

CS 433 Assignment 1

Shrreya Singh 19110136

Design Document

[GithubLink](#)¹

Layering Mode and Protocol Design

The Remote File System network is based on the standard OSI model with simple implementation of the top 3 layers: File Service (Application layer), Crypto (Presentation Layer) and Networking (Transport layer). Overall the code consists of six python files that perform the functions associated with their corresponding layer.

Client Stack Protocol Design

1. `c_app.py`: This is the file service layer (Nth layer) on the client side. It performs the following functions:
 - Receiving and processing user defined inputs [command to execute and choice of encryption]
 - In case, any bad request is made by the user [for example, commands other than `cwd`, `ls`, `cd`, `dwd` and `upd`] or uploading files not present in the path, etc. this layer reports the same back to the user.
 - In case, the request is valid [i.e. wrt to the client service layer], it encodes the data into byte format and sends it to the encryption layer.
 - The headers appended to the data are `command_code` [`cmd_code`, separated by token `@@@`] and `encrypt_code`², separated by separation token `^@^`. Content consists of the

¹ GitHub Link: https://github.com/ShrreyaSingh/CS433_A1_RFS.git

² Each command and encryption mode have been associated with unique codes mentioned below:
`cmd_code`: 1→ `cwd`, 2→ `ls`, 3→ `cd <dir>`, 4→ `dwd file_path`, 5→ `upd file_path`
`encrypt_code`: 1→ `plain`, 2→ `substitute`, 3→ `transpose`

- command and any other relevant file data like (uploading data in case of upd).
- Receives the decrypted response from the server and generates an appropriate message and status for the user.
2. `c_crypto.py`: It provides encryption services to the client host (N-1th layer) for transmitting data over the network and supports three modes for the same:
 - Plain text → No change in the input i.e. no encryption/decryption
 - Substitute → Substituting alphanumeric characters with a fixed offset, for eg. Caesar cipher with offset 2. Here ABCD1, will be substituted with CDEF2.
 - Transpose → Reverse the contents in a word by word manner. For example, ABCD will be changed to DCBA.
 3. `c_trans.py`: Transport (N-2th layer) layer of client side. It takes the encrypted data, establishes a connection with the server and transmits its protocol data unit over the network. It creates a socket object for the client on port 2500, AF_NET (IPv4) and SOCK_STREAM (TCP, protocol used for establishing a connection) using python socket library. It sends back the server's response for decryption which is then sent to the client's service layer.

Server Stack Protocol Design

4. `s_trans.py`: It implements the socket for the server side (N-2th layer) that binds to the specified host and port and listens to connections from clients and accepts the request when the client connects on that host and port. It creates a socket object for the server with AF_NET (IPv4) and SOCK_STREAM (TCP, protocol used for connection) using python socket library. The server is always ON in this model and continuously listens for incoming requests. It receives the data in encoded and encrypted format, and transmits it to the crypto layer (`s_crypto.py` → similar to `c_crypto.py`) for decryption and sends back the server's response to the client.


- 5. `s_app.py`: Service layer (Nth layer) on the server side that processes the request and generates an appropriate response to it. It sends the encoded response with content as actual data or message requested and content header as status [separated by token `@@@`] to the crypto layer for encryption which sends to the transport layer for transmittance.

Code Flow

Client's service layer takes input from the users regarding choice of command and encryption mode, processes it and sends it to the crypto layer for encryption after attaching the necessary headers. The transport (networking N-2th layer) layer creates a client socket, receives the encrypted SDU and sends it to the server once the server accepts the connection request. The server's socket was created by the server transport layer that listens on the specified port until a client makes a request on that port. The encrypted data, on the server side is decrypted by the server's crypto layer and sent to its file service layer for generating an appropriate response. This response is again encrypted with content header status and sent to the transport & networking layer of the server to send it to the client. Client's N-2th layer receives the same, decrypts it (N-1th layer) and sends it to its file service layer for displaying the appropriate message, data and status to the user. The connection ends after the client's request is served, however, the server is continuously ON and listens to incoming requests. For new requests, the connection has to be set up again. All the vertical data transfers between the layers on the same side [i.e. on client or server side] is done in byte encoded format.

Challenges:

The main challenge I faced was dividing the code flow into different layers. Since, we are more or less implementing the 3 OSI layers - Application, Presentation and Transport layer, I needed to decide the exact division of tasks



taking place in these layers. For the application layer, I had to decide the depth to which the data flows and how it should be encrypted. It would be absurd to send encrypted data in all the layers in random order. Instead, the data should follow a proper path like appending the data with proper headers, encrypting it and then sending it to the network. Initially I created separate files for encryption and decryption but given the nature of the tasks, only 1 function in a single file (one each of server and client) is enough since, twice the transpose and shifting encrypted data by negative of the encryption shift gives the actual data [i.e. decryption]. I also had to think about various cases of bad requests and handle them in the appropriate segments of the code like giving some unsupported command, say `echo $PATH`. Thus maintaining the pipeline of the code while maintaining the layered architecture and error handling the different scenarios of the user inputs were the two main challenges faced by me. Further, the current connection is non-persistent and to serve multiple requests in one connection we can incorporate multithreading in our network.

Screenshots:

1) cwd:

Server Side:

```
[(base) shrreyasingh@Shrreyas-MacBook-Pro cs433_A1_rfs % python3 s_trans.py  
Connected by ('127.0.0.1', 52130)
```

Client Side:

```
(base) shrreyasingh@Shrreyas-MacBook-Pro cs433_A1_rfs % python3 c_app.py  
  
Enter any one of the below commands:  
[CWD  
LS  
CD dir_path  
DWD server_file_path  
UPD client_file_path  
  
-----  
cwd  
  
Enter an integer to select the encryption type:  
1 Plain  
2 Substitute  
3 Transpose  
  
-----  
2  
  
Setting up connection ...  
  
Connection closed :)  
-----  
Received: '/Users/shrreyasingh/Desktop/My_files/Courses/CN/CS433_A1_RFS'
```

2) ls

Server Side:

```
(base) shrreyasingh@Shrreyas-MacBook-Pro cs433_A1_rfs % python3 s_trans.py  
Connected by ('127.0.0.1', 52131)
```

Client Side:

```
(base) shrreyasingh@Shrreyas-MacBook-Pro cs433_A1_rfs % python3 c_app.py  
Enter any one of the below commands:  
CWD  
LS  
CD dir_path  
DWD server_file_path  
UPD client_file_path  
-----  
cwd  
Enter an integer to select the encryption type:  
1 Plain  
2 Substitute  
3 Transpose  
-----  
2  
Setting up connection ...  
Connection closed :)  
-----  
Received: '/Users/shrreyasingh/Desktop/My_files/Courses/CN/CS433_A1_RFS'  
(base) shrreyasingh@Shrreyas-MacBook-Pro cs433_A1_rfs % python3 c_app.py  
Enter any one of the below commands:  
CWD  
LS  
CD dir_path  
DWD server_file_path  
UPD client_file_path  
-----  
ls  
Enter an integer to select the encryption type:  
1 Plain  
2 Substitute  
3 Transpose  
-----  
3  
Setting up connection ...  
Connection closed :)  
-----  
Received: *['c_crypto.py', 's_trans.py', '.DS_Store', 'client_files', 'images', '__pycache__', 'README.md', 'c_app.py', '.gitignore', 'server_files', '.git', 'c_trans.py', 's_app.py', 's_crypto.py']*
```

3) cd <dir_path>

Server Side:

```
[(base) shrreyasingh@Shrreyas-MacBook-Pro cs433_A1_rfs % python3 s_trans.py  
Connected by ('127.0.0.1', 52132)
```

Client Side:

```
[(base) shrreyasingh@Shrreyas-MacBook-Pro cs433_A1_rfs % python3 c_app.py  
  
Enter any one of the below commands:  
CWD  
LS  
CD dir_path  
DWD server_file_path  
UPD client_file_path  
  
-----  
cd server_files  
  
Enter an integer to select the encryption type:  
1 Plain  
2 Substitute  
3 Transpose  
  
-----  
1  
  
Setting up connection ...  
  
Connection closed :)  
-----  
Received: 'Directory changed successfully'  
-----  
STATUS: OK
```

4) dwd <file_path>

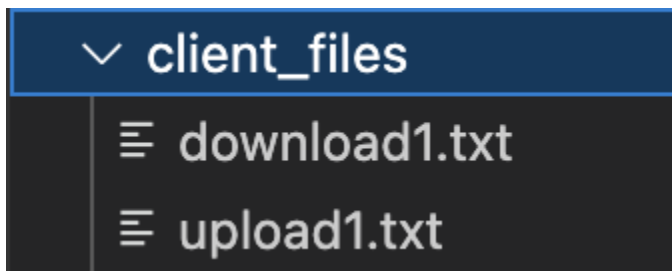
Server Side:

```
[(base) shrreyasingh@Shrreyas-MacBook-Pro cs433_A1_rfs % python3 s_trans.py  
Connected by ('127.0.0.1', 52133)]
```

Client Side:

```
[(base) shrreyasingh@Shrreyas-MacBook-Pro cs433_A1_rfs % python3 c_app.py  
Enter any one of the below commands:  
CWD  
LS  
CD dir_path  
DWD server_file_path  
UPD client_file_path  
-----  
dwd server_files/download1.txt  
Enter an integer to select the encryption type:  
1 Plain  
2 Substitute  
3 Transpose  
-----  
2  
Setting up connection ...  
Connection closed :)  
-----  
Received: 'I am a file downloaded from the server \nand I have multiple lines\nwith some emojis :)\nChecking Encryption'  
-----  
STATUS: OK
```

File (with all its content) is successfully downloaded in the client_files directory from the server:



5) upd <file_path>

Server Side:

```
[(base) shrreyasingh@Shrreyas-MacBook-Pro cs433_A1_rfs % python3 s_trans.py  
Connected by ('127.0.0.1', 52135)
```

Client Side:

```
[(base) shrreyasingh@Shrreyas-MacBook-Pro cs433_A1_rfs % python3 c_app.py  
  
Enter any one of the below commands:  
CWD  
LS  
CD dir_path  
DWD server_file_path  
UPD client_file_path  
  
-----  
upd client_files/upload1.txt  
  
Enter an integer to select the encryption type:  
1 Plain  
2 Substitute  
3 Transpose  
  
-----  
3  
  
Setting up connection ...  
  
Connection closed :)  
  
-----  
Received: 'server_files/upload1.txt'  
-----  
STATUS: OK
```

The file (with all its content) is successfully uploaded to server_files directory from the client

