
**Università degli Studi di Pisa**

**Dipartimento di Ingegneria dell'Informazione**

Corso di Laurea Magistrale in Cybersecurity

# Applied Cryptography: Online Messaging Service

*Studenti*:

Alessandro Niccolini

Luca Canuzzi

Andrea Kalaj

Anno Accademico 2020-2021

# Indice

# Introduzione

Il progetto ha richiesto l'implementazione di un applicativo Client-Server, che permette alle parti di scambiare messaggi di chat attraverso un canale sicuro ed autenticato. Gli utenti sono già registrati sul server tramite l'uso delle chiavi pubbliche. L'obiettivo è quello di dare la possibilità ai vari client di comunicare in modo sicuro su un canale dedicato tramite la creazione di una chiave di sessione. Prima tra i client ed il server, in modo da permettere ai client di inviare richieste al server e successivamente tra client e client in modo da dedicare un canale interamente dedicato alla chat. Tutto tramite l'ausilio di nonce e counter, i primi come prova di freschezza e i secondi per evitare la possibilità di replay da parte di un attaccante.

Tramite l'utilizzo del linguaggio $C++$ e la libreria $OpenSSL$ per gli algoritmi di crittografia e l'uso dei certificati. Di seguito sono riportate le scelte progettuali, gli schemi di scambio di messaggi ed il formato dettagliato di questi ultimi.

# 1 Scelte Progettuali

## 1.1 Protezione da attacchi di tipo replay

Per proteggere ogni fase di comunicazione, client-server e client-client, è stato scelto di difendere ogni messaggio scambiato con l'inserimento in ciascuno di questi un counter. Quindi, è stato scelto di utilizzare un contatore, inizializzato a zero, per ogni coppia client-server e client-client in invio e ricezione.
È stato scelto di usare un singolo counter per ciascuna coppia di attori, quindi ad esempio tra il client Alice e il Server c'è solamente un counter. Nel caso specifico per evitare valori negativi, visto che per numerare i messaggi scambiati si parte da zero, è stato scelto di usare il tipo "*unsigned int*" (con $n$ bit, si ha valori da 0 a $2^n - 1$).

## 1.2 Lista opt-message

Per la gestione dell'applicativo e della chat è stato scelto di usufruire di OPT CODE ovvero dei codici di opzione che vengono inviati come messaggi tra client e server, utili per gestire le fasi della chat.

- chat: il codice viene inviato dal client mittente al server e poi inoltrata al client destinatario, quando il mittente vuole iniziare una chat con il destinatario.
- yeschat: questo codice viene inviato dal client destinatario in caso di risposta positiva alla richiesta di chat.
- nochat: questo codice viene inviato dal client destinatario in caso di risposta negativa alla richiesta di chat.
- chat_quit: questo codice viene inviato quando uno tra i client impegnati nella chat ha intenzione di smettere di chattare e rendersi disponibile per una nuova sessione di chat.
- wait: questo codice viene inviato dal client quando ha intenzione di restare in attesa di chattare.
- stop: questo codice viene inviato quando il client vuole smettere di attendere nella chat.

- closed: questo codice viene utilizzato internamente dal server per indicare la chiusura della sessione con il client.

## 1.3 Implementazione dell'IV

All'interno di tutti i messaggi scambiati c'è la presenza del parametro IV utilizzando la funzione RAND della libreria OpenSSL. Questo parametro viene creato nuovo per ciascun messaggio allo scopo di evitare che qualche informazione utile per scoprire la chiave di sessione vada a finire nelle mani dell'eventuale attaccante.

## 1.4 Fase di Autenticazione Client server

Nella fase di autenticazione tra Client e Server è stato utilizzato l'algoritmo Diffie-Hellman Key Exchange (DHKE), fornito all'interno della libreria Openssl. Permettendo così di realizzare un protocollo per assicurare la corretta creazione di chiavi effimere, e si conclude con la creazione di una chiave di sessione Kcs.

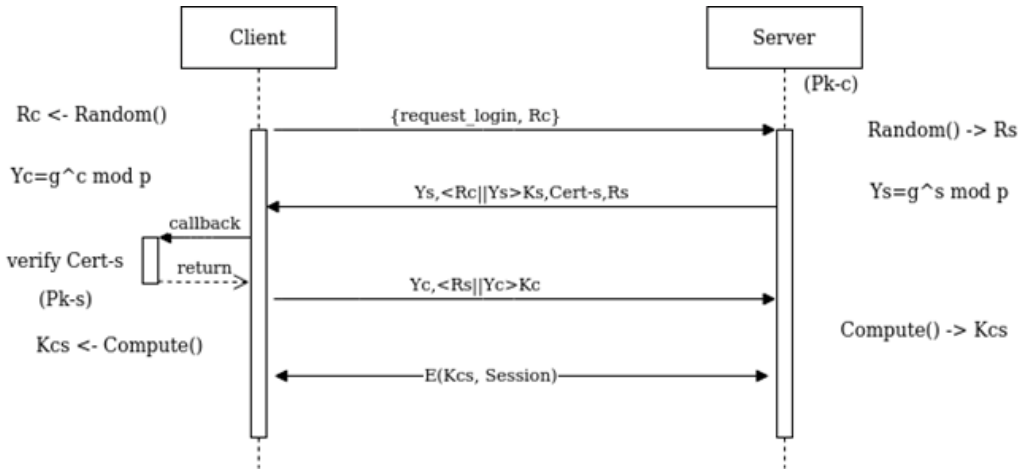
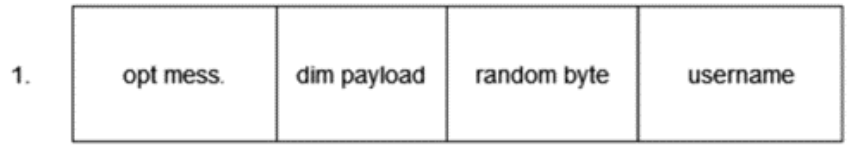


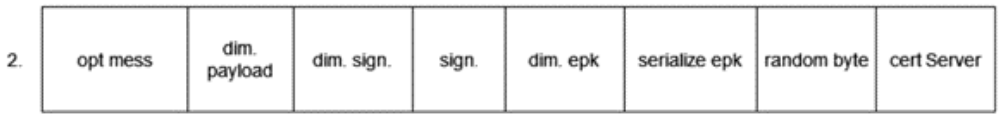

Figura 1.1: Illustrazione algoritmo Diffie-Hellman Key Exchange

## 1.5 Formato dei Messaggi client server

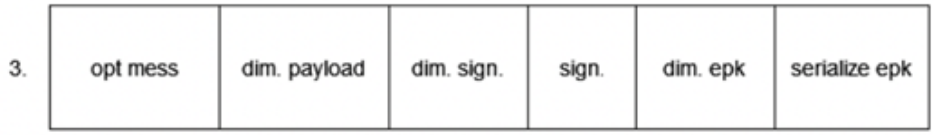
Di seguito è riportato il formato dei messaggi che si scambiano il client ed il server per creare un segreto condiviso e assicurarsi che nella comunicazione dei successivi messaggi un attaccante non possa intromettersi fingendosi uno agli occhi dell'altro. Nel messaggio 1) il client provvede alla creazione di un nonce random ($R_c$), che servirà come dimostrazione di "freschezza", lo trasforma in byte e lo spedisce al server insieme al proprio username. L'opt message viene utilizzato per identificare in quale fase dell'autenticazione ci troviamo, di conseguenza viene settato il formato del messaggio per la relativa fase.




Nel messaggio 2) il server calcola il proprio nonce ($R_s$) e la chiave effimera ($Y_s$). Dopodiché, firma la coppia di valori $R_c$ (il nonce mandato precedentemente dal client) ed $Y_s$ utilizzando la chiave privata del server che solo lui conosce. Inoltre, viene allegato il certificato del Server al messaggio.




Nel messaggio 3) il client crea la propria chiave effimera $Y_c$ e firma la coppia di valori $R_s$ (nonce mandato dal server nell'istante precedente) e $Y_c$.



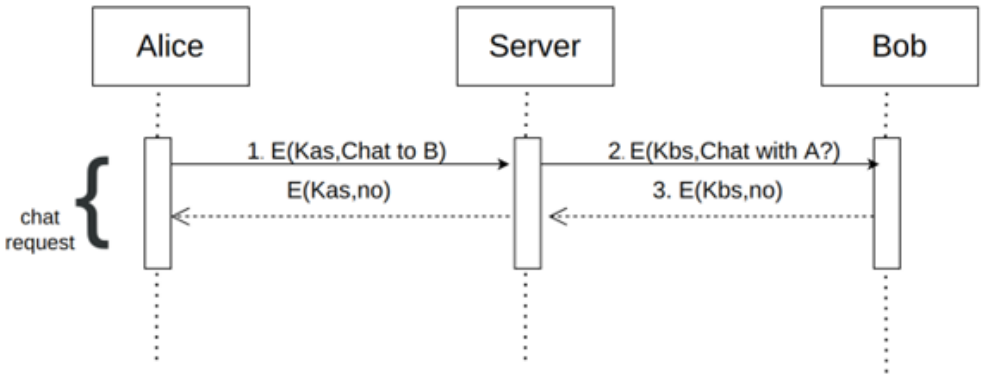

A questo punto ciascuna parte avrà le stesse informazioni, in altre parole la coppia di chiavi effimere appena creata sarà posseduta da entrambe le parti. Permettendo così, la creazione della chiave di sessione Client-Server che servirà nelle prossime interazioni e verrà cancellata solamente nel momento in cui il client decida di effettuare il logout e quindi terminare la sessione con il server.

## 1.6 Fase di richiesta Chat

Il server mantiene una lista di utenti che hanno fatto il login e che si rendono disponibili alla chat. Ogni qual volta un utente si connette oppure lo richiede esplicitamente il server invierà la lista aggiornata dei client che sono disponibili a chattare. Il client si deve specificatamente dichiarare "libero di chattare" per essere inserito in questa lista. Se un utente invia la richiesta di chat oppure ha già ricevuto una richiesta di chat viene rimosso da questa lista. Un utente quindi può inviare una richiesta di chat solo ad un possibile destinatario che si trova in attesa di chat.
L'autenticazione client-client avverrà solo nel caso in cui c'è una risposta positiva alla richiesta di chat.

## 1.7 Fase di richiesta Chat e risposta negativa del destinatario

In caso di risposta negativa del client destinatario, il server si occupa di inoltrare la risposta al client mittente della richiesta.




## 1.8 Fase di richiesta Chat e risposta affermativa del destinatario

Nell'immagine 1.2, è riportato lo schema ad alto livello dello scambio di messaggi che avviene per la richiesta di chat con risposta positiva tra client mittente impersonato da Alice, Server e client destinatario impersonato da Bob.

## 1.9 Messaggi scambiati

Di seguito è riportato il formato dei messaggi che si scambiano il client mittente, il server e il client destinatario per inizializzare la sessione di chat. I client e il server comunicano utilizzando la chiave di sessione precedentemente stabilita e per mezzo cifratura simmetrica autenticata, utilizzando AES GCM 256. Viene dato per scontato che il client ha già richiesto la lista degli utenti interessati a chattare. In caso di risposta negativa alla richiesta di chat, il server si preoccuperà di rendere nuovamente disponibile, ad una nuova chat, il client destinatario e di conseguenza, il client mittente potrà cercare un nuovo interlocutore.
In caso di risposta positiva da parte del destinatario, i due client, devono autenticarsi reciprocamente per stabilire una chiave di sessione ed instaurare un canale sicuro sul quale comunicare. Viene utilizzato lo schema DH effimero per garantire la perfect forward secrecy. I messaggi scambiati tra i client

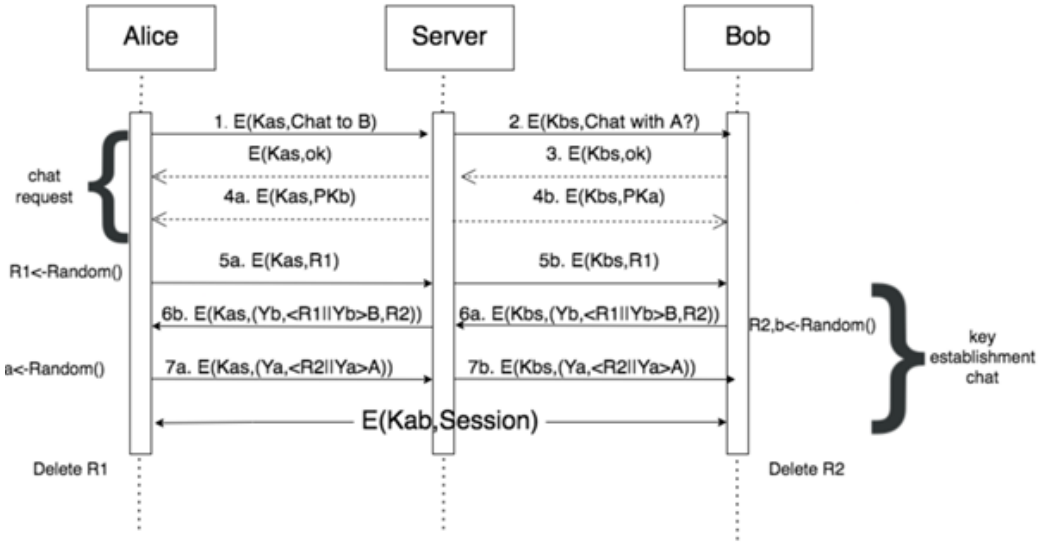
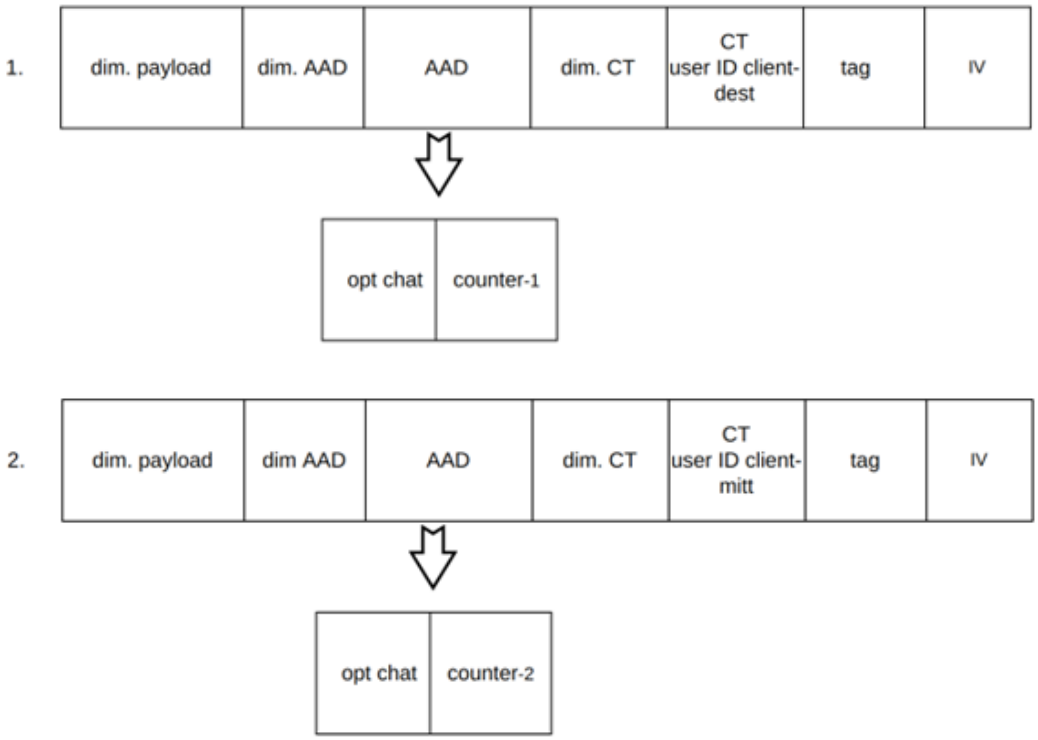



Figura 1.2: Scambio di messaggi risposta di chat affermativa

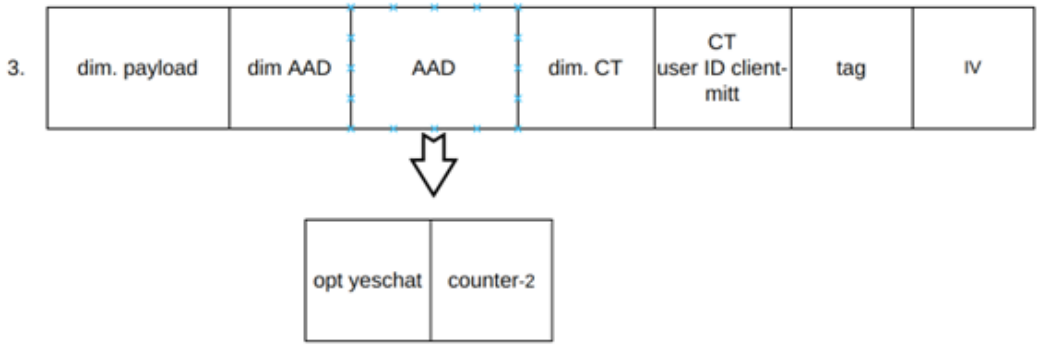
passano attraverso il server che si occupa del solo inoltro. I client scambiano i messaggi sempre in maniera autenticata tramite la chiave di sessione stabilita in precedenza. I messaggi scambiati tra i client verranno inseriti in AAD, in tutte le fasi dall'invio client-server all'inoltro server-client, così da garantire la loro autenticità e integrità. Nel testo cifrato di questi messaggi viene inserito un contenuto fittizio al solo fine di completezza, in questa fase client e server non devono scambiarsi alcun segreto.

Nel messaggio 1) Il mittente, Alice, invia al server la richiesta di chat al server, chiedendo di poter chattare con Bob. l'AAD conterrà il codice operativo che indica la richiesta di chat "chat" e il counter tra mittente e server, $counter - 1$. Nel testo cifrato è presente il nome del destinatario.

Nel messaggio 2) Il server inoltrerà al destinatario, Bob, la richiesta di chat. In questo caso AAD conterrà sempre il codice che indica la richiesta di chat e il counter tra server e destinatario, $counter - 2$. Il testo cifrato conterrà il nome del mittente.

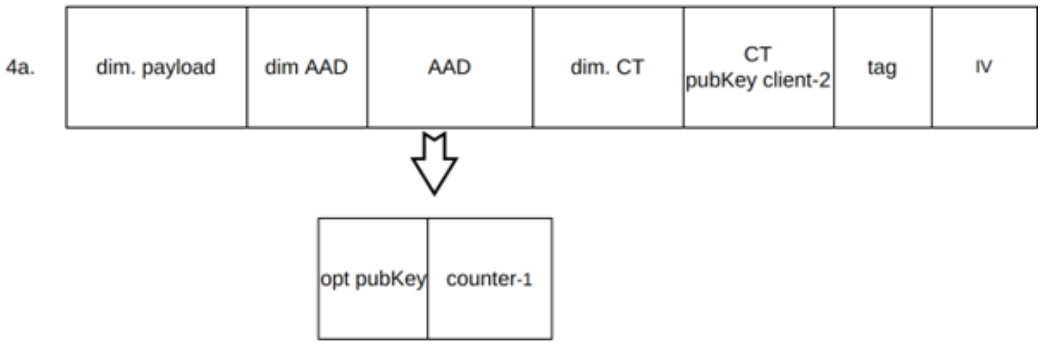


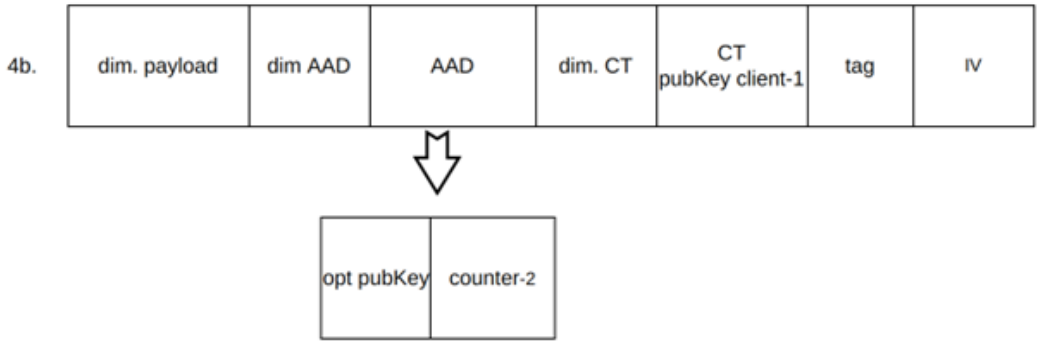

Nel messaggio 3) Il destinatario risponde con risposta positiva inviando al server il messaggio contenente in *AAD* l'opt code "*yeschat*" ed il counter tra server-destinatario, $counter - 2$, per confermare che il messaggio di risposta è "fresco".



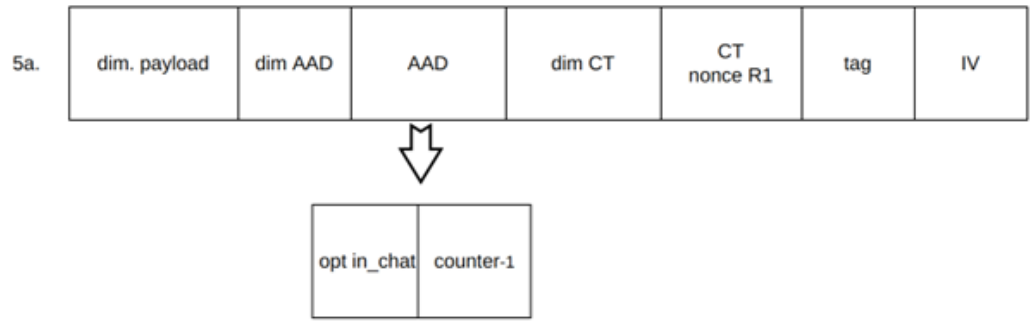

Nel messaggio 4*a*) e 4*b*) Il server invia rispettivamente la chiave pubblica del destinatario ($client - 2$) al mittente e la chiave pubblica del mittente al destinatario ($client - 1$). Anche gli ADD conterranno l'opt pubKey per indicare

ai client la tipologia di messaggio ricevuto e conterranno rispettivamente il $counter - 1$ nel messaggio $4a$ e $counter - 2$ nel messaggio $4b$.

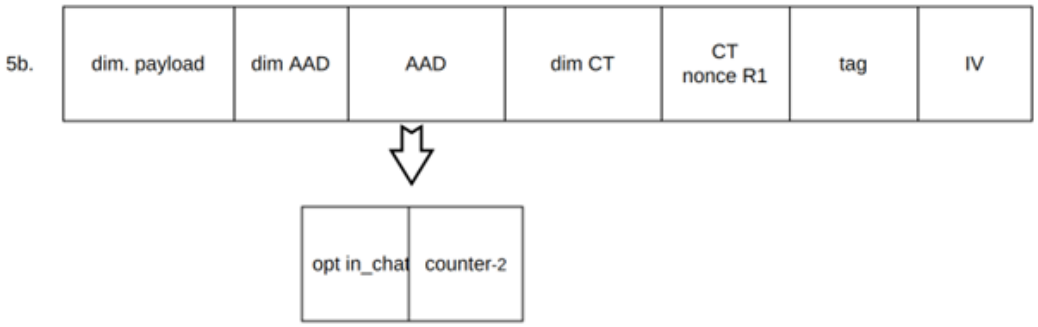


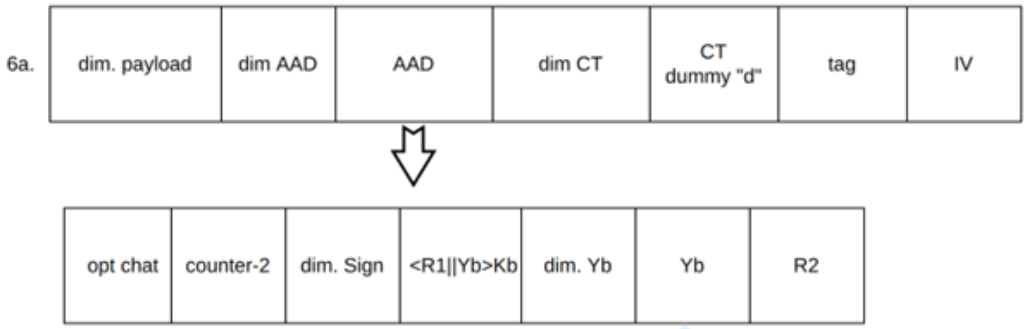

Nel messaggio $5a$) questo messaggio dà inizio alla creazione della chiave di sessione client-client. Precisamente, questo messaggio descrive l'invio del random nonce, $R_1$, da parte del client-1 (mittente). In $AAD$ viene mandato l'opt che descrive che stiamo attualmente nella fase di chat "*in_chat*" ed il counter client-server 1.



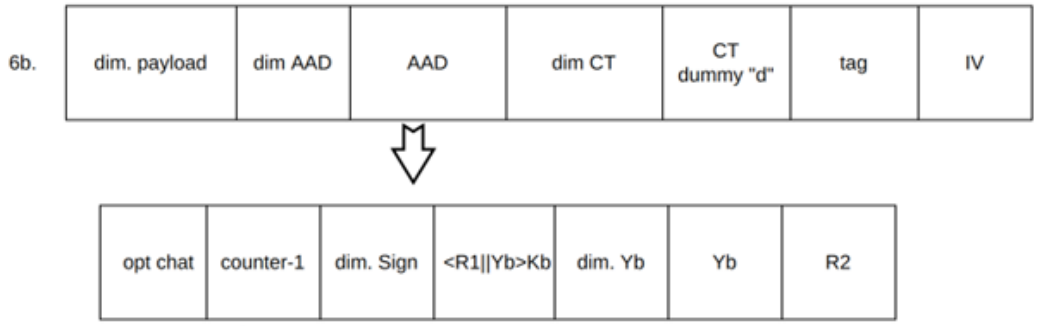
Nel messaggio 5*b*) viene descritto il messaggio utilizzato dal server per inoltrare il nonce random $R1$, generato dal client-1, al client destinatario. in $AAD$ come prima, troviamo lo stesso opt, ma questa volta il counter client-server 2.



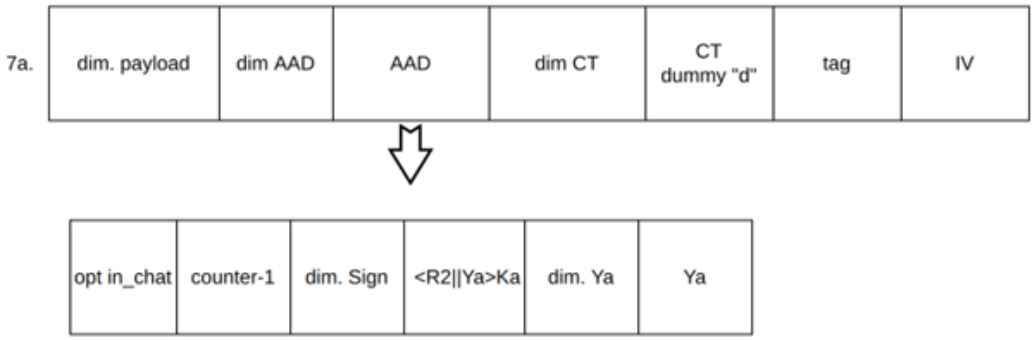

Il messaggio 6*a*) rappresenta il messaggio che viene inviato dal client destinatario al server, in risposta al client mittente. Il messaggio in $AAD$ riporta il nonce $R_1$ concatenato alla chiave effimera firmate, per garantire freschezza, la chiave effimera e il nonce $R_2$. Nel testo cifrato viene inserito un contenuto fittizio il carattere "*d*" solo per completezza.



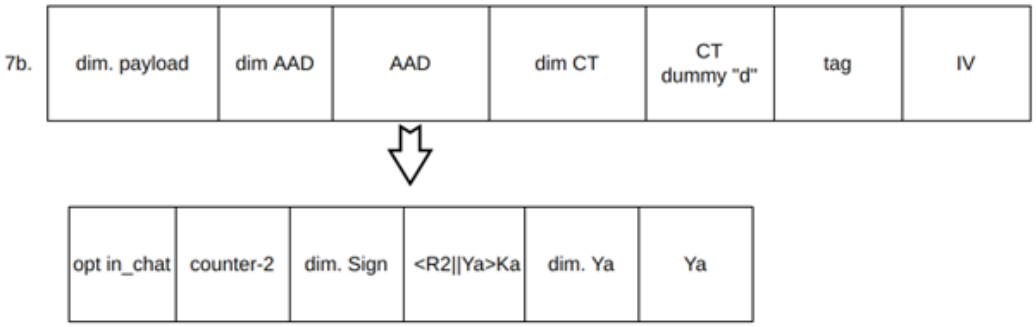

Il messaggio 6*b*) rappresenta il messaggio che viene inoltrato dal server al client mittente. Contiene le stesse informazioni a meno del cambio di counter.




Il messaggio 7$a$) rappresenta il messaggio che viene inviato dal client mittente al server. Il messaggio in $AAD$ riporta la concatenazione del nonce $R_2$ e chiave effimera del mittente firmata e la chiave effimera mittente. Nel testo cifrato viene inserito un contenuto fittizio il carattere "$d$" solo per completezza.



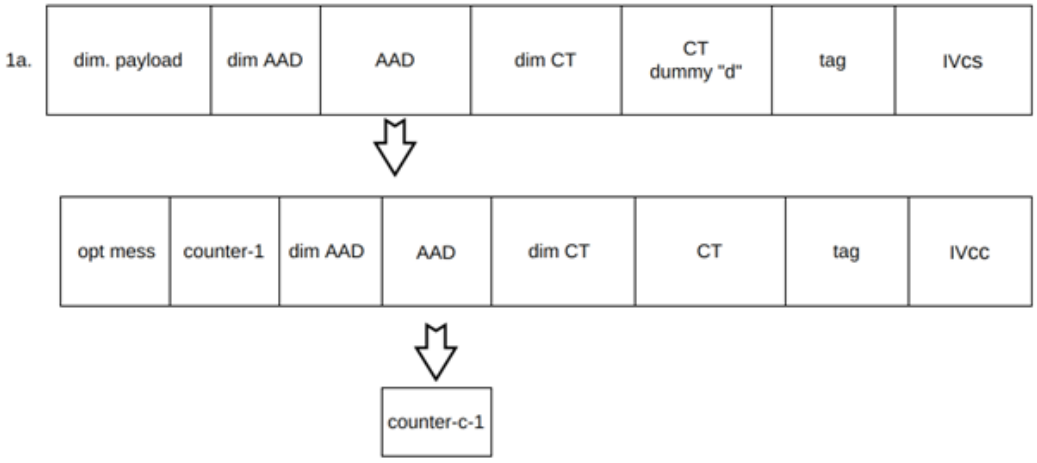

Il messaggio 7$b$) raffigura il messaggio inoltrato dal server al client destinatario della richiesta di chat. Contiene le stesse informazioni viste in precedenza ma viene cambiato il counter server-client2.

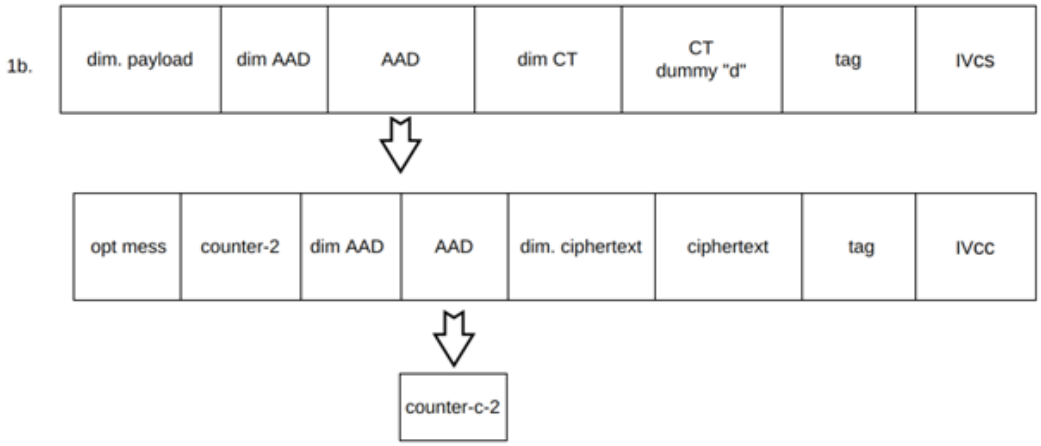


## 1.10 Instaurazione chiavi tra i due client

I due client hanno adesso stabilito la chiave di sessione utile allo scambio di messaggi confidenziali tramite il cifrario *AES GCM* 256. Il server agirà solamente da router di messaggi autenticati, le informazioni importanti scambiate tra i client verranno inserite in *AAD*. Il server verifica l'autenticità del messaggio ricevuto e prepara un nuovo messaggio per il client destinatario contenente le stesse informazioni in *AAD*. Come visto in precedenza non vi è alcun testo cifrato importante tra client e server ma solamente una stringa fittizia di un carattere solo per completezza dell'algoritmo di cifratura e decifratura. Il server, onesto ma curioso, non conosce la chiave di sessione tra i client quindi non sarà in grado di violare l'integrità e la confidenzialità dei messaggi, anche un eventuale tentativo di replay-attack verrebbe sventato dal counter client-client.

### Sessione tra i due client






## 1.11 Gestione chiusura chat

La chiusura da parte di uno dei due client avviene tramite il segnale SIGTSTP (ctrl + z) digitato dal $client-1$ durante la chat, come conseguenza il $client-1$ che ha catturato il segnale invia un messaggio al server contenente l'opt "*chat_quit*". Questo per poter correttamente informare il server delle intenzioni dell'utente. Il server a questo punto informa il $client-2$ con cui stava conversando colui che ha mandato la richiesta ($client-1$). Il messaggio che arriva al $client-2$ lo informa che il $client-1$ è uscito dalla chat e così fa lui di conseguenza. Il $client-2$ inoltre dovrà digitare un carattere per poter uscire dalla chiamata bloccante di lettura del messaggio di chat e dopodichè torna al menu principale terminando il thread *read_in_chat* che si occupava della chat.

## 1.12 Gestione Logout

Il client che vuole effettuare il logout dal server deve scegliere l'opzione 3) del menù. Così facendo esce dal ciclo while del menù, pulisce la memoria utilizzata e termina. Il server è in grado di rilevare da solo la disconnessione da parte del client e agisce di conseguenza, andando ad eliminare i due thread, *handler_read* e *handler_connection*. Pulendo inoltre la memoria sporcata dai dati del client che ora è disconnesso.

# 2 Scelte Implementative

## 2.1 Server

Il server è di tipo TCP, inoltre nel momento successivo alla fase di Autenticazione client-server, quest'ultimo crea due thread per la gestione del client che ha appena effettuato il login:

1. *handler_read*, si attiva solamente quando l'altro thread lo risveglia (per via di un messaggio ricevuto), prende l'opt letto dal thread *handler_connection* e lo analizza, andando ad effettuare l'operazione richiesta dal client
2. *handler_connection*, si blocca in attesa di un messaggio da parte del client. Quando il messaggio arriva al server, questo viene salvato in un buffer comune anche all'*handler_read*. Dopodiché, manda un segnale per svegliare l'altro thread e si mette in attesa del prossimo messaggio.

## 2.2 Client

*Login utenti*
Il client una volta che avvia la connessione con la porta usata dal server, parte subito la fase di autenticazione dopo la quale l'utente riceve la lista di utenti online disponibili per la chat e il menù con il quale può interagire con il server (come riportato nell'immagine sotto).

Il menu funziona nel seguente modo:

1. *Parlare con un altro utente* → successivamente a questa richiesta viene chiesto all'utente di inserire anche il nome del client con cui vuole parlare (la richiesta viene inoltrata al server solamente se l'utente inserito risulta nella lista precedentemente inviata al client, altrimenti l'utente viene indirizzato all'opzione 1 per avere una lista aggiornata della lista di utenti online)

```
[------ ONLINE USERS ------]

 Lista Vuota

[-------------------------]


Digitare il numero corrispondente alla scelta che si vuole fare
1: Parlare con un altro utente
2: Vedere la lista di utenti online
3: Logout dal server
4: Mettersi in attesa di chat
```

2. *Vedere la lista di utenti online* → quest'azione permette di inviare al server una richiesta per ottenere la lista di utenti al momento disponibile per chattare.

3. *Logout dal server* → l'utente vuole terminare la sessione con il server ed andare offline, in altre parole esce dal ciclo while del menu, pulisce i buffer usati e termina.

4. *Mettersi in attesa di chat* → il client crea un thread per la lettura della chat, dopodiché si mette in attesa all'interno di un ciclo while. Se il client vuole uscire prima dell'arrivo della richiesta di chat da parte di un altro client è libero di farlo tramite l'uso del segnale SIGTSPT (*ctrl* + *z*). Altrimenti il client può decidere se accettare oppure no la richiesta di chat da parte dell'altro utente. L'utente può decidere di:

   (a) rifiutare la richiesta, e quindi invia una risposta negativa al server per poter rispondere all'altro client e tornare in attesa di chat; oppure

   (b) accettare la richiesta e iniziare la fase di autenticazione con l'altro client spiegata nel dettaglio nella sezione "Messaggi Scambiati"; alla fine della quale inizierà la vera fase di chat tra i due utenti.

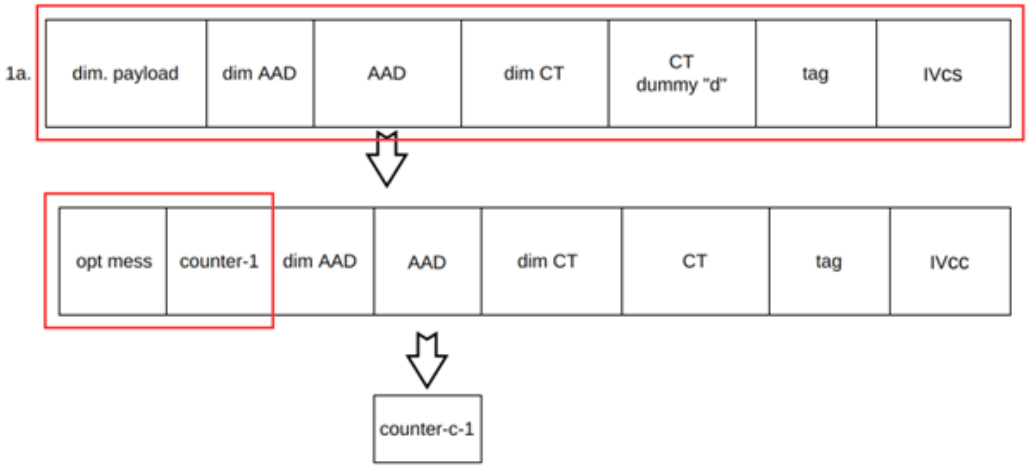
*Utenti disponibili a chattare*
Una delle principali scelte di implementazione riguarda gli utenti online, in particolare, questi vengono considerati online solamente dopo che hanno scelto dal menu l'operazione di attesa di chat (opzione 4). In questo caso il client invia un messaggio al server nel quale indica questa scelta e viene inserito dal server tra gli utenti online. Il server rimuove il client solamente nel caso

questo inizia una chat con un altro utente oppure decida di terminare l'attesa di chat.

### Sessione client client

Dopo che il client ha mandato una richiesta di chat ad un altro utente e questo ha accettato, allora da questo momento parte la fase di autenticazione tra i due utenti al fine di raggiungere una chiave di sessione tra i due e quindi permette di cifrare i messaggi che si scambiano. La parte di creazione della chiave di sessione condivisa è analizzata nel dettaglio nella sezione "Messaggi Scambiati". Dopodichè, inizia la vera fase di chat in cui il server agisce solamente da tramite e legge solamente il codice OPT, mentre il messaggio scambiato è cifrato con la chiave di sessione dei due utenti e quindi non è possibile per il server andare a decifrare il contenuto e vedere il messaggio scambiato.

Nella sezione "Sessione tra i due client" è riportato nel dettaglio tutti i campi che formano il messaggio tra i due utenti, mentre nell'immagine sotto sono evidenziati in rosso i campi che il server legge ed usa, mentre gli altri sono quelli che interessano al client.




### Test e Uso

Per la realizzazione dei certificati dei client, del server e della CA è stato utilizzato SimpleAuthority CA visto a lezione. Infatti i client sono già registrati al momento di avvio del test. In particolare, è stato realizzato uno script (script.sh) per la generazione delle cartelle e dei certificati utilizzati nel progetto, tramite l'utilizzo di Openssl da linea di comando.

Il server tiene traccia delle connessioni dei client e delle richieste che gli arrivano da parte di questi.


Gli utenti creati per testare il progetto e la relativa password (per leggere la chiave privata) sono le seguenti:

- bob bob0
- alice alice
- andrea andrea

Per compilare ed eseguire il server:

```
g++ server.cpp -lpthread -I ./../ -L/usr/local/lib -o server.o -lssl
-lcrypto
./server.o [porta]
```

Per compilare ed eseguire il client:

```
g++ client.cpp -lpthread -I ./../ -L/usr/local/lib -o client.o -lssl
-lcrypto
./client.o localhost [nome_utente] [porta]
```