NSP:MODULE 1

1.Authentication Protocols

- protocols enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys.
- may be one-way or mutual.
- key issues are

➢ confidentiality

– To prevent masquerade and to prevent compromise of session keys essential identification and session key information must be communicated in encrypted form.

➢ Timeliness

– important because of the threat of the message replays.

– Such repalys at worst, could allow an opponent to compromise a session key or successfully impersonate another party.

– At minimum,a successful replay can disrupt operations by presenting parties with messages that appear genuine but are not

2. Replay Attacks

- where a valid signed message is copied and later resent.

– **simple replay**: The opponent simply copies a message and replays it later.

– **Repetition that can be logged**: An opponent can replay a time stamped message within the valid time window.

– **Repetition that cannot be detected**: This situation could arise because the original message could have been suppressed and thus did not arrive at its destination; only the replay message arrives.

– **Backward replay without modification**: This is a replay back to the message sender. This attack is possible if symmetric encryption is used and the sender cannot easily recognize the difference between messages sent and messages received on the basis of content.

Countermeasures of Replay Attacks:

- Attach a sequence numbers to each message used in an authentication exchange.
- New message is accepted only if its sequence number is in the proper order.
- Requires each party to keep track of the last sequence number for each claimant it has dealt with

➢ use of sequence numbers (generally impractical)

Approaches for Replay Attacks

- Timestamps (needs synchronized clocks):

➢ Party A accepts a message as fresh only if the message contains a timestamp that, in A's judgement is close enough to A's knowledge of current time.

➢ This approaches requires that clocks among various participants be synchronized.

- Challenge/response (using unique nonce)

➢ Party A, expecting a fresh message from B, first sends B a nonce(challenge) and requires that subsequent message(response) received from B contains the correct nonce value.

3. Mutual Authentication

- Involves the use of a trusted Key Distribution Center (KDC).
- Each party in the network shares a secret key, known as master key with KDC.
- KDC is responsible for generating keys to be used for a short time over a connection between two parties, known as session keys and for distributing thosekeys using master keys to protect the distribution.

4.Needham-Schroeder Protocol for secret key distribution using a KDC

- original third-party key distribution protocol
- symmetric key protocol
- Based on symmetric algorithm encryption
- Based on public key cryptography
- for session between A B mediated by KDC
- **protocol overview is:**

1. A→KDC: $ID_A \| ID_B \| N_1$ (A sends messages to KDC to identify herself and B.)
2. KDC→A: $E_{Ka}[Ks \| ID_B \| N_1 \| E_{Kb}[Ks \| ID_A]]$ (KDC generate a session key and sends encryption session key copies to A)
3. A→B: $E_{Kb}[Ks \| ID_A]$ (A sends encrypted session key to B where B can decrypt it using B's secret key)
4. B→A: $E_{Ks}[N_2]$ (B recovers the session key and send back nonce encrypted using session key.)
5. A→B: $E_{Ks}[f(N_2)]$ (A replies to B confirming session key and send back fresh message.)

- used to securely distribute a new session key for communications between A & B.
- but is vulnerable to a replay attack if an old session key has been compromised.
- then message 3 can be resent convincing B that is communicating with A.
- modifications to address this require:
  - **timestamps (Denning 81)**
  - **using an extra nonce (Neuman 93)**

5. (Denning 81)

1. IDa || IDb || N1
2. E (Ka, [Ks || IDB || N1 || E (Kb, [Ks || IDA])]) + T
3. E (Kb, [Ks || IDA]) + T
4. E (Ks, N2)
5. E (Ks, f(N2))

To overcome this weakness, the Needham Schroeder protocol is modified to include the addition of a timestamp to steps 2 and 3.

1. IDa || IDb || N1
2. E (Ka, [Ks || IDB || T || N1 || E (Kb, [Ks || IDA || T])])
3. E (Kb, [Ks || IDA || T])
4. E (Ks, N2)
5. E (Ks, f(N2))

To overcome this weakness, the Needham Schroeder protocol is modified to include the addition of a timestamp to steps 2 and 3.

T is a timestamp that assures Alice and Bob that the session key has only just been generated.

Thus , both Alice and Bob know that the key distribution is a fresh exchange

- Timestamp T is encrypted using secured master keys.
- If X is even with knowledge of an old session keys cannot succeed because a replay of step 3 wiil be detected by Bob(B) as untimely.

One-Way Authentication

- required when sender & receiver are not in communications at same time (eg. email)
- have header in clear so can be delivered by email system
- may want contents of body protected & sender authenticated Using Symmetric Encryption
- can refine use of KDC but can't have final exchange of nonces

6.(Neuman 93)

1. A -> B : *IDA* | | *Na*
2. B -> KDC : *IDB* | | *Nb* | | E(*Kb*, [*IDA* | | *Na* | | *Tb*])
3. KDC -> A : E(*Ka*, [*IDB* || *Na* || *Ks* || *Tb*]) || E(*Kb* , [*IDA* || *Ks* || *Tb*]) || *Nb*
4. A:B_> E(*Kb*, [*IDA* || *Ks* || *Tb*]) || E(*Ks*, *Nb*)

Explanation:

1. A initiates the authentication exchange by generating a nonce and sending that plus its identifier to B in plaintext.
2. B alerts the KDC that a session key is needed. Its message to the KDC includes its identifier and B's nonce. The message also includes a block encrypted with the secret key shared by B and the KDC. The block contains the identifier for A, A's Nonce and expiration time.
3. KDC passes on to A, b's Nonce and a block encrypted with the secret key that B shares with KDC. The block serves as a ticket that can be used by A for subsequent authentications.
4. A transmits a ticket to B together with B's Nonce, the latter encrypted using session key. The ticket provides B with secret key that is used to decrypt and recover the Nonce.

Public-Key Approaches:

- have seen some public-key approaches
- if confidentiality is major concern, can use:
- $A \rightarrow B: E_{KUb}[Ks] \| E_{Ks}[M]$
- has encrypted session key, encrypted message
- if authentication needed use a digital signature with a digital certificate:
- $A \rightarrow B: M \| E_{KRa}[H(M)] \| E_{KRas}[T \| ID_A \| KU_a]$

6. KERBEROS

- Kerberos is an authentication service developed at MIT, and is one of the best known and most widely implemented **trusted third party** key distribution systems.

- Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Unlike most other authentication schemes, Kerberos relies exclusively on symmetric encryption, making no use of public-key encryption. Two versions of Kerberos are in common use: **v4 & v5.**
- Users wish to access services on servers.
- Three threats exist:
  - User pretend to be another user.
  - User alter the network address of a workstation.
  - User eavesdrop on exchanges and use a replay attack.

Kerberos Requirements

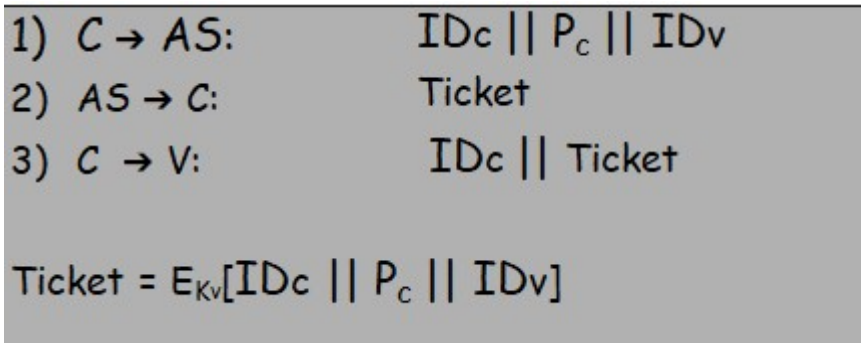
- its first report identified requirements as:
  - **Secure**
    - A network eavesdropper should not be able to obtain the necessary information to impersonate a user.
    - strong enough such that a potential opponent does not find it to be the weak link.
  - **Reliable**
    - should be highly reliable
    - should employ a distributed server architecture,
    - one system able to back up another.
  - **Transparent**
    - user should not be aware that authentication is taking place beyond the requirement to enter a password.
  - **Scalable**
    - capable of supporting large numbers of clients and servers ▪ modular, distributed architecture.
    - implemented using an authentication protocol based on NeedhamSchroeder

Kerberos v4 Overview

- 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.
- a basic third-party authentication scheme
- **have an Authentication Server (AS)**
  – users initially negotiate with AS to identify self.
    - AS provides a non-corruptible authentication credential (ticket granting ticket TGT)
- **have a Ticket Granting server (TGS)**
    - users subsequently request access to other services from TGS on basis of users TGT
- **Terms:**
  - C = Client
  - AS = authentication server
  - V = server
  - IDc = identifier of user on C
  - IDv = identifier of V
  - P_c= password of user on C
  - ADc = network address of C

- Kv = secret encryption key shared by AS and V
- TS = timestamp
- || = concatenation

A Simple Authentication



In this scenario, the user logs on to a workstation and requests access to server V. The client module C in the user's workstation requests the user's password and then sends a message to the AS that includes the user's ID, the server's ID, and the user's password. 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. This ticket is then sent back to C. Because the ticket is encrypted, it cannot be altered by C or by an opponent. With this ticket, C can now apply to V for service. C sends a message to V containing C's ID and the ticket. V decrypts the ticket and verifies that the user ID in the ticket is the same as the unencrypted user ID in the message. If these two match, the server considers the user authenticated and grants the requested service

Kerberos v4 Dialogue

1. obtain ticket granting ticket from AS
   - once per session
2. obtain service granting ticket from TGT
   - for each distinct service required
3. client/server exchange to obtain service
   - on every service request

A More Secure Authentication Dialogue

- Problems:
  - Lifetime associated with the ticket-granting ticket
  - If too short ➔ repeatedly asked for password
  - If too long ➔ greater opportunity to replay
  - The threat is that an opponent will steal the ticket and use it before it expires

To solve these additional problems, we introduce a scheme for avoiding plaintext passwords and a new server, known as the ticket-granting server (TGS).

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. The simple authentication dialogue is as follows:

A more secure authentication dialogue

There are two major problems associated with the previous approach:

Plaintext transmission of the password.

Each time a user has to enter the password.

To solve these problems, we introduce a scheme for avoiding plaintext passwords, and anew server, known as ticket granting server (TGS). The hypothetical scenario is as follows:

Once per user logon session:

1. C >> AS: IDc||IDtgs
2. AS >> C: Ek_C ($Ticket_{tgs}$)

Once per type of service:

3. C >> TGS: IDc||IDv||$Ticket_{tgs}$
4. TGS >> C: $ticket_V$

Once per service session:

5. C >> V: IDc||$ticket_V$

$Ticket_{tgs}$= Ek_{tgs}(IDc||ADc||IDtgs||TS1||Lifetime1) $Ticket_V$= Ek_V(IDc||ADc||IDv||TS2||Lifetime2)

C: Client, AS: Authentication Server, V: Server, IDc : ID of the client, Pc:Password of the client, ADc: Address of client, IDv : ID of the server, Kv: secret key shared by AS and V, ||: concatenation, IDtgs: ID of the TGS server, TS1, TS2: time stamps, lifetime: lifetime of the ticket.

The new service, TGS, issues tickets to users who have been authenticated to AS. Thus, the user first requests a ticket-granting ticket ($Ticket_{tgs}$) 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. Let us look at the details of this scheme:

1. The client requests a ticket-granting ticket on behalf of the user by sending its user's ID and password to the AS, together with the TGS ID, indicating a request to use the TGS service.

2. The AS responds with a ticket that is encrypted with a key that is derived from the user's password.

When this response arrives at the client, the client prompts the user for his or her password, generates the key, and attempts to decrypt the incoming message.

If the correct password is supplied, the ticket is successfully recovered.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.

Now that the client has a ticket-granting ticket, access to any server can be obtained with steps 3 and 4:

3. The client requests a service-granting ticket on behalf of the user. For this purpose, the client transmits a message to the TGS containing the user's ID, the ID of the desired service, and the ticket-granting ticket.

4. The TGS decrypts the incoming ticket and verifies the success of the decryption by the presence of its ID. It checks to make sure that the lifetime has not expired. Then it compares the user ID and network address with the incoming information to authenticate the user. If the user is permitted access to the server V, the TGS issues a ticket to grant access to the requested service.

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 (K_V) known only to the TGS and the server, preventing alteration.

Finally, with a particular service-granting ticket, the client can gain access to the corresponding service with step 5:

5. The client requests access to a service on behalf of the user. For this purpose, the client transmits a message to the server containing the user's ID nd the service-granting ticket. The server authenticates by using the contents of the ticket.

This new scenario satisfies the two requirements of only one password query per user session and

protection of the user password.

Kerbero V4 Authentication Dialogue Message Exchange

Two additional problems remain in the more secure authentication dialogue:

Lifetime associated with the ticket granting ticket. If the lifetime is very short, then the user will be repeatedly asked for a password. If the lifetime is long, then the opponent has the greater opportunity for replay.

Requirement for the servers to authenticate themselves to users. The actual Kerberos protocol version 4 is as follows:

- a basic third-party authentication scheme
- have an Authentication Server (AS)

– users initially negotiate with AS to identify self

–AS provides a non-corruptible authentication credential (ticket granting ticket TGT)

- have a Ticket Granting server (TGS)

–users subsequently request access to other services from TGS on basis of users

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 assword, and p t ctin contents of m ssa e (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 t k
ID_{tgs}	Confirms that this ticket is for the TGS

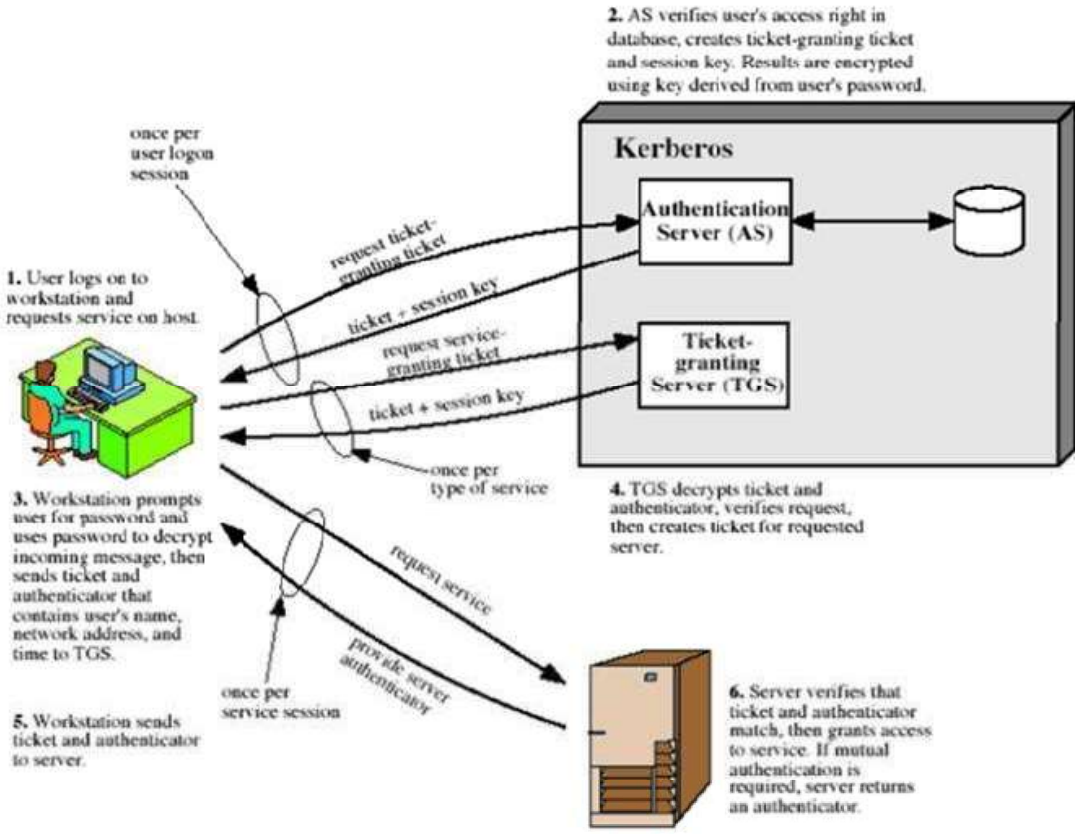
The table given below illustrates the mode of dialogue in V4

(a) Authentication Service Exchange: to obtain ticket-granting ticket
(1) C → **AS:** $ID_c \parallel ID_{tgs} \parallel TS_1$
(2) **AS** → **C:** $\mathrm{E}_{K_c}\left[K_{c,tgs} \parallel ID_{tgs} \parallel TS_2 \parallel Lifetime_2 \parallel Ticket_{tgs}\right]$
$Ticket_{tgs} = \mathrm{E}_{K_{tgs}}\left[K_{c,tgs} \parallel ID_C \parallel AD_C \parallel ID_{tgs} \parallel TS_2 \parallel Lifetime_2\right]$
(b) Ticket-Granting Service Exchange: to obtain service-granting ticket
(3) C → **TGS:** $ID_v \parallel Ticket_{tgs} \parallel Authenticator_c$
(4) **TGS** → **C:** $\mathrm{E}_{K_{c,tgs}}\left[K_{c,v} \parallel ID_v \parallel TS_4 \parallel Ticket_v\right]$
$Ticket_{tgs} = \mathrm{E}_{K_{tgs}}\left[K_{c,tgs} \parallel ID_C \parallel AD_C \parallel ID_{tgs} \parallel TS_2 \parallel Lifetime_2\right]$
$Ticket_v = \mathrm{E}_{K_v}\left[K_{c,v} \parallel ID_C \parallel AD_C \parallel ID_v \parallel TS_4 \parallel Lifetime_4\right]$
$Authenticator_c = \mathrm{E}_{K_{tgs}}\left[ID_C \parallel AD_C \parallel TS_3\right]$
(c) Client/Server Authentication Exchange: to obtain service
(5) C → **V:** $Ticket_v \parallel Authenticator_c$
(6) **V** → **C:** $\mathrm{E}_{K_{c,v}}\left[TS_5 + 1\right]$ (for mutual authentication)
$Ticket_v = \mathrm{E}_{K_v}\left[K_{c,v} \parallel ID_C \parallel AD_C \parallel ID_v \parallel TS_4 \parallel Lifetime_4\right]$
$Authenticator_c = \mathrm{E}_{K_{c,v}}\left[ID_C \parallel AD_C \parallel TS_5\right]$

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
	(a) 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
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
	(b) Ticket-Granting Service Exchange
Message (5)	**Client requests service**
$Ticket_V$	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
IDC	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
IDC	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
	(c) Client/Server Authentication

Kerberos 4 Overview



Kerberos Realms and Multiple Kerberi

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.Such an environment is referred to as a **Kerberos realm**.

The concept of *realm* can be explained as follows.

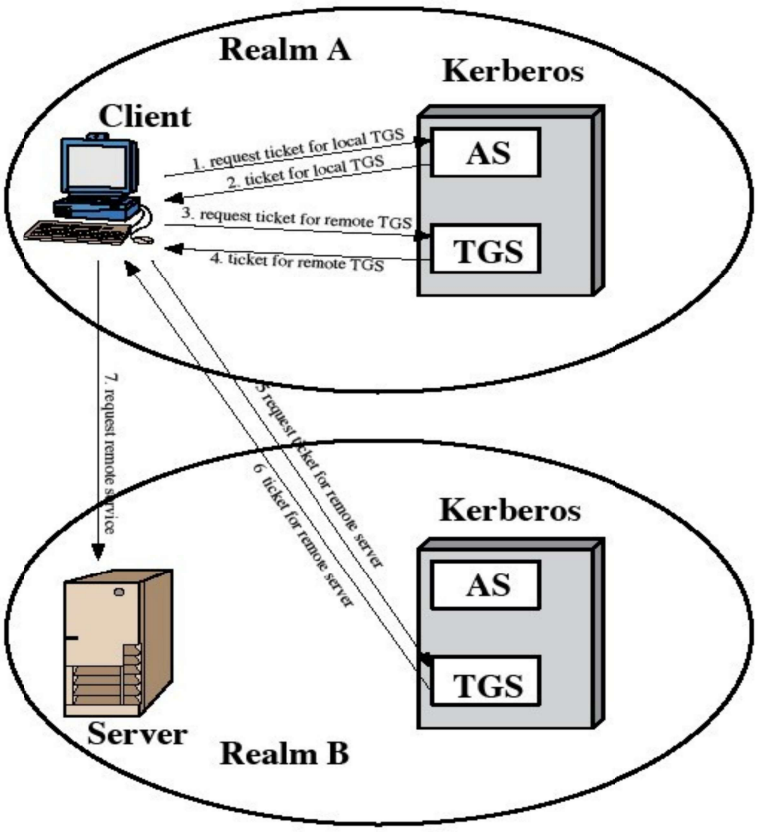


Figure 14.2 Request for Service in Another Realm

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

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 version 5

Version 5 of Kerberos provides a number of improvements over version 4.

- developed in mid 1990's
- provides improvements over v4
  - addresses environmental shortcomings and technical deficiencies
- specified as Internet standard RFC 1510

Differences between version 4 and 5

Version 5 is intended to address the limitations of version 4 in two areas:

Environmental shortcomings

- o encryption system dependence
- o internet protocol dependence
- o message byte ordering
- o ticket lifetime
- o authentication forwarding
- o inter-realm authenticaiton

Technical deficiencies

- o double encryption o PCBC encryption o Session keys
- o Password attacks

The version 5 authentication dialogue

(a) Authentication Service Exchange: to obtain ticket-granting ticket
(1) C → AS: Options ‖ ID_c ‖ $Realm_c$ ‖ ID_{tgs} ‖ $Times$ ‖ $Nonce_1$
(2) AS → C: $Realm_c \| ID_C \| Ticket_{tgs} \| E_{K_c}[K_{c,tgs} \| Times \| Nonce_1 \| Realm_{tgs} \| ID_{tgs}]$
$Ticket_{tgs} = E_{K_{tgs}}[Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times]$
(b) Ticket-Granting Service Exchange: to obtain service-granting ticket
(3) C → TGS: Options ‖ ID_v ‖ $Times$ ‖ ‖ $Nonce_2$ ‖ $Ticket_{tgs}$ ‖ $Authenticator_c$
(4) TGS → C: $Realm_c \| ID_C \| Ticket_v \| E_{K_{c,tgs}}[K_{c,v} \| Times \| Nonce_2 \| Realm_v \| ID_V]$
$Ticket_{tgs} = E_{K_{tgs}}[Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times]$
$Ticket_v = E_{K_v}[Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times]$
$Authenticator_c = E_{K_{c,tgs}}[ID_C \| Realm_c \| TS_1]$
(c) Client/Server Authentication Exchange: to obtain service
(5) C → V: Options ‖ $Ticket_v$ ‖ $Authenticator_c$
(6) V → C: $E_{K_{c,V}}$ [TS_2 ‖ Subkey ‖ Seq#]
$Ticket_v = E_{K_v}[Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times]$
$Authenticator_c = E_{K_{c,V}}[ID_C \| Realm_c \| TS_2 \| Subkey \| Seq\#]$

irst, consider the authentication service exchange. Message (1) is a client request for a ticket-granting ticket. As before, it includes the ID of the user and the TGS. The following new

elements are added:

Realm: Indicates realm of user

Options: Used to request that certain flags be set in the returned ticket

Times: Used by the client to request the following time settings in the ticket:

from: the desired start time for the requested ticket

till: the requested expiration time for the requested ticket rtime: requested renew-till time

Nonce: A random value to be repeated in message (2) to assure that the response is fresh and has not been replayed by an opponent

Message (2) returns a ticket-granting ticket, identifying information for the client, and a block encrypted using the encryption key based on the user's password.
This block includes the session key to be used between the client and the TGS, times specified in message (1), the nonce from message (1), and TGS identifying information.

The ticket itself includes the session key, identifying information for the client, the requested time values, and flags that reflect the status of this ticket and the requested options.

These flags introduce significant new functionality to version 5. For now, we defer a discussion of these flags and concentrate on the overall structure of the version 5 protocol.

Let us now compare the ticket-granting service exchange for versions 4 and 5. We see that message (3) for both versions includes an authenticator, a ticket, and the name of the requested service.

In addition, version 5 includes requested times and options for the ticket and a nonce, all with functions similar to those of message (1).

The authenticator itself is essentially the same as the one used in version 4.

Message (4) has the same structure as message (2), returning a ticket plus information needed by

the client, the latter encrypted with the session key now shared by the client and the TGS.

Finally, for the client/server authentication exchange, several new features appear in version 5. In message (5), the client may request as an option that mutual authentication is required. The authenticator includes several new fields as follows:

Subkey: The client's choice for an encryption key to be used to protect this specific application session. If this field is omitted, the session key from the ticket ($K_{c,v}$) is used.

Sequence number: An optional field that specifies the starting sequence number to be use may be sequence numbered to detect replays.

If mutual authentication is required, the server responds with message (6). This message includes the timestamp from the authenticator. Note that in version 4, the timestamp was incremented by one. This is not necessary in version 5 because the nature of the format of messages is such that it is not possible for an opponent to create message (6) without knowledge of the appropriate encryption keys.

Ticket Flags

The flags field included in tickets in version 5 supports expanded functionality compared to that available in version 4.

X.509 Certificates

Overview:

- **issued by a Certification Authority (CA), containing:**

– version (1, 2, or 3)

– serial number (unique within CA) identifying certificate

– signature algorithm identifier

- issuer X.500 name (CA)
- period of validity (from - to dates)
- subject X.500 name (name of owner)
- subject public-key info (algorithm, parameters, key)
- issuer unique identifier (v2+)
- subject unique identifier (v2+)
- extension fields (v3)
- signature (of hash of all fields in certificate)

- **notation CA<<A>> denotes certificate for A signed by CA**

X.509 defines a framework for the provision of authentication services by the X.500 directory to its users. The directory may serve as a repository of public-key certificates. Each certificate contains the public key of a user and is signed with the private key of a trusted certification authority. In addition, X.509 defines alternative authentication protocols based on the use of public-key certificates.

X.509 is an important standard because the certificate structure and authentication protocols defined in X.509 are used in a variety of contexts. For example, the X.509 certificate format is used in S/MIME), IP Security and SSL/TLS and SET

X.509 is based on the use of public-key cryptography and digital signatures. The standard does not

dictate the use of a specific algorithm but recommends RSA. The digital signature scheme is assumed to require the use of a hash function.

Certificates

The heart of the X.509 scheme is the public-key certificate associated with each user. These user certificates are assumed to be created by some trusted certification authority (CA) and placed in the directory by the CA or by the user.

Version:

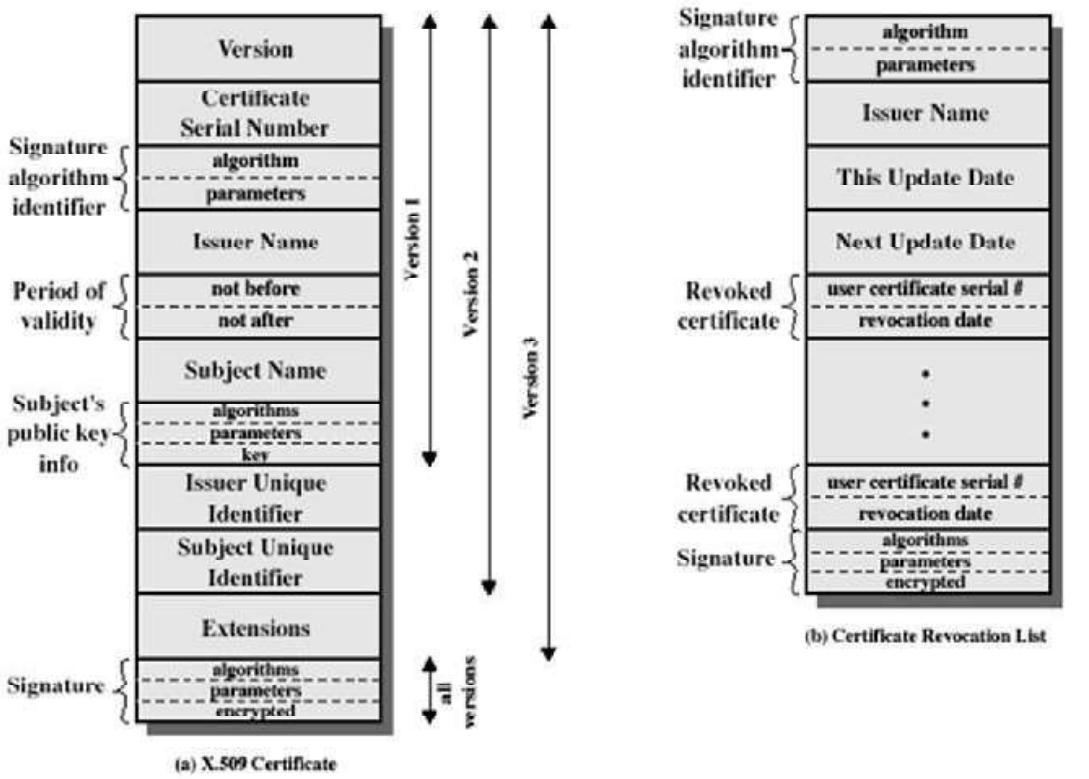
Differentiates among successive versions of the certificate format; the default is version 1. If the Issuer Unique Identifier or Subject Unique Identifier are present, the value must be version 2. If one or more extensions are present, the version must be version 3.

Serial number:

An integer value, unique within the issuing CA, that is unambiguously associated with this certificate.

Signature algorithm identifier:

The algorithm used to sign the certificate, together with any associated parameters. Because this information is repeated in the Signature field at the end of the certificate, this field has little, if any, utility.



Issuer name:

X.500 name of the CA that created and signed this certificate.

Period of validity:

Consists of two dates: the first and last on which the certificate is valid.

Subject name:

The name of the user to whom this certificate refers. That is, this certificate certifies the public key of the subject who holds the corresponding private key.

Subject's public-key information:

The public key of the subject, plus an identifier of the algorithm for which this key is to be used, together with any associated parameters.

Issuer unique identifier:

An optional bit string field used to identify uniquely the issuing CA in the event the X.500 name has been reused for different entities.

Subject unique identifier:

An optional bit string field used to identify uniquely the subject in the event the X.500 name has been reused for different entities.

Extensions:

A set of one or more extension fields. Extensions were added in version 3 and are discussed later in this section.

Signature:

Covers all of the other fields of the certificate; it contains the hash code of the other fields, encrypted with the CA's private key. This field includes the signature algorithm identifie

The standard uses the following notation to define a certificate: CA<<A>> = CA {V, SN, AI, CA, T_A, A, Ap} where

Y <<X>> = the certificate of user X issued by certification authority Y Y {I} = the signing of I by Y. It consists of I with an encrypted hash code appended

The CA signs the certificate with its private key. If the corresponding public key is known to a user, then that user can verify that a certificate signed by the CA is valid.

Obtaining a User's Certificate

User certificates generated by a CA have the following characteristics:

Any user with access to the public key of the CA can verify the user public key that was certified.

No party other than the certification authority can modify the certificate without this being detected.

ecause certificates are unforgeable, they can be placed in a directory without the need for the directory to make special efforts to protect them.

If all users subscribe to the same CA, then there is a common trust of that CA. All user certificates can be placed in the directory for access by all users.

If there is a large community of users, it may not be practical for all users to subscribe to the same CA. Because it is the CA that signs certificates, each participating user must have a copy of the CA's own public key to verify signatures. This public key must be provided to each user in an absolutely secure (with respect to integrity and authenticity) way so that the user has confidence in the associated certificates. Thus, with many users, it may be more practical for there to be a number of CAs, each of which securely provides its public key to some fraction of the users.

Now suppose that A has obtained a certificate from certification authority X_1 and B has obtained a certificate from CA X_2. If A does not securely know the public key of X_2, then B's certificate, issued by X_2, is useless to A.

A can read B's certificate, but A cannot verify the signature. However, if the two CAs have securely exchanged their own public keys, the following procedure will enable A to obtain B's public key:

1. A obtains, from the directory, the certificate of X_2 signed by X_1. Because A securely knows X_1's public key, A can obtain X_2's public key from its certificate and verify it by means of X_1's signature on the certificate.
2. A then goes back to the directory and obtains the certificate of B signed by X_2

Because A now has a trusted copy of X_2's public key, A can verify the signature and securely obtain B's public key.

A has used a chain of certificates to obtain B's public key. In the notation of X.509, this chain is expressed as

X_1<<X_2>> X_2 <<B>>

In the same fashion, B can obtain A's public key with the reverse chain: X_2<<X_1>> X_1 <<A>>

This scheme need not be limited to a chain of two certificates. An arbitrarily long path of CAs can be followed to produce a chain. A chain with N elements would be expressed as

X_1<<X_2>> X_2 <<X_3>>... X_N<<B>>

In this case, each pair of CAs in the chain (X_i, X_{i+1}) must have created certificates for each other.

All these certificates of CAs by CAs need to appear in the directory, and the user needs to know how they are linked to follow a path to another user's public-key certificate. X.509 suggests that CAs be arranged in a hierarchy so that navigation is straightforward.

Figure 14.5, taken from X.509, is an example of such a hierarchy. The connected circles indicate the hierarchical relationship among the CAs; the associated boxes indicate certificates maintained in the directory for each CA entry. The directory entry for each CA includes two types of certificates:

Forward certificates: Certificates of X generated by other CAs

Reverse certificates: Certificates generated by X that are the certificates of other CAs

CA Hierarchy Use

In the example given below , user A can acquire the following certificates from the directory to establish a certification path to B:

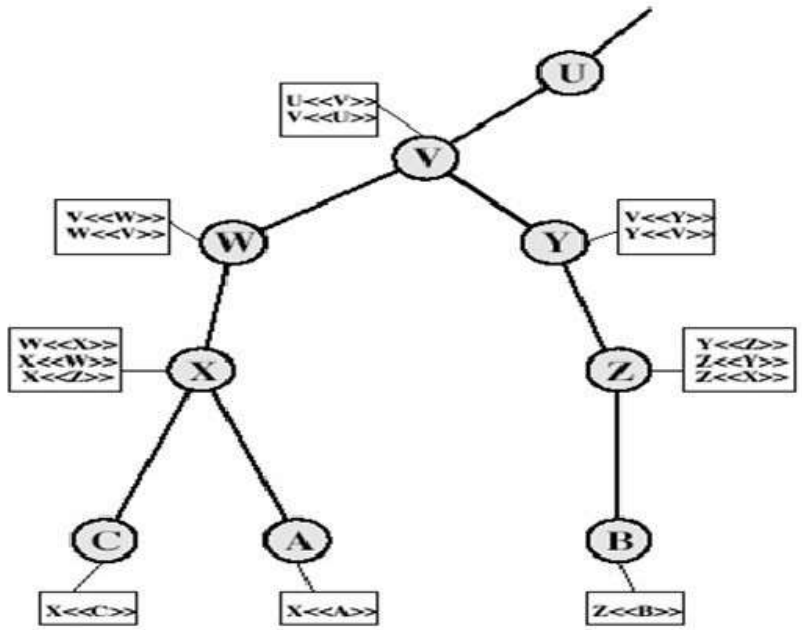
X<<W>> W <<V>> V <<Y>> <<Z>> Z <<B>>

When A has obtained these certificates, it can unwrap the certification path in sequence to recover a trusted copy of B's public key. Using this public key, A can send encrypted

Messages to B. If A wishes to receive encrypted messages back from B, or to sign messages sent to B, then B will require A's public key, which can be obtained from the following certification path:

Z<<Y>> Y <<V>> V <<W>> W <<X>>X <<A>>

B can obtain this set of certificates from the directory, or A can provide them as part of its initial message to B.



Certificate Revocation

- Certificates have a period of validity
- may need to revoke before expiry, for the following reasons eg:

1. user's private key is compromised
2. User is no longer certified by this CA

3. CA's certificate is compromised

- CA's maintain list of revoked certificates

1. the Certificate Revocation List (CRL)

- users should check certs with CA's CRL

Authentication Procedures

X.509 includes three alternative authentication procedures:

- **One-Way Authentication**
- **Two-Way Authentication**
- **Three-Way Authentication**
- all use public-key signatures

One-Way Authentication

- 1 message (A->B) used to establish

– the identity of A and that message is from A
– message was intended for B
– integrity & originality of message

- message must include timestamp, nonce, B's identity and is signed by A

Two-Way Authentication

- 2 messages (A->B, B->A) which also establishes in addition:

– the identity of B and that reply is from B
– that reply is intended for A
– integrity & originality of reply

reply includes original nonce from A, also timestamp and nonce from B

Three-Way Authentication

- 3 messages (A->B, B->A, A->B) which enables above authentication without synchronized clocks
- has reply from a back to B containing signed copy of nonce from B
- means that timestamps need not be checked or relied upon

X.509 Version 3

The X.509 version 2 format does not convey all of the information that recent design and implementation experience has shown to be needed. [FORD95] lists the following requirements not satisfied by version 2:

1. The Subject field is inadequate to convey the identity of a key owner to a public- key user.
2. The Subject field is also inadequate for many applications, which typically recognize entities by an Internet e-mail address, a URL, or some other Internet- related identification.
3. There is a need to indicate security policy information. There is a need to limit the damage that can result from a faulty or malicious CA by setting constraints on the applicability of a particular certificate.
4. It is important to be able to identify different keys used by the same owner at different times.

The certificate extensions fall into three main categories: key and policy information, subject and issuer attributes, and certification path constraints.

Key and Policy Information

These extensions convey additional information about the subject and issuer keys, plus indicators of certificate policy.. For example, a policy might be applicable to the authentication of electronic data interchange (EDI) transactions for the trading of goods within a given price range.

This area includes the following:

Authority key identifier: Identifies the public key to be used to verify the signature on this certificate or CRL.

Subject key identifier: Identifies the public key being certified. Useful for subject key pair updating.

Key usage: Indicates a restriction imposed as to the purposes for which, and the policies under which, the certified public key may be used.

Private-key usage period: Indicates the period of use of the private key corresponding to the public key.. For example, with digital signature keys, the usage period for the signing private key is typically shorter than that for the verifying public key.

Certificate policies: Certificates may be used in environments where multiple policies apply.

Policy mappings: Used only in certificates for CAs issued by other CAs.

Certificate Subject and Issuer Attributes

These extensions support alternative names, in alternative formats, for a certificate subject or certificate issuer and can convey additional information about the certificate subject, to increase a certificate user's confidence that the certificate subject is a particular person or entity. For example, information such as postal address, position within a corporation, or picture image may be required. The extension fields in this area include the following:

Subject alternative name: Contains one or more alternative names, using any of a variety of forms

Subject directory attributes: Conveys any desired X.500 directory attribute values for the subject of this certificate.

Certification Path Constraints

These extensions allow constraint specifications to be included in certificates issued for CAs by other CAs.The extension fields in this area include the following:

Basic constraints: Indicates if the subject may act as a CA. If so, a certification path length constraint may be specified.

Name constraints: Indicates a name space within which all subject names in subsequent certificates in a certification path must be located.

Policy constraints: Specifies constraints that may require explicit certificate policy identification or inhibit policy mapping for the remainder of the certification path.

Module II

ELECTRONIC MAIL SECURITY PRETTY GOOD PRIVACY (PGP)

PGP provides the confidentiality and authentication service that can be used for electronic mail and file storage applications. The steps involved in PGP are

Select the best available cryptographic algorithms as building blocks.

Integrate these algorithms into a general purpose application that is independent of operating system and processor and that is based on a small set of easy-to-use commands.

Make the package and its documentation, including the source code, freely available via the internet, bulletin boards and commercial networks.

Enter into an agreement with a company to provide a fully compatible, low cost