

## Group 7

Jordan Thomson - THMJOR002

Kane Gibson - GBSKAN001

Daniel Gordan - GRDDAN017

### Brief discussion on the server

The server will enable clients to upload and download files. Clients are able to search for all the files that are available for download on the server. These files can be protected or open. The client who uploads the file decides on its protection. The protected files require a password or key. Only once the correct password is entered, can a protected file be downloaded. An incorrect password will result in an informative error message. This server is designed so that no corrupt files can be sent or downloaded. A checksum validation method is used to ensure this integrity. We will describe it in further detail later.

### Features

The server will constantly listen for clients to request a connection. Once the **client has connected** to the server, it will indicate that a new connection has been established. In this server the feature of **multi-threading** has been included. Multi-threading will enable the server to **handle multiple clients simultaneously**. This removes the possibility of client requests causing delays/ interfering with each other on the server.

The two main functions of the server are downloading and uploading of files. When **uploading or downloading** files, a mathematical formula calculates **checksum values** that make it possible to validate the **integrity** of the files at either end system. The TCP protocol itself validates files, but to increase server security further, the checksum validation procedure has been included.

When downloading, a user can request to download a file called '**ListOfFiles.txt**'. After this specific call, the user must provide a list of keys they have for specific protected files (if any). When the server receives this message (that includes the list of keys), it will return a **list of files** which the client has **permission to see**. Clients will be unable to see that a protected file exists for download if they don't have the correct password for it, making them **invisible**.

**Error checking** methods with comprehensive and informative feedback have been implemented. For example, clients are not allowed to upload files that have the same name as an **already existing file** on the server side. The client will be notified and any accidental overwriting is prevented. When entering incorrect keys, users are notified that their password is incorrect. Whether a function request was successful or unsuccessful, the user will be **notified** and the sequence will restart.

The host will store its files in a directory called 'serverStorage'. This means that in the off-chance that the server owner wishes to run a client on the same computer (for testing purposes), there will be no accidental **overwriting** of files (i.e. server and client files not stored in the same directory).

**Any file type** can be uploaded/downloaded, there are no limitations. This has been made possible by the transfer of data as bytes. All information is stored in bytes by computers, and so, we are reading and writing files using bytes allows us to transfer every type of file.

## **Command Messages**

Input is prompted from the user. Some information includes the function (download/upload), filename, and protection(open/protected) etc. The user will provide this information, and their responses will determine the command messages. These command messages indicate whether the client would like to **upload** or **download** a file which will in turn **trigger an action**. Once all the user commands are collected, the server will call the command message indicated and trigger the next stage which is the data transfer.

## **Data Transfer Messages**

Data transfer messages will carry the file **binary data** that will be exchanged between the server and the client. Reading and writing of binary data allows for the transfer of any file.

The client will then indicate to the server whether to upload or download. For an **upload** request the selected file will transfer from the client to the server. For a **download** request the selected file transfers from the server to the client.

## **Control Messages**

Control messages are used to communicate **dialog** between the server and the client. This will ensure that the data transfer between the server and the client is reliable. These messages will also indicate to the client or the server if any **errors** occur and **why** they happened. Meaningful and informative feedback is essential and very useful. This server will include control messages such as whether data transfer of files was successful or not and whether an error had occurred. These messages will **improve communication** between the server and the client, and this extra effort shows professionalism by the server.

## **Header and Body design – When client sends to server**

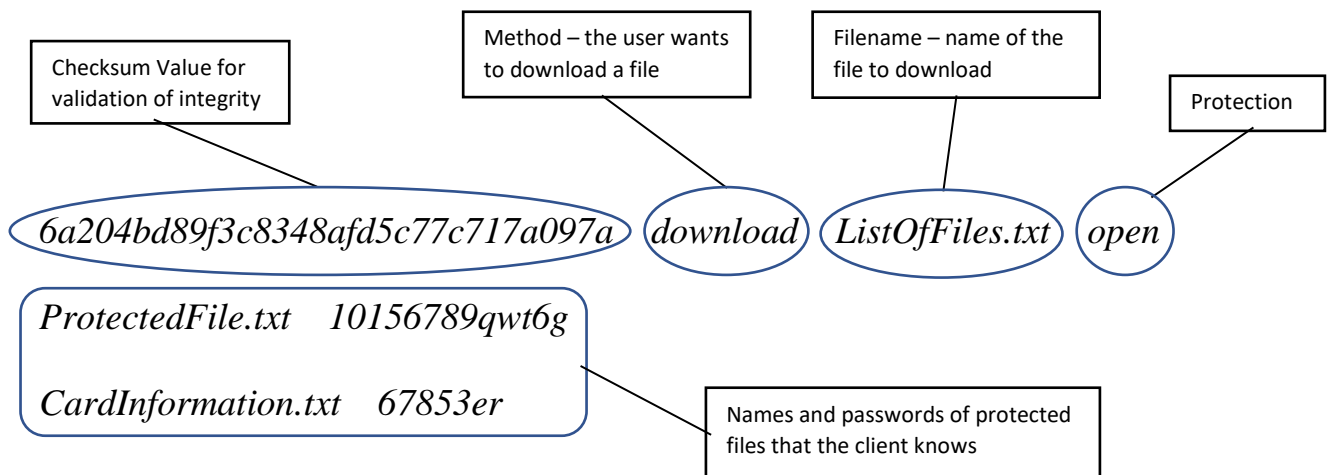
The header design, shown below, is what the **client sends to the server**. The **delimiters** are just blank spaces. The first piece of text will indicate the **length** of the **following message** and will be followed by the checksum. This part of the header will provide data validation and will make sure no corrupted files will be uploaded or downloaded. The next piece of text will define what the user wants to do and what **method** the server needs to run. The two methods that are included are an “**Upload**” method and a “**Download**” method. It is then followed by the **file name**. Whether uploading or downloading, the user will always have to enter a filename. For uploading, the filename is the name of the file the user wants to upload and for downloading, the filename is the name of the file the user wants to download. The header is then followed by a word, either Open or Protected. For uploading a file, this will indicate whether the file being uploaded will be open for everyone to see and download or only accessed by certain people who have the password to this file. For downloading, this will indicate whether the user will have to provide a password to download the file. The header then goes to the next line.

The next part of the header will be a list of files and their corresponding keys. If the user wants to **upload a protected** file, the header will state the name of the file once again followed by the password for that file that the user decided on. If the user wants to **download a protected** file, the

filename will have to be stated once again followed by the password to that file. If the user is downloading the 'ListOfFiles.txt' file, they will have to supply the **list** of files and their keys that they have. This is so that the server can send a list of available files to the client whilst keeping files they don't have the correct password for '**invisible**'. If the client has no passwords, it is left blank.

If the user is uploading or downloading an **open** file, this section will be blank (i.e. no passwords need to be provided). After the header, the body follows. The **body** is the actual data of the file being uploaded or downloaded.

### Example of a message



**OR**

`6a204bd89f3c8348afd5c77c717a097a upload message1.txt open`

*b'This is the message in the file'*

## Client-to-server header fields (protocol)

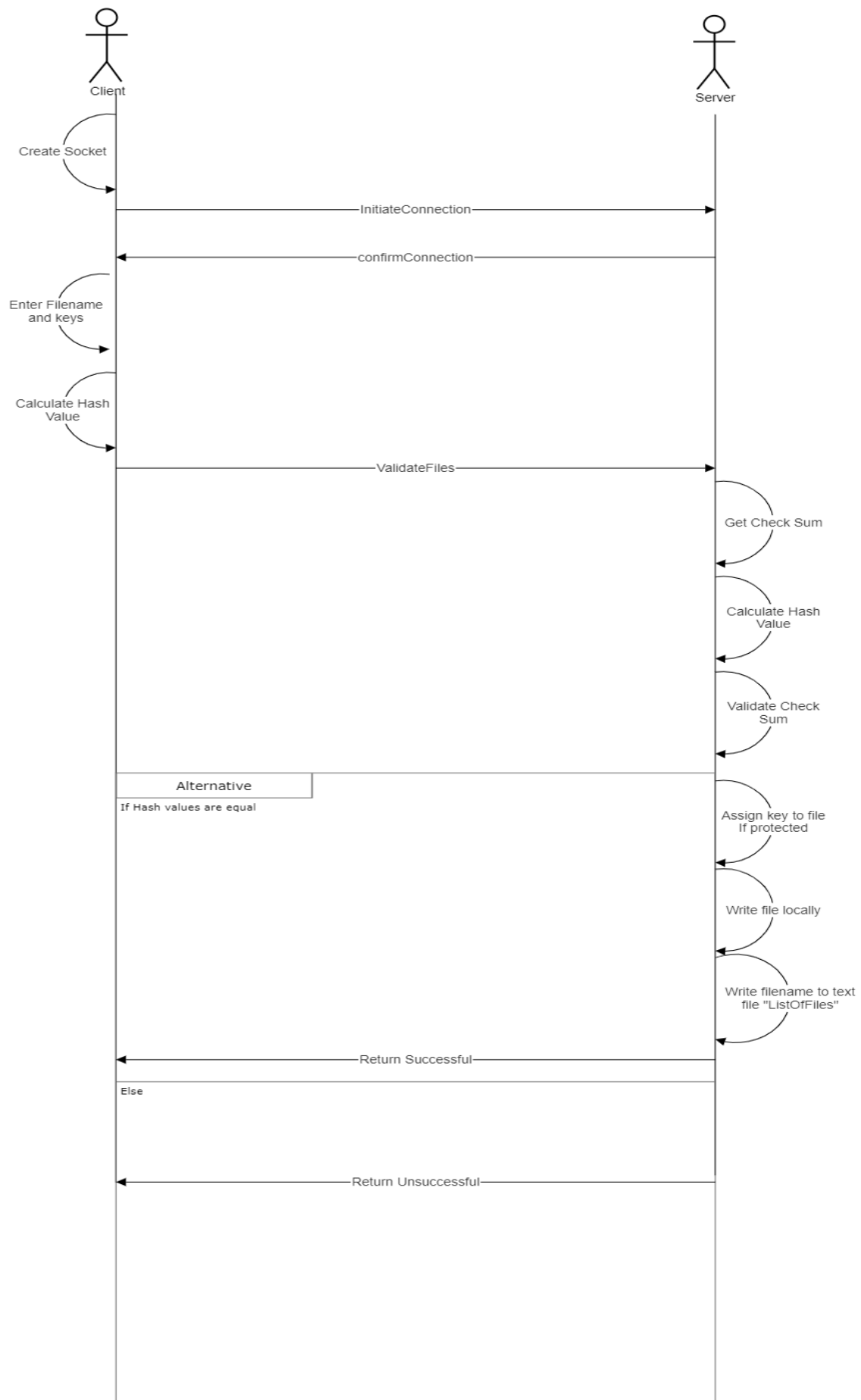
Length of message	sp	Checksum	sp	Method (Upload/Download)	sp	File name	sp	Protection (Open/protected)	cr	lf
Protected file name (that client has a password/key for)	sp	Password/key for file	cr	lf						
...	...	...	...	...						
Protected file name	sp	Password/key for file	cr	lf						
cr	lf									
Entity body										

## Server-to-client header fields (protocol)

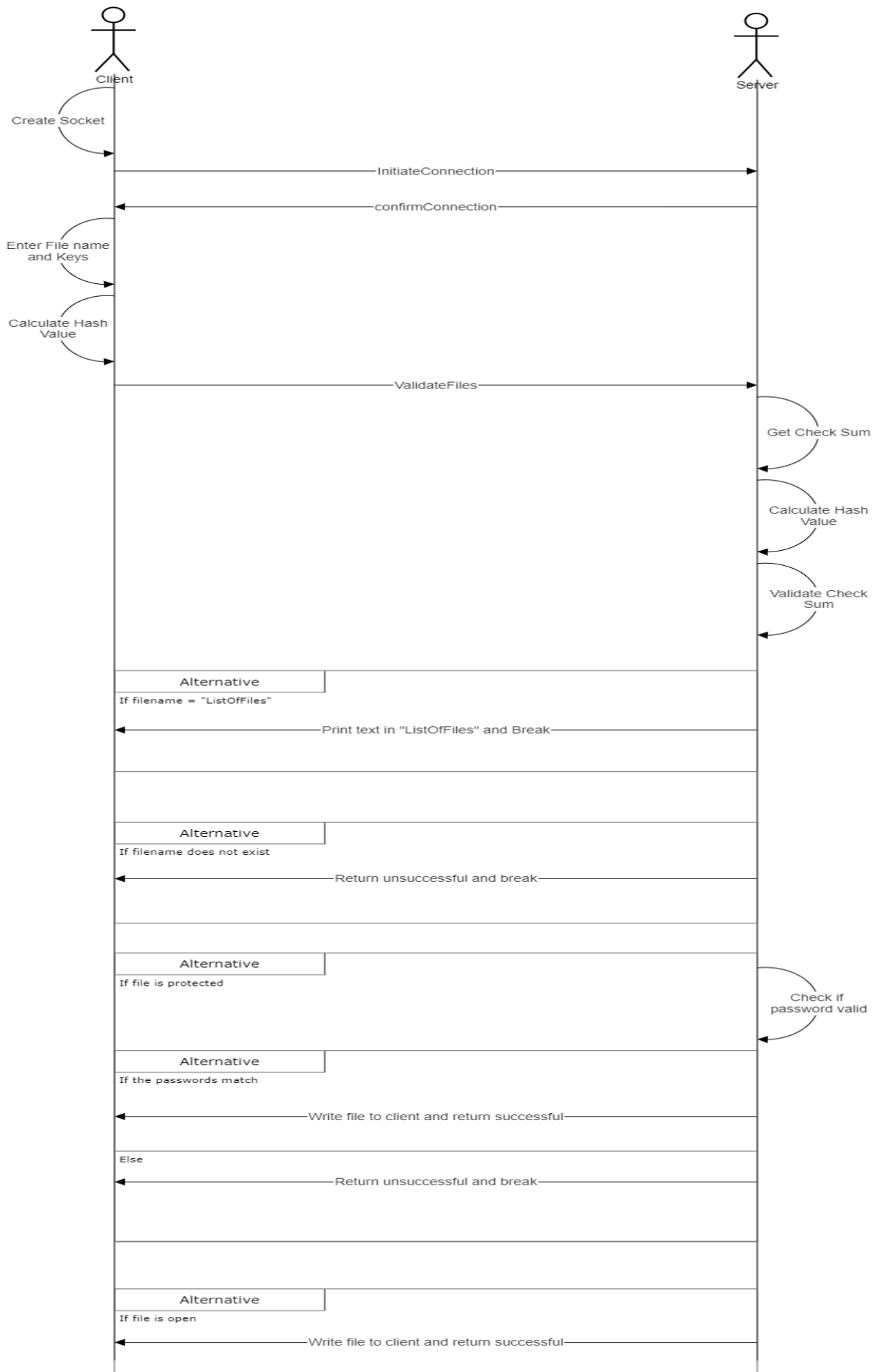
Length of message	sp	Checksum	sp	Success (successful/unsuccessful)	cr	if
Entity body						

This header design will be what the server sends back to the client. The **delimiters** are just blank spaces. It starts off the same as the other header design **beginning** with the “**Length** of message” followed by the **checksum** for data validation. It will then provide the user with confirmation on whether their task was successful or not whether they downloaded a file or uploaded a file. If unsuccessful, the **body** will contain a **reason** for action being a failure. If they requested for a **list of files**, this section will contain the list.

# Upload Sequence Diagram



## Download sequence diagram



## Client Implementation (Kane Gibson GBSKAN001)

This file transfer application enables the client to upload and download files to and from the server. The features and functionality that this application provides can be utilized by the user in a clear, efficient, and fault-tolerant manner. Through a series of prompt messages, the user will input their desired request. The client backend will take user requests, format them, send them to the server for processing, and await a server response. The client will keep the user notified on the status of the request and consistently display informative feedback and output.

### Features

Input prompts are always available and waiting for the user to give a request. This is done using an infinite loop. The very first **input** requested from the user is the function that they would like to perform. The four functions are upload, download, list, and add. The users input '**u**' if they want to upload a file to the server, and '**d**' if they want to download a file from the server. They input '**l**' to get list of the available files on the server, and '**a**' to add a new key they have for a protected file. Depending on this input, the client will continue to prompt the user for information needed to upload or download a file. Here are screenshots of the functions in action:

#### Upload:

```
Would you like to upload(u), download(d), list the available files for download (l),
or add a key for a protected file (a)? (type 'q' to quit)
u
Enter the name of the file to upload. (type 'q' to quit)
IMG.png
Are you uploading a protected(p) or open(o) file? (type 'q' to quit)
o
[UPLOAD SUCCESSFUL] - the file has been succesfully uploaded to the server
```

#### Download:

```
Would you like to upload(u), download(d), list the available files for download (l),
or add a key for a protected file (a)? (type 'q' to quit)
d
Enter the name of the file to download. (type 'q' to quit)
Biggest.jpg
[DOWNLOAD SUCCESSFUL] - file successfully received and saved
```

In particular, if a user wants to download a protected file, they simply need to input the name of the file and not the key. The client will itself search through the list of keys that this user has and automatically add it to the message header. This is extremely convenient for the user who will not have to manually look for the keys for protected files every time.

#### List:

```
Would you like to upload(u), download(d), list the available files for download (l),
or add a key for a protected file (a)? (type 'q' to quit)
l
Available files on server:
Biggest.jpg (open)
Biggest.JPG (open)
BigOne.JPG (open)
cross.jpg (protected)
```

For user convenience and ease-of-use, when typing the command **list (l)**, the client code will automatically make the method 'download' and the filename 'ListOfFiles.txt' for the header information – one line of input instead of three confusing lines.

### Add:

```
Would you like to upload(u), download(d), list the available files for download (l),
or add a key for a protected file (a)? (type 'q' to quit)
a
Enter the name of the file you have a key for. (type 'q' to quit)
tutor.jpg
Enter the key for the file tutor.jpg. (type 'q' to quit)
100%Please
[INPUT SUCCESSFUL: key and file successfully added to list]
```

A client can also immediately end the connection with the server by simply entering the letter 'q' as input. They quit whenever they want to (i.e. they are not required to finish a request that they may accidentally have started). This option is always made clear to the user.

It is important to prevent erroneous data from entering the system and giving unknown error messages when things do go wrong. So, the app is also very fault-tolerant. Some error prevention and data validation techniques (besides checksum previously explained) include:

The exclusion of space characters in file names:

```
Would you like to upload(u), download(d), list the available files for download (l),
or add a key for a protected file (a)? (type 'q' to quit)
u
Enter the name of the file to upload. (type 'q' to quit)
Hello there
File names cannot include space characters
```

Returning clear and informative error messages when an invalid command is given:

```
Would you like to upload(u), download(d), list the available files for download (l),
or add a key for a protected file (a)? (type 'q' to quit)
w
Invalid command.
Would you like to upload(u), download(d), list the available files for download (l),
or add a key for a protected file (a)? (type 'q' to quit)
```

Checking to see if file exists:

```
Would you like to upload(u), download(d), list the available files for download (l),
or add a key for a protected file (a)? (type 'q' to quit)
u
Enter the name of the file to upload. (type 'q' to quit)
100%ForMyAssignment
File does not exist
```

And ensuring certain files exist before they are accessed. For example using this method:

```
# Checks if file exists, else it creates it
def checkListOfKeys():
    file_path = os.path.join(HERE, 'ListOfKeys.txt')
    if not os.path.exists(file_path):
        f = open(file_path, 'wb')
        f.close()
```

Finally, once all required information has been collected and validated, the message is built and sent to the server in the correct format (client-to-server protocol explained previously). The client waits for the server response and outputs the result of the file transfer (successful/unsuccessful) to the user. If a successful download is achieved, the client will store the received/downloaded file and notify the user.



# GRDDAN017 INDIVIDUAL REPORT ON CLIENT

## Features and Description:

When the client is run for the first time, the user is prompted to enter an **IP address** for the host they wish to connect to (function `getIP()` is called). This prompt will test whether the address entered is **valid** or not (an example of **error checking**). The one exception is if the user enters '**local**' as the host IP. This will connect them to the host if it is running on their local computer.

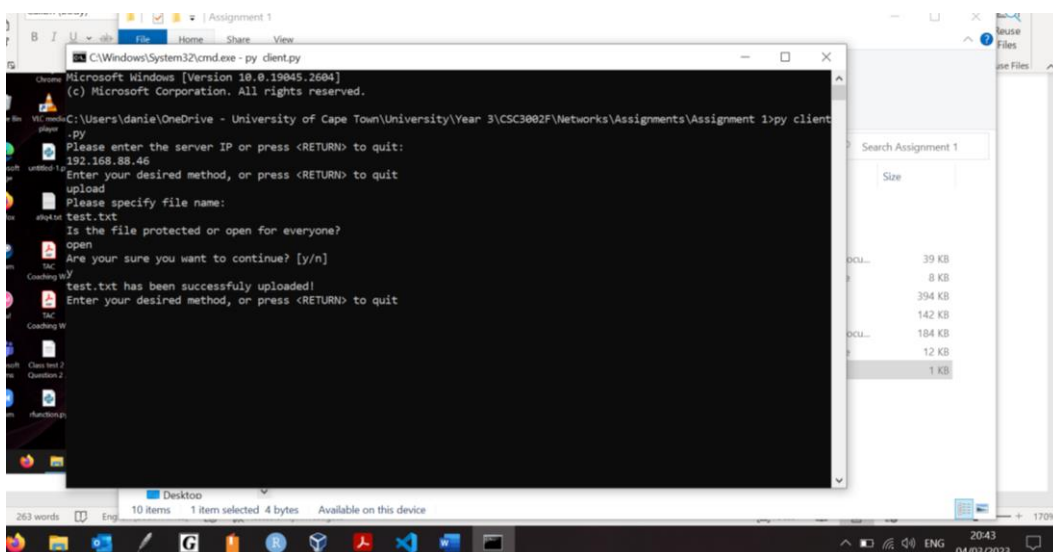
After a **successful connection** is made with the host, the user is entered into a loop in which they are repeatedly asked for input until they press enter (function `user_io()` is called and manages input and output). The user is asked for the method they wish to perform, the filename and the status of the file (open or protected). If the file they wish to upload/download is protected, they must specify the **passcode** for that file. If they wish to **download the ListOfFiles.txt**, then they are prompted to enter a **list** of files and passcodes until they just press enter. During an **instance** of the program running, any keys entered by the user during this time will be **remembered** for later by storing them in a **dictionary**. This means that if they want to get the list of files later in that same instance of the program, they won't have to enter previously entered keys again. Checks are done to ensure files aren't uploaded with spaces in their name and that valid instructions are entered into the console.

The client calculates **hashed** values of messages (using MD5) before sending them and also after receiving them so that it can make comparisons and determine whether something was corrupted or not.

The program checks whether the method they enter is valid. Users are prompted before **performing** any action whether they wish to continue after this point or **cancel** their action.

When the user is **finished** and decides to press return, a closing message is sent to the server and the program is closed.

## Screenshots of the client in action:



*Figure 1: Connecting to a host running on a different machine and uploading a file.*

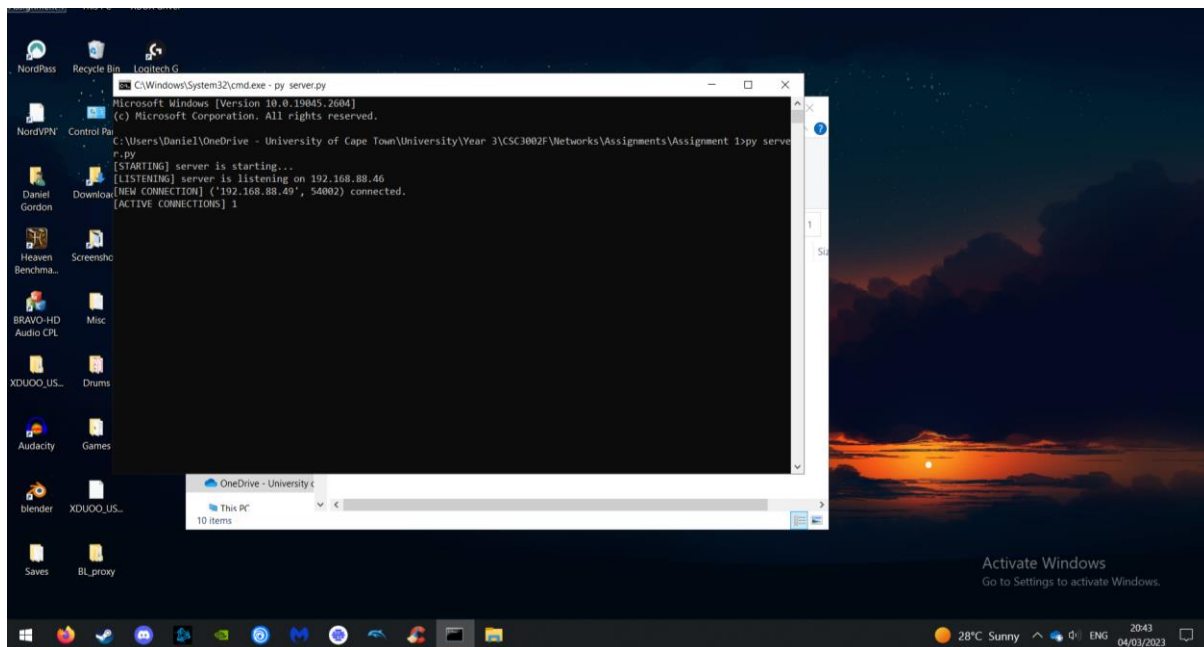


Figure 2: Perspective from the other machine running host.

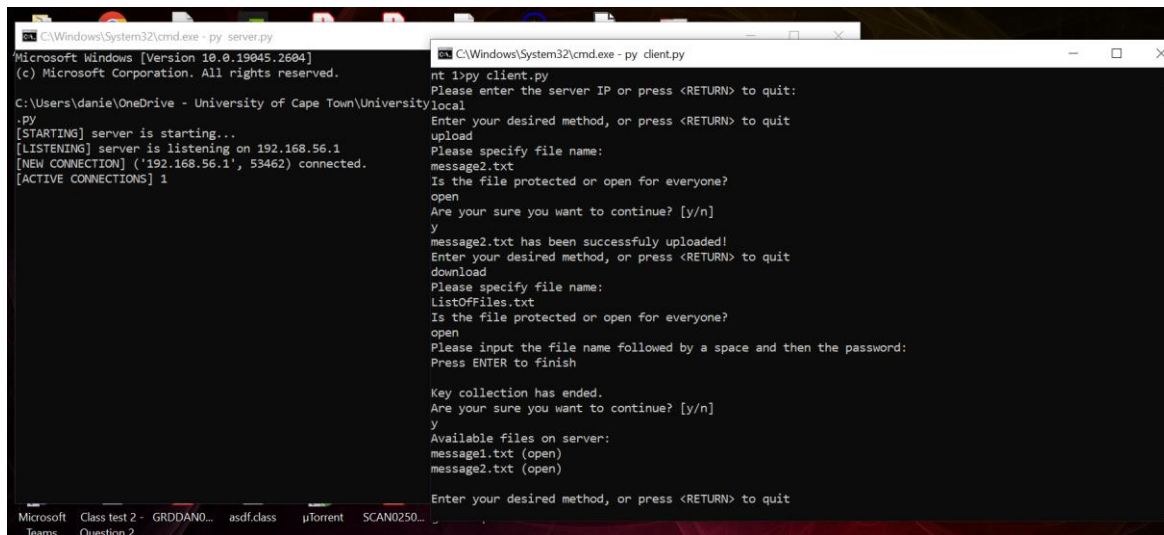
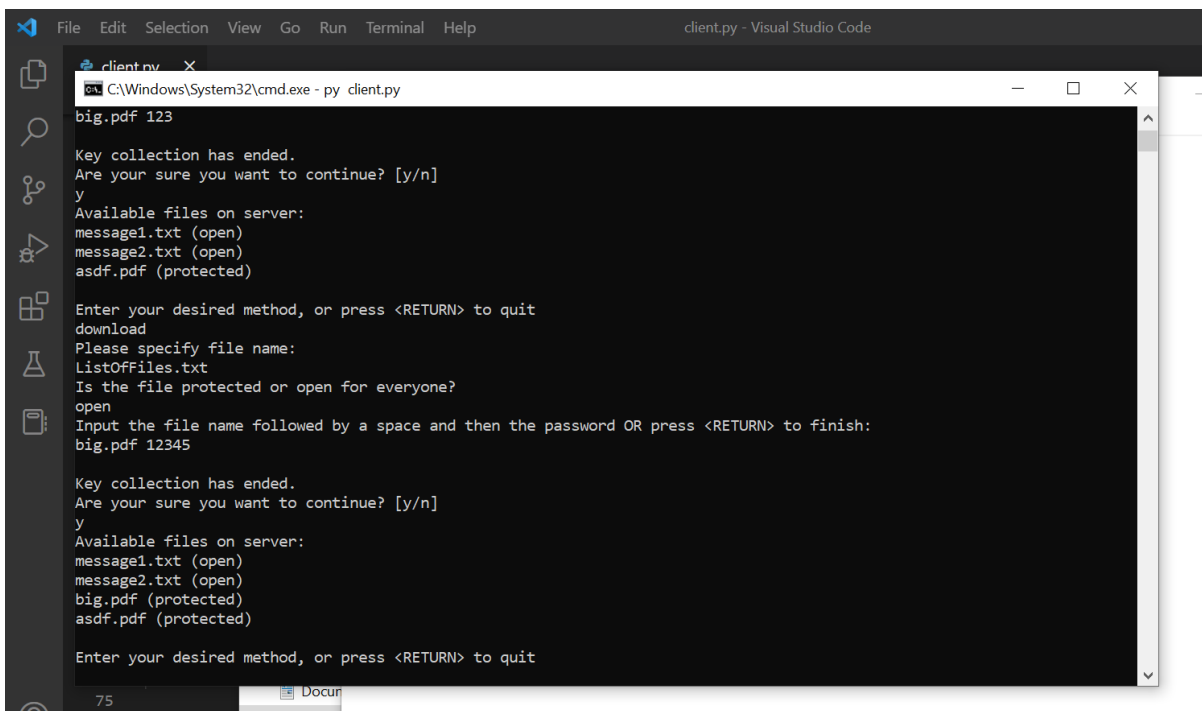


Figure 3: Running host and client on the same machine. Asking for a list of files without any keys (just pressing return).



```
client.py - Visual Studio Code
C:\Windows\System32\cmd.exe - py client.py
big.pdf 123

Key collection has ended.
Are your sure you want to continue? [y/n]
y
Available files on server:
message1.txt (open)
message2.txt (open)
asdf.pdf (protected)

Enter your desired method, or press <RETURN> to quit
download
Please specify file name:
ListOfFiles.txt
Is the file protected or open for everyone?
open
Input the file name followed by a space and then the password OR press <RETURN> to finish:
big.pdf 12345

Key collection has ended.
Are your sure you want to continue? [y/n]
y
Available files on server:
message1.txt (open)
message2.txt (open)
big.pdf (protected)
asdf.pdf (protected)

Enter your desired method, or press <RETURN> to quit
```

*Figure 4: Calling for the list multiple times with passwords for protected files. (Very top got cut off but I entered the correct key for 'asdf.pdf' but the wrong one for 'big.pdf'). This illustrates how the client remembers entered keys in that instance.*

## **Client Implementation - THMJOR002 - Jordan Thomson**

The client code is a Python program which enables any device that can execute it, to upload and download any file types.

The client is first greeted by a prompt welcoming them to the server. It then asks the user whether they would like to upload a file or download a file. The client must then enter the letter “u” for upload, the letter “d” for download and the letter “q” to quit the program. If the user enters anything else, an error will prompt the user to try again. The user is then prompted to enter the name of the file they want to upload or download. If the filename is “ListOfFiles.txt” for downloading purposes, the program will print out the list of files available to them on the server but if it is for uploading purposes, the program will tell the user that you are not allowed to upload a file that has that file name and will restart the process. They are then prompted to say whether the file is open or protected. If it is open then the program will just upload or download the file. If it is protected for uploading purposes, the user must enter the password they want to assign to that file and if it is for downloading purposes, the user must enter the password for the file they want to download.

There are many functions included in this program. The send() function sends files to the server as well as the length of the message along with the header. The hash value of the message is calculated which is then sent to the server. This enables the program to check whether the file has been corrupted.

The receive() function will handle messages that are sent from the host. It also receives a header in a different format that was discussed previously. The file will be validated for by comparing the hash values calculated by the server and the client.

The uploadFile() function will call the send() function with the appropriate parameters to upload a file and the downloadFile() function will do the same. The main difference between these two functions is that if you enter the file name, “ListOfFiles.txt” the download function will print out the files available and the upload function will restrict you from uploading that file.

The userIO() function will prompt the client to select to upload or download a file. It will then execute the appropriate functions.

### **Client example 1**

```
Welcome to the file sharing server! Would you like to upload or download ?
Enter (d/u), or press q to quit
d
Please enter the file name you want to download:
ListOfFiles.txt
open
Please enter the file names and passwords one by one.
When you are completed, hit ENTER.
```

```
available files on server:
daniel.txt (open)
kane.txt (open)
Enter (d/u), or press q to quit
q
Quitting the program
```

### **Example 2**

```
Welcome to the file sharing server! Would you like to upload or download ?
If you would like to see all the files available to you on the server, please select the download option followed by the file name 'ListOfFiles.txt'
Enter (d/u), or press q to quit
d
Please enter the file name you want to download:
daniel.txt
Is the file open or is it protected ?
open
daniel.txt has been downloaded!
```

### Example 3

```
Enter (d/u), or press q to quit
u
Please enter the file name you want to upload:
Jordan.txt
Is the file open or is it protected ?
protected
Please enter the password for the file:
donth4ck
Jordan.txt has been uploaded!
```