
University of Pisa
MSc in Computer Engineering
Foundation of Cybersecurity

# Foundation of Cybersecurity project

Alexander De Roberto
Leonardo Fontanelli
Eugenia Petrangeli

Anno Accademico 2019/2020

# Contents

# 1 Introduzione

L'obiettivo del progetto è di sviluppare una applicazione client-server in cui:

- Il server salva in locale un set di file, ognuno con dimensione non maggiore di 4 GB.
- Il client può richiedere una lista di file disponibili sul server.
- Il client può scaricare uno dei file disponibili. Questa operazione è memory-efficient sia per il client che per il server.
- Il client può caricare i file sul server. Se un file con il nome specificato dall'utente è già esistente questo viene sovrascritto.

Tutte le comunicazioni sono confidenziali, autenticate e reply-protected. È stato utilizzato il linguaggio C++ e la libreria di OpenSSL.

## 2 Scelte Implementative

Di seguito vengono riportate le principali scelte implementative fatte:

- Sia il server che il client sono autenticati con un certificato pubblico che è stato rilasciato da una autorità certificata.
- Viene utilizzata il metodo di cifratura AES-128 in modalità CBC per garantire la confidenzialità.
- Ogni messaggio include un numero sequenziale che sarà incrementato di uno ad ogni messaggio scambiato. Questo viene utilizzato contro il reply-attack. Il numero sequenziale viene inizialmente generato in maniera pseudo randomica con un valore tra 0 e UINT32_MAX / 2. In questo modo garantisce la possibilità di scambiare almeno UINT32_MAX / 2 messaggi. Viene fatto il controllo che questo non assuma gli stessi valori in una sessione. In ogni sessione il client e il server avranno i loro rispettivi numeri sequenziali.
- Le operazioni sono memory efficient in quanto ogni file, quando viene inviato, viene suddiviso in chunck grandi 512 KiB.
- Il server, quando riceve la richiesta di connessione da parte di un client, controlla se questo è presente in una lista privata di client autorizzati. Se così non fosse la connessione con il client termina.
- Per garantire l'autenticazione dei messaggi viene inserito all'interno del messaggio un digest da 32 byte generato con una Keyed Hash Function usando l'algoritmo SHA-256.
- Le chiavi pubbliche del client e del Server vengono ricavate dai rispettivi certificati scambiati in fase di handshake.
- Le chiavi private e i certificati vengono generati utilizzando la libreria OpenSSL. Viene prima generata la chiave privata e successivamente viene utilizzata per creare una richiesta di certificato che a sua volta viene inviata alla Certificate Authority per creare il certificato.
- Le chiavi di sessione, autenticazione e IV vengono generati in maniera pseudo randomica.
- Per quanto riguarda la criptazione e decriptazione simmetrica, IV verrà incrementato per ogni messaggio inviato.
- Nel caso in cui il messaggio atteso dovesse avere una grandezza maggiore rispetto al numero di byte effettivamente ricevuti, verrà visualizzato un messaggio di errore.
- I file con dimensione maggiore a 4 GiB non verranno accettati.

# 3 Handshake

Nella prima fase della comunicazione tra Client e Server, l'Handshake, l'iniziatore è sempre il Client.
Lo schema del handshake proposto è il seguente:

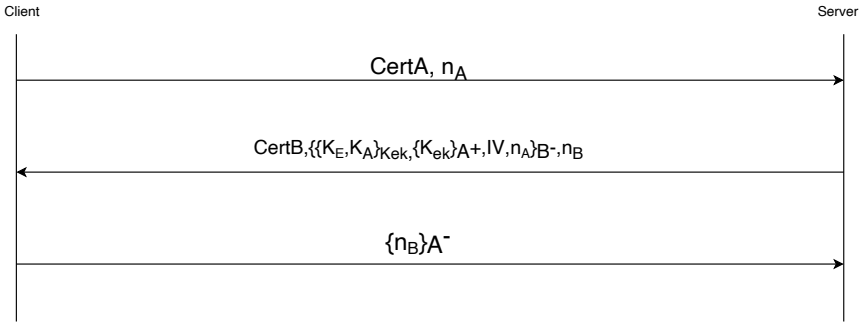


Figure 1: Protocollo di Handshake

La legenda del protocollo è la seguente:

- $Cert_A$ è il certificato del client, rilasciato da un Simple Authority.
- $n_A$ è il numero iniziale che cambierà per ogni messaggio scambiato dal client. Contro il reply attack.
- $n_B$ è il numero inziale che cambierà per ogni messaggio scambiato dal server. Contro il reply attack.
- $K_E$ è la chiave simmetrica di cifratura generata randomicamente dal server.
- $K_A$ è la chiave simmetrica di autenticazione generata randomicamente dal server.
- $K_{ek}$ chiave generata dalla funzione *EVP_SealInit()*.
- $IV$ è il vettore di inizializzazione, generato dal server.
- $A$ è la chiave del Client.
- $B$ è la chiave del Server.

La terza fase dell'handshake, comunica al server che $n_B$ è stato ricevuto correttamente.

# 4 Ban Logic

Di seguito viene proposta la dimostrazione della correttezza della fase di handshake per mezzo della BAN logic.

**Obiettivi**

Key authentication:

- $A \models A \overset{K_a}{\longleftrightarrow} B$
- $A \models A \overset{K_e}{\longleftrightarrow} B$

Key confirmation:

- $A \models B \models A \overset{K_a}{\longleftrightarrow} B$
- $A \models B \models A \overset{K_e}{\longleftrightarrow} B$

**Ipotesi**

- $A \models \#(n_A)$
- $A \models \#(n_B)$
- $A \models \#(A \overset{K_a}{\longleftrightarrow} B)$
- $A \models \#(A \overset{K_e}{\longleftrightarrow} B)$
- $A \models \#(A \overset{K_{ek}}{\longleftrightarrow} B)$
- $A \models \overset{K_{pubCA}}{\longmapsto} CA$
- $B \models \overset{K_{pubCA}}{\longmapsto} CA$
- $CA \models \overset{K_{pubA}}{\longmapsto} A$
- $CA \models \overset{K_{pubB}}{\longmapsto} B$
- $A \models CA \Rightarrow \overset{K_{pubB}}{\longmapsto} B$
- $B \models CA \Rightarrow \overset{K_{pubA}}{\longmapsto} A$
- $A \models B \Rightarrow A \overset{K_a}{\longleftrightarrow} B$
- $A \models B \Rightarrow A \overset{K_e}{\longleftrightarrow} B$
- $A \models B \Rightarrow A \overset{K_{ek}}{\longleftrightarrow} B$
- $A \models B \Rightarrow \#(A \overset{K_a}{\longleftrightarrow} B)$
- $A \models B \Rightarrow \#(A \overset{K_e}{\longleftrightarrow} B)$
- $A \models B \Rightarrow \#(A \overset{K_{ek}}{\longleftrightarrow} B)$

**Protocollo idealizzato**

$$M1 : A \rightarrow B\{K_{pubA}\}_{CA^{-1}}$$

$$M2 : B \rightarrow A\{K_{pubB}\}_{CA^{-1}}, \{\{A \overset{K_a}{\longleftrightarrow} B, A \overset{K_e}{\longleftrightarrow} B\}_{K_{ek}}, \{A \overset{K_{ek}}{\longleftrightarrow} B\}_{K_{pubA}}, IV, n_A\}_{B^{-1}}, n_B$$

$$M3 : A \rightarrow B\{n_B\}_{A^{-1}}$$

**Dimostrazione** Nel primo messaggio il server riceve dal client il certificato del client, dunque utilizzando la *message meaning rule*:

$$\frac{B \models \overset{K_{pubCA}}{\longmapsto} CA, B \triangleleft \{\overset{K_{pubA}}{\longmapsto} A\}_{CA^{-1}}}{B \models CA \mid\sim (\overset{K_{pubA}}{\longmapsto} A)}$$

Questo ci dimostra che il certificato è stato firmato dalla CA. Effettuando anche il controllo della validità il certificato tramite il CRL, possiamo assumere che il certificato del client sia fresco:

$$B \models \#(\overset{K_{pubA}}{\longmapsto} A)$$

Usando adesso la *nonce verification rule*:

$$\frac{B \models \#(\overset{K_{pubA}}{\longmapsto} A), CA \mid\sim \overset{K_{pubA}}{\longmapsto} A}{B \models CA \models \overset{K_{pubA}}{\longmapsto} A}$$

Ed infine usando la *jurisdiction rule*:

$$\frac{B \models CA \Rightarrow \overset{K_{pubA}}{\longmapsto} A, B \models CA \models \overset{K_{pubA}}{\longmapsto} A}{B \models \overset{K_{pubA}}{\longmapsto} A}$$

Nel secondo messaggio, analogamente al primo, viene effettuta la verifica del certificato del server:

$$\frac{A \models \overset{K_{pubCA}}{\longmapsto} CA, A \triangleleft \{\overset{K_{pubB}}{\longmapsto} B\}_{CA^{-1}}}{A \models CA \mid\sim (\overset{K_{pubB}}{\longmapsto} B)}$$

Possiamo quindi assumere che anche il certificato del server sia fresco:

$$A \models \#(\overset{K_{pubB}}{\longmapsto} B)$$

Usando adesso la *nonce verification rule*:

$$\frac{A \models \#(\overset{K_{pubB}}{\longmapsto} B), CA \mid\sim \overset{K_{pubB}}{\longmapsto} B}{A \models CA \models \overset{K_{pubB}}{\longmapsto} B}$$

Ed infine usando la *jurisdiction rule*:

$$\frac{A \models CA \Rightarrow \overset{K_{pubB}}{\longmapsto} B, A \models CA \models\!\!\overset{K_{pubB}}{\longmapsto} B}{A \models\!\!\overset{K_{pubB}}{\longmapsto} B}$$

Per quanto riguarda la seconda parte del secondo messaggio utilizzando ancora la *message meaning rule*:

$$\frac{A \models\!\!\overset{K_{pubB}}{\longmapsto} B, A \triangleleft \{\{A \overset{K_a}{\longleftrightarrow} B, A \overset{K_e}{\longleftrightarrow} B\}_{K_{ek}}, \{A \overset{K_{ek}}{\longleftrightarrow} B\}_{K_{pubA}}, IV, n_A\}_{B^{-1}}}{A \models B \mid\sim (\{A \overset{K_a}{\longleftrightarrow} B, A \overset{K_e}{\longleftrightarrow} B\}_{K_{ek}}, \{A \overset{K_{ek}}{\longleftrightarrow} B\}_{K_{pubA}}, IV, n_A)}$$

$$\frac{A \models \#(n_A)}{A \models \#(\{A \overset{K_a}{\longleftrightarrow} B, A \overset{K_e}{\longleftrightarrow} B\}_{K_{ek}}, \{A \overset{K_{ek}}{\longleftrightarrow} B\}_{K_{pubA}}, IV, n_A)}$$

Usando adesso la *nonce verification rule*:

$$\frac{A \models \#(\{A \overset{K_a}{\longleftrightarrow} B, A \overset{K_e}{\longleftrightarrow} B\}_{K_{ek}}, \{A \overset{K_{ek}}{\longleftrightarrow} B\}_{K_{pubA}}, IV, n_A), A \models B \mid\sim (\{A \overset{K_a}{\longleftrightarrow} B, A \overset{K_e}{\longleftrightarrow} B\}_{K_{ek}}, \{A \overset{K_{ek}}{\longleftrightarrow} B\}_{K_{pubA}}, IV, n_A)}{A \models B \models (\{A \overset{K_a}{\longleftrightarrow} B, A \overset{K_e}{\longleftrightarrow} B\}_{K_{ek}}, \{A \overset{K_{ek}}{\longleftrightarrow} B\}_{K_{pubA}}, IV, n_A)}$$

Ed infine usando la *jurisdiction rule*:

$$\frac{A \models B \models (\{A \overset{K_a}{\longleftrightarrow} B, A \overset{K_e}{\longleftrightarrow} B\}_{K_{ek}}, \{A \overset{K_{ek}}{\longleftrightarrow} B\}_{K_{pubA}}, IV, n_A), A \models B \Rightarrow (\{A \overset{K_a}{\longleftrightarrow} B, A \overset{K_e}{\longleftrightarrow} B\}_{K_{ek}}, \{A \overset{K_{ek}}{\longleftrightarrow} B\}_{K_{pubA}}, IV, n_A)}{A \models (\{A \overset{K_a}{\longleftrightarrow} B, A \overset{K_e}{\longleftrightarrow} B\}_{K_{ek}}, \{A \overset{K_{ek}}{\longleftrightarrow} B\}_{K_{pubA}}, IV, n_A)}$$

$$\frac{A \models (\{A \overset{K_a}{\longleftrightarrow} B, A \overset{K_e}{\longleftrightarrow} B\}_{K_{ek}}, \{A \overset{K_{ek}}{\longleftrightarrow} B\}_{K_{pubA}}, IV, n_A)}{A \models \{A \overset{K_a}{\longleftrightarrow} B, A \overset{K_e}{\longleftrightarrow} B\}_{K_{ek}}, A \models \{A \overset{K_{ek}}{\longleftrightarrow} B\}_{K_{pubA}}, A \models IV, A \models n_A}$$

A questo punto è possibile estrarre la chiave $K_{ek}$:

$$\frac{A \models\!\!\overset{K_{pubA}}{\longmapsto} A, A \triangleleft \{A \overset{K_{ek}}{\longleftrightarrow} B\}_{K_{pubA}}}{B \triangleleft A \overset{K_{ek}}{\longleftrightarrow} B}$$

Una volta ottenuta la chiave $K_{ek}$, procediamo con l'estrazione delle chiavi $K_a$ e $K_e$:

$$\frac{A \models A \overset{K_{ek}}{\longleftrightarrow} B, A \triangleleft \{A \overset{K_a}{\longleftrightarrow} B, A \overset{K_e}{\longleftrightarrow} B\}_{K_{ek}}}{A \models B \mid\sim (A \overset{K_a}{\longleftrightarrow} B, A \overset{K_e}{\longleftrightarrow} B)}$$

Avvalendoci ancora una volta della freschezza di $N_A$:

$$\frac{A \models \#(A \overset{K_a}{\longleftrightarrow} B, A \overset{K_e}{\longleftrightarrow} B), A \models B \mid\sim (A \overset{K_a}{\longleftrightarrow} B, A \overset{K_e}{\longleftrightarrow} B)}{A \models B \models (A \overset{K_a}{\longleftrightarrow} B, A \overset{K_e}{\longleftrightarrow} B)}$$

Infine, per mezzo della *jurisdiction rule*:

$$\frac{A \models B \models A \overset{K_a}{\longleftrightarrow} B, A \models B \Rightarrow A \overset{K_a}{\longleftrightarrow} B}{A \models A \overset{K_a}{\longleftrightarrow} B}$$

$$\frac{A \models B \models A \overset{K_e}{\longleftrightarrow} B, A \models B \Rightarrow A \overset{K_e}{\longleftrightarrow} B}{A \models A \overset{K_e}{\longleftrightarrow} B}$$

# 5 Seconda Fase

Dopo che la fase di Handshake è completata, client e server possono ora comunicare in maniera sicura ed è possibile sottomettere comandi da parte del client. I comandi a disposizione del client sono: *!download, !upload, !list, !help, !quit.*

- Attraverso il comando *!download*, specificando successivamente il nome del file, è possibile richiedere lo scaricamento di un file del server.
- Attraverso *!upload*, specificando il nome del file, è possibile caricare un file sul server.
- Attraverso il comando *!list*, si può richiedere al server la lista dei file disponibili, con le rispettive dimensioni.
- Attraverso il comando *!help*, si possono visualizzare i comandi disponibili.
- Attraverso il comando *!quit*, il client può disconnettersi dal server.

Per ogni operazione sopracitata, è stato definito un protocollo di comunicazione specifico. In modo da garantire la sicurezza nello scambio di dati.

**Protocollo di Download** Il protocollo definito per il download è il seguente:

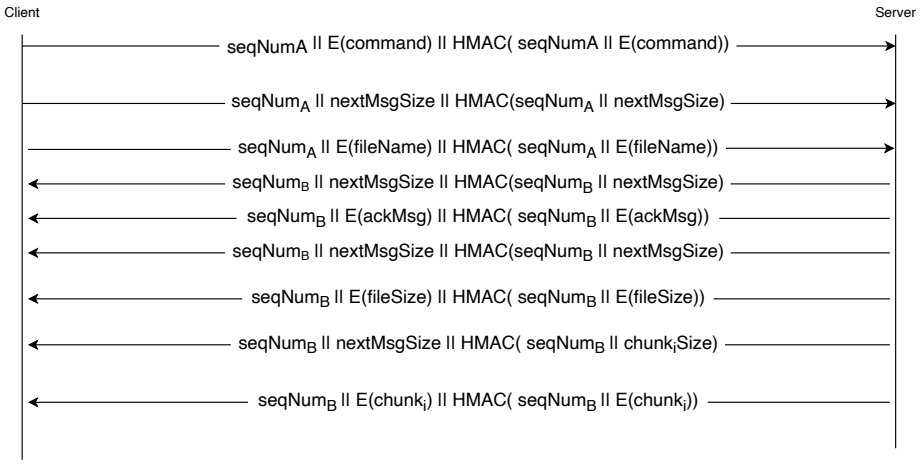


Figure 2: Protocollo di Download

In questo protocollo, il client inizialmente sottomette il comando di Download (criptato) al server, concatenando con il numero di sequenza e l'HMAC. (il comando ha grandezza fissata).
Successivamene, il client invia la grandezza del messaggio successivo (in chiaro, ma concatenando comunque il numero di sequenza e l'HMAC), per preparare

l'invio al server del nome del file desiderato.
Il server controlla innanzitutto se i caratteri inseriti sono "consentiti" (ad esempio se è presente un punto come primo carattere, che potrebbe comportare comportamenti indesiderati) e successivamente controlla se il file è presente nella cartella. Se il file non è presente, il server manda un messaggio di *NACK* al client per segnalare la non presenza del file. Se il file è presente, il server manda un messaggio di *ACK* al client per segnalare la presenza del file. In entrambi i casi, se l'invio dell'ACK/NACK fallisce il server procede alla terminazione della connessione con il client, andando a deallocare le strutture dedicate al client.
Procedendo nel caso di presenza del file, il server invia un messaggio contenente la grandezza del messaggio successivo e successivamente la grandezza del file da inviare. Ed infine, invia (sempre precedendo il messaggio che indica la grandezza del messaggio successivo) i vari chunk che suddividono il file.

**Protocollo di Upload** Il protocollo definito per l'upload è il seguente:

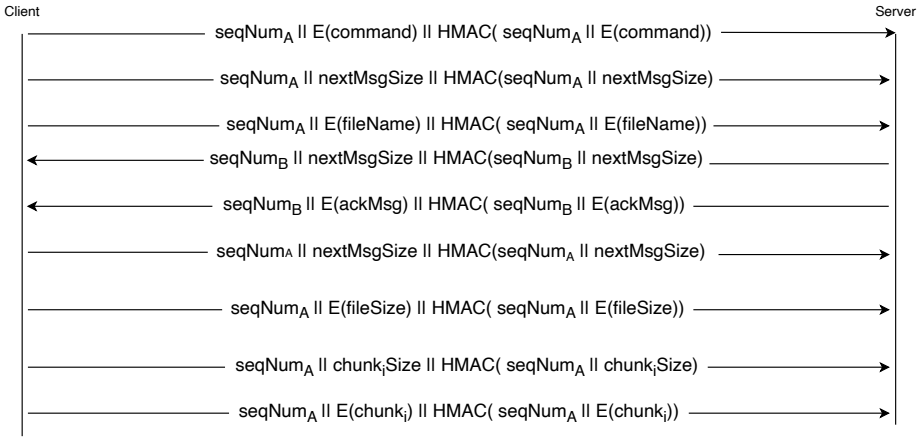


Figure 3: Protocollo di Upload

In questo protocollo, il client inizialmente sottomette il comando di Upload (criptato) al server, concatenando il numero di sequenza e l'HMAC.
Successivamente, il client invia la grandezza del messaggio successivo e il nome del file al server. Il server riceve il nome del file e controlla la correttezza dei caratteri che il client ha inviato, come nel caso del download. Dopo la ricezione dell'ACK il client può sottomettere la grandezza del file (inviando precedentemente la lunghezza del messaggio successivo). Ed infine procede all'invio della grandezza dei chunk e dei chunk stessi.

**Protocollo per Lista dei file** Il protocollo definito per la richiesta della lista dei file è il seguente:

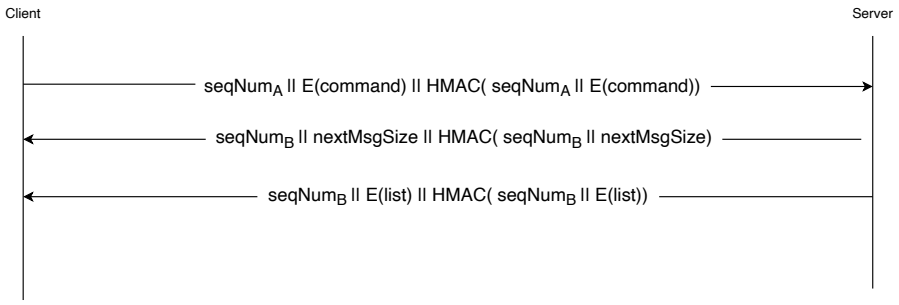


Figure 4: Protocollo per richiesta lista file

In tutti i casi, nel caso di errore di comunicazione da parte del client, quest'ultimo viene disconnesso dal server.