Relazione Progetto Applied Cryptography

Boschi Gianluca, Carli Francesco, Petri Paola

Introduzione

Nel seguente report, delineiamo in primo luogo le scelte progettuali ad alto livello, comprese degli schemi temporali riguardo lo scambio dei messaggi e il loro formato dettagliato con la motivazione delle scelte. Secondariamente sono riportate le scelte implementative, la gestione degli errori e le linee guida per l'uso.

1 Scelte progettuali

1.1 Autenticazione client-server e richiesta di chat

In questa sezione vedremo le fasi di autenticazione, scambio della chiave e richiesta di chat/attesa che avvengono tra client e server. L'autenticazione tra client e server insieme con lo scambio della chiave avvengono tramite lo schema **RSA effimero**.

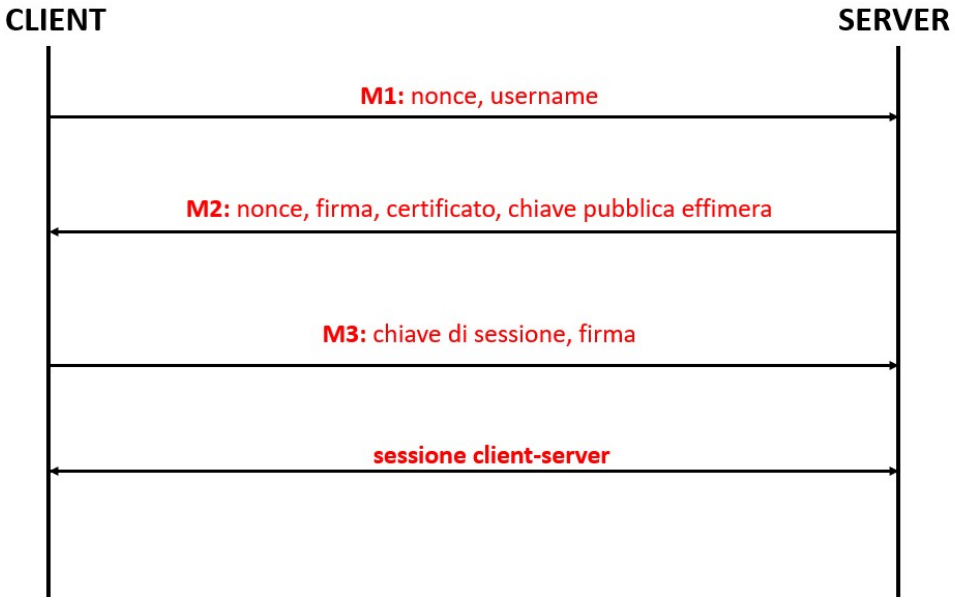

Figure 1: Schema dei messaggi tra client e server per la reciproca autenticazione

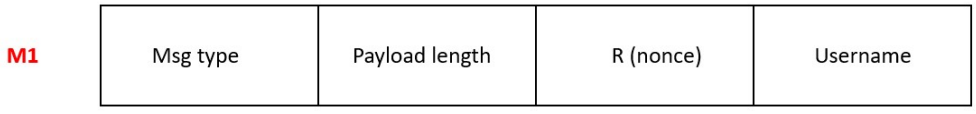

Figure 2: Formato del messaggio M1

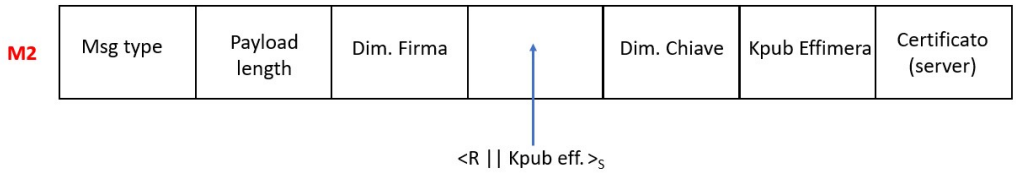

Figure 3: Formato del messaggio M2

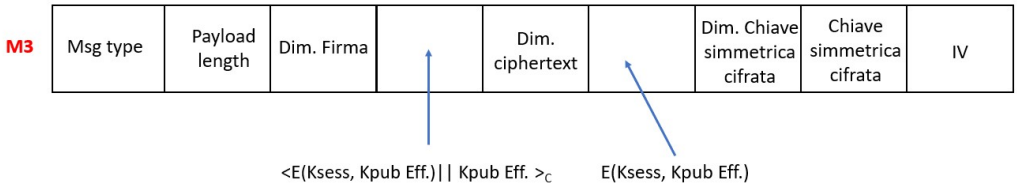

Figure 4: Formato del messaggio M3

- Nel messaggio M1 il client invia al server un messaggio contenente il suo *username*, in modo da richiedere la fase di login/autenticazione. All'interno di M1, c'è una quantità fresh, detta *nonce*, che gli servirà, leggendo la risposta del server, ad essere sicuro di non essere soggetto ad un attacco di tipo *replay*.

- Il server a questo punto genera una coppia di chiavi, pubblica e privata, dette *effimere* perchè utili soltanto allo scambio della chiave di sessione ed eliminate non appena raggiunto lo scopo. La chiave pubblica effimera servirà al client per inviare in maniera confidenziale nel messaggio successivo la chiave di sessione per parlare con il server. Nel messaggio M2 il server invia una quantità firmata (contenente il *nonce* ricevuto dal client concatenato alla *chiave pubblica effimera*) e il suo *certificato* in modo da provare la sua autenticità al client. La chiave pubblica effimera inoltre fa parte dei campi in chiaro del messaggio, in modo da permettere all'altro capo della conversazione di verificare la validità della firma. Il nonce invece viene inserito soltanto all'interno della firma, proprio perchè il client, avendolo generato, dovrebbe già possederlo e quindi una mancata verifica della firma potrebbe anche essere segno di errore nella quantità nonce e perciò di replay attack. Utilizzare la coppia di chiavi effimere garantisce il **Perfect Forward Secrecy**, ovvero la protezione delle sessioni passate in caso di compromissione della chiave privata *"long term"* del server.

- A questo punto il client deve generare la *chiave di sessione* per inviarla al server. Una volta generata, per garantire la sua confidenzialità, la cifrerà con la chiave pubblica effimera appena ricevuta dal server in modo che solo quest'ultimo possa decifrarla con la corrispondente chiave privata effimera. Per quanto riguarda la protezione da attacchi di tipo replay, la *chiave effimera* appena ricevuta dal server sarà inserita nel messaggio (concatenandola alla chiave di sessione cifrata e poi firmata) e fungerà da quantità fresh. Oltre alla firma, il messaggio conterrà la chiave di sessione cifrata ed alcuni parametri utili al server per decifrarla con il meccanismo della **Digital Envelope**. Tramite la firma il client prova la sua autenticità al server; quest'ultimo ha già le chiavi pubbliche di ogni client e può quindi verificare la validità della firma prendendo la chiave pubblica corrispondente all'username che ha ricevuto nel messaggio M1.

- Per garantire la proprietà Perfect Forward Secrecy, le chiavi effimere vengono eliminate subito dopo la decifratura della chiave di sessione, mentre la chiave di sessione verrà eliminata quando il client farà logout dal server.

Dopo il messaggio M3, il client e il server possono comunicare cifrando ogni messaggio tramite la chiave di sessione appena scambiata. I messaggi che seguono lo scambio della chiave di sessione sono tutti cifrati con **AES GCM 256** che permette a due utenti di inviarsi messaggi sia *cifrati* che

autenticati. Tra le quantità inviate nei messaggi troviamo gli AAD, ovvero le quantità non confidenziali ma di cui si desidera provare l'autenticità, il *testo cifrato*, il *tag* (che serve all'algoritmo di decifratura per verificare l'autenticità del messaggio) e l'IV.

Contatori client-server

Per proteggersi dai replay attack, ogni messaggio deve contenere una quantità fresh detta *nonce*, che provi a chi legge ogni messaggio che esso sia recente e non frutto di una replica. La nostra scelta è stata quella di usare un contatore per ogni coppia client-server, inizializzato a 0. Data l'asincronia della conversazione, ogni utente può inviare più di un messaggio alla volta, perciò abbiamo utilizzato un contatore per l'invio dei messaggi e uno per la ricezione, da entrambe le parti. Ogni utente utilizza il proprio contatore per l'invio mettendolo in ogni messaggio, avendo cura di gestirlo adeguatamente, e analogamente utilizza il contatore in ricezione per controllare i messaggi ricevuti.
In questo caso i due capi della conversazione sono client e server e quindi indicheremo con $Cont_{sc}$ il contatore dei messaggi inviati da server a client e con $Cont_{cs}$ il contatore dei messaggi inviati da client a server.

Secondo il nostro protocollo, questo è lo schema dei messaggi che seguono M3:

Figure 5: Schema dei messaggi scambiati tra client e server dopo l'autenticazione per inviare richieste di chat o alternativamente mettersi in attesa di richieste

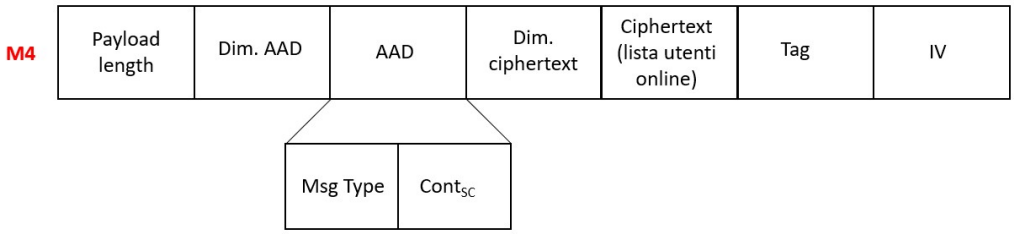

Figure 6: Formato del messaggio M4

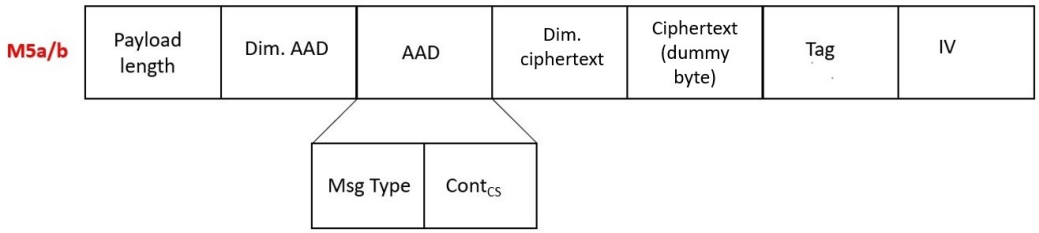

Figure 7: Formato del messaggio M5a/b

- Il messaggio M4 contiene la *lista degli utenti* cifrata tramite la chiave di sessione appena scambiata. Negli AAD viene inserito il *tipo del messaggio* e il *contatore* tra server e client.
- Il messaggio M5a/b contiene la *scelta* del client (escluso il logout): non appena letta la lista degli utenti online, può decidere di mettersi in attesa di una richiesta da parte di un altro client o di inviare ad uno dei client presenti nella lista una richiesta di parlare. La discriminazione della scelta avviene tramite il *msg type* che sarà, anche in questo messaggio, contenuto negli AAD insieme al *contatore* tra client e server. Nel testo cifrato troviamo un "*dummy byte*", che serve soltanto ai fini del funzionamento dell'algoritmo di cifratura e decifratura.

Richieste di chat

Il server mantiene una lista di utenti che hanno fatto il login, e dal momento in cui ha stabilito la chiave di sessione con un utente, il primo messaggio che invierà sarà questa lista, in modo da fargli prendere visione degli utenti con cui parlare. La lista però, conterrà solo i client già loggati che hanno scelto di mettersi in attesa di richieste di chat (M5b), mentre coloro che non hanno ancora effettuato una scelta o che hanno scelto di inviare richieste (M5a), vengono classificati dal server come "busy" e quindi non inseriti nella lista da inviare al client. Un utente può quindi inviare una richiesta di chat solo ad un utente che la sta attendendo.
Al momento della richiesta di chat, l'utente che la riceve ha un timeout per inviare la risposta; in caso positivo si procede con lo scambio della chiave, in caso negativo si distingue tra client 1 e client 2:

- Client 1: è l'utente che ha fatto la richiesta e riceverà nuovamente il messaggio M4 e potrà di nuovo scegliere l'azione da effettuare (richiesta, attesa o logout)
- Client 2: è l'utente che si era messo in attesa di richieste. In seguito ad un rifiuto o allo scadere del timeout, viene messo di nuovo nello stato di attesa.

A seconda dell'azione che sceglie di fare il client, cambiano i messaggi scambiati con il server. Illustriamo nello schema seguente entrambi gli scenari: uno dei due client (2) si mette in attesa di richieste e quindi verrà inserito nella lista degli utenti online. Dal momento in cui l'altro client (1) legge il nome dell'utente online nella lista, invia al server una richiesta per parlare con quel client.

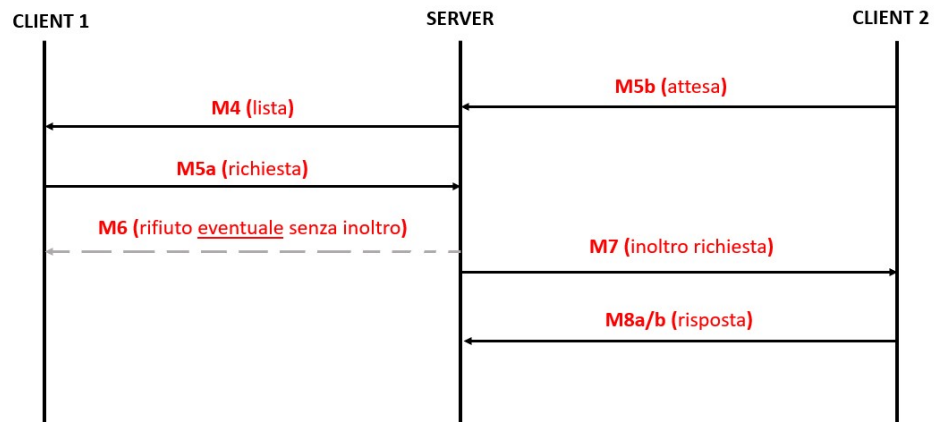

Figure 8: Schema dei messaggi scambiati tra client e server al momento della richiesta di chat

A seconda della risposta del client che riceve la richiesta, si profilano due situazioni:

- Risposta positiva (M8a): il server procede con l'inoltro dell'accettazione e delle chiave pubblica di Client 2 a Client 1.
- Risposta negativa (M8b): il server inoltra il messaggio di rifiuto a Client 1.

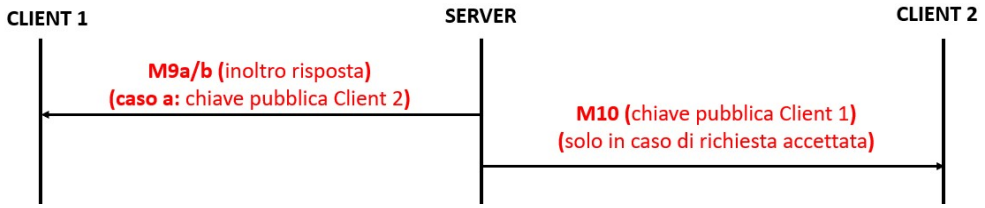

Figure 9: Invio della risposta alla richiesta e delle chiavi pubbliche in caso di richiesta accettata

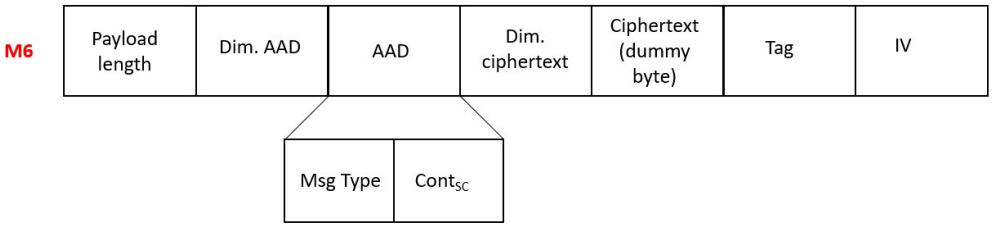

Figure 10: Formato del messaggio M6

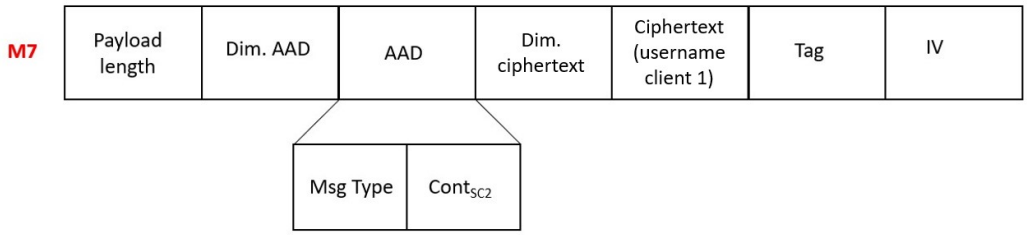

Figure 11: Formato del messaggio M7

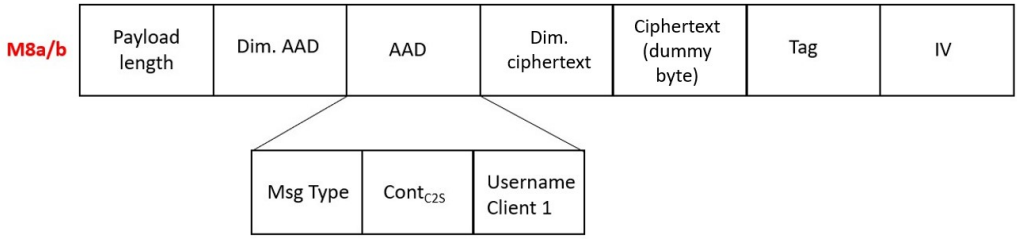

Figure 12: Formato del messaggio M8a/b

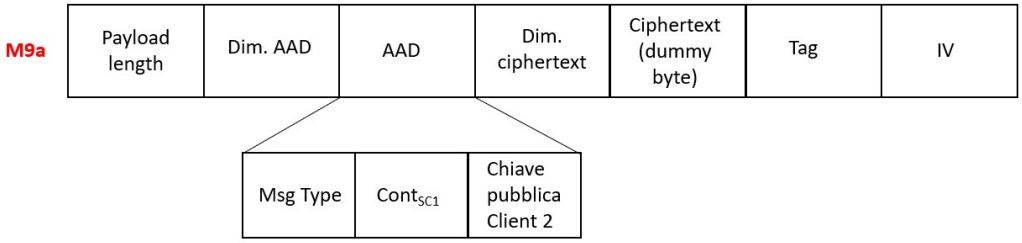

Figure 13: Formato del messaggio M9a

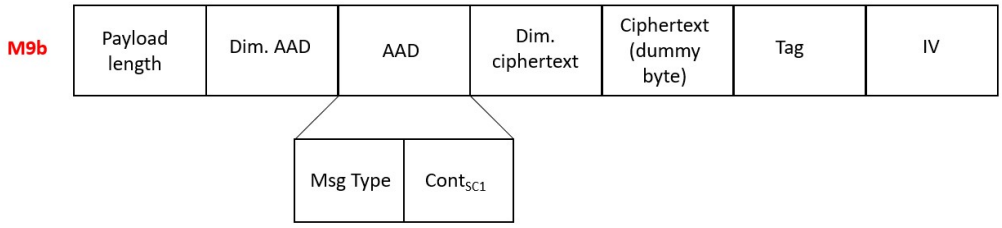

Figure 14: Formato del messaggio M9b

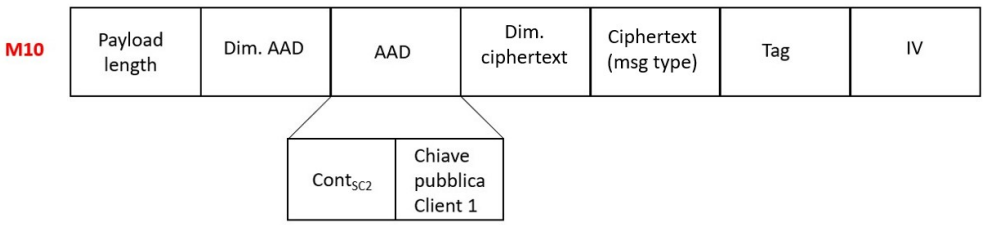

Figure 15: Formato del messaggio M10

- Il messaggio M6 viene inviato solo nel caso in cui l'utente con cui si vuole parlare non è nella lista degli utenti online, è occupato o per altri errori generici. Il messaggio indica un rifiuto della richiesta elaborato da parte del server senza inoltrarla all'altro client. Il *Msg type* presente negli AAD insieme al *contatore* dal server al client 1 indica il rifiuto di richiesta mentre nel ciphertext è presente solo un "*dummy byte*".

- Il messaggio M7 è l'*inoltro* della richiesta verso il client destinatario. In questo messaggio viene cifrato il *nome dell'utente* che ha fatto la richiesta mentre negli AAD troviamo nuovamente il *msg type* e il *contatore* da server a client 2.

- Il messaggio M8a/b contiene la risposta del client alla richiesta di chat, "a" nel caso in cui sia positiva, "b" nel caso in cui sia negativa. Nel ciphertext troviamo nuovamente il *dummy byte* e negli AAD vengono inseriti il *msg type*, il *contatore* da client 2 a server e il *nome* dell'utente che aveva fatto la richiesta.

- Il messaggio M9a viene inviato in caso di risposta *positiva* alla richiesta di chat (M8a). Esso conterrà negli AAD il *msg type* che permetterà al client 1 di discriminare una risposta positiva da quella negativa, il *contatore* da server a client 1 e la chiave pubblica del client 2, che gli servirà per gli step successivi. Nel ciphertext abbiamo inserito nuovamente il *dummy byte*.

- Il messaggio M9b è analogo a M9a, tranne che per il msg type in cui sarà indicato il *rifiuto* della richiesta e per l'assenza della chiave pubblica del client 2.

- Nel messaggio M10, inviato solo se preceduto dall'accettazione di richiesta, il server invia al client 2 la *chiave pubblica* del client 1 inserendola negli AAD insieme al *contatore*. Nel ciphertext viene cifrato il *msg type*, che indica l'invio della chiave pubblica e che quindi avviene solo in caso di richiesta accettata.

1.2 Key Establishment tra due client

Se la richiesta di chat viene accettata, ha inizio la fase in cui i due client devono autenticarsi reciprocamente per scambiare una chiave di sessione con cui iniziare la conversazione. A questo scopo si utilizza lo schema precedente di **RSA Effimero** per garantire la proprietà di Perfect Forward Secrecy. Durante questa fase, ogni messaggio che i due client si scambieranno, passerà attraverso il server, la cui funzione è quella dell'inoltro. Poichè ogni messaggio tra client e server deve essere sempre autenticato e confidenziale qualora ci siano campi che lo necessitino, essi continueranno ad utilizzare la chiave di sessione scambiata precedentemente, e quindi il cifrario **AES GCM 256** per continuare a garantire l'autenticità reciproca. I messaggi scambiati tra client in questa fase saranno inseriti interamente negli AAD, sia da client a server al momento dell'invio, sia da server a client per l'inoltro, in modo da garantirne l'autenticità e l'integrità. Nel ciphertext di questi messaggi sarà inserito soltanto il dummy byte: questo perché non c'è niente di confidenziale tra client e server essendo che quest'ultimo svolge soltanto l'attività di inoltro, ma la chiave di sessione precedentemente scambiata viene usata soltanto per provare l'autenticità reciproca.

Ricordando che ogni messaggio passa attraverso il server, dopo aver verificato la sua autenticità, esso impacchetterà un nuovo messaggio per il client destinatario inserendo tutto il contenuto a lui diretto negli AAD. Indichiamo i messaggi come 1.1, 1.2, 2.1 ecc.. dove 1.1 è il messaggio che va dal

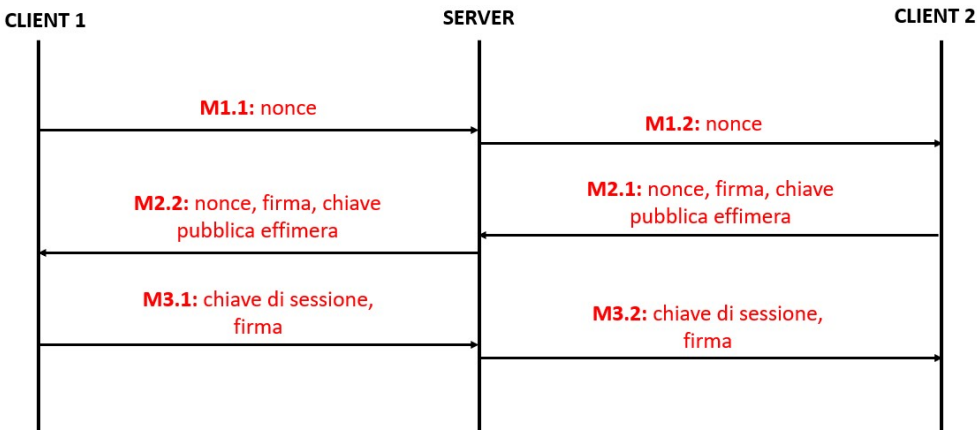

Figure 16: Schema dei messaggi tra due client e per la reciproca autenticazione e scambio della chiave di sessione

client 1 al server e 1.2 è il messaggio che il server inoltra all'altro client e così via. I campi contrassegnati in blu sono quelli destinati all'altro client, che quindi il server inoltrerà soltanto.

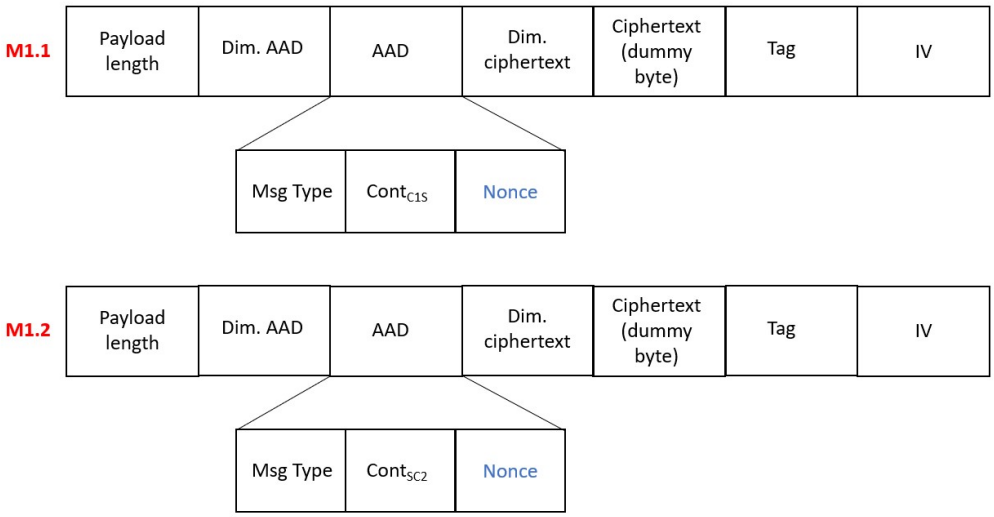

Figure 17: Formato dei messaggi M1.1 e M1.2

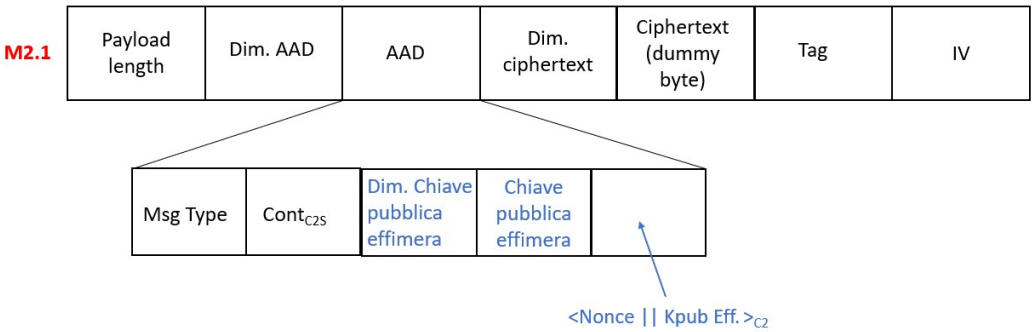

Figure 18: Formato del messaggio M2.1

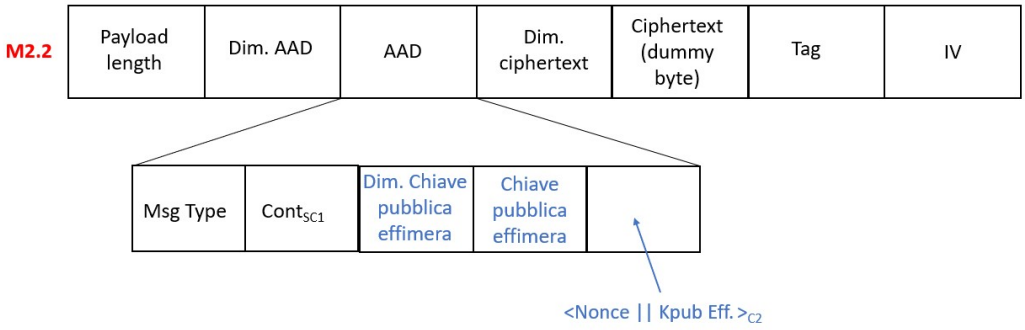

Figure 19: Formato del messaggio M2.2

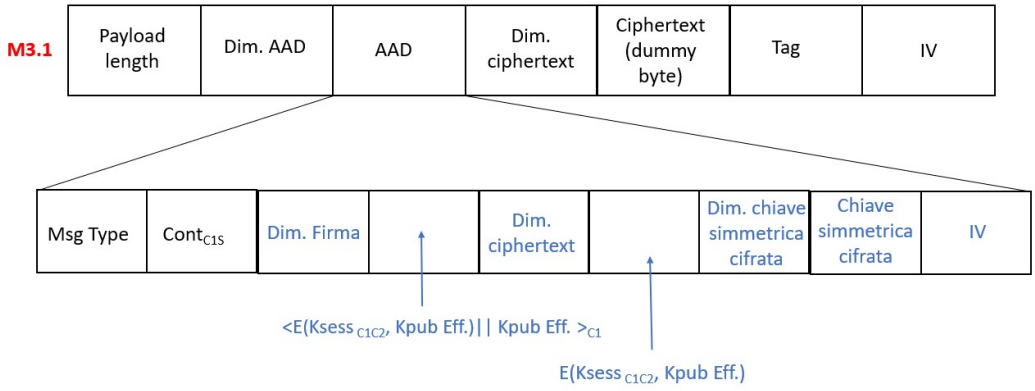

Figure 20: Formato del messaggio M3.1

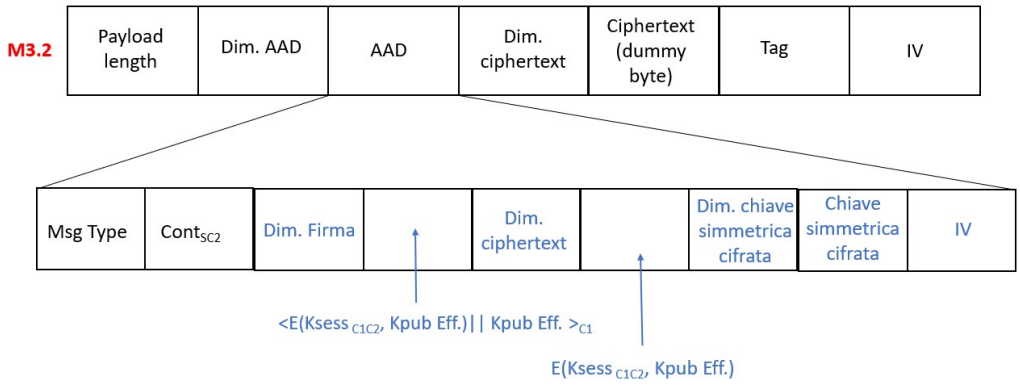

Figure 21: Formato del messaggio M3.2

Come già anticipato, lo schema dello scambio della chiave di sessione e autenticazione tra i due client, segue i principi di RSA effimero, perciò i campi inviati da client a client (inseriti negli AAD), saranno analoghi a quelli dei messaggi M1, M2, M3 (Sezione 1.1)

- Nei messaggi M1.1 e M1.2 il c'è una quantità fresh, detta *nonce*, con scopo analogo a quello del messaggio M1. In questo caso si omette l'username, in quanto l'altro client sa già con chi sta avviando una conversazione.

- Il client 2 (quello che ha ricevuto la richiesta) a questo punto genera la coppia di chiavi effimere, pubblica e privata. Nei messaggi M2.1 e M2.2 si invia una quantità firmata (contenente il *nonce* inviato prima dal client 1 concatenato alla *chiave pubblica effimera*). La chiave pubblica effimera inoltre fa parte dei campi del messaggio, in modo da permettere all'altro capo della conversazione di verificare la validità della firma e quindi l'autenticità del messaggio.

- A questo punto il client 1 deve generare la *chiave di sessione* per inviarla al client 2. Una volta generata, per garantire la sua confidenzialità, cifrerà la chiave di sessione con la chiave pubblica effimera appena ricevuta. Per quanto riguarda la protezione da attacchi di tipo replay, come nel messaggio M3, è garantita dall'inserimento della chiave pubblica effimera nel messaggio (concatenata alla chiave di sessione cifrata e poi firmata) che fungerà da quantità fresh. Oltre alla firma, il messaggio conterrà la chiave di sessione cifrata ed alcuni parametri utili al server per decifrarla con il meccanismo della **Digital Envelope**.

- Per garantire la proprietà Perfect Forward Secrecy, le chiavi effimere vengono eliminate subito dopo la decifratura della chiave di sessione, mentre la chiave di sessione verrà eliminata quando uno dei due client vorrà terminare la conversazione.

1.3 Sessione tra due client

I due client hanno adesso una chiave di sessione con cui iniziare la loro conversazione. Anche in questo caso utilizziamo il cifrario **AES GCM 256** con cui i client possono cifrare il messaggio vero e proprio della conversazione e mettere negli AAD le quantità da autenticare (msg type, contatori..). Come nella fase di scambio della chiave, tutti i messaggi tra client e server sono autenticati, quindi il pacchetto destinato all'altro client viene inserito nel messaggio verso il server tramite gli AAD. Il server verifica l'autenticità del messaggio e in caso, prepara un messaggio per il client destinatario inserendo il messaggio del mittente nuovamente negli AAD. Il ciphertext tra client e server contiene sempre il *dummy byte* perchè non ci sono quantità confidenziali. Negli AAD tra client e server troveremo il *contatore*, il *msg type* e tutto ciò che è destinato all'altro client. Per quanto riguarda quest'ultima quantità, non essendo il server in possesso della chiave di sessione tra i due client, essa è protetta sian in termini di confidenzialità in quanto non può decifrare il ciphertext, ma anche in termini di integrità, in quanto se cambiasse uno dei campi del messaggio, essendo essi inseriti negli AAD tra client e client, il client destinatario se ne accorgerebbe al momento della verifica del tag.

Contatori client-client

Per quanto riguarda la protezione da attacchi di tipo replay, si utilizzano contatori analoghi a quelli tra client e server, ovvero uno in ricezione e uno in invio per ognuno dei due client. Questi contatori sono mantenuti e gestiti esclusivamente dai due client. Nel messaggio tra client e client, più specificatamente negli AAD, infatti troveremo il contatore della "direzione" corrente, mentre negli AAD destinati al server, il contatore dell'ultimo messaggio da server a client, o viceversa, aumentato di una unità. I principi per l'utilizzo dei contatori sono gli stessi di 1.1.

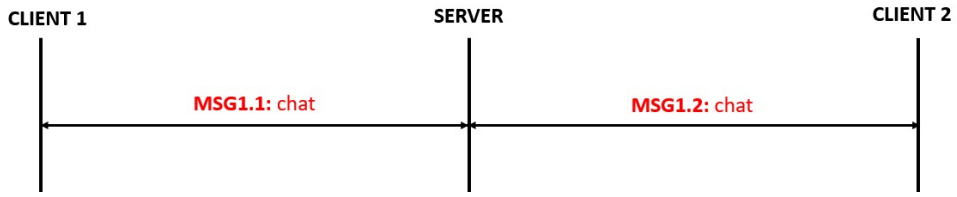

Figure 22: Schema dei messaggi tra due client durante la chat

Di seguito il formato dei messaggi durante la chat nei quali in blu sono riportati i campi destinati all'altro client.

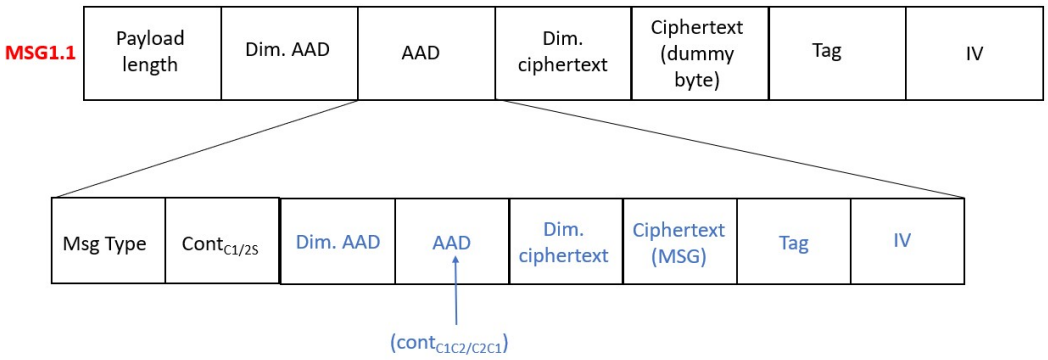

Figure 23: Formato del messaggio durante la chat MSG1.1

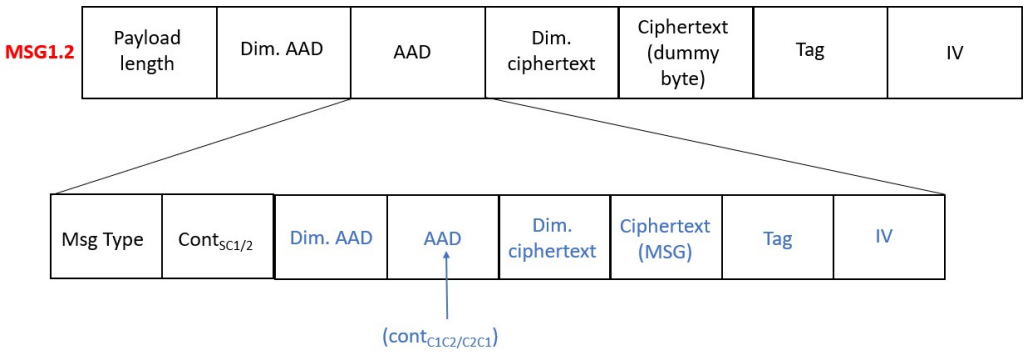

Figure 24: Formato del messaggio durante la chat MSG1.2

1.4 Gestione del logout

L'utente, fino a che non entra in una chat con un altro client può inviare un messaggio di *logout* al server con cui chiudiamo il programma client. Identifichiamo il messaggio di logout come M5c. Il logout può essere selezionato come azione da effettuare subito dopo la ricezione della lista degli utenti online:

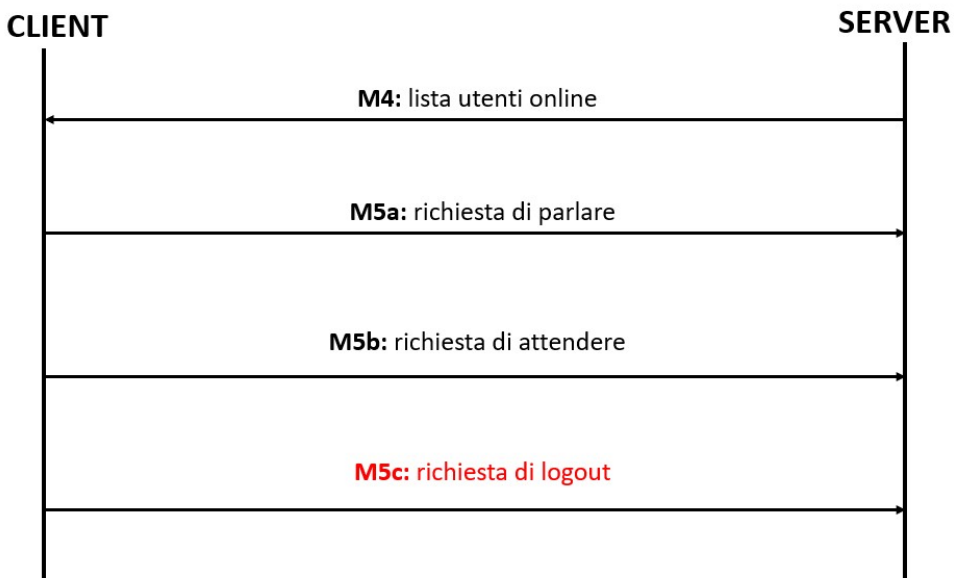

Figure 25: Invio del messaggio con richiesta di logout

Il formato del messaggio è analogo a quello di M5a/b e anche in questo caso il server riesce a discriminare una richiesta di attesa, chat o logout tramite l'identificazione del msg type. Con questo messaggio cade la connessione tra client e server.

2 Implementazione

2.1 Server

Abbiamo implementato un server TCP in modo da avere comunicazioni più affidabili rispetto a UDP per un'applicazione di questo tipo.

2.1.1 Gestione client

Il server ha un'architettura multithread, ciò significa che viene creato un thread per ogni connessione ricevuta da un client. Il thread gestirà fino al momento in cui desidera fare logout. Una volta avviato il programma server, si entra in un ciclo infinito nel quale si attendono connessioni: quando ne arriva una, viene lanciato il thread che gestirà con il client le comunicazioni tramite la socket con cui si è connesso.

2.1.2 Scambio messaggi

Lo scambio dei messaggi avviene attraverso la socket tra client e server per tutte le fasi del programma in quanto ogni thread si occupa di un client e non c'è nessuna fase in cui i client comunicano in maniera diretta senza passare dal server.
I messaggi vengono scritti e letti dalla socket tramite due funzioni *send* e *read*. In generale, la lettura del buffer dalla socket avviene grazie al formato fisso dei messaggi ad ogni fase: solitamente, il primo campo del messaggio è composto da 4 byte, che rappresenteranno in alcuni casi il *msg type* e in altri la *lunghezza del messaggio* restante da leggere. Ogni elemento inviato nel messaggio (es. chiavi, ciphertext..) viene preceduto dalla sua lunghezza in byte che, una volta convertita in intero, permette di fare la lettura dell'elemento in questione dalla socket specificando il numero di byte da leggere corretto. L'ultimo campo, in caso abbia una lunghezza variabile, non è preceduto dalla sua lunghezza in quanto ricavabile dalla lunghezza del messaggio intero letta tra i primi campi privandola di quanto letto fino a quel punto.

2.1.3 Lista degli utenti online

Il server, per poter gestire ogni client, ha bisogno di mantenere alcune informazioni all'interno di una lista di struct, dove ogni struct rappresenta un utente. Dato che il server è multithread, la gestione della lista, così come di altri elementi condivisi, avviene tramite l'uso delle lock. Tra le informazioni nella lista troviamo:

- nome
- file descriptor della socket
- una booleano che rappresenta lo stato (libero/occupato) dell'utente (*busy*)
- due contatori: uno in invio (da aggiornare ad ogni messaggio) e uno in ricezione (con cui effettuare i controlli)
- due booleani (*decision ready*) e (*decision result*) che servono a sincronizzare la decisione dell'utente per quanto riguarda una richiesta di chat.

Stato dell'utente
Importante tra questi campi è lo stato del client: esso viene settato ad occupato al momento del login, e a libero eventualmente solo dopo un messaggio M5b, ovvero di richiesta di attesa.
Dal momento in cui lo stato di un utente è settato a libero, il suo nome comparirà tra gli utenti online nella lista inviata ai client che si trovano di fronte alla scelta dell'azione. Lo stato di un utente che invece sceglie di inviare una richesta di chat, sarà lasciato occupato in modo da non poter ricevere richieste mentre sta scegliendo con chi parlare o se sta attendendo una risposta ad una richiesta già inviata. L'utente in attesa, viene messo sullo stato di occupato dal momento in cui un utente decide di inviargli una richiesta di chat a prescindere dall'esito della richiesta. Qualora decidesse di accettare, troveremmo il suo stato già correttamente impostato, altrimenti gli verrebbe ripristinato lo stato di libero e messo nuovamente in attesa.

2.1.4 Sessione

Come già anticipato, i messaggi durante la sessione tra i due client, continuano a sfruttare la chiave di sessione scambiata precedentemente tra client e server. Il server quando riceve un messaggio del client, si occupa di verificarne l'autenticità tramite il TAG e, in caso positivo, di prendere il contenuto degli AAD privato delle quantità a lui destinato e inviarlo all'altro client seguendo le stesse modalità; anche l'altro client ha necessità di verificare l'autenticità del server, perciò troverà il messaggio proveniente dal client mittente negli AAD. Potrà quindi sfruttare la chiave di sessione scambiata precedentemente con il server per verificare l'autenticità del messaggio e, in caso positivo, andare a leggere il messaggio cifrato con la chiave di sessione scambiata con l'altro client.

2.1.5 Logout

Il server gestisce il logout "definitivo" del client, ovvero quello in cui il programma client termina, avendo cura di eliminare il client dalla lista degli utenti in modo da permettere a quel client di loggarsi nuovamente in futuro senza generare errori, cancellando anche la chiave di sessione come previsto dallo schema della Perfect Forward Secrecy. Per quanto riguarda l'interruzione di una conversazione (che avviene inviando come messaggio nella conversazione la parola "logout"), il server, fungendo da inoltro, fa in modo che i due client, ricevano nuovamente il messaggio M4 con la lista degli utenti online e vengano messi di fronte alla scelta dell'azione da effettuare.
Non è possibile uscire in maniera brutale (segnale SIGINT tramite la sequenza di tasti CTRL+C) dal programma per un client, ma l'unico modo è la richiesta di logout menzionata sopra tramite la scelta dell'azione del logout.

2.2 Client

2.2.1 Login e autenticazione

I principi seguiti per svolgere il login e l'autenticazione sono quelli con i quali abbiamo costruito il formato dei messaggi, spiegato nella sezione precedente (1):

- Il server si autentica inviando il suo certificato. Il client leggerà il certificato e andando a controllare la sua validità (e il fatto che non sia stato revocato) potrà ricavare, in caso positivo, la chiave pubblica del server con la quale verificare la firma inviata nel messaggio M2. La validità della firma implicherà l'autenticità del server.

- Il client si autentica inviando una quantità firmata al server il quale, avendo le chiavi pubbliche associate ad ogni username, può verificarne la validità.

- L'autenticazione tra due client avviene come al punto precedente, ovvero inviandosi una quantità firmata, in quanto essi non sono in possesso di un certificato ma solo delle chiavi pubbliche l'uno dell'altro, inviategli dal server al momento dell'accettazione di una richesta.

2.2.2 Sessione

La gestione della sessione di chat tra i due client avviene secondo le modalità descritte nel formato dei messaggi; il vero destinatario è il client con cui si sta parlando, ma i messaggi vengono scambiati effettivamente tra client e server o viceversa e inseriti ogni volta negli AAD in modo da mantenere la sessione autenticata. Anche i messaggi tra client hanno i loro AAD, contenenti i contatori con scopo già delineato nella sezione precedente.
Per gestire l'invio e la ricezione dei messaggi e aggiornare i contatori corretti, ci siamo serviti di due thread aggiuntivi oltre a quello del programma client, uno per la gestione dei messaggi da inviare e uno per la gestione dei messaggi da ricevere. Questi due thread vengono avviati non appena prima dell'inizio della sessione di chat tra i due client e cancellati al suo termine. Entrambi i thread necessitano dei due contatori riguardanti la conversazione con il server, uno da inviare e aggiornare e uno per i controlli, delle chiavi di sessione sia con il client con cui si parla, sia con il server e del file descriptor della socket con cui stiamo comunicando col server. I contatori relativi alla conversazione tra i due client saranno inizializzati al momento dell'avvio dei thread.

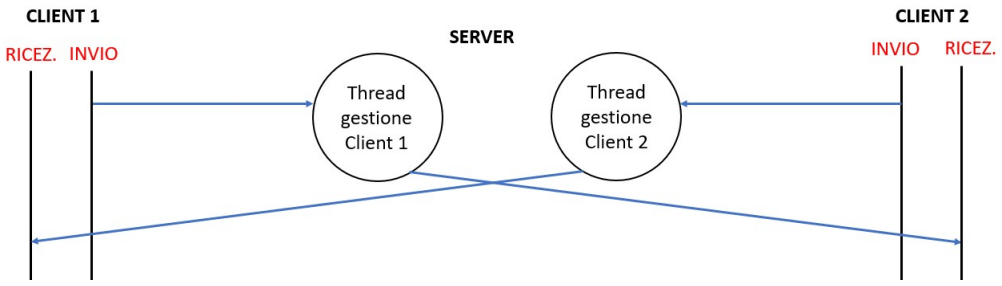

Figure 26: Gestione dei thread durante la sessione di chat tra client

2.2.3 Fine della sessione tra client

Dal momento in cui un client invia come messaggio durante la sessione di chat la parola "logout", si procede con la chiusura della sessione tra i due client. Il thread che gestisce l'invio dei messaggi controlla ogni volta che il messaggio sia uguale o meno a "logout" e, se il controllo ha esito positivo, il thread viene terminato non appena inviato il messaggio. Prima di terminare, invia un segnale SIGUSR1 all'altro thread (che si occupa della ricezione dei messaggi) in modo da segnalargli la fine della conversazione e fare in modo che termini.
Il server riceve il messaggio di fine sessione e può riconoscerlo nonostante non sia in grado di decifrare il messaggio destinato all'altro client grazie al msg type, che avrà un valore diverso da quello della conversazione standard. Inoltrerà poi il messaggio all'altro client in modo che i thread del client che non ha chiesto la disconnessione dalla conversazione vengano terminati. Il comportamento di questi ultimi è analogo e contrario a quello dei thread del client che ha richiesto il logout, ovvero il thread in ricezione invia il segnale all'altro thread e successivamente termina.
Il server si occupa di inviare il messaggio M4 contenente la lista degli utenti online e mettere i due client di fronte alla scelta dell'azione da effettuare.

3 Gestione errori

Gli errori vengono gestiti tramite alcune macro. Il server gestisce eventuali errori di comunicazione con i client rimanendo attivo e consistente ma chiudendo il thread del client avendo cura di cancellare l'utente dalla lista e cancellando eventuali chiavi di sessione già generate. Lato client, se avviene un errore inatteso, tramite le macro viene terminato il client stesso e il server rimane in uno stato consistente segnalando l'errore in questione.

4 Linee guida per l'uso

Per l'utilizzo del programma si lanci prima `main_server.cpp` e successivamente un client tramite `client_test.cpp` (la compilazione deve avvenire tramite l'inclusione delle librerie `lpthread` e `lcrypto`).
Se si desidera testare una conversazione tra i due client, si avvii un secondo client e si metta uno dei due in attesa al momento della scelta dell'azione da effettuare, successivamente si faccia inviare all'altro la richiesta di chat verso il client in attesa.
I nomi utenti con cui si può lanciare i client sono "paola", "giallu" e "france" e le password per l'utilizzo delle chiavi private sono i nomi degli utenti stessi.