UNIVERSITÀ DI PISA

Cloud Storage

MSc Computer Engineering / Artificial Intelligence and Data Engineering

Foundations of Cybersecurity

Gaetano Sferrazza

Emanuele Tinghi

Veronica Torraca

A.Y. 2022-2023

Contents

1. Introduction

The aim of this project is to implement a Client-Server application that emulates a simple and secure Cloud Storage, offering confidentiality, integrity and authenticity through authenticated encryption, and reliability through TCP connection.

In particular:

- Each user has a **dedicated storage** on the server, with exclusive access
- An user can **Upload**, **Download**, **Rename**, **Delete** a file to/from the Cloud Storage or request the **List** of files in their cloud storage, in a safe way
- Users have the CA certificate and a long-term RSA key-pair which is password-protected
- Users are pre-registered on the server
- Server has its own certificate signed by the CA, and knows the RSA public key of each user

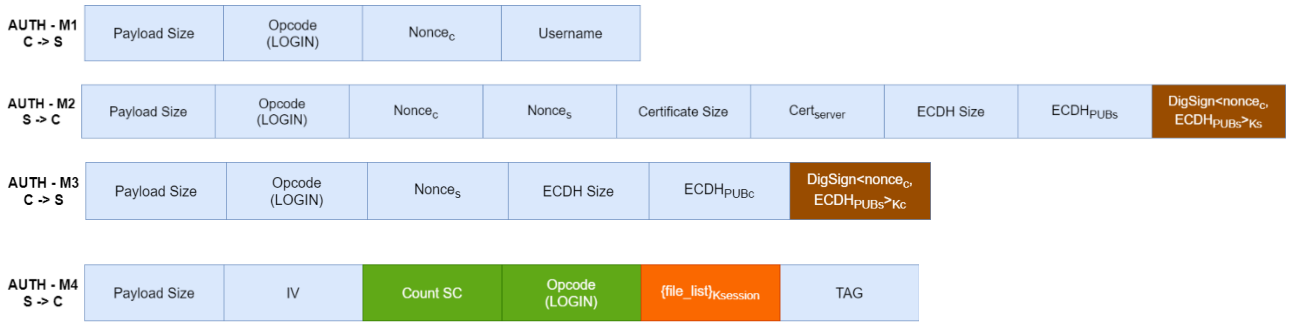
The project was implemented using the C++ programming language and the OpenSSL library.

2. Protocols

The protocols chosen for security requirements are the following:

- A **random nonce** is generated against the **Reply Attack** during the authentication phase
- In the authentication phase, the **Digital Signature** and the Hash (included in the Digital Signature algorithm) are used to guarantee **Authentication** and **Integrity**
- An incremental **counter** is used to ensure **Freshness** against the Reply attack during the session
- To guarantee the PFS (**Perfect Forward Secrecy**), the **ECDHKE** (Elliptic-curve Diffie-Hellman Key Exchange) is used for generating the symmetric session keys. Moreover, a **random IV** is generated for each encryption with the symmetric key, to avoid predictability
- To provide CIA (**Confidentiality-Integrity-Authenticity**) to the encrypted messages exchanged during the sessions, the **AES_128_GCM** mode is used

3. Authentication



Message field colors, valid in all phases:

At the beginning Client and Server must authenticate each other.

The process starts with the Client sending the message *AUTH - M1*, containing the username and the client nonce.

If the Server recognizes the username, it replies with the message *AUTH - M2*, sending the server nonce and the server certificate, alongside its ECDH public key just generated. In addition, also the signed hash of client nonce and server ECDH public key are sent.

When the Client receives *AUTH - M2*, he (she) sends back *AUTH - M3*, containing its own ECDH public key generated and the signed hash of its ECDH key and server nonce previously received, to be verified by the server through the client public key, pubKc.

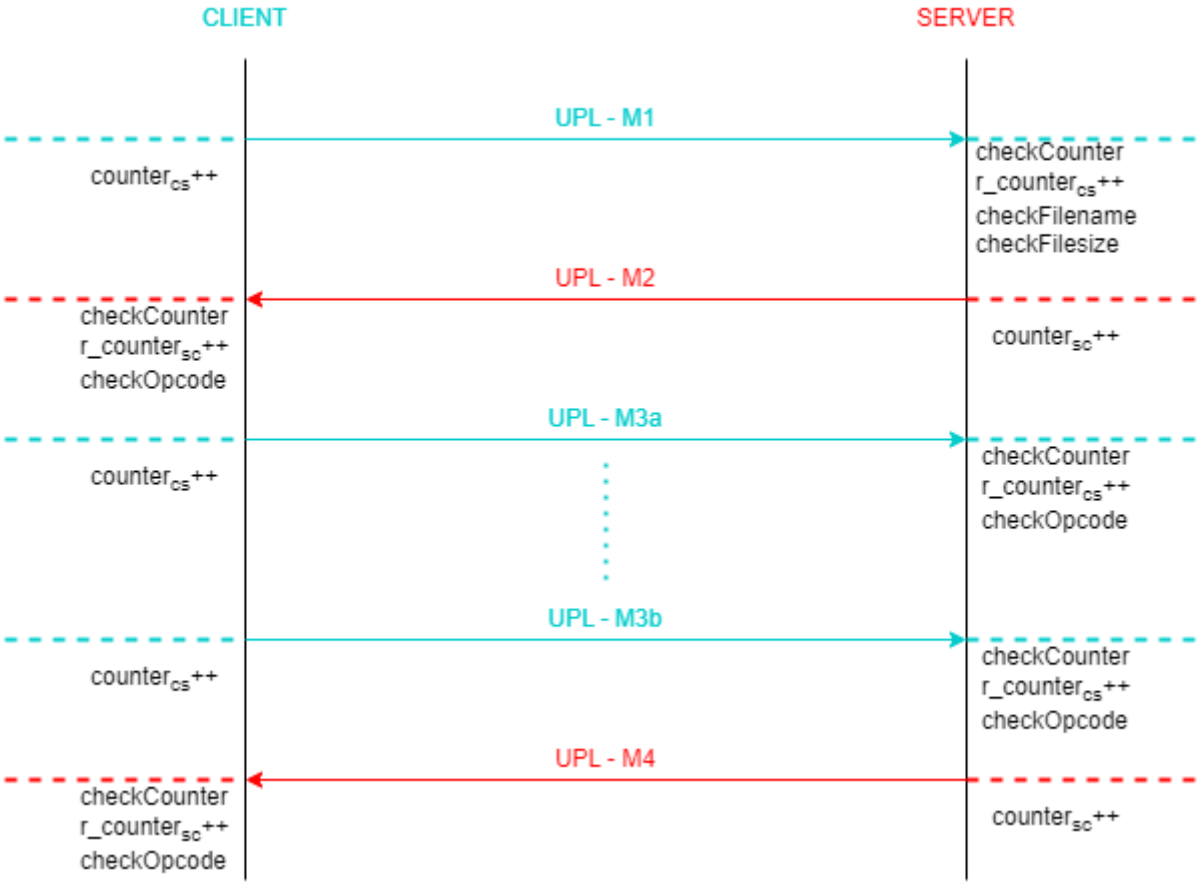
After the verifications, the two sides can derive the session key, enabling the use of symmetric encryption.

The last message of this phase (*AUTH – M4*) is the file list in the user's cloud storage, used as an ack sent by the Server to the Client to assess the correctfullness of the login.

If the client is not registered on the server on in case of any type of error in any message, the connection will be closed.

4. Operations

4.1 Upload



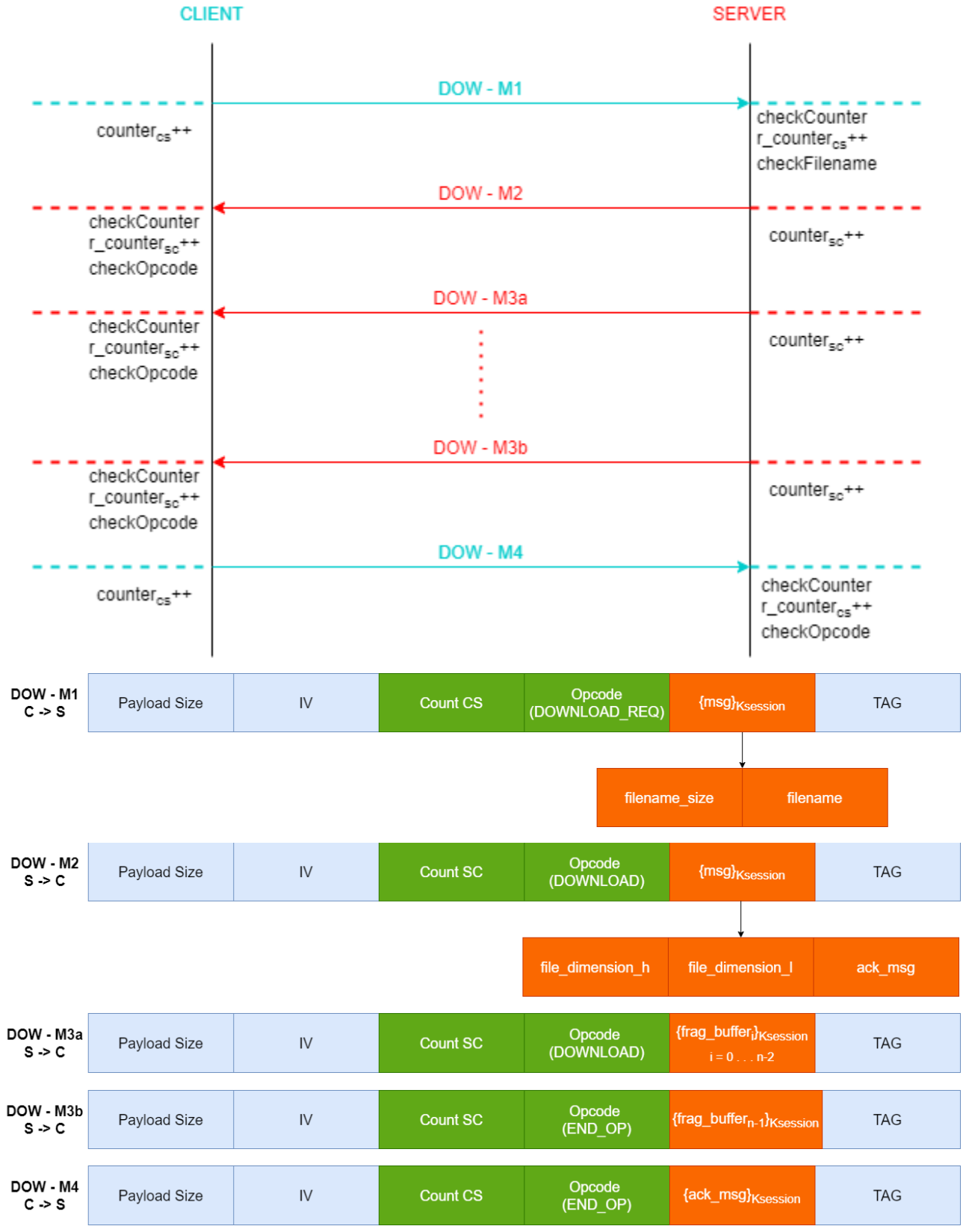
When an upload operation is requested, the Client sends the message *UPL - M1* to the Server, specifying the size of the file to be uploaded (with two variables because uint32_t type ranges from 0 to 2^32-1 and file can be up to 4GB= 2^32 bytes, but the uint64_t type can not be used because of the hton/ntoh functions, that take a maximum of 32 bit as argument) and the filename.

When the Server receives the message, it checks both the dimensions (that must be ≤4GB) and the filename (for the filename, the check is done through a whitelist because to be a valid name it can not contain special characters and, moreover, there must be no other file with the same name into the cloud storage). The Server then sends *UPL - M2* to the Client, containing the result of the checks, that may be positive or negative, i.e. the upload request may have been accepted or rejected, the latter if there already exists another file with the same name or if the name is not valid.

If everything went well, the request is accepted, the Client starts sending the file to the Server (splitting it into chunks if needed, i.e. if its total dimensions exceed 8KiB) as shown in *UPL - M3a/UPL - M3b*.

The last message exchanged is *UPL - M4*, through which the Server informs the Client whether the upload was successful or not.

4.2 Download



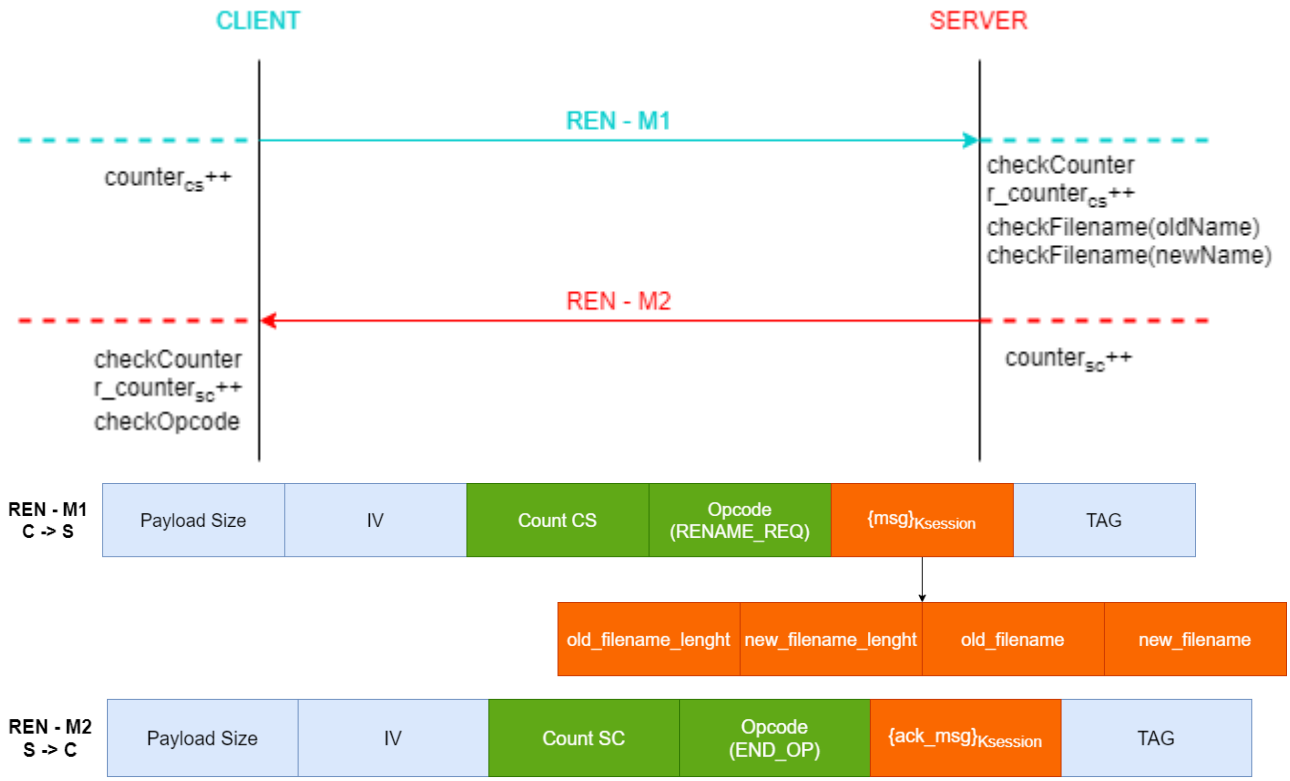
When a download operation is requested the Client sends the message *DOW - M1* to the Server, specifying the filename of the file to be downloaded.

When the Server receives the message, it checks both the correctness of the filename and also whether the file has been previously uploaded and it is already present on the cloud storage. It then sends *DOW - M2* to the Client, containing the result of the checks and the dimensions of the file (that will be < 0 if the file is not found on the cloud).

If the file is found, the Server starts sending it to the Client (dividing it into chunks if needed, i.e. if its size exceeds 8KiB) as shown in *DOW - M3a/DOW - M3b*.

The last message exchanged is *DOW -M4*, with which the Client informs the Server whether the download was successful or not.

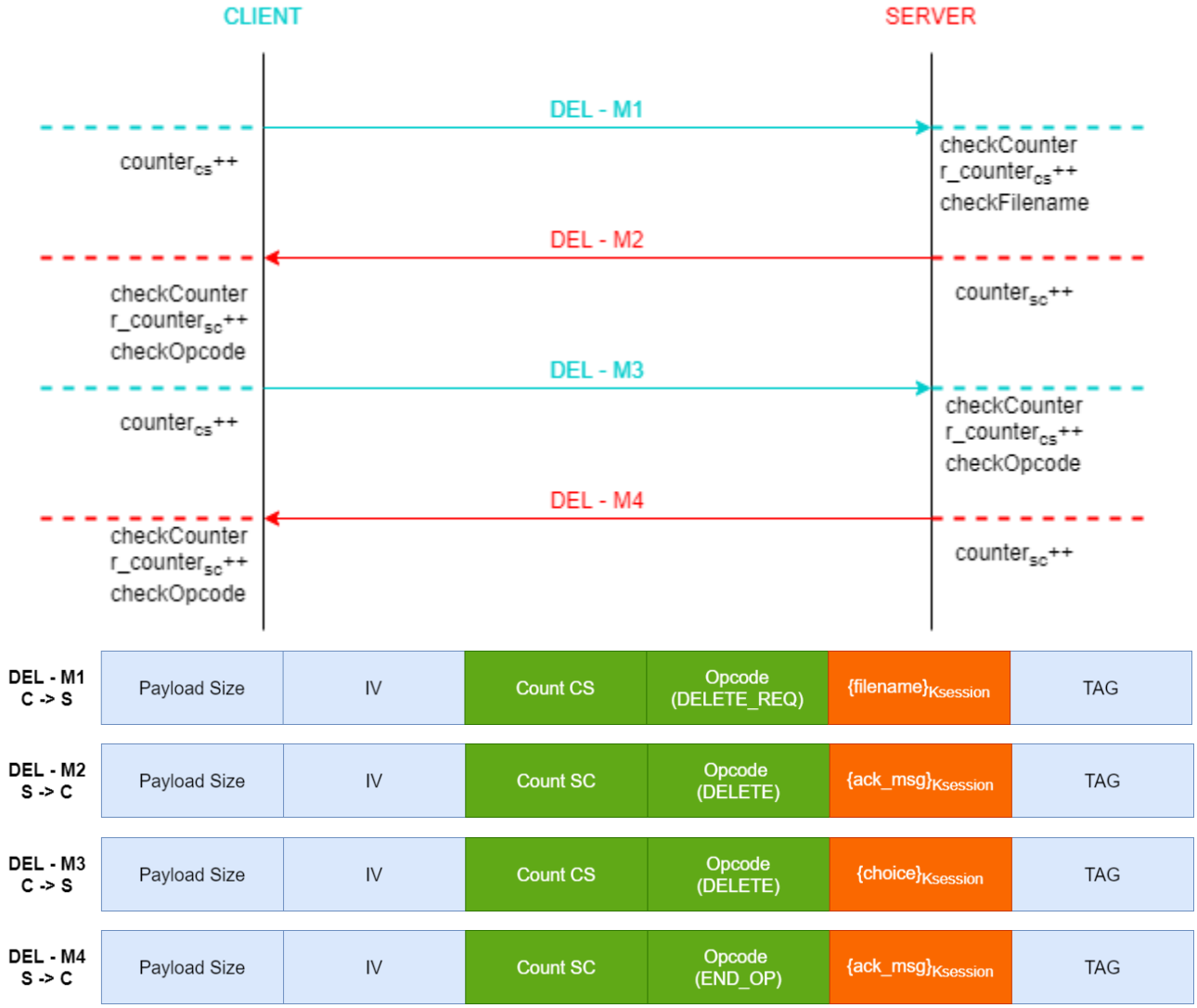
4.3 Rename



When a rename operation is requested the Client send the message *REN - M1* to the Server, specifying the filename of the file to be renamed and the new filename.

When the Server receives the message, it checks both if the old and the new filenames are valid names and if the file with to be renamed is present on the cloud storage. It then sends *REN - M2* to the Client, containing the result of the operation (successful or not).

4.4 Delete



DEL - M1 C -> S	Payload Size	IV	Count CS	Opcode (DELETE_REQ)	{filename}$_{Ksession}$	TAG
DEL - M2 S -> C	Payload Size	IV	Count SC	Opcode (DELETE)	{ack_msg}$_{Ksession}$	TAG
DEL - M3 C -> S	Payload Size	IV	Count CS	Opcode (DELETE)	{choice}$_{Ksession}$	TAG
DEL - M4 S -> C	Payload Size	IV	Count SC	Opcode (END_OP)	{ack_msg}$_{Ksession}$	TAG

When a delete operation is requested the Client send the message *DEL - M1* to the Server, specifying the filename of the file to be deleted.

When the Server receives the message, it checks the validity of the filename and whether the file is present on the cloud storage. It then sends *DEL - M2* to the Client, containing the result of the check.

The Client must, at this point, confirm with *DEL - M3* that she (he) wants to effectively execute the action or not.

Depending on the response received from the Client, the Server deletes (or not) the file from the cloud storage, sending an ack to the Client with the *DEL – M4*.

4.5 List

LIST - M1 C -> S	Payload Size	IV	Count CS	Opcode (FILE_LIST_REQ)	$\{username\}_{Ksession}$	TAG
LIST - M2a S -> C	Payload Size	IV	Count SC	Opcode (FILE_LIST)	$\{send\_frag_i\}_{Ksession}$ $i = 0 \ldots n\text{-}2$	TAG
LIST - M2b S -> C	Payload Size	IV	Count SC	Opcode (END_OP)	$\{send\_frag_{n-1}\}_{Ksession}$	TAG

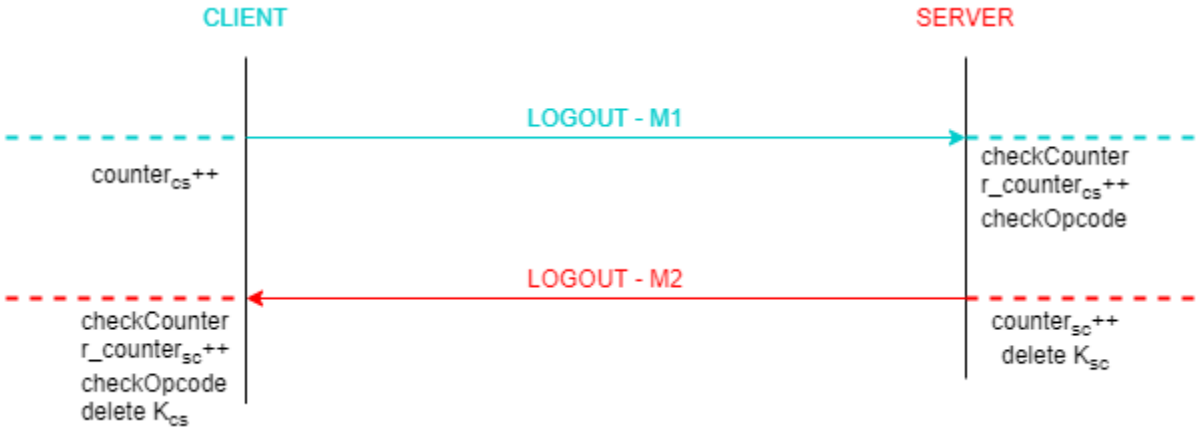
When a list operation is requested the Client send the message *LIST - M1*, requesting the file list from the Server.

When the Server receives the message, it checks how many and which files are present.

It then sends this data to the Client, dividing it into chunks if needed, i.e. if the dimension exceeds 8KiB, as shown in *LIST - M2a/LIST - M2b*.

The Client prints on screen the payload of these messages while it receives them.

4.6 Logout



LOGOUT - M1 C -> S	Payload Size	IV	Count CS	Opcode (LOGOUT)	$\{logout\_msg\}_{Ksession}$	TAG
LOGOUT - M2 S -> C	Payload Size	IV	Count SC	Opcode (END_OP)	$\{ack\_msg\}_{Ksession}$	TAG

When a logout operation is requested the Client send the message *LOGOUT - M1* to the Server. The Server acknowledges the Client of receiving the message with *LOGOUT - M2* and then proceeds to delete the session information related to the Client. When the Client receives LOGOUT - M2 it does the same.

In case of error in any message, the logout is performed anyway.