ARTIFICIAL INTELLIGENCE AND DATA ENGINEERING
project report for the examination
of
Foundations of Cybersecurity

CLOUD STORAGE

Student:
Alessandro Diana

AA. 2021/2022

Contents

List of Figures

1 Application

1.1 Description

The objective of this project is the realization of a client-server application that resembles the behavior of a Cloud Storage.
Each user has a "dedicated storage" in the server and any user will not be able to access the "dedicated storage" of any other user, but only his own.
Users must be able to perform the main operations (described in the next sections) for managing files within the cloud in a secure manner. The application must be protected both from malicious actions performed (voluntarily or involuntarily) by users and from attempted attacks by external adversaries.

1.2 Project Guidelines and General Requirement

The application must have these requirements:

- Users are pre-registered on the server;
- When the client application starts, the Server and Client must authenticate.
  – Server must authenticate with the public key certified by the certification authority.
  – Client must authenticate with the public key, pre-shared with the server.
- During authentication a symmetric session key must be negotiated.
  – The negotiation must provide Perfect Forward Secrecy.
  – The entire session must be encrypted and authenticated.
  – The entire session must be protected against replay attacks.
- Each user has a "dedicated storage" on the server and one user cannot access to "dedicated storage" of another user.

1.3 Functional requirements

1.3.1 Clients

Once the user is connected to the service, i.e. after the client has successfully established the connection with the server, the user must be able to do the following operations:

- Upload: the user can upload a file, maximum size 4 GB, from the client to the server. The user must specify the filename, on the client machine, of the file he wants to send and the filename under which the file is to be saved on the server. If it is not possible to save it with the specified filename, the operation fails and the file is not sent;
- Download: the user can download a file, from the server to the client. The user specifies a file on the server machine, through the filename under which it is saved on the server, and then the server sends the requested file to the user. The filename of any downloaded file must be the filename used to store the file on the server. If this is not possible, the file is not downloaded;
- Delete: the user can delete a file from the server. The user specifies a file on the server machine then the server asks the user for confirmation. If the user confirms, the file is deleted from the server.

- List: the user can see a list of the filenames of the available files in his dedicated storage. The user asks the server the list and then the client prints to screen the list.
- Rename: the user can rename a file from the server. The user specifies a file on the server machine, through the filename under which it is saved on the server. Within the request, the client sends the new filename. If the renaming operation is not possible, the filename is not changed;
- LogOut: the user can log out from the server. The client has to gracefully close the connection with the server.

1.3.2 Server

The server must:

- allows multiple users to one server: the server shall be able to accept a connection by a client;
- be able to respond to client requests;
- resembles a Cloud Storage.

1.4 System architecture

The system is organized with a client-server architecture, as shown in figure 2.
The project consists of 3 programs: common.h (which contains variables and functions useful for both the server and the client), client.cpp (the code relating to the client) and server.cpp (the code relating to the server). The server files are located inside the 'ServerFiles' folder while the client files are located inside the 'ClientFiles' folder.

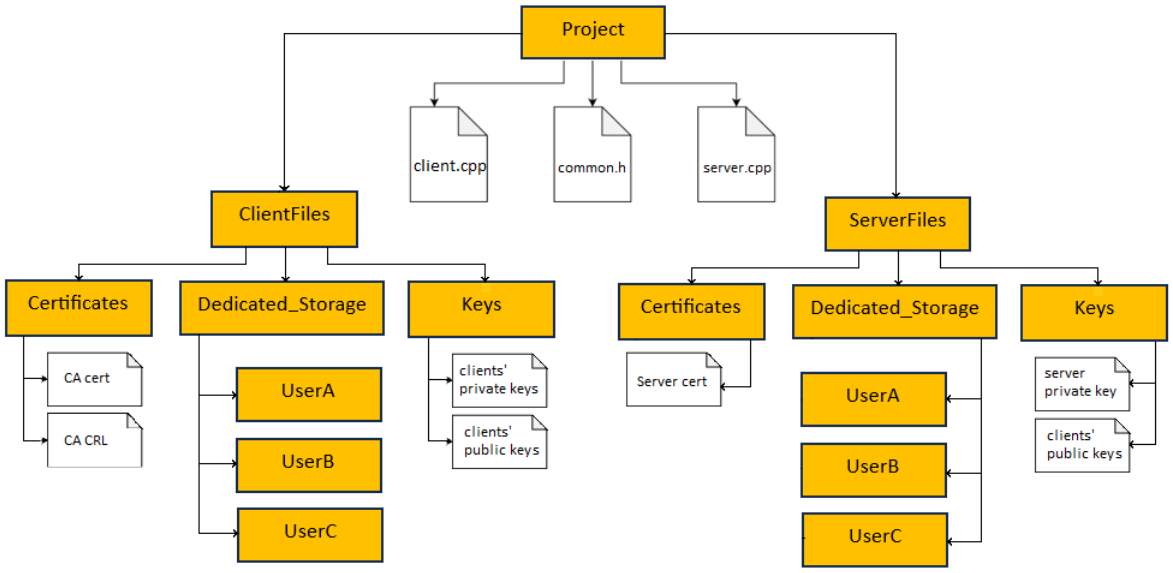

Figure 1: Rappresentation of system architecture.

2 Implementation choice

2.1 System assumption and semplification

- users have already the CA certificate;
- each user have already a long-term RSA key-pair;
- server has its certificate signed by the CA;
- server knows the username of every registered user;
- server knows the RSA public key of every user, which is stored in the 'Server Files' folder;
- "Dedicated_Storage" already allocated. Dedicated storage space is represented by folders in '/ServerFiles/Dedicated_Storage'. Each registered user has a folder that will represent their dedicated storage, it will be named after the username of the owner user;
- the cmd_code (operation identification code) in exchanged messages is always authenticated to guarantee the real intention of the user;
- The nonce exchanged during messages is also authenticated to ensure its truthfulness and guarantee the source of the message;
- Users' public keys being public are not protected while private keys are protected by passwords to guarantee their secrecy and exclusive use by the owner;
- The user keys are all stored in the same 'ClientFiles/keys' folder;
- Messages exchanged between client and server are always authenticated and encrypted;
- The client must be able to simulate different users with different spaces of their own. This is why I decided that there will be a folder for each user representing their own space. Thus each user can have their files managed in a separate folder from the other users. For this reason, when a user enters a path, it will be considered as starting from his folder;
- Permitted file names must only contain terms from the white list. File names containing spaces within them are not permitted;
- File names have a maximum permitted length (100 characters by default).

2.2 Authentication

To start the application, the server will start as first and will wait to be contacted by the clients that want to use the cloud storage. When a client contacts the server they will have to authenticate each other to establish an authenticated and encrypted connection. The server and each client have a 2048-bit RSA public and private key pair.

- The client authenticates by signing with its private key and the server will verify with the corresponding public key it already has.
- The server authenticates using a certificate released by a trusted certification authority. The client can obtain the server's public key from the certificate to verify the signature;
- The client and server after authenticating themselves will establish a (symmetrical) key for the session using EC Diffie Hellman authenticated through the server and client signatures;
- Every message contains nonces to avoid replay attacks.

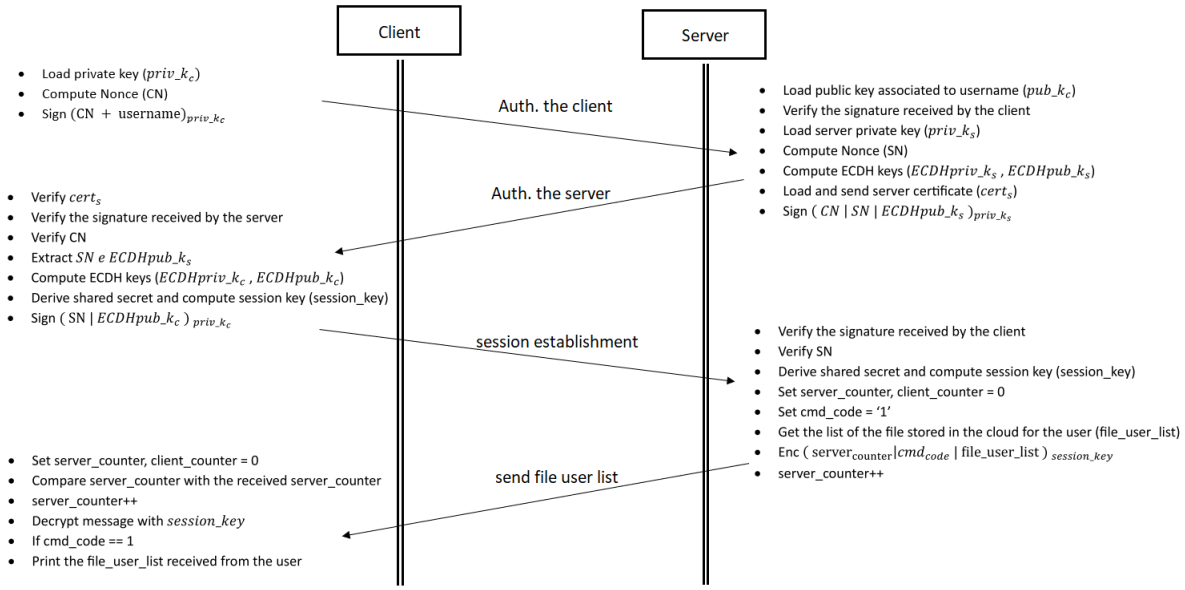

Figure 2: Illustrative figure of the session's establishment protocol.

2.3 Authenticated encryption

The messages of each session between a client and the server are encrypted, authenticated, and protected from replay attacks (through the use of a nonce). AES-GSM-128 is used to provide authenticated encryption. The symmetric key used in the communication is a session key calculated by the two parties of the communication (client and server). After encryption in general, each message has the following structure(shown in figure 3): The initialization vector

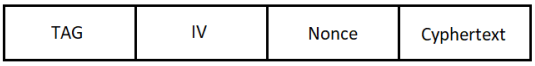

Figure 3: Illustrative figure of the general message structure with AES-GSM-128.

(IV) has a length of 12 bytes and is randomly generated before each encryption. The tag has a size of 16 bytes. The nonce (both user and server) are implemented as counters that are initialized to zero at the start of the session and are incremented by 1 with each encrypted message. The client and server each have their nonce. Both the IV and the nonce are authenticated only. The nonce approves the freshness of the message. A message is fresh if the received nonce is equal to the nonce stored in the receiver.

2.4 Messages of the operations

Each operation has a different protocol with a different number of messages exchanged and a different structure. Operations are identified by a code (cmd_code). Each operation is characterized by a code and each message related to that operation must have that code, except for error messages which has the code related to the error (a generic code for all errors). With each message received, the client or server checks that the nonce is correct and that the operation

code matches the expected or error code. Figure 4 shows a table 4 with a description of the commands and their code.

Operation	Command	Cmd_code	Description
Error	<no command>	-1	Indicates that an error has occurred.
Logout	!logout	0	The user closes the connection and exits the program.
List	!list	1	The user requests the list of files he has saved on the cloud server.
Upload	!upload	2	The user sends a file to the cloud server, which will save it in the user's dedicated storage area.
Download	!download	3	The user requests and receives a file saved in the cloud server.
Rename	!rename	4	The user changes the name of a file saved on the cloud server.
Delate	!delate	5	The user deletes a file saved on the cloud server.
Help	!help	<no cmd_code>	The user asks for information on one or all of the commands he can execute.

Figure 4: Table showing the commands with their codes and explanations.

Below there are the protocols for each operation and the structure of the messages exchanged. In the figures shown, the part in green indicates that it is an encryption-protected part while the orange parts are authenticated parts. The general packet for the authenticated and encrypted communications is shown in figure 5.

Figure 5: Illustrative image of a general packet for the authenticated and encrypted communications.

In this protocol, an operation starts with a request from the client. The request will be a message in the format seen above with the cmd_code of the operation it wants to request from the server. When exchanging messages for an operation the cmd_code of the messages (both

response and request) must always have the same cmd_code (the one inherent to the requested operation). The only other cmd_code accepted during an operation is the one identifying an error message, in which case the error will be displayed and then the operation terminated.

2.4.1 Logout

When the user wants to close the connection, he sends the request to the server with his username (1). The server when it receives the request checks the username, sends a message to the client (2) notifying it of a successful closure or an error that has occurred, and then closes the connection. The client when receiving the server's reply will display the message and close the connection. The protocol for this operation is shown in the figure 6.

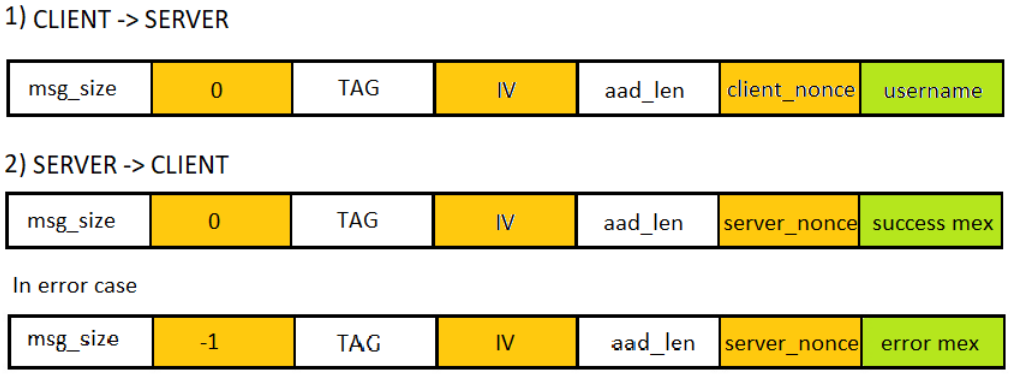

Figure 6: Illustrative image of the messages exchanged for the logout operation.

2.4.2 List

When the user wants to see the list of his files on the server, he sends the request(1). The server receives the request and checks that the username matches the client served and then creates the list of saved file names that will be sent to the client(2).
The protocol for this operation is shown in the figure 7.

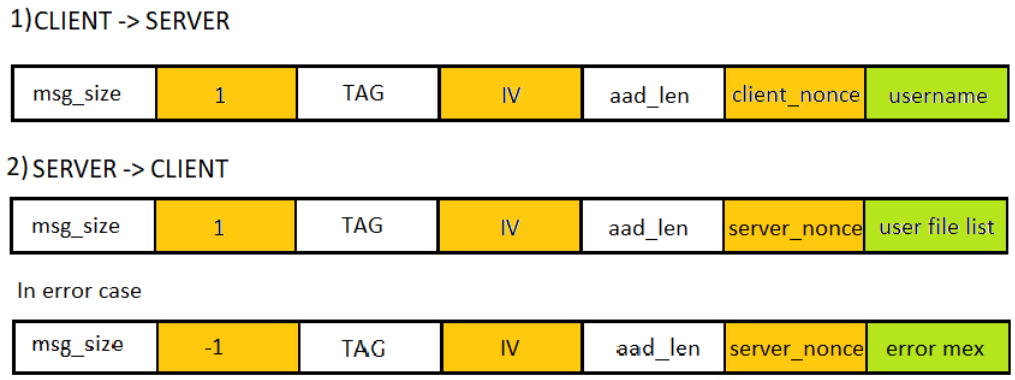

Figure 7: Illustrative image of the messages exchanged for the list operation.

2.4.3 Upload

The user enters the name of a file he wants to upload to the server. The client checks the validity of the file name, if it is present in the user's folder, and also checks that the file size is within the project specification (less than 4GBi). After these checks, the client sends a request to the server with the size and the name of the file it wants to upload (1). The server checks the validity of the file name and checks if already exists the file in the dedicated storage. If the file is present the server sends an error message to the client and the operation fails(2). If the file isn't present the server sends a message to the client(2). The client receives the response from the server and if all is ok sends the encrypted file (3). The server decrypts and saves the received files, then sends to the client a success message if everything has been done or an error message if an error has occurred (4). The client receives and displays the message received from the server to the user.
The protocol for this operation is shown in the figure 8.

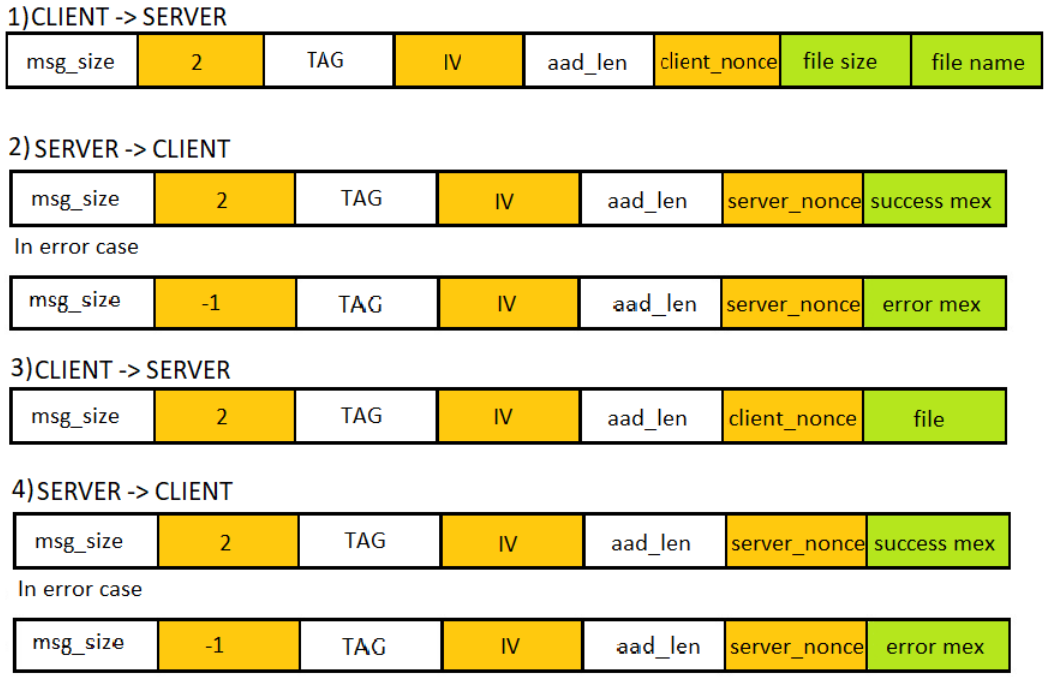

Figure 8: Illustrative image of the messages exchanged for the upload operation.

2.4.4 Download

The user enters the name of the file to be downloaded from the server. The name of the file entered is checked to ensure that the user does not already have a file with the same name in his folder. If the checks fail, the operation ends before sending the request to the server, otherwise the client sends the request with the requested file name (1). The server checks the name received and checks whether it has a file with that name present in the user's memory space. In the event of an error, the server sends an error message to the client(2). Otherwise, the server sends two messages to the client, the first containing the size of the requested file (2)

and the second containing the file(3). The client receives the response from the server, in case of an error the message is shown.
The protocol for this operation is shown in the figure 9.

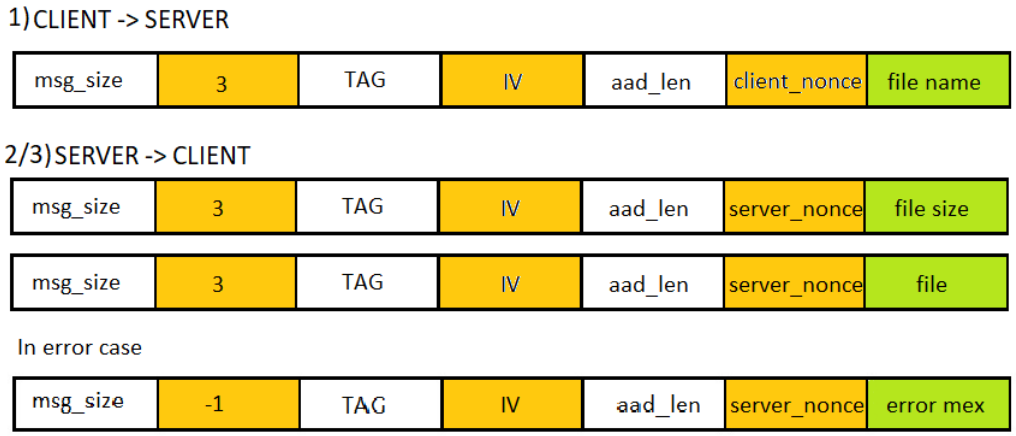

Figure 9: Illustrative image of the messages exchanged for the download operation.

2.4.5 Rename

The user enters the old name of the file to be changed and the desired new name. Both names are checked and then sent to the server (1). The server receives and checks that it has a file with the old name and that it has no files saved under the new name. In the event of an error, the server responds to the client with an error message (2). If no errors occur, the server renames the specified file and responds with a success message to the client (2). The client receives the message from the server and shows it to the user.
The protocol for this operation is shown in the figure 10.

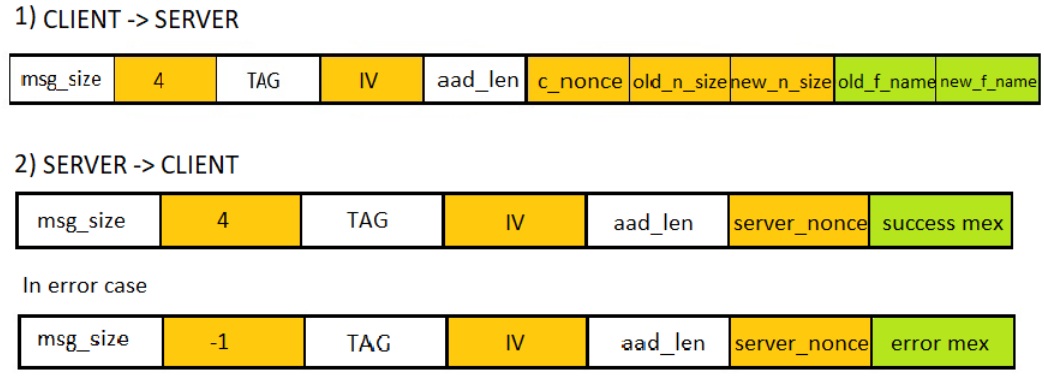

Figure 10: Illustrative image of the messages exchanged for the rename operation.

2.4.6 Delete

The user enters the name of the file to be deleted from the server. The client checks the name entered and sends it to the server (1). The server receives the name from the client and checks whether it has a file with that name or not. If the checks are passed and the server has a file saved with that name it sends the client a deletion confirmation for the given file (2). In the event of an error, it sends an error message and terminates the deletion operation (2). In case of an error, the client receives the error message to be shown to the user and terminates the delete operation. If all is ok, the client receives the confirmation request and shows it to the user who must choose whether to confirm the deletion (type y) or to block the operation (type n). The client takes the user's choice and sends it to the server (3). The server receives the user's choice, then finishes or terminates the delete operation and sends a message to the client notifying the success or failure of the operation (4).
The protocol for this operation is shown in the figure 11.

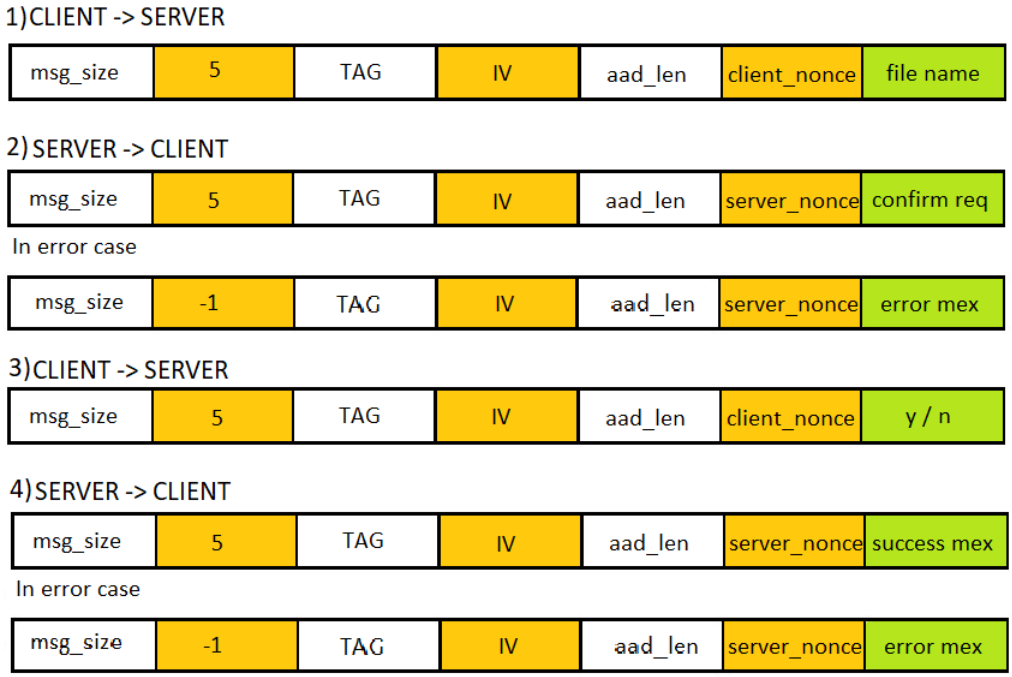

Figure 11: Illustrative image of the messages exchanged for the delete operation.

2.4.7 Help

This command is only present on the client, it will be used to support users by explaining every other command.

3 Implementation

The application is developed in c++ with the help of the OpenSSL library for the security part. The application code is divided into three files:

- Server: contains the code relating to the server;
- Client: contains the code relating to the client;
- Common: contains variables and functions useful for both the server and the client.

The complete application code is available on GitHub[1].

The following folders can be found in the project folder:

- ClientFiles: containing all files (keys, certificates, files) concerning the client;
- ServerFiles: containing all files (keys, certificates, saved user files) concerning the server;
- test files: a folder containing the files provided for testing the application. There are small .txt files and two larger .tar files. These files can be copied to the individual users' folders and then used to test the application.

The folder also contains a file called Password.txt. The passwords for the private keys of the server and the three pre-registered users are stored in this file. The passwords are required to initiate the authenticated and secure connection between the client and the server.

3.1 Application view

In this section, I will briefly explain the functioning and features of the application by showing a view of it.

The pictures show an example of a connection between a user and the server in which the user requests each available operation at least once (the list command will be called several times) and then disconnects. Each operation produces printouts on both the client and the server showing the requested actions, how and if the request was carried out, the results or any error messages.

Figure 12 shows the first phase of authentication between server and client and the establishment of an authenticated and encrypted connection. After the connection has been established, the client will show the user a legend of all recognized commands, while the server sends the list of saved files for that user, which is shown to the user. The user will then be asked to upload the file 'old.txt' to the server, after which he will be asked to change the name of the file 'old.txt' now present on the server to the new name 'new.txt'.

Figure 2 shows the following step. The user requests the list of files on the server to verify that the rename has happened, then requests the deletion of the 'new.txt' file, which he must confirm to complete the operation. He then downloads the remaining file to the cloud (the largest of the test files, about 79 MBi). The client shows to the user the message of successful download and also how many bytes were downloaded. The user then disconnects from the server and closes this session. The server will stand by for future connections from the usual or other users.

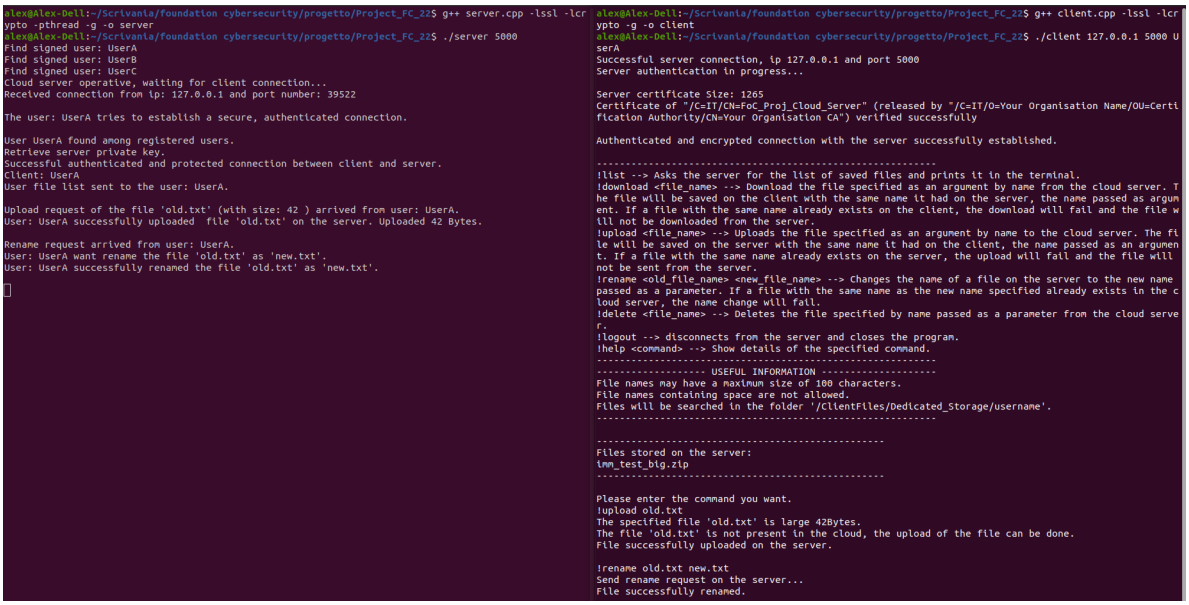

Figure 12: First view of the application.

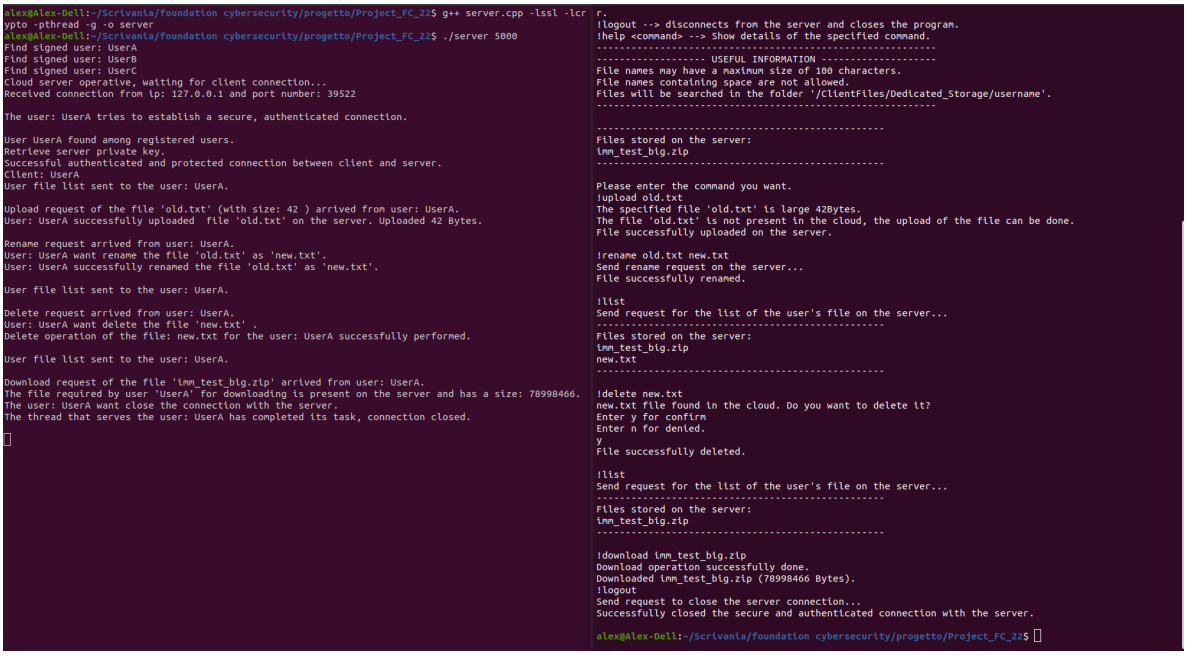

Figure 13: Second view of the application.

Bibliography

[1] *Project link on github*, `https://github.com/Arzazrel/Project_FC_22.git`, Accessed: 2023-01-15.