I have used socket programming in python to implement the Remote File Service (RFS). I have used a TCP connection, since it ensures that packets reach the destination.

<u>Commands:</u>

CWD: Get the current working directory
LS: Get the constituents of the current working directory
CD: Change the current directory
DWD: Download files
UPD: Upload files

<u>Command Format:</u>

“CWD”
“LS”
“CD {Path/Folder name}” . Eg: “CD C:/IIT Gandhinagar”
“DWD {filename}”. Eg “DWD test.txt”
“UPD {filename}”. Eg: “UPD trial.png”

Also for each command query, the mode of encryption (also the offset in case of substitution encryption) remains the same. The encryption type changes (if it changes) after the previous command is entirely completed. Encryption methods are written in the crypto.py file, which are imported in the client and server files.

Encryption Method	**Decryption Method**
Plain Text: Keep it as it is.	No decryption required.
Substitution: Offset each alphanumeric character by n places.	decode_offset(): This function shifts it back from the offset places and consequently decrypt
Transpose: Reverse the content	Transpose: Called back to again reverse the content and hence decrypt it

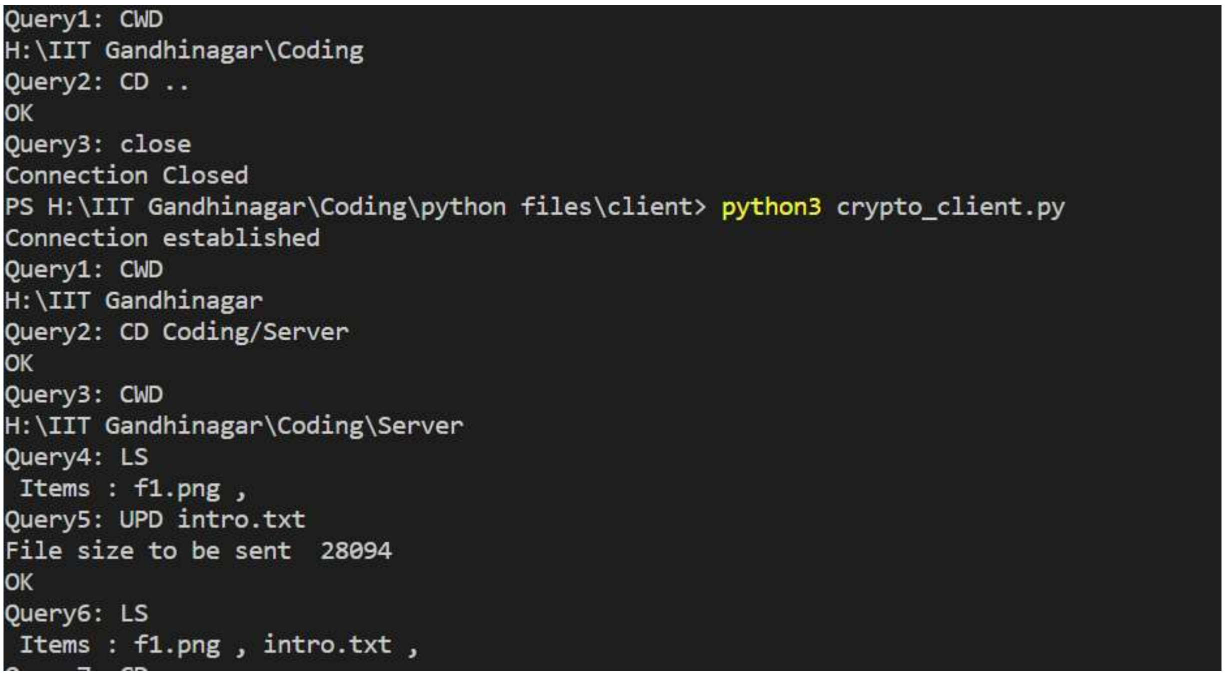
Note that numericals in substitution offset aren’t shifted by n places. They’re shifted n times such that numeric value remains between 0 to 9. i.e., new value = (old value +n)mod10

The encryption method adds a header in the encrypted string corresponding to which layer of encryption is used. Substitution further has a header containing the offset.

Modules: socket, os, base64, random

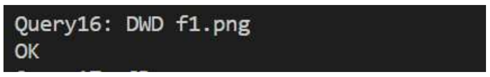
I have used the base64 library for reading the files and converting them to bytes for proper transmission of data bytes from client to server and vice versa. The base64 module is used to convert byte-like objects to bytes.

Screenshot of commands:



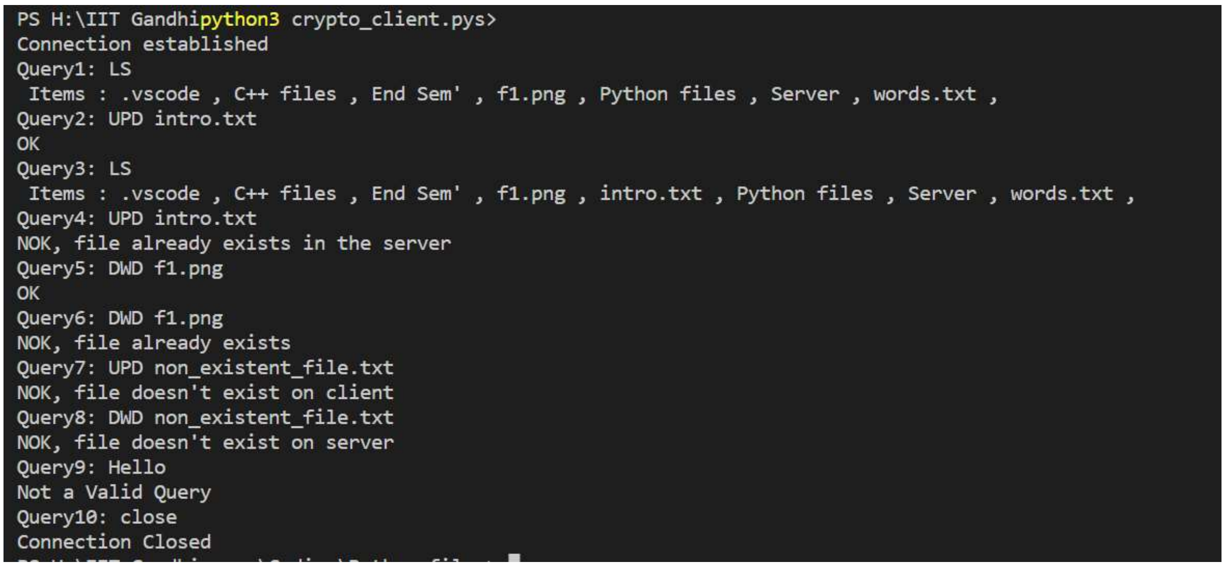
```
Query1: CWD
H:\IIT Gandhinagar\Coding
Query2: CD ..
OK
Query3: close
Connection Closed
PS H:\IIT Gandhinagar\Coding\python files\client> python3 crypto_client.py
Connection established
Query1: CWD
H:\IIT Gandhinagar
Query2: CD Coding/Server
OK
Query3: CWD
H:\IIT Gandhinagar\Coding\Server
Query4: LS
 Items : f1.png ,
Query5: UPD intro.txt
File size to be sent  28094
OK
Query6: LS
 Items : f1.png , intro.txt ,
```

We are initially at H: \IIT Gandhinagar\Coding. We then use the CD command to go to the folder named Server inside the Coding folder. We then list the items present in the Server folder. We see only one image f1.png is present. After using the UPD command, we again use LS and find that intro.txt got added to the items present in the Server folder.
Finally we use the DWD command to download the png file from the Server folder to the client folder.



```
Query16: DWD f1.png
OK
```

Other functionalities:
This RFS also tells users if the files being uploaded or downloaded have already been uploaded or downloaded before respectively. It also lets the user know in case of errors where the user tries to upload or download non-existent files. It also lets the user know when the command given is invalid.

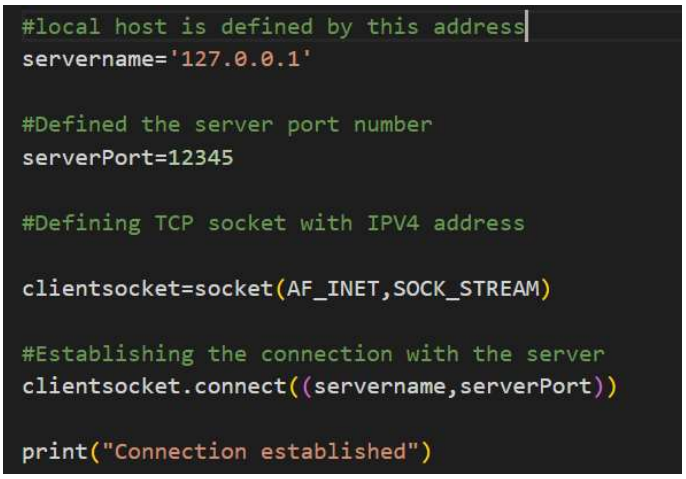


```
PS H:\IIT Gandhipython3 crypto_client.pys>
Connection established
Query1: LS
 Items : .vscode , C++ files , End Sem' , f1.png , Python files , Server , words.txt ,
Query2: UPD intro.txt
OK
Query3: LS
 Items : .vscode , C++ files , End Sem' , f1.png , intro.txt , Python files , Server , words.txt ,
Query4: UPD intro.txt
NOK, file already exists in the server
Query5: DWD f1.png
OK
Query6: DWD f1.png
NOK, file already exists
Query7: UPD non_existent_file.txt
NOK, file doesn't exist on client
Query8: DWD non_existent_file.txt
NOK, file doesn't exist on server
Query9: Hello
Not a Valid Query
Query10: close
Connection Closed
```

Below are the explanations of the Client Side, Server Side and Crypto layer programs:

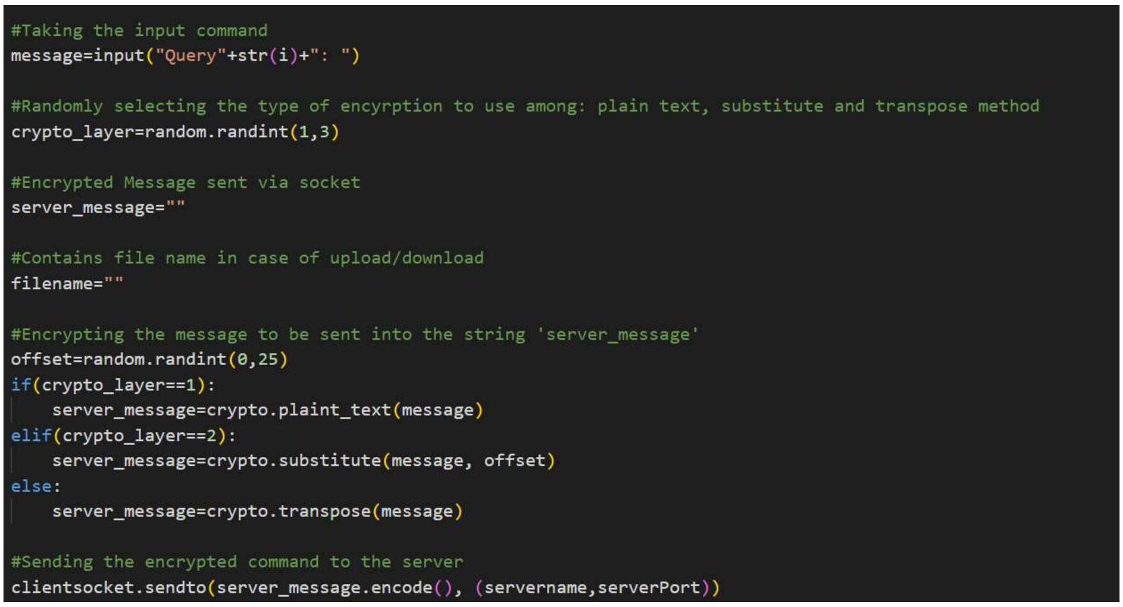
1.)Client Side:

a) Connection is established:



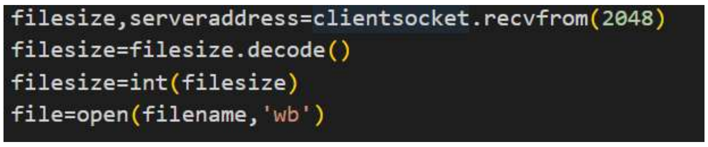
```
#local host is defined by this address
servername='127.0.0.1'

#Defined the server port number
serverPort=12345

#Defining TCP socket with IPV4 address

clientsocket=socket(AF_INET,SOCK_STREAM)

#Establishing the connection with the server
clientsocket.connect((servername,serverPort))

print("Connection established")
```

b) Take command input from the user and send the encrypted command to the server. Client randomly chooses the mode of encrypting the data



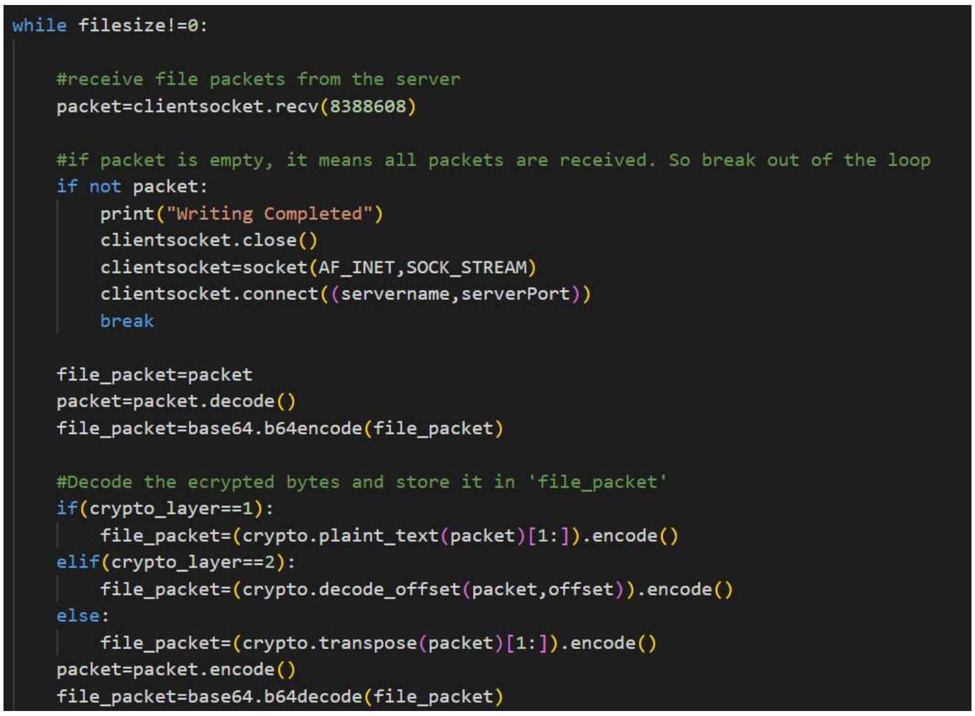
```
#Taking the input command
message=input("Query"+str(i)+": ")

#Randomly selecting the type of encyrption to use among: plain text, substitute and transpose method
crypto_layer=random.randint(1,3)

#Encrypted Message sent via socket
server_message=""

#Contains file name in case of upload/download
filename=""

#Encrypting the message to be sent into the string 'server_message'
offset=random.randint(0,25)
if(crypto_layer==1):
    server_message=crypto.plaint_text(message)
elif(crypto_layer==2):
    server_message=crypto.substitute(message, offset)
else:
    server_message=crypto.transpose(message)

#Sending the encrypted command to the server
clientsocket.sendto(server_message.encode(), (servername,serverPort))
```

c) If the command is DWD, i.e., download a file, the client first receives the file size from the server and then the databytes for the files which will be written in a new file on the client side and as a result the file gets downloaded.



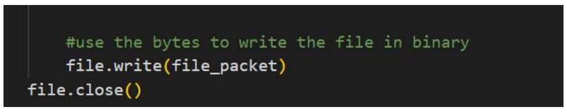
```
filesize,serveraddress=clientsocket.recvfrom(2048)
filesize=filesize.decode()
filesize=int(filesize)
file=open(filename,'wb')
```

It also decrypts the bytes before



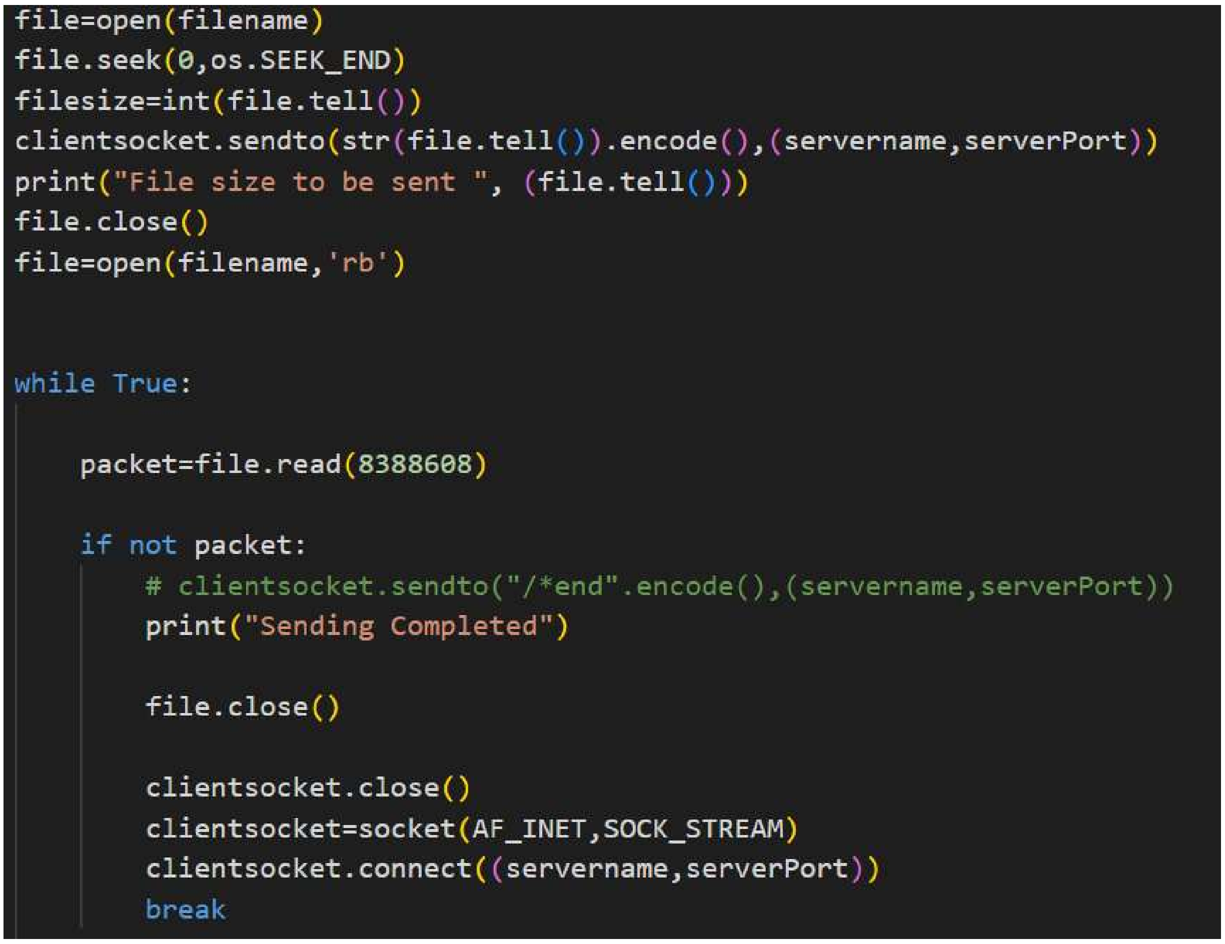
```
while filesize!=0:

    #receive file packets from the server
    packet=clientsocket.recv(8388608)

    #if packet is empty, it means all packets are received. So break out of the loop
    if not packet:
        print("Writing Completed")
        clientsocket.close()
        clientsocket=socket(AF_INET,SOCK_STREAM)
        clientsocket.connect((servername,serverPort))
        break

    file_packet=packet
    packet=packet.decode()
    file_packet=base64.b64encode(file_packet)

    #Decode the ecrypted bytes and store it in 'file_packet'
    if(crypto_layer==1):
        file_packet=(crypto.plaint_text(packet)[1:]).encode()
    elif(crypto_layer==2):
        file_packet=(crypto.decode_offset(packet,offset)).encode()
    else:
        file_packet=(crypto.transpose(packet)[1:]).encode()
    packet=packet.encode()
    file_packet=base64.b64decode(file_packet)
```

As we can see above, after each DWD command, the tcp connection is broken and then again re-established. Connection is broken also from the server side. This is necessary for it to know that all the bytes have been transferred and no more remain.
Finally use the decrypted bytes to write the file.



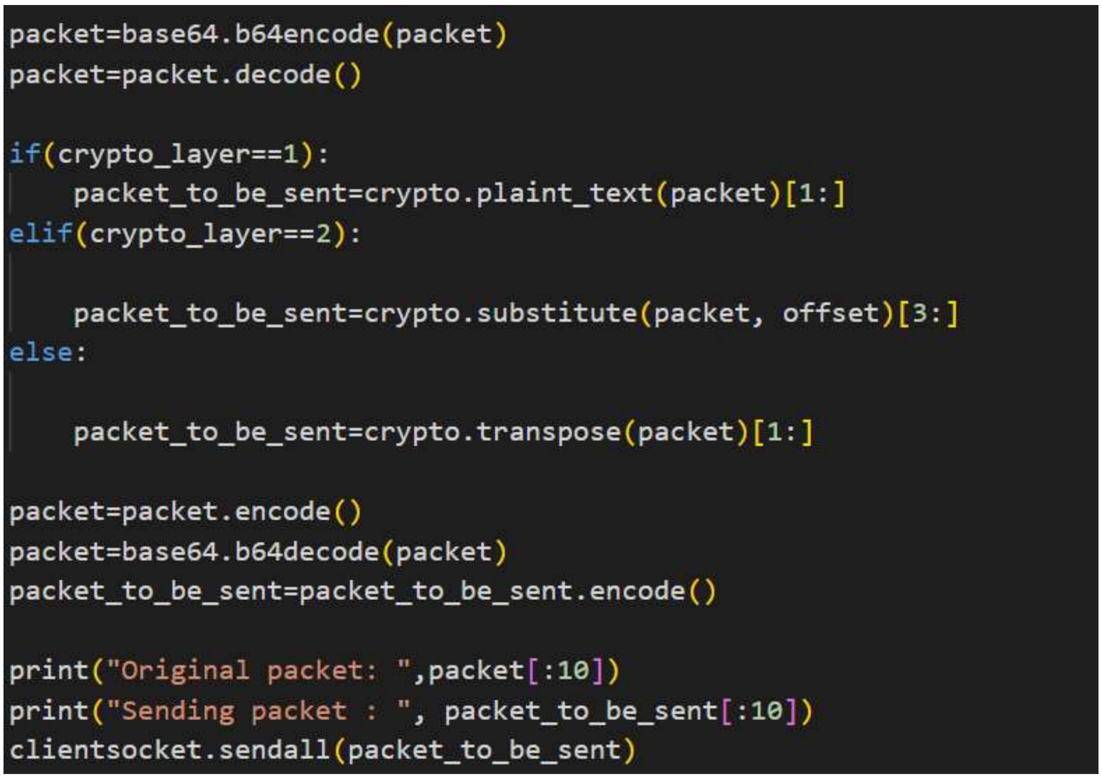
```
    #use the bytes to write the file in binary
    file.write(file_packet)
file.close()
```

d) If the command in UPD, i.e., upload, the client opens the file in binary format and starts reading the file.. If no bytes are read, we have reached the end of the file and the client breaks out of the loop. Furthermore, same as DWD, tcp connection is broken and re-established after each UPD command.



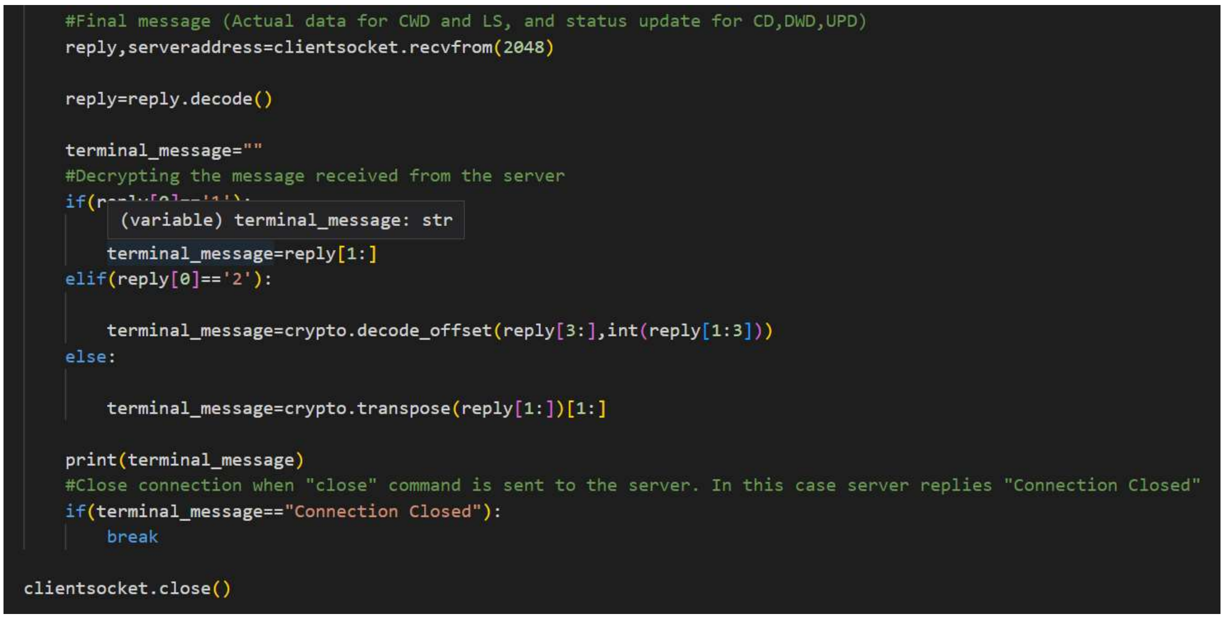
```
file=open(filename)
file.seek(0,os.SEEK_END)
filesize=int(file.tell())
clientsocket.sendto(str(file.tell()).encode(),(servername,serverPort))
print("File size to be sent ", (file.tell()))
file.close()
file=open(filename,'rb')


while True:

    packet=file.read(8388608)

    if not packet:
        # clientsocket.sendto("/*end".encode(),(servername,serverPort))
        print("Sending Completed")

        file.close()

        clientsocket.close()
        clientsocket=socket(AF_INET,SOCK_STREAM)
        clientsocket.connect((servername,serverPort))
        break
```

It then encrypts the read bytes, and then sends the encrypted bytes to the server for uploading



```
packet=base64.b64encode(packet)
packet=packet.decode()

if(crypto_layer==1):
    packet_to_be_sent=crypto.plaint_text(packet)[1:]
elif(crypto_layer==2):

    packet_to_be_sent=crypto.substitute(packet, offset)[3:]
else:

    packet_to_be_sent=crypto.transpose(packet)[1:]

packet=packet.encode()
packet=base64.b64decode(packet)
packet_to_be_sent=packet_to_be_sent.encode()

print("Original packet: ",packet[:10])
print("Sending packet : ", packet_to_be_sent[:10])
clientsocket.sendall(packet_to_be_sent)
```

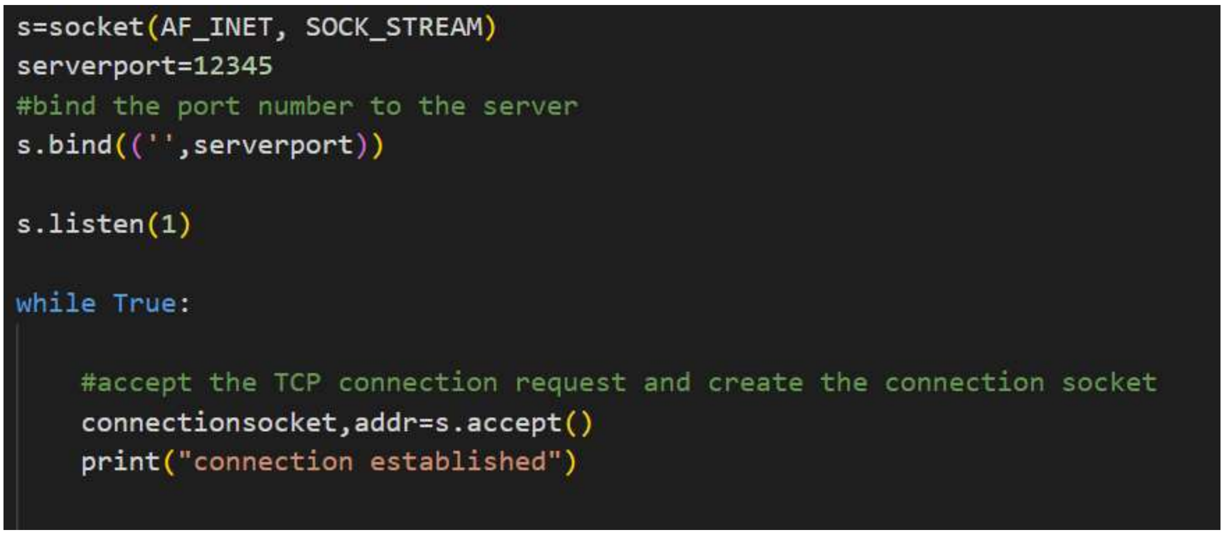
e) Finally it receives the final reply from the server. In case of CWD and LS commands, the reply contains the current working directory and the list of all the items in that directory respectively. In case of other commands, the final reply consists of the status of the operation, whether the command successfully executed or not. If the reply received is “Connection Closed”, it breaks out of the loop and the connection is broken.



```
    #Final message (Actual data for CWD and LS, and status update for CD,DWD,UPD)
    reply,serveraddress=clientsocket.recvfrom(2048)

    reply=reply.decode()

    terminal_message=""
    #Decrypting the message received from the server
    if(reply[0]=='1'):

        terminal_message=reply[1:]
    elif(reply[0]=='2'):

        terminal_message=crypto.decode_offset(reply[3:],int(reply[1:3]))
    else:

        terminal_message=crypto.transpose(reply[1:])[1:]

    print(terminal_message)
    #Close connection when "close" command is sent to the server. In this case server replies "Connection Closed"
    if(terminal_message=="Connection Closed"):
        break

clientsocket.close()
```

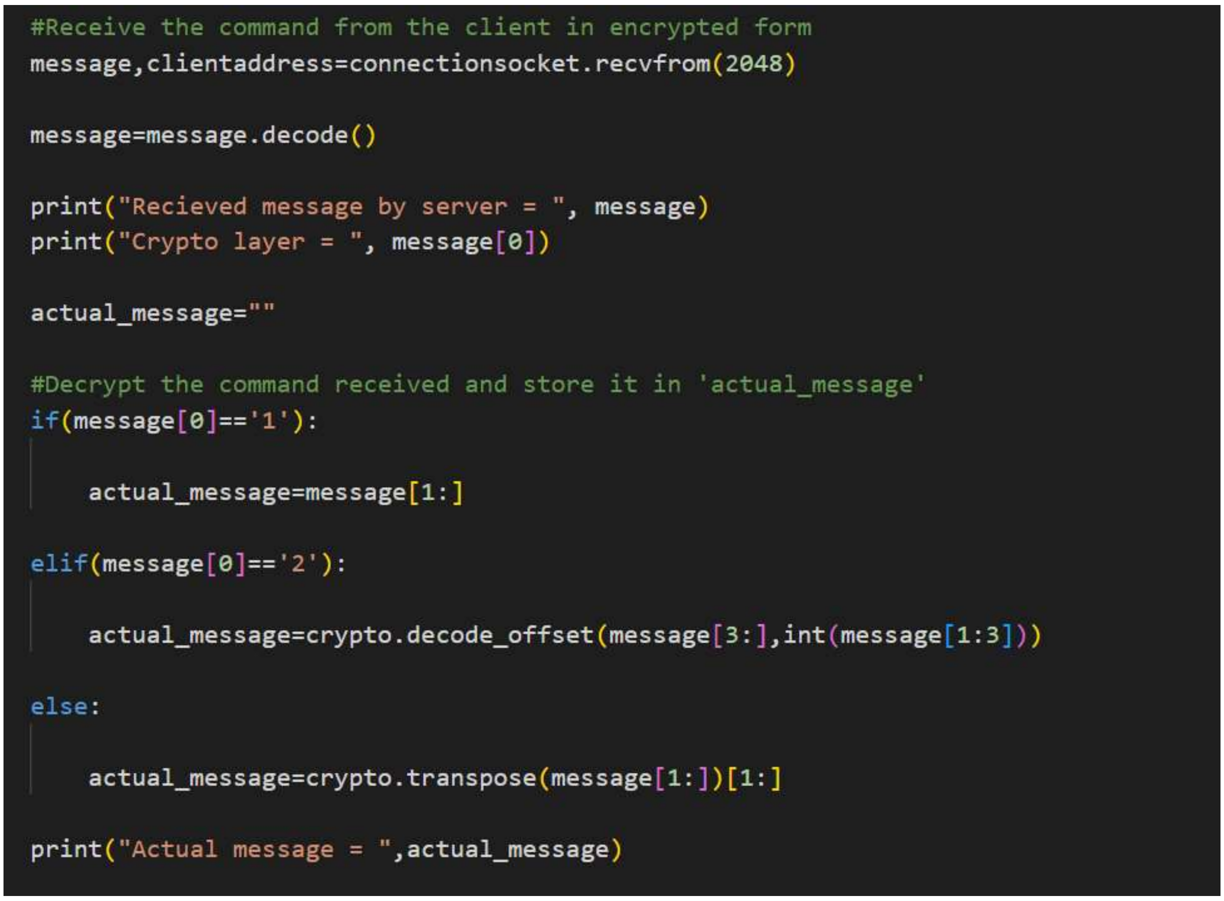
2.) Server Side:

a) Socket is created and is binded with the server port number: After this the connection simply waits for a tcp connection request.



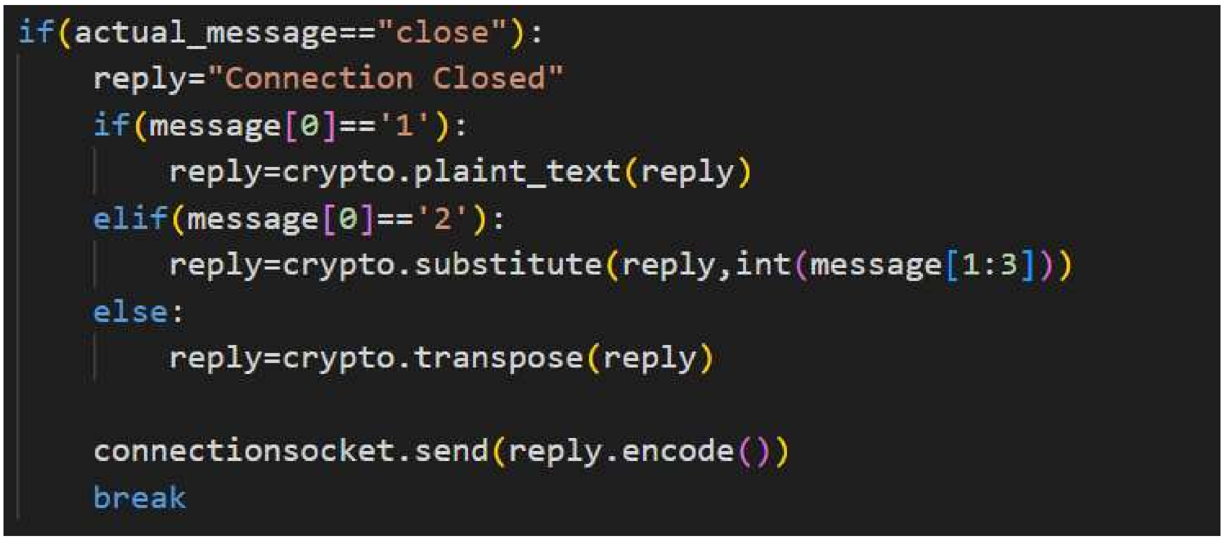
```python
s=socket(AF_INET, SOCK_STREAM)
serverport=12345
#bind the port number to the server
s.bind(('',serverport))

s.listen(1)

while True:

    #accept the TCP connection request and create the connection socket
    connectionsocket,addr=s.accept()
    print("connection established")
```

b) It then receives the command input from the client in encrypted form. This command is then decrypted to get the exact command.



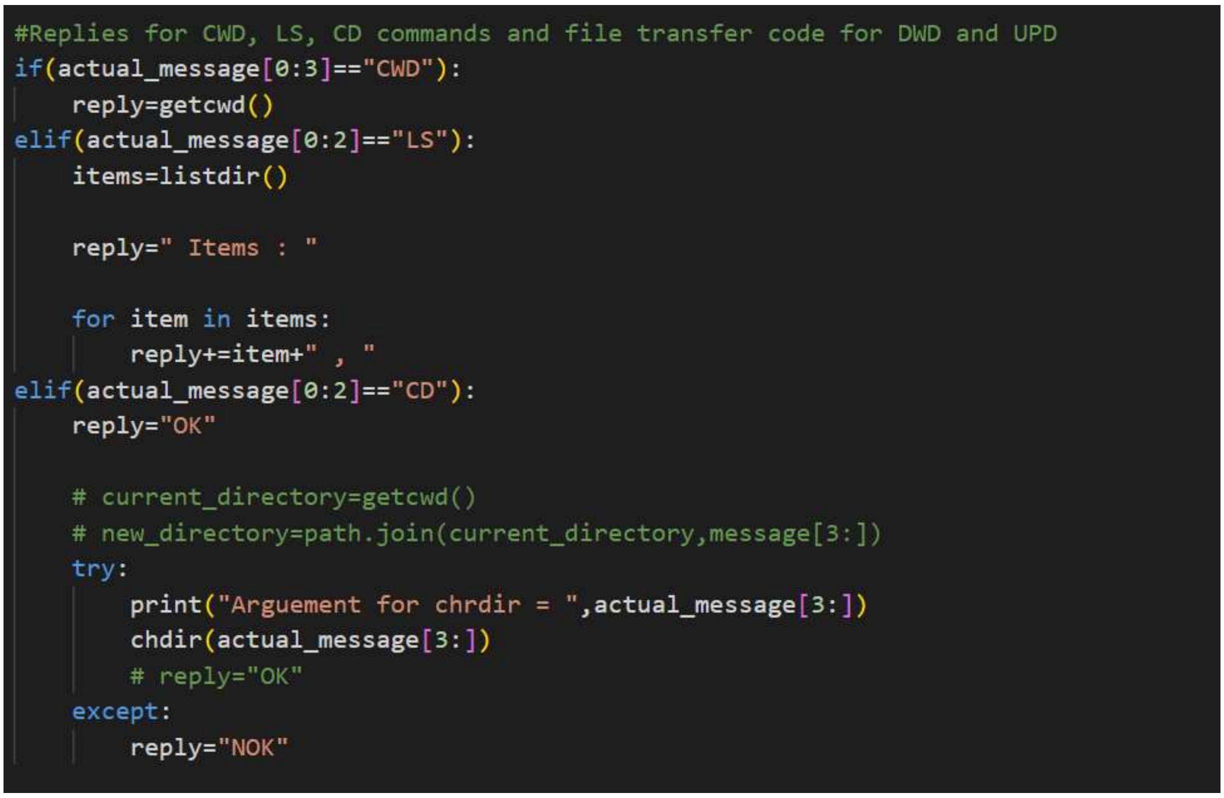
```python
#Receive the command from the client in encrypted form
message,clientaddress=connectionsocket.recvfrom(2048)

message=message.decode()

print("Recieved message by server = ", message)
print("Crypto layer = ", message[0])

actual_message=""

#Decrypt the command received and store it in 'actual_message'
if(message[0]=='1'):

    actual_message=message[1:]

elif(message[0]=='2'):

    actual_message=crypto.decode_offset(message[3:],int(message[1:3]))

else:

    actual_message=crypto.transpose(message[1:])[1:]

print("Actual message = ",actual_message)
```

c) A reply variable is created which is sent to the client in response to its request. If the message was “close”, server replies the “Connection Close” message to the client, which then breaks the connection:



```
if(actual_message=="close"):
    reply="Connection Closed"
    if(message[0]=='1'):
        reply=crypto.plaint_text(reply)
    elif(message[0]=='2'):
        reply=crypto.substitute(reply,int(message[1:3]))
    else:
        reply=crypto.transpose(reply)

    connectionsocket.send(reply.encode())
    break
```

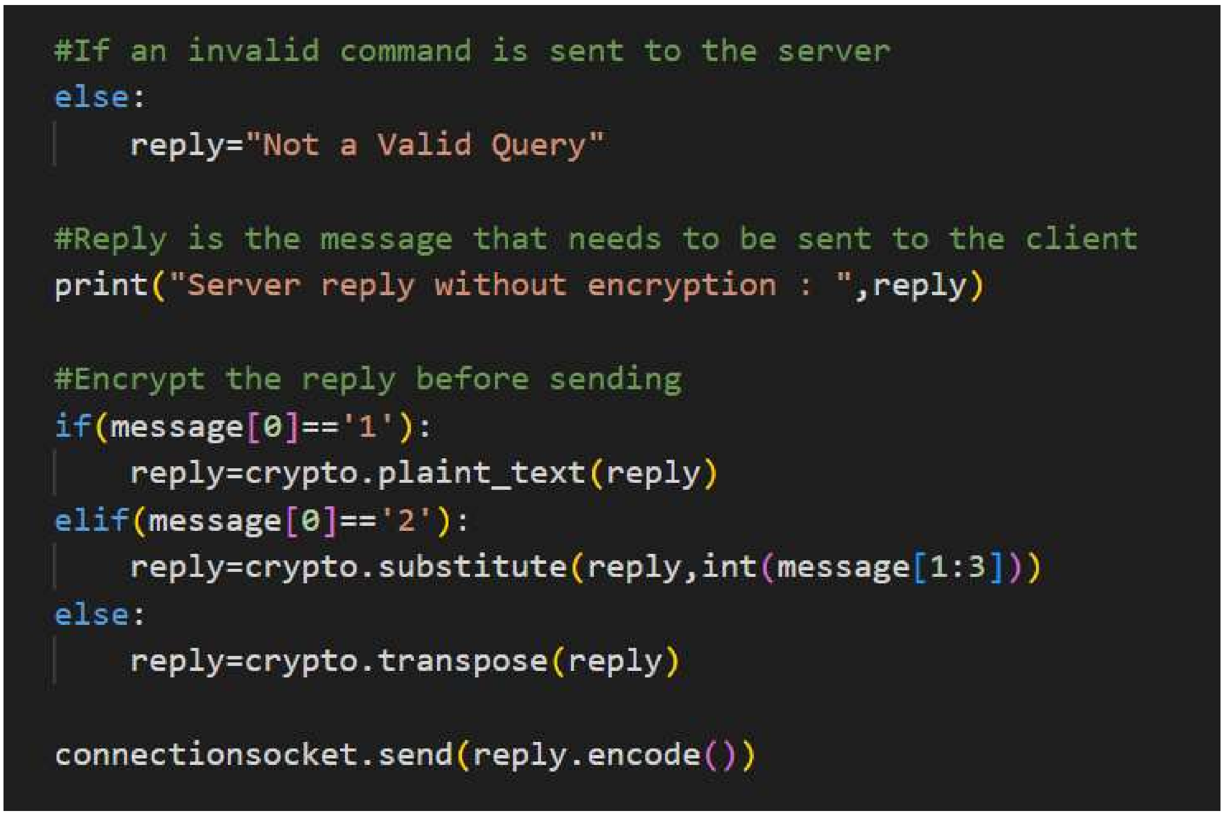
Os library is used to get the current directory, list of files in the directories, and change the directory:



```
#Replies for CWD, LS, CD commands and file transfer code for DWD and UPD
if(actual_message[0:3]=="CWD"):
    reply=getcwd()
elif(actual_message[0:2]=="LS"):
    items=listdir()

    reply=" Items : "

    for item in items:
        reply+=item+" , "
elif(actual_message[0:2]=="CD"):
    reply="OK"

    # current_directory=getcwd()
    # new_directory=path.join(current_directory,message[3:])
    try:
        print("Arguement for chrdir = ",actual_message[3:])
        chdir(actual_message[3:])
        # reply="OK"
    except:
        reply="NOK"
```

d) Similar logics for UPD and DWD are used on the client side. Encrypted bytes are received, which are then decrypted before writing on the server side for UPD, and read bytes are encrypted which are then sent to the client for the DWD command.

e) If the command received is not among CWD,LS,CD,UPD and DWD, the server replies "Not a valid query". Finally, the reply is encrypted and then sent to the client.



```
#If an invalid command is sent to the server
else:
    reply="Not a Valid Query"

#Reply is the message that needs to be sent to the client
print("Server reply without encryption : ",reply)

#Encrypt the reply before sending
if(message[0]=='1'):
    reply=crypto.plaint_text(reply)
elif(message[0]=='2'):
    reply=crypto.substitute(reply,int(message[1:3]))
else:
    reply=crypto.transpose(reply)

connectionsocket.send(reply.encode())
```

Challenges:

Error handling was one major challenge. Avoiding server crashing and disconnecting in case of wrong commands was very difficult to implement.

Expanding this to all types of file transfer was very difficult. It was easy for .txt files but reading bytes of png, jpg, pdf etc.files and encrypting them was very difficult, since reading file in binary doesn't actually give bytes but byte like objects. I had to convert those byte-like objects using the base64 library into bytes and then convert them to string for encryption before sending the message.
Detecting the actual problem with these png, jpg files and then finding base64 library for implementing file transfer was very challenging.

Limitations:

The above RFS uses non persistent tcp connection. It doesn’t resolve multiple queries in a single handshake and needs to get disconnected and connected again between queries.