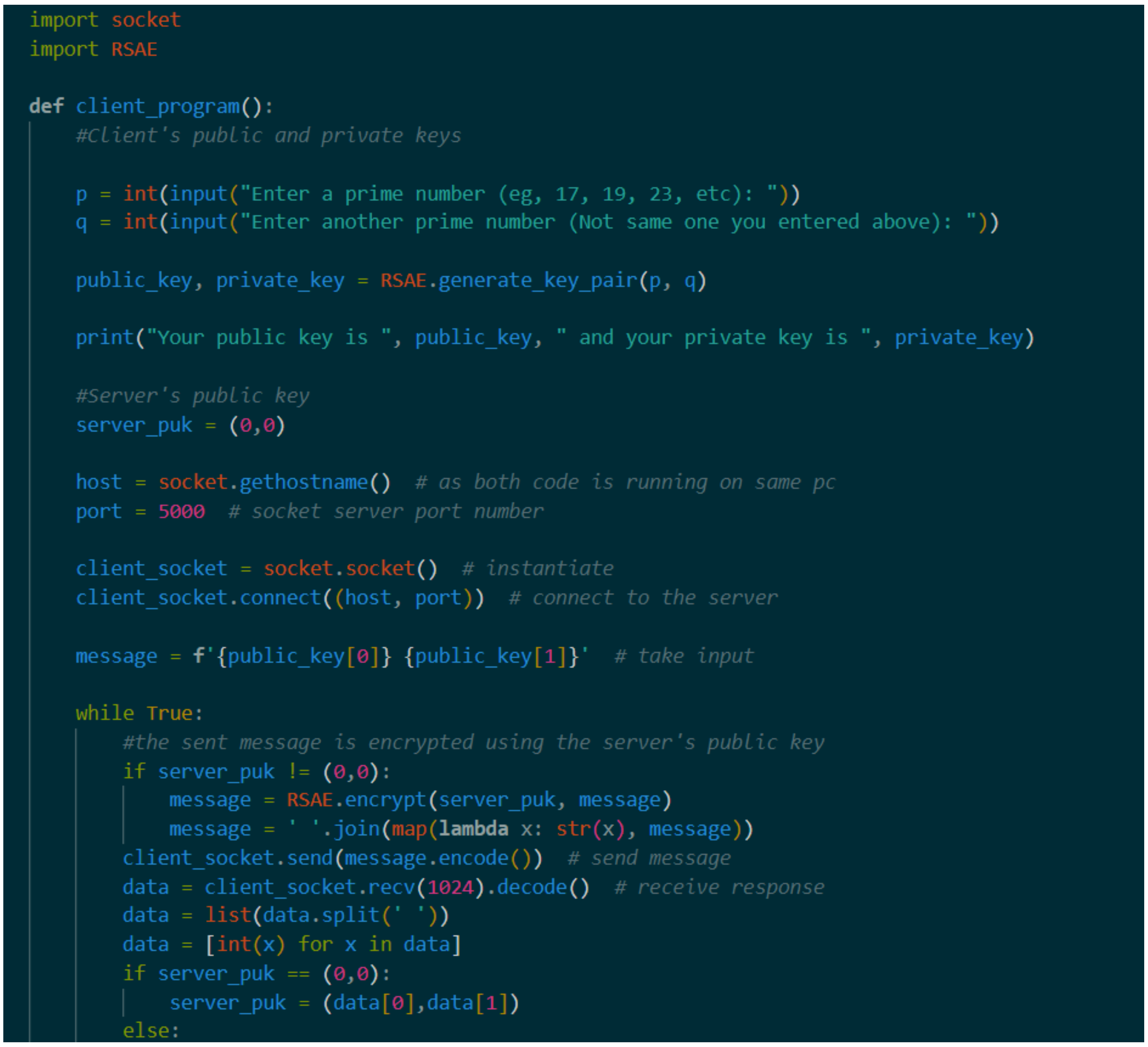
```python
import socket
import RSAE

def server_program():
    #Server's public and private keys

    p = int(input("Enter a prime number (eg, 17, 19, 23, etc): "))
    q = int(input("Enter another prime number (Not same one you entered above): "))

    public_key, private_key = RSAE.generate_key_pair(p, q)

    print("Your public key is ", public_key, " and your private key is ", private_key)

    #Client's public key
    client_puk = (0,0)
    sent_puk = False

    # get the hostname
    host = socket.gethostname()
    port = 5000  # initiate port no above 1024

    server_socket = socket.socket()  # get instance
    # look closely. The bind() function takes tuple as argument
    server_socket.bind((host, port))  # bind host address and port together

    # configure how many client the server can listen simultaneously
    server_socket.listen(2)
    conn, address = server_socket.accept()  # accept new connection
    print("Connection from: " + str(address))
    while True:
        # receive data stream. it won't accept data packet greater than 1024 bytes
        data = conn.recv(1024).decode()
        data = list(data.split(' '))
        data = [int(x) for x in data]
```



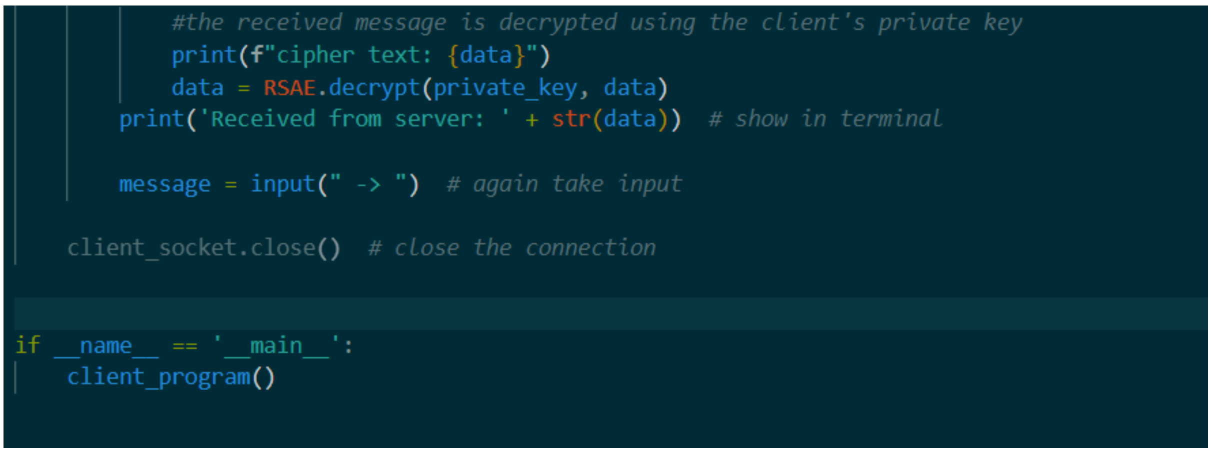
```python
        else:
            #the received message is decrypted using the server's private key
            print(f"cipher text: {data}")
            data = RSAE.decrypt(private_key, data)
        print("from connected user: " + str(data))
        if sent_puk:
            data = input(' -> ')
            #the sent message is encrypted using the client's public key
            data = RSAE.encrypt(client_puk, data)
            data = ' '.join(map(lambda x: str(x), data))
        else:
            data = f'{public_key[0]} {public_key[1]}'
            sent_puk = True

        conn.send(data.encode())  # send data to the client

    conn.close()  # close the connection


if __name__ == '__main__':
    server_program()
```

RSA_socket_client.py

This is a screenshot from RSA_socket_client.py. First the client has to choose p and q to generate its public and private key. Then the client sends its public key to the server and then the server sends its public key. After the public keys are shared, the messages are encrypted using the RSA cipher.



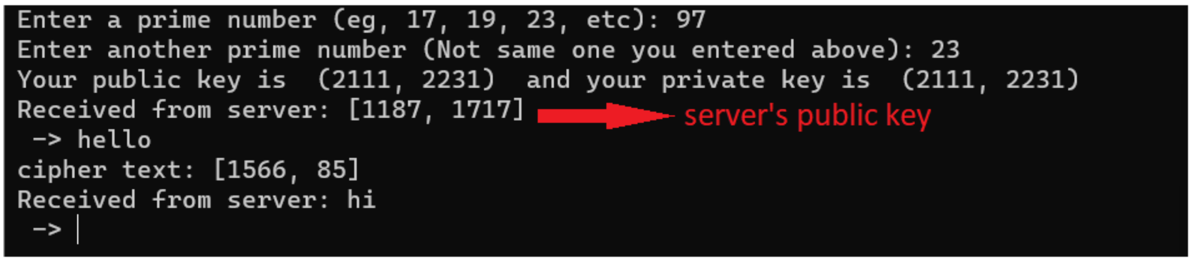
```python
import socket
import RSAE

def client_program():
    #Client's public and private keys

    p = int(input("Enter a prime number (eg, 17, 19, 23, etc): "))
    q = int(input("Enter another prime number (Not same one you entered above): "))

    public_key, private_key = RSAE.generate_key_pair(p, q)

    print("Your public key is ", public_key, " and your private key is ", private_key)

    #Server's public key
    server_puk = (0,0)

    host = socket.gethostname()  # as both code is running on same pc
    port = 5000  # socket server port number

    client_socket = socket.socket()  # instantiate
    client_socket.connect((host, port))  # connect to the server

    message = f'{public_key[0]} {public_key[1]}'  # take input

    while True:
        #the sent message is encrypted using the server's public key
        if server_puk != (0,0):
            message = RSAE.encrypt(server_puk, message)
            message = ' '.join(map(lambda x: str(x), message))
        client_socket.send(message.encode())  # send message
        data = client_socket.recv(1024).decode()  # receive response
        data = list(data.split(' '))
        data = [int(x) for x in data]
        if server_puk == (0,0):
            server_puk = (data[0],data[1])
        else:
```



```python
            #the received message is decrypted using the client's private key
            print(f"cipher text: {data}")
            data = RSAE.decrypt(private_key, data)
        print('Received from server: ' + str(data))  # show in terminal

        message = input(" -> ")  # again take input

    client_socket.close()  # close the connection


if __name__ == '__main__':
    client_program()
```

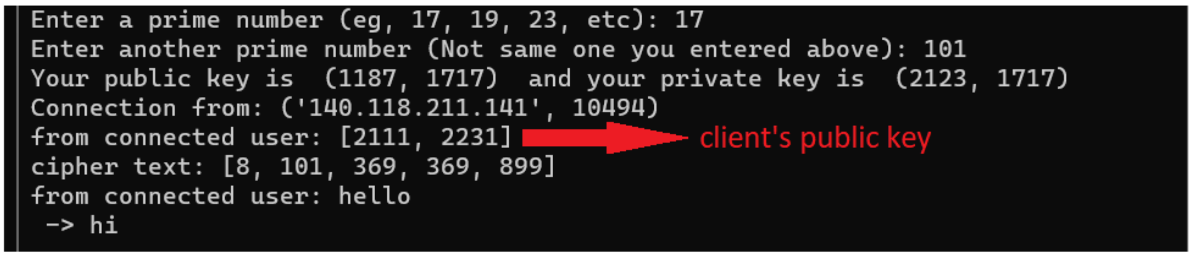
RSA minichat snapshots

Client:



```
Enter a prime number (eg, 17, 19, 23, etc): 97
Enter another prime number (Not same one you entered above): 23
Your public key is  (2111, 2231)  and your private key is  (2111, 2231)
Received from server: [1187, 1717]
 -> hello
cipher text: [1566, 85]
Received from server: hi
 -> 
```

Server:

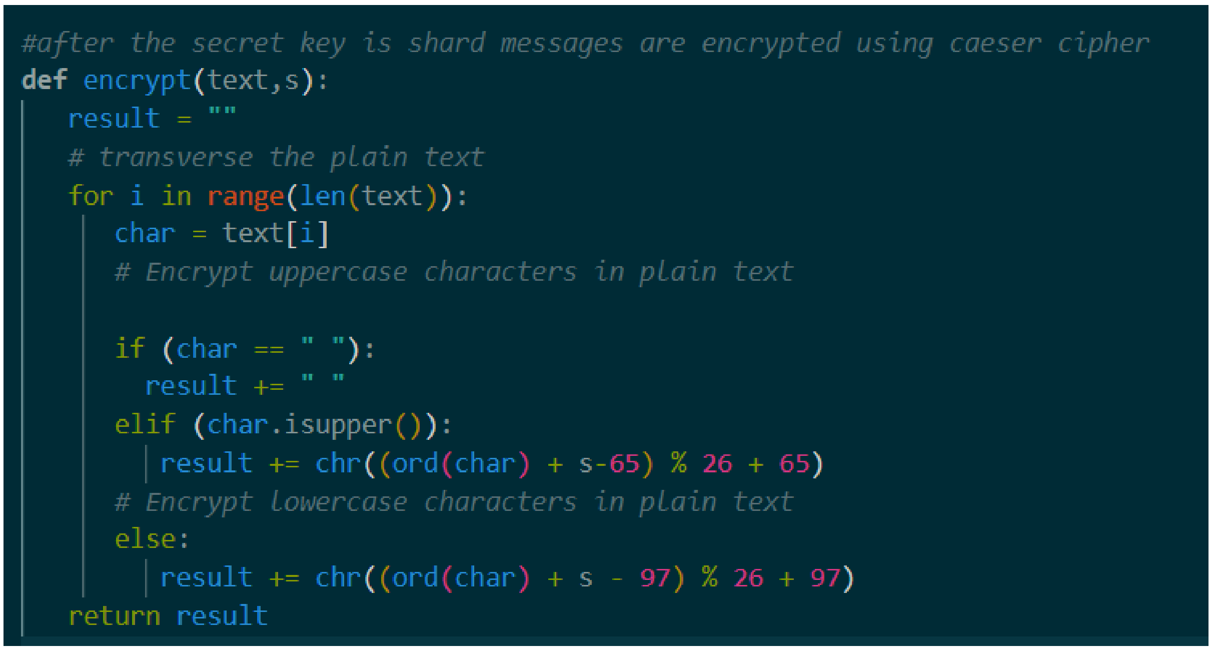


```
Enter a prime number (eg, 17, 19, 23, etc): 17
Enter another prime number (Not same one you entered above): 101
Your public key is  (1187, 1717)  and your private key is  (2123, 1717)
Connection from: ('140.118.211.141', 10494)
from connected user: [2111, 2231]
cipher text: [8, 101, 369, 369, 899]
from connected user: hello
 -> hi
```

DH_socket_server.py

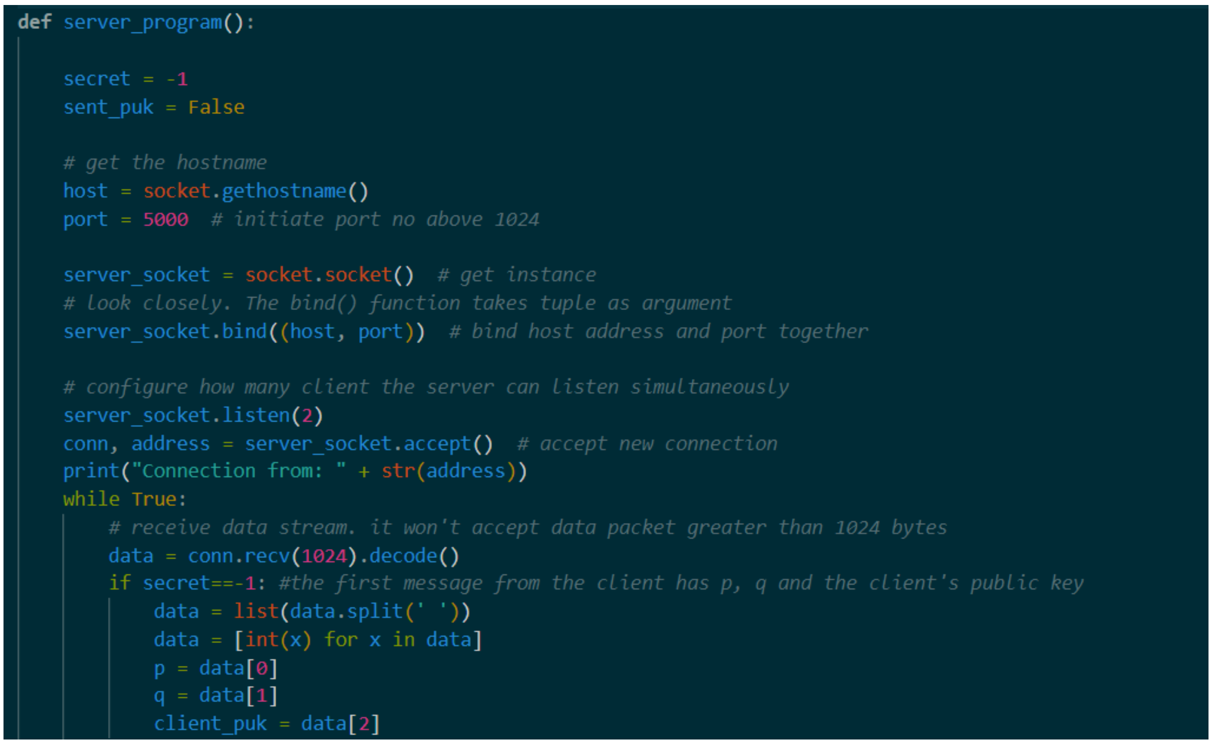
This code shares a secret key with the client using the Diffie-Hellman Key Exchange. First the client chooses p, q and its private key, then its public key is calculated and sent to the server. Then the server chooses its private key, calculates its public key and the secret, and sends its public key to the client. Finally the client calculates the secret and then the messages are encrypted with the secret key using Caesar Cipher.

The Caesar Cipher function:

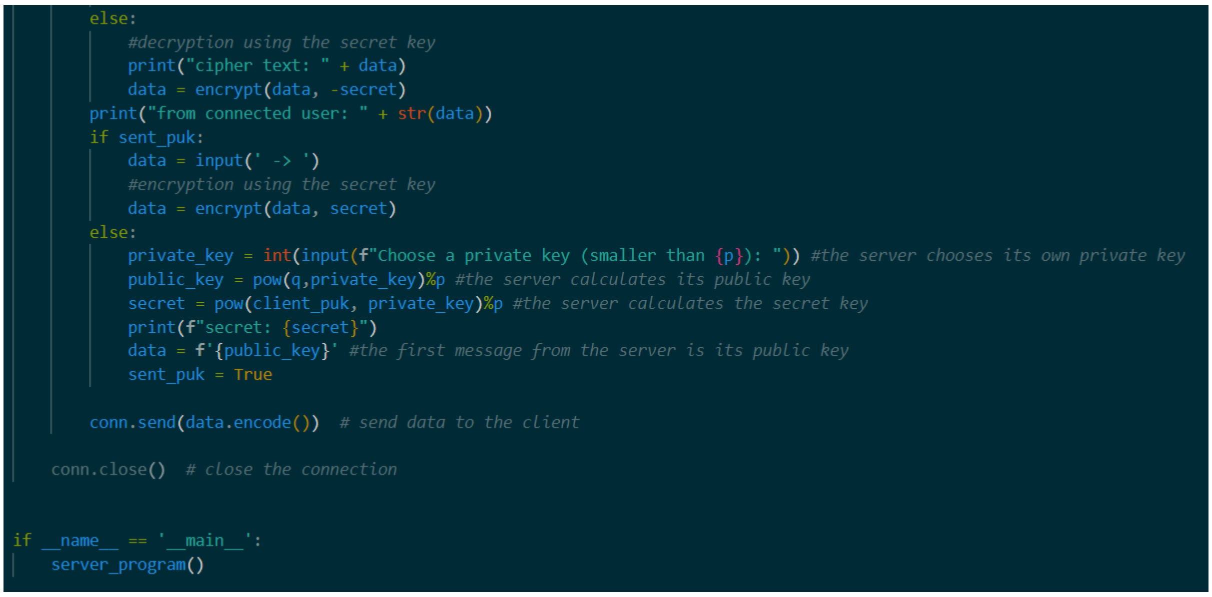


```python
#after the secret key is shard messages are encrypted using caeser cipher
def encrypt(text,s):
    result = ""
    # transverse the plain text
    for i in range(len(text)):
        char = text[i]
        # Encrypt uppercase characters in plain text

        if (char == " "):
          result += " "
        elif (char.isupper()):
            result += chr((ord(char) + s-65) % 26 + 65)
        # Encrypt lowercase characters in plain text
        else:
            result += chr((ord(char) + s - 97) % 26 + 97)
    return result
```

The code:

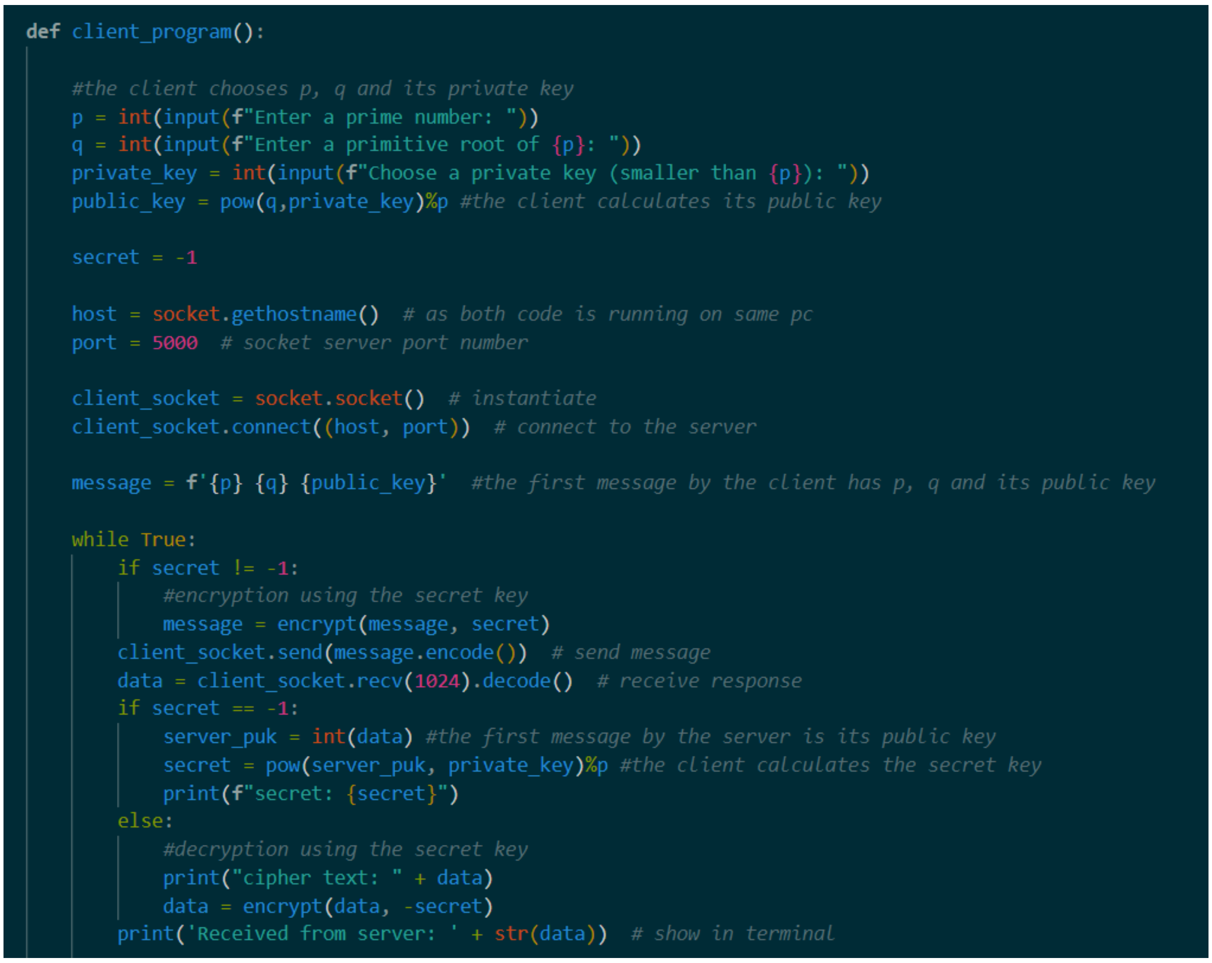


```python
def server_program():

    secret = -1
    sent_puk = False

    # get the hostname
    host = socket.gethostname()
    port = 5000  # initiate port no above 1024

    server_socket = socket.socket()  # get instance
    # look closely. The bind() function takes tuple as argument
    server_socket.bind((host, port))  # bind host address and port together

    # configure how many client the server can listen simultaneously
    server_socket.listen(2)
    conn, address = server_socket.accept()  # accept new connection
    print("Connection from: " + str(address))
    while True:
        # receive data stream. it won't accept data packet greater than 1024 bytes
        data = conn.recv(1024).decode()
        if secret==-1: #the first message from the client has p, q and the client's public key
            data = list(data.split(' '))
            data = [int(x) for x in data]
            p = data[0]
            q = data[1]
            client_puk = data[2]
```

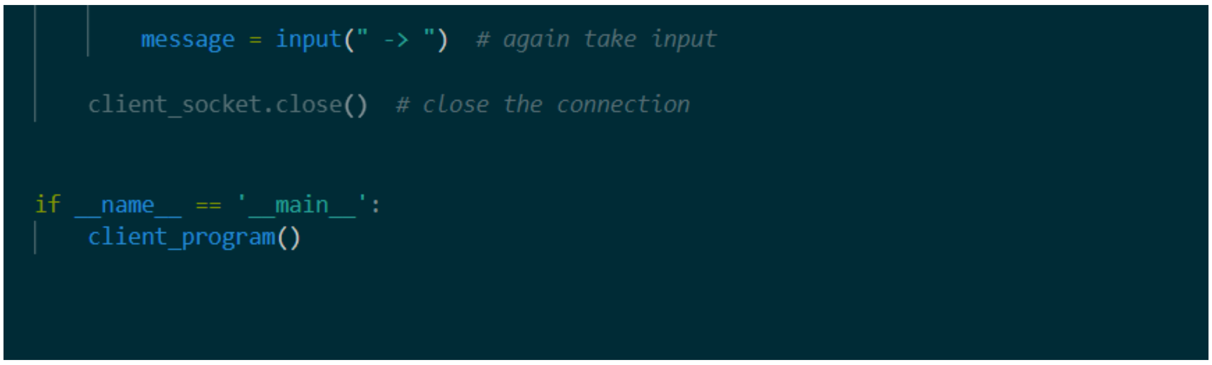


```python
        else:
            #decryption using the secret key
            print("cipher text: " + data)
            data = encrypt(data, -secret)
        print("from connected user: " + str(data))
        if sent_puk:
            data = input(' -> ')
            #encryption using the secret key
            data = encrypt(data, secret)
        else:
            private_key = int(input(f"Choose a private key (smaller than {p}): ")) #the server chooses its own private key
            public_key = pow(q,private_key)%p #the server calculates its public key
            secret = pow(client_puk, private_key)%p #the server calculates the secret key
            print(f"secret: {secret}")
            data = f'{public_key}' #the first message from the server is its public key
            sent_puk = True

        conn.send(data.encode())  # send data to the client

    conn.close()  # close the connection


if __name__ == '__main__':
    server_program()
```

DH_socket_client.py



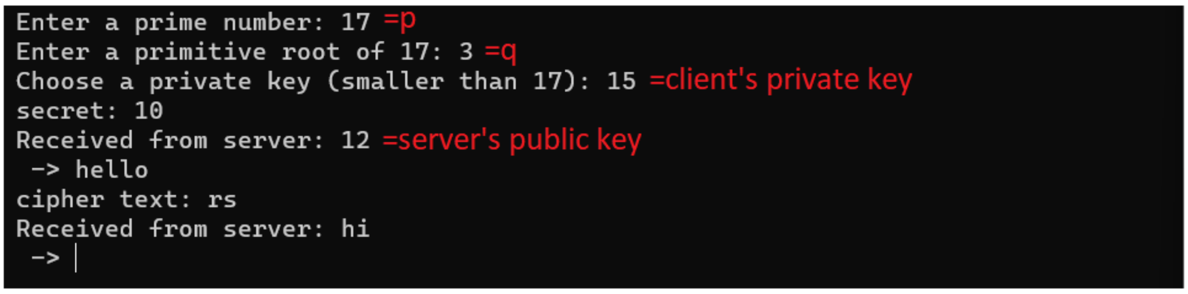
```python
def client_program():

    #the client chooses p, q and its private key
    p = int(input(f"Enter a prime number: "))
    q = int(input(f"Enter a primitive root of {p}: "))
    private_key = int(input(f"Choose a private key (smaller than {p}): "))
    public_key = pow(q,private_key)%p #the client calculates its public key

    secret = -1

    host = socket.gethostname()  # as both code is running on same pc
    port = 5000  # socket server port number

    client_socket = socket.socket()  # instantiate
    client_socket.connect((host, port))  # connect to the server

    message = f'{p} {q} {public_key}'  #the first message by the client has p, q and its public key

    while True:
        if secret != -1:
            #encryption using the secret key
            message = encrypt(message, secret)
        client_socket.send(message.encode())  # send message
        data = client_socket.recv(1024).decode()  # receive response
        if secret == -1:
            server_puk = int(data) #the first message by the server is its public key
            secret = pow(server_puk, private_key)%p #the client calculates the secret key
            print(f"secret: {secret}")
        else:
            #decryption using the secret key
            print("cipher text: " + data)
            data = encrypt(data, -secret)
        print('Received from server: ' + str(data))  # show in terminal
```



```
        message = input(" -> ")  # again take input

    client_socket.close()  # close the connection


if __name__ == '__main__':
    client_program()
```

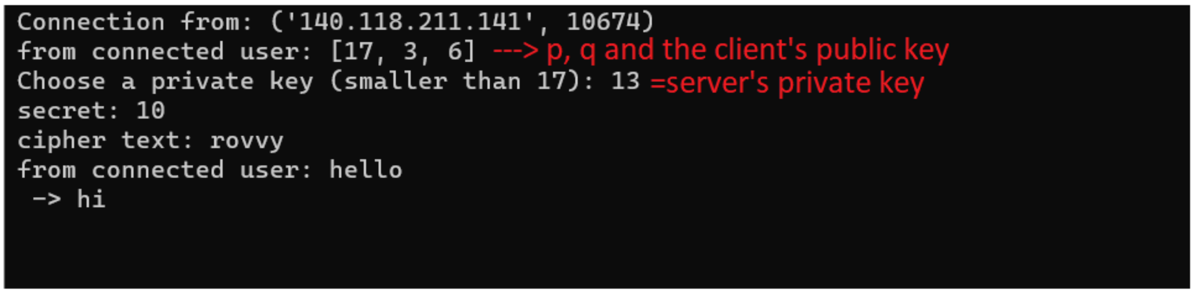
DH minichat snapshots

Client:



```
Enter a prime number: 17 =p
Enter a primitive root of 17: 3 =q
Choose a private key (smaller than 17): 15 =client's private key
secret: 10
Received from server: 12 =server's public key
 -> hello
cipher text: rs
Received from server: hi
 ->
```

Server:



```
Connection from: ('140.118.211.141', 10674)
from connected user: [17, 3, 6] ---> p, q and the client's public key
Choose a private key (smaller than 17): 13 =server's private key
secret: 10
cipher text: rovvy
from connected user: hello
 -> hi
```