Socket-Based File Sharing Server

Computer Networks and Design

CNT 3004 Section 4

10/29/2024

Ryan Moorgen

Payne Persons

Matthew Rigg

Table of Contents

[Introduction and Objective 2](#_lzy814zerkyy)

[System Architecture and Design 3](#_esmuzmwybo1d)

[Implementation Details 5](#_betghu9q3mgt)

[Experimental Results and Analysis 8](#_hvqq39v1jd0d)

[Problems Faced 11](#_xma5hpmrbdc3)

[What was learned 12](#_8ixtwdsnruu3)

[Contribution Table 13](#_7arcl6m95pvb)

[Conclusions and Future Work 14](#_yrpmnbjmpy0w)

## 

## Introduction and Objective

Computer networking is the process of connecting devices to share data and resources. This is accomplished through the development of servers, which acts as a medium for multiple clients to interact with and share material across devices. One such example of this is the internet, where many people can open a search engine like Google.com and be able to access a multitude of various articles, videos, and other content being made by people all across the globe and shared amongst everyone.

The purpose of this project is to create a small scale network that aids in being able to perform file transfers over local computers. The functions included in the implementation of our program includes file uploads and downloads, the ability to delete files, error handling for overwriting files, being able to create subfolders, and being able to create and monitor various directories. In addition to these features, further implementation on the security front of the program leads to the use of encryption keys and passcodes which help with monitoring and restricting access to unauthorized users.

This project has a wide range of practical applications, such as for small businesses, educational institutions, and personal use where secure and efficient file sharing is essential. By enabling local file transfers with added features such as directory management and user authentication, this system serves as a cost-effective solution for small scale environments. The integration of encryption keys and passcode protections not only enhances security but also builds trust among users, ensuring that sensitive data remains private. With potential scalability, this project can serve as a stepping stone for developing more advanced networking systems, opening up opportunities for collaborative workspaces, automated backups, and streamlined communication within a network.

## System Architecture and Design

This project has three categories that all components of the architecture fall under, software, security, and hardware.

There are two software components. The first is the three python files, Client.py, sever.py, and analysis\_component.py. The Client.py file contains all of the information and functions necessary for a client to connect and interact with the server. These functions include uploading files to the server, downloading files from the server, sending commands to the server, including asking for the creation of a subfolder, the deletion of a file, and to send a list of directories to the client, and sending a ping to the server to test the connection between the two. The server.py file contains all of the information and functions necessary for the server to run, and manage and interact with clients. These functions include receiving a file that the client uploaded, sending a file for the client to download, receiving commands to, among other things, create a subfolder, delete a file, and sending a list of all the directories, sending a ping to the client to test the connection between the two, and handling multithreading to allow more than one client to connect. The analysis\_component.py file contains all of the information and functions necessary for a client or server to collect information relating to upload and downloading files between the two of them. The other software component is the many python libraries that the three files use. These libraries and their versions are listed in the requirements.txt file. Upon downloading the project, the user must install the libraries on their machine using said file.

There are three security components to the project. The first is a passcode that a client must type in upon connecting to the server. The passcode is “Rosebud26”. This prevents parties that are not supposed to be connecting to the server from connecting to the server. The other two components are the two encryption algorithms that the system uses to encrypt messages between the server and clients. These algorithms are RSA Encryption and AES Encryption. Messages are encrypted via RSA and file bytes are encrypted via AES. The only things that these algorithms rely on are the files that store their various pre-generated encryption and decryption keys. RSA’s encryption key is stored in public\_key.pem and its decryption key is stored in private\_key.pem. AES’ encryption key is stored in cipher\_key.pem and its decryption key is stored in nonce.pem. The reason that they come pre-generated with the system is to ensure that the server and all of its clients have the same encryption and decryption keys. If they do not, at best the system will have a loss of data integrity as messages are decrypted incorrectly and at worst the two will be completely unable to communicate. These two components exist so that if nefarious parties intercept data between the server and client, they won’t be able to understand it due to it being encrypted.

The project has two hardware components. The first is a computer for the server to run on. This computer can be turned off at any time, as the clients simply have to type in the server computer IP address and Port number when the Client.py file prompts them to, and they can connect. However, the users must have their own way of communicating this to clients. Which leads to the next hardware component, the client’s computer. All clients must have their own computers, all of which must be connected to the same network as the server, otherwise they will be unable to connect. Both the client and server computers must be Windows 11 computers, as the software components were designed for that kind of operating system.

## Implementation Details

The Implementation of this project consisted of three python files, server.py, Client.py, and analysis\_component.py. Each one consisted of the implementation of the information and functions that the server and its clients would need in order to interact.

The Client.py file contained the implementation for the client to connect to and interact with the server. First, the file imported the os, socket, and rsa libraries. It also imported AES from Cryptodome and everything from the analysis\_component file. It then defined two global variables, SIZE and FORMAT, which defined the size of a message sent or received over a socket, and the format in which that message would and should be encoded. It then went into encryption and decryption key loading. First, it created a global array called key, which contained one index equal to 1, meaning the program would assume that the keys would be successfully loaded. It then defined the function load\_keys, which loaded the encryption and decryption keys for RSA and AES encryption. Each key was stored in a file. If the files existed, the keys would be loaded and returned, else it would print an error message and set keys[0] equal to 1. The program then created four global variables, public\_key, private\_key, cipher\_key, and nonce, which would be the RSA and AES encryption and decryption keys. It then defined these variables using the return values of load\_keys().

The next function was file\_download(), which handled how clients downloaded files from the server and sent those files to the user’s Downloads folder. First, it defined the function to find the path to the Downloads folder. Then, it would receive the server’s message and send an acknowledgement. If the message said that the file is not found or no file name was sent, the function returned. If it wasn’t any of those, it means that the message was the file size, and would be converted into an integer. This was followed by a TQDM progress bar being created using the makeTQDM() function. It then would get the file name and make sure it doesn’t already exist in the Downloads folder. If it does, it would add a number at the end of the name, which it’d increase until the file name does not exist. It then would receive the file in chunks, since it can’t handle receiving the whole file all at once. Upon receiving each chunk, it’d update the progress bar. Once all chunks had been received, it’d decrypt the file data using an AES decryptor that would use the nonce variable, and write all the data to the file. It then would delete the decryptor and progress bar and print a message saying the download was a success and that the file can be found in the Downloads folder.

The next function was file\_upload(), which handled the client uploading files to the server. First, it made sure that the client specified a name for the file, else it would return 1. Then, it’d handle if the client was trying to override a file in the server’s server files by receiving a message form the server that is either a 1 or a 0, followed by the “@” symbol and some text, which it’d separate by splitting the message where the “@” was and store each in an array called “override”. If override[0] was 1, it meant that the client was trying to override a file. The program would then tell the client and ask if they wanted to override the file. It would then input the client’s response and send it to the server. If the client typed an invalid response, it would tell them and return 1. If override[0] was 0, it meant the client wasn’t trying to override a file. If it was something else, an error had occurred, which the program would print and then return 0. If it made it this far, the program would then check if the file existed. If it did, it’d get its size, send it to the server and wait for an acknowledgement before continuing. When it got it, it would then encrypt all of the file data using an AES encryptor, send it to the server, and delete the encryptor.

The final part of Client.py was the main function, which was executed only if the keys were successfully loaded(keys[0] equals 1). First, it asked for the IP address and Port number of the server the client wishes to connect to. It then used that information to create a tuple called ADDR and used that to create a socket. It then set a variable called waiting equal to 1, meaning the program would wait for the server’s response before doing anything. It then entered a while loop wherein if waiting was 1, it would print the server’s message. If the server’s response was “Access Denied. Passcode was incorrect.”, it means the client didn’t input the correct passcode upon initial connection, which would cause the while loop to break. If the while loop didn’t break, it’d take the client’s input, and send it to the server. If the input’s first word was “upload” it set waiting equal to the return value of the file\_upload() function. If it was “download”, it’d execute the file\_download() function, and set waiting equal to 0, since the server’s message would just be file data. If the first word was “logout”, it’d break the while loop. The while loop was repeated until any of the breaking conditions were met. It then printed the final disconnect message and closed the socket.

The server.py file contains the implementation for the server to handle and interact with clients. First, it imported the os, socket, threading, and rsa libraries. It also imported AES from Cryptodome and everything from the analysis\_component file. It then defined global variables IP as the IP address of the host, PORT as 4450, ADDR as a tuple of IP and PORT, SIZE, FORMAT, BASE\_DIR, which would be the directory server files would be stored at, with the value “server\_files”, and PASSCODE, which was the passcode that clients would need to access the server, as “Rosebud26”. It then defined the function load\_or\_generate\_keys(), which did the same thing as load\_keys() on the client side, except if the files didn’t exist, it created the keys using random characters and stored them in the files according to their names. The program then created four global variables, public\_key, private\_key, cipher\_key, and nonce, which would be the RSA and AES encryption and decryption keys. It then defined these variables using the return values of load\_or\_generate\_keys().

The next function that the program defined was client\_handling(), which defined how the server would handle a client. First, it asked the client for the passcode. If the passcode was incorrect, it’d send them the Access Denied message, disconnect them, print that they were disconnected, close the connection, and return. If they got the passcode right, it’d send them the welcome message and enter a while loop. The while loop starts by receiving the client’s message. It then analyzed the first word. If it was “upload”, it’d call the upload\_file() function. If it was “download”, it’d call the download\_file() function. If it was “delete”, it’d call the file\_delete() function. If it was “dir”, it’d call the directory\_list() function. If it was “subfolder” it’d call the subfolder\_manager() function. In each calling of these functions, it passed the whole message through so that the function could use that for whatever information it may need from it. If the word was “logout”, the while loop would break. If it was “ping”, it’d send a ping back to the client. If it was anything else, it’d tell the client that the command was invalid. If any error was encountered, it would send that error to the client. If the client disconnected, it would print that they disconnected and close the socket.

The next function was file\_upload(), which handled how the server received files the client uploaded. If no name was specified, it’d tell the client and return. It then got the path to the file. If the file already existed, it’d send the client a 1, along with a message asking if they want to override it. If the client said “yes”, it’d tell the client it’s overriding the file. If they said “no”, it would add a number to the end of the file name until it wasn’t a duplicate. If it was anything else, it’d tell the client that they sent an invalid response and return. If no files were being overridden, it’d send the client a 0, along with a message. It then functions like the file\_download() in Client.py, getting the file size, sending an acknowledgement, and receiving file data in chunks, all while updating a TQDM progress bar, and using an AES decryptor to decrypt the file data.

The next function was the file\_download() function, which handled when the client wished to download a file from the server. First, it checked that the client said a file name. If they didn’t, it’d tell the client and return. Next, it got the file path and size. If the file existed, it’d send the size and the file data, which was encrypted by an AES encryptor. If the file didn’t exist, it’d tell the client. Followed by this was the file\_delete() function, which first checked that the client specified a file name. It then got the file path and if it existed, deleted the file and told the client. If it didn’t exist, it told the client that it doesn’t exist and returned. The directory\_list() function defines sending the directory list of all files in the BASE\_DIR to the client. If there are none, it sends “No Files Found”, else it sends the list. The next function is the subfolder\_manager() function, which handles if the client wants to delete or create a subfolder. First it checked that the file specified an action and path. If they didn’t, it told them and returned. It then got the client’s action and path. If the action was “create”, it made the directory in the path the client specified and told them. If the action was “delete”, it checked that the path existed. If it did, it deleted the folder and told the client. If it didn't, it told the client and returned. If the client said anything else as an action, it told them that that action was an invalid action.

The final function in server.py was the main() function, which started by creating a socket and binding it to ADDR. That socket then listened. Once a client connected, it printed their IP address and the fact that they’ve connected, created a thread for them targeting client\_handling(), and printed the number of active threads.

Between both files, all messages that did not contain file data were encrypted with RSA and encoded using utf-8. The maximum size that a message could receive was 1,024 bytes.

The final file was the analysis\_component.py file. This file started by importing the tqdm library and then defining a function called makeTQDM, which took in the file\_size and used that to make a TQDM progress bar, which it then returned.

## Experimental Results and Analysis

The following text is a detailed documentation of the experimental results of running the code for this project.

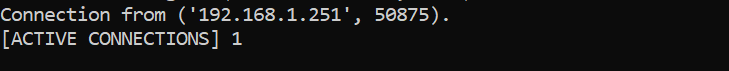
The server was successfully created and displayed its information on the computer hosting it.



Clients were then able to enter the server’s IP Address and Port Number.

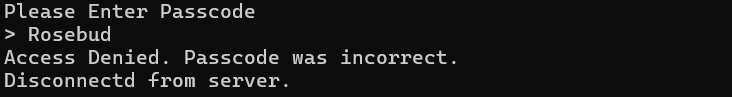


The server was able to receive those clients, allowing the two computers to connect.



Upon connection, the server then prompted the client to enter the passcode needed to access the server. When the client entered the wrong passcode, they were denied access to the server and were disconnected.

Client Side:



Server Side:



When the client reconnected and entered the correct passcode, they were allowed access to the server and sent a welcome message.



When the client began uploading a file to the server, their computer displayed the message that the file was being uploaded.

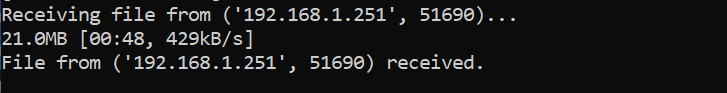


The server began receiving the file data, displaying its progress

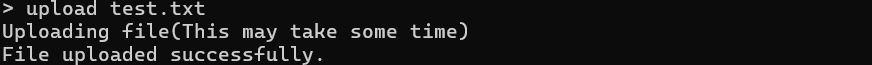


Once the server finished receiving the file, the client was notified.

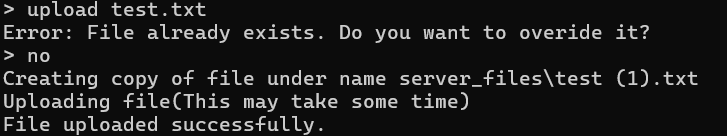
Server Side:



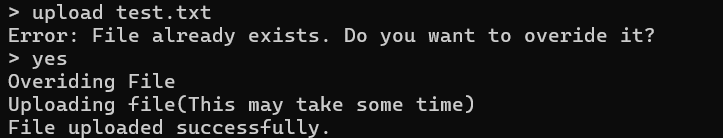
Client Side:



When the client uploaded a file that the server already had, the server would ask them if they wanted to override the file. When the client said no, the server created a copy of that file.



When the client did it again, they said yes, and the server overrided that file.



When the Client tried to upload a file that did not exist on their computer, the program told them that the file did not exist.



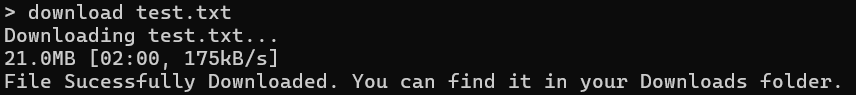
When the client failed to specify the name of the file they were uploading, the server told them that a file name was required.



When the client tried to download a file from the server, the server sent them the file data. A progress bar was displayed on the client’s computer.



Upon completion, the program then stored the file in the client’s download folder.



When the client tried to download a file that the server did not have, the server told the client.



When the client failed to specify the name of the file they were downloading, the server would tell them that a file name is required.



When the client deleted a file that was stored on the server, the server did so and told the client.



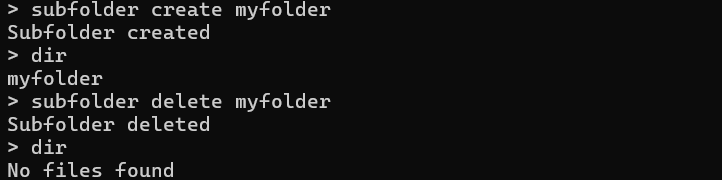
When the client tried to delete a file that the server did not have stored, the server told them.



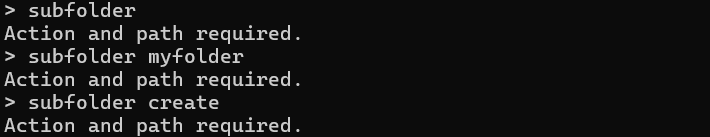
When the client failed to specify the name of the file they wished to delete, the server told them.



When the client created a subfolder within the server’s directory, the server did so and told the client. The client was able to see their subfolder in the server’s directory. The client then deleted that subfolder and saw that it no longer existed within the server’s directory.



When the client did not enter a full subfolder command, the server told them.



When the client sent a ping to the server to test the connection between the two of them, the server sent one back.



When the client entered an invalid command, the server told them.



The client then logged off of the server.



As they did so, the server printed the message that a client disconnected.



The server was also able to handle many connections at the same time using multithreading.



## Problems Faced

One of the problems that was encountered during this project was dealing with encrypting file bytes when we were uploading/downloading files. Originally, RSA Encryption was used to encrypt bytes, but this did not work. For reasons that are still unknown, the program had issues decrypting the bytes whenever sending or receiving files. However, the solution to this problem is not unknown. Instead of using RSA to encrypt file bytes, AES was used to encrypt the file bytes.

Another problem that was faced was with uploading files. When the file size and file data was sent from the client, the server would receive the file size, and a part of the file data in one execution of recv(). Fixing this required lots of trial and error with many different solutions, but the one that showed the most promise was having a millisecond timeout between the client sending the file size and file data. But a millisecond is a lot of time, especially during instances when Round Trip Times would be lower than one millisecond. So another solution was sought out and found. Instead of waiting a full millisecond, the client will wait for an acknowledgement after sending the size before beginning to send the data. This was a better solution because it allowed for fast Round Trip Times to be maintained throughout the program.

## 

## What was learned

What I learned throughout the making of this project is how computer networking works and what goes into making it happen. From understanding how to develop the server-side, I learned about how to handle client connections and how to give clients various performable actions along these connections. This includes being able to upload and download various files, and organizing these files as they please through the use of subfolder creations and the ability to delete/override files, all while doing it from a remote device using a program we created.

*– Ryan Moorgen*

Over the course of this project I had to relearn a lot of basic python knowledge I had forgotten about since I finished CS Problem Solving And Solution. Before this project, I only knew how to define functions and do basic logic in python, now I have a basic understanding of socket programming. I also learned about basic networking concepts, like when a client asks a server for a file to download, what is actually happening on a coding level.

*– Payne Persons*

I learned a variety of things from this project. First off, I learned basic socket programming, like sending messages back and forth over a socket. I then took that socket and used that to connect two computers to each other using their IP addresses. After that, I learned how to make those messages more secure using RSA Encryption. However, if you want to send files over those messages, you have to use AES to send each byte. Unfortunately, I did not have time to fully research why that is what it is. I also learned that sometimes, the recv() function will receive more than one message if it has the room, which can lead to data errors, and how to solve that issue by having the sender wait for an acknowledgement after sending the file size but before sending the file data. Another thing I learned was how to use TQDM to keep track of file upload/download progress. From looking at the other group members’ code, I learned how to send files over sockets and became very familiar with the OS library in python. And lastly, I finally memorized how to run and install files using the command prompt from doing it over and over again during this project.

*– Matthew Rigg*

## Contribution Table

| Names: | Contribution: | Participation (%) |
| --- | --- | --- |
| Ryan Moorgen | –Developed a python module used to run the server.  –Developed a python module used for performance analysis.  –Recorded code commentary for the video  –Recorded the code runtime. | 34% |
| Matthew Rigg | –Enabled Connection between server and client.  –Programmed RSA and AES encryption.  –Wrote Architecture, Implementation, Experimental Results, and Problems Section of Report.  –Recorded code commentary for video. | 33% |
| Payne Persons | –Developed Client Functions.  –General Debugging.  –Recorded code commentary for the video. | 33% |

## 

## Conclusions and Future Work

Computer Networking is the process of connecting devices to share data and resources. The purpose of this project was to create a small scale computer network that allowed computers to connect and transfer files to one another over their local network. The project allowed for many file transfer functions, including uploading, downloading, and deleting files. The project also took into account many different security measures, including passcode protection and data encryption. Taking all of these things into account led to the development and implementation of many software, security, and hardware components.

When the code for this project was run, it met expectations for what it was designed and implemented to do, and very few problems were encountered that were not unable to be solved. Every function that the client and server needed to be able to execute were successfully carried out. The only major problems that were encountered during experimental testing were fixing the issue with RSA file byte encryption and the issue with the recv() function taking in multiple messages at various points within the program. Luckily, both of the problems had solutions that were easily able to be designed and implemented into the program without any major compatibility issues.

In terms of future work to be done, there are some things that are worth exploring and adding to the project. One, the use of a graphical user interface to make our project more intuitive to use and understand for non-technical users. Two, it is regrettable that there was not enough time to research why RSA does not work for encrypting file bytes. Perhaps a non-AES solution can be found, meaning the program would only need to store two keys instead of four, allowing for the program to take up less space on a user’s computer. And three, the quest for ways to improve file upload and download times, especially for large files, must continue, in the hopes of finding ways for these functions to take less time. But until then, we hope that this project is able to fulfill all potential users’ needs for adequate transferring of files.