Authentication II

Computer and Network Security

Emilio Coppa

Authentication protocols based on a shared secret

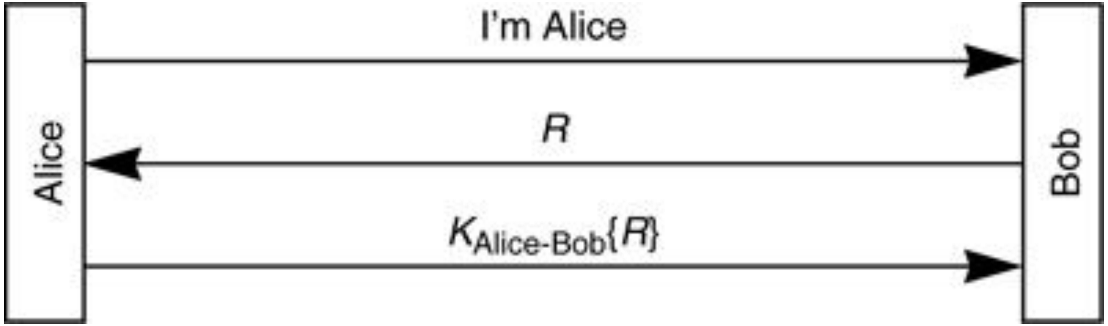
How to perform authentication?

We can design different protocols. To make them stronger we can use different techniques:

- timestamps: prevent attackers from reusing old messages; require clock synchronization among entities

- nonce (or challenge): random numbers, never re-used (Number used only ONCE), used to "challenge" someone to encrypt/decrypt something. In some protocols, nonce could be not random but a squence number.

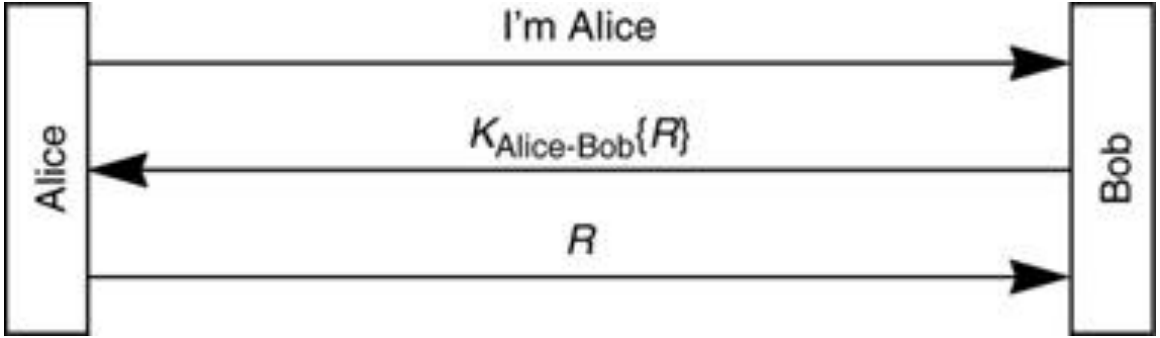
Challenge/response based on shared secret



- R is a nonce with limited lifetime
- function to encrypt it can be an hash function
- Bob authenticates Alice, but not the opposite, hence it is not mutual.
- if K_{AB} is derived from a password, then an attacker can still guess the password
- offline password guessing: a valid pair is known to the attacker
- if Bob (e.g., the server) is compromised then Alice is in trouble
- … still an improvement w.r.t. sending password in clear (or hash of password)

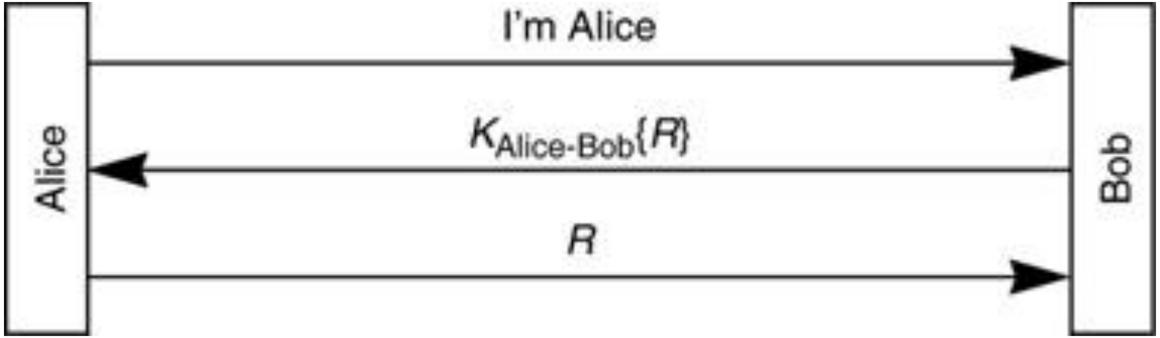
Problem: attacker may impersonate Bob and challenge Alice with R

Challenge/response based on shared secret (2)



- similar to previous protocol
- requires that encryption is reversible (decryption)

Challenge/response based on shared secret (3)



Bob shows to Alice that he knows the right key. Hence, Alice could verify that the message is generated by Bob (she can decrypt correctly R) but she is not challenging Bob: she cannot be sure that the other is not just reusing old messages since she is not proposing a new fresh challenge.

Nonce: how to avoid reuse?

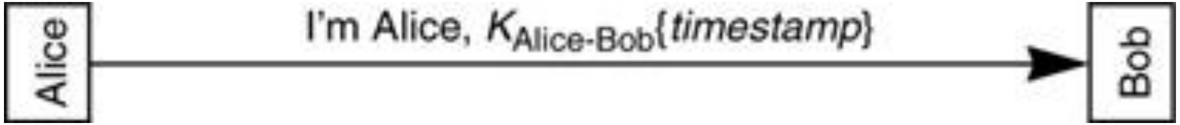
Ideally, to avoid reuse of nonces, an entity A should keep track of:

1. nonces chosen by A to avoid to “propose” the same challenge twice to B
2. nonces chosen by B to avoid to “respond” the same challenge twice to B

Problem: (1) and (2) are not practical: too many values to remember. Hence, an entity will:

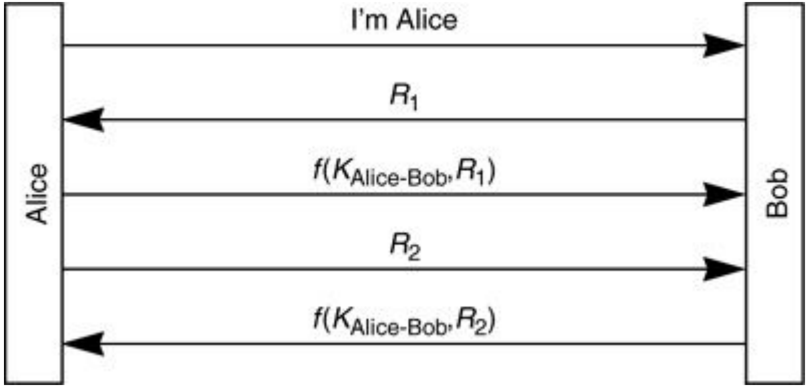
- choose nonces randomly: very unlikely to pick the same number twice, hence achieving (1) without the need of storing past nonces
- not detect if a nonce was already used in the past by B: this is the reason why for getting mutual authentication we need that both entities generate a “fresh” challenge. Even if an attacker reuse an old challenge from Bob, Alice can detect that is not talking to Bob since the attacker is not able to solve the new “fresh” challenge within a limited amount of time

Use of a timestamp to limit replay attacks



- Bob and Alice have reasonably synchronized clocks (can be a weakness: we need a protocol to set securely the correct time)
- Alice encrypts the current time, Bob decrypts the result and makes sure the result is acceptable (i.e., within an acceptable clock skew)
- efficient, no intermediate states
- if Bob remembers timestamps until they expire, then no replaying attacks
- if multiple servers with same secret K, since timestamp in not tied to Bob then Alice could send K{Bob|timestamp} to prevent problems
- no mutual authentication

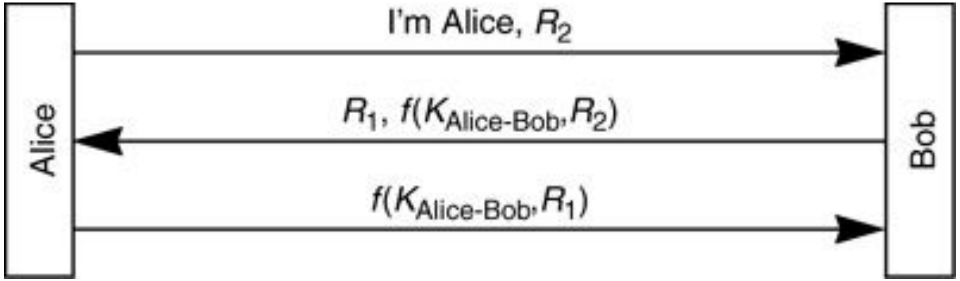
Mutual authentication



I'm Alice
R_1
$f(K_{\text{Alice-Bob}}, R_1)$
R_2
$f(K_{\text{Alice-Bob}}, R_2)$
Alice
Bob

- double challenge/response authentication protocol
- number of messages can be reduced
- Alice and Bob should keep track of nonce over time to avoid replay attacks

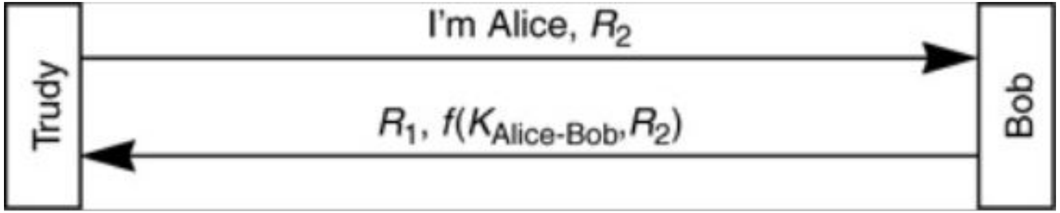
Mutual authentication (optimized)


Alice
Bob
I'm Alice, R_2
R_1, $f(K_{\text{Alice-Bob}}, R_2)$
$f(K_{\text{Alice-Bob}}, R_1)$

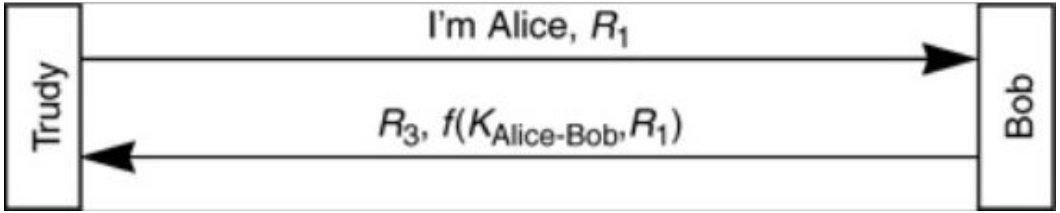
- fewer messages, hence more efficient
- weak to a reflection attack
- weak to offline password guessing on the side of the protocol initiator (where the attacker is more likely to be)

Mutual authentication (optimized): reflection attack

Session #1:



Session #2:



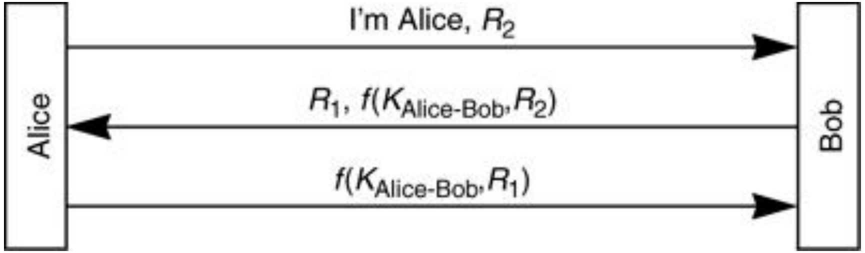
- Trudy wants to impersonate Alice to Bob
- two sessions: the 2nd will be incomplete but will allow Trudy to complete the 1st one

How to prevent reflection attacks

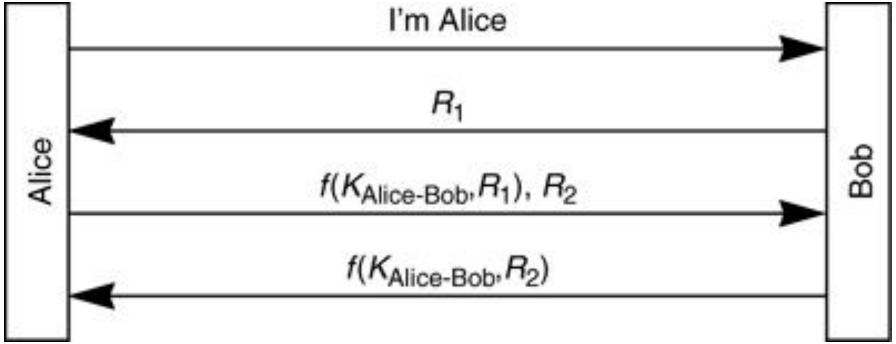
- use different keys: K_{AB} vs $-K_{AB}$ vs or $K_{AB} + 1$

- different challenges, i.e. challenge from initiator different from challenge from responder:
  - e.g., even number at initiator's side, odd number at responder's side

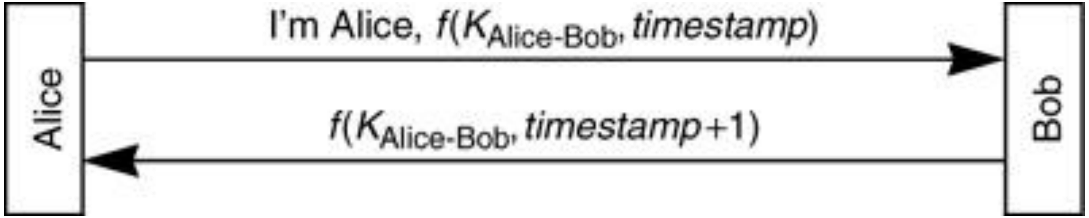
Mutual authentication (optimized): offline password guessing


Alice
Bob
I'm Alice, R_2
R_1, $f(K_{Alice\text{-}Bob}, R_2)$
$f(K_{Alice\text{-}Bob}, R_1)$

Attacker can impersonate Alice, send challenge to Bob, get response and then proceed working offline trying to guess the right password. Notice that inefficient mutual authentication protocol was not affected by this. We can mitigate this by adding a message in the protocol:


Alice
Bob
I'm Alice
R_1
$f(K_{Alice\text{-}Bob}, R_1)$, R_2
$f(K_{Alice\text{-}Bob}, R_2)$

Mutual authentication with timestamps


Alice
Bob
I'm Alice, $f(K_{\text{Alice-Bob}}, \textit{timestamp})$
$f(K_{\text{Alice-Bob}}, \textit{timestamp}+1)$

- crucial to alter the timestamp in the response from Bob
- clock synchronization is crucial

Real-world protocols: ISO/IEC 9798-2

Slightly variants of the previous protocols are defined by ISO/IEC 9798-2:

- one-pass unilateral authentication:

$$\boxed{A \rightarrow B : E_K(ts_A, B)}$$

where ts_A is either a timestamp or a sequence number.

- two-pass unilateral authentication:

$$\boxed{\begin{array}{l} B \rightarrow A : N_B \\ A \rightarrow B : E_K(N_B, B) \end{array}}$$

where N_B is a nonce.

- two-pass mutual authentication:

$$\boxed{\begin{array}{l} A \rightarrow B : E_K(ts_A, B) \\ B \rightarrow A : E_K(ts_B, A) \end{array}}$$

This is just the composition of two unilateral authentications.

- three-pass mutual authentication:

$$\boxed{\begin{array}{l} B \rightarrow A : N_B \\ A \rightarrow B : E_K(N_A, N_B, B) \\ B \rightarrow A : E_K(N_B, N_A) \end{array}}$$

(credits)

Issues and security analysis: [paper] [paper]

Practical problem of sharing a secret with each entity

In practice, in a large network, it is not possible to share a secret pairwise:

Number of Keys: if n users in a network then

- we need $\frac{n \cdot (n-1)}{2}$ keys.
- each user has to store n-1 keys, which is reasonable
- adding one user to the network, we have to transfer securely n keys (hard!)

Practical problem of sharing a secret with each entity (2)

Possible alternative approaches:

- share a secret with a centralized trusted party

- use public-key schemes

Authentication protocols based on a trusted party

Trusted Party

Scenario:

- A and B share a secret key with C (K_{AC} or K_{Alice} and K_{BC} or K_{Bob})
- C is also called authentication server or Key Distribution Center (KDC)
- A and B might not be human user but also entity of the system (e.g., printer, databases etc.)

Goals:

- authenticate A (or A and B)
- [optional] decide on a session key K_{AB} to be used between A and B for short time, where K_{AB} is known only to A and B, randomly chosen and never used in the past communications

Attacker’s capabilities

Trudy (attacker) can:

- be a legitimate user of the system (share a key with C)
- sniff and spoof messages
- concurrently run more than one session with A, B and C
- different execution of the protocol can be done interleaved
- T is able to convince A and/or B to start a new session with T
- might know old session keys

Attacker’s limits

Trudy:

- is NOT able to guess random numbers chosen by A or B
- does not know keys K_{AC} and K_{BC} (in general does not know secret keys of other users)
- is not able to decode in a short time messages encrypted with unknown keys

Authentication with a trusted party

1. A sends to C:

   A, B

2. C chooses K (secret shared key) and sends to A:

   $K_{AC}(K)$ and $K_{BC}(K)$

3. A decodes and computes K and sends to B:

   C, A, $K_{BC}(K)$

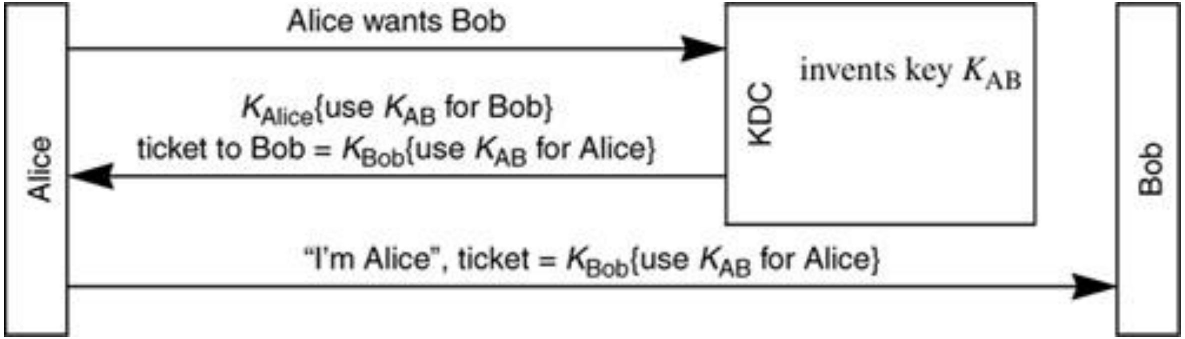
4. B decodes $K_{BC}(K)$ finds K and sends to A:

   K(Hello A, this is B)

In several protocols, the message generated by C for Bob (containing the session key) but sent to Alice is called a "ticket".

Authentication with a trusted party (2)

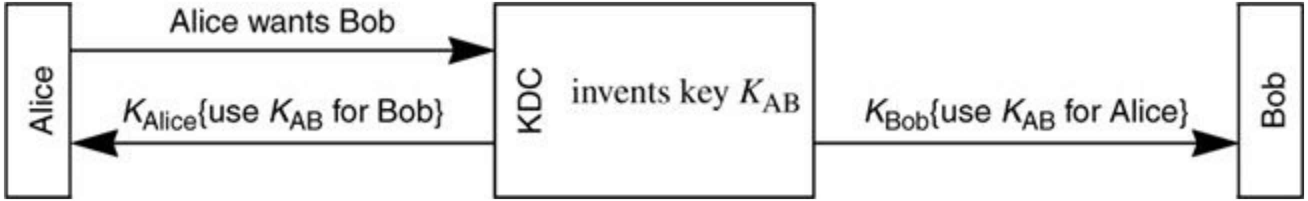
Protocol:


Alice wants Bob
invents key K_{AB}
KDC
K_{Alice}\{use K_{AB} for Bob\}
ticket to Bob = K_{Bob}\{use K_{AB} for Alice\}
Alice
Bob
"I'm Alice", ticket = K_{Bob}\{use K_{AB} for Alice\}

- in several protocols, the message generated by KDC for Bob (containing the session key) but sent to Alice is called a "ticket"
- this protocol is weak against a MITM attack

Authentication with a trusted party

A variant of the previous protocol:


Alice wants Bob
Alice
K_{Alice}\{use K_{AB} for Bob\}
KDC
invents key K_{AB}
K_{Bob}\{use K_{AB} for Alice\}
Bob

- KDC needs to communicate with both Alice and Bob
- Attacker may easily trick KDC to send messages to Bob: DoS? Unlikely, but still a concern
- Problem: This requires synchronization: Alice may send something encrypted to Bob, before Bob has received the shared key

MITM attack: authentication with a trusted party

1. A sends to T (instead of A sends to C) :

   A, B

2. immediately T sends to C:

   A,T

3. C chooses K and sends to A:

   $K_{AC}(K)$ and $K_{TC}(K)$

4. A sends to B:

   C, A, $K_{TC}(K)$

5. T intercepts the message sends to A:

   K(Hello A, this is B)

Problem: Alice believes to have a shared key with Bob, instead the key is with Trudy.

Authentication with a trusted party (revised)

1. A sends to C:

   $A, K_{AC}(B)$

2. C chooses K (secret shared key) and sends to A:

   $K_{AC}(K)$ and $K_{BC}(K)$

3. A decodes and computes K and sends to B:

   $C, A, K_{BC}(K)$

4. B decodes $K_{BC}(K)$ finds K and sends to A:

   K(Hello A, this is B)

Trudy now does not know
that A wants to talk to B.

Is this enough to make the protocol secure?

MITM attack: authentication with a trusted party (revised)

1. A sends to T (instead of A sends to C) :

   A, K_{AC}(B)
2. T intercept the message and replays old message:

   A, K_{AC}(T)
3. C chooses K and sends to A:

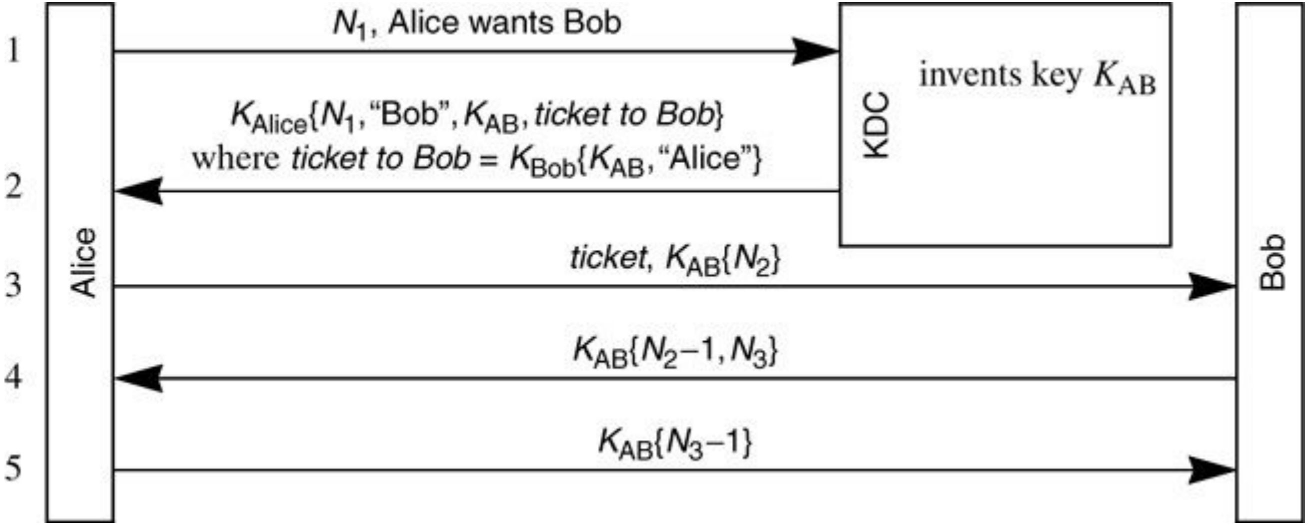
   K_{AC}(K) and K_{TC}(K)
4. A sends to B:

   C, A, K_{TC}(K)
5. T intercepts the message, understand that A is trying to talk to B and sends to A:

   K(Hello A, this is B)

Problem: Alice believes to have a shared key with Bob, instead the key is with Trudy. The only difference now is that Trudy has to wait the end of the attack to know the identity B.

Needham–Schroeder (NS) symmetric protocol



1
2
3
4
5
Alice
KDC
Bob
N_1, Alice wants Bob
invents key K_{AB}
$K_{Alice}\{N_1, \text{“Bob”}, K_{AB}, \textit{ticket to Bob}\}$
where *ticket to Bob* = $K_{Bob}\{K_{AB}, \text{“Alice”}\}$
ticket, $K_{AB}\{N_2\}$
$K_{AB}\{N_2{-}1, N_3\}$
$K_{AB}\{N_3{-}1\}$

Remark. KDC does not communicate with Bob

NS replay attack (Denning and Sacco)

Bob is not communicating with KDC and does not know when the shared key has been generated whether KDC is still active. Suppose that an old session key K'_{AB} has been compromised then:

1. A chooses N (nonce) and sends to C: A, B, N_1
2. C chooses K and sends to A: $K_{AC}(N_1, B, K_{AB}, K_{BC}(K_{AB}, A))$
3. A decodes, checks N_1 and B and sends to B: $K_{BC}(K_{AB}, A). K_{AB}(N_2)$
4. T blocks the message for B, replays (as A) to B: $K_{BC}(K'_{AB}, A), K'_{AB}(N''_2)$
5. B decodes, chooses nonce N_2 and sends to T (not to A): $K'_{AB}(N''_2-1, N_3)$
6. T sends to B: $K'_{AB}(N_3 - 1)$

NS replay attack (2)

Hence, to carry out the attack, two sessions are needed:

- session #1:
  - Trudy records $K_{BC}(K'_{AB}, A)$
  - Trudy (later) is able to recover K'_{AB}

- session #2:
  - Trudy intercepts messages from A to B
  - Trudy reuse old $K_{BC}(K'_{AB}, A)$
  - Bob is convinced to have a new shared key with Alice, which however it is known to Trudy

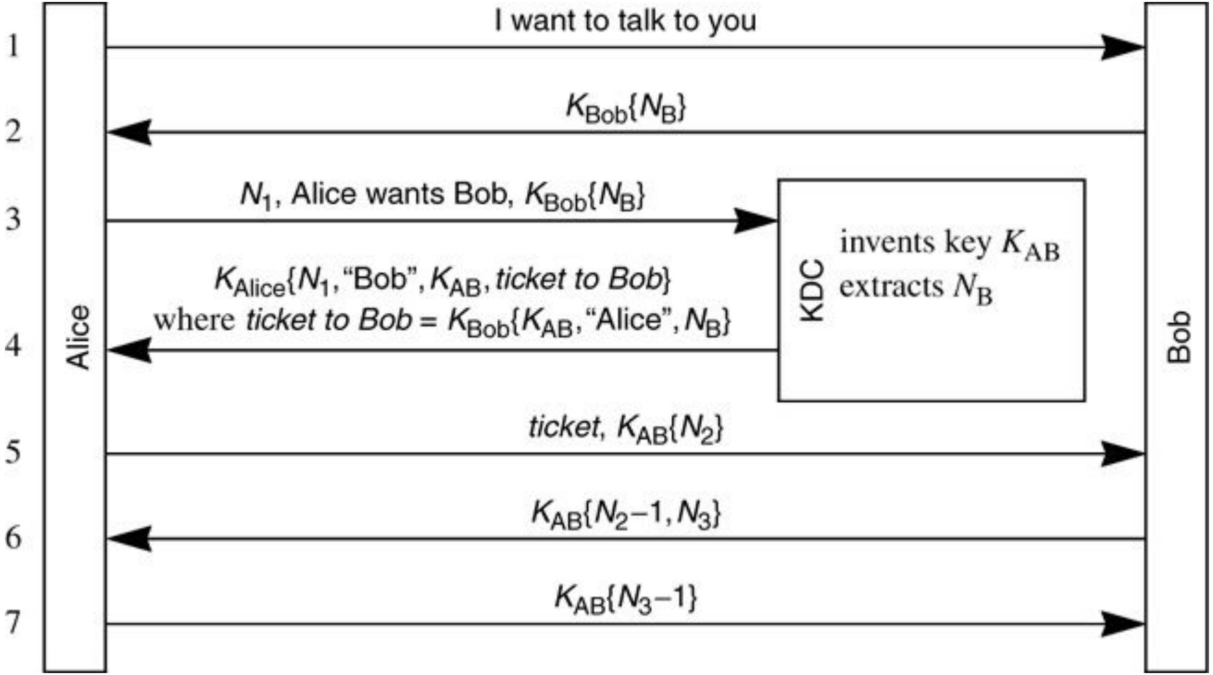
How to prevent replay attacks?

Mitigations to fix replay attacks can be based on:

- timestamps: a message is valid only in a small time window
- sequence numbers: A and B remember sequence number of exchanged messages to avoid replay attacks in which the attacker sends old messages
- nonce: used in some protocols, but not easy mitigation (NS is already using nonce...)

Needham-Schroeder symmetric protocol (fixed)

Mitigation based on a nonce.


Alice
Bob
KDC
1
I want to talk to you
2
$K_{Bob}\{N_B\}$
3
N_1, Alice wants Bob, $K_{Bob}\{N_B\}$
invents key K_{AB}
extracts N_B
4
$K_{Alice}\{N_1, \text{"Bob"}, K_{AB}, \textit{ticket to Bob}\}$
where *ticket to Bob* = $K_{Bob}\{K_{AB}, \text{"Alice"}, N_B\}$
5
ticket, $K_{AB}\{N_2\}$
6
$K_{AB}\{N_2-1, N_3\}$
7
$K_{AB}\{N_3-1\}$

Last message could be safely removed

Needham-Schroeder symmetric protocol (fixed variant)

Kerberos is a well-known protocol inspired by NS where KDC exchanges messages with both A and B and uses a timestamp to prevent replay attacks:

1. A chooses N and sends to B:

   A, N

2. B chooses N’ and sends to C:

   B, N’, K_{BC}(N, A, t)

3. C sends to A:

   K_{AC}(B, N, K_{AB}, t) , ticket = K_{BC}(A, K_{AB}, t), N’

4. A sends to B:

   K_{BC}(A, K_{AB}, t), K_{AB}(N’)

Standard: Kerberos v4/v5

Challenge-response Symmetric Key

What do we learn from previous attacks?

- Timestamps are valid only in a small time window
- Sequence numbers attached to messages are useful (to avoid replication attacks)
- Nonce: we should carefully use them (and we require good random number generators)

Kerberos

- Kerberos 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)

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 a given time window

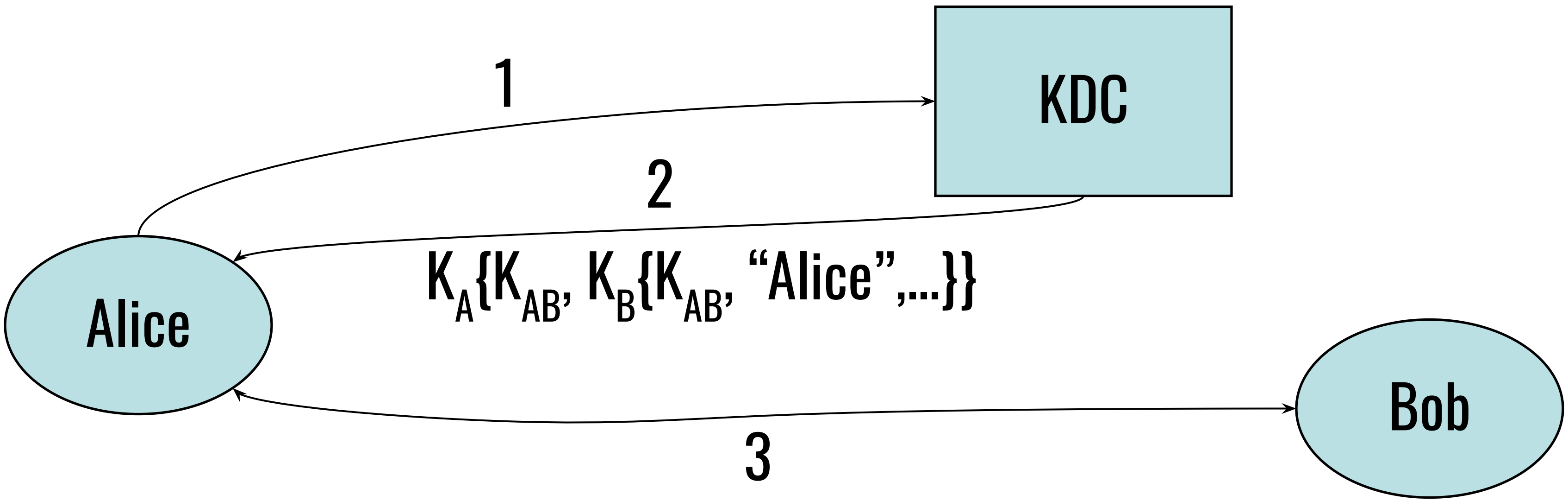
Kerberos

- 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
  - etc.

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 is able to decode and checks the message

Tickets

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

Kerberos: simplified version (some fixes later)

A asks for a ticket TicketB for B:

1. A sends to C: A, B, N (N nonce), t_A (timestamp)

2. C sends to A: TicketB, $K_{AC}(K_{AB}, N, L, B)$
   (where: TicketB = $K_{BC}(K_{AB}, A, L, t_A)$, L is “lifetime of ticket”)

3. [A checks N and knows ticket lifetime]
   A sends to B: TicketB, $K_{AB}(A, t_A)$ [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]

Session key and Ticket-granting Ticket (TGT)

- 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)

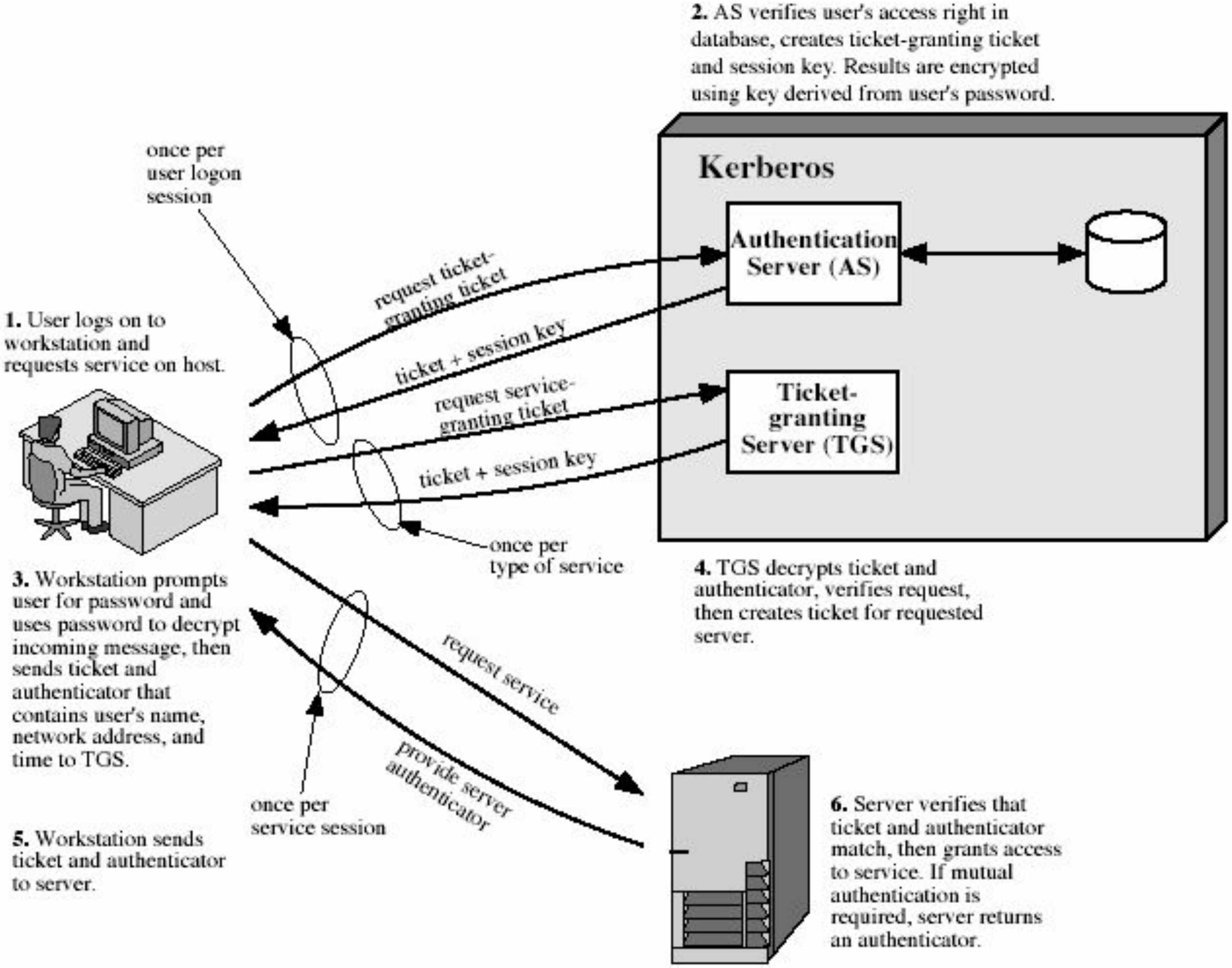
Session key and Ticket-granting Ticket (TGT)

- Subsequent requests from Alice to KDC use TGT in the initial message

- Subsequent tickets provided by KDC for accessing server V are decoded using K_{VC}

- User provides password only once

- No password is stored



Kerberos
Authentication Server (AS)
Ticket-granting Server (TGS)
once per user logon session
request ticket-granting ticket
ticket + session key
request service-granting ticket
ticket + session key
once per type of service
request service
provide server authenticator
once per service session

1. User logs on to workstation and requests service on host.

2. AS verifies user's access right in database, creates ticket-granting ticket and session key. Results are encrypted using key derived from user's password.

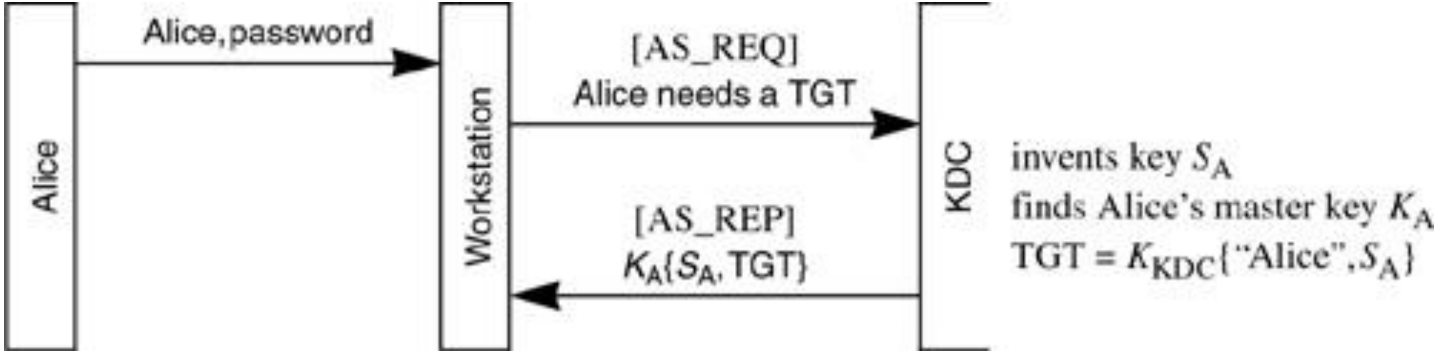
3. Workstation prompts user for password and uses password to decrypt incoming message, then sends ticket and authenticator that contains user's name, network address, and time to TGS.

4. TGS decrypts ticket and authenticator, verifies request, then creates ticket for requested server.

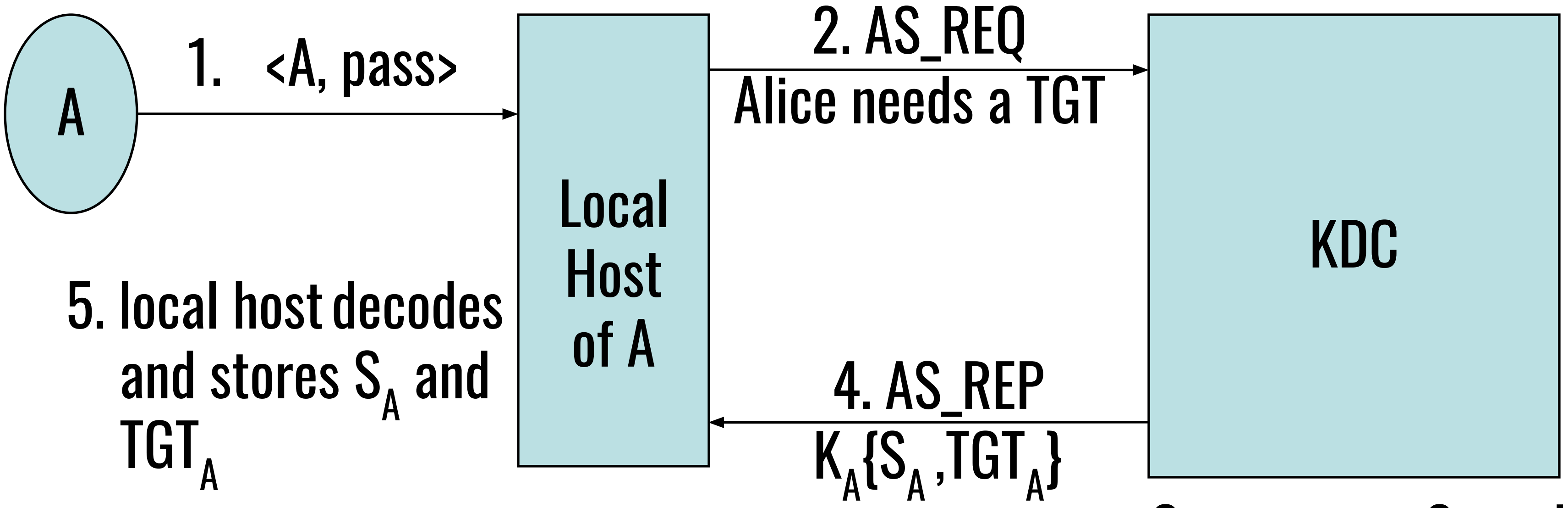
5. Workstation sends ticket and authenticator to server.

6. Server verifies that ticket and authenticator match, then grants access to service. If mutual authentication is required, server returns an authenticator.

Login


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

Login (simplified)



A

1. <A, pass>

Local Host of A

2. AS_REQ Alice needs a TGT

KDC

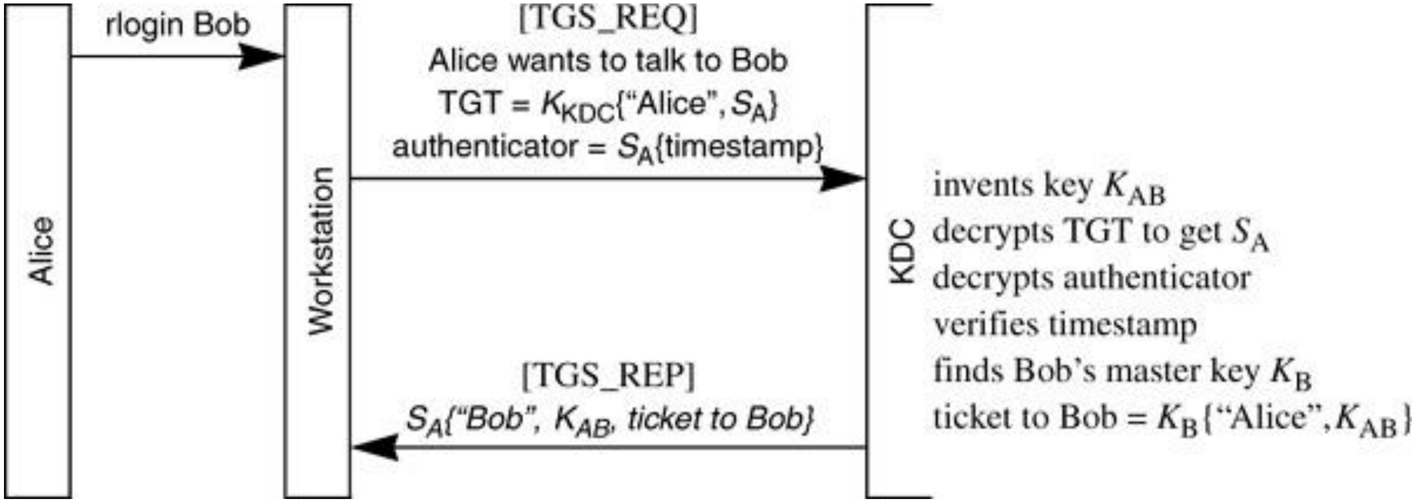
4. AS_REP $K_A\{S_A, TGT_A\}$

5. local host decodes and stores S_A and TGT_A

3. generates S_A and $TGT_A = K_{TGS}\{A, S_A\}$

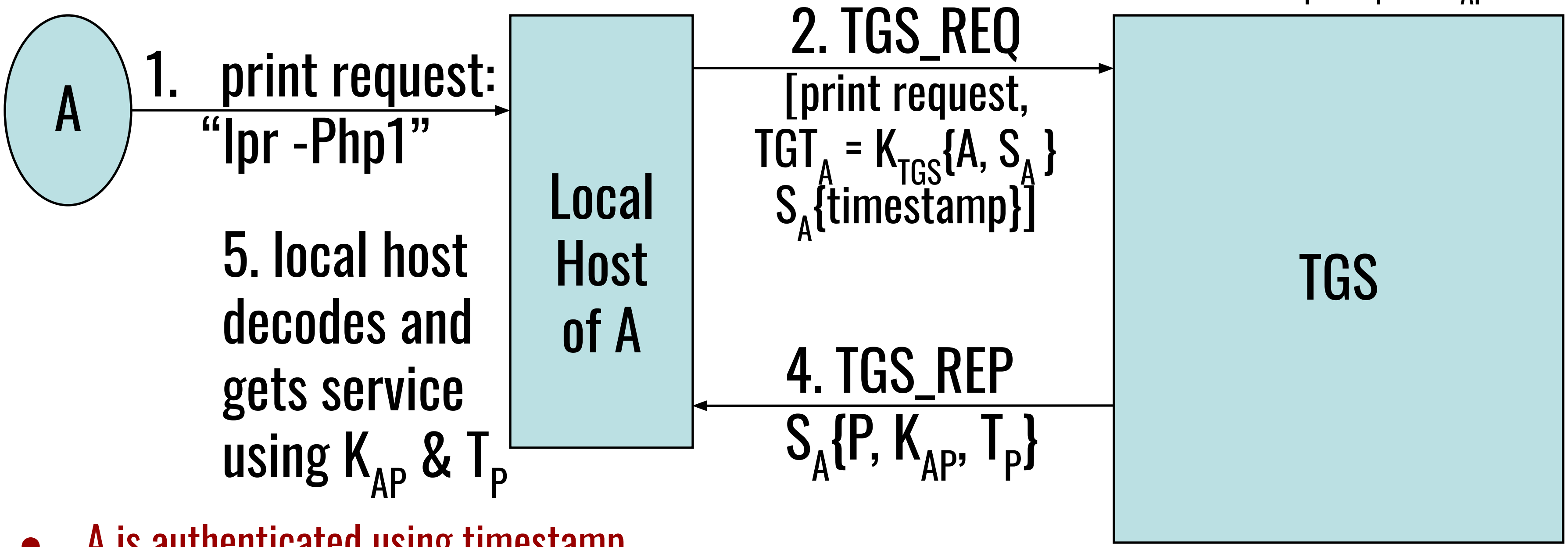
- Local host of A = (current) Alice’s workstation
- S_A = session key for A

Ticket request


Alice
Workstation
KDC
rlogin Bob
[TGS_REQ]
Alice wants to talk to Bob
TGT = K_{KDC}{"Alice", S_A}
authenticator = S_A{timestamp}
invents key K_{AB}
decrypts TGT to get S_A
decrypts authenticator
verifies timestamp
finds Bob's master key K_B
ticket to Bob = K_B{"Alice", K_{AB}}
[TGS_REP]
S_A{"Bob", K_{AB}, ticket to Bob}

Ticket request (simplified)

3. generates K_{AP}, decodes TGT_A
Checks S_A\{timestamp\}
authenticator generates Ticket
for printer: $T_P = K_P\{A, K_{AP}\}$



1. print request: “lpr -Php1”

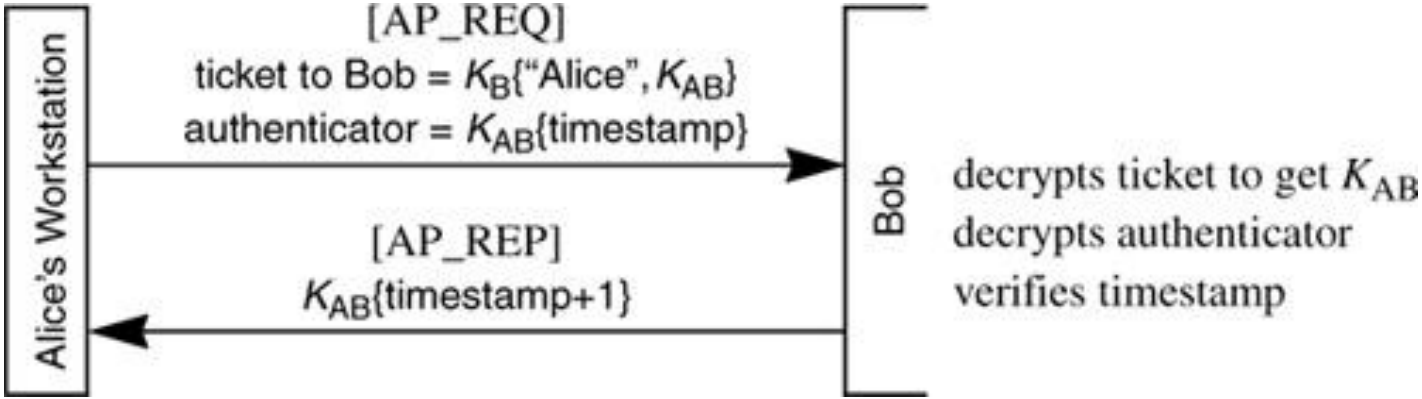
2. TGS_REQ
[print request,
$TGT_A = K_{TGS}\{A, S_A\}$
S_A\{timestamp\}]

4. TGS_REP
$S_A\{P, K_{AP}, T_P\}$

5. local host decodes and gets service using K_{AP} & T_P

- A is authenticated using timestamp
- P = printer

Use of ticket



Alice's Workstation

[AP_REQ]

ticket to Bob = K_B{"Alice", K_{AB}}

authenticator = K_{AB}\{timestamp\}

Bob

decrypts ticket to get K_{AB}

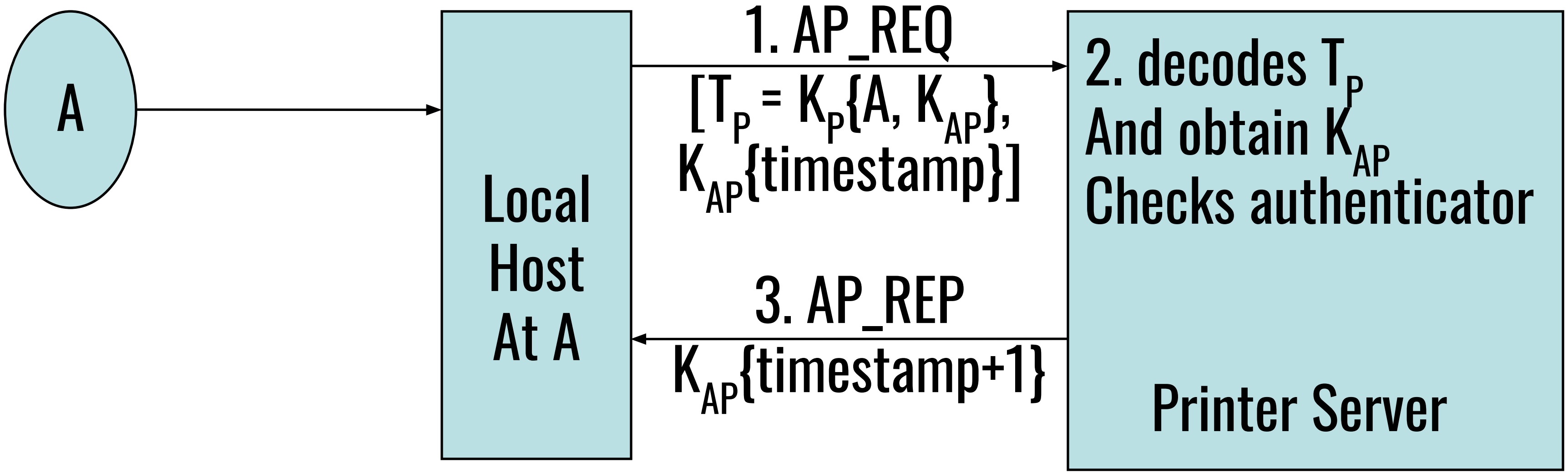
decrypts authenticator

verifies timestamp

[AP_REP]

K_{AB}\{timestamp+1\}

Use of Ticket for printer P



A

Local Host At A

1. AP_REQ

$[T_P = K_P\{A, K_{AP}\}$, $K_{AP}\{\text{timestamp}\}]$

2. decodes T_P
And obtain K_{AP}
Checks authenticator

3. AP_REP

$K_{AP}\{\text{timestamp+1}\}$

Printer Server

- printer request is managed by A’s local host
- there is mutual authentication using timestamp

Authentication and time 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)
  - replay to a server (when there are many servers)
  - Authenticator DOES NOT guarantee data integrity (a MAC is required)
- Vulnerability: many instances of same server all using same master key. Replay attack!

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 (master-slaves)
  - A single KDC stores information concerning principal (safer)
  - Periodically upload information to other KDC

Kerberos - Performance

- KDC stores only TGT and tickets

- Most work is on client

- KDC is involved only at login to provide TGT

- KDC uses only permanent information

Message types

- AS_REQ
  - Used when asking for the initial TGT.
- AS_REPLY (also TGS_REP)
  - Used to return a ticket, either a TGT or a ticket to some other principal.
- AP_REQ (also TGS_REQ)
  - Used to talk to another principal (or the TGS) using a ticket (or a TGT).
- AP_REQ_MUTUAL
  - This was intended to be used to talk to another principal and request mutual authentication. In fact, it is never used; instead, applications know whether mutual authentication is expected.
- AS_ERR
  - Used for the KDC to report why it can't return a ticket or TGT in response to AS_REQ or TGS_REQ.
- PRIV
  - This is a message that carries encrypted integrity-protected application data.
- SAFE
  - This is a message that carries integrity-protected application data.
- AP_ERR
  - Used by an application to report why authentication failed.

Ticket (Alice, Bob)

It is 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

It is encrypted with session key

- Alice's name, instance and realm
- checksum
- 5-millisecond timestamp
- timestamp (time in seconds)
- pad of 0s to make authenticator multiple of eight octets

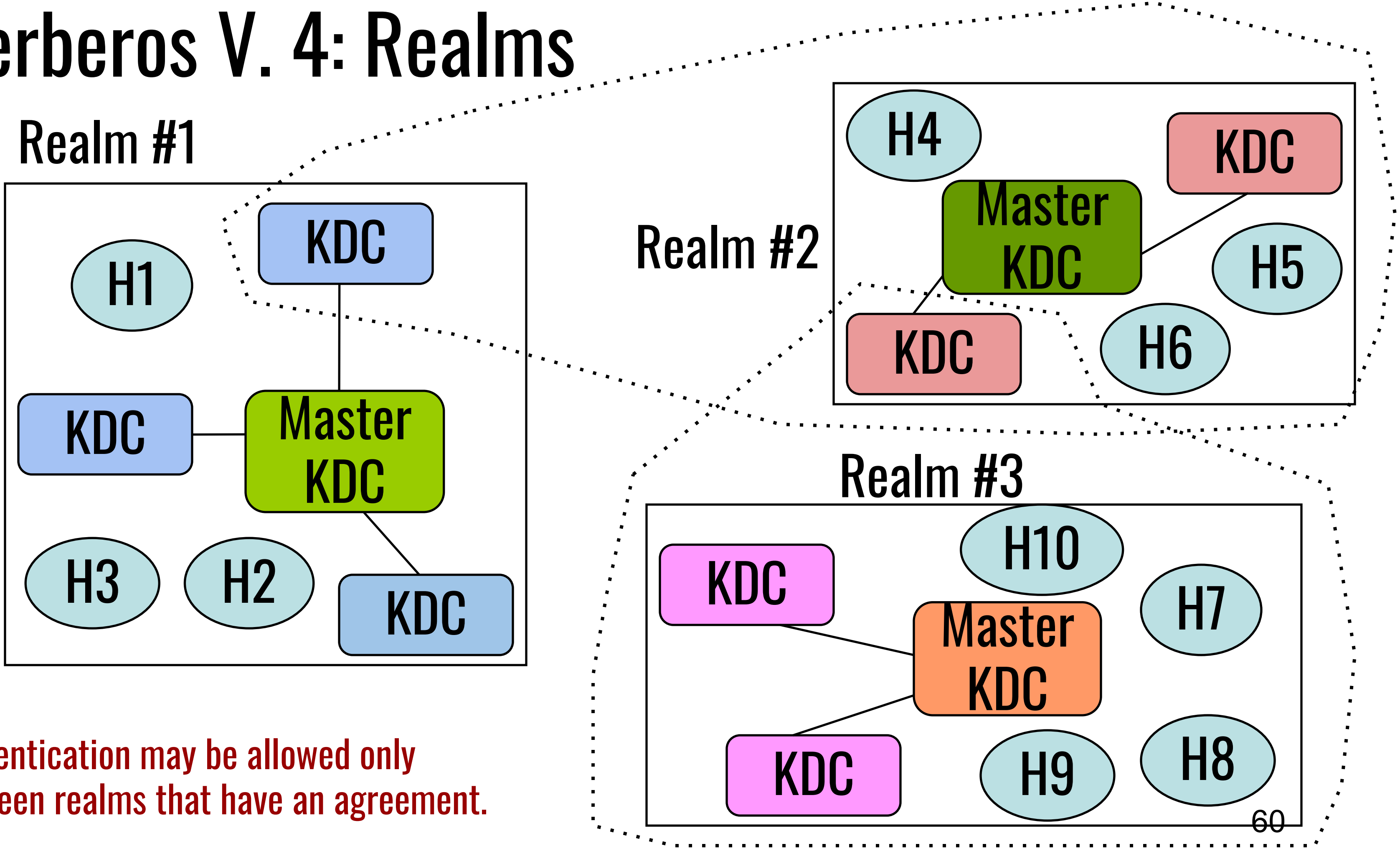
Kerberos: Realms

- In very large systems security and performance issues suggest to use not only a domain but more (i.e., several KDC):

REALM

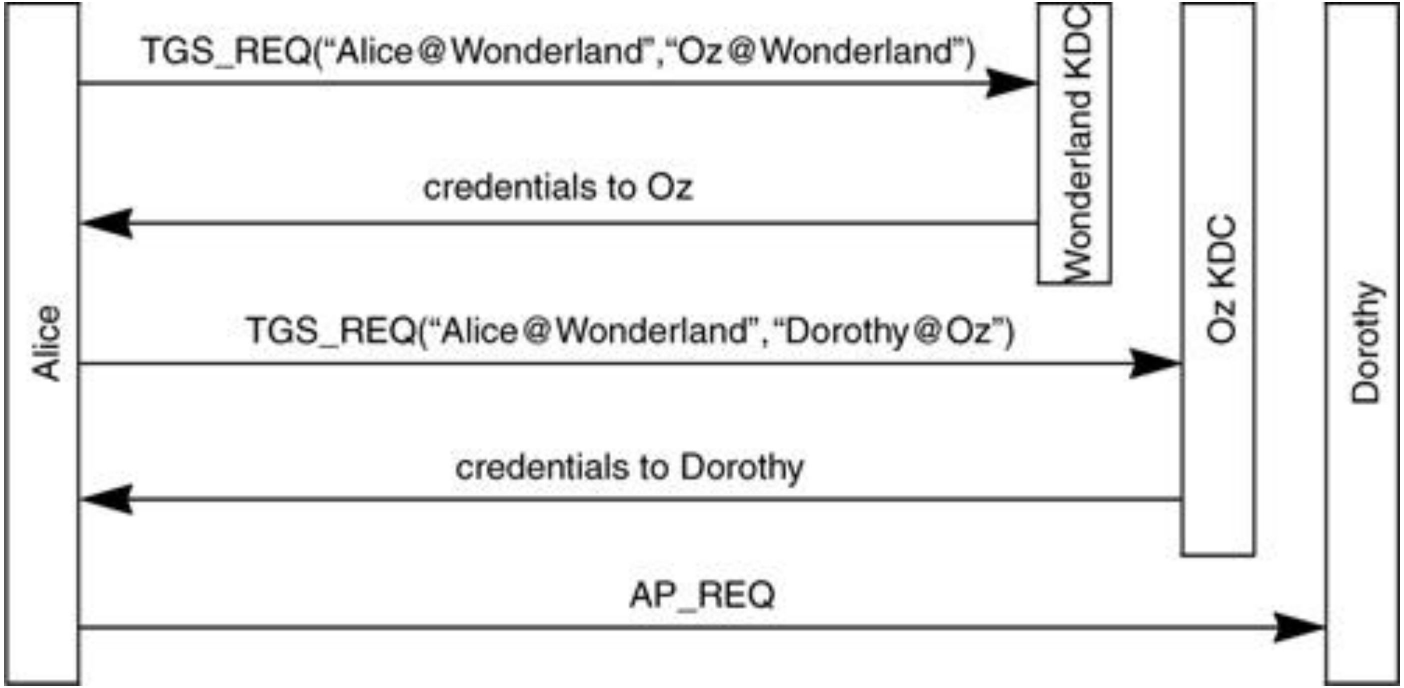
- a single realm in real-world: unlikely since this would mean that all organizations have to trust a single entity…
- each realm has a different master KDC
- all KDCs share the same KDC master key
- two KDCs in different realms have different databases of users

Kerberos V. 4: Realms

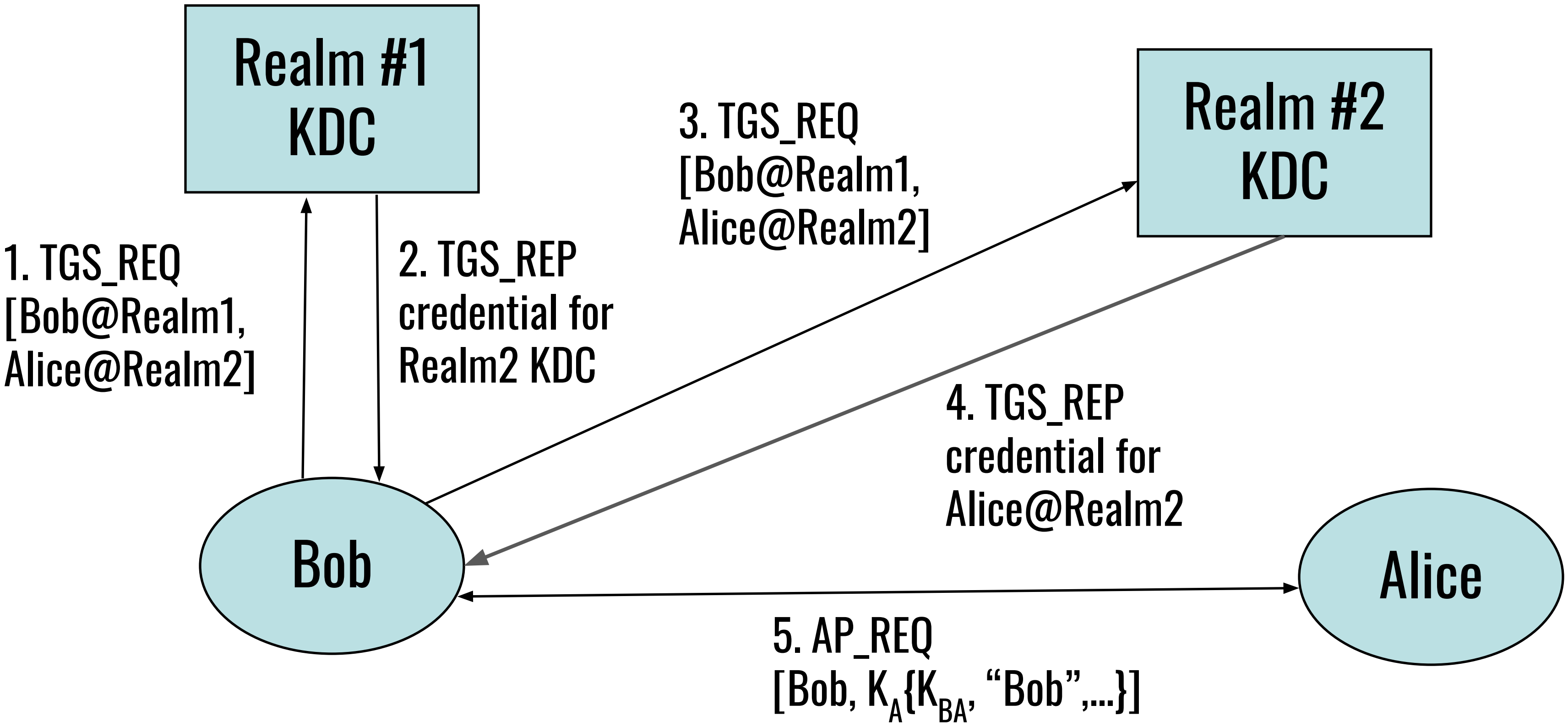


Authentication may be allowed only between realms that have an agreement.

Authentication between realms


Alice
Wonderland KDC
Oz KDC
Dorothy
TGS_REQ("Alice@Wonderland","Oz@Wonderland")
credentials to Oz
TGS_REQ("Alice@Wonderland","Dorothy@Oz")
credentials to Dorothy
AP_REQ

Authentication between realms



Realm #1 KDC

Realm #2 KDC

Bob

Alice

1. TGS_REQ [Bob@Realm1, Alice@Realm2]

2. TGS_REP credential for Realm2 KDC

3. TGS_REQ [Bob@Realm1, Alice@Realm2]

4. TGS_REP credential for Alice@Realm2

5. AP_REQ [Bob, K_A\{K_{BA}, “Bob”,...\}]

Other features

- key version numbers
  - for supporting changes of master keys while keeping existing tickets valid

- encryption for privacy and integrity
  - DES + Propagating/Plaintext Cipher Block Chaining (PCBC)
  - problem: in PCBC two adjacent blocks can be swapped

- encryption for integrity only: custom checksum algorithm

Kerberos: version 5 (RFC 4120)

- Same philosophy
- Major 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

Additional details on Kerberos v4/v5

- RFC 4120: [URL]
- Chapter 13. Charlie Kaufman, Radia Perlman, and Mike Speciner. Network Security - Private Communication in a Public World. Prentice Hall.
- Chapter 14. Charlie Kaufman, Radia Perlman, and Mike Speciner. Network Security - Private Communication in a Public World. Prentice Hall.

Credits

These slides are based on material from:

- Slides of Prof. D’Amore from CNS 2019-2020
- Christof Paar and Jan Pelzl. Understanding Cryptography: A Textbook for Students and Practitioners. Springer. http://www.crypto-textbook.com/
- Charlie Kaufman, Radia Perlman, and Mike Speciner. Network Security - Private Communication in a Public World. Prentice Hall.
- Wikipedia (english version)