CSC3002F Assignment 1: Networks

GROUP 60: Done by SLMTAA007, PDCPRA001, MSLGRE001

Features provided by the protocol:

While the core feature of our protocol is to enable peer-to-peer file sharing with the use of a TCP connection, there are additional features within this core feature that work to make our protocol effective. These features include:

- One of the two primary features is the uploading of files to the server by clients. This feature is facilitated by a series of communication between the server and client where the server requests for the file name and desired privacy of the file being uploaded before requesting that the file be sent in byte form. When the file name is received, it is compared to the directory of file names stored and if it matches a file name already in the directory, the user is prompted to either enter a different file name or abort the request. This is done to avoid duplicate files and associated errors. The directory checked is the passwords text file which stores a directory of the different files in the database with their associated privacy and passwords (if protected). When the privacy is sent, it is checked to determine whether it is open or not. If it is open, the file is immediately requested to be sent in byte form and an entry is created in the passwords file for the given file name with open privacy. If it is not open, the desired file password is first requested before it is requested that the file be sent in byte form and the entry is created in the passwords file. To send the file, the client must read the bytes of the file they wish to send and send all of these bytes to the server. The server will then incrementally receive these bytes 4096 bytes at a time, before eventually writing all the bytes to a newly created file in the Database.
- The other primary feature is that of the client downloading files from the server. The communication surrounding this feature first entails the server sending the list of files available for downloading by the client, after which the client chooses their desired file by entering its file name and sending it to the server. Thereafter, the server's response will either inform the user that the file name is invalid and prompt them to enter a new one, or either prompt the user to enter the file's password if the file is protected or send the file in byte form if it is open. In the case that the file requires a password, the server will inform the client of this requirement. Thereafter, it will check each password attempt sent by the client and communicate to the client whether it is correct or not. If the client has 3 incorrect attempts, they are informed that they exceeded the maximum password attempts and the connection socket is closed. Once the user enters the correct password or if the file is open, the file data is sent to the client in byte form, which the client will use to populate a file with the data on their side similarly to how it is done with the uploading as explained above.
- To determine the reliability of the file transfer either way, a String hexadecimal hash code is created on both the server and client side using the data sent/received. The hash code sent to the receiver is compared to a hash code calculated at the receiver to determine whether any bytes were lost in transition or if the bytes were sent incorrectly. If

the hash codes do not match, the receiver will notify the sender to of this and prompt them to reconnect and try again. The calculation of the hash code is done using the hashlib library and its use ensures adequate reliability of our protocol in terms of data transfer.

- Lastly, to ensure file security of protected files, we used a “passwords.txt” file which stored the file names currently in the database, along with their privacy and password in the format: “fileName/privacy/password/”. This file is appended to every time a new file is uploaded and read from every time a download request is received.

Client implementation:

SLMTAA007: The client is run using command line arguments for the server IP address and port number in that order respectively. These arguments are retrieved and used to connect to the server. A message is then received from the server prompting the client to indicate their intentions to either send or receive a file or quit the application. If the client requests to download a file, it is first checked whether a message was received from the server indicating that the database is empty and no files are available for download. If this is the case, the user is notified, the connection is closed and the application is exited. If not, the client then receives a list of files from the server which is printed out. Thereafter, the user is prompted to select a file. If they enter a name that does not exist in the database, the user is prompted to enter a new file name until one is valid. If their requested file requires a password, this is handled by requesting the password and checking the message received from the server to determine its correctness. Once a correct password is entered, the user is notified of this and the program moves onto downloadFile() where it performs the file download. On the other hand, if the user exceeds the maximum password attempts of 3 incorrect passwords, they are notified of this and the connection and application is closed accordingly. If the file requested is not protected, the program moves to downloadFile immediately. Thereafter, the user is requested to enter their desired directory to which they would like to download the file. The file is iteratively received in byte form and written to a new file on the client side as received in packets of 4096 bytes. A hash code used to check the data reliability at the end is also iteratively updated. Once the end tag is received (“<END>”), a hash code is received from the server and compared to that calculated on the client side. If they match, the server and user is notified. If not, they are notified and the file downloaded is deleted. In both instances, the connection and application is then closed. Should the user choose to upload a file instead, the user is prompted to enter the file name of the file they wish to send. Should this file name not exist in the current directory, the user will be prompted to enter a new file name until one does or until they choose to abort, in which case the connection and application is closed. If the client is trying to upload a file that already exists in the database, they are prompted to enter a new file name until this is not the case or they abort. Once the file name is valid, the user is prompted to enter the desired privacy of the file which is sent to the server. If the privacy is not to be open, the user is prompted to enter the desired password which is sent to the server before the upload process begins. The data is sent by reading it from the file 4096 bytes at a time and iteratively sending the data and updating the hash code until the end of the file is reached. At this point, the end tag is sent followed by the hash code. The server response regarding the comparison of hash codes is then received. The result of this comparison is then printed before the socket and application is closed.

PDCPRA001: When the client is run, it prompts the user to input the IP and port which they would like to connect to. The client then connects to the server. The server asks the client to make a choice dictating whether they want to upload a file to the server or download one from the server. When the client requests to download from the server, the downloadFile() method is run which prompts the client to enter sufficient information for the download to take place (correct file name and password). Thereafter, the getFile() method is run which retrieves the file data from the server and creates a file on the client's machine. When the client asks to upload a file, the uploadFile() method is called which similarly exchanges in a dialogue with the server such that all the necessary information for file upload is retrieved. The method directly sends the data to the server where it is then saved to the server database. To handle the formatting of messages sent to the server, I created variables at the top of the script holding data for the header tags that are sent to the server. At many points of the client-server dialogue, the client may escape the program by typing 'Q'. This is to avoid the user being stuck in awkward program states. Error handling is also used wherever an obvious problem may occur so that the client gracefully exits and outputs an error message.

MSLGRE001: When the client runs, it prompts the user to enter the connection details of the live server. Once connected, the client queries the user whether they want to upload to the server, download from the server or quit. If the user wants to download from the server, the client receives communication from the server and either denies or permits downloading depending on the existence of files in the database (no files means no downloading). If there are files present, the client proceeds to query the user for the files' passwords (if needed). After the password is confirmed, the file in question is received chunk-by-chunk from the server in chunks of 4096 bytes. As the chunks are written in, a hash is updated in order to confirm successful file delivery later on. When the file is finished downloading (indicated by the presence of an "<END>" tag), the hashes of the written file and the hash calculated by the server are compared. If they are the same, it is a successful download and the client closes their connection. If they are not the same, the file is deleted and the client requests the user to reconnect and retransmit. Now, in the case of uploading a file, the client queries the user to input the name of the file they want to upload. After checking for the presence of this file and checking that a file with that name doesn't already exist in the database, the user will then be prompted to choose a privacy for the file. If they choose it to be open, the file will then be uploaded to the server chunk-by-chunk with an accompanying hash that is updated every file-read. If they choose it to be 'closed' then they are prompted to input a password that is at least 1 character in length. The password file is updated with appropriate data in both open and closed cases and a hash is checked on the server to see if the transfer was successful. Throughout many different stages in the program, the user has an option to enter 'Q' to quit the program and close the connection.

Protocol Specification:

The main functionality of the protocol lies in the use of headers, and how those headers are keys to understanding the purpose of the data they precede. The header in this program is of the following form:

<x><xxxxxxx>DATA
<TYPE><SUBTYPE>DATA

- <x> : This is a single integer ranging from 0 to 2. Each integer represents the type of message that is being transmitted. The 3 types of messages can be described as control, command, data. <0> means it is a control message, <1> meaning command, and <2> meaning data.
- <xxxxxxx> : This is a 7 digit code that narrows down the "subtype" of the message. Each message is accompanied by this 7 digit code to express the functionality of the message. For example, a subtype code of "REQFILE" would be used to indicate that a message was sending the file required by either server or client. Here is a breakdown of the used subtypes:

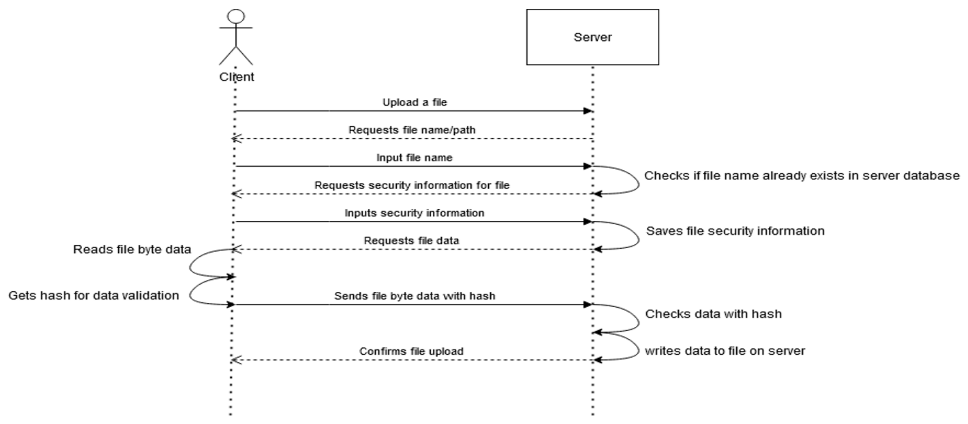
CLIRQST = Client request	SFILSEN = Server File Send	PASSTRY = Password Attempt
HEXSEND = Hash Send	PASSEND = Password Send	PACCEPT = Password Accept
FILPRIV = File Privacy	FILENAM = File name send	PREQUES = Password Request
NMREJCT = Name Rejection	PREJECT = Password Reject	INVNAME = Invalid Name
REQFILE = Request File Name	LISTFIL = List Files	PRIVREC = Receive Privacy
CONNSCS = Connection Success	RCVREDY = Ready to Receive	NONPASS = Password not needed
INVDATA = Failed Hash	VLDDATA = Successful Hash	MPTYDTB = empty database

- DATA : This field represents the data of the message. This can be in the form of a string or it may be a sequence of bytes. When file data is being sent, the data will have a "tail" which is used to properly process the file data ("<END>").

The subtype of the message is of particular importance. It is used to communicate between the server and client whether repeated actions are required when the client is inputting information. An example: when the server sends a message with the subtype "PREJECT" a while loop in client is capable of repeatedly requesting the user to keep trying the password for up to 2 more attempts.

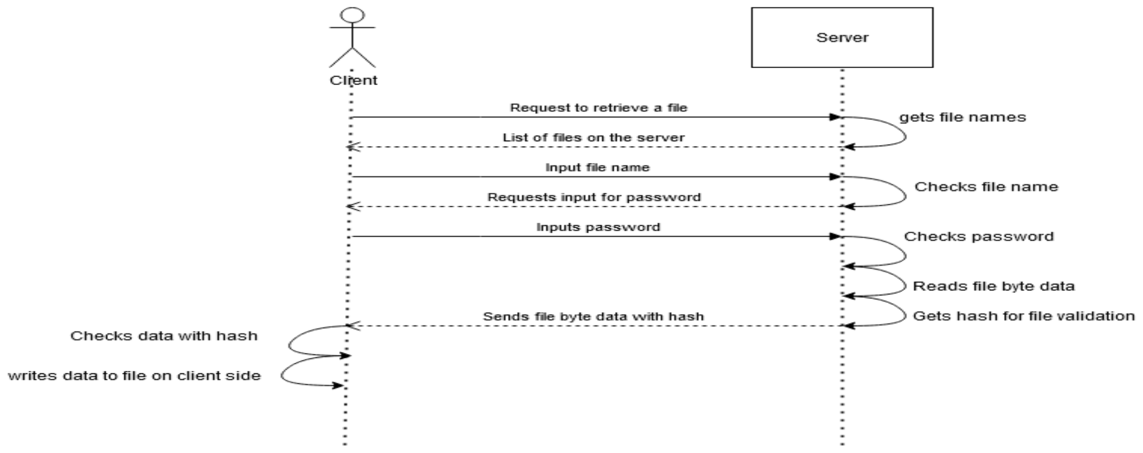
Sequence Diagrams:

Client uploads a file to the server:


Client
Server
Upload a file
Requests file name/path
Input file name
Checks if file name already exists in server database
Requests security information for file
Inputs security information
Saves file security information
Requests file data
Reads file byte data
Gets hash for data validation
Sends file byte data with hash
Checks data with hash
writes data to file on server
Confirms file upload

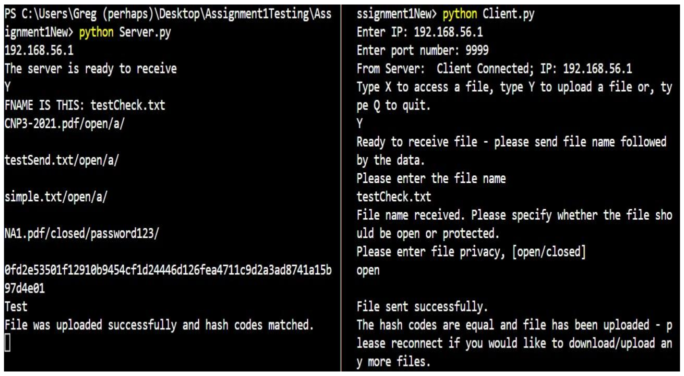
This diagram explains the sequence of events when the client selects to upload a file to the server. The server requests for the client to input all necessary information for the file to be uploaded (the file name, password and location) and then sends the data to the server. The file data is validated, by the use of hashing, before a copy is saved to the server.

Client downloads a file from the server:


Client
Server
Request to retrieve a file
gets file names
List of files on the server
Input file name
Checks file name
Requests input for password
Inputs password
Checks password
Reads file byte data
Gets hash for file validation
Sends file byte data with hash
Checks data with hash
writes data to file on client side

When the client requests to download a file from the server, the server will display all the files it has stored. The client must then input all the required information to download a file (the file name and password) whereafter the server will respond to the client by sending the file data. Similarly to the client upload sequence, the file data is validated by the use of a hash before it is saved to the client.

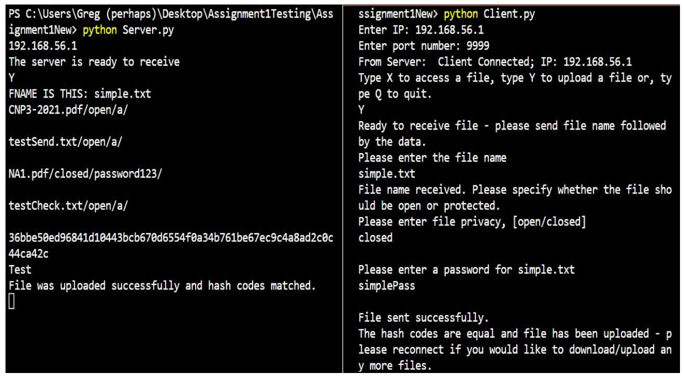
SECTION 3:



```
PS C:\Users\Greg (perhaps)\Desktop\Assignment1Testing\Ass
ignment1New> python Server.py
192.168.56.1
The server is ready to receive
Y
FNAME IS THIS: testCheck.txt
CNP3-2021.pdf/open/a/

testSend.txt/open/a/

simple.txt/open/a/

NA1.pdf/closed/password123/

0fd2e53501f12910b9454cf1d24446d126fea4711c9d2a3ad8741a15b
97d4e01
Test
File was uploaded successfully and hash codes matched.
```

```
ssignment1New> python Client.py
Enter IP: 192.168.56.1
Enter port number: 9999
From Server:  Client Connected; IP: 192.168.56.1
Type X to access a file, type Y to upload a file or, ty
pe Q to quit.
Y
Ready to receive file - please send file name followed
by the data.
Please enter the file name
testCheck.txt
File name received. Please specify whether the file sho
uld be open or protected.
Please enter file privacy, [open/closed]
open

File sent successfully.
The hash codes are equal and file has been uploaded - p
lease reconnect if you would like to download/upload an
y more files.
```

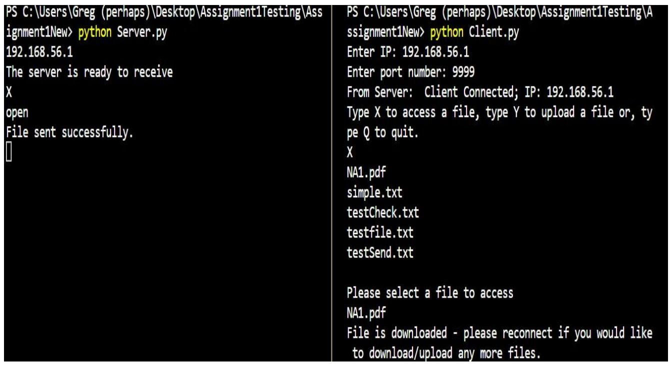
Fig 1: Uploading an open file to server.



```
PS C:\Users\Greg (perhaps)\Desktop\Assignment1Testing\Ass
ignment1New> python Server.py
192.168.56.1
The server is ready to receive
Y
FNAME IS THIS: simple.txt
CNP3-2021.pdf/open/a/

testSend.txt/open/a/

NA1.pdf/closed/password123/

testCheck.txt/open/a/

36bbe50ed96841d10443bcb670d6554f0a34b761be67ec9c4a8ad2c0c
44ca42c
Test
File was uploaded successfully and hash codes matched.
```

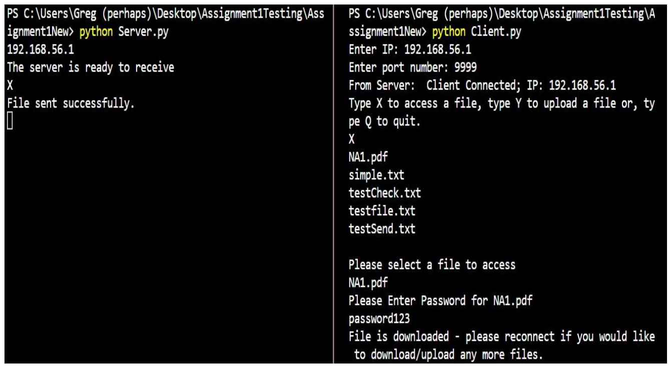
```
ssignment1New> python Client.py
Enter IP: 192.168.56.1
Enter port number: 9999
From Server:  Client Connected; IP: 192.168.56.1
Type X to access a file, type Y to upload a file or, ty
pe Q to quit.
Y
Ready to receive file - please send file name followed
by the data.
Please enter the file name
simple.txt
File name received. Please specify whether the file sho
uld be open or protected.
Please enter file privacy, [open/closed]
closed

Please enter a password for simple.txt
simplePass

File sent successfully.
The hash codes are equal and file has been uploaded - p
lease reconnect if you would like to download/upload an
y more files.
```

Fig 2: Uploading a closed file to server.



```
PS C:\Users\Greg (perhaps)\Desktop\Assignment1Testing\Ass
ignment1New> python Server.py
192.168.56.1
The server is ready to receive
X
open
File sent successfully.
```

```
PS C:\Users\Greg (perhaps)\Desktop\Assignment1Testing\A
ssignment1New> python Client.py
Enter IP: 192.168.56.1
Enter port number: 9999
From Server:  Client Connected; IP: 192.168.56.1
Type X to access a file, type Y to upload a file or, ty
pe Q to quit.
X
NA1.pdf
simple.txt
testCheck.txt
testfile.txt
testSend.txt

Please select a file to access
NA1.pdf
File is downloaded - please reconnect if you would like
 to download/upload any more files.
```

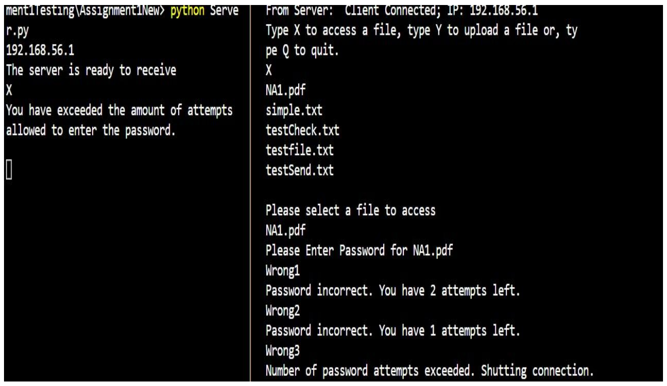


```
PS C:\Users\Greg (perhaps)\Desktop\Assignment1Testing\Ass
ignment1New> python Server.py
192.168.56.1
The server is ready to receive
X
File sent successfully.
```

```
PS C:\Users\Greg (perhaps)\Desktop\Assignment1Testing\A
ssignment1New> python Client.py
Enter IP: 192.168.56.1
Enter port number: 9999
From Server:  Client Connected; IP: 192.168.56.1
Type X to access a file, type Y to upload a file or, ty
pe Q to quit.
X
NA1.pdf
simple.txt
testCheck.txt
testfile.txt
testSend.txt

Please select a file to access
NA1.pdf
Please Enter Password for NA1.pdf
password123
File is downloaded - please reconnect if you would like
 to download/upload any more files.
```

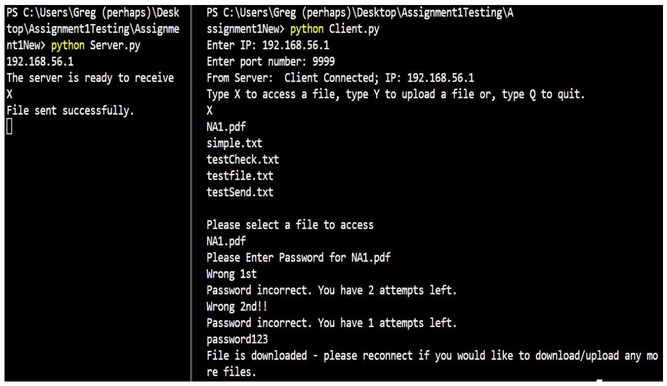
Fig 3: Downloading open file from server.

Fig 4: Downloading closed file from server with 1 (successful) attempt



```
ment1Testing\Assignment1New> python Serve
r.py
192.168.56.1
The server is ready to receive
X
You have exceeded the amount of attempts
allowed to enter the password.
```

```
From Server:  Client Connected; IP: 192.168.56.1
Type X to access a file, type Y to upload a file or, ty
pe Q to quit.
X
NA1.pdf
simple.txt
testCheck.txt
testfile.txt
testSend.txt

Please select a file to access
NA1.pdf
Please Enter Password for NA1.pdf
Wrong1
Password incorrect. You have 2 attempts left.
Wrong2
Password incorrect. You have 1 attempts left.
Wrong3
Number of password attempts exceeded. Shutting connection.
```

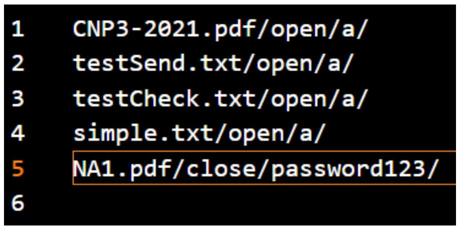


```
PS C:\Users\Greg (perhaps)\Desk
top\Assignment1Testing\Assignme
nt1New> python Server.py
192.168.56.1
The server is ready to receive
X
File sent successfully.
```

```
PS C:\Users\Greg (perhaps)\Desktop\Assignment1Testing\A
ssignment1New> python Client.py
Enter IP: 192.168.56.1
Enter port number: 9999
From Server:  Client Connected; IP: 192.168.56.1
Type X to access a file, type Y to upload a file or, type Q to quit.
X
NA1.pdf
simple.txt
testCheck.txt
testfile.txt
testSend.txt

Please select a file to access
NA1.pdf
Please Enter Password for NA1.pdf
Wrong 1st
Password incorrect. You have 2 attempts left.
Wrong 2nd!!
Password incorrect. You have 1 attempts left.
password123
File is downloaded - please reconnect if you would like to download/upload any mo
re files.
```

Fig 5: Downloading closed file from server, 3 failed password attempts.

Fig 6: Attempting download from server, password failed 3 times.



```
CNP3-2021.pdf/open/a/
testSend.txt/open/a/
testCheck.txt/open/a/
simple.txt/open/a/
NA1.pdf/close/password123/

```

- **Fig 7:** An example of what passwords.txt might look like. Note the use of hashes in **Fig 1** and **2**.