UNIVERSITÀ DI PISA

Department of Engineering

Cloud Storage

Cybersecurity Systems

Authors :
Eloi Comiran
Michael Asante
Farzaneh Moghani

Supervised by :
Prof. Gianluca Dini
Michele La Manna
Mariano Basile

A.Y. 2021-2022

Github: `https://github.com/fmoghani/2022-Cybersecurity`

Contents

1 Introduction

This project goal is to build a client–server application with enhanced security for message exchanges and file operations. Using this application, both client and server must be able to authenticate each other, establish a session key, have a secured dedicated storage on the server and client can perform the following operations:

- upload a file to his dedicated storage (with a maximum size of 4Gb)
- download a file from the his dedicated storage
- rename a file from his dedicated storage
- delete a file from his dedicated storage
- list the files from his dedicated storage
- logout from a session

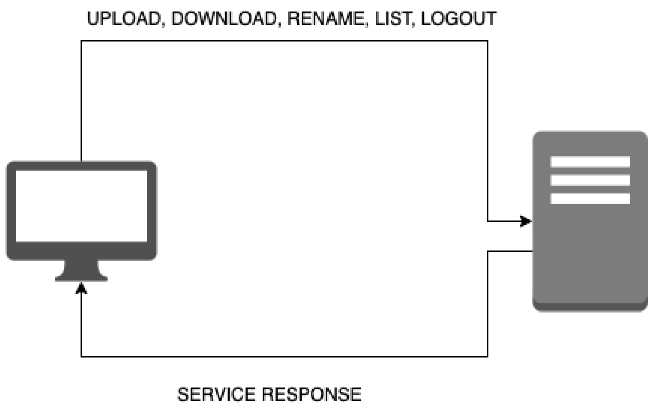

Figure 1: General structure of the system

All these operations must grant client secrecy about the files on his storage.
From a technical point of view, we consider that client and server pre-share some cryptographic material. Users have the following:

- the CA certificate
- a long-term RSA key pair (password protected)

On the other end server possess the following information:

- his own certificate, signed by CA
- it knows the username of every registered user
- it knows each user's public RSA key
- each user's dedicated storage is already allocated

The key exchange must provide perfect secrecy in order to encrypt and authenticate each session. Also the session must be protected against replay attacks.
In this paper, we first present the different protocols implemented in this application. We will then explain the global structure of the implementation. Following this description, we will use the BAN logic so we can assume that our protocol is secure. Eventually we will conclude about the safety of our application and the further progress that could be made.

2 Functional requirements

- Clients will have to authenticate with their private's key password in order to begin a session with the server. Once they are authenticated, the session is entirely encrypted and initial handshakes provides perfect forward secrecy.
- Server can only connect to one person at a time. To authenticate itself to the clients, it uses his certificate signed by the certification authority.

3 Protocols

3.1 Initial handshake

To establish authentication between the server and the client, first, client sends a fresh nonce to the server. Every time a nonce is required through the protocol it is computed as follows: 16 random bytes are generated using OpenSSL PRNG to ensure unpredictability and these bytes are concatenated with a timestamp which ensures uniqueness. Using RSA algorithm, server generates a temporal pair of private and public keys ($Tpriv_{key}$, $Tpub_{key}$). Server authenticates the ephemeral public key ($Tpub_{key}$) by signing it with the private key associated with his certificate. Server then sends to the client a message containing the temporary public key, the signature of the nonce concatenated with the public key signed by server's long-term private key, its certificate and a new nonce. Nonces deny the attacker the possibility of replay attacks. When the client receives the message, it first authenticates the temporary public key using server's certificate. Afterwards, a 512-bits session key (session key and authentication key), K, is generated, encrypted with the ephemeral public key and sent over to the server through M3. Client also sends the signature of server's nonce concatenated with the session key to complete authentication. When receiving message 3, server retrieves session key and complete authentication with nonce's signature. An additional protocol was included in M4 for ban logic. After client receives message 4 which contains the encrypted Nonce and session key, it now trusts the server's key. Eventually, server deletes the ephemeral key pair after the session ends. This protocol's sequence diagram is represented on *Figure 2*.

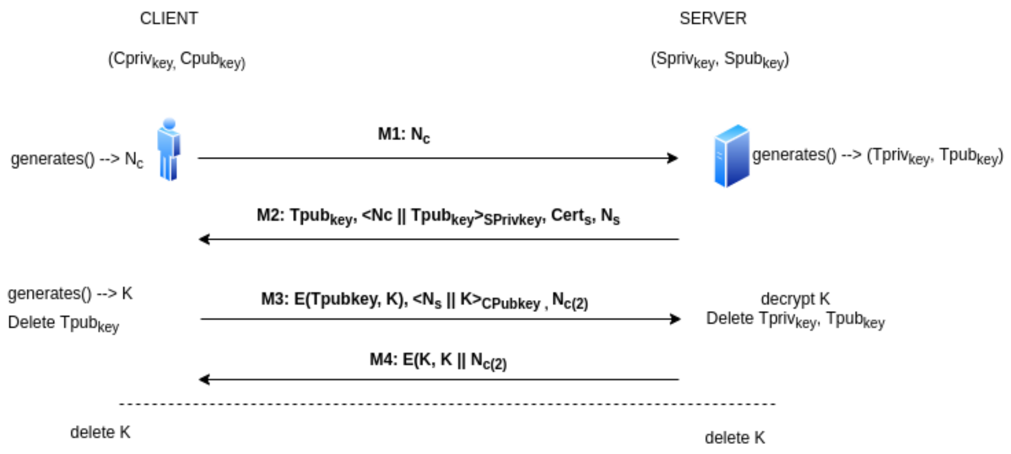
CLIENT
($Cpriv_{key}$, $Cpub_{key}$)
SERVER
($Spriv_{key}$, $Spub_{key}$)
generates() --> N_c
M1: N_c
generates() --> ($Tpriv_{key}$, $Tpub_{key}$)
M2: $Tpub_{key}$, $<Nc \| Tpub_{key}>_{SPrivkey}$, $Cert_s$, N_s
generates() --> K
Delete $Tpub_{key}$
M3: E(Tpubkey, K), $<N_s \| K>_{CPubkey}$, $N_{c(2)}$
decrypt K
Delete $Tpriv_{key}$, $Tpub_{key}$
M4: E(K, K $\|$ $N_{c(2)}$)
delete K
delete K

Figure 2: Sequence diagram of Ephemeral RSA key exchange

The diagram above outlines the process a server and a client has to go through to start message exchanges. The protocol grants Perfect Forward Secrecy.

3.2 Client-Server communication

After the initial handshake between server and client is completed, anytime one wants to send a message, we go through the following steps:

- Calculate the cipher text with AES_256_cbc(K_{sec}, plaintext).
- Create the digest by concatenating the authentication key, the counter and the ciphertext and hashing the result as:
  HMAC(K_{auth}, counter || ciphertext)
- Send the concatenated text as:
  (digest || ciphertext)

When a message is received, the following operations are performed to extract it:

- The received text is splitted to obtain the cipher text and the digest.
- Using the cipher text, the counter and K_{auth} exchanged during the Initial handshake, receiver computes the digest on his side.
- If it corresponds to the digest received, the ciphertext is decrypted and the plaintext is used.

With this method, replay attacks are prevented inside a session because if an attacker wants to replay and send the same message, the counter will not be the same as before and will therefore lead to a different digest. A received message is always authenticated before is decrypted which helps prevent Padding Oracle attacks. Moreover, an Initialization vector(IV) is used in the encryption and decryption of plain/cipher text. The IV is created using OpenSSL's PRNG. This ensures that sequence of IVs appear as random sequence. The counter prevents replay attacks within a session. Data Masking (obfuscation) of the plaintext makes it irrelevant to the attacker even if tempered with and makes it difficult to perform traffic analysis.
Each digest is created out of the authentication key, the counter and the ciphertext such that re-ordering attacks are also not possible by the attacker.

4 Implementation

4.1 File System

File system is represented on *Figure 3*. Server stores users' information in the folder `users_infos`, this folder contain a folder for each user named after user's username. And in each user specific folder (`khabib` in our example on *Figure 3*), server stores the public key associated with the user and a `files` folder which is the dedicated storage. The `prvKey.pem` file inside the `server_infos` folder contains the private key associated with server's certificate. On user side, private RSA key and file containing user's username are stored inside the `user_infos` folder. And when a user wants to upload a file, he should put it in the same folder as the executable. Also when downloading a file, the file will be written at the same level as the executable.

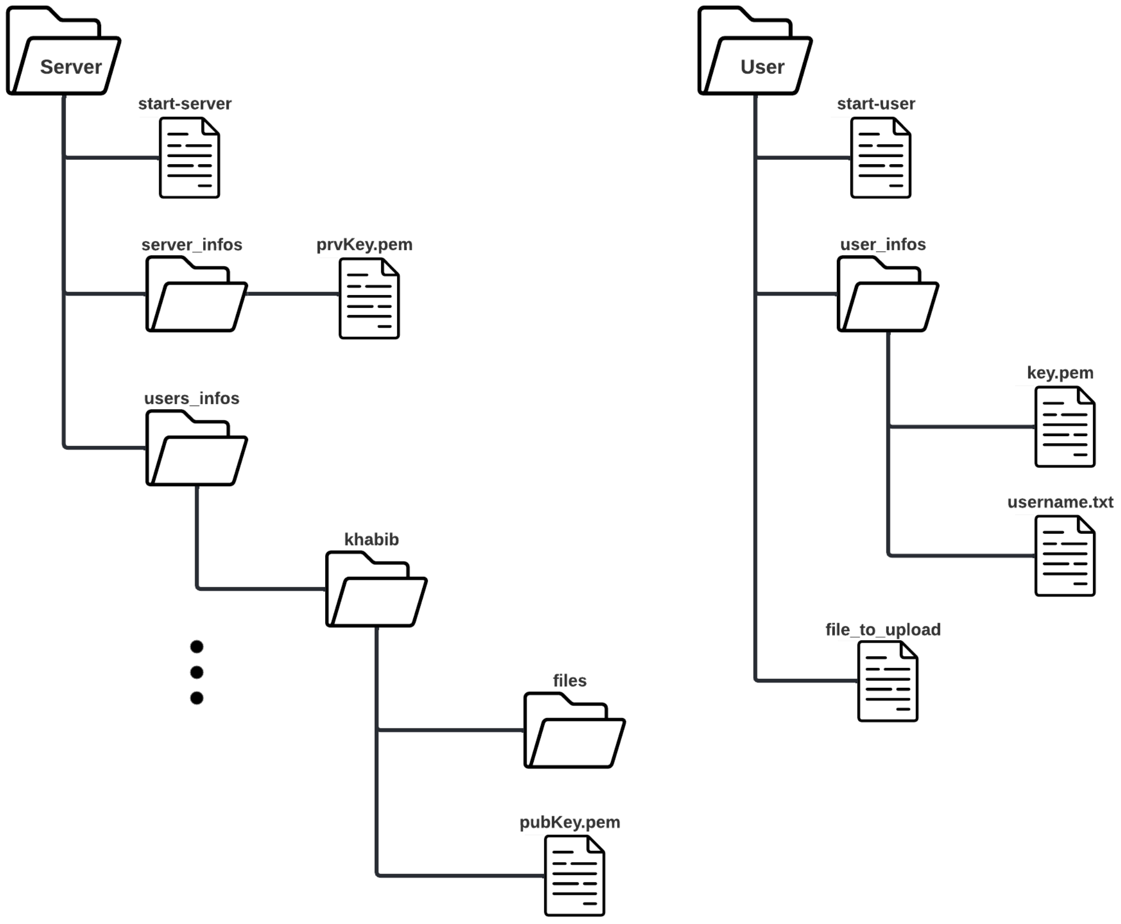

Figure 3: Chart representing the filesystem of our application

Certificates storage is not showed in *Figure 3* but it is a simple folder containing all certificates and placed at the same level as `Server` and `User` folders.
In the following parts, we will present the functions implemented. To develop our application, we created a class for server and a class for client. All constants are stored in a `const.h` file and a `utils.h` contains functions used multiple times in the code. Also about nonces, all the nonces were created by generating 16 random bytes with OpenSSL's PRNG to which is concatenated a timestamp to ensure uniqueness, this timestamp also

contributes to nonce's unpredictability since every machine will have a unique timing in generating the nonce which increases entropy.

4.2 Operations

The list of oeprations that are performed by the Client and the Server with respective Sequence diagrams are represented in this section:

- List

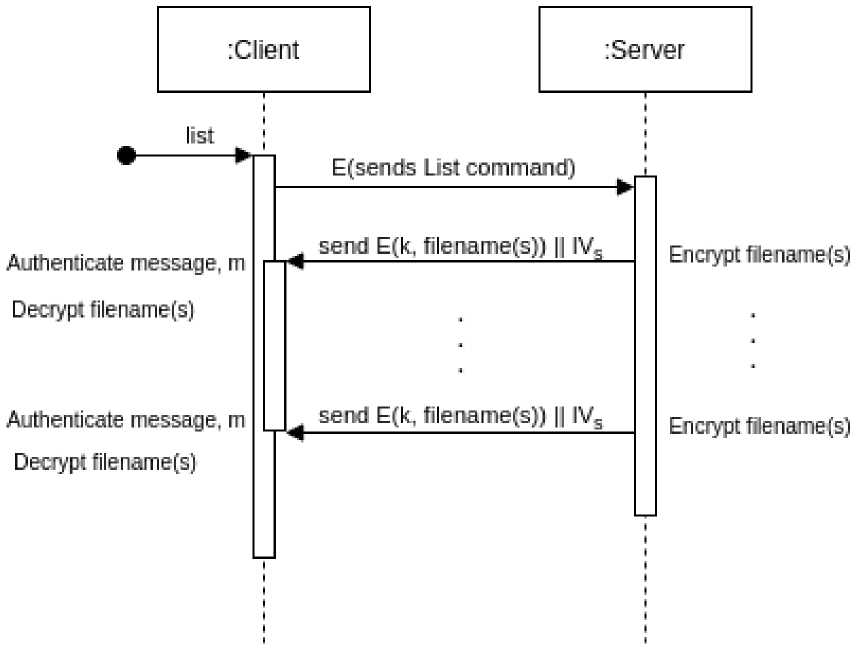

Figure 4: Sequence diagram for List operation.

To list files, a request for the operation is sent to the server. The server then loops through the users directory structure, generates the number of files available as well as the names of the files and sends it over encrypted to the user's request.

- Rename

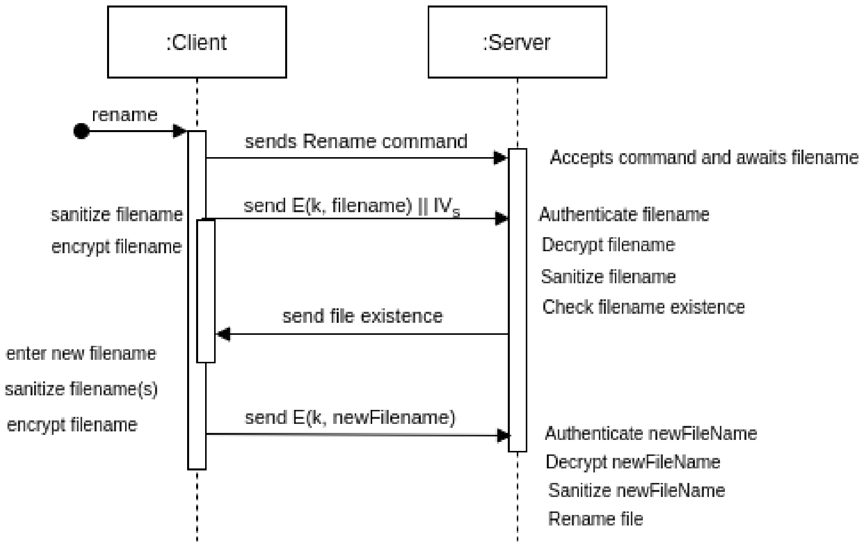

Figure 5: Sequence diagram for Rename operation.

To rename a file, the user will first have to request and enter the old filename. The filename is sanitized to check any unapproved character. After the filename is encrypted, it is sent to the server for which the server confirms if the file exists. If the file exists, then the user is asked for a new filename which is also encrypted and sent over to the server. The server uses the rename function which accepts the oldfilePath name and newfilePath name to change the file from the user's directory.

- Delete

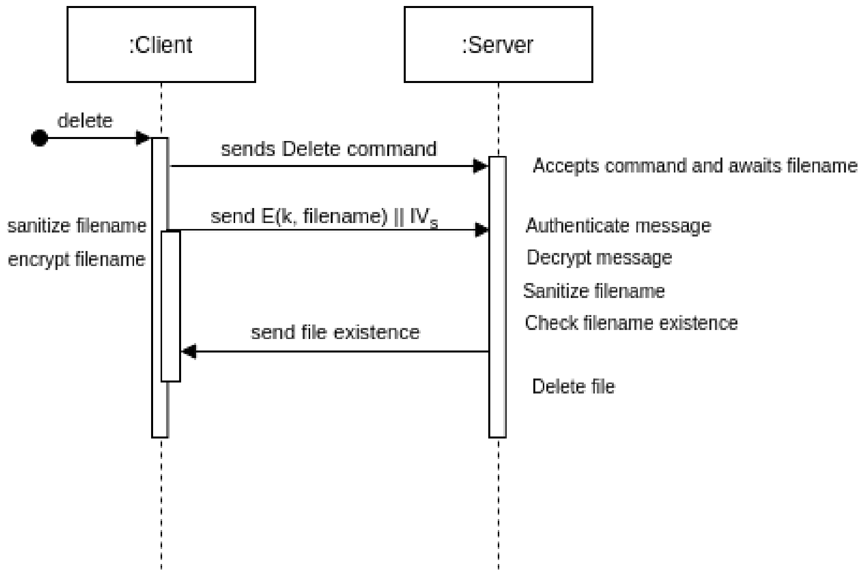

Figure 6: Sequence diagram for Delete operation.

For a client to delete a file from its server directory, a process is initiated with the client sending a delete request to the server. The client therefore enters the name of the file to delete from its storage space. First, filename is checked with the utility function for the valid path name, the filename is then encrypted and sent to the server. The server then checks if the file with the name exists in the client storage. If false, the server gets out of the function without doing anything. If true, then the server decrypts the filename, generates the path, free the buffers and removes the file the remove() method.

- Download

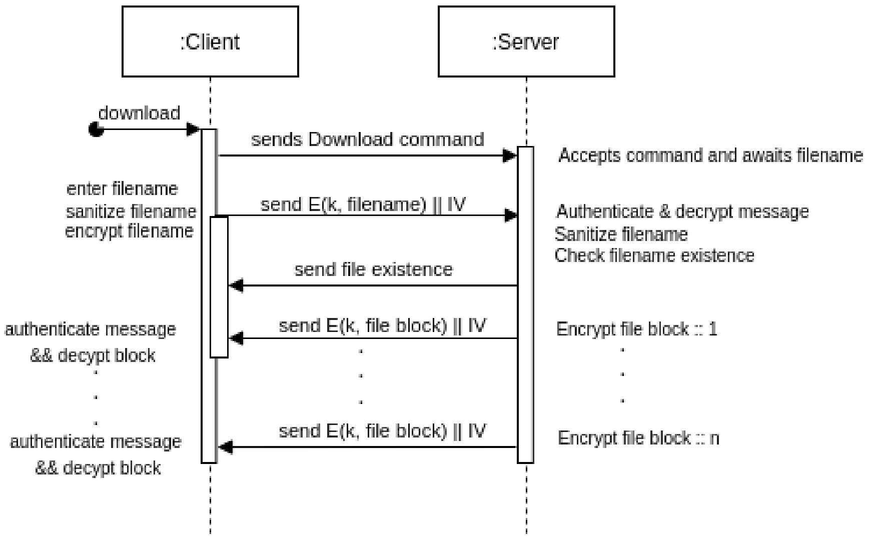

Figure 7: Sequence diagram for Download operation.

User can type the command "download" to request for a download operation. Then they should insert the filename, after checking the validity of the file name, it will be encrypted and sent to the server. Again, since the file is divided into smaller buffer size, client will receive, decrypt and read each block. Eventually, user can store the whole file on its local.

- Upload

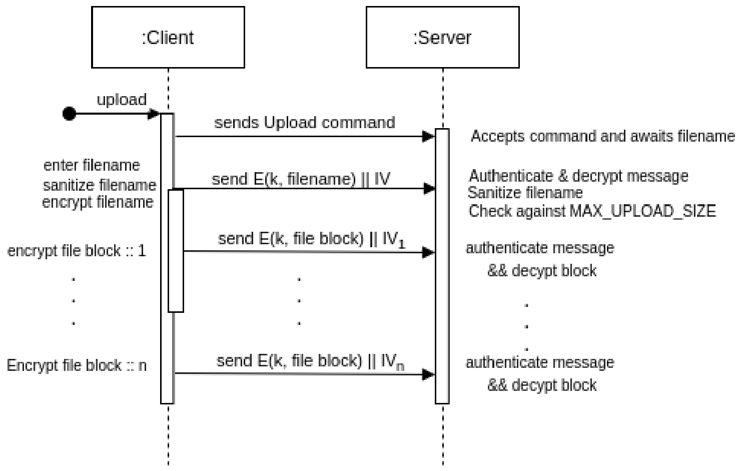

Figure 8: Sequence diagram for Upload operation.

In this function, the user initiates the process with "upload", it then asks for the file name. Afterwards the validity of the name is checked as well as the size of file. If the size of file is bigger than the MAX_UPLOAD_SIZE(4 GB). The system shows an error if this file exceeds the maximum supported file size. Then we send both file size and initial vector to the server. To upload the file, first we need to divide it into chunks, encrypt it and send it to the server. A buffer size is used to send the file. When we get to the end of file we print out the file has been successfully uploaded.

- Logout

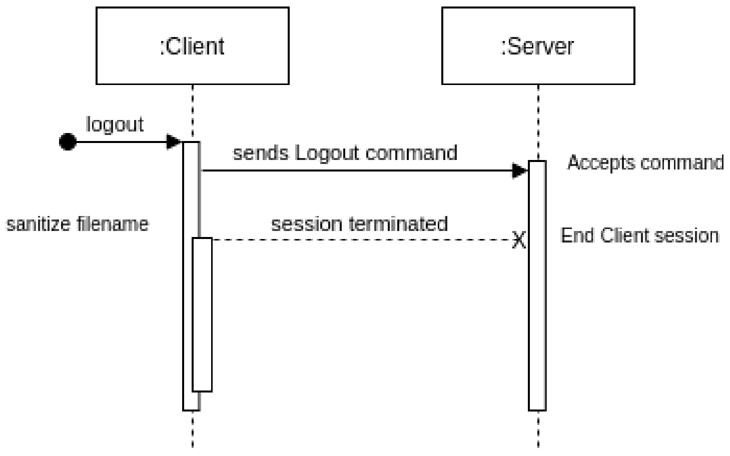

Figure 9: Sequence diagram for Logout operation.

The logout function upon instantiated enables the client to close its socket connection to the server and sets its connection status to 0. However, it is important to note that the server does not close connection as because other clients might be performing operations concurrently at the same time.

5 BANS

To prove the correctness of the protocols that we used in the implementation of this project we exploit the ban logic.
To do so, we will first explain the real protocol and construct the idealized protocol. We will then establish the objectives of the protocol and the assumptions of the following type:

- Keys
- Trust
- Freshness

Eventually we will use these assumptions and the idealized protocol in order to verify that the objectives can be obtained.

5.1 Idealized protocol

Using idealized protocol, we write the idealized protocol.

- $\mathrm{M2\ S \rightarrow C : \{\langle N_C \rangle_{TPubKey}\}_{K_S{}^{-1}}}$
- $\mathrm{M3\ C \rightarrow S : \{\langle N_S \rangle_K\}_{K_C{}^{-1}}, \{S \overset{K_{SC}}{\longleftrightarrow} C, \#(S \overset{K_{SC}}{\longleftrightarrow} C)\}}_{TPubKey}$
- $\mathrm{M4\ S \rightarrow C :} \{\langle N_{C_2} \rangle_{K_{SC}}\}_{K_{SC}}$

5.2 Assumptions

- Keys

$$\mathrm{C \mid\equiv \xmapsto{PubK_S} S, S \mid\equiv \xmapsto{PubK_C} C}$$

- Trust

$$\mathrm{S \mid\equiv C \Rightarrow S \overset{K_{SC}}{\longleftrightarrow} C}$$

- Freshness

$$\mathrm{C \mid\equiv \#(S \overset{K_{SC}}{\longleftrightarrow} C)}$$

5.3 Objectives

- Key establishment

$\mathrm{S \mid\equiv S \overset{K_{SC}}{\longleftrightarrow} C, C \mid\equiv S \overset{K_{SC}}{\longleftrightarrow} C, C \mid\equiv \xmapsto{TPubKey} S}$

- Key confirmation

$\mathrm{S \mid\equiv C \mid\equiv S \overset{K_{SC}}{\longleftrightarrow} C, C \mid\equiv S \mid\equiv S \overset{K_{SC}}{\longleftrightarrow} C}$

5.4 Analysis

- After M2

$\mathrm{C} \mid\equiv \mathrm{S} \mid\sim \langle \mathrm{N_C} \rangle_{TPubKey}$ and $\mathrm{C} \mid\equiv \#(\mathrm{N_C})$ using the nonce verification rule we get one of our objectives: $\mathbf{C} \mid\equiv \xmapsto{\mathbf{TPubKey}} \mathbf{S}$

- After M3

We immediately have that $\mathbf{C} \mid\equiv \mathbf{S} \xleftrightarrow{\mathbf{K_{SC}}}$
$\mathrm{S} \mid\equiv \mathrm{C} \mid\sim \mathrm{S} \xleftrightarrow{\mathrm{K_{SC}}} \mathrm{C}$ and $\mathrm{S} \mid\equiv \mathrm{C} \mid\sim \langle \mathrm{N_S} \rangle_{K_{SC}}$ so $\mathrm{S} \mid\equiv \#(\mathrm{N_S})$ and using the nonce verification rule we get the following objective:

$$\mathbf{S} \mid\equiv \mathbf{C} \mid\equiv \mathbf{S} \xleftrightarrow{\mathbf{K_{SC}}} \mathbf{C}$$

And using the jurisdiction rule and our trust assumption, we get a fourth objective: $\mathbf{S} \mid\equiv \mathbf{S} \xleftrightarrow{\mathbf{K_{SC}}} \mathbf{C}$

- After M4

$\mathrm{C} \mid\equiv \mathrm{S} \mid\sim \mathrm{S} \xleftrightarrow{\mathrm{K_{SC}}} \mathrm{C}$ and also $\mathrm{C} \mid\equiv \#(\mathrm{N_{C_2}})$. We can then use the nonce verification rule for the last time and get our last objective: $\mathbf{C} \mid\equiv \mathbf{S} \mid\equiv \mathbf{S} \xleftrightarrow{\boldsymbol{K_{SC}}} \mathbf{C}$
So this analysis guarantees that our protocol is correct, in fact the last message was added after reasoning with the BAN logic.

6 Conclusion

What we have done so far in this project is: creating a working client – server application, securing the communications from any passive adversary. And even if the adversary would retrieve a session key, only the corresponding session would be compromised. We also proved in this paper that our protocol is secure in theory.
For further progress, we could support the upload and download of folders containing files themselves. Right now, users can only upload and download files in their dedicated storage.