Plan of Talk

- **Kerberos**

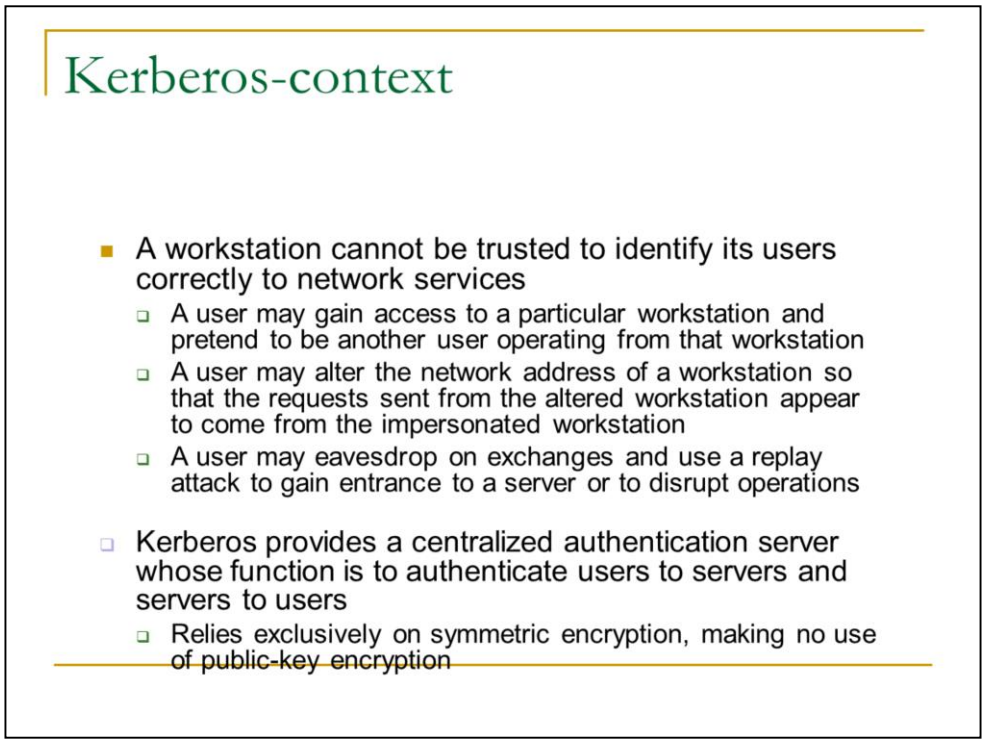
Kerberos

Kerberos

is an authentication server

developed as a part Project Athena, MIT

- Kerberos provides centralised private-key third-party authentication in a distributed network



Kerberos is an authentication service developed as part of Project Athena at MIT.

The problem that Kerberos addresses is this: Assume an open distributed environment

in which users at workstations wish to access services on servers distributed

throughout the network. We would like for servers to be able to restrict access to

authorized users and to be able to authenticate requests for service. In this environment,

a workstation cannot be trusted to identify its users correctly to network

services. In particular, the following three threats exist:

1. A user may gain access to a particular workstation and pretend to be another

user operating from that workstation.

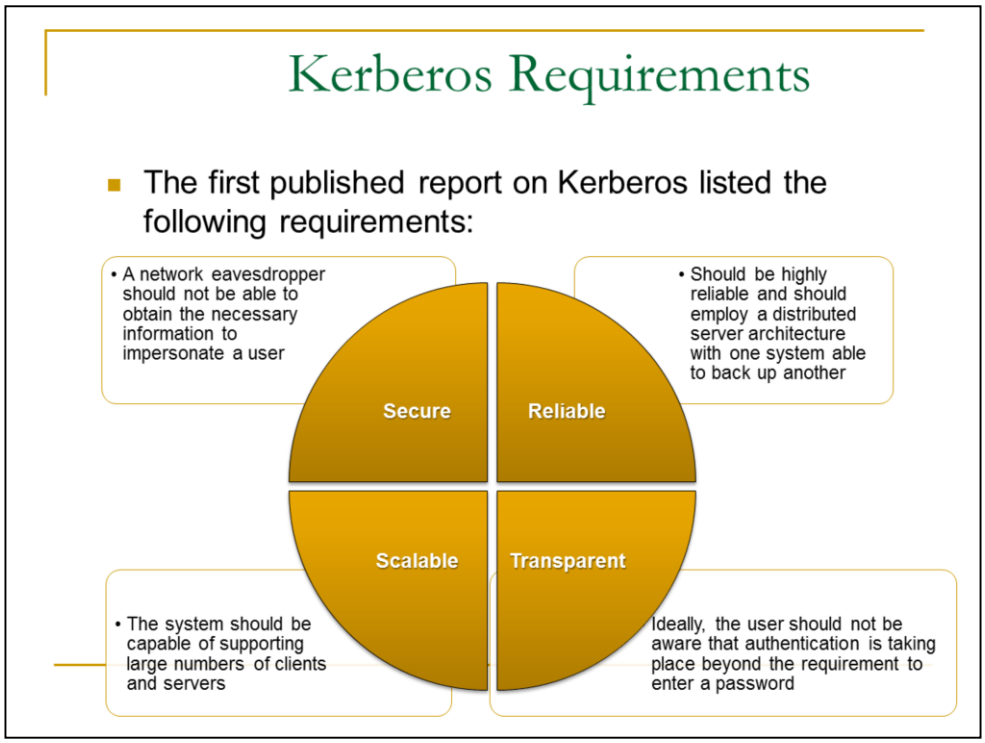
2. A user may alter the network address of a workstation so that the requests

sent from the altered workstation appear to come from the impersonated workstation.

3. A user may eavesdrop on exchanges and use a replay attack to gain entrance to a server or to disrupt operations.

In any of these cases, an unauthorized user may be able to gain access to services and data that he or she is not authorized to access. Rather than building in elaborate authentication protocols at each server, Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Unlike most other authentication schemes described in this book, Kerberos relies exclusively on symmetric encryption, making no use of public-key encryption.

Two versions of Kerberos are in common use. Version 4 [MILL88, STEI88] implementations still exist. Version 5 [KOHL94] corrects some of the security deficiencies of version 4 and has been issued as a proposed Internet Standard (RFC 4120 and RFC 4121).



Kerberos assumes a
distributed client/server architecture and employs one or more Kerberos servers to
provide an authentication service.

The first published report on Kerberos [STEI88] listed the following requirements.

• Secure: A network eavesdropper should not be able to obtain the necessary information to impersonate a user. More generally, Kerberos should be strong enough that a potential opponent does not find it to be the weak link.

• Reliable: For all services that rely on Kerberos for access control, lack of availability of the Kerberos service means lack of availability of the supported services. Hence, Kerberos should be highly reliable and should employ a distributed
server architecture with one system able to back up another.

• Transparent: Ideally, the user should not be aware that authentication is taking place beyond the requirement to enter a password.

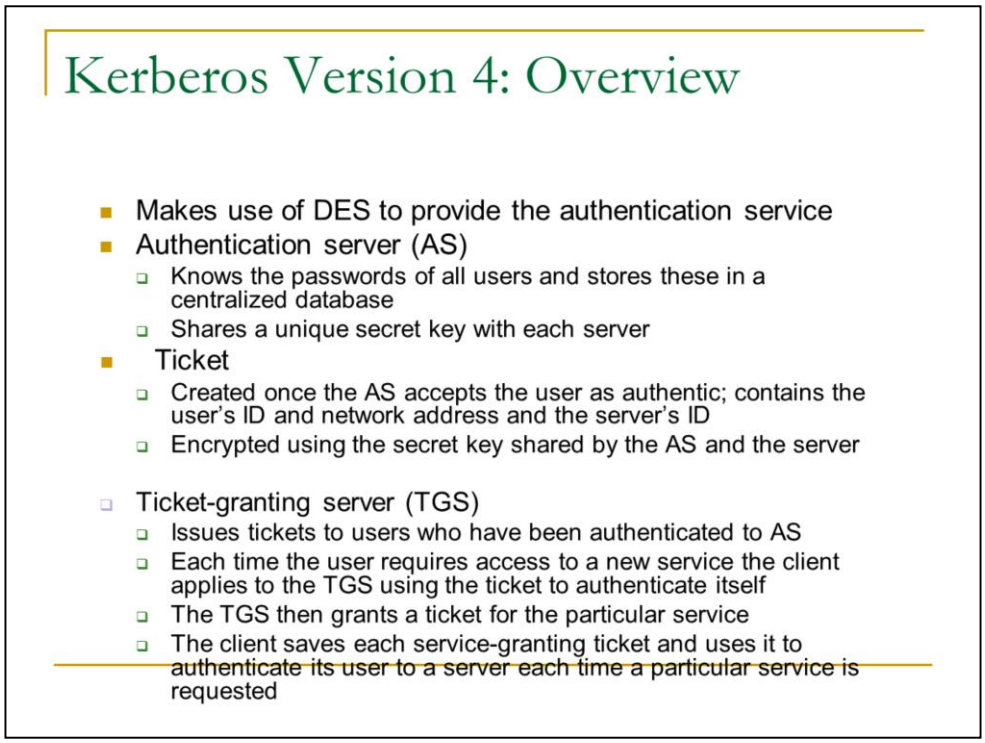
• Scalable: The system should be capable of supporting large numbers of clients and servers. This suggests a modular, distributed architecture.

To support these requirements, the overall scheme of Kerberos is that of a trusted third-party authentication service that uses a protocol based on that proposed
by Needham and Schroeder [NEED78], which was discussed in Section 15.2.

It is trusted in the sense that clients and servers trust Kerberos to mediate their mutual authentication. Assuming the Kerberos protocol is well designed, then the authentication service is secure if the Kerberos server itself is secure.

Kerberos cont.

- Implemented using a variant of Needham-Schroeder protocol.
- There are two versions- versions 4 and 5



Version 4 of Kerberos makes use of DES, in a rather elaborate protocol, to provide

the authentication service. Viewing the protocol as a whole, it is difficult to see the

need for the many elements contained therein. Therefore, we adopt a strategy used

by Bill Bryant of Project Athena [BRYA88] and build up to the full protocol by looking

first at several hypothetical dialogues. Each successive dialogue adds additional

complexity to counter security vulnerabilities revealed in the preceding dialogue.

In an unprotected network environment, any

client can apply to any server for service. The obvious security risk is that of impersonation.

An opponent can pretend to be another client and obtain unauthorized

privileges on server machines. To counter this threat, servers must be able to

confirm the identities of clients who request service. Each server can be required to
undertake this task for each client/server interaction, but in an open environment, this places a substantial burden on each server.

An alternative is to use an authentication server (AS) that knows the passwords of all users and stores these in a centralized database. In addition, the AS shares a unique secret key with each server. These keys have been distributed physically
or in some other secure manner.

The AS checks its database to see if the user has supplied the proper password for this user ID and whether this user is permitted access to server V. If
both tests are passed, the AS accepts the user as authentic and must now convince
the server that this user is authentic. To do so, the AS creates a ticket that contains
the user's ID and network address and the server's ID. This ticket is encrypted using the secret key shared by the AS and this server.

TGS, issues tickets to users who have been authenticated to
AS. Thus, the user first requests a ticket-granting ticket from the AS. The client module in the user workstation saves this ticket. Each time the user requires
access to a new service, the client applies to the TGS, using the ticket to authenticate
itself. The TGS then grants a ticket for the particular service. The client saves each service-granting ticket and uses it to authenticate its user to a server each time
a particular service is requested.

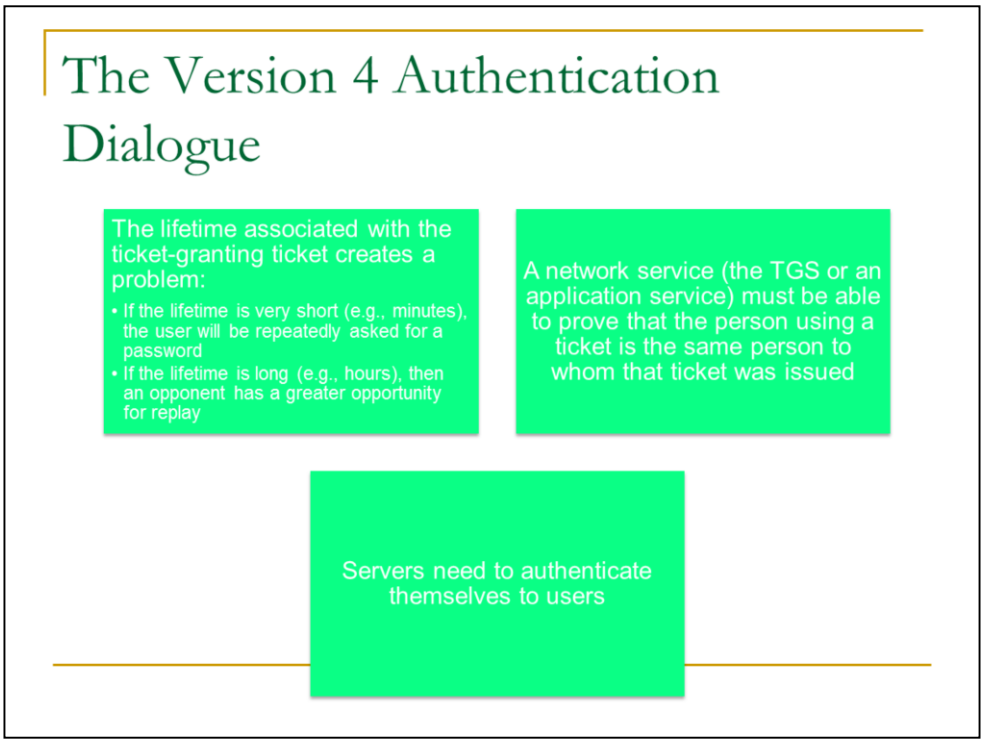
Because only the correct user should know the password, only the correct user can recover the ticket. Thus, we have used the password to obtain credentials from

Kerberos without having to transmit the password in plaintext. The ticket itself consists of the ID and network address of the user, and the ID of the TGS. This corresponds to the first scenario. The idea is that the client can use this ticket to request multiple service-granting tickets. So the ticket-granting ticket is to be reusable.

However, we do not wish an opponent to be able to capture the ticket and use it. Consider the following scenario: An opponent captures the login ticket and waits until the user has logged off his or her workstation. Then the opponent either gains access to that workstation or configures his workstation with the same network address as that of the victim. The opponent would be able to reuse the ticket to spoof the TGS. To counter this, the ticket includes a timestamp, indicating the date and time at which the ticket was issued, and a lifetime, indicating the length of time for which the ticket is valid (e.g., eight hours). Thus, the client now has a reusable ticket and need not bother the user for a password for each new service request. Finally, note that the ticket-granting ticket is encrypted with a secret key known only to the AS and the TGS. This prevents alteration of the ticket. The ticket is reencrypted with a key based on the user's password. This assures that the ticket can be recovered only by the correct user, providing the authentication.

The service-granting ticket has the same structure as the ticket-granting ticket. Indeed, because the TGS is a server, we would expect that the same elements are needed to authenticate a client to the TGS and to authenticate a client to an application server. Again, the ticket contains a timestamp and lifetime. If the user wants access to the same service at a later time, the client can simply use the previously acquired service-granting ticket and need not bother the user for a password. Note that the ticket is encrypted with a secret key known only to the TGS and the

server, preventing alteration.



The heart of the first problem is the lifetime associated with the ticket-granting ticket. If this lifetime is very short (e.g., minutes), then the user will be repeatedly asked for a password. If the lifetime is long (e.g., hours), then an opponent has a greater opportunity for replay. An opponent could eavesdrop on the network and capture a copy of the ticket-granting ticket and then wait for the legitimate user to log out. Then the opponent could forge the legitimate user's network address and send the message of step (3) to the TGS. This would give the opponent unlimited access to the resources and files available to the legitimate user.

Similarly, if an opponent captures a service-granting ticket and uses it before it expires, the opponent has access to the corresponding service.

Thus, we arrive at an additional requirement. A network service (the TGS or an application service) must be able to prove that the person using a ticket is the same person to whom that ticket was issued.

The second problem is that there may be a requirement for servers to authenticate themselves to users. Without such authentication, an opponent could sabotage the configuration so that messages to a server were directed to another location. The false server would then be in a position to act as a real server and capture any information from the user and deny the true service to the user.

Table 15.1 (page 484 in textbook)

Summary of Kerberos Version 4 Message Exchanges

(1) C → AS $ID_c \| ID_{tgs} \| TS_1$

(2) AS → C $E(K_c, [K_{c,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}])$

$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2])$

(a) Authentication Service Exchange to obtain ticket-granting ticket

(3) C → TGS $ID_v \| Ticket_{tgs} \| Authenticator_c$

(4) TGS → C $E(K_{c,tgs}, [K_{c,v} \| ID_v \| TS_4 \| Ticket_v])$

$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2])$

$Ticket_v = E(K_v, [K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4])$

$Authenticator_c = E(K_{c,tgs}, [ID_C \| AD_C \| TS_3])$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

(5) C → V $Ticket_v \| Authenticator_c$

(6) V → C $E(K_{c,v}, [TS_5 + 1])$ (for mutual authentication)

$Ticket_v = E(K_v, [K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4])$

$Authenticator_c = E(K_{c,v}, [ID_C \| AD_C \| TS_5])$

(c) Client/Server Authentication Exchange to obtain service

Table 15.1, shows the actual Kerberos protocol.

Table 15.1a shows the technique for distributing the session key. As before,

the client sends a message to the AS requesting access to the TGS. The AS responds

with a message, encrypted with a key derived from the user's password that

contains the ticket. The encrypted message also contains a copy of the session key,

where the subscripts indicate that this is a session key for C and TGS. Because

this session key is inside the message encrypted with K_c , only the user's client can

read it. The same session key is included in the ticket, which can be read only by

the TGS. Thus, the session key has been securely delivered to both C and the TGS.

Armed with the ticket and the session key, C is ready to approach the TGS. As before, C sends the TGS a message that includes the ticket plus the ID of the requested service [message (3) in Table 15.1b].

The reply from the TGS in message (4) follows the form of message (2). The message is encrypted with the session key shared by the TGS and C and includes a session key to be shared between C and the server V, the ID of V, and the timestamp of the ticket. The ticket itself includes the same session key.

C now has a reusable service-granting ticket for V. When C presents this ticket, as shown in message (5), it also sends an authenticator. The server can decrypt the ticket, recover the session key, and decrypt the authenticator.

If mutual authentication is required, the server can reply as shown in message (6) of Table 15.1. The server returns the value of the timestamp from the authenticator, incremented by 1, and encrypted in the session key. C can decrypt this message to recover the incremented timestamp. Because the message was encrypted by the session key, C is assured that it could have been created only by V. The contents of the message assure C that this is not a replay of an old reply.

Finally, at the conclusion of this process, the client and server share a secret key. This key can be used to encrypt future messages between the two or to exchange a new random session key for that purpose.

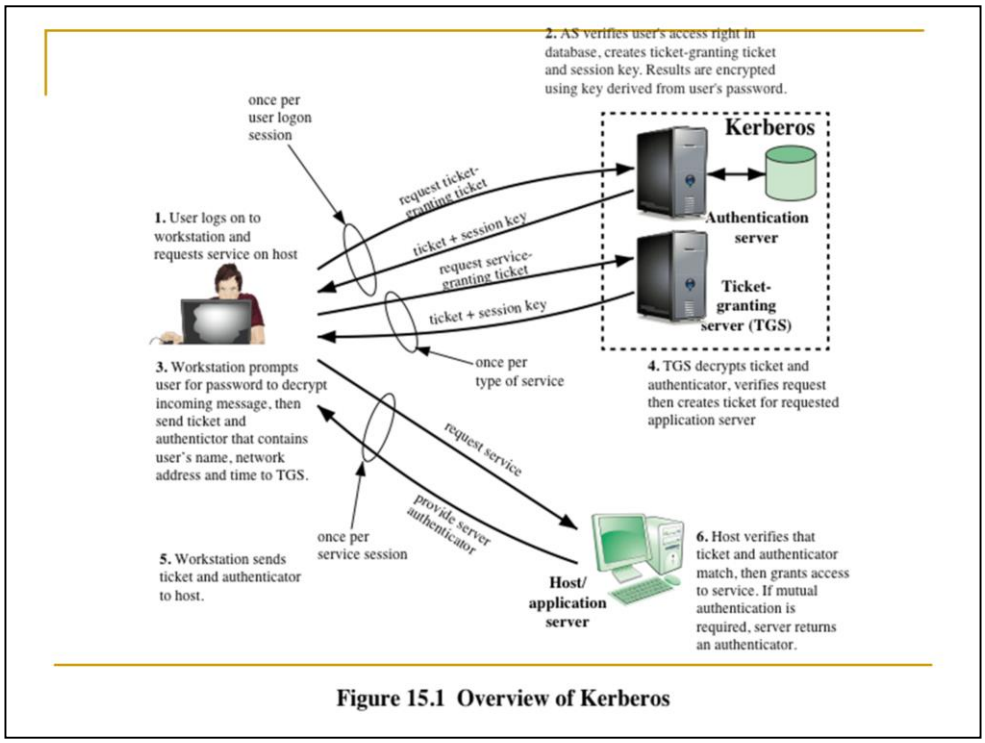


Figure 15.1 Overview of Kerberos

Figure 15.1 provides an overview.

First, consider the problem of captured ticket-granting tickets and the need

to determine that the ticket presenter is the same as the client for whom the ticket

was issued. The threat is that an opponent will steal the ticket and use it before it

expires. To get around this problem, let us have the AS provide both the client and

the TGS with a secret piece of information in a secure manner. Then the client can

prove its identity to the TGS by revealing the secret information—again in a secure

manner. An efficient way of accomplishing this is to use an encryption key as the

secure information; this is referred to as a session key in Kerberos.

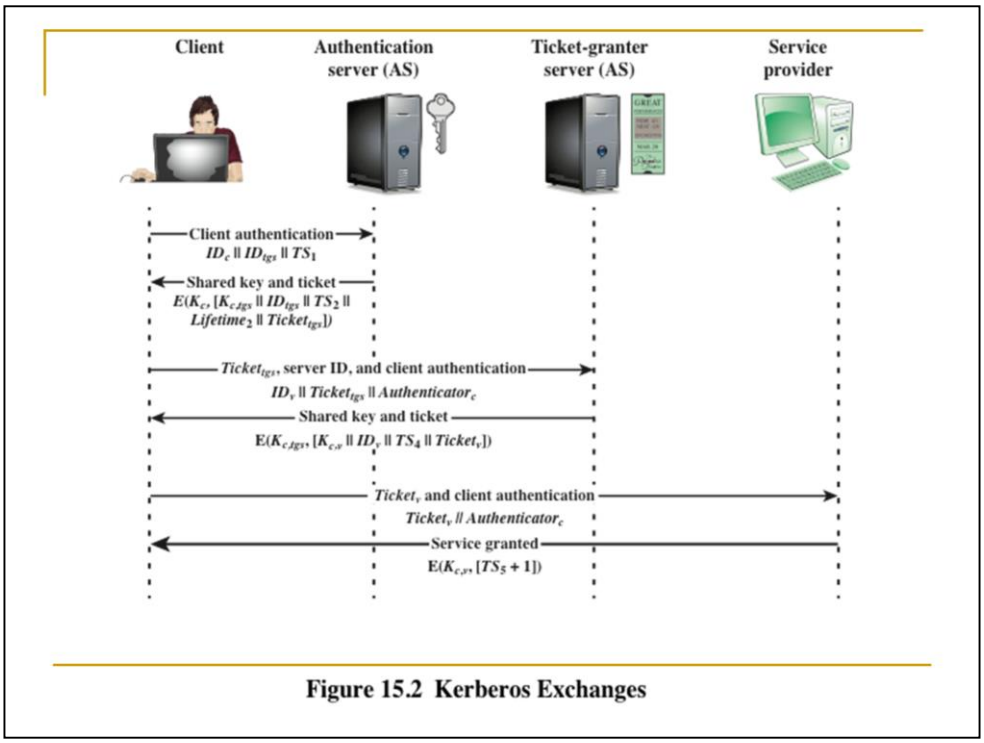


Figure 15.2 Kerberos Exchanges

Figure 15.2 illustrates the Kerberos exchanges among the parties.

Table 15.2 Rationale for the Elements of the Kerberos Version 4 Protocol
(page 1 of 3)

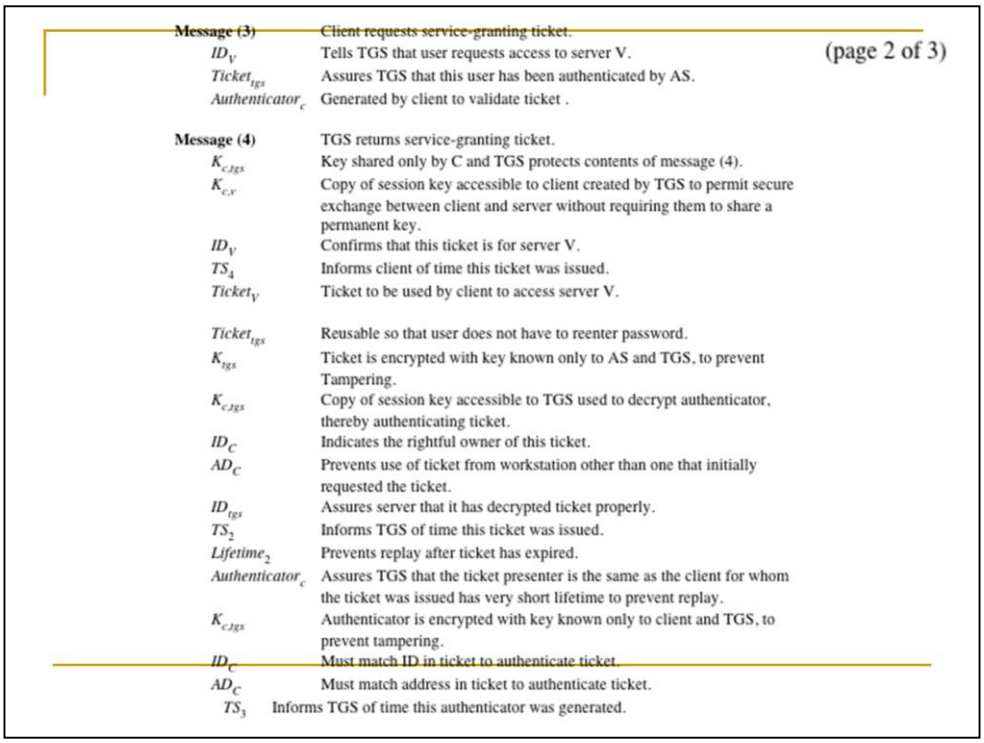
Message (1)	Client requests ticket-granting ticket.
ID_C	Tells AS identity of user from this client.
ID_{tgs}	Tells AS that user requests access to TGS.
TS_1	Allows AS to verify that client's clock is synchronized with that of AS.
Message (2)	AS returns ticket-granting ticket.
K_c	Encryption is based on user's password, enabling AS and client to verify password, and protecting contents of message (2).
$K_{c,tgs}$	Copy of session key accessible to client created by AS to permit secure exchange between client and TGS without requiring them to share a permanent key.
ID_{tgs}	Confirms that this ticket is for the TGS.
TS_2	Informs client of time this ticket was issued.
$Lifetime_2$	Informs client of the lifetime of this ticket.
$Ticket_{tgs}$	Ticket to be used by client to access TGS.

(This table can be found on pages 487 – 488 in the textbook)

Table 15.2

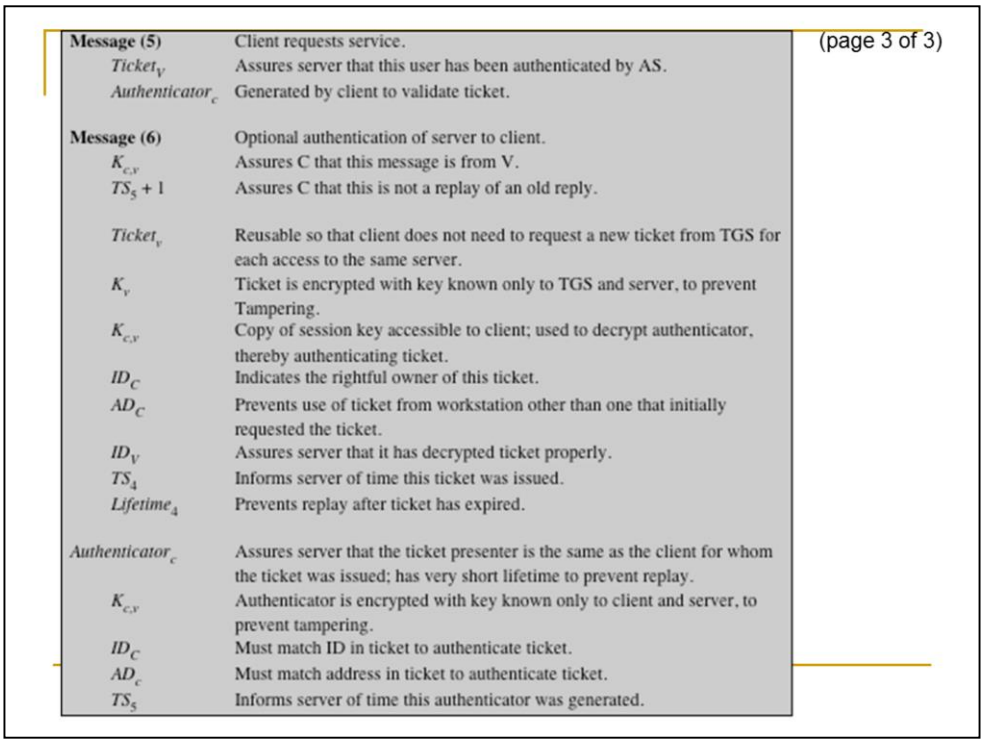
summarizes the justification for each of the elements in the Kerberos protocol.

Page 1 – authentication service exchange



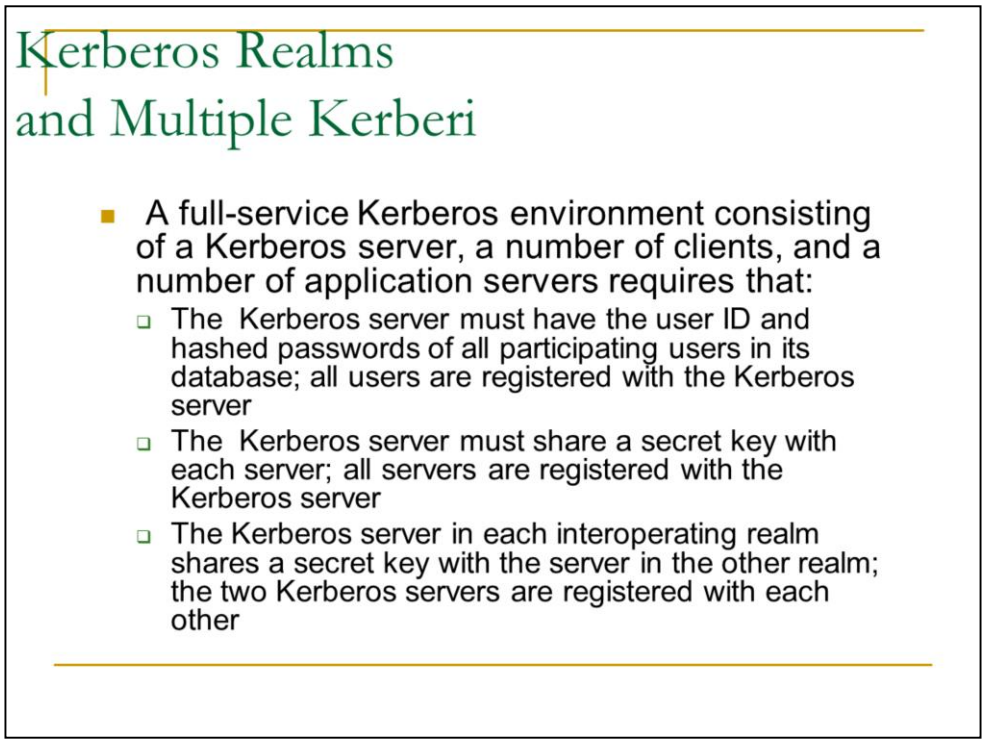
Message (3)	Client requests service-granting ticket.
ID_V	Tells TGS that user requests access to server V.
$Ticket_{tgs}$	Assures TGS that this user has been authenticated by AS.
$Authenticator_c$	Generated by client to validate ticket .
Message (4)	TGS returns service-granting ticket.
$K_{c,tgs}$	Key shared only by C and TGS protects contents of message (4).
$K_{c,v}$	Copy of session key accessible to client created by TGS to permit secure exchange between client and server without requiring them to share a permanent key.
ID_V	Confirms that this ticket is for server V.
TS_4	Informs client of time this ticket was issued.
$Ticket_V$	Ticket to be used by client to access server V.
$Ticket_{tgs}$	Reusable so that user does not have to reenter password.
K_{tgs}	Ticket is encrypted with key known only to AS and TGS, to prevent Tampering.
$K_{c,tgs}$	Copy of session key accessible to TGS used to decrypt authenticator, thereby authenticating ticket.
ID_C	Indicates the rightful owner of this ticket.
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket.
ID_{tgs}	Assures server that it has decrypted ticket properly.
TS_2	Informs TGS of time this ticket was issued.
$Lifetime_2$	Prevents replay after ticket has expired.
$Authenticator_c$	Assures TGS that the ticket presenter is the same as the client for whom the ticket was issued has very short lifetime to prevent replay.
$K_{c,tgs}$	Authenticator is encrypted with key known only to client and TGS, to prevent tampering.
ID_C	Must match ID in ticket to authenticate ticket.
AD_C	Must match address in ticket to authenticate ticket.
TS_3	Informs TGS of time this authenticator was generated.

Page 2 – ticket-granting service exchange



Message (5)	Client requests service.	(page 3 of 3)
$Ticket_V$	Assures server that this user has been authenticated by AS.	
$Authenticator_c$	Generated by client to validate ticket.	
Message (6)	Optional authentication of server to client.	
$K_{c,v}$	Assures C that this message is from V.	
$TS_5 + 1$	Assures C that this is not a replay of an old reply.	
$Ticket_v$	Reusable so that client does not need to request a new ticket from TGS for each access to the same server.	
K_v	Ticket is encrypted with key known only to TGS and server, to prevent Tampering.	
$K_{c,v}$	Copy of session key accessible to client; used to decrypt authenticator, thereby authenticating ticket.	
ID_C	Indicates the rightful owner of this ticket.	
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket.	
ID_V	Assures server that it has decrypted ticket properly.	
TS_4	Informs server of time this ticket was issued.	
$Lifetime_4$	Prevents replay after ticket has expired.	
$Authenticator_c$	Assures server that the ticket presenter is the same as the client for whom the ticket was issued; has very short lifetime to prevent replay.	
$K_{c,v}$	Authenticator is encrypted with key known only to client and server, to prevent tampering.	
ID_C	Must match ID in ticket to authenticate ticket.	
AD_c	Must match address in ticket to authenticate ticket.	
TS_5	Informs server of time this authenticator was generated.	

Page 3 – client/server authentication exchange



A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers requires the following:

1. The Kerberos server must have the user ID and hashed passwords of all participating users in its database. All users are registered with the Kerberos server.

2. The Kerberos server must share a secret key with each server. All servers are registered with the Kerberos server.

Networks of clients and servers under different administrative organizations typically constitute different realms. That is, it generally is not practical or does not conform to administrative policy to have users and servers in one administrative domain registered with a Kerberos server elsewhere. However, users in one realm may need access to servers in other realms, and some servers may be willing to provide service to users from other realms, provided that those users are authenticated. Kerberos provides a mechanism for supporting such interrealm authentication. For two realms to support interrealm authentication, a third requirement is added:

3. The Kerberos server in each interoperating realm shares a secret key with the server in the other realm. The two Kerberos servers are registered with each other.

The scheme requires that the Kerberos server in one realm trust the Kerberos server in the other realm to authenticate its users. Furthermore, the participating servers in the second realm must also be willing to trust the Kerberos server in the first realm.

Kerberos Realm

- A set of managed nodes that share the same Kerberos database
- The database resides on the Kerberos master computer system, which should be kept in a physically secure room
- A read-only copy of the Kerberos database might also reside on other Kerberos computer systems
- All changes to the database must be made on the master computer system
- Changing or accessing the contents of a Kerberos database requires the Kerberos master password

A Kerberos realm is a set of managed nodes that

share the same Kerberos database. The Kerberos database resides on the Kerberos

master

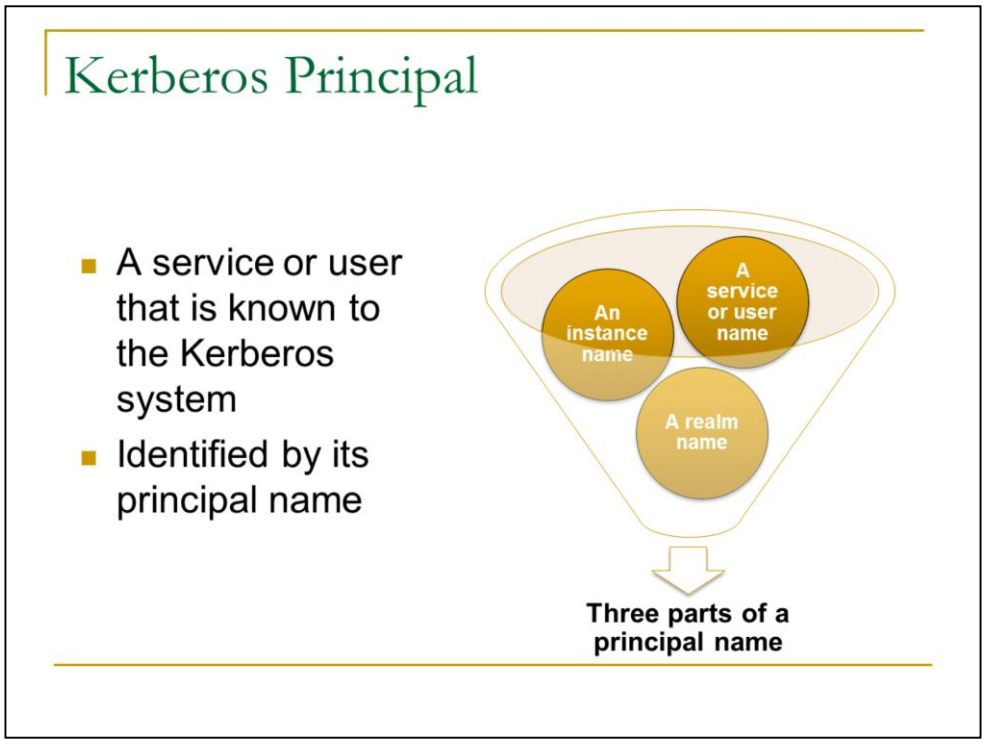
computer system, which should be kept in a physically secure room. A read-only

copy of the Kerberos database might also reside on other Kerberos computer

systems. However, all changes to the database must be made on the master computer

system. Changing or accessing the contents of a Kerberos database requires

the Kerberos master password.

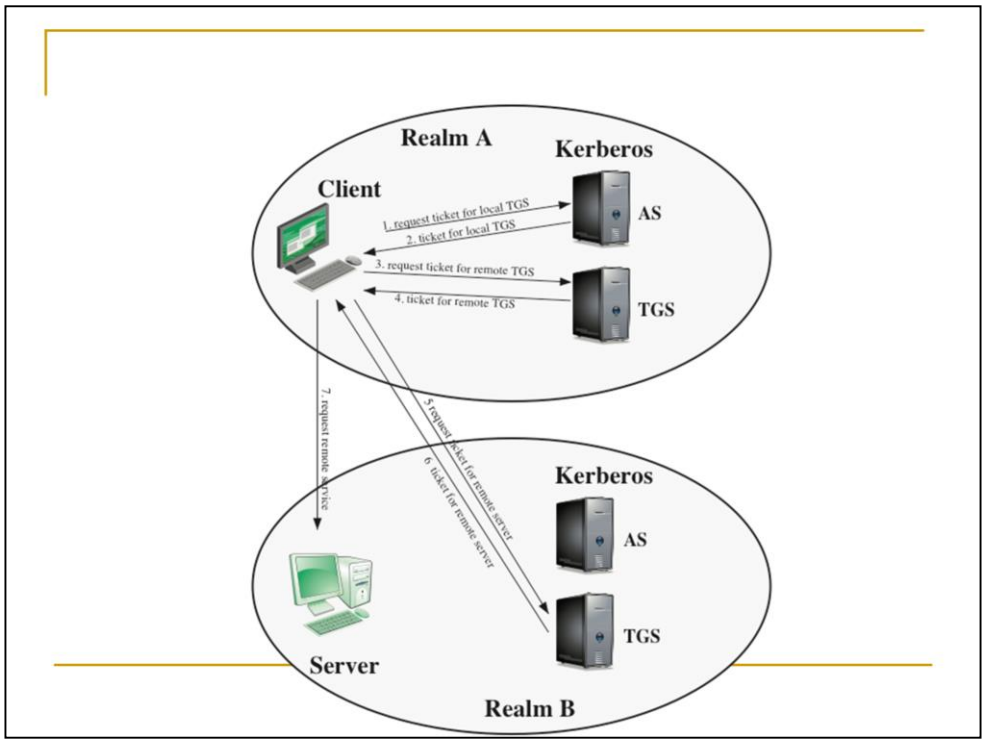


A related concept is that of a Kerberos principal ,

which is a service or user that is known to the Kerberos system. Each Kerberos

principal is identified by its principal name. Principal names consist of three parts: a

service or user name, an instance name, and a realm name.



With these ground rules in place, we can describe the mechanism as follows

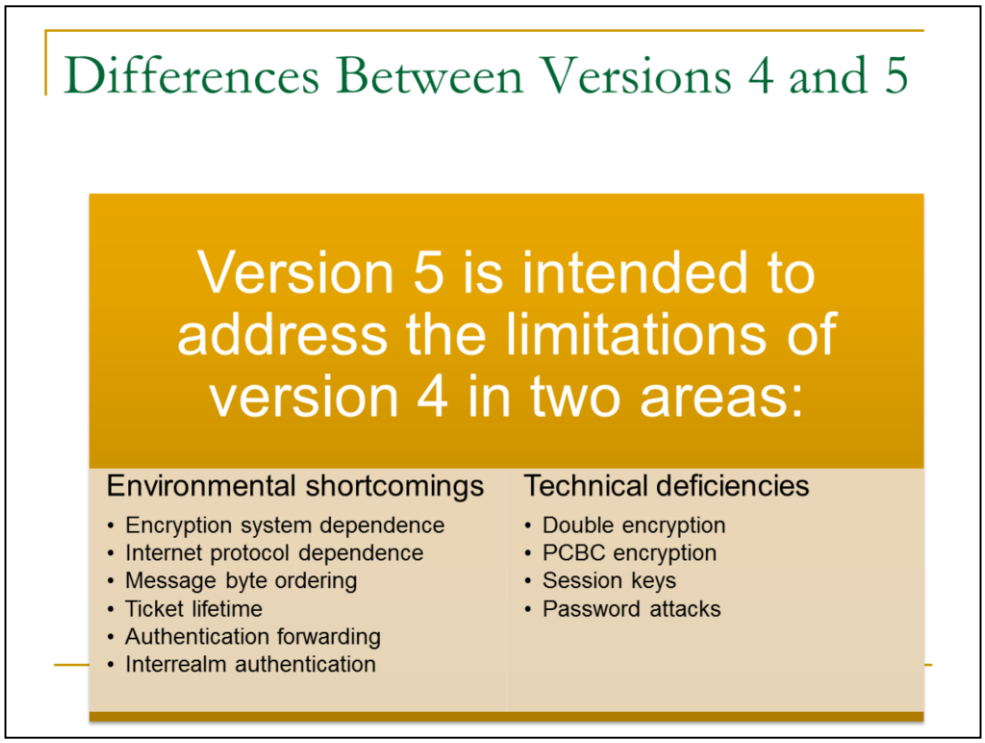
(Figure 15.3): A user wishing service on a server in another realm needs a ticket for

that server. The user's client follows the usual procedures to gain access to the local

TGS and then requests a ticket-granting ticket for a remote TGS (TGS in another

realm). The client can then apply to the remote TGS for a service-granting ticket for

the desired server in the realm of the remote TGS.



Version 5 is intended to address the limitations

of version 4 in two areas: environmental shortcomings and technical deficiencies.

Let us briefly summarize the improvements in each area.

Kerberos version 4 was developed for use within the Project Athena environment

and, accordingly, did not fully address the need to be of general purpose. This

led to the following environmental shortcomings .

1. Encryption system dependence: Version 4 requires the use of DES. Export

restriction on DES as well as doubts about the strength of DES were thus of

concern. In version 5, ciphertext is tagged with an encryption-type identifier so

that any encryption technique may be used. Encryption keys are tagged with a

type and a length, allowing the same key to be used in different algorithms and allowing the specification of different variations on a given algorithm.

2. Internet protocol dependence: Version 4 requires the use of Internet Protocol (IP) addresses. Other address types, such as the ISO network address, are not accommodated. Version 5 network addresses are tagged with type and length, allowing any network address type to be used.

3. Message byte ordering: In version 4, the sender of a message employs a byte ordering of its own choosing and tags the message to indicate least significant byte in lowest address or most significant byte in lowest address. This techniques works but does not follow established conventions. In version 5, all message structures are defined using Abstract Syntax Notation One (ASN.1) and Basic Encoding Rules (BER), which provide an unambiguous byte ordering.

4. Ticket lifetime: Lifetime values in version 4 are encoded in an 8-bit quantity in units of five minutes. Thus, the maximum lifetime that can be expressed is $2^8 * 5 =$ 1280 minutes (a little over 21 hours). This may be inadequate for some applications (e.g., a long-running simulation that requires valid Kerberos credentials throughout execution). In version 5, tickets include an explicit start time and end time, allowing tickets with arbitrary lifetimes.

5. Authentication forwarding: Version 4 does not allow credentials issued to one client to be forwarded to some other host and used by some other client. This capability would enable a client to access a server and have that server access another server on behalf of the client. For example, a client issues a request to a print server that then accesses the client's file from a file server, using the client's credentials for access. Version 5 provides this capability.

6. Interrealm authentication: In version 4, interoperability among N realms

requires on the order of N^2 Kerberos-to-Kerberos relationships, as described earlier. Version 5 supports a method that requires fewer relationships, as described shortly.

Apart from these environmental limitations, there are technical deficiencies in the version 4 protocol itself. Most of these deficiencies were documented in [BELL90], and version 5 attempts to address these. The deficiencies are the following.

1. Double encryption: Note in Table 15.1 [messages (2) and (4)] that tickets provided to clients are encrypted twice—once with the secret key of the target server and then again with a secret key known to the client. The second encryption is not necessary and is computationally wasteful.

2. PCBC encryption: Encryption in version 4 makes use of a nonstandard mode of DES known as propagating cipher block chaining (PCBC). It has been demonstrated that this mode is vulnerable to an attack involving the interchange of ciphertext blocks [KOHL89]. PCBC was intended to provide an integrity check as part of the encryption operation. Version 5 provides explicit integrity mechanisms, allowing the standard CBC mode to be used for encryption. In particular, a checksum or hash code is attached to the message prior to encryption using CBC.

3. Session keys: Each ticket includes a session key that is used by the client to encrypt the authenticator sent to the service associated with that ticket. In addition, the session key may subsequently be used by the client and the server to protect messages passed during that session. However, because the same ticket may be used repeatedly to gain service from a particular server, there is

the risk that an opponent will replay messages from an old session to the client or the server. In version 5, it is possible for a client and server to negotiate a subsession key, which is to be used only for that one connection. A new access by the client would result in the use of a new subsession key.

4. Password attacks: Both versions are vulnerable to a password attack. The message
from the AS to the client includes material encrypted with a key based on the client's password.10 An opponent can capture this message and attempt to decrypt it by trying various passwords. If the result of a test decryption is of the proper form, then the opponent has discovered the client's password and may subsequently use it to gain authentication credentials from Kerberos. This is the same type of password attack described in Chapter 21, with the same kinds of countermeasures being applicable. Version 5 does provide a mechanism known as pre-authentication, which should make password attacks more difficult, but it does not prevent them.

Table 15.3
Summary of Kerberos Version 5 Message Exchanges

(1) C → AS *Options* || *IDc* || *Realmc* || *IDtgs* || *Times* || *Nonce1*
(2) AS → C *Realmc* || *IDC* || *Tickettgs* || E(*Kc*, [*Kc,tgs* || *Times* || *Nonce1* || *Realmtgs* || *IDtgs*])
Tickettgs = E(*Ktgs*, [*Flags* || *Kc,tgs* || *Realmc* || *IDC* || *ADC* || *Times*])

(a) Authentication Service Exchange to obtain ticket-granting ticket

(3) C → TGS *Options* || *IDv* || *Times* || || *Nonce2* || *Tickettgs* || *Authenticatorc*
(4) TGS → C *Realmc* || *IDC* || *Ticketv* || E(*Kc,tgs*, [*Kc,v* || *Times* || *Nonce2* || *Realmv* || *IDv*])
Tickettgs = E(*Ktgs*, [*Flags* || *Kc,tgs* || *Realmc* || *IDC* || *ADC* || *Times*])
Ticketv = E(*Kv*, [*Flags* || *Kc,v* || *Realmc* || *IDC* || ADC || *Times*])
Authenticatorc = E(*Kc,tgs*, [*IDC* || *Realmc* || *TS1*])

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

(5) C → V *Options* || $Ticket_v$ || $Authenticator_c$
(6) V → C $E_{Kc,V}$ [TS_2 || *Subkey* || *Seq#*]
Ticketv = E(*Kv*, [Flags || *Kc,v* || *Realmc* || *IDC* || *ADC* || *Times*])
Authenticatorc = E(*Kc,v*, [*IDC* || *Realmc* || *TS2* || *Subkey* || *Seq#*])

(c) Client/Server Authentication Exchange to obtain service

Table 15.3 summarizes the basic version

5 dialogue. This is best explained by comparison with version 4 (Table 15.1).

Summary

- Kerberos