Kerberos and Mediated Key Exchange

Lecture 05

Software Security Engineering

Winter 2023
Thompson Rivers University

Kerberos, the three headed dogs that guard hades,

as protocol to key exchange.

Establishing a session key between two entities.

Kerberos is an old idea in computer science,

and is in significant use today.

A TIME-TESTED TOOL

Imagine we have printers and some employees.
It’s not reasonable to give everyone a single username for each printer and expect to type password for every use.

Imagine we have printers and some employees.
It’s not reasonable to give everyone a single username for each printer and expect to type password for every use.

Kerberos abstract away authentication from delivery of the service.

The core idea is

Key Distribution Center (KDC)

Key Distribution Center (KDC)

- a central trusted party
- knows all the nodes in the network
- has authentic channel with all the nodes
- allows for mediated key exchange

what can go wrong?

what can go wrong?

Availability

(Single point of failure)

KDC Operation (in principle)

- Alice ⟶ KDC: “I want to talk to Bob”
- KDC invents a random key K_{AB}
- KDC ⟶ Alice: {use K_{AB} for Bob}$_{KA}$
- KDC ⟶ Bob: {use K_{AB} for Alice}$_{KB}$
- what can go wrong? (rather than KDC’s availability)

{Message}$_{key}$ means the content of the message is protected with the key

KDC Operation (in principle)

- Alice → KDC: “I want to talk to Bob”
- KDC invents a random key K_{AB}
- KDC → Alice: $\{\text{use } K_{AB} \text{ for Bob}\}_{KA}$
- KDC → Bob: $\{\text{use } K_{AB} \text{ for Alice}\}_{KB}$
- what can go wrong? (rather than KDC’s availability)
  - Bob should be online

KA = Alice’s Key
KB = Bob’s Key

Only KDC and the particular user know the key

KDC Operation (in practice)

- Alice → KDC: “I want to talk to Bob”
- KDC invents a random key K_{AB}
- KDC → Alice:
  - {use K_{AB} for Bob}$_{KA}$
  - {use K_{AB} for Alice}$_{KB}$
    - this is called a ticket
- Alice → Bob: “Hi I’m Alice! ticket = {use K_{AB} for Alice}$_{KB}$

KDC Operation (in practice)

- Alice → KDC: "I want to talk to Bob"
- KDC invents a random key K_{AB}
- KDC → Alice:
  - {use K_{AB} for Bob}$_{KA}$
  - {use K_{AB} for Alice}$_{KB}$
    - this is called a **ticket**
- Alice → Bob: "Hi I'm Alice! ticket = {use K_{AB} for Alice}$_{KB}$
- what can go wrong?

This ticket does not have any notion of time.

Alice can uses the ticket years later.

Needham-Schroeder Protocol

- goal is key transport on insecure networks (like the Internet)
  - e.g., you print a document at TRU
- use of a trusted third party to mediate keys for people
  - you don't need to do key exchange with everyone before communicating
- two types
  - symmetric key
    - goal: establish a session key between Alice and Bob
  - public key
    - goal: provide mutual authentication between Alice and Bob
- both protocols insecure as proposed!
  - because crypto is hard

N-S Symmetric

- Notation:
  - (A)lice, (B)ob, (S)erver (trusted by both A and B)
  - K_{xy} symmetric key known only by X and Y
  - N_x a random nonce generated by X
  - $\{data\}_{Kxy}$ data is encrypted with K_{xy}
- Protocol:
  - $A \rightarrow S : A, B, N_A$
  - $S \rightarrow A : \{N_A, K_{AB}, B, \{K_{AB}, A\}_{KBS}\}_{KAS}$
  - $A \rightarrow b : \{K_{AB}, A\}_{KBS}$
  - $B \rightarrow A : \{N_B\}_{KAB}$
  - $A \rightarrow B : \{N_B - 1\}_{KAB}$
- where is the flaw?

N-S Symmetric

- Protocol:
  - $A \rightarrow S : A, B, N_A$
  - $S \rightarrow A : \{N_A, K_{AB}, B, \{K_{AB}, A\}_{KBS}\}_{KAS}$
  - $A \rightarrow b : \{K_{AB}, A\}_{KBS}$
  - $B \rightarrow A : \{N_B\}_{KAB}$
  - $A \rightarrow B : \{N_B - 1\}_{KAB}$
- where is the flaw?
  - replay attack. if Eve learns one single key only once (by compromising the system), Eve can start with line 3
  - Thus is secure under assumption that Eve never learn the key. But is vulnerable in a way that if Eve only learn the key once, the protocol is broken forever.

One fix

- amend the first two line to:
  - $A \longrightarrow B : A$
  - $B \longrightarrow A : \{A, N'_B\}_{KBS}$
  - $A \longrightarrow S : \{A, B, N_A, \{A, N'_B\}_{KBS}\}_{KAS}$
  - $S \longrightarrow A : \{N_A, K_{AB}, B, \{K_{AB}, A, N'_B\}_{KBS}\}_{KAS}$
- why does this fix the flaw?

N-S Public Key

- Notation:
  - A, B, S the same
  - K_{PX} - public key for X (= A, B, or S)
  - K_X- private key for X (paired with K_{PX})
  - $\{message\}_{PX}$ - encrypted for X
  - $\{message\}_X$ - signed by X
- Protocol:
  - $A \longrightarrow S : A, B$
  - $S \longrightarrow A : \{K_{PB}, B\}_{KS}$
  - $A \longrightarrow B : \{N_A, A\}_{KPB}$
  - $B \longrightarrow S : B, A$
  - $S \longrightarrow B : \{K_{PA}, A\}_{KS}$
  - $B \longrightarrow A : \{N_A, N_B\}_{KPA}$
  - $A \longrightarrow B : \{N_B\}_{KPB}$
- where is the flaw?

N-S Mafia Fraud

- $A \longrightarrow S : A, E$
- $S \longrightarrow A : \{K_{PE}, E\}_{KS}$
- $A \longrightarrow E : \{N_A, A\}_{KPE}$
- $E \longrightarrow B : \{N_A, A\}_{KPB}$
    - E can decrypt this ans so know N_A
- $B \longrightarrow E : \{N_A, N_B\}_{KPA}$
    - E cannot decrypt this so does not learn N_B
- $E \longrightarrow A : \{N_A, N_B\}_{KPA}$
    - E can just reply it verbatim
- $A \longrightarrow E : \{N_B\}_{KPE}$
    - E learns N_B by design
- $E \longrightarrow E : \{N_B\}_{KPB}$
    - success

Lowe fixed this flaw

N-S-Lowe

- Notation:
  - A, B, S the same
  - K_{PX} - public key for X (= A, B, or S)
  - K_X- private key for X (paired with K_{PX})
  - $\{message\}_{PX}$ - encrypted for X
  - $\{message\}_X$ - signed by X
- Protocol:
  - $A \longrightarrow S : A, B$
  - $S \longrightarrow A : \{K_{PB}, B\}_{KS}$
  - $A \longrightarrow B : \{N_A, A\}_{KPB}$
  - $B \longrightarrow S : B, A$
  - $S \longrightarrow B : \{K_{PA}, A\}_{KS}$
  - $B \longrightarrow A : \{N_A, N_B, \mathbf{B}\}_{KPA}$
  - $A \longrightarrow B : \{N_B\}_{KPB}$

Now for Kerberos, based on symmetric N-S

Many-to-Many Authentication

- how to prove identity when requesting services on Network (e.g., the Internet)
  - many users, many services (mail, printer, servers, etc.)
  - “single sign-on” (SSO)
- naive solution: every server knows every user password
  - insecure: break into one server, compromise all users
  - inefficient: to change password, user must contact all servers

Enter Kerberos

Requirements

- security
  - against attacks by passive eavesdroppers
  - against attacks by actively malicious users
- transparency
  - users shouldn’t notice authentications taking place
  - password entering fine, as long as not all the time
- scalability
  - lots of users, lots of servers

Threats

- user impersonation
  - malicious user with access to a workstation pretended to be another user from same workstation
- network address impersonation
  - malicious user changes network address of their workstation to impersonate another workstation
- eavesdropping, tampering, replay
  - malicious user eavesdrops, tampers, or replay other users' conversations to gain unauthorized access

Solution: Trusted Third Party (TTP)

- user proves identity to trusted third party (TTP), requests a ticket for service
- TTP knows all users and services, can grant access
- user gets a ticket
- ticket is used to access service
- TTP is **authentication service** on the network
  - convenient (but also single point of failure!)
  - requires high level of physical security

Ticket Requirements

- ticket gives holder access to a network service
- ticket proves that a user has authenticated
- user should not be able to create a ticket
- user should not be able to delegate tickets

Ticket Logistics

- authentication service encrypts some information with a key known to the server
  - e.g., the printer can decrypt it, but not the user
- the user simply forwards the ticket to the printer, but cannot
- create one or read it
- server decrypts the ticket and verifies the information

Ticket Contents

- ticket must include everything to prevent abuse
  - user using tickets to other servers
  - user using tickets after they lose access
    - e.g., they've been fired
  - user giving tickets to other users to use
- ticket includes:
  - user name
  - server name
  - address of user's workstation
  - ticket lifetime

Naive Authentication

- protocol:
  - user sends password to authentication server
  - server provides an encrypted ticket
- problems:
  - insecure: eavesdropper sees the password and can impersonate
  - inconvenient: need to send the password each time to get the ticket
    - separate authentication for email, printing, etc.

Two-Step Authentication

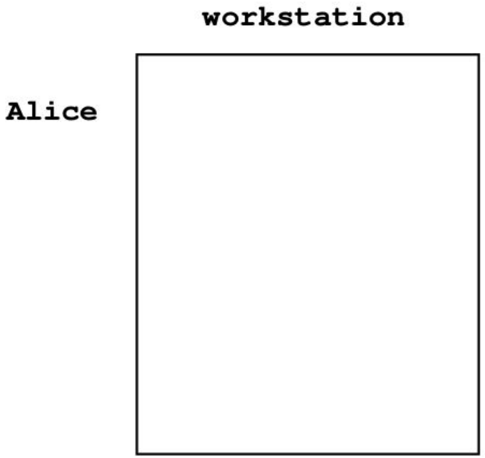
- protocol:
  - user authenticates to the key distribution centre (KDC)
  - gets a special ticket granting service (TGS) ticket
  - user gives TGS ticket to TGS server when needed
  - gets encrypted service ticket (e.g., for printer)
  - user gives ticket to printer

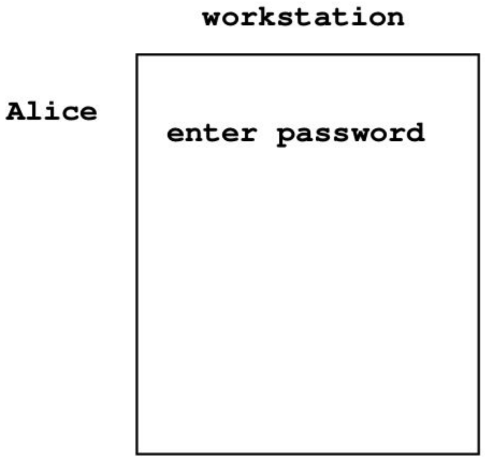
Threats to Two Step

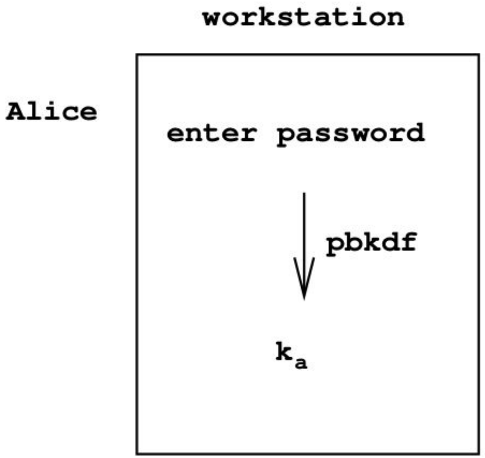
- ticket hijacking
  - malicious user steals service ticket
  - uses it on the same workstation
    - network address verification doesn’t help
  - server must verify that the user who gives the ticket is the same who was issued
- no server authentication
  - attacker may misconfigure the network so they receive messages sent to server
  - deny service or capture private information

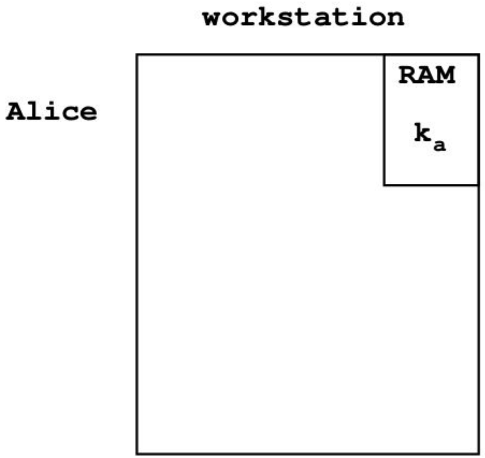
Kerberos

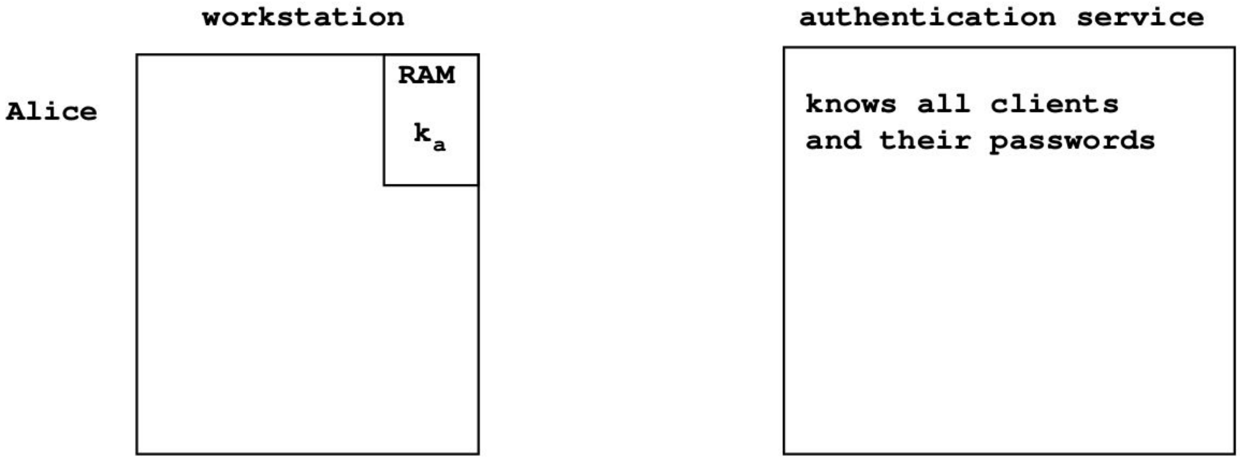
- K_C is a long-term key of client C
  - derived from the user's password
- K_{TGS} is a long-term key of the TGS
  - known by KDC and TGS
- K_V is a long-term key of network service V
  - known to V and TGS; each V has its own key
- $K_{C,TGS}$ is a short-term session key b/w C and TGS
  - created by KDC, known to C and TGS
- $K_{C,V}$ is a short-term session key b/w C and V
  - created by TGS, known to C and TGS

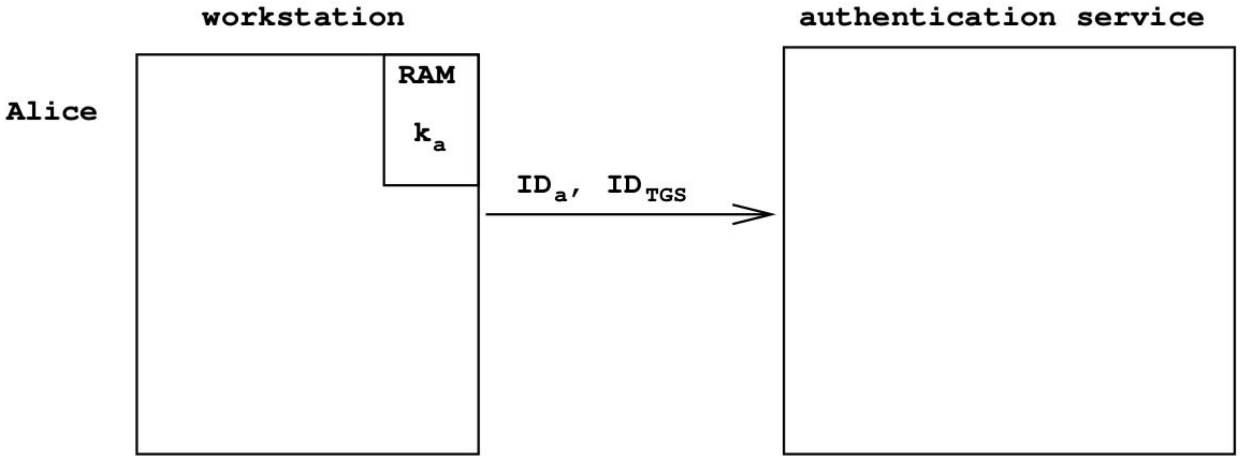

workstation
Alice

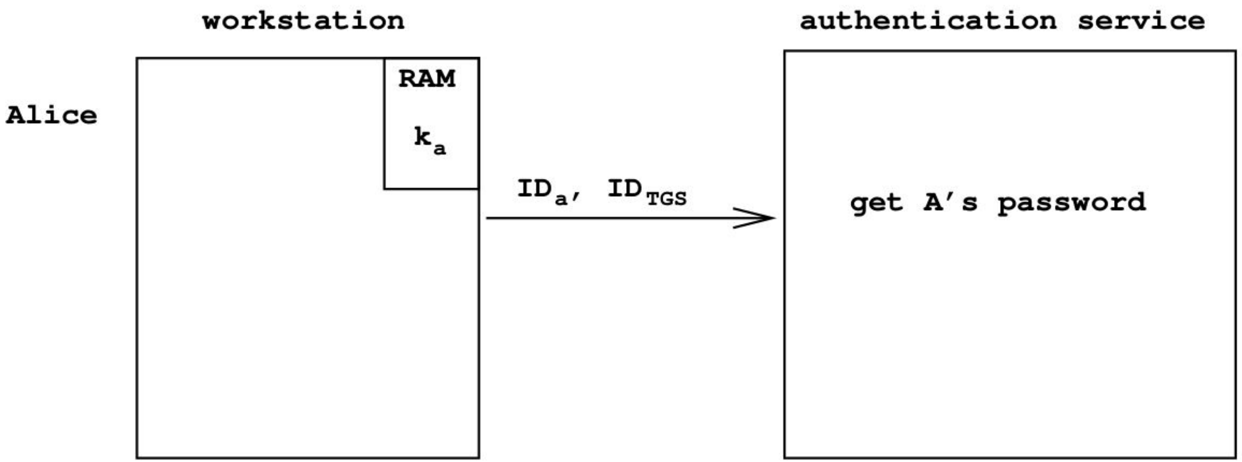

workstation
Alice
enter password

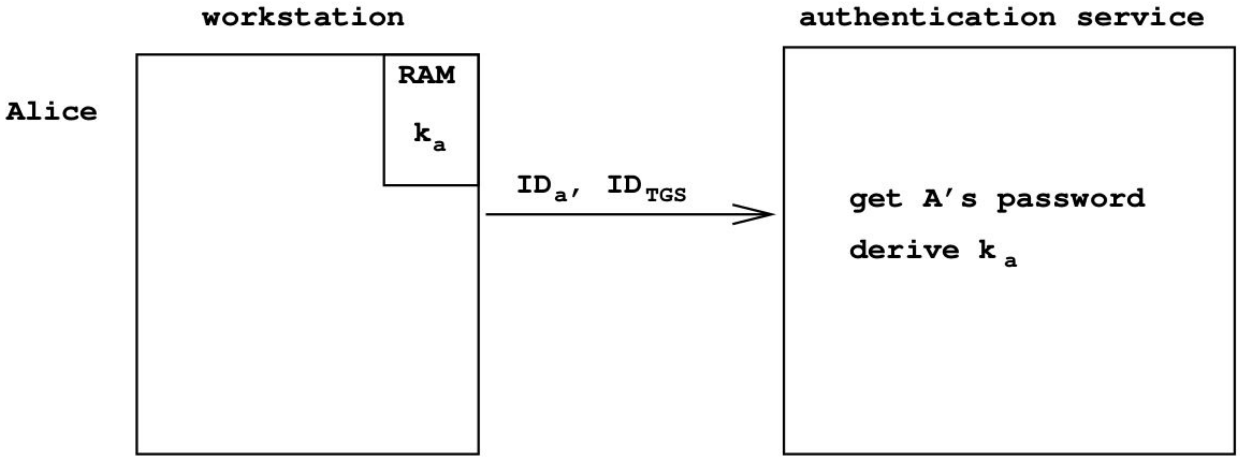

workstation
Alice
enter password
pbkdf
k_a

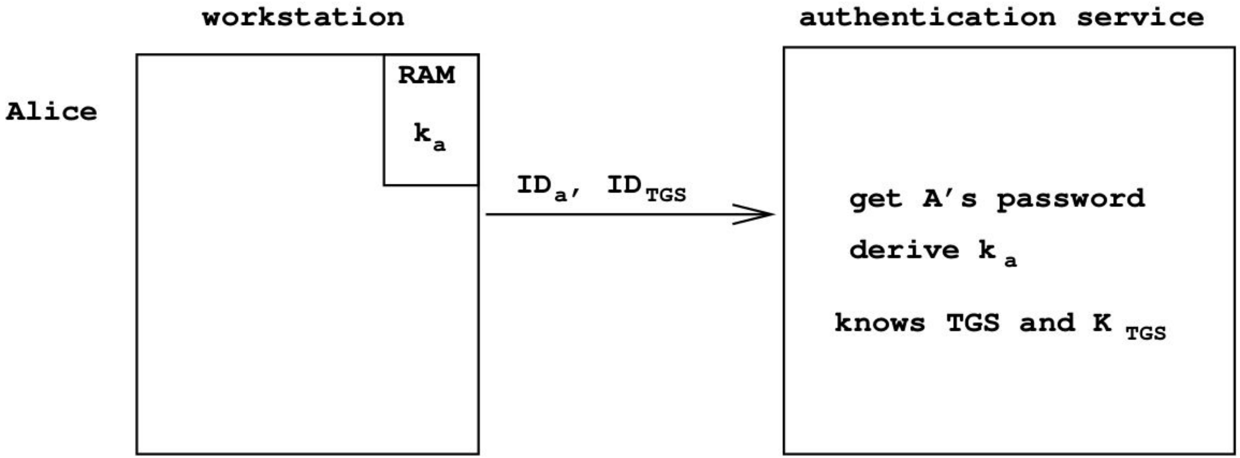

workstation
Alice
RAM
k_a

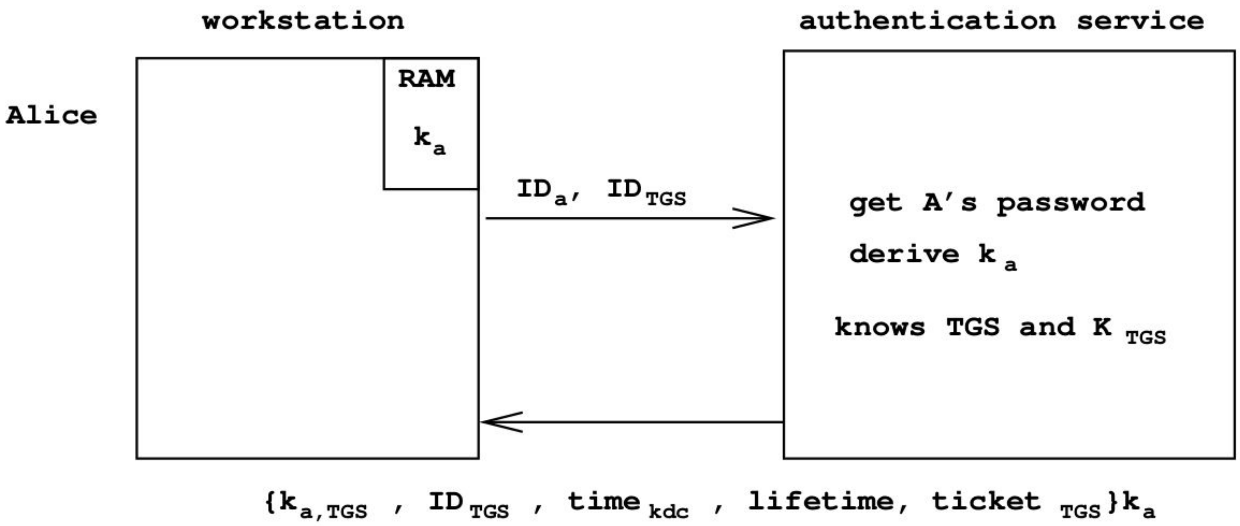

workstation
authentication service
RAM
Alice
k_a
knows all clients
and their passwords

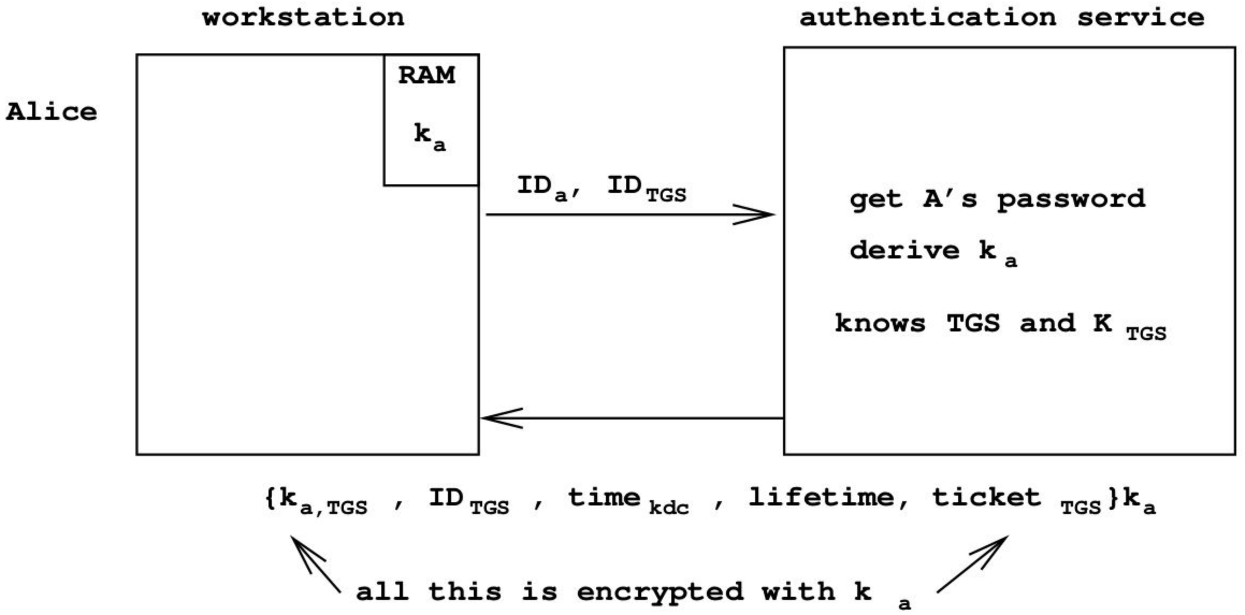

workstation
authentication service
Alice
RAM
k_a
ID_a, ID_{TGS}

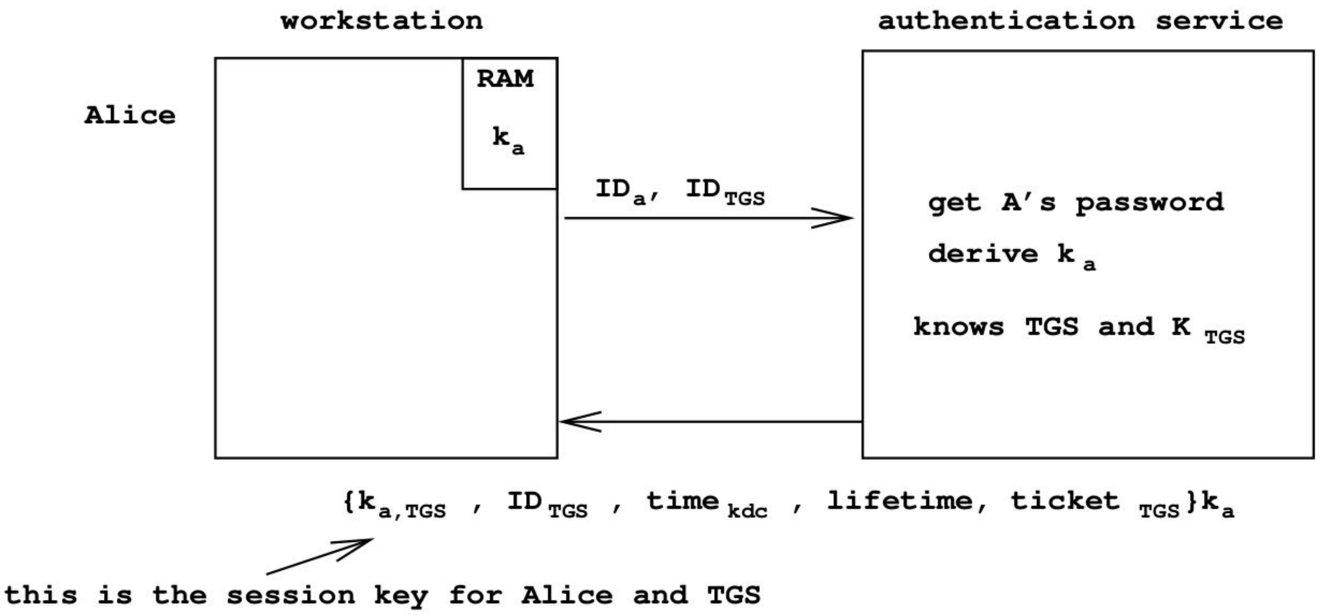

workstation
authentication service
Alice
RAM
k_a
ID_a, ID_TGS
get A's password

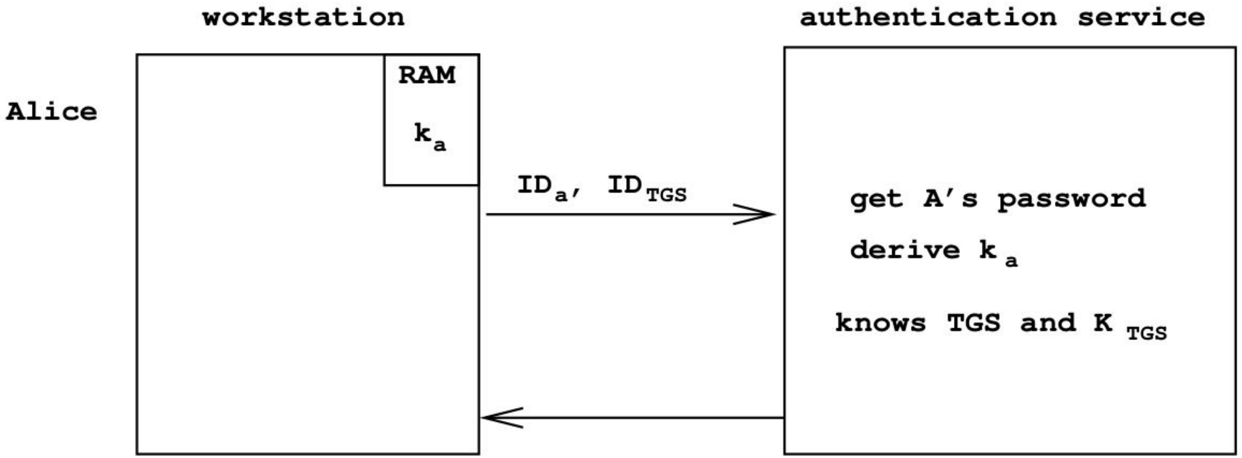

workstation
authentication service
Alice
RAM
k_a
ID_a, ID_{TGS}
get A's password
derive k_a


workstation
authentication service
Alice
RAM
k_a
ID_a, ID_{TGS}
get A's password
derive k_a
knows TGS and K_{TGS}


workstation
authentication service
Alice
RAM
k a
ID a, ID TGS
get A's password
derive k a
knows TGS and K TGS
{k a,TGS , ID TGS , time kdc , lifetime, ticket TGS}k a

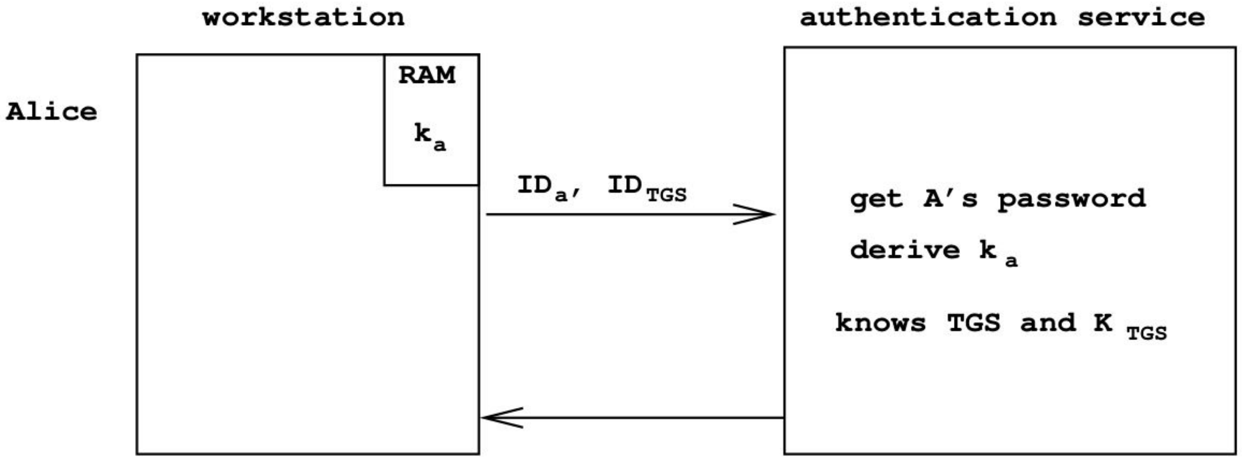

workstation
authentication service
Alice
RAM
k a
ID a, ID TGS
get A's password
derive k a
knows TGS and K TGS
{k a,TGS , ID TGS , time kdc , lifetime, ticket TGS}k a
all this is encrypted with k a


workstation
authentication service
Alice
RAM
k_a
ID_a, ID_{TGS}
get A's password
derive k_a
knows TGS and K_{TGS}
$\{k_{a,TGS}$, ID_{TGS} , $time_{kdc}$, lifetime, $ticket_{TGS}\}k_a$
this is the session key for Alice and TGS



$$\{k_{a,TGS}, ID_{TGS}, time_{kdc}, lifetime, ticket_{TGS}\}k_a$$

$$ticket_{TGS} = \{k_{A,TGS}, ID_A, time_{KCD}, lifetime, ID_{TGS}, addr_A\}k_{TGS}$$

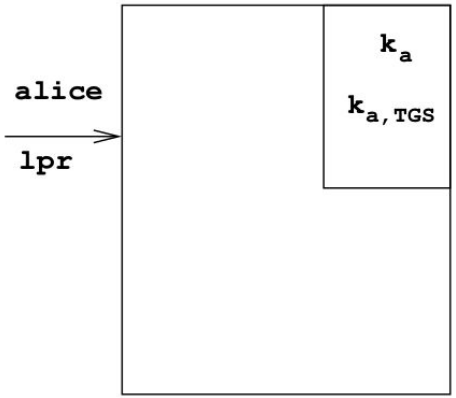


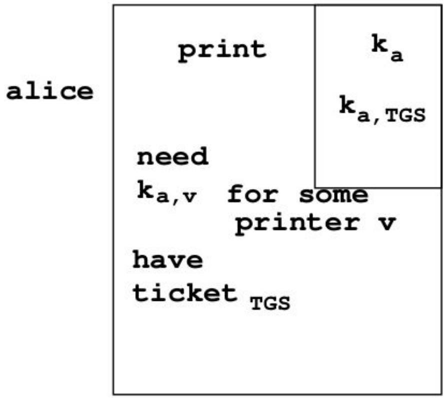
$\{k_{a,TGS}$, ID_{TGS} , $time_{kdc}$, lifetime, $ticket_{TGS}\}k_a$

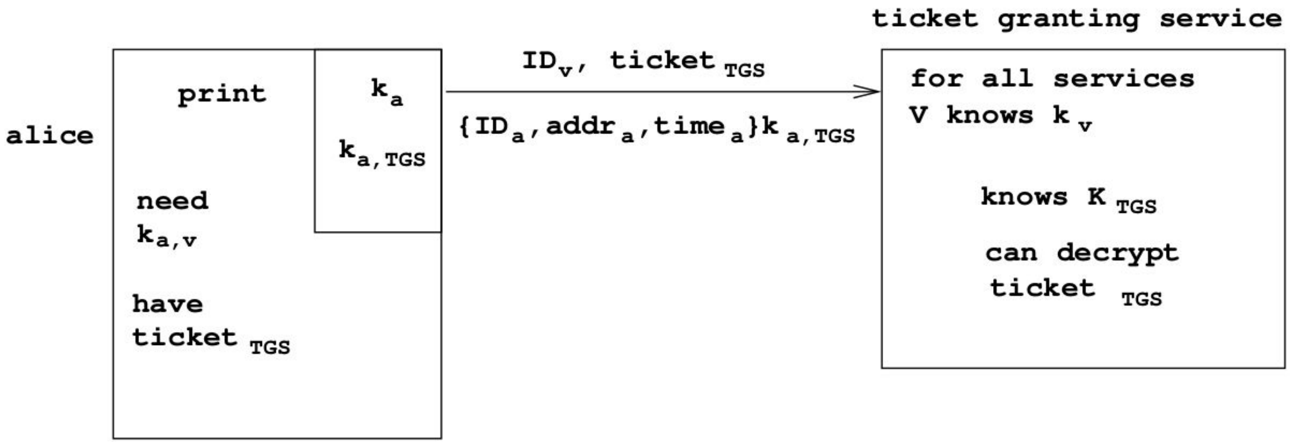
$ticket_{TGS}=\{k_{A,TGS}$, ID_A , $time_{KCD}$, lifetime, ID_{TGS}, $addr_A\}k_{TGS}$

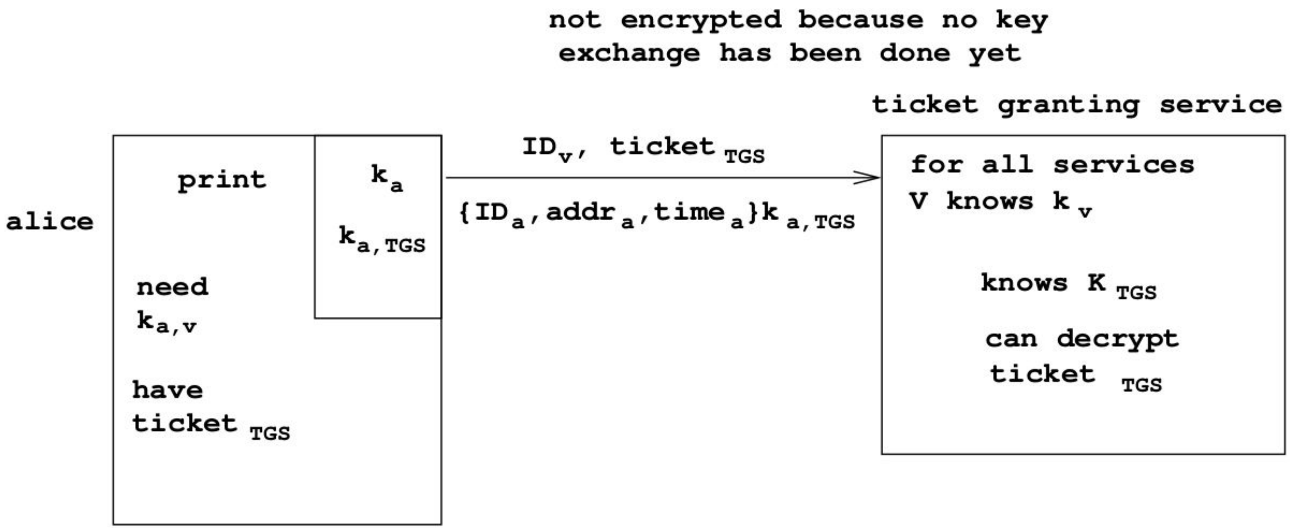
Alice can't understand this, but is expected to deliver it to TGS

alice
wants
to
print

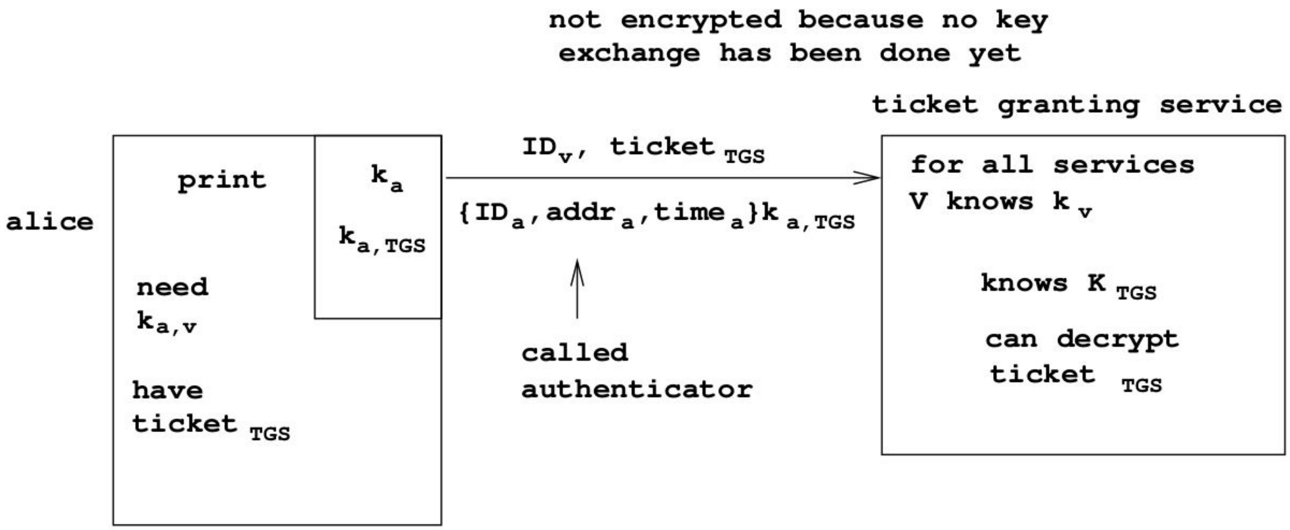

alice
lpr
k_a
$k_{a,TGS}$

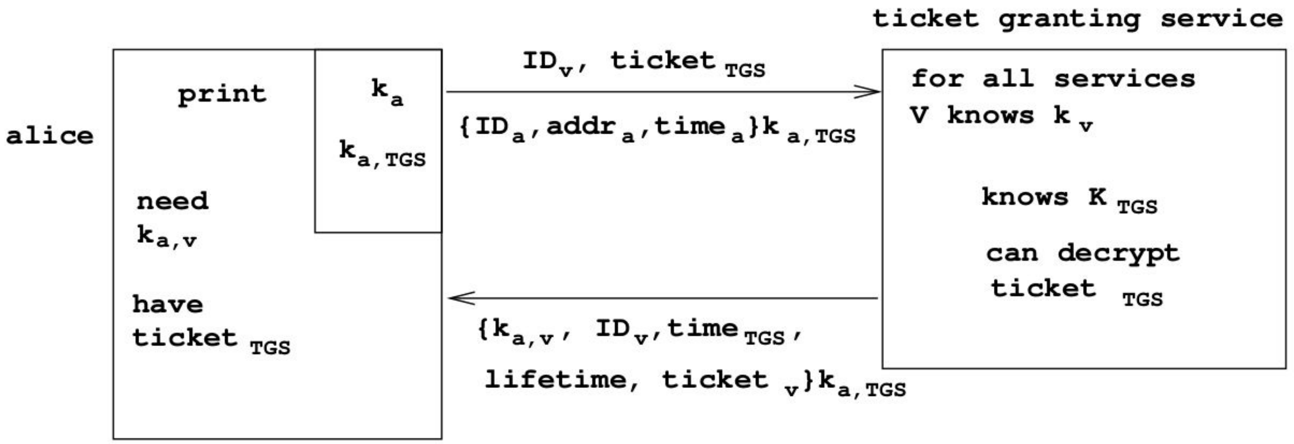

alice
print
k_a
$k_{a,TGS}$
need
$k_{a,v}$ for some printer v
have
$ticket_{TGS}$

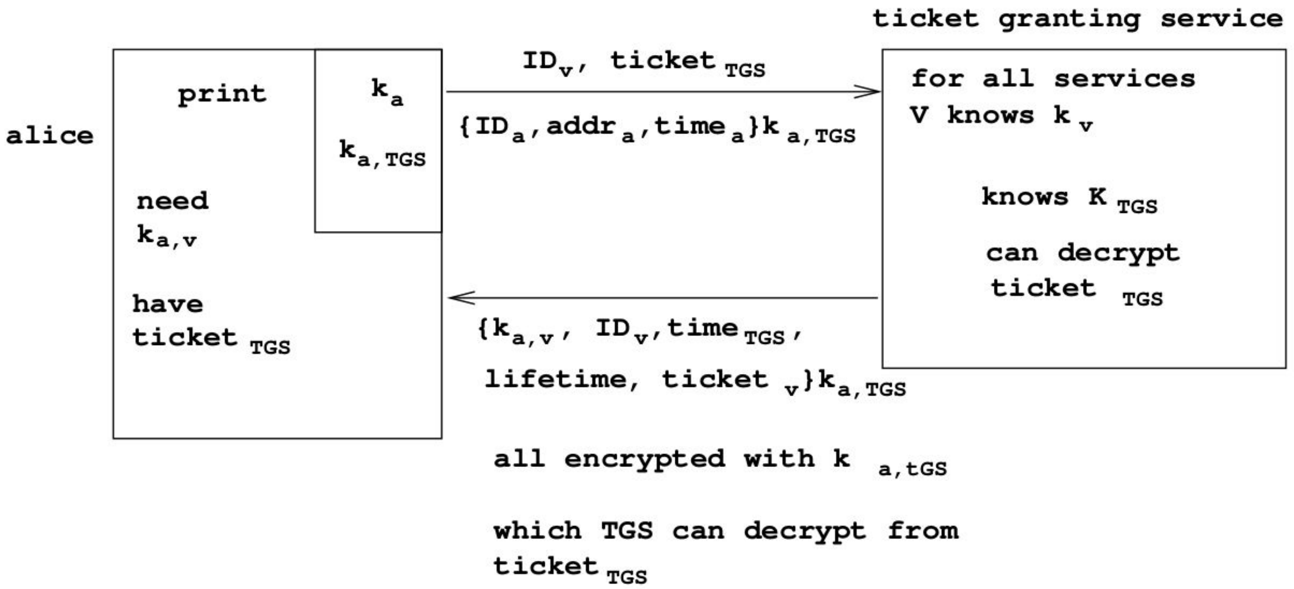

ticket granting service
alice
print
k_a
$k_{a,TGS}$
need
$k_{a,v}$
have
ticket $_{TGS}$
ID_v, ticket $_{TGS}$
$\{ID_a, addr_a, time_a\}k_{a,TGS}$
for all services
V knows k_v
knows K_{TGS}
can decrypt
ticket $_{TGS}$

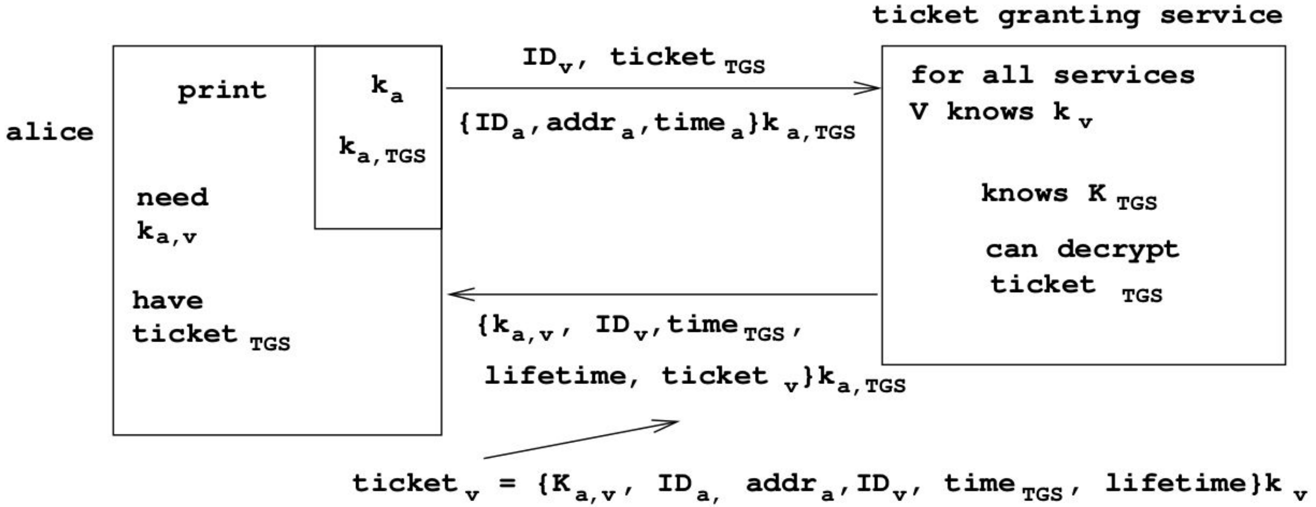


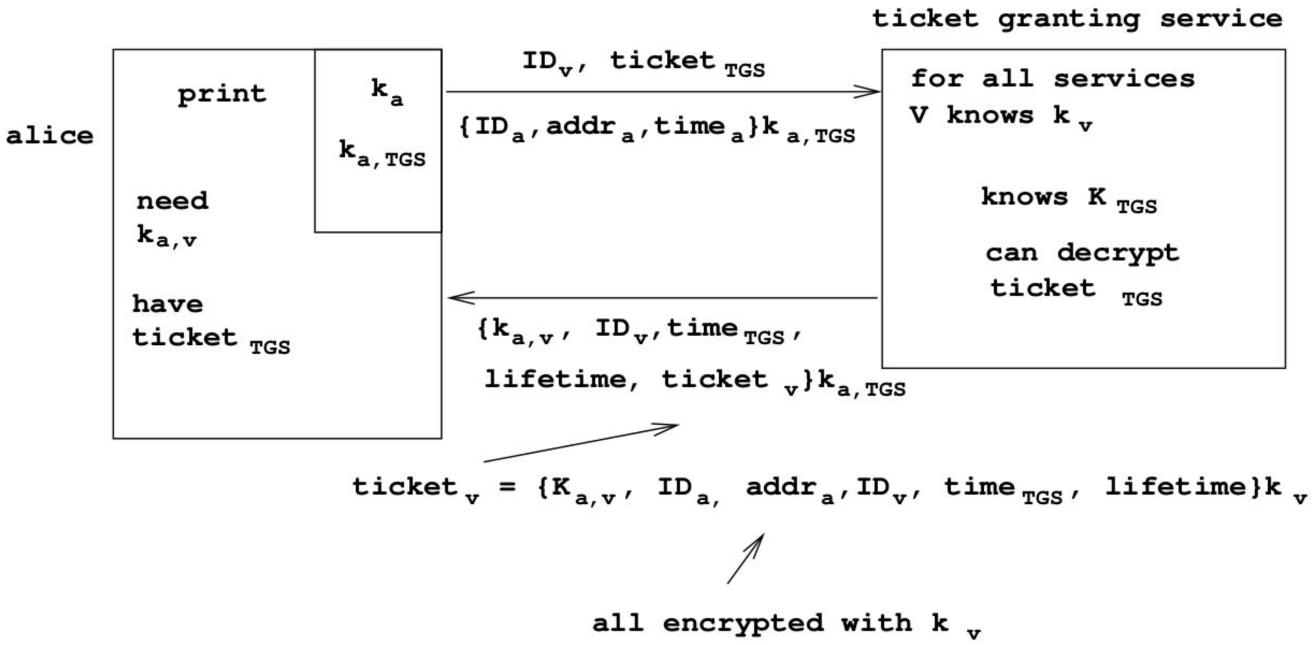
not encrypted because no key
exchange has been done yet
ticket granting service
alice
print
k_a
$k_{a,TGS}$
need
$k_{a,v}$
have
ticket $_{TGS}$
ID_v, ticket $_{TGS}$
$\{ID_a, addr_a, time_a\}k_{a,TGS}$
for all services
V knows k_v
knows K_{TGS}
can decrypt
ticket $_{TGS}$


not encrypted because no key exchange has been done yet
ticket granting service
alice
print
k_a
$k_{a,TGS}$
need $k_{a,v}$
have $ticket_{TGS}$
ID_v, $ticket_{TGS}$
$\{ID_a, addr_a, time_a\}k_{a,TGS}$
called authenticator
for all services V knows k_v
knows K_{TGS}
can decrypt $ticket_{TGS}$

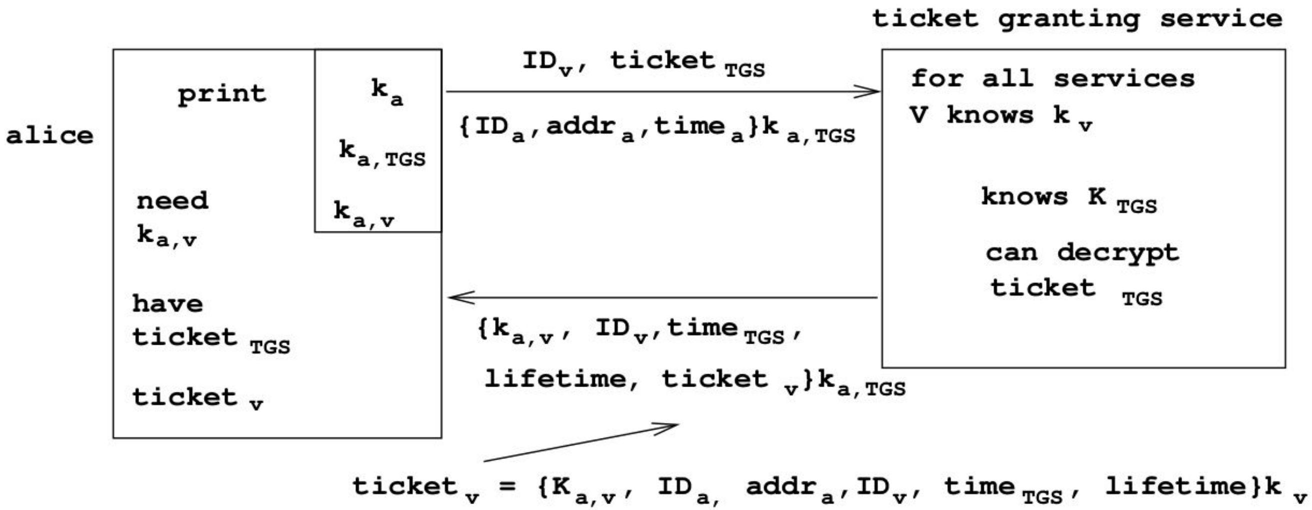

ticket granting service
alice
print
k_a
$k_{a,TGS}$
need
$k_{a,v}$
have
ticket $_{TGS}$
ID_v, ticket $_{TGS}$
$\{ID_a, addr_a, time_a\}k_{a,TGS}$
for all services
V knows k_v
knows K_{TGS}
can decrypt
ticket $_{TGS}$
$\{k_{a,v}$, ID_v, $time_{TGS}$,
lifetime, ticket $_v\}k_{a,TGS}$


ticket granting service
alice
print
k_a
$k_{a,TGS}$
need $k_{a,v}$
have $ticket_{TGS}$
ID_v, $ticket_{TGS}$
$\{ID_a, addr_a, time_a\}k_{a,TGS}$
for all services
V knows k_v
knows K_{TGS}
can decrypt $ticket_{TGS}$
$\{k_{a,v}$, ID_v, $time_{TGS}$,
lifetime, $ticket_v\}k_{a,TGS}$
all encrypted with $k_{a,tGS}$
which TGS can decrypt from $ticket_{TGS}$

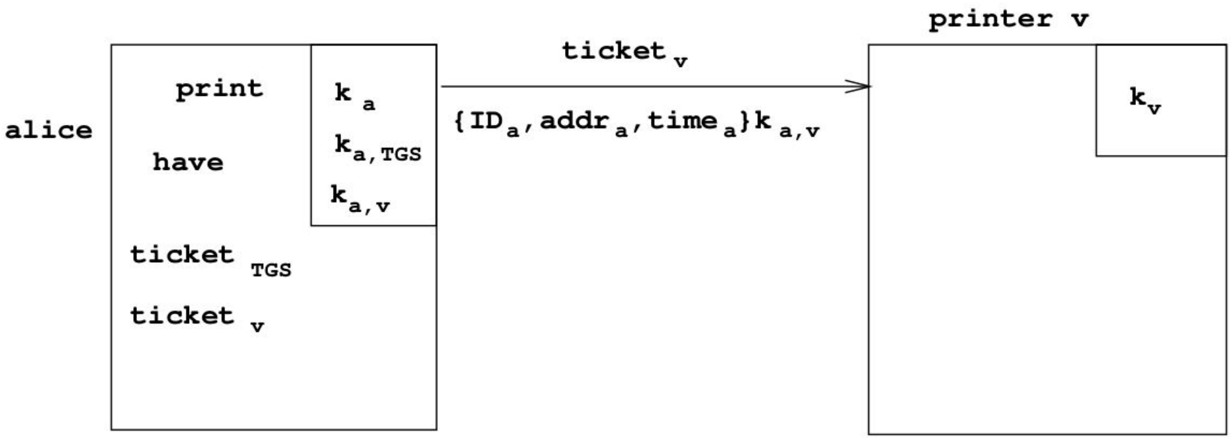

ticket granting service
alice
print
k_a
$k_{a,TGS}$
need $k_{a,v}$
have ticket $_{TGS}$
ID_v, ticket $_{TGS}$
$\{ID_a, addr_a, time_a\}k_{a,TGS}$
for all services V knows k_v
knows K_{TGS}
can decrypt ticket $_{TGS}$
$\{k_{a,v}$, ID_v, $time_{TGS}$, lifetime, ticket $_v\}k_{a,TGS}$
ticket $_v$ = $\{K_{a,v}$, ID_a, $addr_a$, ID_v, $time_{TGS}$, lifetime$\}k_v$

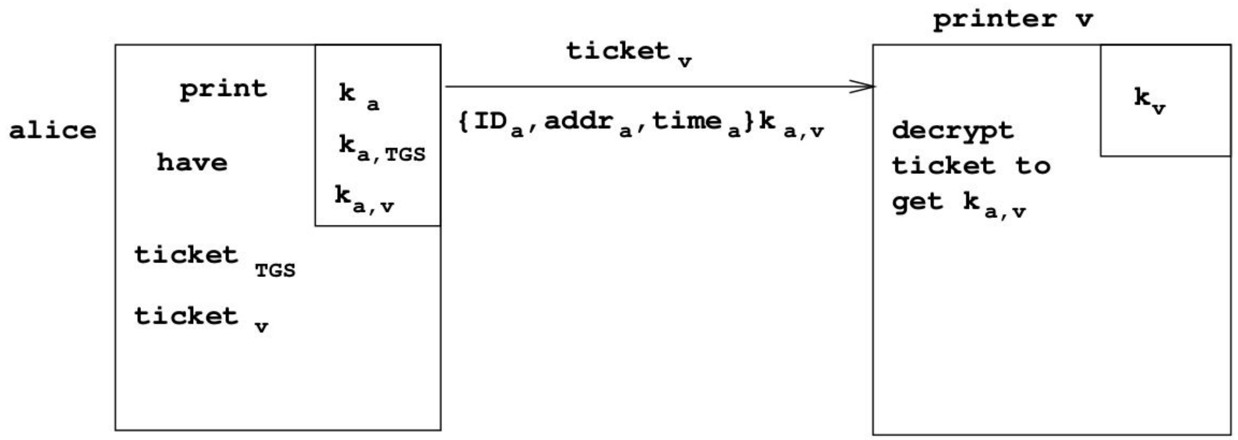


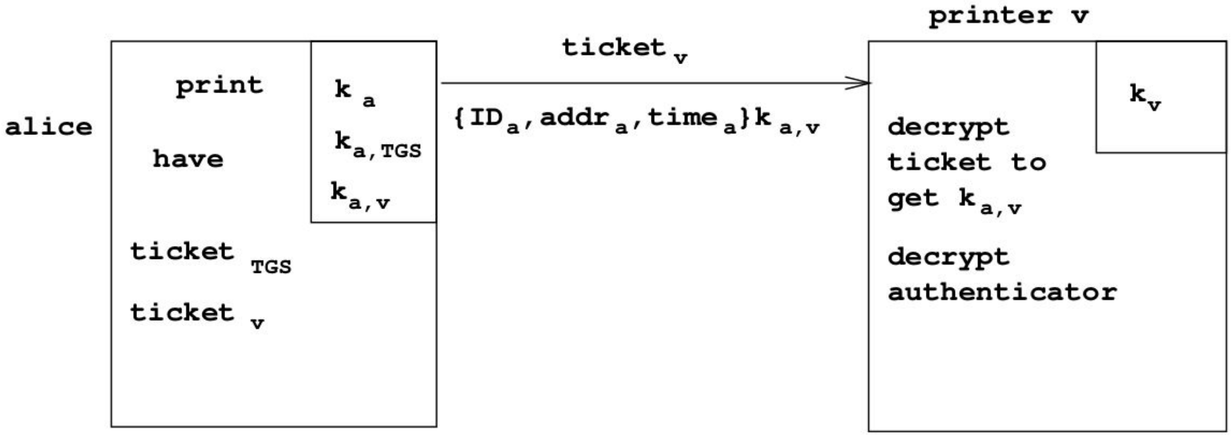
ticket granting service
alice
print
k_a
$k_{a,TGS}$
need $k_{a,v}$
have $ticket_{TGS}$
ID_v, $ticket_{TGS}$
$\{ID_a, addr_a, time_a\}k_{a,TGS}$
for all services
V knows k_v
knows K_{TGS}
can decrypt $ticket_{TGS}$
$\{k_{a,v}, ID_v, time_{TGS},$
$lifetime, ticket_v\}k_{a,TGS}$
$ticket_v = \{K_{a,v}, ID_a, addr_a, ID_v, time_{TGS}, lifetime\}k_v$
all encrypted with k_v

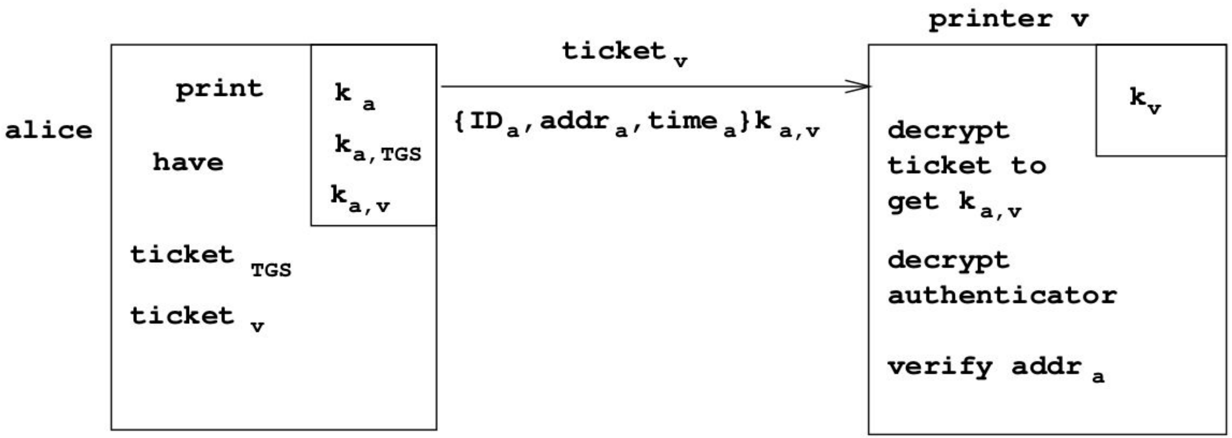


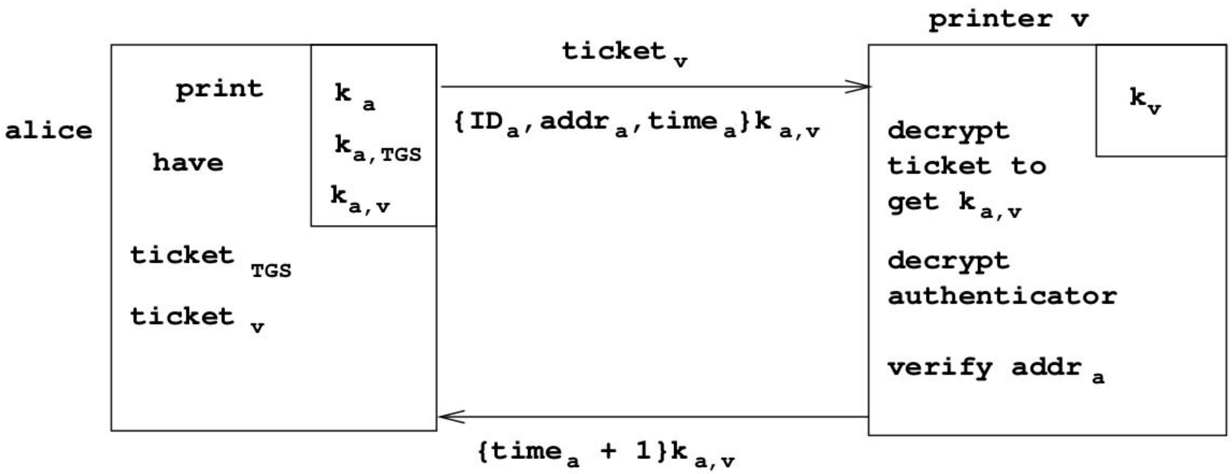
ticket granting service
alice
print
k_a
$k_{a,TGS}$
$k_{a,v}$
need $k_{a,v}$
have ticket $_{TGS}$
ticket $_v$
ID_v, ticket $_{TGS}$
$\{ID_a, addr_a, time_a\}k_{a,TGS}$
for all services
V knows k_v
knows K_{TGS}
can decrypt ticket $_{TGS}$
$\{k_{a,v}, ID_v, time_{TGS}$, lifetime, ticket $_v\}k_{a,TGS}$
ticket $_v$ = $\{K_{a,v}, ID_a, addr_a, ID_v, time_{TGS}$, lifetime$\}k_v$


alice
print
have
k_a
$k_{a,TGS}$
$k_{a,v}$
$ticket_{TGS}$
$ticket_v$
$ticket_v$
$\{ID_a, addr_a, time_a\}k_{a,v}$
printer v
k_v

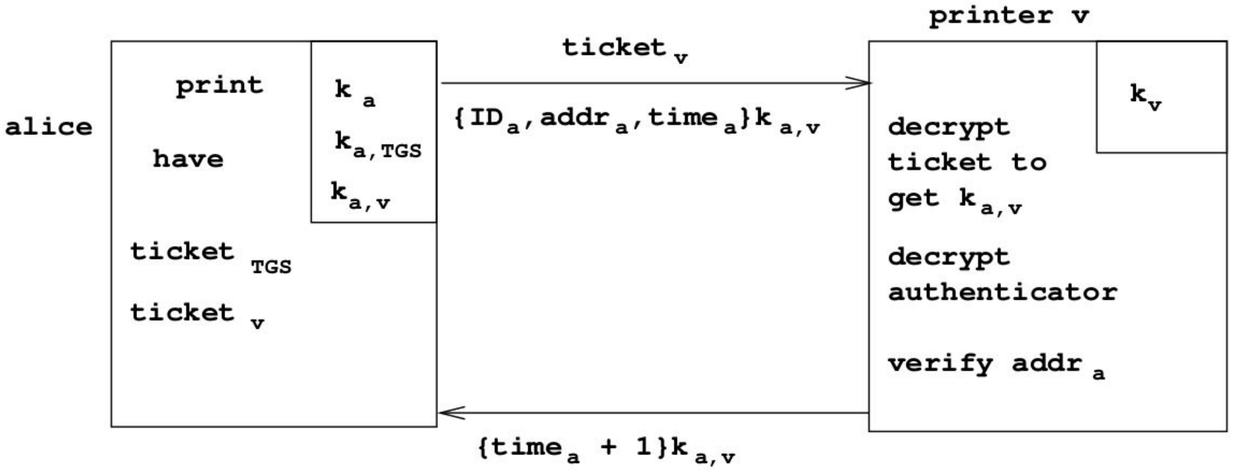

alice
print
have
k a
k a,TGS
k a,v
ticket TGS
ticket v
ticket v
{ID a, addr a, time a}k a,v
printer v
k v
decrypt ticket to get k a,v


alice
print
have
ticket TGS
ticket v
k a
k a,TGS
k a,v
ticket v
{ID a, addr a, time a}k a,v
printer v
k v
decrypt
ticket to
get k a,v
decrypt
authenticator


alice
print
have
k_a
$k_{a,TGS}$
$k_{a,v}$
$ticket_{TGS}$
$ticket_v$
$ticket_v$
$\{ID_a, addr_a, time_a\}k_{a,v}$
printer v
k_v
decrypt ticket to get $k_{a,v}$
decrypt authenticator
verify $addr_a$



printer v
alice
print
have
k_a
$k_{a,TGS}$
$k_{a,v}$
$ticket_{TGS}$
$ticket_v$
$ticket_v$
$\{ID_a, addr_a, time_a\}k_{a,v}$
k_v
decrypt ticket to get $k_{a,v}$
decrypt authenticator
verify $addr_a$
$\{time_a + 1\}k_{a,v}$



compute a non-trivial function on $time_a$ that proves knowledge of $k_{a,v}$ and thereby knowledge of k_v

Kerberos in Large Networks

- one KDC isn't enough
- network is divided into **realms**
  - KDCs in different realms have different key databases
- to access a service in another realm, users must:
  - get a ticket for home-realm TGS from home-realm KDC
  - get a ticket for remote-realm TGS from home-realm TGS
    - i.e., were the remote-realm TGS just a normal home-realm network service
  - get ticket for remote service from that realm's TGS
  - use remote-realm ticket to access service

Important Ideas in Kerberos

- short-term **session keys**
  - long-term secrets used only to secure delivery of short-term keys
  - separate session key for each user-server pair
    - re-used by multiple sessions between same user/server
- symmetric crypto only
  - fast, no expensive operations
- trusted third party
  - new users only need to register a password

Important Ideas in Kerberos

- proof of identity based on **authenticators**
  - client encrypts his identity, addr, time with session key
    - knowledge of key proves client has authenticated to KDC
    - also prevents replays if clocks are globally synchronized
  - server learns this key separately
    - via encrypted ticket that client can't decrypt
    - verifies client's authenticator