University of Pisa

Academic year 2021/22
Course of Applied cryptography

Cloud storage project report

Alessandro Zanatta[a] and Gianmarco Pastore[b]

[a]matr. 647047 – a.zanatta1@studenti.unipi.it
[b]matr. 646236 – g.pastore11@studenti.unipi.it

Contents

1 Key agreement protocol

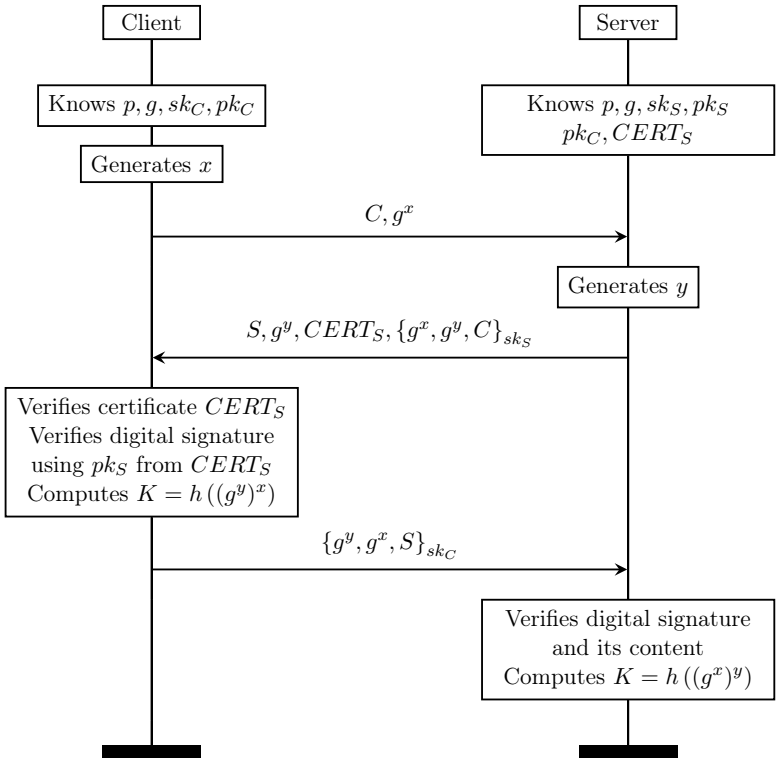

Figure 1: Key agreement protocol with PFS (ISO/IEC IS 9798-3)

The protocol in fig. 1 guarantees the following properties: authentication (of both client and server), key agreement, key freshness, and Perfect Forward Secrecy.

These properties will now be discussed:

Authentication The authentication of the client and of the server is given by the digital signatures. Moreover, the asymmetric keys used are also authentic, as the server's has the client's public key, and the client receives the server's certificate, which is verified using the root CA's certificate.

Key agreement The use of (authenticated) Diffie-Hellman ensures the agreement on the same symmetric key by both client and server.

Key freshness The key freshness, and in general the protection from replay attacks, is guaranteed by the signature. In particular, the Diffie-Hellman half keys are used "as nonces".

PFS Perfect Forward Secrecy is given by the use of Diffie-Hellman, as the future keys are not related in any way to the past ones.

2 Message format

2.1 Key agreement message format

The message format for the key agreement protocol is the one shown in fig. 2. The "type" field is the one used to indicate the message type. The payload is sent as pairs created as the payload length (2 bytes) and the actual content (variable length). For example, the first message of the key agreement protocol for the user "bob" is represented in fig. 3.

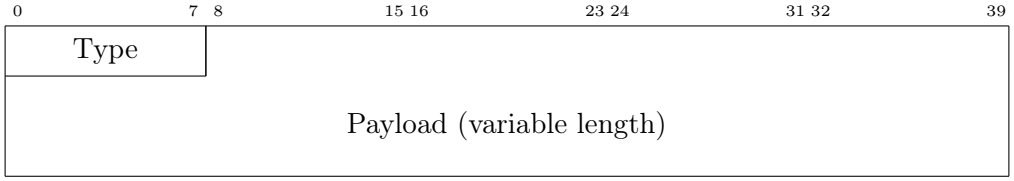

Figure 2: Key agreement message format

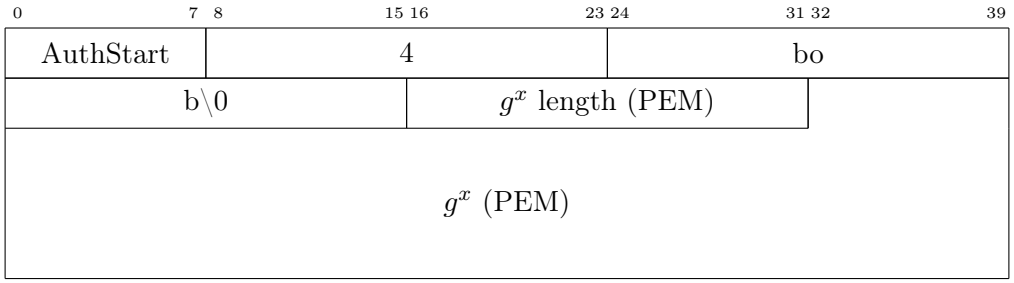

Figure 3: Key agreement example (first message)

2.2 Transport message format

Figure 4 shows the message format for the transport protocol used by the application. The type field is used as in the key agreement message format. The sequence number is used to prevent replay attacks. In particular, it is 32-bits long, it starts from zero, and, before the maximum value 2^{32} is reached, the connection between client and server is gracefully closed. The used encryption cipher is AES-256 GCM, therefore we also send the IV and the tag, along with the application payload. This encryption mode has been chosen as it allows to guarantee authenticity of the encrypted data. As it can be seen from the message format, the type and sequence number of the message are also authenticated (using GCM). The payload is encoded as previously described in section 2.1.

Legend:

- authenticated
- encrypted and authenticated

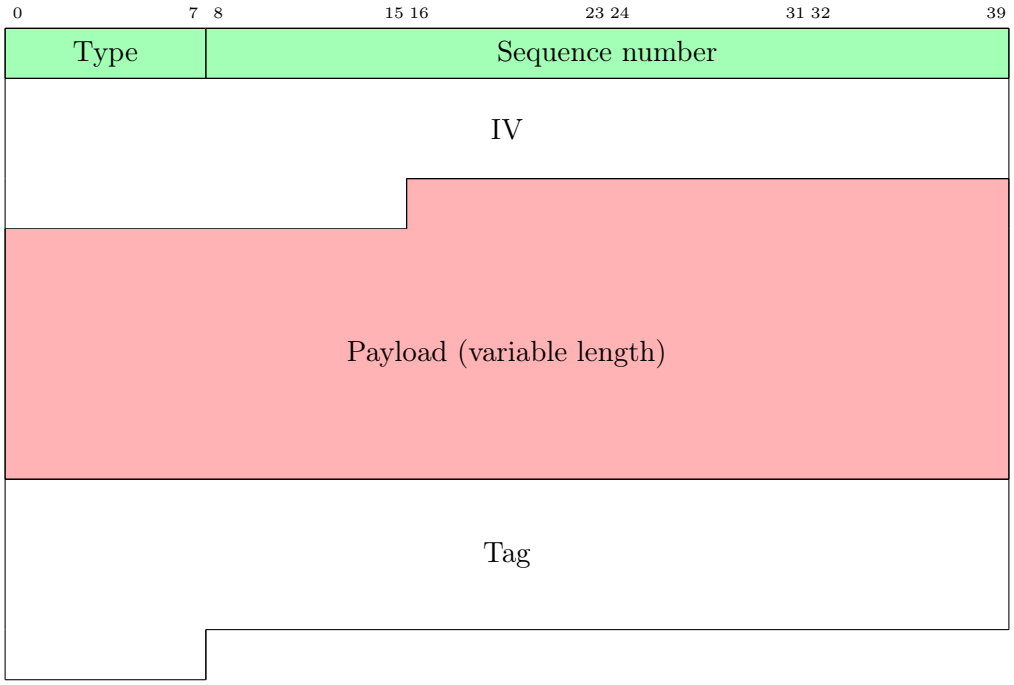

Figure 4: Transport protocol

3 Operations

The required operations have been implemented. The sequence diagram for each operation is reported in this section.

3.1 Upload

Figure 5 shows the sequence diagram for upload.

The client requests the upload of a file by sending a filename. The filename is checked to exist locally on the client-side, and to not be larger than 4GB. The server also checks that the filename is valid, meaning that:

- the file doesn't already exist on the user's storage
- the filename does not attempt a path traversal

If the above holds, the client can proceed to uploading the file. The upload is done by chunks of size 2^{15} bytes each. When uploading the file, the server additionally checks that the file size of 4GB is not exceeded. To indicate the end of the upload, we use a different message type.

If an error occurs on the client-side, the client can notify the server and abort the upload. The partially uploaded file on the server storage is deleted.

3.2 Download

The download (fig. 6) is very similar to the upload procedure, but with inverted roles. The relevant security checks are the following:

- The filename to download exists
- The filename to download is not outside of the user's storage directory (path traversal)
- The filename on which the client saves the downloaded file does not exist (no overwriting by error)

3.3 Delete

The delete operation (fig. 7) requires the client to send the filename to delete. After checking its validity as in the previous sections, the server asks for a confirmation. If the client confirms, the file is deleted, otherwise the operations is aborted. Notice that in the third message only authentication of the message type is required, and there is no meaningful encrypted value, therefore we encrypt a randomly generated sequence of byte of length 12.

3.4 List

File listing is straightforward (fig. 8). The client requests the listing, and the server simply reads the file list in the user's storage and sends it back.

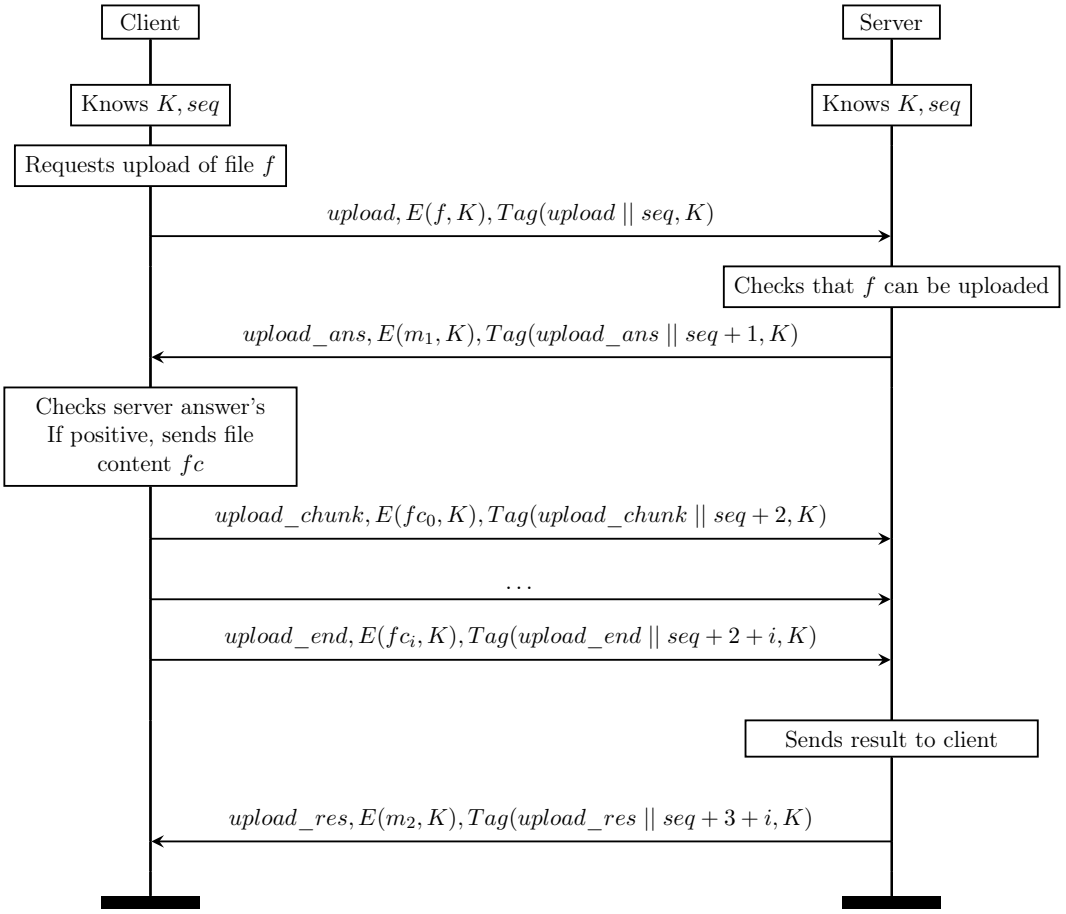

Figure 5: File upload

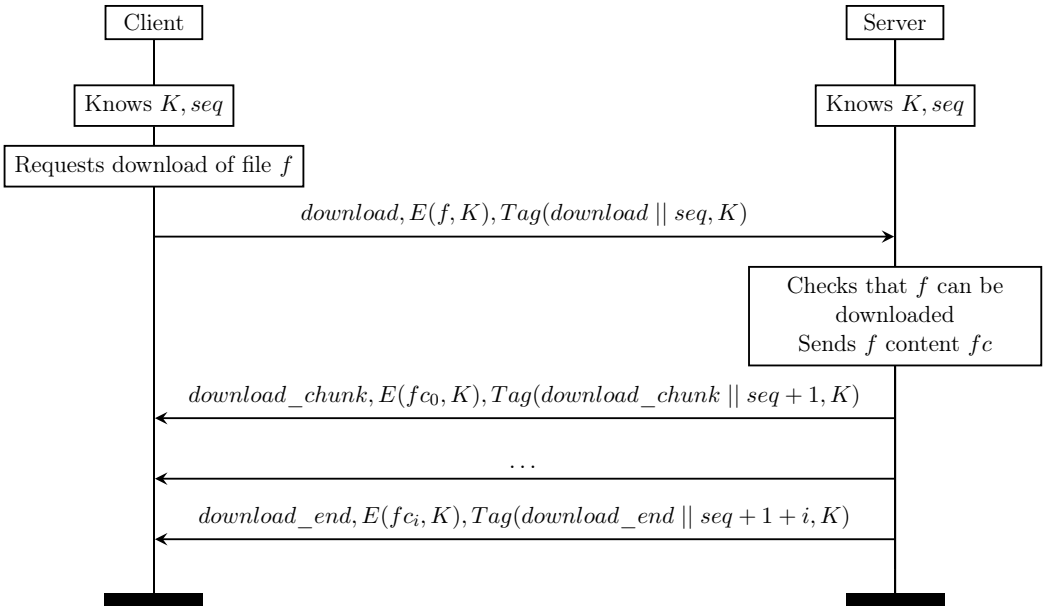

Figure 6: File download

3.5 Rename

File renaming requires the client to send two filenames, one for the file to rename, and one for the new name of the file. Both are checked to be valid (as previously seen), the first one is checked to exist, and the latter is checked to not exist. If the above holds, then the file is renamed.

3.6 Logout

For the logout, the client is always the initiator. Logout can happen in two different situations:

- the client can request it from the main menu, or by sending a SIGINT signal
- the client can request it when the sequence number approaches the maximum possible one. A certain threshold from the maximum sequence number is kept to allow the logout procedure to execute

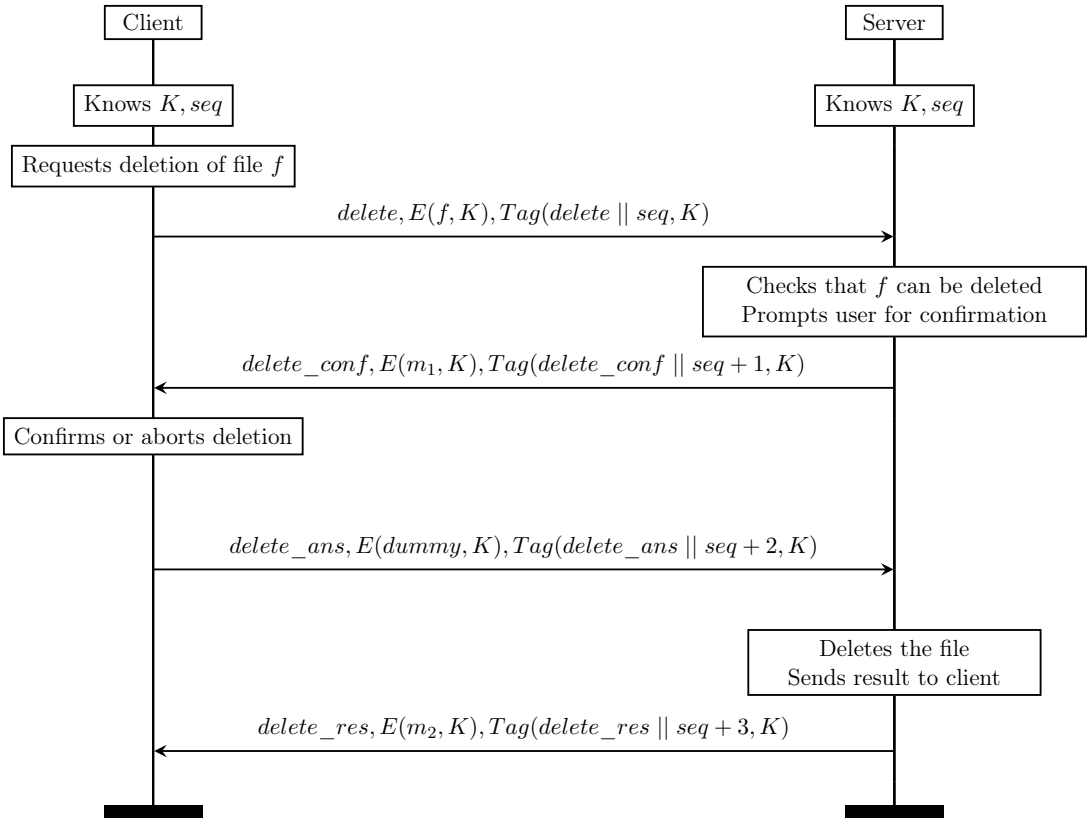

Figure 7: File delete

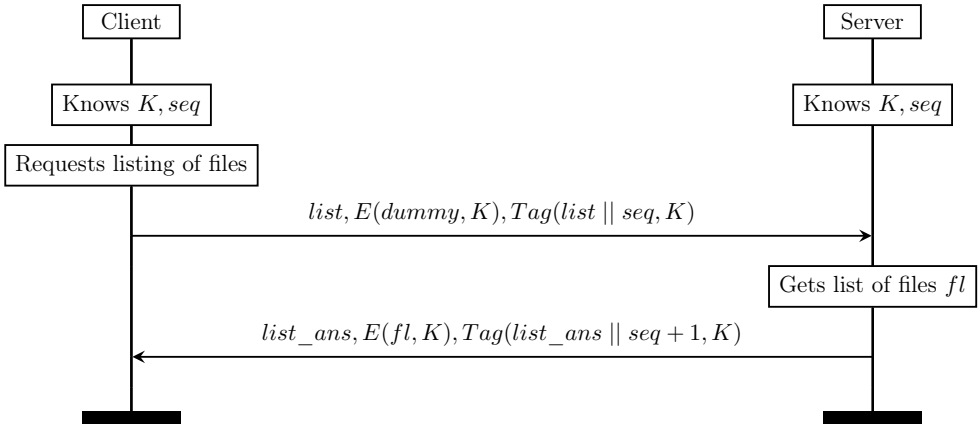

Figure 8: File listing

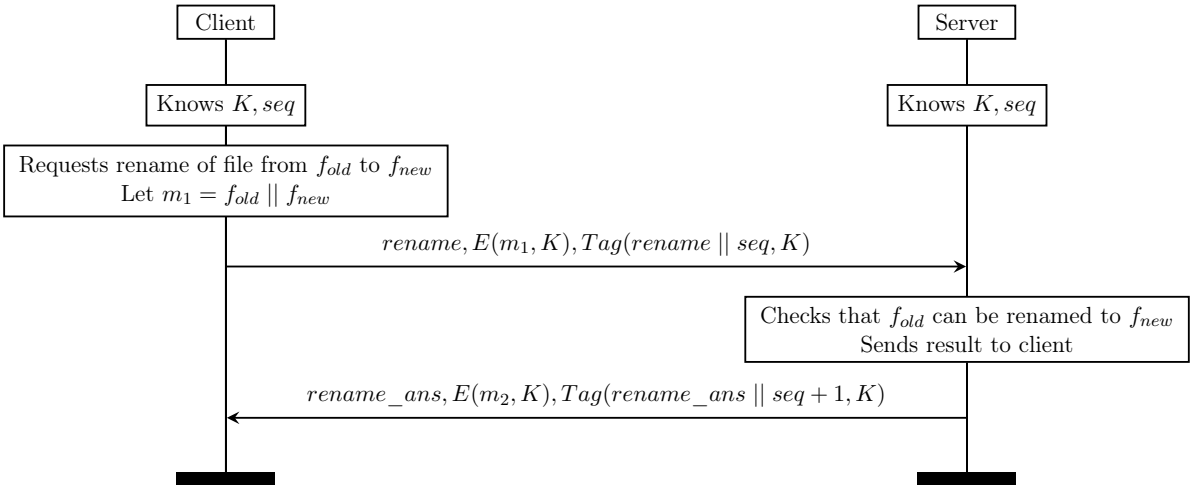

Figure 9: File rename

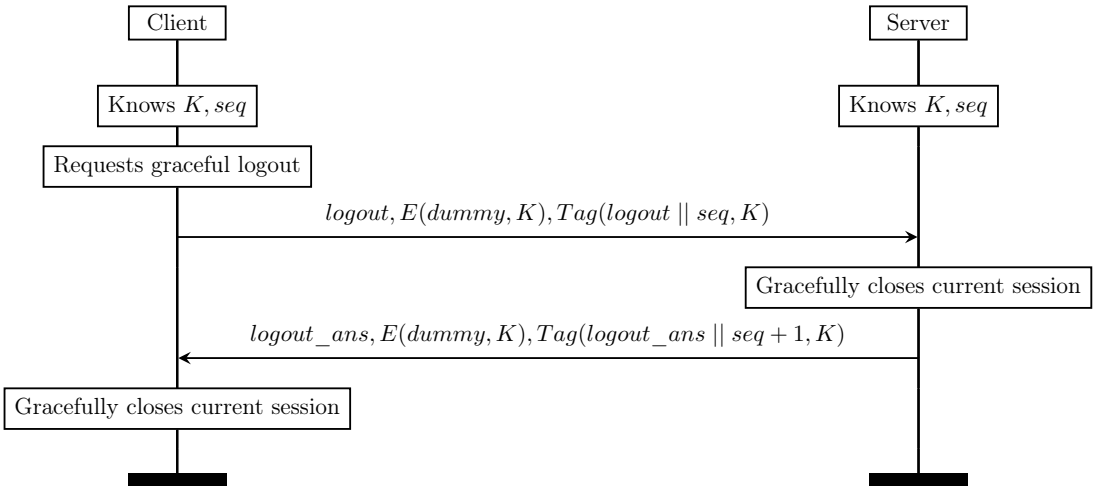

Figure 10: Log out