A first overview on Kerberos

What is Kerberos?

- Kerberos is an authentication protocol that provides authentication in distributed systems
  - Guarantees safe access to network resources (e.g., printer, databases etc.)
  - There is a central authority that allows to reduce the number of passwords that users must memorize
- Reference
  - proposed by MIT http://web.mit.edu/kerberos/www/dialogue.html
  - free download in US and Canada (after 2000, in most locations)
  - widespread use (most operating systems)
- Key Features
  - Mutual authentication
  - Single Sign-On
  - Ticket-based
  - Symmetric cryptography
  - RFC 4120

Only version 4 covered

- Serves as a historical reference, introducing the core architecture and design philosophy of Kerberos

- Later versions (e.g., v5) introduce protocol refinements, additional features, and enhanced security measures, while preserving the original conceptual model

Kerberos scenario

- A needs to access service provided by B
    - Authentication of A
    - Optional: authentication of B
    - Optional: decide session keys for secret communication and/or authentication
- C is trusted server (authority that shares keys with A and B)
- Idea: use ticket to access services; tickets are valid in each time window

More on Kerberos

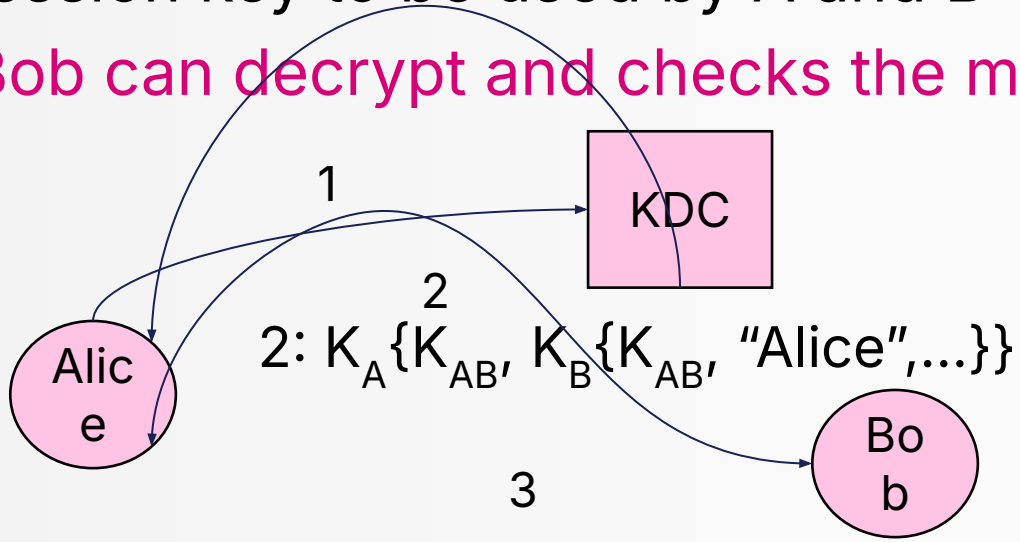
- KDC (Key Distribution Center) is the server (both trusted and physically safe)
- Messages are safe with respect to cryptographic attacks and data integrity
- Kerberos provides security for applications like
  - telnet
  - rtools (rlogin, rcp, rsh)
  - network file systems (NFS/AFS)
  - e-mail
  - printer servers
  - ...

Preliminaries

- Each user (also named as principal) has a master secret key with KDC
- For human users master secret key is derived from password
- For system resources, keys are defined while configuring the application
- Each principal is registered by the KDC
- All master keys are stored in the KDC database, encrypted with the KDC master key

Tickets, Alice, Bob, KDC

- Ticket of Alice for Bob: $K_B\{K_{AB}, \text{“Alice”}, \ldots\}$
- K_A master key of Alice, K_B master key of Bob
- K_{AB} session key to be used by A and B
- only Bob can decrypt and checks the message



Tickets

- a ticket is encrypted with
  secret key associated wit
  service
- ticket basically contains
  - sessionkey
  - username
  - client network addre
  - servicename
  - lifetime
  - timestamp

Kerberos: simplified version

A asks for a ticket TicketB for B

1. A sends to C: (A, B, N) *(N nonce)*
2. C sends to A: (TicketB, $K_{AC}(K_{AB}, N, L, B)$)
   L = "lifetime of ticket"
3. [A checks N and knows ticket lifetime]
   A sends to B: (TicketB, $K_{AB}(A,t_A)$) [ticket+authenticator]
4. [B checks that A's identity in TicketB and in authenticator are the same, time validity of ticket]
   B sends to A: $K_{AB}(t_A)$
   [in this way shows knowledge of t_A]

TicketB = $K_{BC}(K_{AB}, A, \text{"lifetime"}, \text{timestamp})$, N nonce, K_{AB} session key; "lifetime"= validity of ticket; t_A timestamp

Session key and credentials

- Messages between host and KDC should be protected using the master key (derived from user's password)
- For each request to the KDC
  - user must type the password each time, or
  - user's password is temporarily stored (to avoid the user the need of retyping)
- **All above solutions are inadequate!**

Session key and Ticket-Granting Ticket (TGT)

Proposed solution to reduce # of times user types the password and/or master key

At initial login a session key S_A is derived for Alice by KDC

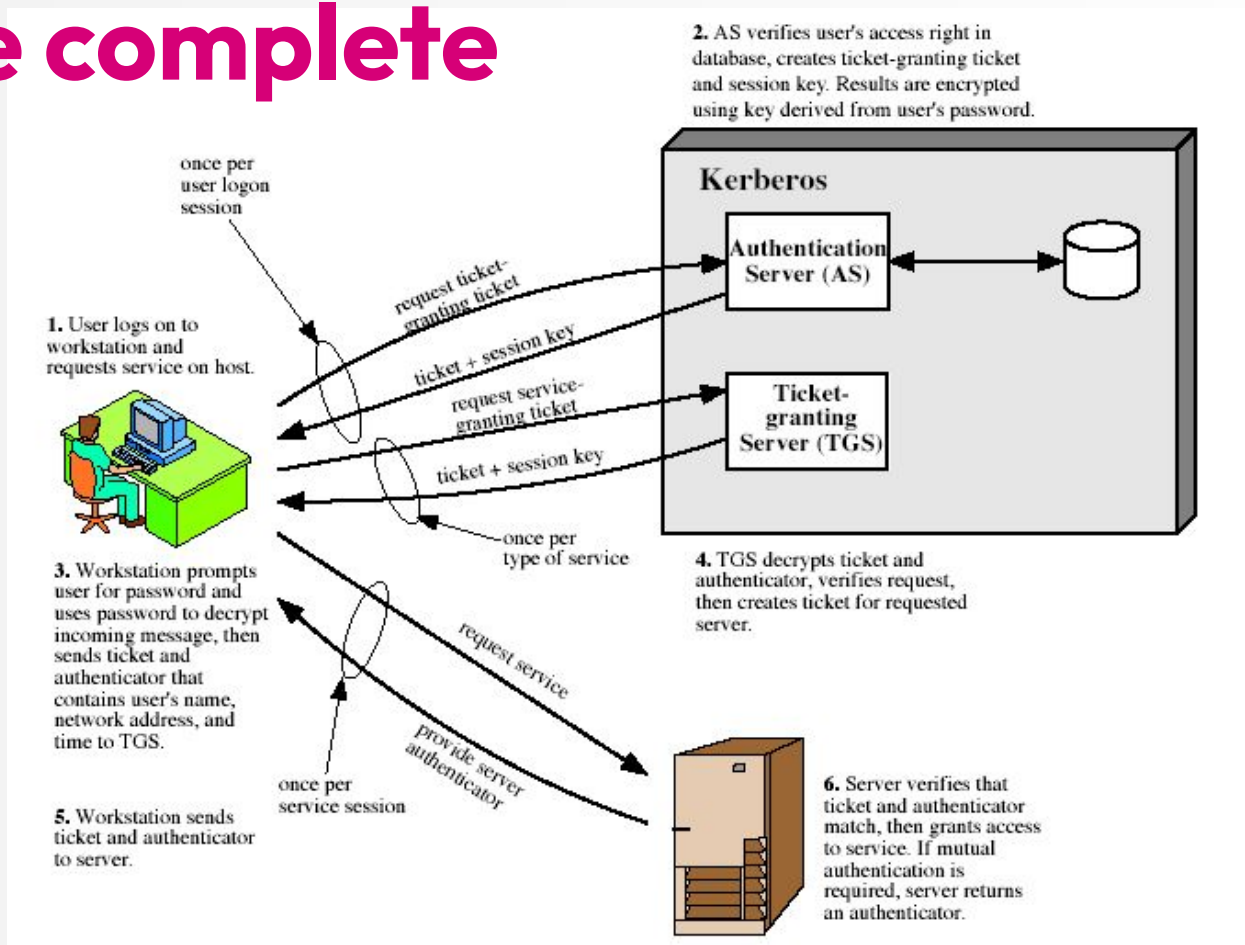
S_A has a fixed lifetime (e.g., 1 day, 4 hours)

KDC gives Alice a TGT that includes session key S_A and other useful information to identify Alice (encrypted with KDC's master key)

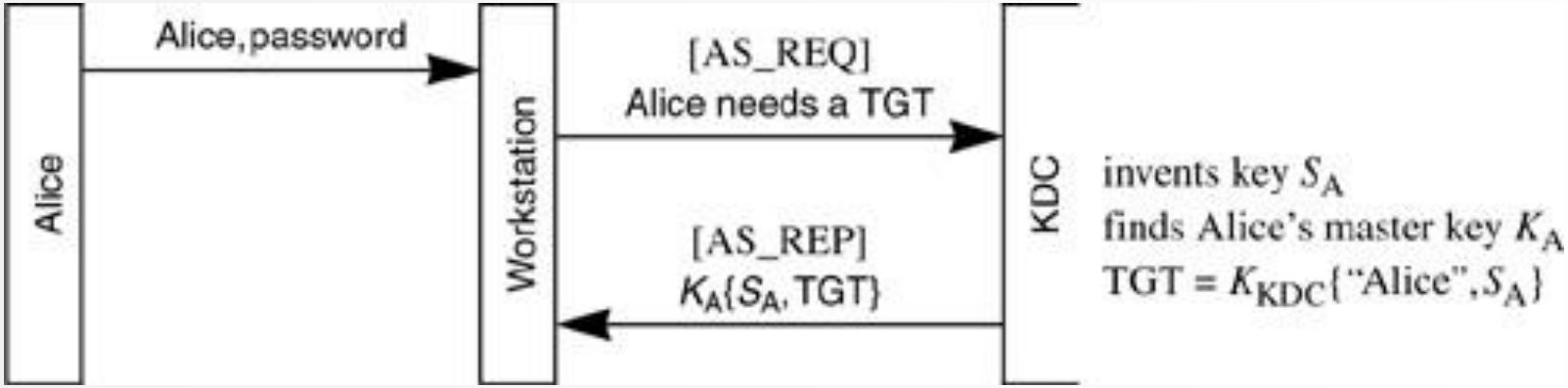
TGT

- Subsequent requests from Alice to KDC use TGT in the initial message
- Subsequent tickets provided by KDC for accessing server V are decrypted using K_{VC}
- User provides password only once
- No password is stored

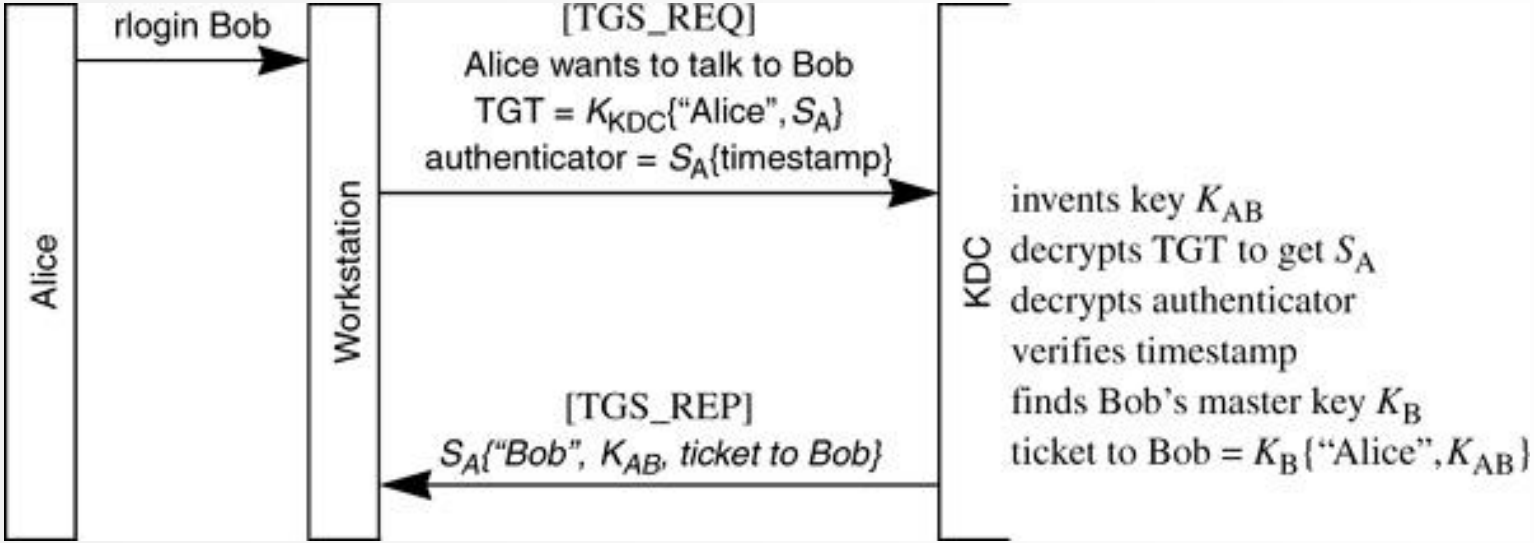
More complete



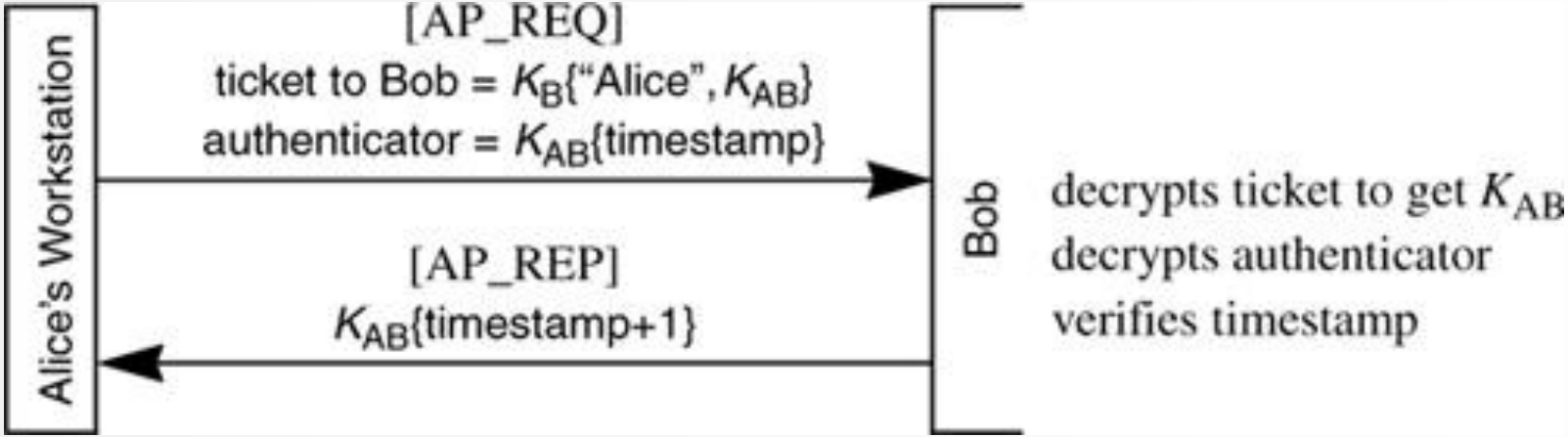
Login


Alice
Alice, password
Workstation
[AS_REQ]
Alice needs a TGT
[AS_REP]
$K_A\{S_A, TGT\}$
KDC
invents key S_A
finds Alice's master key K_A
TGT = K_{KDC}{"Alice", S_A}

Ticket request



Use of ticket



Authentication and synchronization

- Authenticator: K_X{timestamp} – K_X is a session key
- Global synchronous clock is required
- Authenticator is used to avoid
  - replay of old messages sent to the same server by the adversary (old messages are eliminated)
  - Original authenticator DOES NOT guarantee data integrity (a MAC is required)
- Vulnerability: many instances of same server all using same master key. Replay attack!
  - how could it be avoided? Synchronized replay caches

KDC and TGS

- KDC and TGS are similar (the same?). Why do we need two different entity?
  - Historical reasons
  - One KDC can serve different systems (1 KDC, many TGS)
- Multiple copies of KDC, sharing same KDC master key - availability and performance
- Consistency issues in KDC databases
  - A single KDC stores information concerning principal (safer)
  - Periodically upload information to other KDC

Kerberos – performance

- KDC stores only
  - Master key of the KDC (used to encrypt the KDC's principal database)
  - Principal database (list of all user and service identities)
  - Long-term keys for each principal (user keys, service keys)
- Most work is on client
- KDC is involved only at login to provide TGT
- KDC uses only permanent information

Main message types

Message	Direction	Purpose
AS_REQ	Client → AS	Request a TGT
AS_REP	AS → Client	Reply with TGT and session key
TGS_REQ	Client → TGS	Request a service ticket
TGS_REP	TGS → Client	Reply with service ticket and client–service session key
AP_REQ	Client → Application Server	Present service ticket and authenticator
AP_REP	Application Server → Client	Optional reply proving the server's identity

Ticket (Alice, Bob; v4)

Encrypted with Bob's key

- Alice's name, instance and realm
- Alice's Network Layer address
- Session key for Alice, Bob
- Ticket lifetime, units of 5 minutes
- KDC's timestamp when ticket made
- Bob's name and instance
- Pad of 0s to make ticket length multiple of eight octets

Authenticator (v4)

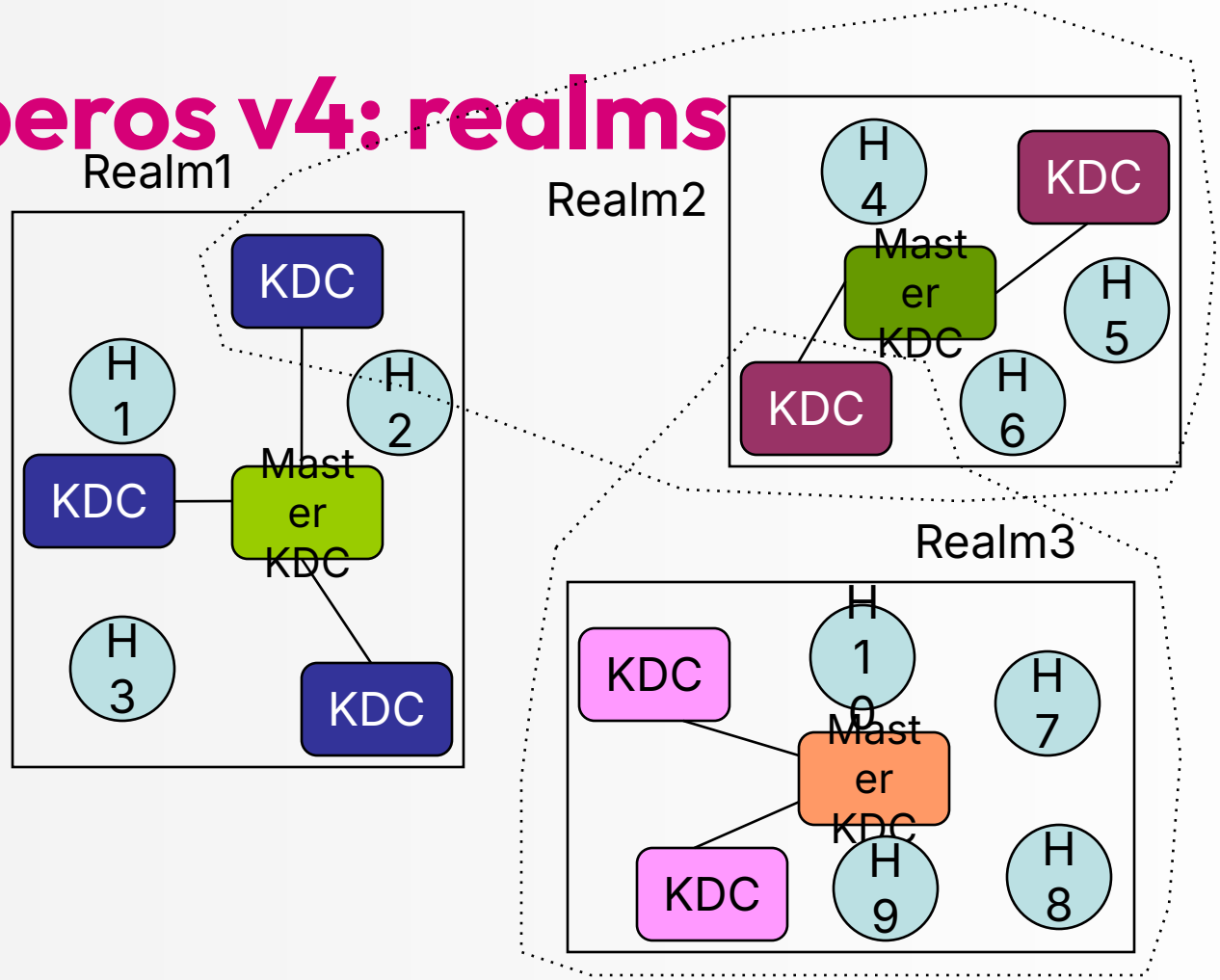
Encrypted with session key

- Alice's name, instance and realm
- Alice's Network Layer address
- checksum
- 5-millisecond timestamp (fine-grained component to uniquely identify authenticators created within the same second)
- Timestamp (time in seconds from epoch, coarse granularity for freshness checks)
- Pad of 0s to make authenticator length multiple of eight octets

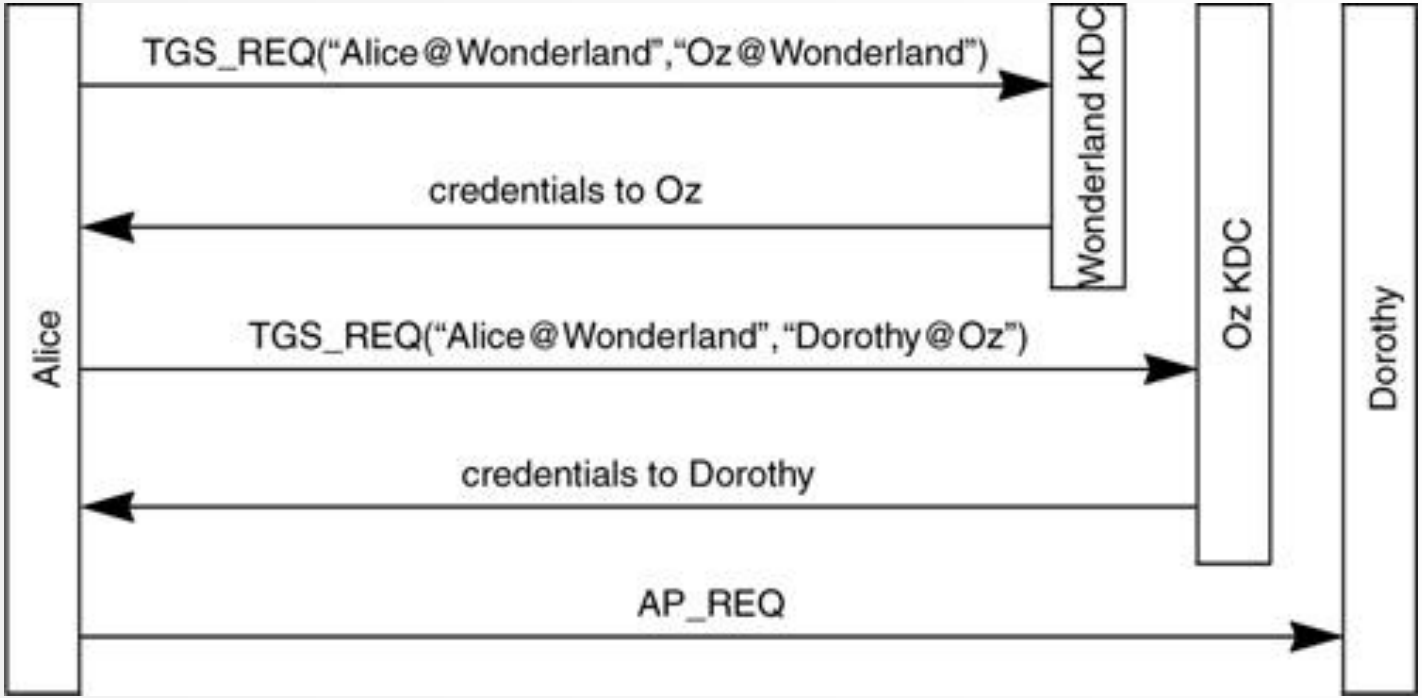
Kerberos: realms

- In very large systems, security and performance considerations suggest using more than one domain—often multiple KDCs
- A **realm** is the administrative boundary in Kerberos
- Each realm has its own master KDC
- All KDCs within the same realm share the same KDC master key
- KDCs in different realms maintain separate user databases
- KDC may store a shared key with other cross-realm principals for trusted KDCs in other realms (used for inter-realm authentication)

Kerberos v4: realms



Authentication between realms



Kerberos: overview on version 5

- Same philosophy, but significant changes
- Integrity of messages, authentication using nonce (not only timestamps)
- Flexible encoding: many optional fields,
    - allows future extensions
    - overhead
- Major extensions to the functionality
- Delegation of rights: Alice allows Bob to access:
    - her resources for a specified amount of time
    - a specific subset of her resources
- Renewable tickets: tickets can be used for long time
- More encryption methods (Kerberos designed for DES)
- Hierarchy of realms