2. **Higher Level:** Lower layer functions are used to create a protocol that enables a receiver to verify the authenticity of message

The different types of functions that may be used to produce an authenticator are as follows:

1. **Message encryption:** The cipher text of the entire message serves as its authenticator.

2. **Message Authentication Code (MAC):** A public function of the message and a secret key that produces a fixed length value serves as the authenticator.

3. **Hash function:** A public function that maps a message of any length into a fixed length hash value, which serves as the authenticator.

## KERBEROS

Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Kerberos relies exclusively on conventional encryption, making no use of public-key encryption.

### Motivation

A distributed architecture consists of dedicated user workstations (clients) and distributed or centralized servers. In this environment, there are three approaches to security:

- Rely on each individual client workstation to assure the identity of its user or users and rely on each server to enforce a security policy based on user identification (ID).
- Require that client systems authenticate themselves to servers, but trust the client system concerning the identity of its user.
- Require the user to prove his or her identity for each service invoked. Also require that servers prove their identity to clients.

The following are the **requirements for Kerberos**:

- **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 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 Needham and Schroeder.

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

Two versions of Kerberos are in common use. **Version 4** and **Version 5**

## Kerberos Version 4

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

### Simple Authentication Dialogue

In an unprotected network environment, any client can apply to any server for service. The obvious security risk is that of impersonation. To counter this threat, servers must be able to confirm the identities of clients who request service. 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. The simple authentication dialogue is as follows:

**1.** C >> AS: $ID_c||P_c||ID_v$

**2.** AS >> C: Ticket

**3.** C >> V: $ID_c||$Ticket

Ticket= $EK_v(ID_c||ADc||ID_v)$

C : Client,
AS : Authentication Server,
V : Server, $ID_c$ : ID of the client,
$P_c$ : Password of the client,
$AD_c$ : Address of client, $ID_v$ : ID of the server,
$K_v$ : secret key shared by AS and V,
|| : concatenation.

### 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 a new server, known as ticket granting server (TGS). The hypothetical scenario is as follows:

**Once per user logon session:-**

1. C >> AS: $ID_c||ID_{tgs}$
2. AS >> C: $Ek_c(Ticket_{tgs})$

**Once per type of service:**

3. C >> TGS: $ID_c||ID_v||Ticket_{tgs}$
4. TGS >> C: $ticket_v$

**Once per service session:**

5. C >> V: $ID_c||\ Ticket_v$

$Ticket_{tgs}= Ekt_{gs}(ID_c||AD_c||IDt_{gs}||TS_1||Lifetime_1)$

$Ticket_v= Ek_v(ID_c||AD_c||ID_v||TS2||Lifetime_2)$

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 and 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.

**Kerberos 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
    - Users subsequently request access to other services from TGS on basis of users TGT

| (a) Authentication service exchange: to obtain ticket granting ticket |
| --- |
| (1) $C \rightarrow AS : ID_C \text{ II } ID_{tgs} \text{ II } TS_1$<br><br>(2) $AS \rightarrow C : EKc [ K_{c,tgs} \text{ II } ID_{tgs} \text{ II } TS_2 \text{ II } Lifetime_2 \text{ II } Ticket_{tgs}]$ |
| (b) Ticket-Granting Service Exchange: to obtain service-granting ticket |
| |

(3) $C \rightarrow TGS: ID_v \parallel Ticket_{tgs} \parallel Authenticator_c$

(4) $TGS \rightarrow C: EK_{c,tgs}[K_{c,y} \parallel ID_v \parallel TS_4 \parallel Ticket_v]$

$Ticket_{tgs} = E_{K,tgs}[K_{c,tgs} \parallel ID_C \parallel AD_C \parallel ID_{tgs} \parallel TS_2 \parallel Lifetime_2]$

$Ticket_v = E_{Kv}[K_{c,v} \parallel ID_C \parallel AD_C \parallel ID_v \parallel TS_4 \parallel Lifetime_4]$

$Authenticator_C = E_{Ktgs}[ID_C \parallel AD_C \parallel TS_3]$

(c) Client/Server Authentication Exchange: to obtain service

(5) $C \rightarrow V : Ticket_v \parallel Authenticator_c$

(6) $V \rightarrow C: E_{kc,v}[TS_5 + 1]$

$Ticket_v = EK_v[K_{c,v} \parallel ID_C \parallel AD_C \parallel Id_v \parallel TS_4 \parallel Lifetime_4]$

$Authenticator_c = EK_{tgs}[ID_C \parallel AD_C \parallel TS_3]$

**Kerberos 4 Overview**

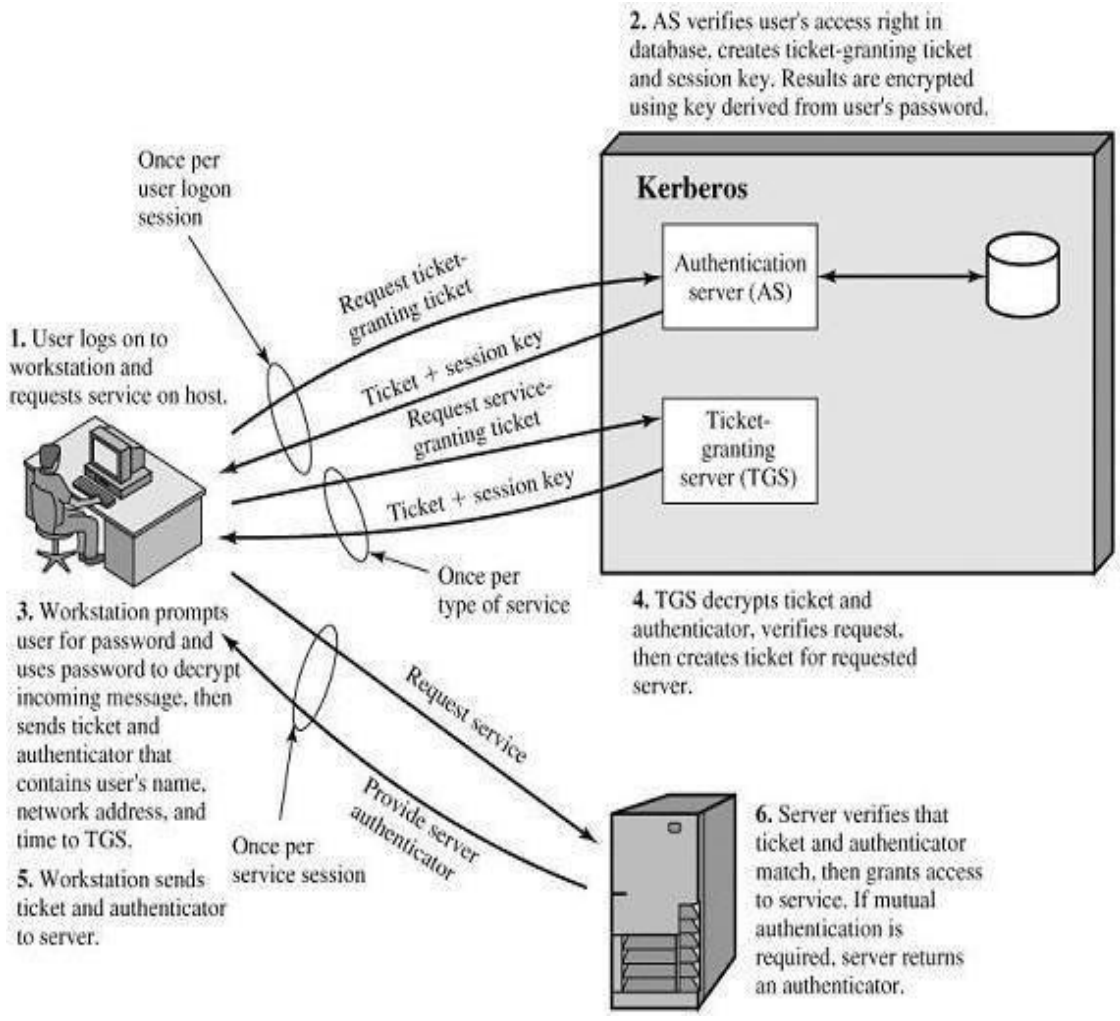


**Fig 4.1 Overview of Kerberos 4**

**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:

4. 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.
5. 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.**

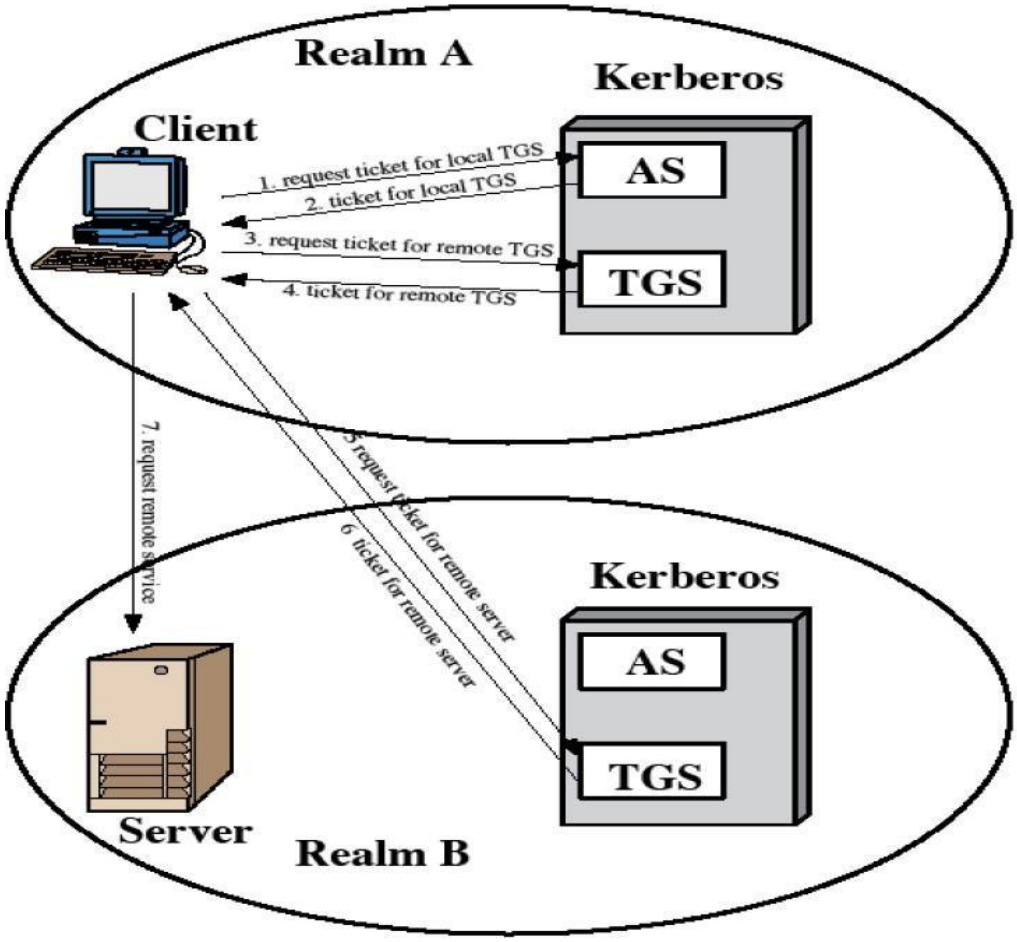


**Fig .Request for service in another Realm**

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 inter realm authentication. For two realms to support inter realm authentication, a third requirement is added:

6. 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.

**The details of the exchanges illustrated in Fig 2are as follows:**

$C \rightarrow AS$ :$ID_C$ II $ID_{tgs}$ II $TS_1$

$AS \rightarrow C$ :$EK_c[K_{c,tgs}$ ii $ID_{tgs}$ II $TS_2$ II $Lifetime_2$ II $Ticket_{tgs}$

$C \rightarrow TGS$ :$ID_{tgsrem}$ II $Ticket_{tgs}$ II $Authenticator_c$

$TGS \rightarrow C$ :$E\ K_{c,tgs}[K_{c,tgsrem}$ II $ID_{tgsrem}$ II $TS_4$ II $Ticket_{tgsrem}$

$C \rightarrow TGS_{rem}$ :$ID_{vrem}$ II $Ticket_{tgsrem}$ II $Authenticator_c$

$TGS_{rem} \rightarrow C$ :$EK_{c,tgsrem}$ [ $K_{c,vrem}$ II $ID_{vrem}$ II $TS_6$ II $Ticket_{vrem}$:

$C \rightarrow V_{rem}$ :$Ticket_{vrem}$ II$Authenticator_c$

**Differences between Versions 4 and 5**

Version 5 is intended to address the limitations of version 4 in two areas: environmental shortcomings and technical deficiencies.

**Environmental shortcomings**:

7. **Encryption system dependence:**

Version 4 requires the use of DES. In version 5, ciphertext is tagged with an encryption type identifier so that any encryption technique may be used.

8. **Internet protocol dependence:**

Version 4 requires the use of Internet Protocol (IP) addresses. Version 5 network addresses are tagged with type and length, allowing any network address type to be used.

9. **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 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.

10. **Ticket lifetime:**

Lifetime values in version 4 are encoded in an 8-bit quantity in units of five minutes. In version 5, tickets include an explicit start time and end time, allowing tickets with arbitrary lifetimes.

11. **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. Version 5 provides this capability.

**Technical deficiencies in the version 4 protocol:**

- Double encryption
- PCBC encryption
- Session keys
- Password attacks

**The Version 5 Authentication Dialogue**

| **(a) Authentication Service Exchange: to obtain ticket-granting ticket** |
|---|
| (1) $C \rightarrow AS$ : Options II $ID_c$ II $Realm_c$ II Times II $Nonce_1$ |
| (2) $AS \rightarrow C$ : $Realm_c$ II $ID_c$ II $Ticket_{tgs}$ II $EK_c$ [ $K_{c,tgs}$ II Times II $Nonce_1$ II $Realm_{tgs}$ II $ID_{tgs}$] |
| $Ticket_{tgs} = EK_{tgs}$ [Flags II $K_{c,tgs}$ II $Realm_c$ II $ID_c$ II $AD_c$ II Times] |
| **(b) Ticket – Granting Service Exchange:to obtain service-granting ticket** |
| (3) $C \rightarrow TGS$: Optionns II $ID_v$ II Times II $Nonce_1$ |
| (4) $TGS \rightarrow C$ : $Realm_c$ II $ID_c$ II $Ticket_v$ II $EK_{c,tgs}[K_{c,v}$ II Times II $Nonce_2$ II $Realm_v$ II $ID_v]$ |
| $Ticket_{tgs} = EK_{tgs}$[Flags II $K_{c,tgs}$ II $Realm_c$ II $ID_c$ II $AD_c$ II Times] |
| $Ticket_v = Ek_v$[[Flags II $K_{c,v}$ II $Realm_c$ II $ID_c$ II $AD_c$ II Times] |

$Authenticator_c = EK_{c,tgs}[ID_c \text{ II } Realm_c \text{ II } TS_1]$

**(c) Client/Server AUTHENTICATION Exchange: to obtain service**

(5) $C \rightarrow V$ : $Options \text{ II } Ticket_v \text{ II } Authenticator_c$
(6) $V \rightarrow C$ : $EK_{c,v}$ [ $TS_2$ II subkey II Seq #]
$Ticketv = EK_v[Flags \text{ II } K_{c,v} \text{ II } Realm_c \text{ II } ID_c \text{ II } AD_c \text{ II Times}]$
$Authenticator_c = E_{Kc,v}[ID_c \text{ II } Realm_c \text{ II } TS_2 \text{ II Subkey II Seq\#}]$

First, consider the authentication service exchange. Message (1) is a client request for a ticket-granting ticket. 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
    - $r_{time}$ : 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 replaced 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.

Let us now compare the ticket-granting service exchange for versions 4 and 5.

We see that message (3) for both versions include 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.

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: