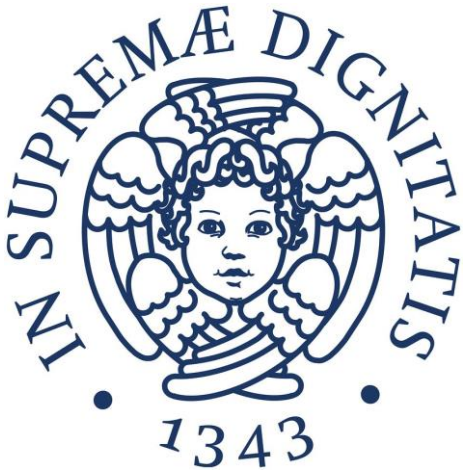
Università di Pisa



A.A. 2018/2019

CyberSecurity project

Davide Coccomini
Maurizio Pulizzi

INDEX

1 Introduction

The main scope of this project is to realize a secure client-server system that will be used to exchange files. In particular, the following operations have been implemented:

- The client can connect to the server and be authenticated;
- The client can ask for a list of the files on the server and receive a file containing this information;
- The client can download a file from the server to a local directory;
- The client can upload a file from a local directory to the server;
- The server can receive connections from verified clients;
- The server can read the list of the files in its directory and send this information to a client;
- The server can send one of its files to a client;
- The server can receive a new file, sent by the client, and store it.

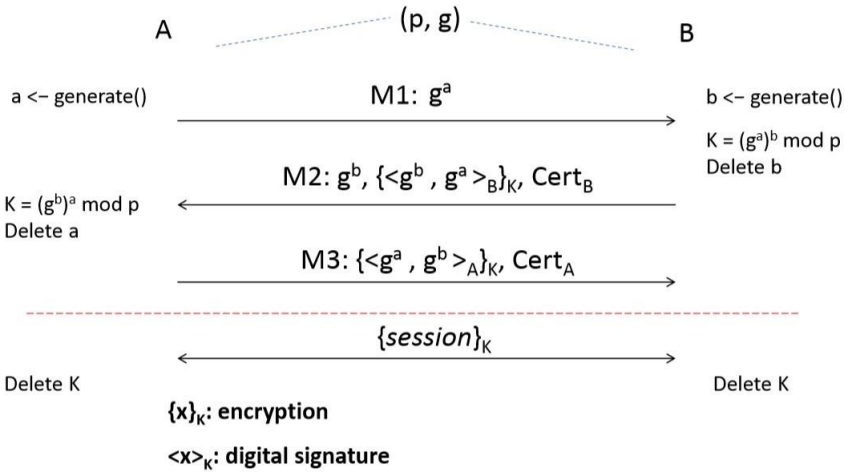
Both sides are also able to manage error situations maintaining a substantial state. The main attacks the communications are protected from are:

- Replay attacks;
- Tampering;
- Man in the Middle attacks;
- Oracle attacks.

2 Protocols

2.1 Initial handshake

To grant mutual key and entity authentication, we use the Station-to-Station protocol, a cryptographic key agreement scheme. The protocol is based on Diffie-Hellman but solve the problem of the Man-in-the-Middle attack. STS grant also the perfect forward secrecy and the direct authentication. To implement this protocol, it's needed that both parties have signature keys, which are used to sign messages. This protocol is used when the client connect to the server and it is summarized in the following scheme:



The messages M2 and M3 grant the perfect forward secrecy thanks to the creation of a and b, used in Diffie Hellman that are deleted at the end of each session. In this way, an attacker will not be able to reconstruct the message even if he obtains Ya and Yb.

2.2 Communication

After the initial handshake to grant the security on the messages exchanged between client and server, every time one of them wants to start a communication will do the following steps:

1. Calculate the cipher text as:
   $AES128(K_{sec}, plainText)$
2. Create the digest as:
   $SHA256(K_{aut}, counter||cipherText)$
3. Send the concatenated text:
   $digest||cipherText$

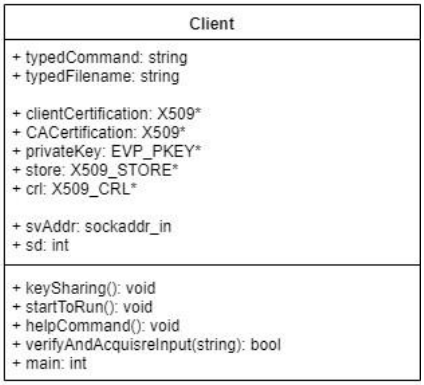
When the message is received, the following operations are made to extract the information:

1. The received text is splitted obtaining the digest and the cipher text;
2. The digest is checked by recalculating it starting from the received cipher text and comparing the result with the received one;
3. If the digest is correct, the cipher text is decrypted and the plain text is ready to be used;

3 Implementation

3.1 Classes

To implement the client and the server, two classes have been developed as follow:



These classes implement all the methods used from the user to interact with the server and from the server to react to the user requests. There are also the methods used for the initial handshake, implementing the station-to- station protocol, and the ones to certify the user identity.

3.2 Utility functions

Client and server use the methods contained in the file utilityFunctions.cpp. In particular, in this file there are:

- createDigest(): receives the cipher text and its size and generates a digest using HMAC;
- checkDigest(): receives a digest and a cipher text, generates a new digest starting from the received cipher text and checks if the received digest is equal with the generated one;
- sendSize(): sends to the other side the size of the message that is about to be sent using the explained communication protocol;
- receiveSize(): receives a concatenated text from the other side and extracts the size of the message that is about to be sent, as explained before;
- sendString(): calls to sendSize sending the plain text size and then sends a concatenated text using the explained communication proto- col;

- receiveString(): calls to receiveSize receiving the plain text size and then receives the concatenated text sent by the other side. Then extract the plain text as explained before;
- sendFile(): calls to sendSize sending the file size and then sends the file content using a variable number of sends based on the file size;
- receiveFile(): calls to receiveSize receiving the file size and then recei- ves the file blocks concatenating them to obtain the total file;

3.3 File transfer

Particular attention needs to be done about the files. In general, before starting a file transfer from the server to the client or vice versa, both sides calculate the number of chunks of fixed dimension that have to be sent (or received) and then the two sides start to make many sends and receives in each of which, only a little piece of the file is transferred. This specification is needed because of the memory efficiency problem. Indeed, in presence of very big files, it become very difficult (or impossible) to transfer the whole file in one sending. This because to transfer some piece of information, it have to be temporary stored in the RAM and this become very difficult for files of big sizes, due to the hardware limits. With this approach, instead of doing one big transfer, we make many transfers of fixed size (512 bytes), each of which doesn't fill the available memory.

4 BAN Logic Analysis

4.1 Session Keys exchange protocol

Message 1	$A \rightarrow B$: Ya
Message 2	$B \rightarrow A$: Yb, { $\langle Yb, Ya \rangle_B$ }$_{Kab}$ }, B's certificate
Message 3	$A \rightarrow B$: Ya, { $\langle Ya, Yb \rangle_A$ }$_{Kab}$ }, A's certificate

4.2 Idealized protocol

Message 1	$A \rightarrow B$: Ya
Message 2	$B \rightarrow A$: Yb, { {Yb, Ya}$_{Kb}^{-1}$ }$_{Kab}$, {|-^{Kb} ->B}$_{Kca}^{-1}$
Message 3	$A \rightarrow B$: Ya, { {Ya, Yb}$_{Ka}^{-1}$}$_{Kab}$, {|-^{Ka} - A}$_{Kca}^{-1}$

4.3 Assumptions

A |≡ |-^{Kca} ->CA ,	B |≡ |-^{Kca} ->CA
A |≡ CA ==> |-^{Kb} ->B ,	B |≡ CA ==> |-^{Ka} ->A
A |≡ #(Ya) ,	B |≡ #(Yb)
A |≡ #(a) ,	B |≡ #(b)

4.4 Objectives

- Key authentications: 1) A \|≡ A <-^{Kab}-> B , 2) B \|≡ A <-^{Kab}-> B
- Key confirmations: 3) A \|≡ B \|≡ A <-^{Kab}-> B , 4) B \|≡ A \|≡ A <-^{Kab}-> B
- Key freshness: 5) A \|≡ #(A <-^{Kab}-> B) , 6) B \|≡ #(A <-^{Kab}-> B)

4.5 Objectives verifications

Message 1:

- Assumptions: $B \lhd Ya$, $B \mid\equiv \#(b)$, B compute $Kab = (Ya)^b$
- Assertion: $B \mid\equiv \#(Kab)$ [*Key freshness verified for B*]

Message 2:

- Assumptions: $A \lhd \{ \mid\sim^{Kb} \rightarrow B \}_{Kca}^{-1}$, $A \mid\equiv \mid\sim^{Kca} \rightarrow CA$
- Assertion: $A \mid\equiv CA \mid\sim \mid\sim^{Kb} \rightarrow B$

After decrypting the certificate:

- Assumptions: $A \mid\equiv CA \mid\equiv \mid\sim^{Kb} \rightarrow B$, $A \mid\equiv CA \Rightarrow \mid\sim^{Kb} \rightarrow B$
- Assertion: for *Jurisdiction rule* $A \mid\equiv \mid\sim^{Kb} \rightarrow B$

- Assumptions: $A \lhd Yb$, $A \mid\equiv \#(a)$, A compute $Kab = (Yb)^a$
- Assertion: $A \mid\equiv \#(Kab)$ [*Key freshness verified for A*]

Decryption of $\{ \{Yb, Ya\}_{Kb}^{-1} \}_{Kab}$:

- Assumptions: $A \lhd \{ \{Yb, Ya\}_{Kb}^{-1} \}_{Kab}$, $A \mid\equiv A \leftarrow^{Kab} \rightarrow B$ (yet to be verified!)
- Assertion: $A \lhd \{Yb, Ya\}_{Kb}^{-1}$, for *message meaning rule* $A \mid\equiv B \mid\sim \{Yb, Ya\}_{Kb}^{-1}$

Signature verification:

- Assumptions: $A \lhd \{Yb, Ya\}_{Kb}^{-1}$, $A \mid\equiv \mid\sim^{Kb} \rightarrow B$
- Assertions: $A \lhd (Yb, Ya)$, for *message meaning rule* $A \mid\equiv B \mid\sim (Yb, Ya)$

$A \mid\equiv \#(Ya)$ implies that $A \mid\equiv \#(Yb,Ya)$.
$A \mid\equiv \#(Yb,Ya)$ and $A \mid\equiv B \mid\sim (Yb, Ya)$ implies, for *nonce verification rule*, that $A \mid\equiv B \mid\equiv (Yb, Ya)$.
$A \mid\equiv B \mid\equiv (Yb, Ya)$ implies that $A \mid\equiv A \leftarrow^{Kab} \rightarrow B$ and $A \mid\equiv B \mid\equiv A \leftarrow^{Kab} \rightarrow B$.
[*Key* authentication *and key confirmation verified for B*]

Message 3:

- Assumptions: $B \triangleleft \{ |-^{Ka} \rightarrow A \}_{Kca}^{-1}$, $B |\equiv |-^{Kca} \rightarrow CA$
- Assertion: $B |\equiv CA |\sim |-^{Ka} \rightarrow A$

After decrypting the certificate:

- Assumptions: $B |\equiv CA |\equiv |-^{Ka} \rightarrow A$, $B |\equiv CA \Longrightarrow |-^{Ka} \rightarrow A$
- Assertion: for *Jurisdiction rule* $B |\equiv |-^{Ka} \rightarrow A$

Decryption of $\{ \{Yb, Ya\}_{Kb}^{-1} \}_{Kab}$:

- Assumptions: $B \triangleleft \{ \{Ya, Yb\}_{Ka}^{-1} \}_{Kab}$, $B |\equiv A <-^{Kab}-> B$ (yet to be verified!)
- Assertion: $B \triangleleft \{Ya, Yb\}_{Ka}^{-1}$, for *message meaning rule* $B |\equiv A |\sim \{Ya, Yb\}_{Ka}^{-1}$

Signature verification:

- Assumptions: $B \triangleleft \{Ya, Yb\}_{Ka}^{-1}$, $B |\equiv |-^{Ka} \rightarrow A$
- Assertions: $B \triangleleft (Ya, Yb)$, for *message meaning rule* $B |\equiv A |\sim (Ya, Yb)$

$B |\equiv \#(Yb)$ implies that $B |\equiv \#(Ya, Yb)$.
$B |\equiv \#(Ya, Yb)$ and $B |\equiv A |\sim (Ya, Yb)$ implies, for *nonce verification rule*, that $B |\equiv A |\equiv (Ya, Yb)$.
$B |\equiv A |\equiv (Ya, Yb)$ implies that $B |\equiv A <-^{Kab}-> B$ and $B |\equiv A |\equiv A <-^{Kab}-> B$.
[*Key* authentication *and key confirmation verified for A*]