Primo approfondimento

Il protocollo Kerberos

Il protocollo Kerberos

- Kerberos 4
  - Descrizione del protocollo attraverso una lettura “commentata” del “Dialogo in 4 scene” di Bill Bryant e Theodore Ts'o
  - I limiti della versione 4 del protocollo e della sua implementazione del MIT
- Kerberos 5
  - Differenze rispetto alla versione 4
  - Funzionamento in maggior dettaglio
  - Cross-Authentication

Kerberos 4

Designing an Authentication System: a Dialogue in Four Scenes

- http://web.mit.edu/kerberos/dialogue.html
- Copyright Notices:
  - Copyright 1988, 1997 Massachusetts Institute of Technology. All Rights Reserved.
  - Originally written by Bill Bryant, February 1988.
  - Cleaned up and converted to HTML by Theodore Ts'o, February, 1997. An afterword describing the changes in Version 5 of the Kerberos protocol was also added.
  - Permission to use, copy, modify, and distribute this documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the documentation without specific, written prior permission. M.I.T. makes no representations about the suitability of this documentation for any purpose. It is provided "as is" without express or implied warranty.

In Italiano

- http://www.securegroup.it/kerberos/psaut.htm
- Attenzione alla traduzione:
    - “clumsily” ⇒ “poco sicuro”

      Io lo tradurrei in “maldestramente”
    - “I take it your system isn't this stupid” ⇒ “Penso che il tuo sistema sia proprio stupido”

      “Presumo che il tuo sistema non sia così stupido”

Il disegno di un sistema di autenticazione: Dialogo in quattro scene

- Il sistema di Autenticazione viene man mano definito e rifinito nel corso delle 4 scene che compongono il dialogo. Il disegno scaturisce quasi "naturalmente" dalla sfida intellettuale dei due protagonisti-antagonisti:
  - Athena (una giovane ed ambiziosa esordiente nel campo dello sviluppo dei sistemi di rete)
  - Euripide (un vecchio e navigato sviluppatore di sistemi di rete)
- E' una gradevole e simpatica lettura che consiglio a tutti.

Antefatto (prima scena)

- Athena, infastidita dalla lentezza del sistema time-sharing su cui sta lavorando, immagina di risolvere tali problemi sostituendo il grosso calcolatore centrale con workstations personali, e racconta ad Euripide (che sta lavorando vicino a lei) questa sua idea.

- Il dialogo tra Athena ed Euripide evidenzia che Athena ha in mente un'architettura client-server.

...ed alla fine della prima scena...

Euripide:	Il tuo nuovo sistema basato su Workstation sembra davvero bello, Tina. Quando ci sarà, sai cosa farò? Cercherò di scoprire il tuo username e farò in modo che la mia workstation creda, che io sia te. Poi contatterò il tuo server di posta elettronica e leggerò la tua posta. Poi guarderò tutti i tuoi documenti personali, e magari li cancellerò tutti, e poi...
Athena:	Lo faresti davvero?
Euripide:	Certo! Come faranno tutti questi SERVER a scoprire che io non sono te?
Athena:	Oddio, non lo so. Credo che debba pensarci un po' su.
Euripide:	Lo credo anch'io. Fammi sapere quando hai scoperto come fare.

Regole del “gioco”

- L’utente interagisce con i vari servizi attraverso dei programmi opportuni: i CLIENT, ma Athena conduce il dialogo come i CLIENT fossero gli utenti.

Definizione del problema

Athena:	Bene. Per iniziare riassumo il problema che ho risolto. In un ambiente di rete aperto, i computer che forniscono servizi devono essere in grado di verificare l'identità della persona che ha richiesto il servizio. Se io contatto il mail server e gli chiedo la mia posta, il server deve essere in grado di verificare che io sia effettivamente chi dico di essere, giusto?
Euripide:	Giusto.

Athena:	Il problema potrebbe essere risolto maldestramente facendo in modo che il mail server richieda la mia password prima di permettermi di usarlo. Io provo la mia identità al server fornendogli la mia password.
Euripide:	Si. è davvero un sistema maldestro. Infatti con tale sistema ogni server deve conoscere la tua password, per poterla verificare. Se il sistema ha più di mille utenti, come quello che ci sarebbe qui, ogni server dovrebbe conoscere più di mille password. Inoltre, se tu vuoi cambiare la tua password dovrai contattare ogni server per notificarglielo. Presumo che il tuo sistema non sia così stupido.
Athena:	Il mio sistema non è stupido. Funziona così: Non solo gli utenti hanno una password, ma anche i servizi hanno una loro password. Ogni utente conosce la propria password, ogni servizio conosce la propria password, e c'è un SERVIZIO di AUTHENTICAZIONE (AS) che conosce TUTTE le sia quelle degli utenti, che quelle dei servizi. Il Servizio di Autenticazione conserva le password in un unico database centralizzato.

Authentication Service

- In una architettura CLIENT-SERVER, affinché un determinato SERVER possa essere sicuro dell'identità del CLIENT, o deve identificarlo direttamente, oppure può fidarsi di quello che gli dice un AUTHENTICATION SERVER.

Un nome per l'AS

- Dopo un breve battibecco sulle reali funzioni del "traghettatore" dell'Inferno Dantesco (o se volete della mitologia Greca), in mancanza di nomi migliori, i due decidono di chiamarlo Caronte.

Euripide:	Allora chiamiamo il servizio di autenticazione "Caronte" per ora.
Athena:	Ok. Credo che ora io debba descrivere il sistema, vero?

Un concetto nuovo: Il Service Ticket

- L’accesso ad un Servizio, avviene grazie ai seguenti tre passaggi:
  - Il CLIENT prova all’Authentication Server la propria identità (attraverso una coppia username-password).
  - L’Authentication Server, verificata l’identità del CLIENT, gli restituisce un SERVICE TICKET, che viene cifrato con la password del Servizio a cui si vuole accedere, e che contiene informazioni relative al CLIENT.
  - Il CLIENT usa questo SERVICE TICKET per richiedere l’accesso al Servizio desiderato.

Dentro il Service Ticket

- Il Service Ticket deve contenere almeno
  - UserName (in modo che il SERVER possa valutare se il CLIENT che richiede i suoi servizi è chi dice di essere)
  - ServiceName (in modo che il SERVER possa valutare la correttezza della decifratura)
  - NetworkAddress della WS in cui è iniziata la procedura (in modo da evitare che ST rubati possano essere usati da altri CLIENT)

$$TICKET = \{UserName, WS\_address, ServiceName\}_{\mathbf{K\_service}}$$

Euripide:	Sembra che nel tuo sistema io debba chiedere un biglietto ogni volta voglia utilizzare un qualunque servizio. In un giorno di lavoro io dovrò probabilmente accedere a molti servizi, spesso più volte. Ad esempio vorrò controllare la mia posta varie volte. Devo richiedere un nuovo biglietto ogni volta che devo guardare la mia posta? Se è così non sono sicuro che il tuo sistema mi piaccia.
Athena:	Ah . . . Beh, non vedo perché i biglietti non debbano essere riusabili. Se tu ottieni un biglietto per un server di posta, dovresti poterlo utilizzare più volte. Il client di posta potrebbe conservare una copia del biglietto e quando hai bisogno dello stesso servizio, questi può semplicemente inviare la copia del biglietto che conserva.

Da migliorare

- Un tale sistema, prevede che per ogni accesso ad un dato servizio, debba essere necessaria una richiesta all'AS
    - Tale problema si supera facendo in modo che il TICKET per un determinato servizio sia riutilizzabile dal CLIENT

Euripide:	Così va meglio. Ma io ho ancora problemi. Sembra che comunque io debba dare a Caronte la mia password ogni volta che voglio usare un servizio per il quale non ho un Ticket. Io effettuo un login e voglio accedere ai miei files. Invio una richiesta a Caronte per il ticket appropriato e questo significa che devo aver usato la mia password. Poi voglio leggere la mia posta. Un'altra richiesta a Caronte, devo immettere nuovamente la mia password. Voglio stampare sulla stampante di rete: un'altra richiesta a Caronte e, hai un'idea del disagio?
Athena:	Uh, Si, Ce l'ho
Euripide:	E se ciò non fosse abbastanza, c'è un altro problema molto serio che dovresti considerare. Ogni volta che invii la tua password a Caronte, su una rete insicura, io potrei monitorare la rete e copiare la password. Non essendo criptata in alcun modo, potrei utilizzarla a tuo nome, accedendo a qualunque servizio come se fossi tu.
A t h e n a : (sighs)	Questi sono problemi seri. Credo che ho bisogno di tornare al tavolo di progettazione per rivedere la situazione.

Problemi

- Un TICKET per ogni servizio richiede che il CLIENT debba contattare l'AS per ogni servizio da utilizzare.

- La password deve essere comunicata all'AS via rete ed IN CHIARO.

Dopo una notte insonne…

Athena:	Conviene partire con un riesame dei problemi. Li invertirò in modo che diventino requisiti del sistema.
Athena:	Il primo requisito: gli utenti devono inserire la password solo una volta, quando accendono la loro workstation ed iniziano la loro sessione di lavoro in rete. Questo implica che tu non devi inserire la tua password ogni volta che hai bisogno di un biglietto per un nuovo servizio. Il secondo requisito: le password non devono circolare attraverso la rete, almeno non in un formato leggibile o utilizzabile.
Euripide:	Perfetto.

Athena:	Iniziamo con il primo requisito: devi inserire la tua password solo una volta. Sono riuscita ad ottenere questo risultato inventando un nuovo servizio. E' chiamato "Servizio di concessione biglietti" (Ticket Granting Service), in breve TGS, un servizio che rilascia biglietti agli utenti che hanno già provato la loro identità a Caronte. Anche per utilizzare questo servizio è necessario un biglietto, chiamato "Biglietto per il servizio di rilascio biglietti" (Ticket Granting Ticket), in breve TGT. In pratica il TGT è una nuova versione di Caronte. E' un servizio che ha infatti accesso al database delle password degli utenti e dei servizi e ti fornisce un biglietto dopo aver verificato la tua identità. Solo che ti fornisce solo il biglietto valido per il nuovo Servizio che accorda i biglietti o TGS.

Athena:	Il sistema di autenticazione ora funziona così: accendi il computer ed inserisci il tuo nome utente ed una tua password. A questo punto un programma sul tuo computer, che possiamo chiamare kinit, contatta il computer server sul quale gira Caronte. Grazie ai dati immessi provi la tua identità a Caronte, il quale emette un biglietto, il famoso TGT (Ticket Granting Ticket), il biglietto che ti autorizza ad usare il servizio di concessione biglietti o TGS, che viene recuperato dal programma che abbiamo chiamato kinit. Ora se tu vuoi guardare la tua posta e non hai il relativo biglietto, puoi utilizzare il tuo TGT per ottenerlo, senza dovere reinserire la password.

Ticket-Granting Service

- Il Ticket-Granting Service è un servizio di rete che è capace di fornire dei TICKET per i vari servizi, ad utenti che hanno già dimostrato la loro identità all'AS, e che hanno un Ticket (Ticket-Granting Ticket) con il quale possono accedere ai servizi del TGS

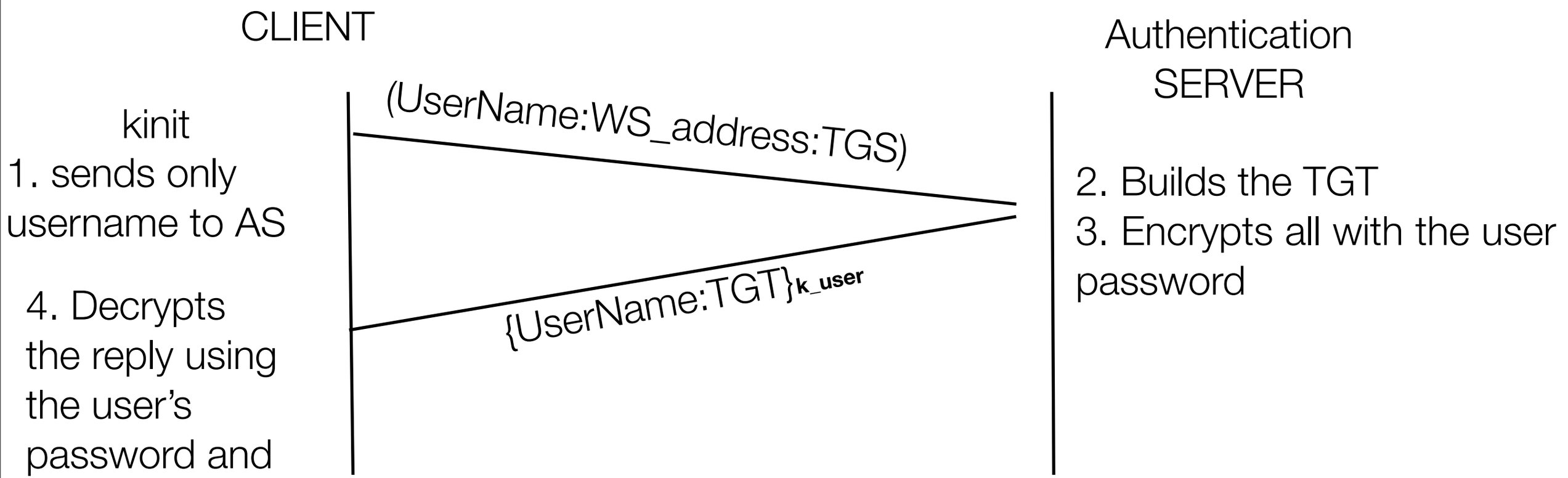
Euripide:	Dovrò richiedere un nuovo "TGT" ogni volta che devo accedere ad un nuovo servizio di rete?
Athena:	No. Ricordati che l'altra volta eravamo d'accordo che i biglietti possono essere riutilizzati. Una volta che hai acquisito un TGT, non hai più bisogno di acquisirne altri. Puoi utilizzare il tuo TGT per acquisirne quanti ne vuoi.
Euripide:	Va bene. Sembra che abbia senso. E siccome puoi riutilizzare i biglietti, una volta che il TGS ti ha fornito un biglietto per un particolare servizio, non hai più bisogno di riottenerne un'altro.
Athena:	Sii. Non è un sistema elegante?
Euripide:	Ok. Diamola per buona, per ora. . . A patto che tu non debba spedire la tua password in formato leggibile su queste rete totalmente insicura quando fai la richiesta per un TGT a Caronte.

Athena:	Come ho detto ho risolto anche quel problema. Quando ho detto che devo contattare Caronte per ottenere un TGT, si potrebbe pensare che io debba inviargli la password in formato leggibile attraverso la rete, fino al suo server. Ma può non essere così. Ecco che cosa succede in realtà. Quando tu inserisci il nome utente e la password, abbiamo detto che questi dati vengono gestiti da un programma che sta sulla workstation che si chiama kinit, il quale contatta Caronte a nome dell'utente per ottenere il TGT. Kinit non invia la password a Caronte, ma invia solo il nome utente.
Euripide:	Bene.

Athena:	Caronte utilizza il nome utente per recuperare la relativa password nel database. Poi Caronte prepara il pacco di dati che contengono il TGT. Prima di inviarti questi dati, Caronte utilizza la tua password, o meglio, la password del nome utente che hai inserito, come chiave per criptare l'intero pacchetto di dati che contiene il TGT. Il programma kinit sulla tua workstation riceve il TGT criptato con la tua password. Kinit tenta di decriptarlo con la password che tu inserisci, che non andrà mai nella rete, e non sarà memorizzata da nessuna parte. Se kinit riesce, l'autenticazione è positiva, avrai il TGT con cui potrai ottenere i biglietti per i servizi a cui hai diritto. Che ti sembra?

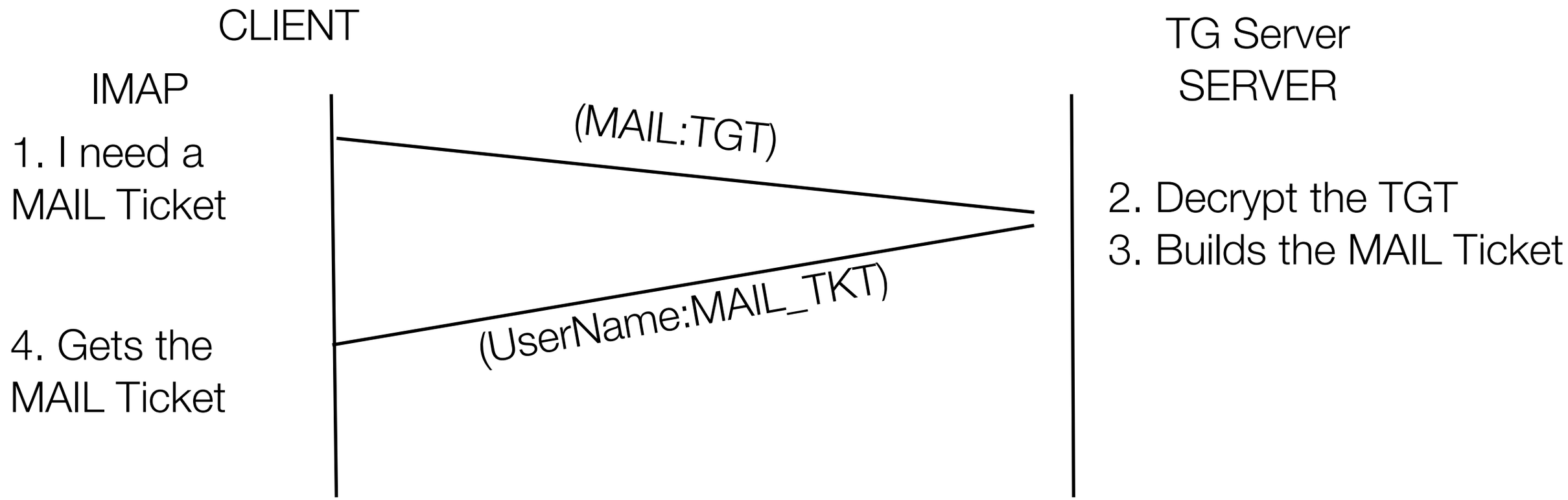
Che ci sembra?

1 - Ottengo il TGT



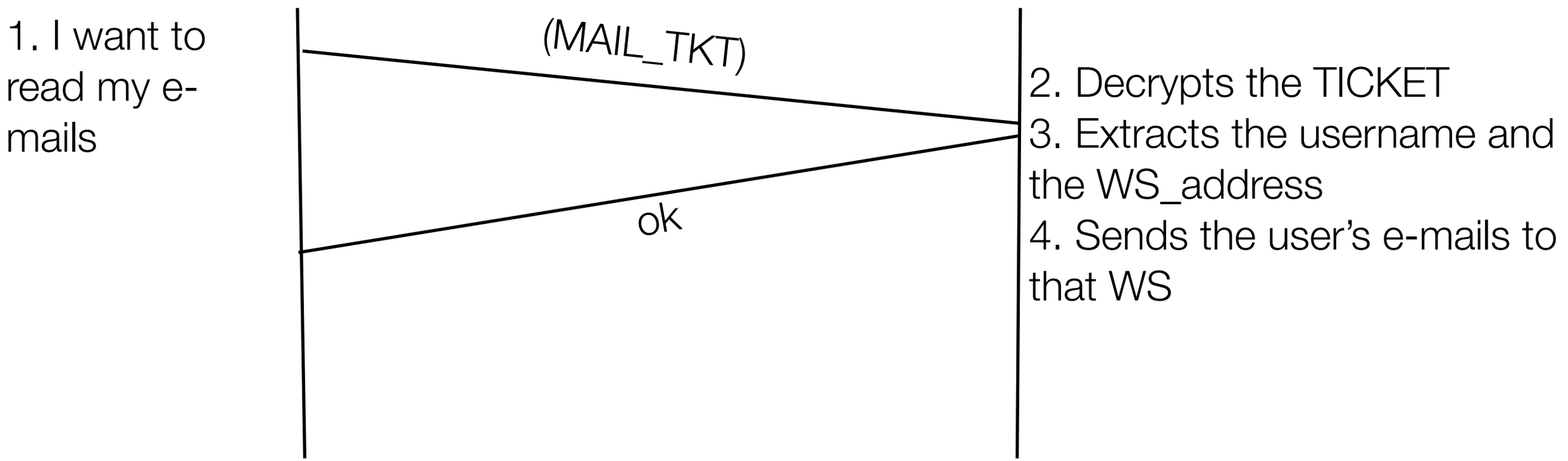
$$TGT = \{UserName, WS\_address, TGS\}_{K\_TGserver}$$

2 - Con il TGT ottengo il Ticket



$TGT = \{UserName,WS\_address,TGS\}_{K\_TGserver}$
$MAIL\_TKT = \{UserName,WS\_address,MAIL\}_{K\_MAILserver}$

3 - Uso il Ticket



MAIL_TKT = {UserName,WS_address,MAIL}$_{\mathbf{K\_MAILserver}}$

Tickets riutilizzabili?

- Fino ad ora abbiano assunto che i Tickets possano essere riutilizzati (per evitare la noia di dover inserire la password ad ogni richiesta) ma risulta evidente che se “perdo” un Ticket di questo tipo, chiunque lo “trovi” può leggere la mia posta… per sempre!

$$TKT = \{UserName, WS\_address, service, timestamp, lifespan\}_{\mathbf{K\_service}}$$

Tickets riutilizzabili!

- Ma anche se si definisce un tempo di vita “ragionevole” (ad esempio la durata media di una sessione di lavoro) ogni sessione di lavoro che dura meno del “tempo di vita” del Ticket può permetterne l’uso da parte di altri utenti

Athena:	Ho paura che siamo incappati in un problema serio (sospira)

…dopo un’altra notte insonne…

Athena:	Come al solito preferisco riformulare il problema. I biglietti possono essere riutilizzati, anche se entro un periodo breve di tempo, diciamo otto ore. Se qualcuno si impossessa dei biglietti e riesce ad usarli prima che scadano, noi non siamo in grado di fare nulla per fermarlo.
Euripide:	Esatto. Questo è il problema.
Athena:	Noi possiamo risolvere il problema se progettiamo dei biglietti che non possono essere riutilizzati.
Euripide:	Ma in questo modo dovrai ottenere un nuovo biglietto ogni volta che vorrai utilizzare un nuovo servizio, anche nella stessa sessione di lavoro.
Athena:	Vero. Infatti questa è una soluzione a dir poco sgradevole (Pausa) Ah, dov'ero rimasta...? (pensa per un momento)

Athena:	Va bene. Riepilogo il problema di nuovo, questa volta nella forma di un requisito. Un servizio di rete deve essere in grado di provare che la persona che ha il biglietto per accedere a quel servizio, è la stessa persona che ha ottenuto regolarmente il biglietto. Ripercorriamo di nuovo il processo di autenticazione e vediamo se riesco a trovare un modo efficace per illustrare la mia soluzione a questo problema. Io voglio utilizzare un dato servizio che si trova su una rete aperta ed insicura. Io accedo a quel servizio dalla mia workstation, grazie ad un idoneo programma CLIENT. Il CLIENT invia al servizio, attraverso la rete insicura, tre cose: il mio nome utente, l'indirizzo IP della mia workstation ed il mio biglietto per poter accedere al servizio.

MAIL_REQ = (UserName,WS_address,MAIL_TKT)

Athena:	Anche nel biglietto vi sono il nome utente della persona a cui è stato rilasciato legalmente il biglietto, l'indirizzo IP del computer utilizzato, la data ed ora del rilascio e la durata, il periodo di validità del biglietto dal momento del rilascio. Questi dati sono stati criptati da Caronte utilizzando la chiave del servizio stesso, e quindi solo il servizio può visionarne il contenuto.

MAIL_REQ = (UserName,WS_address,MAIL_TKT)
MAIL_TKT = {UserName, WS_address, timestamp, lifespan}$_{\mathbf{K\_MAILservice}}$

Athena:	… ma… Il servizio non è in grado di provare in maniera certa che chi sta usando un biglietto valido, a nome di un utente che lo ha regolarmente ottenuto, sia davvero il legittimo proprietario, perché, nel momento in cui il biglietto viene usato effettivamente, il servizio non condivide più alcun segreto con l'utente. … Così ho pensato l'altra notte, perché non dare a Caronte il compito di generare un "segreto" per l'utente, e fare in modo che utente e servizio condividano tale "segreto" ?

Athena:	Caronte dà una copia di questa session key all'utente ed una copia al servizio. Quando il servizio riceve un Ticket da un utente, esso può verificare l'identità dell'utente usando la session key
Euripide:	Aspetta un secondo. Come fa Caronte a dare ad entrambi la session key?

CARONTE REPLY = [Session_Key, TKT]
TKT = {Session_Key, UserName, WS_address, timestamp, lifespan}$_{\mathbf{K\_service}}$

Session Key

- Il problema dell'utilizzo da parte di altri di Tickets non ancora scaduti, si risolve permettendo a CLENT e SERVER di condividere uno stesso "segreto": la SESSION KEY.

- L'AS genera la SESSION KEY, e ne mette una copia nel Service Ticket (per il SERVER) ed una nel Ticket-Granting Ticket (per il CLIENT)

Authenticator

- Il CLIENT usa la propria copia della Session Key per cifrare l'Autenticatore, che viene inviato, insieme al TICKET, al SERVER in questione

- L'Authenticatore può essere usato solo una volta, e per questo ha un breve tempo di vita

$AUTHENTICATOR = \{UserName, WS\_address, lifespan, timestamp\}_{K\_session}$
$TKT = \{UserName, WS\_address, ServiceName, K\_session, lifespan, timestamp\}_{K\_server}$

Protezione della session key

- A questo punto sembrerebbe che basti appropriarsi della coppia

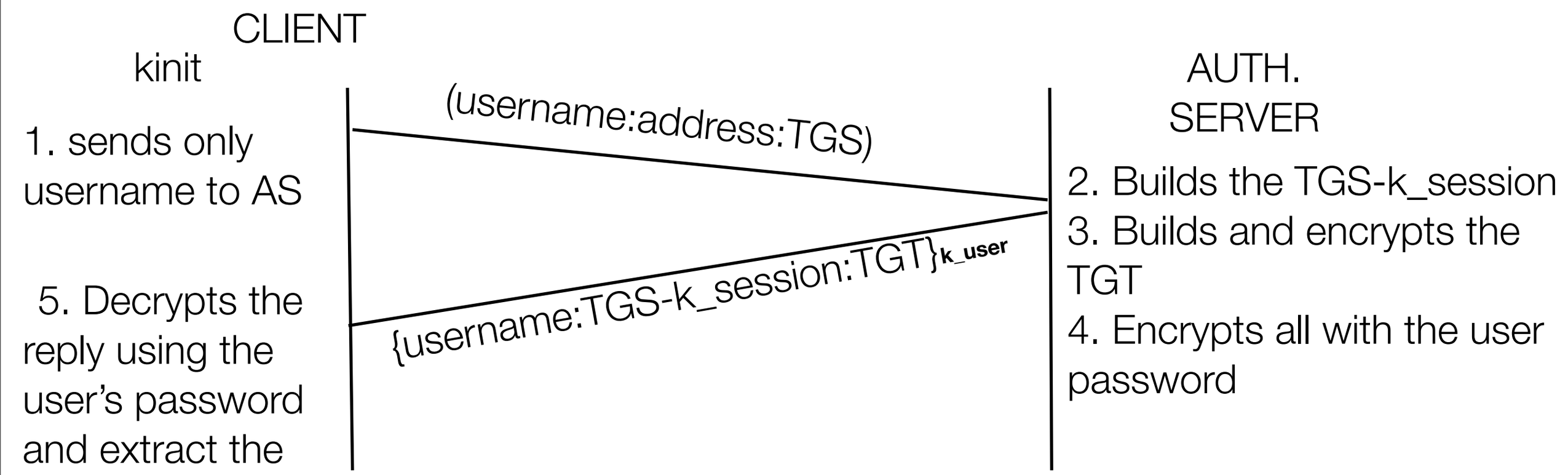
Session_Key | Ticket

per riuscire ad impersonare l'utente.

CARONTE REPLY = [Session_Key, TKT]
TKT = {Session_Key, UserName, WS_address, timestamp, lifespan}$_{\mathbf{K\_service}}$

Athena:	A questo ho pensato per un po' l'altra notte, ma credo che nel processo di acquisizione dei biglietti che ho delineato, sia evidente come non è possibile rubare gli autenticatori in quel modo. Tu siedi al computer, ed usi Kinit per ottenere il tuo TGT. Kinit ti richiede lo username, ed una volta ottenuto, lo manda a Caronte. Caronte trova la password associata allo username, e procede a costruire il TGT per te. Come parte di questo processo, Caronte crea la Session_Key che tu condividerai con il servizio di Ticket-Granting. Caronte mette una copia della Session_Key nel TGT, e mette la tua copia nel pacchetto di dati che sta per spedirti. Ma prima di efferruare la spedizione, Caronte cifra l'intero pacchetto usando la tua password. Caronte spedisce il pacchetto attraverso la rete. Chiunque copi il pacchetto, non lo potrà utilizzare poiché è crittografato con la tua password. In particolare nessuno può rubare la Session_Key del TGS. Kinit riceve il pacchetto e ti chiede la password. Se tu immetti la password corretta, kinit può decifrare il pacchetto ed accedere alla tua copia della Session_Key.

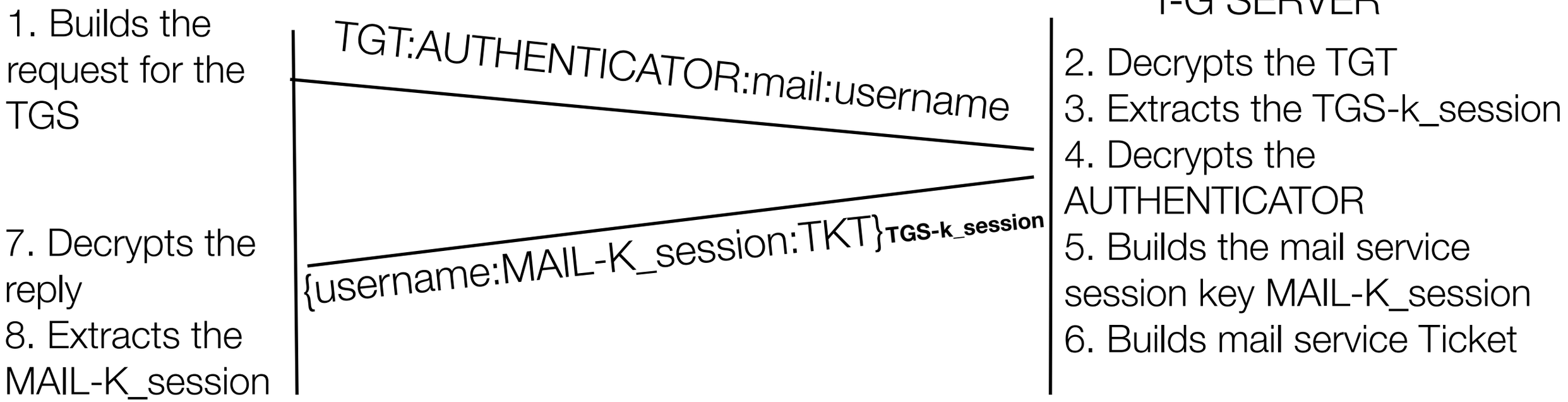
L’acquisizione del TGT



AUTHENTICATOR = {UserName,WS_address,timestamp}**TGS-k_session**
TGT = {UserName,WS_address,TGS,TGS-K_session,lifespan,timestamp}**K_TGserver**

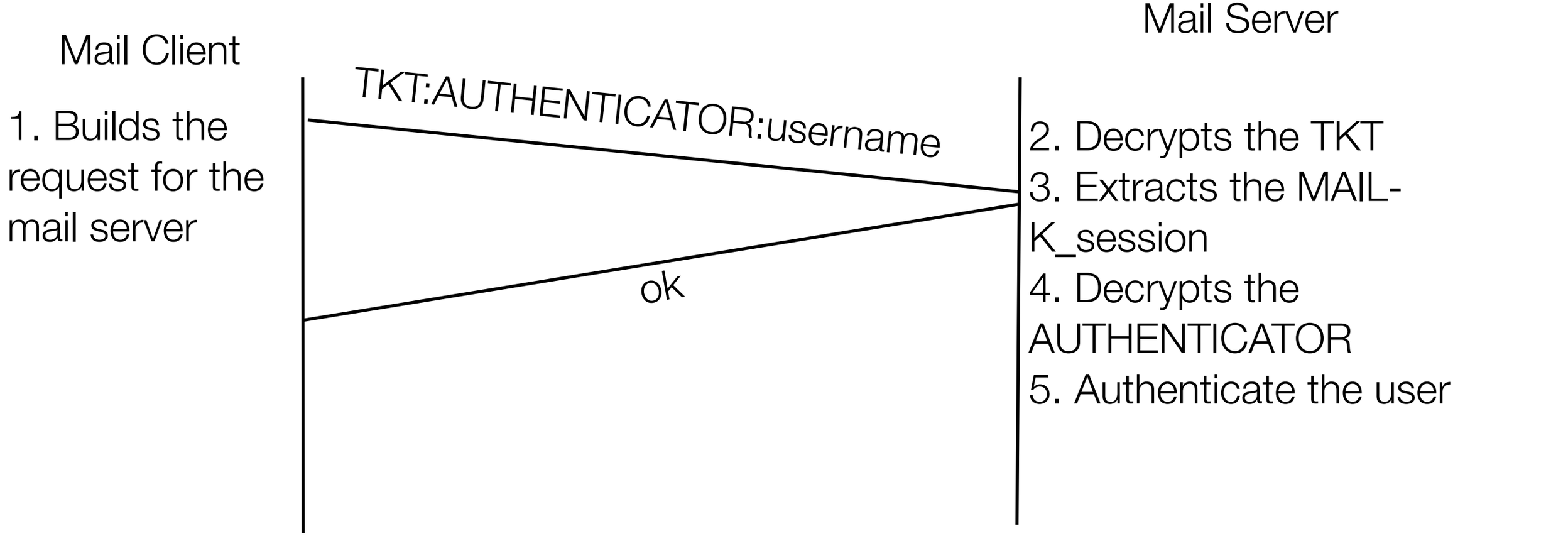
Athena:	Adesso che hai ottenuto la tua prima autenticazione, vuoi accedere alla tua posta. Fai partire il mail client. Questo cerca un Ticket valido per il servizio Mail, e non lo trova (dopo tutto non hai ancora provato ad accedere alla tua posta). Il Mail Client dece usare il TGT per chiedere al TGS un Ticket per il servizio di Mail. Il Mail Client costruisce un auteticatore per la transazione con il TGS e lo cifra con la tua copia della Session_Key del TGS. Il Client manda quindi a Caronte l'autenticatore, il TGT, il tuo username, l'indirizzo della tua WorkStation, ed il nome del servizio di posta. Il TGS riceve questo roba, e fa partire la verifica dell'autenticazione. Se tutto va a buon fine, il TGS otterrà una copia della Session_Key del TGS che condivide solo con te. Adesso il TGS costruisce per te un Ticket per il servizio di posta, e durante questo processo, crea una nuova Session_Key che tu condividerai con il servizio di posta. Il TGS prepara quindi il pacchetto da mandare ndietro alla tua workstation. Il pacchetto contiene il Ticket, e la tua copia della Session_Key per il servizio di posta. Ma prima di spedire il pacco, cifra il tutto usando la sua copia della Session_Key del TGS. Fatto questo, manda il pacchetto.

L’acquisizione del TICKET



AUTHENTICATOR = {UserName,WS_address,timestamp}**TGS-k_session**
TGT = {UserName,WS_address,TGS,TGS-K_session,lifespan,timestamp}**K_TGserver**
TKT = {UserName,WS_address,MAIL,MAIL-K_session,lifespan,timestamp}**K_MAILserver**

L’utilizzo del Ticket



AUTHENTICATOR = {UserName,WS_address,timestamp}**MAIL-K_session**
TKT = {UserName,WS_address,mail,MAIL-K_session,lifespan,timestamp}**K_MAILserver**

Athena:	Se qualcuno copiasse il pacchetto contenente il biglietto per la tua posta e la corrispondente chiave di sessione, non potrebbe usarla, in quanto solo tu ed il TGS conoscete la vostra chiave di sessione condivisa. Quindi il ladro non potrà venire a conoscenza della chiave di sessione condivisa con il server di posta, e quindi non potrà mai utilizzare eventuali biglietti validi rubati. Con questo sistema, ritengo che noi siamo al sicuro. Che ne pensi?
Euripide:	Forse.
Athena:	Forse! E' tutto quello che sai dire!
Euripide:	(ridendo) Non arrabbiarti. Dovresti avermi imparato a conoscere dopo tutto questo tempo. Forse sono stato cattivo, e tu sei stata sveglia per metà notte.
Athena:	Pthhhhh!

Euripide:	Va bene, per tre quarti della notte. Comunque il sistema comincia a sembrarmi accettabile. Questo meccanismo della chiave di sessione risolve un problema a cui ho pensato l'altra notte. Il problema della autenticazione reciproca. Pausa.

Euripide:	(Pausa.) Ti dispiace se parlo io per qualche minuto?
Athena:	Fai pure.
Euripide:	Sei così gentile. (Euripide si schiarisce la voce) L'altra notte, probabilmente nel momento in cui le visioni delle chiavi di sessione e degli autenticatori danzavano nella tua testa, io stavo cercando di trovare nuovi problemi nel sistema, e ne ho trovato uno che ho pensato che fosse davvero molto serio. Te lo illustro per mezzo del seguente scenario.

Euripide:	Immagina di essere stufa del tuo attuale lavoro, e sei arrivata alla determinazione che è nel tuo interesse cambiarlo. Vuoi stampare il tuo curriculum aggiornato direttamene sulla stampante laser dell'azienda "wizz-bang", in modo che i cacciatori di cervelli che vi sono la ed in generale i potenziali datori di lavoro possano venire a conoscenza delle tue capacità. Quindi tu dai i comandi appropriati per stampare il documento su quella stampante. Il client ottiene il biglietto per il servizio di stampa in grado di stampare su quella stampante. Almeno tu credi che stai stampando sulla stampante giusta. Ma in realtà potrebbe non essere così. Supponi che un qualche individuo senza scrupoli -- diciamo il tuo capo -- abbia reindirizzato la tua richiesta ed i tuoi biglietti verso un'altra stampante, diciamo quella che sta nel suo ufficio. Ed abbia configurato il suo server di stampa per fregarsene dei tuoi biglietti e del loro contenuto, e per stampare comunque ciò che hai chiesto, in realtà ad un'altra stampante. Quindi il servizio restituisce un messaggio di autenticazione falso, nonostante i biglietti non siano rivolti a lui, lui procede con l'autenticazione, invia al client il segnale di via libera per la stampa, ed il tuo documento viene sottratto in maniera fraudolenta.

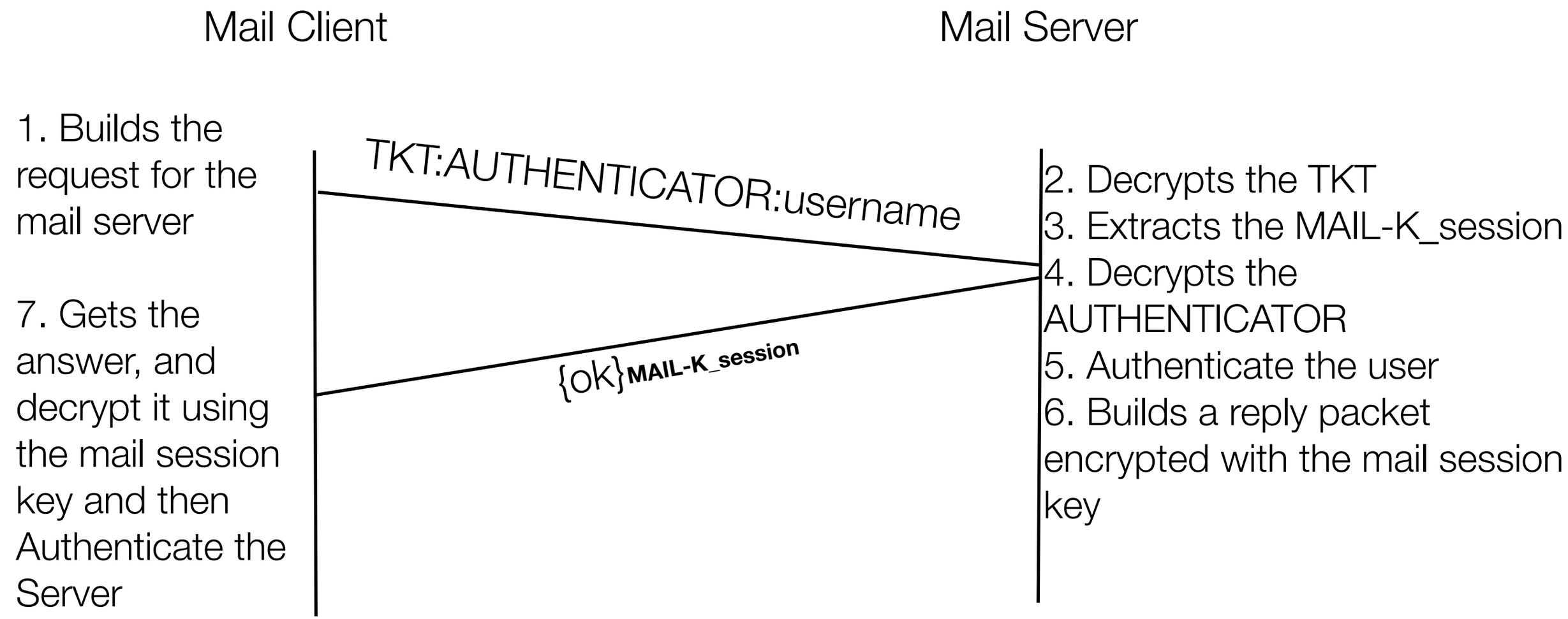
Euripide:	Possiamo enunciare il problema in un altro modo. Senza le chiavi di sessione e gli autenticatori, Caronte può proteggere i propri server dai falsi utenti, ma non può proteggere i propri utenti dai falsi server. Il sistema, per essere sicuro per tutti, ha bisogno di fare in modo che anche i client, prima di inviare informazioni importanti ad un servizio, possano autenticare i server. Un sistema, cioè, che consenta l'autenticazione reciproca. Ma la chiave di sessione risolve questo problema, a condizione che i programmi client siano progettati in maniera adeguata. Per tornare allo scenario del server di stampa, io voglio che il mio programma client che chiede ad un determinato server di stampare un documento magari riservato, sia in grado di assicurarsi che il server con cui sta dialogando è davvero il server che dice di essere, e non un server impostore.

Euripide:	Ed ecco che cosa deve fare il client per accertarsene. Io do il comando per stampare un determinato documento ad un determinato server di stampa. Assumo di avere il biglietto per quel servizio e la corrispondente chiave di sessione. Il client utilizza la chiave di sessione per creare un autenticatore e lo invia, insieme al biglietto a ciò che lui crede essere un determinato server di stampa. Il client NON INVIA per ora il documento riservato, ma aspetta la risposta del servizio. Ammettiamo che il pacchetto venga inviato al servizio autentico. Questo decripta il biglietto con la sua chiave segreta, estrae la chiave di sessione, quindi utilizza la chiave di sessione per decriptare l'autenticatore. Fatto ciò, il server fa tutte le verifiche. Essendo io ad aver fatto la richiesta, tutte le verifiche confermano la mia identità.

Euripide:	A questo punto il server deve rispondere al client di avermi autenticato, e che è disposto a stampare il documento. Ma deve farlo in modo da consentire al client di verificare la sua identità di server legittimo. Per farlo utilizza la sua copia della chiave di sessione per cifrare la risposta e poi la invia al client. Il client riceve il pacchetto dal server, e tenta di decifrarlo con la chiave di sessione. Se vi riesce. e legge il corretto messaggio di risposta, il mio client può essere sicuro che a rispondere è stato il server autentico, l'unico altro a conoscere la chiave di sessione. Quindi il client può inviare in maniera sicura il documento riservato al server di stampa. Immaginiamo ora che il mio pazzo capo, abbia cambiato la configurazione del sistema e del suo server di posta in modo da farmi credere che la sua stampante sia in realtà quella dell'azienda a cui voglio inviare il mio curriculum riservato. Il mio client invia la richiesta di stampa, con biglietto e autenticatore ed aspetta la risposta.

Euripide:	In questo caso il falso server di stampa che intende darmi una falsa autorizzazione a stampare, non può farlo, in quanto non potendo decifrare il mio biglietto (conosciuto solo dal vero server di stampa), non avrà la chiave di sessione con cui criptare la risposta. Il mio client non invierà nulla a quel server finché non riceverà la conferma della sua identità. Alla fine il client si stancherà di aspettare e mi informerà di non aver ricevuto alcuna risposta dal server, evitando così che un mio documento riservato finisca nelle mani del nemico. Sai, penso che abbiamo una base solida su cui partire per implementare il Sistema di Autenticazione Caronte
Athena:	Forse. In ogni caso, a me non piace il nome "Caronte".

Mutua Autenticazione



AUTHENTICATOR = {UserName,WS_address,timestamp}**MAIL-K_session**
TKT = {UserName,WS_address,mail,MAIL-K_session,lifespan,timestamp}**K_MAILserver**

Limiti di Kerberos4

- Kerberos4 ha avuto un grande successo ed è stato utilizzato in tutto il mondo, nonostante il suo disegno fosse specifico per l'ambiente "Athena" del MIT. Questo ha dato origine, sia a limitazioni di esportabilità che a limitazioni funzionali (legate alla specificità sia del disegno, che dell'implementazione del MIT).

- Inoltre sono state evidenziate deficienze tecniche nel disegno e nell'implementazione del MIT

Limiti “implementativi”

- Sistema di crittografia
- Dipendenza dal protocollo IP
- Message byte ordering
- Durata del Ticket
- Authentication Forwarding
- Formato dei Principals
- Cross-authentication

Sistema di crittografia

- Kerberos4 usa solo il DES (Data Encryption Standard) per cifrare i messaggi. Ciò ha reso difficoltosa l'esportazione di Kerberos4 al di fuori degli USA (l'esportazione di codice contenente DES, era soggetta a forti restrizioni da parte del governo Americano)

- Per poter esportare il codice di Kerberos4 del MIT, è stato necessario ripulirlo da tutti i riferimenti a crittografia (compreso i commenti). Al codice così ottenuto ("Bones") è stata poi aggiunto il codice di crittografia, ottenendo così l'E-Bones.

Dipendenza da IP

- Il disegno di Kerberos4 è legato all'uso del protocollo IP. Questo ne impedisce l'implementazione in ambienti che non usano tale protocollo.

Message byte order

- La scelta fatta nell'implementazione del MIT, rende semplice la comunicazione tra host con lo stesso "byte ordering", ma non seguendo le usuali convenzioni, preclude l'interoperabilità con macchine che non hanno un "byte ordering" usuale.

Durata del Ticket

- Per come viene codificata, la massima durata del Ticket, nella versione 4, è di 21 ore e 15 minuti.

Authentication Forwarding

- Nella versione 4, le credenziali prodotte per un determinato Client ed un determinato host, non possono essere forwardate su un altro host ed essere utilizzate da un altro Client.

Formato del nome dei Principals

- Il formato dei principals nella versione 4 ha solo 3 componenti (nome, istanza e REALM) ognuno dei quali può essere lungo fino a 39 caratteri. Inoltre scelte implementative hanno escluso il punto “.” come possibile carattere.

Inter-REALM Authentication

- Kerberos4 prevede la “cross-realm” authentication, ma la gestione di n REALMs richiede un numero di chiavi O(n2)

Debolezze Tecniche

- Doppia cifratura del Ticket
- Crittografia PCBC
- Autenticatori e reply detection
- Password vulnerabili da attacchi di tipo “forza bruta”
- Session Keys
- Cryptographic Checksum

Doppia cifratura

- Il messaggio di risposta a qualunque Client, viene cifrato o con la password dell'utente (nel caso di TGT) o con la session key del TGS e contiene, tra le altre cose, il Ticket, che è cifrato con la password del servizio.

PCBC

- Kerberos4 usa un metodo non standard per cifrare i messaggi DES. Questo metodo, “plain and cipher-block-chaining” (PCBC) fu un tentativo per fornire in una sola operazione sia crittografia dei dati che protezione dell’integrità, ma è risultato vulnerabile ad uno speciale attacco “block-exchange”.

Authenticators e reply detection

- La validità di un autenticatore è in un campo dell'autenticatore stesso, e quindi un server è in grado di scartare autenticatori scaduti, ma nella versione 4 non è stata implementata la gestione della lista degli autenticatori già usati e quindi, nell'intervallo di validità dell'autenticatore, questo potrebbe essere riutilizzato.

Passwords vulnerabili

- La risposta iniziale del Server Kerberos viene crittografata con una chiave ottenuta con un algoritmo noto, a partire dalla sola password dell'utente. Questo permette di scoprire la password dell'utente.

Session Keys

- Ogni Ticket rilasciato dal KDC contiene una chiave relativa a quel Ticket, che Client e Server usano per cifrare la comunicazione. Tuttavia siccome molti Clients usano un Ticket molte volte durante una sessione, è possibile che un “intruder” possa rimandare lo stesso messaggio a Client o Server non sufficientemente protetti.

Cryptographic checksum

- Il checksum di un messaggio crittografato (chiamato anche MAC Message Authentication Code o hash o funzione digest) nella versione 4 è basato su un algoritmo quadratico. In realtà l'implementazione del MIT non segue completamente l'algoritmo, e l'affidabilità di tale implementazione non è nota.

Kerberos 5

(Kerberos5 VS Kerberos4)

Migliorie rispetto a K4

- Crittografia
- Indirizzi di rete
- Modifica del Ticket
- Struttura dei nomi dei Principals
- Supporto per autenticazione Inter-REALM

Crittografia

- Per migliorare l'esportabilità del codice, l'uso della crittografia è stato relegato in moduli software ben distinti e facilmente separabili dal resto del codice.

- Quando viene usata la crittografia (che sia testo cifrato o chiavi) l'oggetto cifrato viene etichettato con un identificatore, in modo che il ricevente possa usare il giusto algoritmo per decifrare.

- L'integrità dei messaggi cifrati è delegata al sistema di crittografia, e laddove il sistema usato non sia sufficientemente sicuro, si usa un sistema di hashing prima della crittografia.

Indirizzi di rete

- Quando in un messaggio di rete appare un indirizzo di rete, esso viene etichettato con tipo e lunghezza, in modo che il ricevente possa interpretarlo correttamente. Questo permette di inserire in un unico Ticket tutti i protocolli e tutti gli indirizzi di rete dell'host.

Codifica

- I messaggi di rete del protocollo Kerberos5 sono descritti usando gli standard ISO ASN.1 (Abstract Syntax Notation One) e codificati secondo le “Basic Encoding Rules” dell’ASN.1

Tickets

- Oltre a contenere nuovi campi e flags, i ticket sono stati divisi in due parti: una rimane cifrate, mentre la seconda (che contiene solo il nome del servizio) viene trasmessa in chiaro. In questo modo servers con multiple identità riescono a selezionare la chiave corretta per poter decifrare il Ticket.

- Invece di timespamp e lifetime, i nuovi ticket hanno due timestamps (inizio e fine validità)

Struttura dei Principals

- I principals sono codificati in due parti: il REALM ed il resto.

Inter-REALM

- Nella versione 5 i REALMs cooperano attraverso una gerarchia definita dai nomi dei REALMs. In questo modo, il numero delle chiavi da gestire per la cross-authentication è lineare con il numero di REALMs.

Nuove funzionalità

- Tickets
- Supporto per Autorizzazione ed Accounting
- Pre-authentication
- Subsession key
- Sequence number

Tickets V5

- Un Ticket versione 5 contiene campi addizionali sia per “timestamps” che per “flags”.
  - Initial Ticket exchange flag
  - Renewable flag
  - Forwardable flag

AAA

- Anche se Kerberos non si occupa direttamente di Autorizzazione ed Accounting, fornisce un meccanismo per la trasmissione “a prova di corruzione” delle informazioni di autorizzazione ed accounting all’interno del Ticket.

Pre-authentication

- Per complicare il “furto” di passwords, Kerberos5 mette a disposizione dei campi aggiuntivi sia nella fase di acquisizione del Ticket iniziale che nelle successive.

- I campi aggiuntivi possono essere usati per passwords generate da apparati portatili, o per assicurarsi dell’identità del client PRIMA di spedirgli il Ticket

Subsession Key

- Siccome i Tickets sono riutilizzabili, per evitare i problemi che possono insorgere dall'utilizzo con molti clients e per molto tempo della session key, è possibile fare in modo che Client e Server negozino una nuova chiave, per ogni connessione.

Sequence number

- Mentre in Kerberos4 i messaggi cifrati (KRB_SAFE e KRB_PRIV) usano come campo di controllo il timestamp, nella versione 5, le applicazioni possono scegliere tra timestamp o un “sequence number”.

Kerberos5

Un po’ più in dettaglio

- I Messaggi Kerberos
  - AS_REQ/REP, TGS_REQ/REP, AP_REQ/REP
- Crittografia
  - Encryption type (enctype), Encryprion key, Salt, Key version number (kvno)
- Cross-REALM Authentication

I Messaggi Kerberos

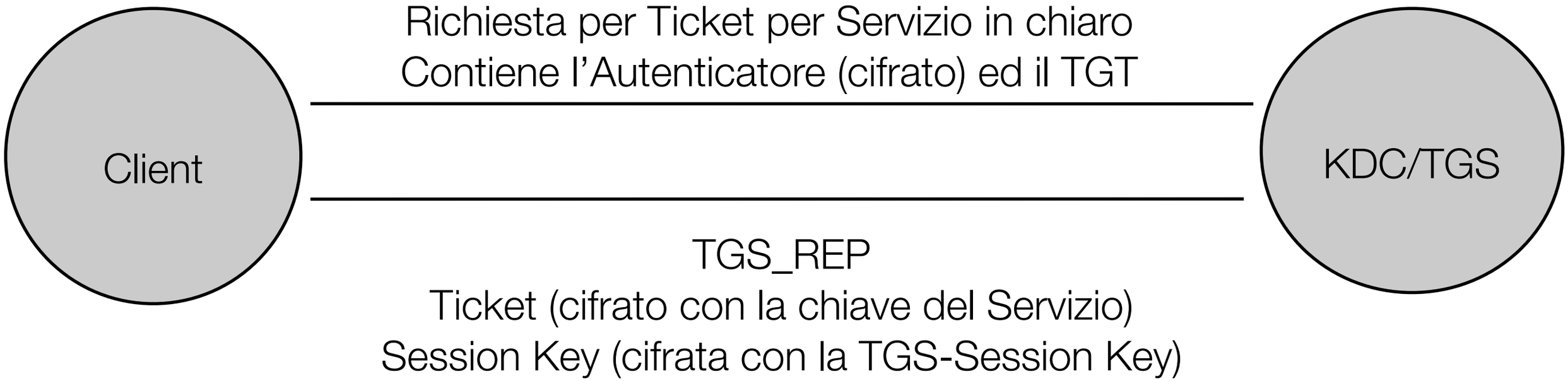
- Principali: Dialogo Client - KDC e Client - Server
  - AS_REQ/AS_REP
  - AP_REQ/AP_REP
  - TGS_REQ/TGS_REP
- Ulteriori messaggi
  - KRB_CRED
  - KRB_SAFE
  - KRB_PRIV

Kinit: Client - KDC/AS

- AS_REP contiene (tra l'altro):
  - TGT (crittografato con la chiave del TGS)
  - TGS-Session Key (crittografata con la chiave dell'utente)
- Il TGT contiene una copia della TGS-Session Key

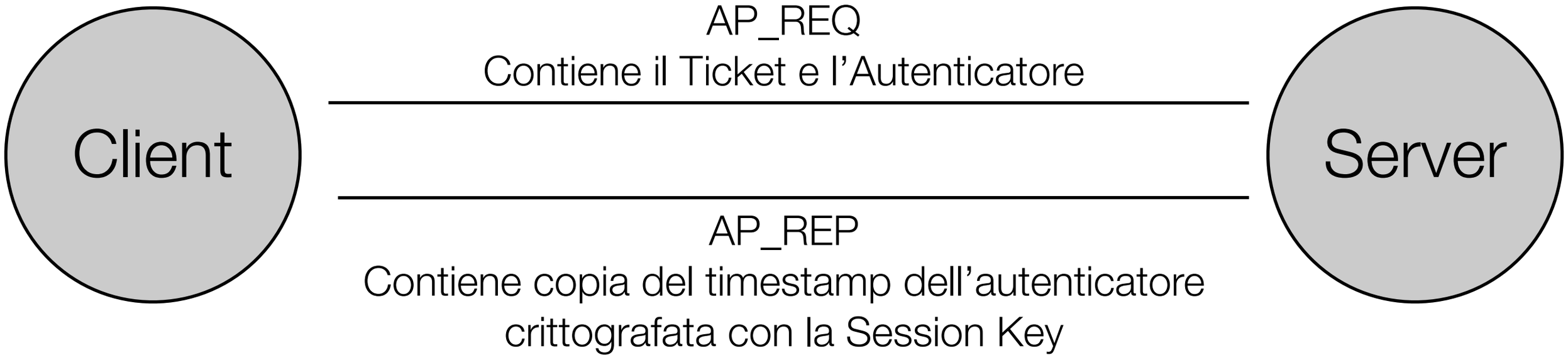
Mail: Client - KDC/TGS

- Nel TGS_REQ
  - Autenticatore (crittografato con la TGS-Session Key)
  - TGT (crittografato con la chiave del TGS)
- TGS_REP contiene

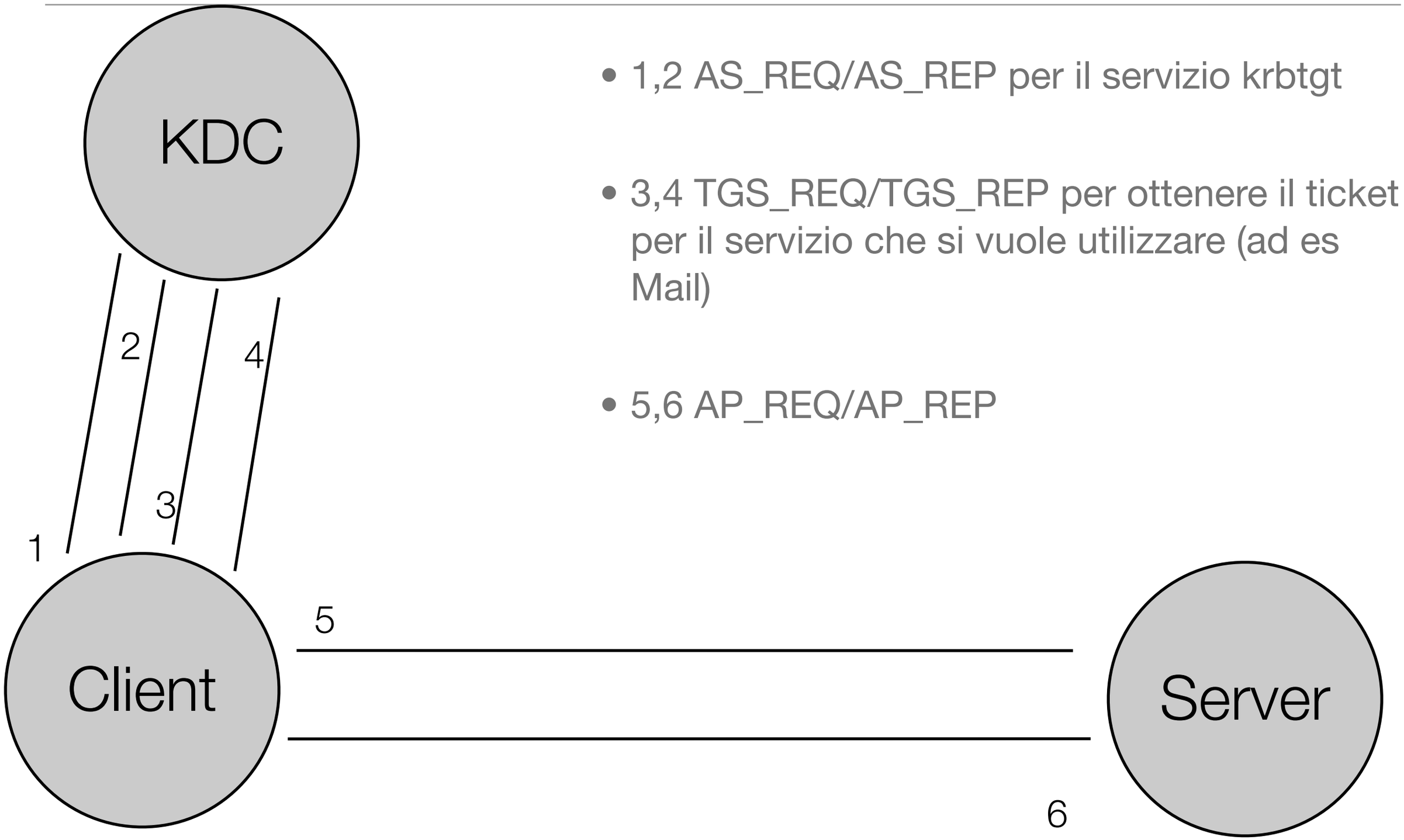


Mail: Client - Server

- AP_REQ
  - Ticket (crittografato con la chiave del Servizo Applicativo)
  - Autenticatore (crittografato con la Session Key). Viene gererato un Autenticatore per ogni AP_REQ

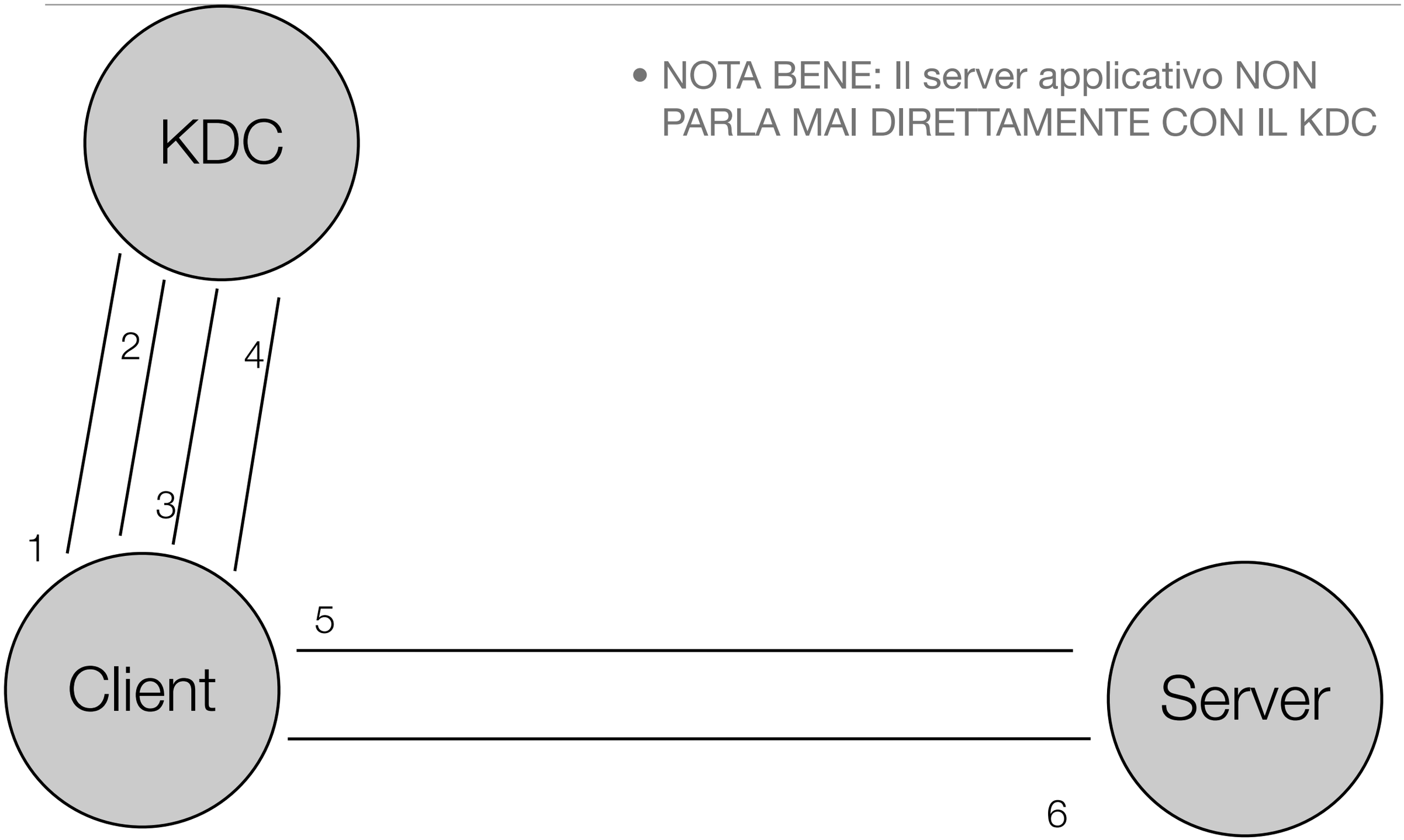


Client, KDC, Server



- 1,2 AS_REQ/AS_REP per il servizio krbtgt
- 3,4 TGS_REQ/TGS_REP per ottenere il ticket per il servizio che si vuole utilizzare (ad es Mail)
- 5,6 AP_REQ/AP_REP

Client, KDC, Server



- NOTA BENE: Il server applicativo NON PARLA MAI DIRETTAMENTE CON IL KDC

Uteriori messaggi

- KRB_CRED
  - Contiene un Ticket (normalmente KRBTGT) e la Session Key associata.
  - Usato per “forwardare” le credenziali
- KRB_SAFE
  - Usato per ottenere l’integrità dei dati del protocollo.
- KRB_PRIV
  - Usato per ottenere la protezione (cifratura) dei dati

Crittografia

- In protocollo Kerberos5 non ha limiti riguardo al supporto per algoritmi di cifratura (dipende dalle implementazioni). Ad es.
  - M.I.T. 1.2 - DES, 3DES (>1.3 RC4, AES 128 e 256 bits)
  - Windows2000 AD - DES, RC4
- Le parti cifrate sono etichettate con il tipo di cifratura.

Chiavi e numero di versione

- Ogni principal ha in genere tante chiavi quanti sono i tipi di cifratura supportati

- E’ possibile avere anche più chiavi con il medesimo encryption type, che vengono distinte tra loro usando il Key Version Number (kvno)

Key: vno 3, ArcFour with HMAC/md5, no salt
Key: vno 3, ArcFour with HMAC/md5, Version 5 - No Realm
Key: vno 3, ArcFour with HMAC/md5, Version 5 - Realm Only
Key: vno 3, Triple DES cbc mode with HMAC/sha1, no salt
Key: vno 3, DES with HMAC/sha1, no salt
Key: vno 3, DES cbc mode with RSA-MD5, no salt
Key: vno 3, DES cbc mode with CRC-32, Version 4
Key: vno 3, DES cbc mode with CRC-32, AFS version 3

Sale

- Le chiavi sono ottenute attraverso una funzione di HASH (CRC32, MD5, SHA-1) di
  - Password
  - Salt
    - Username@REALM (standard V5)
    - None (come in Kerberos4)
    - OnlyRealm (come in AFS)
    - NoRealm

Encryption types

- Il supporto per diversi titpi di cifratura richiede che venga negoziato il corretto “Encryption type” tra il client ed il KDC

- Per ogni dialogo client-KDC ci sono tre diversi encryption types che devono essere negoziati

  - Risposta (rep)

  - Ticket (tkt)

  - Session Key (ses)

Feb 06 09:53:20 kdc.example.com krb5kdc[2444](info): AS_REQ (7 etypes {18 17 16 23 1 3 2}) 192.168.5.100: ISSUE: authtime 1170752000, etypes {rep=16 tkt=23 ses=16}, user1@EXAMPLE.COM for krbtgt/EXAMPLE.COM@EXAMPLE.COM

Cross authentication

- Possibilità per un utente di un determinato REALM di accedere a servizi di altri REALMs
- Si ottiene attraverso l'instaurazione di relazioni di fiducia tra i REALM
  - Dirette
  - Transitive
  - Gerarchiche

Relazioni dirette

- Sono alla base delle autenticazioni cross-realm, e delle relazioni di fiducia transitive e gerarchiche
- Possono essere anche uni-direzionali (ma in questo caso si perdono le proprietà di transitività)

- REALM-A, REALM-B
  - krbtgt/REALM-B@REALM-A
  - krbtgt/REALM-A@REALM-B

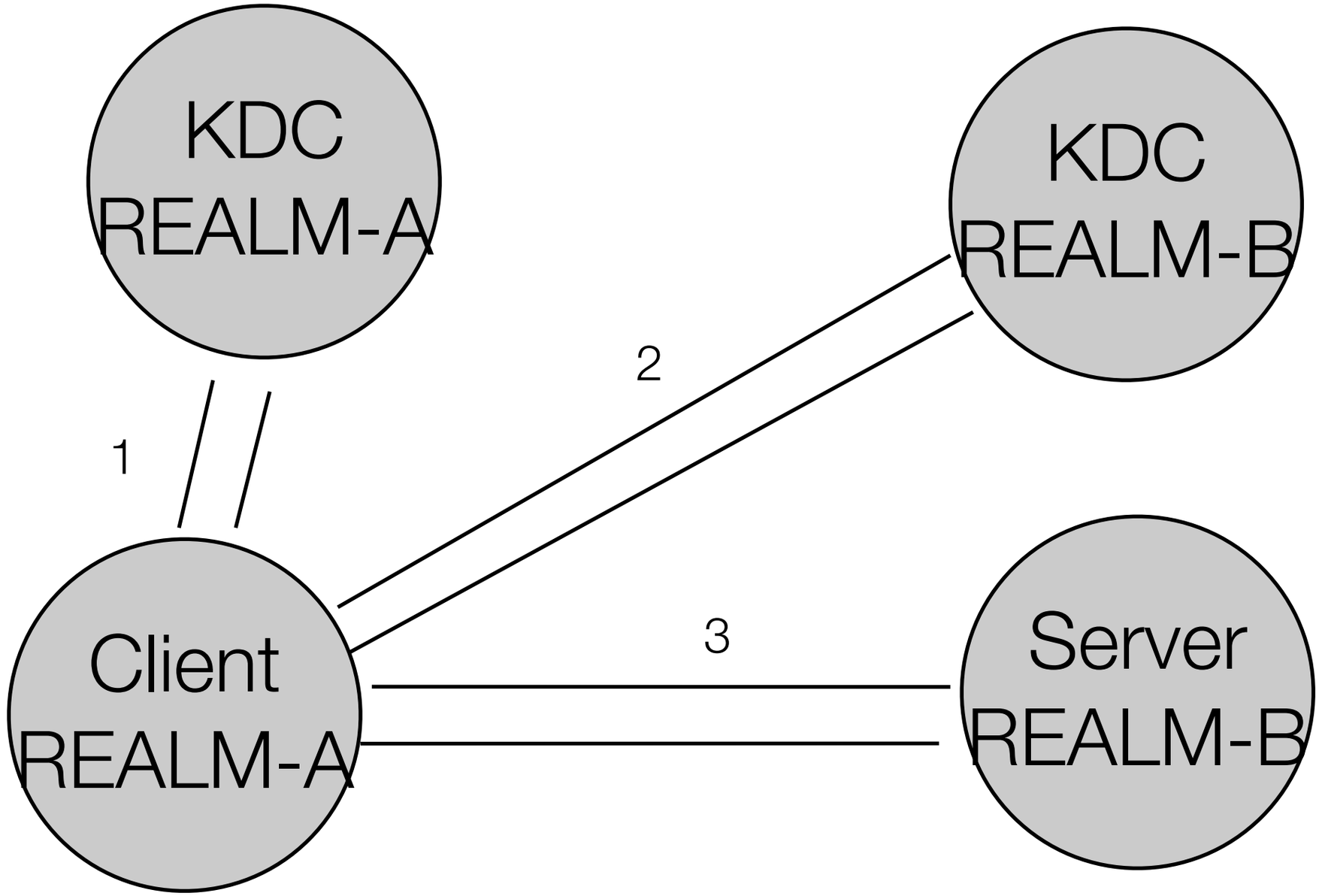
Principio di funzionamento

- Naturale estensione del funzionamento di Kerberos5, se si accetta che un TGS di un REALM possa considerare validi i TGT remoti, ossia emessi da un TGS remoto DEL QUALE SI FIDA (attraverso le relazione di fiducia)
    - I TGT remoti vengono rilasciati dal TGS remoto (e non dall'AS)

In maggior dettaglio

- Il CLIENT che vuole accedere ai servizi del REALM esterno, chiede al PROPRIO KDC il “cross-realm TGT”

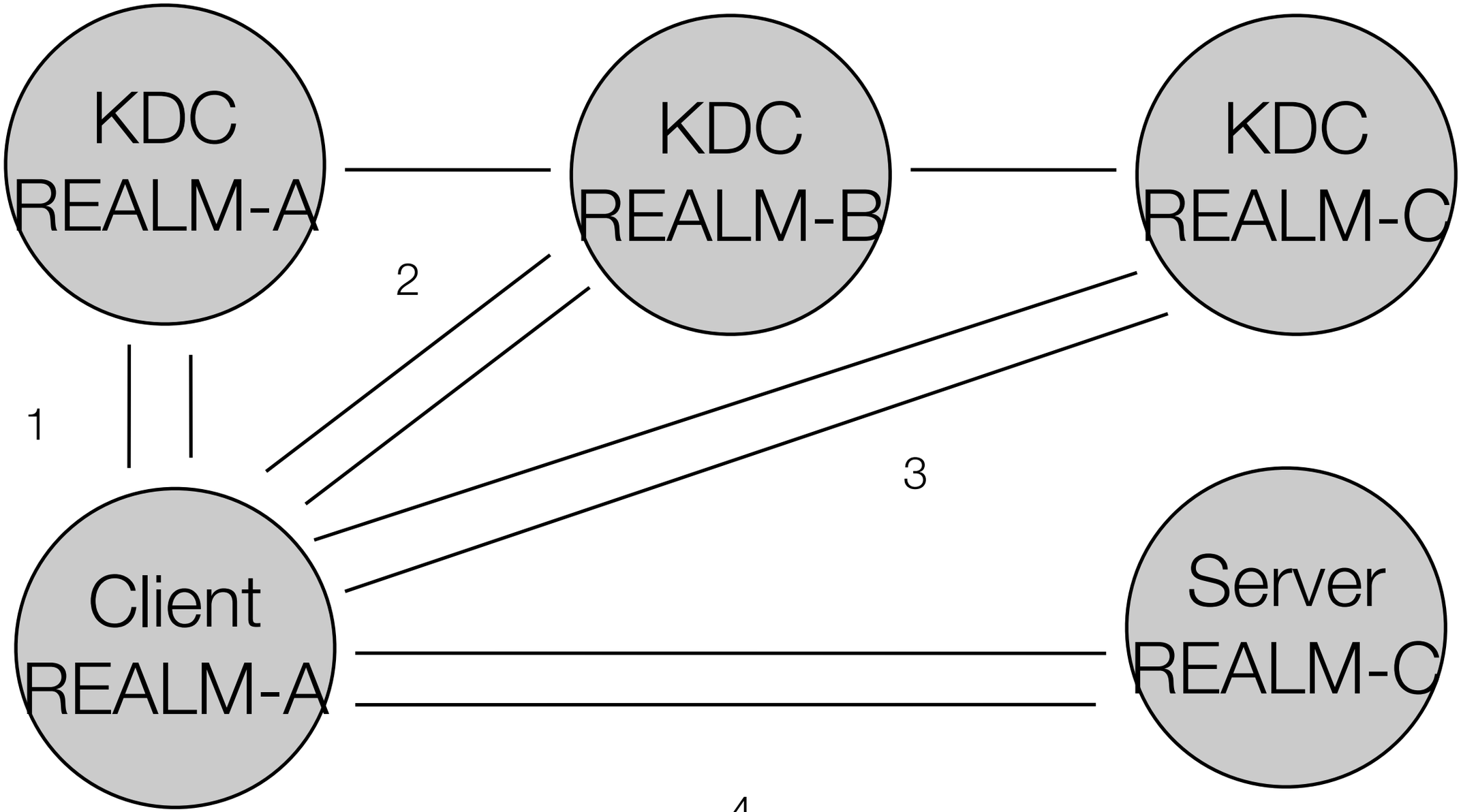
- Presenta questo speciale TGT al KDC del REALM esterno per ottenere il “service Ticket”



1 - Il client chiede al proprio KDC (usando un normale TGS_REQ) il ktbtgt/REALM-B@REALM-A
2 - Manda una TGS_REQ al KDC del REALM-B presentandogli il cross-realm TGT. Il KDC del REALM-B fornisce il service ticket per client@REALM-A
3 - Il client manda un normale AP_REQ al server del REALM-B

Relazioni transitive

- Se REALM-A <-> REALM-B
- e REALM-B <-> REALM-C
- allora
- REALM-A <-> REALM-C

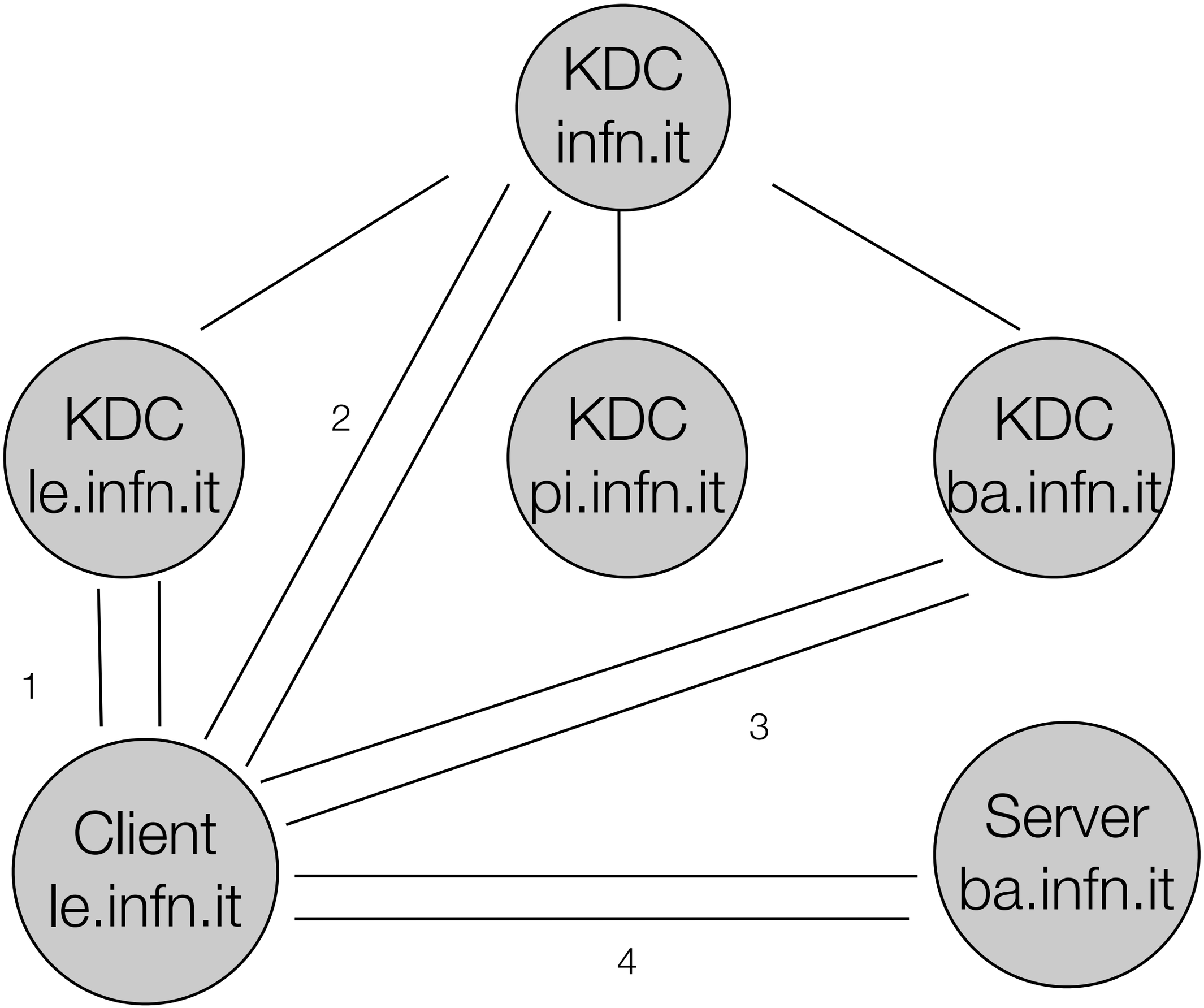


Sia il Client che tutti i KDC devono conoscere il “percorso” lungo la catena di relazioni [capaths] nel file di configurazione (/etc/krb5.conf)

Relazioni gerarchiche

- Se i vari REALMs corrispondono ai nomi a dominio e se sono in gerarchia tra di loro (es. infn.it, le.infn.it, pi.infn.it, lnf.infn.it, lngs.infn.it ecc. ecc.) e se sono state definite le relazioni di fiducia tra i REALMs adiacenti, allora la transitività si ottiene automaticamente.

- Non è necessario configurare i percorsi di transitività [capaths]


KDC
infn.it
KDC
le.infn.it
KDC
pi.infn.it
KDC
ba.infn.it
1
2
3
4
Client
le.infn.it
Server
ba.infn.it

Sicurezza nella cross-authentication

- Un utente di un REALM cross-autenticato non comparirà mai come utente locale

- L'amministratore del KDC remoto può impersonare CHIUNQUE del suo REALM (non di altri REALM)

- Per default il check dell'autorizzazione non permette ad utenti di REALM esterni a loggarsi su accounts locali

Ultime novità

LDAP KDB

- A partire dalla versione 1.6 di Kerberos5 MIT (rilasciata il 9 gennaio 2007) è possibile memorizzare il database delle password su un server LDAP.
    - Replica del database via protocollo LDAP (Multi Master)

PKINIT

- Serve per ottenere il TGT usando il proprio certificato X.509
- Usato da Windows AD
- Standardizzato solo di recente (RFC 4556 giugno 2006)
- Supporto iniziale in heimdal 0.8 (beta) e MIT Kerberos5 1.6 (alpha)

Bibliografia

- M.I.T. http://web.mit.edu/kerberos
- OpenAFS Best Practice WorkShop http://www.pmw.org/afsbpw06/kerbtut.html
- ZeroShell (by Fulvio Ricciardi) http://www.zeroshell.net/kerberos/Kerberos-introduction/
- Heimdal 0.7 http://www.h5l.se/manual/heimdal-0-7-branch/info/heimdal.html
- Heimdal 0.8 http://www.h5l.se/manual/HEAD/info/heimdal.html

RFCs

- RFC 1964 - The Kerberos Version 5 GSS-API Mechanism
- RFC 3961 - Encryption and Checksum Specifications for Kerberos 5
- RFC 4120 - The Kerberos Network Authentication Service (V5) (Obsolete the most known rfc1510)
- RFC 4121 - The Kerberos Version 5 GSS-API Mechanism: Version 2
- RFC 4556 - Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)
- RFC 4557 - Online Certificate Status Protocol (OCSP) Support for PKINIT