UNIVERSITÀ DI PISA

Progetto Applied Cryptography

Leonardo Talerico mat. 564903
Morucci Davide mat. 548058
Gallo Riccardo mat. 627026

Ipotesi di Design

Il sistema Client Server è multithread:

1) I Client hanno due thread, il primo utilizzato per inviare comandi al Server e gestirne le risposte, il secondo utilizzato invece per attendere eventuali richieste di chat e gestirle. La gestione della chat avviene poi con l'utilizzo di due ulteriori thread, uno utilizzato per inviare messaggi e l'altro per riceverli.
2) Il Server istanzia un thread per ogni utente Client che deve gestire. Istanzia inoltre due ulteriori thread per gestire ogni chat.

I Client hanno tutti due socket, il primo con il quale eseguono la Connect con il Server, mentre il secondo è utilizzato per ricevere richieste di comunicazione.

Le connessioni utilizzate saranno di tipo **TCP**, in quanto assicura che i messaggi arrivino tutti nell'ordine corretto e non danneggiati/alterati.

Client - Server

Autenticazione di Client, Server e PFS

→ Quando l'applicazione Client viene mandata in esecuzione, si ha l'autenticazione di Client e Server, l'invio dello username da parte del Client e infine la fase di PFS tra i due per lo scambio della chiave e dei parametri che di lì in avanti saranno utilizzati per lo scambio di messaggi tra i due.

1) Il Server invia al Client un nonce R casuale
2) Il Server invia inoltre al Client il proprio certificato
3) Il Client invia al Server un nonce R' casuale
4) Il Client invia lo username inserito dall'utente
5) Il Client invia la chiave pubblica temporanea che ha generato, è che sarà usata nella successiva parte relativa alla PFS
6) Il Client invia al Server il nonce R inizialmente ricevuto, concatenato con la chiave pubblica temporanea, concatenato a sua volta con lo username precedentemente inviato. Il tutto è firmato con la chiave privata del Client

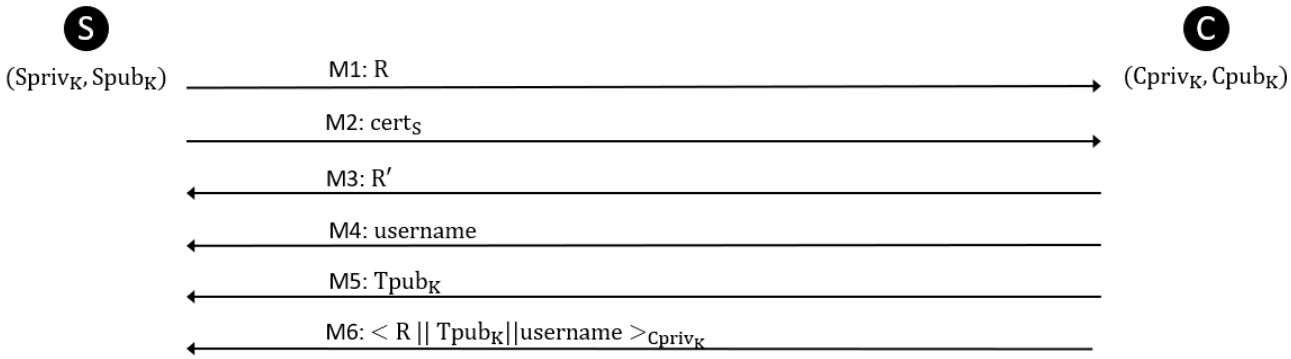
→Viene richiesto lo username dell'utente, in modo tale da poter ricavare la chiave pubblica per la fase di autenticazione.

Osservazione: Ci si assicura che lo username non sia già online.

Osservazione: sul lato Client sarà presente una cartella contente vari file username_public.pem e username_private.pem contenenti rispettivamente chiave pubblica e privata dell'utente "username".
Questo meccanismo è usato così che possano essere presenti più utenti sulla stessa macchina, in modo da poter lavorare direttamente in localhost.

→ Il Server contiene già le chiavi pubbliche degli utenti che possono accedere al servizio.

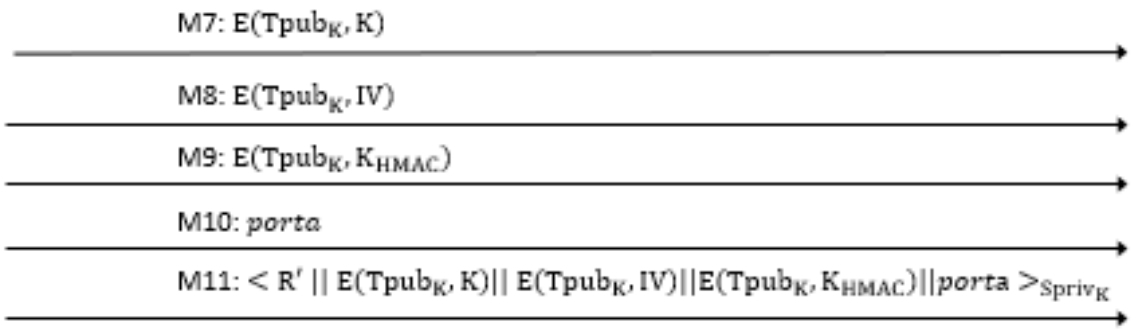
Osservazione: Il Server verifica, tramite la chiave pubblica del Client in suo possesso, la firma di quest'ultimo e che il contenuto della firma sia conforme a quanto ricevuto precedentemente.



Il Client, dopo aver ricevuto il certificato, certifica prima di tutto la CA e successivamente verifica che sia valido, e lo utilizza per verificare la firma ricevuta.

→Avviene la fase di **Perfect Forward Secrecy** tra Client e Server, con lo scambio della chiave di sessione K che sarà utilizzata per i futuri scambi di messaggi, l'invio dell'IV e della chiave $\mathrm{K_{HMAC}}$ (diversa dalla precedente) che sarà utilizzata per generare l'HMAC tra Client e Server:

7) Il Server genera una chiave di sessione simmetrica randomica e la invia al Client cifrandola con la chiave pubblica temporanea del Client
8) Il Server genera un IV randomicamente, lo cifra utilizzando la chiave pubblica temporanea e lo invia al Client
9) Il Server genera una chiave di sessione simmetrica randomica (HMAC) e la invia al Client cifrandola con la chiave pubblica temporanea del Client
10) Il Server invia al Client la porta sulla quale dovrà mettersi in ascolto
11) Il Server invia al Client il "nonce" R' concatenato con la chiave di sessione K cifrata con la chiave pubblica temporanea del Client, IV cifrato con la chiave pubblica temporanea del Client, K_HMAC cifrata con la chiave pubblica temporanea del Client e la porta, il tutto firmato con la chiave privata del Server



Osservazione: Il Server invia al Client la porta su cui dovrà mettersi in ascolto per eventuali richieste di chat. Ogni utente viene identificato da una porta diversa, mentre l'indirizzo IP è lo stesso "127.0.0.1".

(Formato messaggi → vedere **qui**)

Durante l'esecuzione del protocollo

- Il Client decifra i messaggi M7, M8 e M9 utilizzando la propria chiave privata temporanea, ricavando l'IV e le chiavi di sessione, che verranno poi utilizzati per le successive comunicazioni.
- Il Client verifica l'autenticità del messaggio M11 utilizzando la chiave pubblica del Server, controllando che il nonce R', le chiavi di sessione, l'IV e la porta siano conformi a quelli precedentemente inviati.

Da questo momento tutti i messaggi scambiati tra Client e Server sono cifrati utilizzando AES256 + HMAC (del ciphertext + IV)

→ Il meccanismo di cifratura AES sfrutta un IV che viene incrementato per ogni messaggio.
Il fatto che l'IV cambi per ogni messaggio inviato/ricevuto fa sì che, in caso di un attacco *record and playback* interno alla sessione, l'HMAC ricevuto e quello generato per il confronto risulterebbero differenti (dato che utilizzano IV diversi).

Comandi

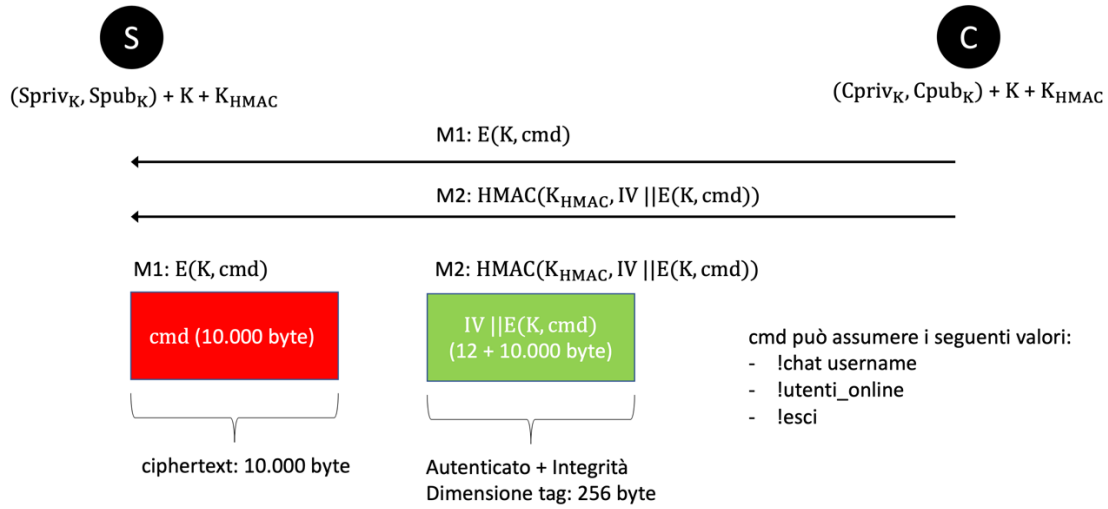
→ L'utente può inviare (cifrando con la chiave di sessione condivisa con il Server) i seguenti comandi:

- **!utenti_online**: dopo aver ricevuto questo comando, il Server invia verso il Client una lista di tutti gli utenti attualmente connessi. La lista viene inviata cifrata.
- **!chat username**: dopo aver ricevuto questo comando, il Server invia un messaggio all'utente username, quest'ultimo può accettare o meno la

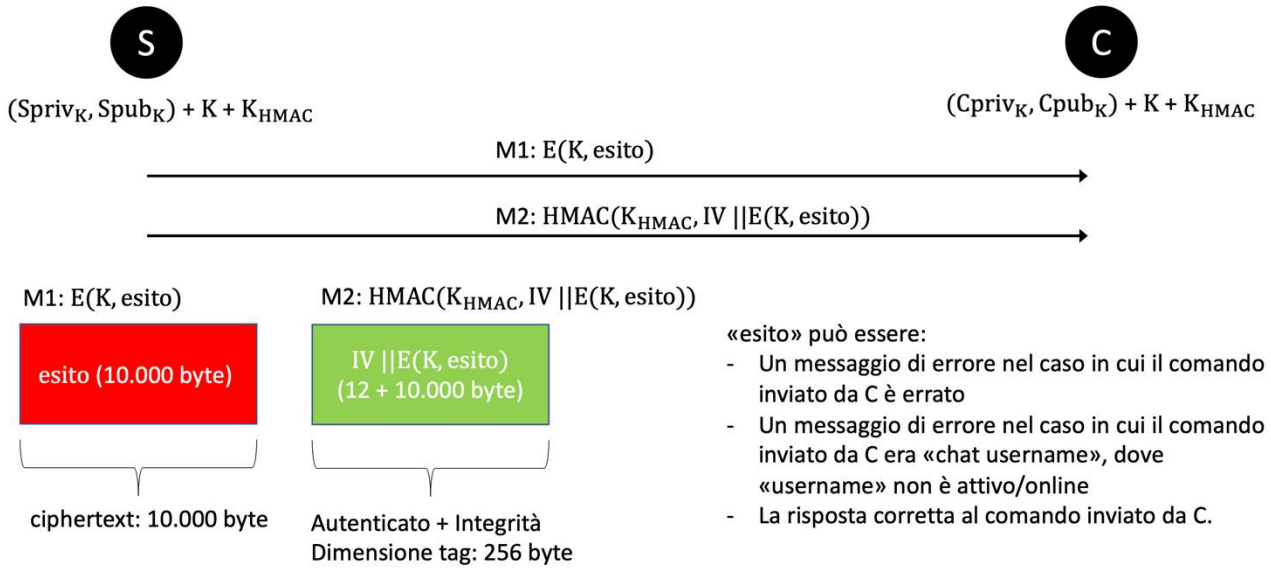
richiesta di connessione, in caso affermativo, viene avviata la fase di chatting. (vedere meglio a seguire)

- !**esci**: dopo aver ricevuto questo comando, il Server invia verso il Client un messaggio di ricevuta disconnessione e chiude i socket, i thread e dealloca le varie strutture utilizzate. Il Client, una volta ricevuto il messaggio di disconnessione, effettua le medesime chiusure e deallocazioni.

→ Il Client invia il comando al server e lo cifra con la chiave simmetrica K. Inoltre invia anche un tag, calcolato andando ad effettuare l'HMAC (con chiave K_{HMAC}) dell'IV concatenato al ciphertext del comando



→ quando il Server riceve il comando lo decifra e verifica che sia corretto. In base a questa verifica invia un "esito", cifrato con la chiave simmetrica K, e un tag calcolato con l'HMAC dell'IV concatenato al ciphertext dell'esito (calcolato con chiave K_{HMAC}).



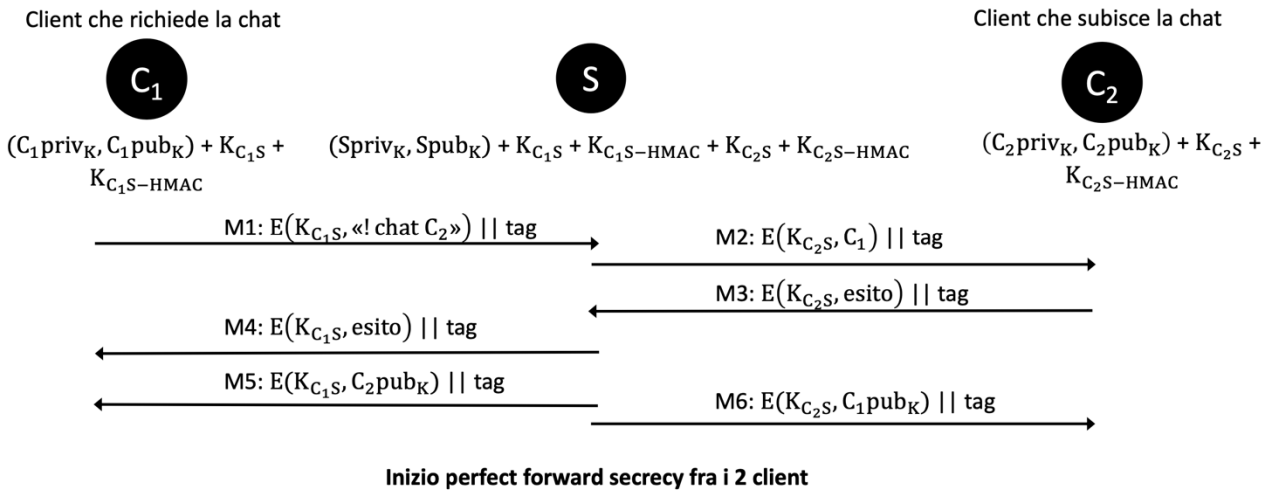
Chat

Inizializzazione Chat

Nei seguenti step si suppone che C_1 sia il Client che richiede di comunicare con il Cient C_2. Inoltre, il Server verrà indicato con la lettera S.

Step:

1) C_1 invia ad S la richiesta di comunicazione con C_2, cifrata con la chiave di sessione K_{C_1S} (chiave condivisa fra C_1 e S).
2) S decifra il messaggio M1 ricevuto da C_1 e invia a C_2 un messaggio, cifrato con la chiave K_{C_2S} (chiave condivisa fra C_2 e S), in cui viene indicato che C_1 vuole iniziare una chat.
   Osservazione: prima di inoltrare la richiesta a C_2, S deve verificare che C_2 sia online e che non stia già comunicando con altri utenti. Se si verifica una di queste due condizioni, S notificherà a C_1 di mancata comunicazione.
3) C_2 comunica ad S la scelta effettuata, cifrata con la chiave di sessione K_{C_2S}
   a. Se la risposta è "y", il C_2 ha accettato la richiesta di comunicazione da C_1
   b. Se la risposta è "n", il C_2 ha rifiutato la richiesta di comunicazione da C_1
4) S decifra il messaggio M3 e inoltra a C_1 la risposta di C_2, cifrata con la chiave di sessione K_{C_1S}
5) Se C_2 ha accettato la richiesta di comunicazione, S invia a C_1 la chiave pubblica di C_2, cifrata con la chiave di sessione K_{C_1S}
6) Se C_2 ha accettato la richiesta di comunicazione, S invia a C_2 la chiave pubblica di C_1, cifrata con la chiave di sessione K_{C_2S}



Inizio perfect forward secrecy fra i 2 client

Significato messaggi:
- tag = HMAC($K_{C_iS-HMAC}$, IV ||E(K, …))
- M1: E(K, ! chat C_2) || tag → l'utente vuole comunicare con il client C_2
- M2: E(K_{C_2S}, C_1) || tag → il server dice al client C_2 che C_1 vuole iniziare la chat
- esito = «y»/«n»

(Formato messaggi → vedere **qui**)

Perfect Forward Secrecy ($C_1 - S - C_2$)

Adesso inizia la fase di perfect forward secrecy fra C_1 e C_2 (e S come intermediario per lo scambio dei messaggi) per lo scambio di una chiave di sessione $K_{C_1C_2}$ che i due client condivideranno e useranno per la cifratura dei messaggi che vorranno scambiarsi.

7) C_2 invia un nonce R a S. Inoltre, per garantire l'autenticità del messaggio, C_2 invia ad S un tag calcolato con la chiave $K_{C_2S-HMAC}$ contenente l'IV concatenato al nonce.
   S verifica l'autenticità del messaggio M7.
8) S inoltra il messaggio il nonce R a C_1. Inoltre, per garantire l'autenticità del messaggio, S invia a C_1 un tag calcolato con la chiave $K_{C_1S-HMAC}$ contenente l'IV concatenato al nonce.
   C_1 verifica l'autenticità del messaggio M8.
9) C_1 genera una coppia di chiavi pubbliche e private temporanee e invia ad S la chiave pubblica temporanea
10) C_1 firma con la propria chiave privata il nonce R concatenato con la chiave pubblica temporanea e invia tutto a S. Inoltre, per garantire l'autenticità del messaggio, C_2 invia ad S un tag calcolato con la chiave $K_{C_1S-HMAC}$ contenente l'IV concatenato alla firma.
11) S inoltra a C_2 la chiave temporanea

12) S inoltra il messaggio M11 a C_2. Inoltre, per garantire l'autenticità del messaggio, S invia a C_2 un tag calcolato con la chiave $K_{C_2S-HMAC}$ contenente l'IV concatenato alla firma.
C_2 verifica l'autenticità del messaggio M12

13) C_1 invia un nonce R' a S. Inoltre, per garantire l'autenticità del messaggio, C_1 invia ad S un tag calcolato con la chiave $K_{C_1S-HMAC}$ contenente l'IV concatenato al nonce.
S verifica l'autenticità del messaggio M13.

14) S inoltra il messaggio il nonce R' a C_2. Inoltre, per garantire l'autenticità del messaggio, S invia a C_2 un tag calcolato con la chiave $K_{C_2S-HMAC}$ contenente l'IV concatenato al nonce.
C_2 verifica l'autenticità del messaggio M14.

15) C_2 genera una chiave di sessione simmetrica randomica $\mathrm{K_{C_1C_2}}$ e la invia a S cifrandola con la chiave pubblica temporanea

16) C_2 genera un IV randomicamente, lo cifra utilizzando la chiave pubblica temporanea e lo invia a S

17) C_2 genera la chiave HMAC di sessione randomica $\mathrm{K_{C_1C_2-HMAC}}$ e la invia a S cifrandola con la chiave pubblica temporanea

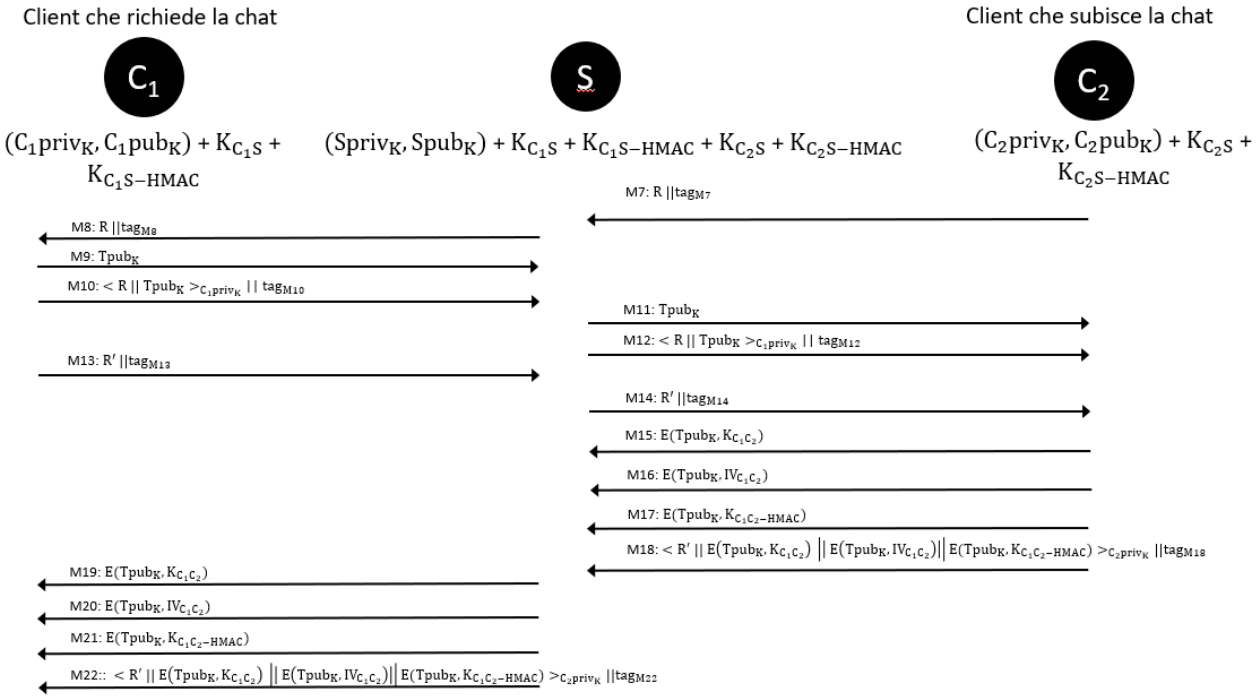
18) C_2 invia a S il "nonce" R' concatenato con la chiave di sessione $\mathrm{K_{C_1C_2}}$ cifrata con la chiave pubblica temporanea, IV cifrato con la chiave pubblica temporanea e $\mathrm{K_{C_1C_2-HMAC}}$, cifrata con la chiave pubblica temporanea, il tutto firmato con la chiave privata di C_2. Inoltre, per garantire l'autenticità del messaggio, C_2 invia ad S un tag calcolato con la chiave $K_{C_2S-HMAC}$ contenente l'IV concatenato alla firma.
S verifica l'autenticità del messaggio M18.

19) S inoltra il messaggio M15 a C_1

20) S inoltra il messaggio M16 a C_1

21) S inoltra il messaggio M17 a C_1

22) S inoltra il messaggio M18 a C_1. Inoltre, per garantire l'autenticità del messaggio, S invia a C_1 un tag calcolato con la chiave $K_{C_1S-HMAC}$ contenente l'IV concatenato alla firma.
C_1 verifica l'autenticità del messaggio M22.



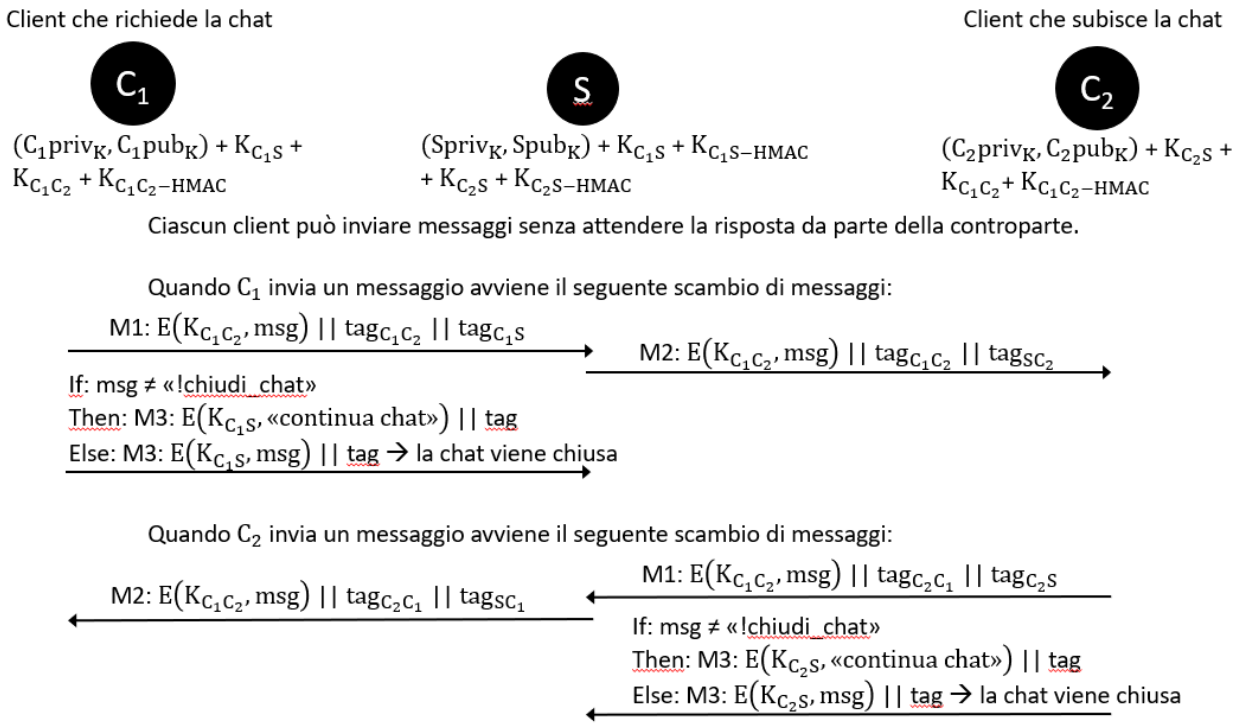
(Formato messaggi → vedere **qui**)

→ la chiave simmetrica scambiata verrà usata per cifrare i messaggi che C_1 invierà a C_2 e viceversa, tenendo sempre conto che i messaggi passeranno dal Server S (che però non potrà decifrarli perché non sarà in possesso della chiave di sessione $K_{C_1C_2}$).

Scambio messaggi

Step:

1) C_1 invia ad S il messaggio per C_2 cifrato con la chiave simmetrica tra C_1 e C_2, il tag per C_2 e il tag per il server calcolati con i relativi IV
2) S invia a C_2 il messaggio cifrato proveniente da C_1 con il tag generato da C_1 e con il tag che il server genera per C_2
3) C_1 invia a S un messaggio cifrato con la loro chiave simmetrica per informalo sull'eventuale chiusura della chat o meno



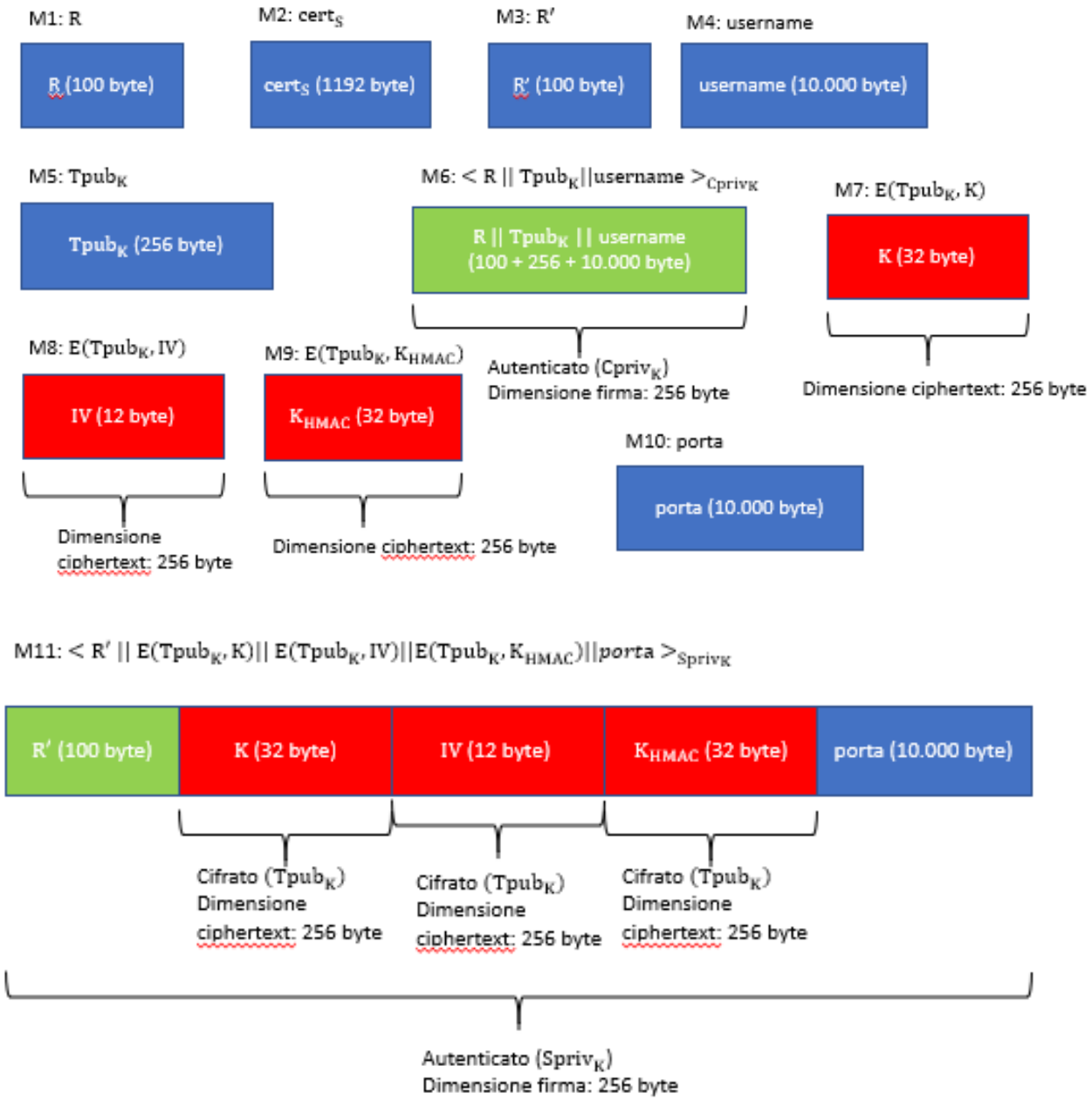
(Formato messaggi → vedere **qui**)

→ Quando uno dei due Client vuole terminare la chat, invia un messaggio del tipo **!chiudi_chat** in modo da far capire all'altro Client di voler terminare la sessione.
Alla ricezione del messaggio, il Client ricevente chiuderà il socket e terminerà l'applicazione. Lo stesso procedimento di chiusura verrà eseguito anche lato Client mittente.

Formato dei messaggi

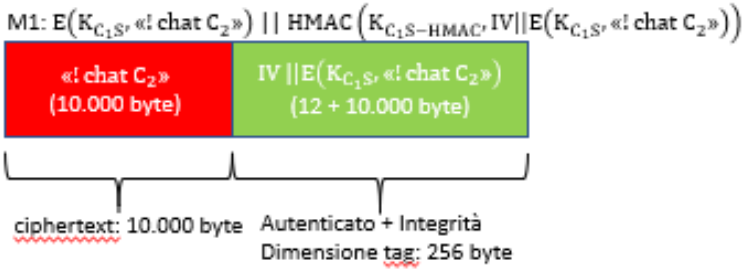
Autenticazione di Client, Server e PFS

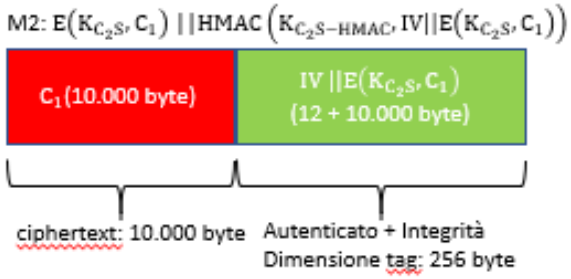
(torna al grafico → clicca **qui**)

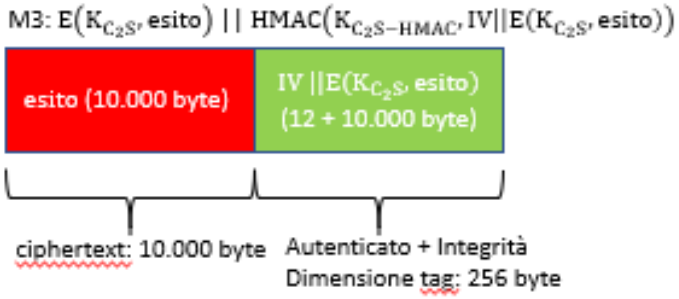


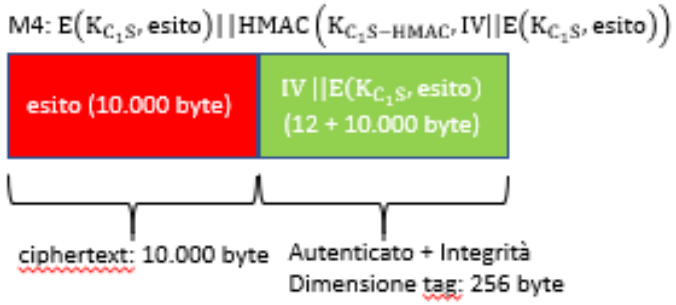
Inizializzazione chat_

(torna al grafico → clicca **qui**)

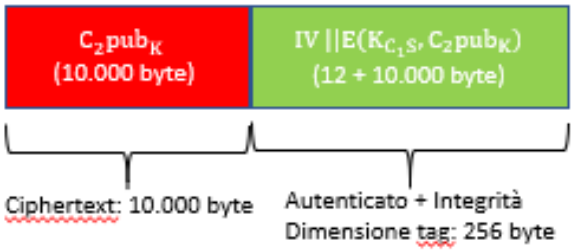




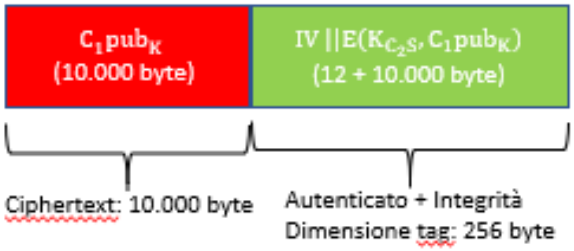




M5:$E(K_{C_1S}, C_2pub_K) || HMAC(K_{C_1S-HMAC}, IV||E(K_{C_1S}, C_2pub_K))$



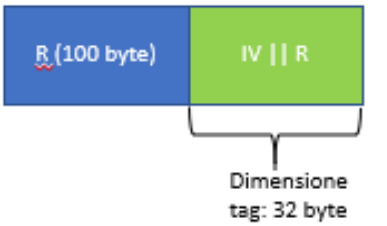
M6: $E(K_{C_2S}, C_1pub_K) || HMAC(K_{C_2S-HMAC}, IV||E(K_{C_2S}, C_1pub_K))$



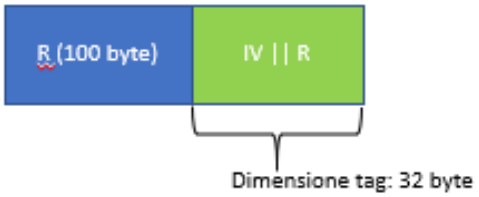
Perfect forward secrecy $(C_1 - S - C_2)$_

(torna al grafico → clicca **qui**)

M7 R || $HMAC(K_{C_2S-HMAC}, IV || R)$

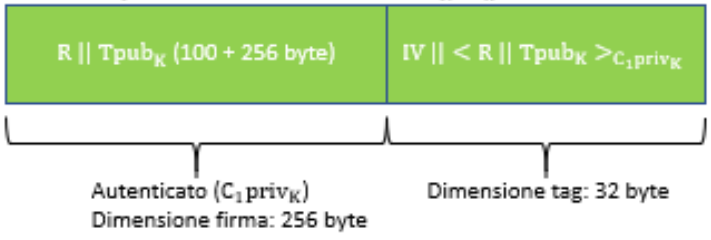


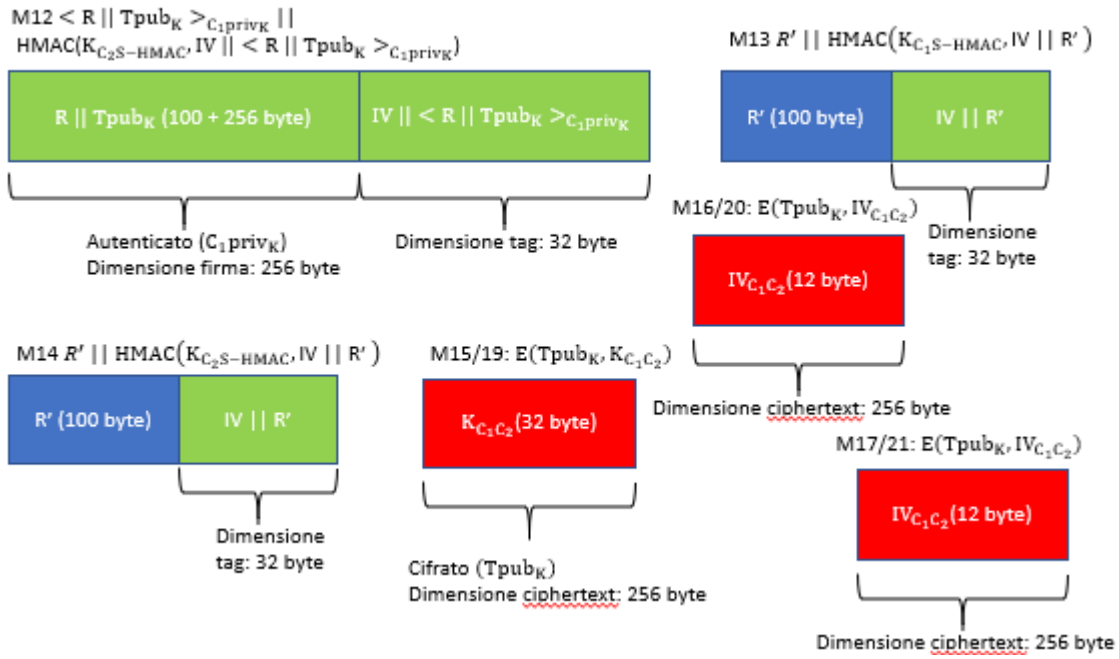
M8 R || $HMAC(K_{C_1S-HMAC}, IV || R)$

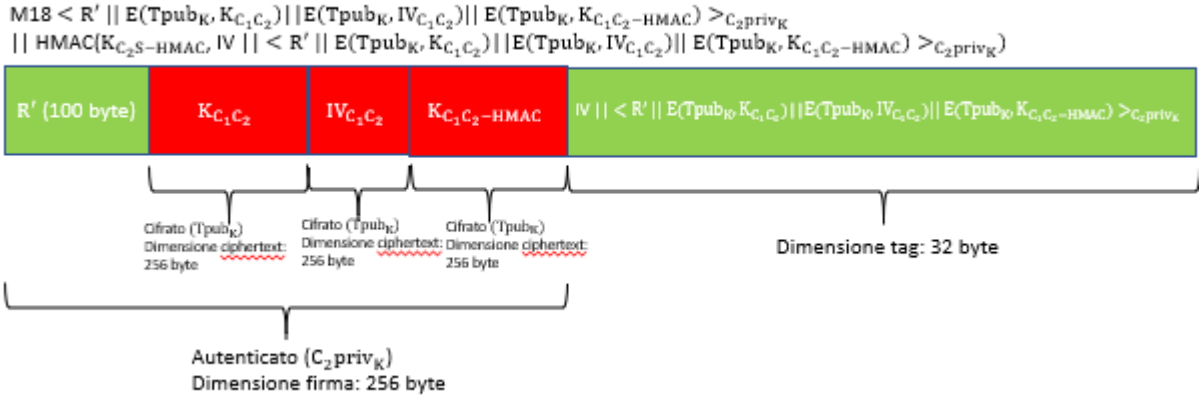


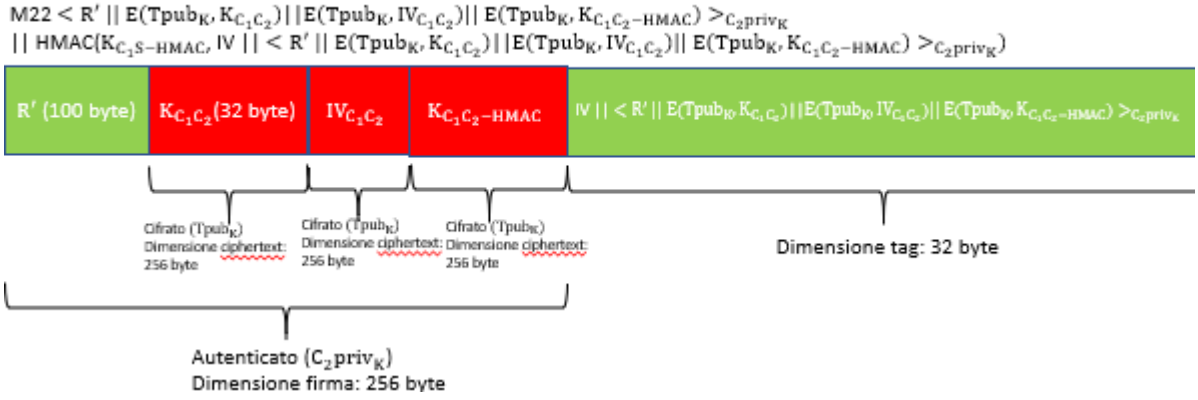
M9 & M11: $Tpub_K$

M10 $< R || Tpub_K >_{C_1priv_K}$ ||
$HMAC(K_{C_1S-HMAC}, IV || < R || Tpub_K >_{C_1priv_K})$




M12 < R || Tpub_K >_C1privK || HMAC(K_C2S-HMAC, IV || < R || Tpub_K >_C1privK)
R || Tpub_K (100 + 256 byte)
IV || < R || Tpub_K >_C1privK
Autenticato (C1privK)
Dimensione firma: 256 byte
Dimensione tag: 32 byte
M13 R' || HMAC(K_C1S-HMAC, IV || R')
R' (100 byte)
IV || R'
Dimensione tag: 32 byte
M16/20: E(Tpub_K, IV_C1C2)
IV_C1C2 (12 byte)
Dimensione ciphertext: 256 byte
M14 R' || HMAC(K_C2S-HMAC, IV || R')
R' (100 byte)
IV || R'
Dimensione tag: 32 byte
M15/19: E(Tpub_K, K_C1C2)
K_C1C2 (32 byte)
Cifrato (Tpub_K)
Dimensione ciphertext: 256 byte
M17/21: E(Tpub_K, IV_C1C2)
IV_C1C2 (12 byte)
Dimensione ciphertext: 256 byte

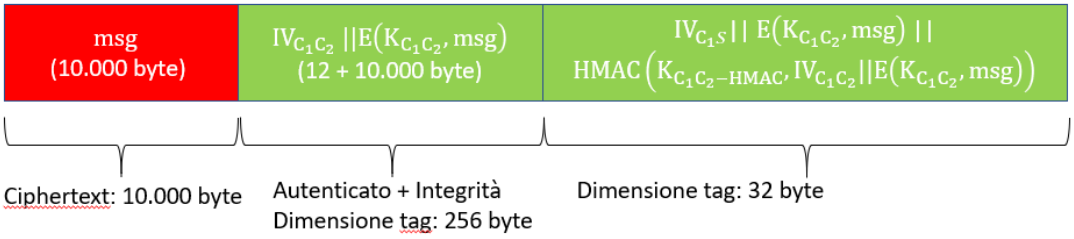

M18 < R' || E(Tpub_K, K_C1C2) || E(Tpub_K, IV_C1C2) || E(Tpub_K, K_C1C2-HMAC) >_C2privK
|| HMAC(K_C2S-HMAC, IV || < R' || E(Tpub_K, K_C1C2) || E(Tpub_K, IV_C1C2) || E(Tpub_K, K_C1C2-HMAC) >_C2privK)
R' (100 byte)
K_C1C2
IV_C1C2
K_C1C2-HMAC
IV || < R' || E(Tpub_K, K_C1C2) || E(Tpub_K, IV_C1C2) || E(Tpub_K, K_C1C2-HMAC) >_C2privK
Cifrato (Tpub_K) Dimensione ciphertext: 256 byte
Cifrato (Tpub_K) Dimensione ciphertext: 256 byte
Cifrato (Tpub_K) Dimensione ciphertext: 256 byte
Dimensione tag: 32 byte
Autenticato (C2privK)
Dimensione firma: 256 byte


M22 < R' || E(Tpub_K, K_C1C2) || E(Tpub_K, IV_C1C2) || E(Tpub_K, K_C1C2-HMAC) >_C2privK
|| HMAC(K_C1S-HMAC, IV || < R' || E(Tpub_K, K_C1C2) || E(Tpub_K, IV_C1C2) || E(Tpub_K, K_C1C2-HMAC) >_C2privK)
R' (100 byte)
K_C1C2 (32 byte)
IV_C1C2
K_C1C2-HMAC
IV || < R' || E(Tpub_K, K_C1C2) || E(Tpub_K, IV_C1C2) || E(Tpub_K, K_C1C2-HMAC) >_C2privK
Cifrato (Tpub_K) Dimensione ciphertext: 256 byte
Cifrato (Tpub_K) Dimensione ciphertext: 256 byte
Cifrato (Tpub_K) Dimensione ciphertext: 256 byte
Dimensione tag: 32 byte
Autenticato (C2privK)
Dimensione firma: 256 byte

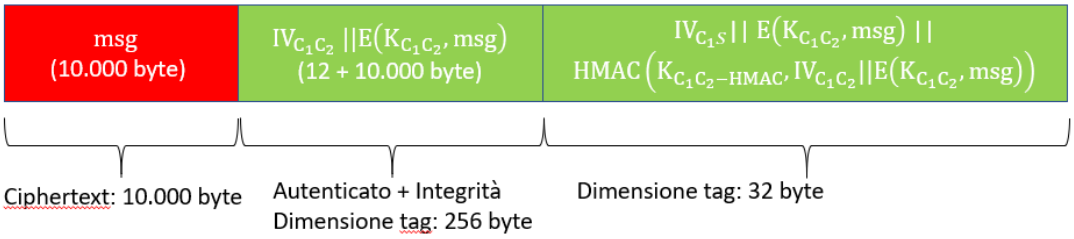
Scambio messaggi_

(torna al grafico → clicca qui)

M1: $E(K_{C_1C_2}, msg) \,||\, HMAC\left(K_{C_1C_2-HMAC}, IV_{C_1C_2}||E(K_{C_1C_2}, msg)\right) \,||$
$HMAC(\, K_{C_1S-HMAC}, IV_{C_1S} \,||\, E(K_{C_1C_2}, msg) \,||\, HMAC\left(K_{C_1C_2-HMAC}, IV_{C_1C_2}||E(K_{C_1C_2}, msg)\right))$



M2: $E(K_{C_1C_2}, msg) \,||\, HMAC\left(K_{C_1C_2-HMAC}, IV_{C_1C_2}||E(K_{C_1C_2}, msg)\right) \,||$
$HMAC(\, K_{C_2S-HMAC}, IV_{SC2} \,||\, E(K_{C_1C_2}, msg) \,||\, HMAC\left(K_{C_1C_2-HMAC}, IV_{C_1C_2}||E(K_{C_1C_2}, msg)\right))$



M3: $E(K_{C_1S}, «continua\ chat») \,||\, HMAC\left(K_{C_1S-HMAC}, IV||E(K_{C_1S}, «continua\ chat»)\right)$