# Centralized Authentication – Kerberos

# Outline

- Introduction to Centralised authentication
- NTLM
- Needham-Schroeder protocol
- Kerberos v4
- Kerberos v5

# Distributed Client-Server System

- There is a distributed client-server architecture.
  - Users are using machines in an open, distributed environment.
  - They need to be able to access services on servers in different locations.
- Servers should only serve authenticated & authorised users.

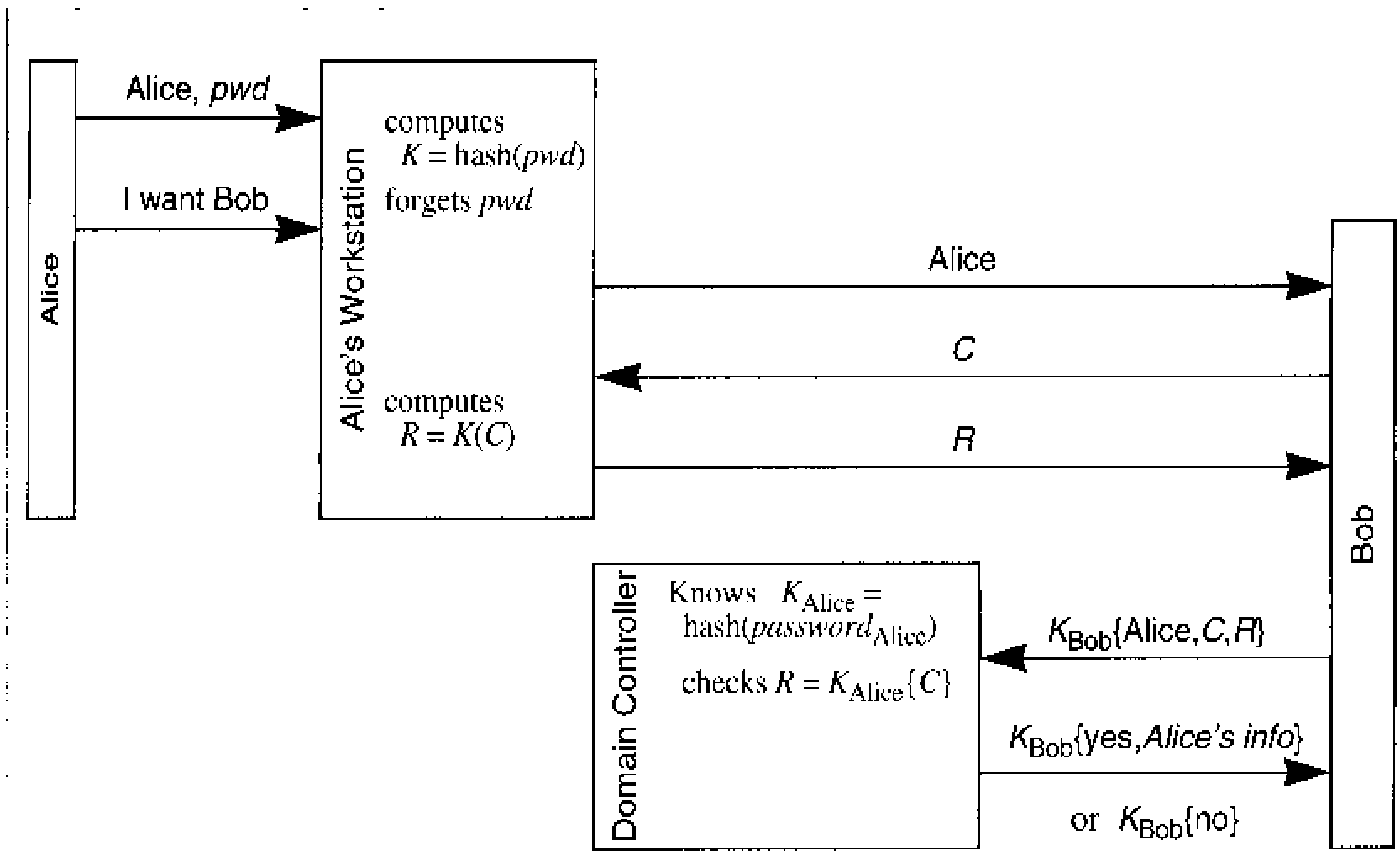
# Centralised Approach

- There is a centralised authentication server (AS) who manages all the long-term user credentials
- The centralised AS assists other servers to authenticate the clients and establish session keys

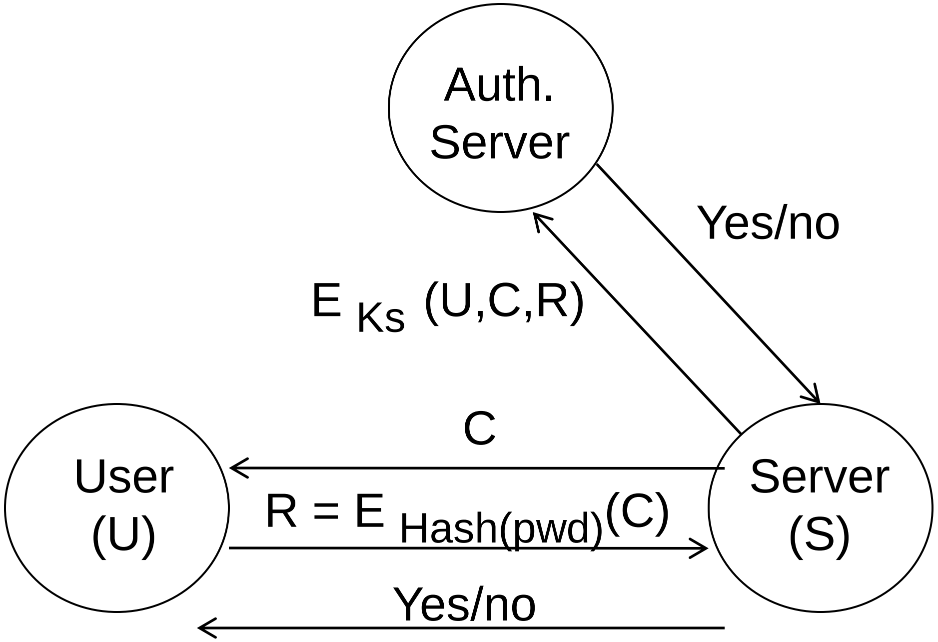
# Centralised Approach

- Windows security mechanisms:
  - – NTLM: used in Windows NT
  - – Kerberos: used since Windows 2000

# NTLM


Alice, pwd
I want Bob
Alice
Alice's Workstation
computes
K = hash(pwd)
forgets pwd
computes
R = K(C)
Alice
C
R
Bob
Domain Controller
Knows K_Alice =
hash(password_Alice)
checks R = K_Alice{C}
K_Bob{Alice, C, R}
K_Bob{yes, Alice's info}
or K_Bob{no}

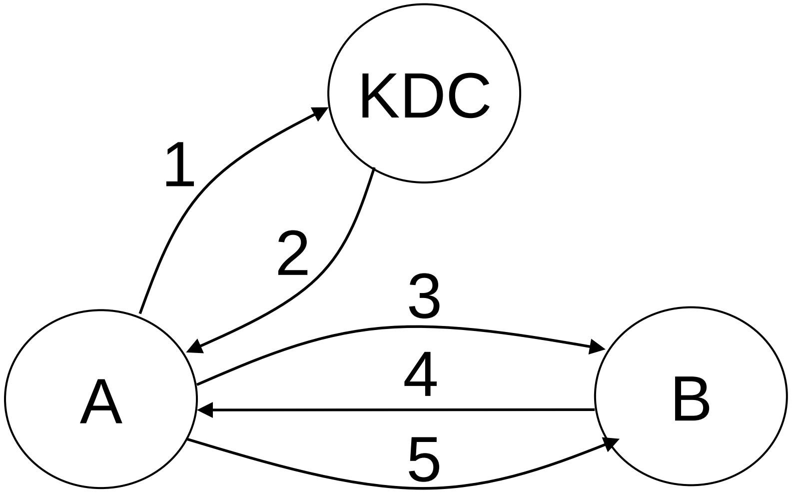
# NTLM




- Auth. Server has: hashed pwds
- Server sends a challenge – nonce C
- User sends a response R – encrypted C with hashed pwd
- Encrypt R is forwarded to Auth. Server (with S's key shared with AS)

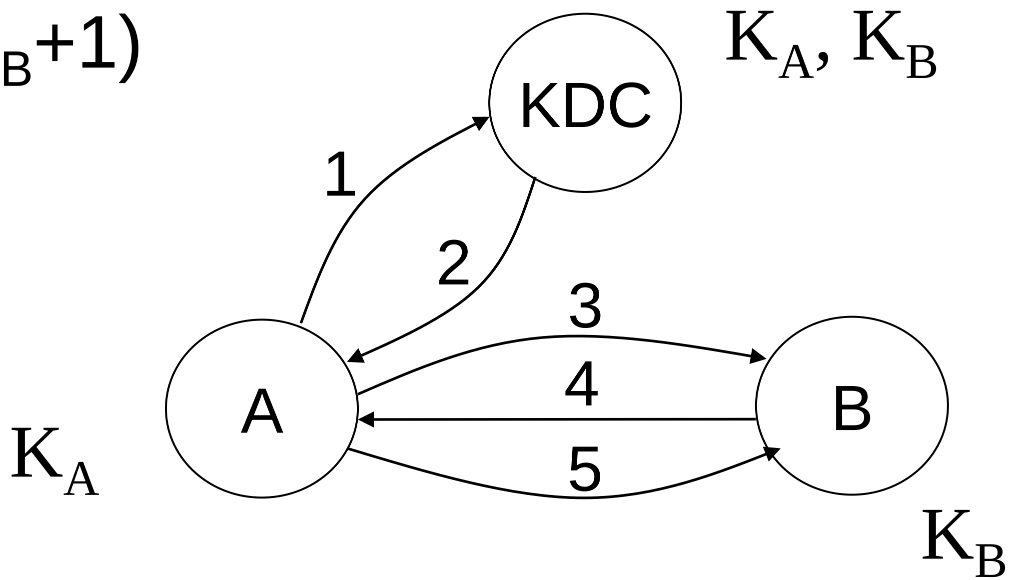
# Needham-Schroeder protocol

1. Alice wants to talk to Bob and contacts the KDC.
2. KDC issues a session key and a ticket that contains the same session key.
3. Alice sends the ticket to Bob (who then has the session key).
4. Bob acknowledges receiving the session key.
5. Alice responds.



# Needham-Schroeder protocol

1. A → KDC: A, B, $N_A$
2. KDC → A: $E_{K_A}$( $N_A$, B, $K_{AB}$, $E_{K_B}(K_{AB}$, A))
3. A → B: $E_{K_B}(K_{AB}$, A)
4. B → A: $E_{K_{AB}}(N_B)$
5. A → B: $E_{K_{AB}}(N_B+1)$




$N_X$: Nonce

# Needham-Schroeder protocol

- The NS protocol is vulnerable to a **replay attack,** in which an attacker *C* can impersonate *A* to cheat *B* by using **a compromised old session key *K***.

  3'. C(A) → B: $E_{K_B}(K_{AB}, A)$

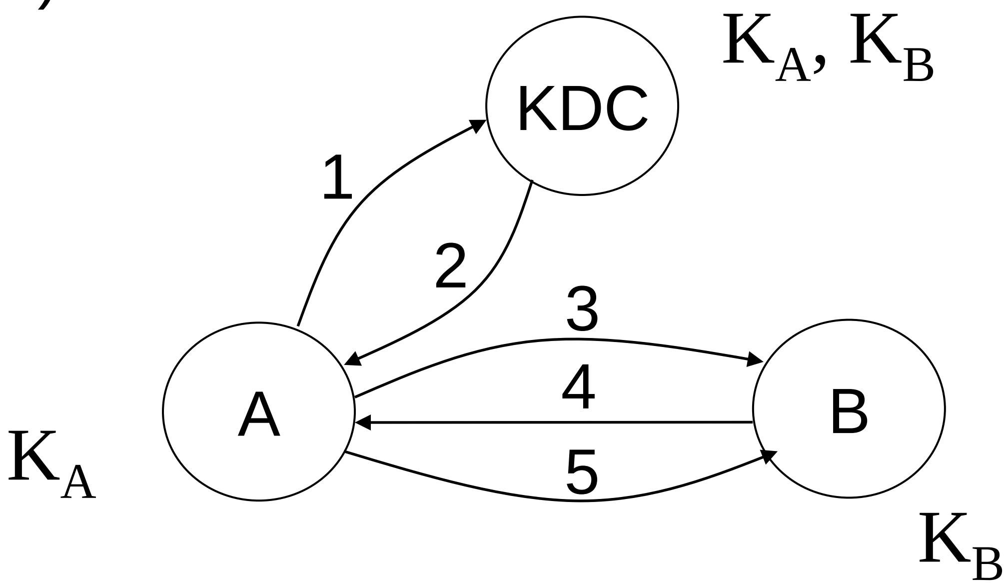
  4'. B → (A)C: $E_{K_{AB}}(N'_B)$

  5'. C(A) → B: $E_{K_{AB}}(N'_B+1)$

- **Repairing**: Insert a timestamp *T* into the key certificate for Bob.

# Modified Needham-Schroeder protocol

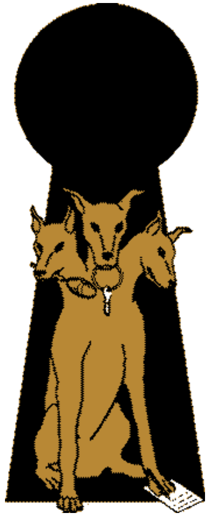
1. A → KDC: A, B, $N_A$
2. KDC → A: $E_{K_A}(N_A, B, K_{AB}, T, E_{K_B}(K_{AB}, A, T))$
3. A → B: $E_{K_B}(K_{AB}, A, T)$
4. B → A: $E_{K_{AB}}(N_B)$
5. A → B: $E_{K_{AB}}(N_B+1)$



# Modified Needham-Schroeder protocol

- Suppress-replay attacks: Attacks against time synchronisation.
  - A sender’s clock is ahead of the intended recipient’s clock
  - An attacker intercepts a message from a sender and replays it later.
- Enforce the requirement that parties regularly check their clocks against the KDC’s clock

# Kerberos



- Named after the three headed watchdog that guarded the gates of Hades in Greek mythology.
- It is an authentication service developed at MIT as part of project Athena.
- Scenario:
  - Users are using workstations in an open, distributed environment. They need to be able to access services on servers in different locations. There is a distributed client/server architecture.
  - Servers should only serve authorised users and should be able to authenticate requests.
- Kerberos is an example of an *Authentication and authorisation infrastructure (AAI)*.

# Kerberos

Kerberos has three kinds of servers:

- **Kerberos authentication server (AS):**

  A centralized trusted authentication server that issues long lifetime tickets for the whole system.

- **Ticket-granting servers (TGS) :**

  Issue short lifetime tickets.

- **Service servers (S) :**

  Provide different services.

# Kerberos Architecture

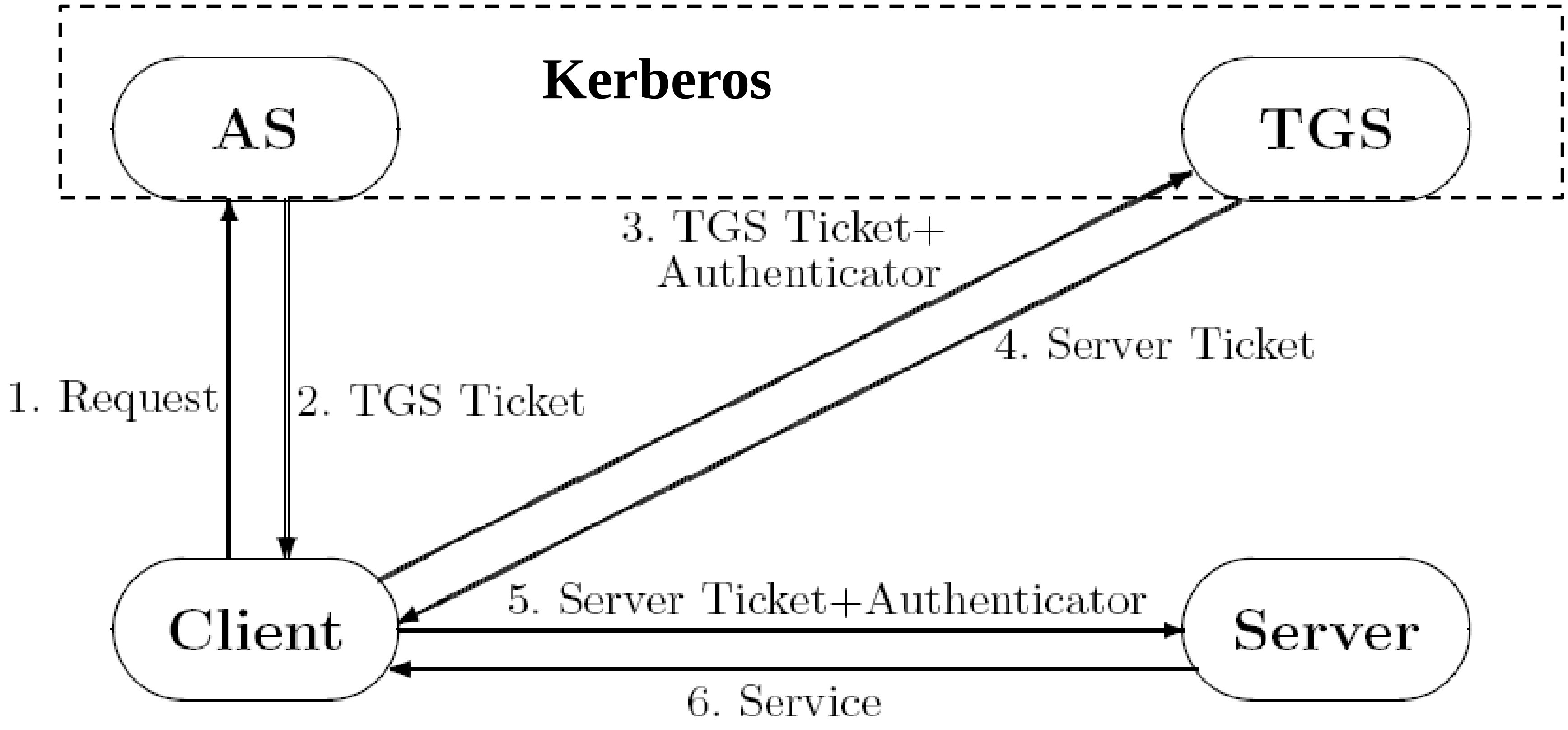



**Figure 1.** The Framework of Kerberos

# Kerberos Operation Overview

- Once per user logon session:
  - (1) $C \rightarrow AS: ID_C, ID_{tgs}$
  - (2) $AS \rightarrow C: E(K_C, K_{c,tgs}), Ticket_{tgs}$
- Once per type of service:
  - (3) $C \rightarrow TGS: ID_C, ID_v, Ticket_{tgs}, Auth_{c,tgs}$
  - (4) $TGS \rightarrow C: E(K_{c,tgs}, K_{c,v}), Ticket_v$
- Once per service session:
  - (5) $C \rightarrow V: ID_C, Ticket_v, Auth_{c,v}$

# Kerberos Protocol V4

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

2: $AS \rightarrow C$: $E_{K_c}[K_{c,tgs}, ID_{tgs}, TS_2, Lifetime_2, Ticket_{tgs}]$

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

3: $C \rightarrow TGS$: $ID_V, Ticket_{tgs}, Authenticator_C$

$Authenticator_C = E_{K_{c,tgs}}[ID_C, AD_C, TS_3]$

4: $TGS \rightarrow C$: $E_{K_{c,tgs}}[K_{C,V}, ID_V, TS_4, Lifetime_4, Ticket_V]$

$Ticket_V = E_{K_v}[K_{c,v}, ID_C, AD_C, ID_V, TS_4, Lifetime_4]$

5: $C \rightarrow V$: $Ticket_V, Authenticator_C$

$Authenticator_C = E_{K_{c,v}}[ID_C, AD_C, TS_5]$

6: $V \rightarrow C$: $E_{K_{c,v}}[TS_5 + 1]$

# Step 1: Client requests…

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

- Once the user is authenticated to the Client (C), the Client sends the authentication server a request on the behalf of the user:
  - This request includes a time-stamp ($TS_1$) and two identities:
    - $ID_C$ - to inform AS of the user
    - $ID_{tgs}$ - to inform AS of the Ticket Granting Service required.
  - There may be multiple TGS's.

# Step 2: AS responds…

$AS \rightarrow C$: $E_{K_c}[K_{c,tgs}, ID_{tgs}, TS_2, Lifetime_2, Ticket_{tgs}]$

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

- A session key, $K_{c,tgs}$ , is generated for secure communication with the ticket granting server indicated by $ID_{tgs}$.
  - A time-stamp ($TS_2$) is specified, as is a lifetime ($Lifetime_2$) for the ticket.
  - $Ticket_{tgs}$ – This is for access to TGS: It includes:
    - The same session key, identity, time-stamp and lifetime.
    - $ID_C$ indicating the user.
    - $AD_C$ indicated the address of the client/user.

# Step 3: Ticket Granting request

$C \rightarrow TGS: ID_V, Ticket_{tgs}, Authenticator_C$

$Authenticator_C = E_{K_{c,tgs}}[ID_C, AD_C, TS_3]$

- The client now has a ticket to communicate with a ticket granting service, and in this step it communicates with the TGS to request a Server ticket.
  - $ID_V$ indicates the relevant server.
  - $Ticket_{tgs}$ is the client's permission to access the TGS.
  - $Authenticator_C$
    - Only C and TGS can open it.
    - It is used by TGS to authenticate C.
    - Contains $ID_C$, $AD_C$, $TS_3$.

# Step 4: Ticket granting response

$$TGS \rightarrow C: \ E_{K_{c,tgs}}[K_{C,V}, ID_V, TS_4, Lifetime_4, Ticket_V]$$

$$Ticket_V = E_{K_V}[K_{c,v}, ID_C, AD_C, ID_V, TS_4, Lifetime_4]$$

The TGS returns a ticket to C, granting access to server/service V.

- The message is encrypted:
  - Provides confidentiality and authentication.
- A key, $K_{C,V}$ , for C to talk to V.
- $ID_V$ is the identity of the server
- There is a new time-stamp ($TS_4$) and a lifetime for the new ticket.

# Step 5: Client request (of server)

$C \rightarrow V$: $\text{Ticket}_V$, $\text{Authenticator}_C$

$\text{Authenticator}_C = E_{K_{c,v}}[ID_C, AD_C, TS_5]$

- The client now communicates with V for access.
  - $\text{Ticket}_V$
  - $\text{Authenticator}_C$ - Only C and V can open it
    - Used by V to authenticate C.
    - Contains $ID_C$, $AD_C$, $TS_5$.

## Step 6: Server response (to client)

$$V \rightarrow C: \ E_{K_{c,v}}[TS_5 + 1]$$

- In this step the server acknowledges the message from the client.

# Inter-realm

- A **realm** is a Kerberos server, set of clients and a set of application servers, such that:
  - The Kerberos server has the user ID's and hashed passwords of all participating users. All users are registered with the Kerberos server.
  - The Kerberos server shares a secret key with each server, each of which is registered with the Kerberos server.

# Inter-realm

- To authenticate across realms we need another property.
  – Each Kerberos server shares a secret key with the Kerberos servers in other realms.
- Some changes are needed in the protocol, and some extra steps are needed too.
  – The ticket requests now reference a service in a remote realm.

# Inter-realm

3: $C \rightarrow TGS: ID_{TGS_r}, Ticket_{TGS}, Authenticator_C$

$Authenticator_C = E_{K_{c,tgs}}[ID_C, AD_C, TS_3]$

4: $TGS \rightarrow C: \ E_{K_{c,tgs}}[K_{C,TGS_r}, ID_{TGSr}, TS_4, Lifetime_4, Ticket_{TGS_r}]$

$Ticket_{TGS_r} = E_{K_{TGS,TGSr}}[K_{c,TGS_r}, ID_C, AD_C, ID_{TGS_r}, TS_4, Lifetime_4]$

5: $C \rightarrow TGS_r: ID_{Vr}, Ticket_{TGSr}, Auth_C$

$Auth_C = E_{K_{c,tgsr}}[ID_C, AD_C, TS_5]$

6: $TGS_r \rightarrow C: \ E_{K_{c,tgsr}}[K_{C,Vr}, ID_{Vr}, TS_6, Ticket_{Vr}]$

$Ticket_{Vr} = E_{K_{vr}}[K_{c,v}, ID_C, AD_C, ID_{Vr}, TS_6, Lifetime_6]$

7: $C \rightarrow V_r: \ Ticket_{Vr}, Auth_C$

$Auth_C = E_{K_{c,vr}}[ID_C, AD_C, TS_7]$

# Kerberos V4 Limitations

- **Encryption:** V4 uses DES only. V5 allows any encryption method.
- **Restricted ticket lifetime:** V4 uses an 8 bit lifetime, for a maximum of about 21 hours. V5 allows the specification of start and end times.
- **Authentication forwarding:** V4 does not allow credentials issued to one client to be forwarded to another host. Consider the following example of when this might be desirable: 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.
- **Double encryption** of the tickets in steps two and four. This is unnecessary and inefficient.
- **Offline dictionary attack:** The message from the authentication server to the client (step 2) can be captured. A **password attack** against it can be launched where success occurs if the decrypted result is of an appropriate form.

# Kerberos V5

1: $C \rightarrow AS$: Options, $ID_C$, $R_C$, $ID_{tgs}$, Times, $N_1$

2: $AS \rightarrow C$: $R_C$, $ID_C$, $Ticket_{tgs}$, $E_{Kc}[K_{c,tgs}$, Times, $N_1$, $R_{tgs}$, $ID_{tgs}]$

$Ticket_{tgs} = E_{K_{tgs}}[$Flags, $K_{c,tgs}$, $R_C$, $ID_C$, $AD_C$, Times$]$

3: $C \rightarrow TGS$: Options, $ID_V$, Times, $N_2$, $Ticket_{tgs}$, $Auth_C$

$Auth_C = E_{K_{c,tgs}}[ID_C, R_C, TS_1]$

4: $TGS \rightarrow C$: $R_C$, $ID_C$, $Ticket_V$, $E_{K_{c,tgs}}[K_{C,V}$, Times, $N_2$, $R_V$, $ID_V]$

$Ticket_V = E_{K_V}[$Flags, $K_{c,v}$, $R_C$, $ID_C$, $AD_C$, Times$]$

5: $C \rightarrow V$: Options, $Ticket_V$, $Auth_C$

$Auth_C = E_{K_{c,v}}[ID_C$, $R_C$, $TS_2$, Subkey, Seq#$]$

6: $V \rightarrow C$: $E_{K_{c,v}}[TS_2$, Subkey, Seq#$]$

# Kerberos V5

- **N** is for Nonce
- **R** is for Realm
- **Options** provides a request for certain **flags** (indicating properties) to be set in the returned ticket, e.g.,
  - PRE-AUTHENT: AS authenticates the client before issuing a ticket
  - HW-AUTHENT: Hardware based initial authentication is employed
  - RENEWABLE: A ticket with this flag set includes two expiration times
    - One for this specific ticket
    - One for the latest permissible expiration time

    A client can have a ticket renewed, if the ticket is not reported stolen
  - FORWARDABLE: A new ticket-granting ticket with a different network address may be issued based on this ticket.
  - ……

# Kerberos V5

- **Times** are requested by the client for ticket configuration.
  - *from*: a start-time.
  - *till*: the requested expiration time.
  - *rtime*: requested renew-till time, i.e. allow continued use until.
- **Subkey** is an optional sub-encryption key used to protect a specific session of an application. The default is the session key.
- The **Sequence number (Seq#)** is an optional sequence start number to be used by the server. It is used to protect the system from replay attacks.