

Kerberos

1 Storia

Il protocollo **Kerberos** fornisce un servizio di autenticazione in ambito distribuito. Esso fu inizialmente sviluppato negli anni '80 nell'ambito di un progetto di ricerca svolto dall'MIT insieme a IBM, che aveva l'obiettivo di realizzare un sistema di computing distribuito, il quale permettesse di connettersi a vari servizi da terminali presenti in un campus.

Le prime tre versioni di Kerberos furono usate solo internamente presso l'MIT, mentre la versione 4, ideata nel 1987 da Steve Miller e Clifford Neuman, fu resa pubblica. Infine, la versione 5 fu ideata nel 1990 da John Kohl e Clifford Neuman per risolvere alcune limitazioni della versione 4, e fu standardizzata tramite RFC (attualmente RFC 4120).

2 Autenticazione in ambito distribuito

Lo scenario in cui Kerberos si applica è appunto l'autenticazione in ambito distribuito: gli utenti usano dei terminali, delle workstation, dai quali devono poter accedere ai servizi forniti da dei server. I server devono effettuare l'*autenticazione* e l'*autorizzazione* degli utenti,¹ cioè devono essere in grado di:

- autenticare le richieste di servizi, cioè verificare le identità degli utenti da cui provengono le richieste;
- consentire gli accessi solo a utenti autorizzati.

Ci sono vari modi di affrontare questo problema. Una prima soluzione è un meccanismo di autenticazione point-to-point, basato sul verificare al livello di rete che una determinata workstation stia comunicando con un determinato server. Tale meccanismo potrebbe magari essere facile da implementare, ma è insicuro, perché non verifica quale sia l'utente che sta utilizzando una workstation in un determinato momento: un utente potrebbe "impadronirsi" illegalmente di una workstation e ottenere così l'accesso ai servizi per cui essa è autorizzata.

Un'altra soluzione è eseguire l'autenticazione tra gli utenti e i server, tipicamente tramite user ID e password. Essa può funzionare, ma ha dei notevoli limiti:

¹Quando si parla di Kerberos si usa spesso, per brevità, il solo termine "autenticazione", ma il sistema Kerberos esegue sia l'autenticazione che l'autorizzazione.

- È inefficiente, perché in ambito distribuito ogni server deve memorizzare la lista di user ID, password e autorizzazioni degli utenti. Ciò significa, ad esempio, che per cambiare la propria password un utente dovrebbe modificarla individualmente presso ciascun server.
- È insicura, perché replicare su ogni server le informazioni sensibili di autenticazione e autorizzazione allarga la superficie dei possibili attacchi. Infatti, se anche la maggior parte dei server fossero molto ben protetti, un singolo server che sia invece esposto (per qualunque motivo) sarebbe sufficiente a compromettere la sicurezza di tutti i server: un attaccante potrebbe ottenere le informazioni di autenticazione dal server esposto e usarle per accedere anche agli altri servizi. In altre parole, la sicurezza del sistema sarebbe forte solo quanto l'elemento più debole.

La soluzione fornita da Kerberos è un sistema centralizzato (server Kerberos) che si occupa di verificare l'autenticazione e l'autorizzazione degli utenti: prima di rivolgersi a un server per richiedere un servizio, l'utente si deve rivolgere al server Kerberos per richiedere l'autenticazione e l'autorizzazione.

Kerberos prende il nome dal cane a tre teste Cerbero, che nella mitologia greca protegge l'Ade. Le tre teste rappresentano i tre importanti servizi di Authentication, Authorization e Accounting/Auditing (tenere traccia di tutto ciò che gli utenti fanno), detti "AAA", "le tre A". Di questi tre servizi Kerberos implementa principalmente l'autenticazione, e implicitamente anche l'autorizzazione, mentre non implementa l'accounting/auditing.

Il protocollo di autenticazione Kerberos si basa sul Needham-Schroeder a chiave segreta, e come tale utilizza solo la crittografia simmetrica. Esso è un protocollo complesso, dunque verrà presentato in modo incrementale, partendo da delle versioni semplificate per arrivare poi al reale protocollo della versione 4, e infine si introdurranno le differenze tra le versioni 4 e 5.

3 Autenticazione semplice

Questa prima versione semplificata di Kerberos introduce nella rete un **Authentication Server/Service (AS)** fidato (chiamato anche *Key Distribution Center, KDC*), che conosce tutti gli username e le password degli utenti registrati e sa a quali server ogni utente è autorizzato ad accedere.

Un utente invia all'AS le proprie credenziali (username e password) e la richiesta di accesso a un dato server/servizio V (può essere che uno stesso server fornisca più servizi, e in tal caso le autorizzazioni sono comunque gestite separatamente per ciascun servizio). Se verifica che le credenziali sono corrette e che l'utente è autorizzato ad accedere al servizio richiesto, l'AS rilascia all'utente un **ticket**, un messaggio che l'utente dovrà esibire al server V come prova dell'avvenuta autenticazione e autorizzazione. Si noti che l'AS non interagisce mai con il server V : esso invia il ticket all'utente, non al server, e spetta poi all'utente "spendere" tale ticket inviandolo al server.

Il ticket è cifrato con una chiave simmetrica K_V che è condivisa solo dall'AS e dal server V , così:

- solo il server V può decifrare il ticket per verificare le informazioni che contiene, mentre l'utente o un attaccante non possono modificare tali informazioni in quanto non conoscono K_V ;
- se usando K_V il server decifra correttamente il ticket (cioè ottiene un messaggio decifrato che ha la struttura corretta di un ticket, stabilita dal protocollo) allora ha la prova che esso sia stato effettivamente generato da AS, l'unico server con cui V condivide la chiave K_V .

Se ci sono più server per cui si deve gestire l'autenticazione allora l'AS condivide con ciascuno di essi una chiave *diversa*, non la stessa K_V , in modo che un eventuale server compromesso non comprometta anche la sicurezza di tutti gli altri.

Le informazioni che il ticket contiene sono le seguenti:

- Il nome (ID) dell'utente.
- Il nome (ID) del server, necessario perché il server deve verificare che il ticket sia stato generato per il servizio che esso fornisce e non per un altro. Altrimenti, un utente potrebbe richiedere l'autorizzazione per un servizio e poi usare il ticket per accedere a un altro servizio, per il quale magari non è autorizzato.
- L'indirizzo di rete della workstation da cui è stata fatta la richiesta all'AS. Esso lega l'autenticazione e l'autorizzazione alla macchina da cui è stata fatta la richiesta, oltre che all'utente, per cercare di impedire a un attaccante di rubare il ticket cifrato e riutilizzarlo da un'altra workstation.

Allora, dopo aver decifrato il ticket e prima di consentire l'accesso al proprio servizio il server V controlla che:

- l'utente che ha fatto la richiesta sia quello indicato dall'ID presente nel ticket;
- la richiesta provenga dalla macchina il cui indirizzo di rete è indicato nel ticket;
- il ticket sia stato generato per il servizio fornito da V , ovvero che l'ID del server indicato nel ticket coincida con l'ID di questo server.

3.1 Messaggi in notazione formale

Per precisare il funzionamento del protocollo è utile rappresentare i messaggi scambiati in una notazione più formale. A tale scopo, siano:

- C il terminale/workstation client;
- AS l'authentication server;

- V il server che fornisce il servizio richiesto;
- ID_C l'ID dell'utente che usa il client C ;
- ID_V l'ID del server V ;
- P_C la password dell'utente che usa il client C ;
- AD_C l'indirizzo di rete del client C ;
- K_V la chiave segreta condivisa tra AS e V ;
- \parallel l'operatore di concatenazione;
- $E_K(M)$ la cifratura di un messaggio M con una chiave simmetrica K .

Usando questi simboli, i tre messaggi scambiati dal protocollo sono:

- (1) $C \rightarrow AS: ID_C \parallel P_C \parallel ID_V$
- (2) $AS \rightarrow C: \text{Ticket}$
dove $\text{Ticket} = E_{K_V}(ID_C \parallel AD_C \parallel ID_V)$
- (3) $C \rightarrow V: ID_C \parallel \text{Ticket}$

Si ricorda che l'AS genera il ticket e invia il messaggio (2) solo se verifica, in base ai contenuti del messaggio (1), che le credenziali ID_C, P_C dell'utente sono corrette e che l'utente è autorizzato ad accedere al servizio identificato da ID_V .

3.2 Problemi

Questa prima soluzione ha due principali problemi:

- La password dell'utente è inviata all'AS in chiaro, dunque un attaccante potrebbe rubarla.
- Il ticket è utilizzabile una sola volta, quindi ogni volta che si vuole accedere a un servizio:
 - l'utente deve reinserire la propria password, il che è scomodo;
 - il client deve contattare l'AS per ottenere un nuovo ticket, perciò l'AS (e complessivamente l'esecuzione del protocollo di autenticazione) rischia di diventare un collo di bottiglia per tutte le richieste.

Inserire ogni volta la password è scomodo per l'utente, e dover contattare ogni volta

4 Autenticazione in due fasi

Il problema dell'invio della password può essere risolto mediante uno schema che non preveda il passaggio della password in chiaro, il quale verrà presentato a breve. Invece, per rimediare al fatto che il ticket sia utilizzabile una sola volta si separano le funzioni di autenticazione e autorizzazione, introducendo un nuovo server, il **Ticket Granting Server (TGS)**, che gestisce le autorizzazioni, mentre l'AS gestisce solo l'autenticazione. Grazie a tale separazione, l'autenticazione e l'autorizzazione possono avere tempi di validità diversi:

- l'utente si autentica una sola volta presso l'AS, ricevendo un *ticket di autenticazione* che è riutilizzabile per tutto il periodo di durata dell'autenticazione (qualche ora, o una sessione di lavoro, ecc.);
- ogni volta che deve accedere a un qualche server V , l'utente usa il ticket di autenticazione per richiedere al TGS un *ticket di autorizzazione* monouso spendibile presso il server V .

Prima di rilasciare un ticket di autenticazione l'AS verifica le credenziali dell'utente, mentre prima di rilasciare un ticket di autorizzazione il TGS verifica che l'utente (già autenticato dall'AS) sia autorizzato ad accedere al servizio richiesto.

Questo schema prende il nome di **autenticazione in due fasi**, anche se sarebbe forse più corretto parlare di una fase di autenticazione e una fase di autorizzazione.

4.1 Funzionamento del protocollo

Per implementare l'autenticazione in due fasi servono le seguenti chiavi simmetriche segrete:

- K_{TGS} , condivisa solo tra l'AS e il TGS (se ci sono più TGS, l'AS condivide una chiave diversa con ciascuno di essi);
- K_V , condivisa solo tra il TGS e un server V (se ci sono più server, il TGS condivide una chiave diversa con ciascuno di essi);
- K_C , condivisa solo tra l'AS e il client e diversa per ogni utente. Essa è derivata dalla password dell'utente e serve ad autenticare l'utente senza bisogno di inviare la password, risolvendo così il problema del passaggio delle password in chiaro.

Per prima cosa l'utente invia all'AS una richiesta di autenticazione, specificando il proprio ID e l>ID del TGS che gestisce le autorizzazioni per i servizi a cui l'utente vorrà accedere (dato che potrebbero esserci TGS diversi, ciascuno dei quali si occupa di un insieme di servizi diversi):

$$(1) C \rightarrow AS: ID_C \parallel ID_{\text{TGS}}$$

In risposta, l'AS genera un ticket di autenticazione chiamato **TGS ticket**, $\text{Ticket}_{\text{TGS}}$, che è cifrato con la chiave K_{TGS} condivisa solo da AS e TGS, in modo da dare al TGS la prova che tale ticket è stato effettivamente generato (ovvero che l'utente è stato autenticato) dall'AS. Tale ticket è definito come

$$\text{Ticket}_{\text{TGS}} = E_{K_{\text{TGS}}}(\text{ID}_C \parallel \text{AD}_C \parallel \text{ID}_{\text{TGS}} \parallel \text{time}_{\text{AS}} \parallel \text{lifetime}_1)$$

dove time_{AS} è un timestamp che indica il momento in cui il ticket è stato generato dall'AS e lifetime_1 indica il tempo di durata del ticket.

Prima di consegnare il ticket all'utente è necessario verificare che quest'ultimo conosca la password corretta. A tale scopo l'AS recupera la password corrispondente a ID_C , applica una qualche funzione per derivare da essa (in modo univoco/deterministico) una corrispondente chiave simmetrica K_C , e cifra ulteriormente con K_C il ticket (già cifrato con K_{TGS}). Il ticket *doppiamente cifrato* così ottenuto viene inviato al client:

$$(2) \text{ AS} \rightarrow \text{C}: E_{K_C}(\text{Ticket}_{\text{TGS}})$$

Quando il client riceve il ticket doppiamente cifrato, per poterlo spendere presso il TGS deve decifrare la cifratura esterna, ed è in grado di farlo solo se l'utente ha inserito la password corretta, dalla quale il client deriva infatti (mediante la stessa funzione applicata dal server) la chiave K_C necessaria per la decifratura. In sostanza, l'autenticazione dell'utente avviene in modo implicito: se l'utente non conosce la password corretta allora il client non riesce a estrarre dal messaggio (2) un ticket di autenticazione spendibile.

Con il messaggio (2) si conclude la fase di autenticazione. Quanto poi l'utente vuole accedere a un servizio si passa alla fase di autorizzazione, che il client avvia inviando al TGS un messaggio contenente il proprio ID, l'ID del server V a cui vuole accedere e il TGS ticket precedentemente ottenuto dall'AS:

$$(3) \text{ C} \rightarrow \text{TGS}: \text{ID}_C \parallel \text{ID}_V \parallel \text{Ticket}_{\text{TGS}}$$

Il TGS usa la chiave K_{TGS} per decifrare il $\text{Ticket}_{\text{TGS}}$ ricevuto, ottenendo così la prova che l'utente è stato autenticato dall'AS, poi verifica se l'utente ha l'autorizzazione per il servizio richiesto, e se sì genera un ticket di autorizzazione Ticket_V che l'utente potrà spendere presso il server V . Tale ticket ha contenuti analoghi a quelli del ticket di autenticazione ed è cifrato con la chiave K_V che solo il TGS e V condividono:

$$\text{Ticket}_V = E_{K_V}(\text{ID}_C \parallel \text{AD}_C \parallel \text{ID}_V \parallel \text{time}_{\text{TGS}} \parallel \text{lifetime}_2)$$

Dopo averlo generato, il TGS invia questo ticket al client:

$$(4) \text{ TGS} \rightarrow \text{C}: \text{Ticket}_V$$

Infine, il client presenta il Ticket_V al server V ,

$$(5) \text{ C} \rightarrow \text{V}: \text{ID}_C \parallel \text{Ticket}_V$$

che ne valida la correttezza decifrandolo con K_V e in caso positivo consente al client di accedere al servizio.

Il TGS ticket può essere ripresentato più volte al TGS, entro il periodo di validità indicato da time_{AS} e lifetime_1 . Invece, il ticket di autorizzazione Ticket_V è pensato per essere presentato una sola volta, ma ha comunque un certo tempo di validità, dunque contiene anch'esso un timestamp di emissione e un lifetime, come mostrato poco fa.

Riassumendo, i messaggi scambiati sono i seguenti:

- fase di autenticazione:
 - (1) $C \rightarrow AS: ID_C \parallel ID_{TGS}$
 - (2) $AS \rightarrow C: E_{K_C}(\text{Ticket}_{TGS})$
dove $\text{Ticket}_{TGS} = E_{K_{TGS}}(ID_C \parallel AD_C \parallel ID_{TGS} \parallel \text{time}_{AS} \parallel \text{lifetime}_1)$
- fase di autorizzazione:
 - (3) $C \rightarrow TGS: ID_C \parallel ID_V \parallel \text{Ticket}_{TGS}$
 - (4) $TGS \rightarrow C: \text{Ticket}_V$
dove $\text{Ticket}_V = E_{K_V}(ID_C \parallel AD_C \parallel ID_V \parallel \text{time}_{TGS} \parallel \text{lifetime}_2)$
 - (5) $C \rightarrow V: ID_C \parallel \text{Ticket}_V$

In questo schema sono presenti due aspetti chiave che si ritroveranno anche nel vero protocollo Kerberos:

- Il servizio di autenticazione e il servizio di autorizzazione danno prova dell'avvenuta autenticazione / autorizzazione mediante un ticket cifrato con una chiave simmetrica condivisa solo tra il servizio che genera il ticket e il servizio dove tale ticket dovrà essere speso.
- L'autenticazione dell'utente avviene in modo implicito sfruttando un'ulteriore cifratura del TGS ticket con una chiave derivata dalla password dell'utente.

4.2 Problemi

Il protocollo appena descritto ha due principali problemi:

- Un attaccante potrebbe rubare il Ticket_{TGS} o il Ticket_V e successivamente ripresentarlo (entro il suo lifetime), cioè effettuare un attacco a replay. Siccome i ticket contengono l'indirizzo di rete AD_C della macchina client che li ha richiesti, per poterli spendere l'attaccante dovrebbe o rimandarli dalla stessa macchina (ad esempio aspettando che essa si liberi, che l'utente smetta di usarla) oppure mandarli da un'altra macchina falsificando l'indirizzo di rete (ad esempio, nel pacchetto IP l'indirizzo del mittente può essere liberamente modificato se non sono implementati servizi di sicurezza come IPSec al livello di rete).

- Il server V non è autenticato sul client: quando l'utente, dopo aver ottenuto l'autorizzazione all'accesso, inizia a comunicare con il server per usufruire del servizio fornito, inviando una o più richieste e ricevendo una o più risposte, non ha prova che sia effettivamente il server V a ricevere le richieste e generare le risposte. Se un attaccante facesse in modo che i messaggi dell'utente siano inviati a un server falso, riuscirebbe a impedire la comunicazione con il vero server e potrebbe rubare informazioni confidenziali contenute nelle richieste.

5 Autenticatori e chiavi di sessione

Per impedire gli attacchi a replay si introduce un **autenticatore** che consente ai server TGS e V di verificare che l'utente che presenta il ticket è effettivamente quello per il quale il ticket è stato emesso. Al fine di generare e verificare questi autenticatori usando solo la cifratura simmetrica servono delle chiavi condivise tra il client e i server TGS e V .

Invece, per autenticare il server V , cioè realizzare la mutua autenticazione tra client e server, e anche per fornire confidenzialità tra C , TGS e V si introducono delle **chiavi di sessione**. Tali chiavi sono inoltre le stesse che vengono usate per gli autenticatori.

La condivisione delle chiavi di sessione si basa sul protocollo Needham-Schroeder a chiave segreta. Infatti, nell'introdurre Kerberos si era detto che esso si basa su Needham-Schroeder, ma il protocollo presentato finora è ben diverso da Needham-Schroeder:

Needham-Schroeder a chiave segreta	Autenticazione in due fasi
(1) $A \rightarrow S: ID_A \parallel ID_B \parallel N_a$	(1) $C \rightarrow AS: ID_C \parallel ID_{TGS}$
(2) $S \rightarrow A: E_{K_{AS}}(N_a \parallel K_{AB} \parallel ID_B$ $\parallel E_{K_{BS}}(K_{AB} \parallel ID_A))$	(2) $AS \rightarrow C: E_{K_C}(\text{Ticket}_{TGS})$ dove $\text{Ticket}_{TGS} = E_{K_{TGS}}(\dots)$
(3) $A \rightarrow B: E_{K_{BS}}(K_{AB} \parallel ID_A)$	(3) $C \rightarrow TGS: ID_C \parallel ID_V \parallel \text{Ticket}_{TGS}$
(4) $B \rightarrow A: E_{K_{AB}}(N_b)$	(4) $TGS \rightarrow C: \text{Ticket}_V$ dove $\text{Ticket}_V = E_{K_V}(\dots)$
(5) $A \rightarrow B: E_{K_{AB}}(N_b - 1)$	(5) $C \rightarrow V: ID_C \parallel \text{Ticket}_V$

ad esempio, esso prevede la partecipazione di quattro attori (C , AS , TGS e V) invece che tre (A , B e S). Ci sono però alcune analogie tra i due protocolli:

- Il messaggio (2) dell'autenticazione in due fasi è cifrato due volte, come lo è $K_{AB} \parallel ID_A$ nel messaggio (2) di Needham-Schroeder.
- Se si guardano separatamente le due fasi di autenticazione e autorizzazione, includendo il messaggio (3) in entrambe, allora ciascuna fase ha esattamente tre attori ed è molto simile ai primi tre passi di Needham-Schroeder: il client C corrisponde all'host A , il server che genera il ticket (AS nella prima fase e TGS nella seconda) corrisponde al server S e il server che riceve il ticket (TGS nella prima fase e V

nella seconda) corrisponde all'host B . Manca però la doppia cifratura del Ticket_V restituito dal server nella seconda fase, cioè nel messaggio (4).

Con l'introduzione delle chiavi di sessione necessarie per risolvere i problemi discussi prima il protocollo diventa ancora più simile a Needham-Schroeder, perché in ciascuna delle due fasi il server che genera il ticket genera anche una chiave di sessione da condividere tra il client e il server dove il ticket verrà speso.

6 Protocollo Kerberos v4

Con l'introduzione degli autenticator e delle chiavi di sessione si arriva finalmente al protocollo completo Kerberos versione 4. Esso usa le tre chiavi persistenti (cioè definite in fase di configurazione e usate per tutte le sessioni) K_C , K_{TGS} e K_V , già impiegate nel precedente protocollo semplificato, più due chiavi di sessione (che sono generate insieme ai ticket e rimangono valide solo per la durata dei ticket):

- $K_{C,\text{TGS}}$, la chiave di sessione tra C e TGS, generata dall'AS e trasmessa in modo sicuro solo a C e TGS;
- $K_{C,V}$, la chiave di sessione tra C e V , generata dal TGS e trasmessa in modo sicuro solo a C e V .

L'esecuzione del protocollo inizia con la richiesta di autenticazione da parte del client, che rispetto a prima comprende in più il timestamp time_C della richiesta:

(1) $C \rightarrow \text{AS}: \text{ID}_C \parallel \text{ID}_{\text{TGS}} \parallel \text{time}_C$

L'AS risponde con un messaggio con doppia cifratura, simile a quello di Needham-Schroeder:

(2) $\text{AS} \rightarrow C: E_{K_C}(K_{C,\text{TGS}} \parallel \text{ID}_{\text{TGS}} \parallel \text{time}_{\text{AS}} \parallel \text{lifetime}_1 \parallel \text{Ticket}_{\text{TGS}})$
dove $\text{Ticket}_{\text{TGS}} = E_{K_{\text{TGS}}}(K_{C,\text{TGS}} \parallel \text{ID}_C \parallel \text{AD}_C \parallel \text{ID}_{\text{TGS}} \parallel \text{time}_{\text{AS}} \parallel \text{lifetime}_1)$

Tale messaggio contiene il $\text{Ticket}_{\text{TGS}}$ doppiamente cifrato, insieme a delle altre informazioni che, essendo cifrate solo con K_C , possono essere lette dal client:

- la chiave di sessione $K_{C,\text{TGS}}$;
- l'ID del TGS per cui il ticket è valido e le informazioni sul periodo di validità del ticket (time_{AS} e lifetime_1).

Il $\text{Ticket}_{\text{TGS}}$, cifrato prima con K_{TGS} e poi con K_C , contiene invece le informazioni che servono al TGS:

- la stessa chiave di sessione $K_{C,\text{TGS}}$ presente nella anche parte esterna del messaggio (come in Needham-Schroeder, la chiave di sessione è inserita sia nella cifratura esterna che in quella interna);

- le informazioni sull'utente, sulla macchina e sul TGS per cui il ticket è stato emesso e sul periodo di validità del ticket stesso, uguali a quelle presenti nella versione semplificata del protocollo.

Se l'utente inserisce la password corretta, il client deriva da essa la chiave K_C corretta e la usa per decifrare il messaggio (2), ottenendo così la chiave di sessione $K_{C,TGS}$ e il Ticket_{TGS} .

Quando poi l'utente vuole accedere a un determinato servizio, il client manda al TGS un messaggio

$$(3) C \rightarrow TGS: ID_V \parallel \text{Ticket}_{TGS} \parallel \text{Authenticator}_C$$

dove $\text{Authenticator}_C = E_{K_{C,TGS}}(ID_C \parallel AD_C \parallel \text{time}_C)$

contenente:

- l'identificatore ID_V del server a cui vuole accedere;
- il TGS ticket precedentemente ottenuto dall'AS;
- l'autenticatore Authenticator_C , calcolato cifrando con la chiave di sessione $K_{C,TGS}$ l'ID dell'utente, l'indirizzo di rete del client e un timestamp time_C .

Il TGS usa innanzitutto la propria chiave K_{TGS} per decifrare il Ticket_{TGS} , ottenendo così la chiave di sessione $K_{C,TGS}$ e le varie altre informazioni contenute nel ticket. Successivamente usa la chiave $K_{C,TGS}$ appena acquisita per decifrare l'autenticatore Authenticator_C . Se la decifratura avviene correttamente (e i valori di ID_C , AD_C e time_C sono corretti) il TGS ha la prova che il client conosca la chiave di sessione $K_{C,TGS}$. Inoltre, siccome per ottenere $K_{C,TGS}$ il client deve aver decifrato la risposta dell'AS cifrata con K_C , e solo l'utente che ha richiesto all'AS il Ticket_{TGS} conosce la password da cui è derivata K_C , si ha la prova che il Ticket_{TGS} sia stato inviato al TGS dallo stesso utente che ha richiesto tale ticket.

Se un attaccante volesse effettuare l'attacco a replay discusso prima, adesso non potrebbe più presentare solo un Ticket_{TGS} rubato, ma dovrebbe anche presentare un Authenticator_C valido. Essendo cifrato con la chiave di sessione $K_{C,TGS}$, un autenticatore rubato sarebbe riutilizzabile solo durante la stessa sessione di autenticazione, ma anche nella stessa sessione il TGS si accorgerebbe, grazie al timestamp presente in Authenticator_C , che l'autenticatore non è "fresco", non è stato generato per la richiesta di autorizzazione corrente, e allora non lo considererebbe valido, bloccando così l'attacco.

Tornando alla normale esecuzione del protocollo, il TGS verifica che il ticket sia valido (ad esempio non scaduto) e che l'utente sia autorizzato ad accedere al servizio richiesto, poi risponde al client con un altro messaggio doppiamente cifrato:

$$(4) TGS \rightarrow C: E_{K_{C,TGS}}(K_{C,V} \parallel ID_V \parallel \text{time}_{TGS} \parallel \text{Ticket}_V)$$

dove $\text{Ticket}_V = E_{K_V}(K_{C,V} \parallel ID_C \parallel AD_C \parallel ID_V \parallel \text{time}_{TGS} \parallel \text{lifetime}_2)$

Esso è composto da

- il Ticket_V , contenente una nuova chiave di sessione $K_{C,V}$ e cifrato con K_V in modo che solo il server V possa leggerlo,
- l'ID del server V presso cui il Ticket_V può essere speso e il timestamp di emissione di tale ticket (ma non il lifetime del ticket, a differenza del messaggio (2), perché il Ticket_V è pensato per essere speso immediatamente e una volta sola dal client, che dunque non ha bisogno di conoscerne il lifetime),
- la nuova chiave di sessione $K_{C,V}$ (anche qui essa è presente sia nella cifratura esterna che in quella interna),

il tutto cifrato con la chiave di sessione $K_{C,TGS}$. Si osservi che la doppia cifratura di questo messaggio è resa possibile dalla condivisione della chiave di sessione in fase di autenticazione. Infatti, il protocollo Needham-Schroeder su cui quest'interazione si basa richiede che il server (qui TGS) condivida una chiave con ciascuno dei due client (qui C e V), mentre in assenza della chiave di sessione generata nella fase 1 il TGS avrebbe una chiave condivisa con V (K_V) ma non una condivisa con C .

Quando il client riceve il messaggio (4) lo decifra con $K_{C,TGS}$, ottenendo così la seconda chiave di sessione $K_{C,V}$ e il Ticket_V . Quest'ultimo viene inviato al server V per richiedere l'accesso al servizio, insieme a un autenticatore come quello del messaggio (2), con il quale il client dimostra di conoscere la chiave $K_{C,V}$, ovvero di essere il richiedente del Ticket_V :

$$(5) \quad C \rightarrow V: \text{Ticket}_V \parallel \text{Authenticator}_V \\ \text{dove } \text{Authenticator}_V = E_{K_V}(\text{ID}_C \parallel \text{AD}_C \parallel \text{new_time}_C)$$

La cifratura di Authenticator_V con la chiave di sessione K_V , valida solo per la durata del Ticket_V , impedisce il replay del messaggio tra sessioni di autorizzazione diverse, mentre il timestamp new_time_C contenuto nell'autenticatore, diverso dal precedente timestamp time_C , impedisce a un attaccante di rubare l'autenticatore Authenticator_C inviato insieme al Ticket_{TGS} e riutilizzarlo per autenticare l'invio del Ticket_V .

Il server V decifra il Ticket_V con K_V , controlla che sia valido e usa la chiave $K_{C,V}$ da esso estratta per validare anche l'autenticatore. Grazie agli autenticatori si è risolto il problema degli attacchi a replay, ma rimane ancora il problema di autenticare il server sul client. In effetti, ripensando al protocollo Needham-Schroeder, dopo i tre messaggi per la condivisione della chiave ne vengono scambiati altri due per realizzare l'autenticazione: uno dei due host invia all'altro un nonce cifrato con la chiave di sessione e l'altro risponde con il nonce decifrato, modificato e ricifrato. Nel caso di Kerberos il timestamp new_time_C contenuto in Authenticator_V può già essere usato come nonce, dunque per autenticare il server sul client è sufficiente manca solo un messaggio equivalente all'ultimo messaggio di Needham-Schroeder:²

²Nella fase 1 non serve invece un analogo messaggio per autenticare il TGS sul client, dato che tale autenticazione avviene già "naturalmente" nel momento in cui il TGS usa la propria chiave K_{TGS} per estrarre la chiave di sessione $K_{C,TGS}$ dal Ticket_{TGS} e poi cifra con la chiave di sessione il messaggio di risposta (4) mandato al client.

(6) $V \rightarrow C: E_{K_{C,V}}(\text{new_time}_C + 1)$

il server modifica il timestamp (precedentemente decifrato per verificare l'autenticatore) e lo invia al client ricifrato con la stessa chiave di sessione $K_{C,V}$ con cui era cifrato Authenticator_V .

Decifrando con $K_{C,V}$ quest'ultimo messaggio e verificando di ottenere il timestamp modificato corretto, il client ha la prova che il server conosca la chiave di sessione $K_{C,V}$, ovvero che il server possieda la chiave K_V necessaria per estrarre dal Ticket_V tale chiave di sessione. Siccome solo V condivide con il TGS la chiave K_V , ciò dimostra che il server con cui C sta comunicando è veramente V . Si è così effettuata l'autenticazione del server sul client.

Riassumendo, i messaggi scambiati nell'esecuzione del protocollo Kerberos v4 sono:

- autenticazione con l'AS per ottenere il TGS ticket:
 - (1) $C \rightarrow \text{AS}: \text{ID}_C \parallel \text{ID}_{\text{TGS}} \parallel \text{time}_C$
 - (2) $\text{AS} \rightarrow C: E_{K_C}(K_{C,\text{TGS}} \parallel \text{ID}_{\text{TGS}} \parallel \text{time}_{\text{AS}} \parallel \text{lifetime}_1 \parallel \text{Ticket}_{\text{TGS}})$
dove $\text{Ticket}_{\text{TGS}} = E_{K_{\text{TGS}}}(K_{C,\text{TGS}} \parallel \text{ID}_C \parallel \text{AD}_C \parallel \text{ID}_{\text{TGS}} \parallel \text{time}_{\text{AS}} \parallel \text{lifetime}_1)$
- richiesta al TGS per ottenere il ticket di servizio:
 - (3) $C \rightarrow \text{TGS}: \text{ID}_V \parallel \text{Ticket}_{\text{TGS}} \parallel \text{Authenticator}_C$
dove $\text{Authenticator}_C = E_{K_{C,\text{TGS}}}(\text{ID}_C \parallel \text{AD}_C \parallel \text{time}_C)$
 - (4) $\text{TGS} \rightarrow C: E_{K_{C,\text{TGS}}}(K_{C,V} \parallel \text{ID}_V \parallel \text{time}_{\text{TGS}} \parallel \text{Ticket}_V)$
dove $\text{Ticket}_V = E_{K_V}(K_{C,V} \parallel \text{ID}_C \parallel \text{AD}_C \parallel \text{ID}_V \parallel \text{time}_{\text{TGS}} \parallel \text{lifetime}_2)$
- richiesta al server per ottenere il servizio:
 - (5) $C \rightarrow V: \text{Ticket}_V \parallel \text{Authenticator}_V$
dove $\text{Authenticator}_V = E_{K_{C,V}}(\text{ID}_C \parallel \text{AD}_C \parallel \text{new_time}_C)$
 - (6) $V \rightarrow C: E_{K_{C,V}}(\text{new_time}_C + 1)$

7 Differenze tra v4 e v5

Nel passaggio dalla versione 4 alla versione 5 sono stati apportati molti miglioramenti sia all'implementazione che al protocollo. I principali miglioramenti, alcuni dei quali verranno approfonditi a breve, sono i seguenti:

- nuovi algoritmi crittografici;
- lifetime dei ticket flessibili, rappresentati da un timestamp di inizio e uno di fine invece che da un semplice valore di durata;
- definizione di una sorta di sintassi per i nomi degli utenti e degli host;
- possibilità di **autenticare tra diversi realm**.

7.1 Nomi degli utenti

In Kerberos si chiamano *principal* gli utenti e gli host (o più in generale gli attori) che partecipano al protocollo.

Nella versione 5 i nomi dei principal possono essere definiti specificando più componenti, e non solo username e indirizzo del client. Generalmente, il nome di un principal è diviso in tre parti:

- *primary*, che nel caso di un utente è lo username, mentre nel caso di un server è la stringa “host”;
- *instance*, che nel caso di un utente può essere nullo oppure il ruolo dell’utente, mentre nel caso di un server è il nome dell’host;
- *realm*, il nome del realm (dominio) Kerberos.

Il formato è *primary/instance@REALM*, dunque ad esempio:

- il nome di un utente potrebbe essere `barbara/admin@LAB.EDU`;
- il nome di un server potrebbe essere `host/printerHP@LAB.EDU`.

7.2 Autenticazione tra diversi realm

Un **realm** è l’insieme di server/machine e di utenti gestiti dallo stesso server Kerberos. Per reti grandi e complesse un solo server Kerberos potrebbe non essere sufficiente, dunque tali reti vengono tendenzialmente organizzate in più realm, ciascuno con un server Kerberos che ha le chiavi degli utenti, del TGS e dei server presenti in tale realm. Da un lato, ciò permette di scalare molto meglio, evitando colli di bottiglia, ma dall’altro introduce il problema di come utenti registrati sul server Kerberos di un realm possano autenticarsi in realm diversi. Il miglioramento più significativo introdotto in Kerberos v5 è proprio una soluzione a questo problema.

Perché un utente del realm realm_1 possa accedere a un server V di un altro realm realm_2 deve innanzitutto esistere una relazione di fiducia tra i server Kerberos dei due realm (configurata da un amministratore di sistema). Poi, si eseguono i seguenti passi:

1. L’utente si autentica presso l’AS del proprio realm (l’unico AS che conosce la password di questo utente), ottenendo così il $\text{Ticket}_{\text{TGS}_1}$ per il TGS del proprio realm, TGS_1 .
2. L’utente richiede al TGS_1 il ticket per il server V . Il TGS_1 riconosce che V appartiene al realm_2 , dunque non genera un ticket per V , ma invece genera un $\text{Ticket}_{\text{TGS}_2}$ da inviare al TGS del realm_2 , TGS_2 . Tale ticket è cifrato con una chiave condivisa tra TGS_1 e TGS_2 (tale chiave modella appunto la relazione di fiducia tra i server Kerberos dei due realm).

3. L'utente presenta il $\text{Ticket}_{\text{TGS}_2}$ al TGS_2 e richiede il ticket per il server V . Allora, il TGS_2 verifica se l'utente è autorizzato ad accedere al server e, in caso positivo, emette il Ticket_V richiesto.
4. L'utente utilizza infine il Ticket_V per accedere al server V del realm_2 .