Public Key Crypto

Public Key Crypto

- Alice has a private secret key K and a public key PK
- Alice gives everyone PK
- to encrypt M, compute $C = E_{PK}(M)$
- to decrypt C, compute $M = D_K(C)$
- only Alice has K, so only Alice can compute D_K
- E and D are related, K and PK are related
- we assume that knowing E, D, PK, and lots of Cs and Ms does not reveal K

Public Key Signature

- Alice has a private secret key K and a public key PK
- Alice gives everyone PK
- to sign M, compute $S = E_K(H(M))$
- to verify S is a signature for M, anyone can check if $H(M) = D_{PK}(S)$
- only Alice has K, so only Alice can compute signatures

Public Key Signatures

- somewhat like a real-world signature
  - only I can create the signature
- exception
  - private keys can be stolen / copied
  - may not know it is compromised
  - signing things you do not agree with

Public key crypto is slow.
So usually we use it only to exchange
encryption keys used for block ciphers
like AES, and to sign hashes of messages.

Two kinds of secret keys:
session keys and **long-term** keys.

Two kinds of secret keys:
session keys and **long-term** keys.
Session keys are used for a short time

Two kinds of secret keys:
session keys and **long-term** keys.
Session keys are used for a short time
e.g., a random session key is used to encrypt
network traffic when for a single socket
from connect() to close()

Two kinds of secret keys:
session keys and **long-term** keys.
Session keys are used for a short time
e.g., a random session key is used to encrypt
network traffic when for a single socket
from connect() to close()
a different session key is used next time.

Two kinds of secret keys:
session keys and **long-term** keys.
Session keys are used for a short time
e.g., a random session key is used to encrypt
network traffic when for a single socket
from connect() to close()
a different session key is used next time.
e.g., each downloading of same webpage
has a different session key.

Long-term keys don't change, used a long time

Long-term keys don't change, used a long time
used for many sessions.

Long-term keys don't change, used a long time
used for many sessions.
session keys are ephemeral

Long-term keys don't change, used a long time
used for many sessions.
session keys are ephemeral
long-term keys must be managed more carefully

If my session key is compromised
attacker can learn what happened in this session

If my session key is compromised
attacker can learn what happened in this session
not all previous sessions that happened.

Compromising a long-term key may reveal
all previous session keys and allow
for unlimited future impersonation

Compromising a long-term key may reveal
all previous session keys and allow
for unlimited future impersonation
compromised session key can only allow
continued impersonation for a session.

Perfect Forward Secrecy:
the compromise of a long-term secret does
not impact the secrecy of previous (finished) sessions.

Perfect Forward Secrecy:
the compromise of a long-term secret does
not impact the secrecy of previous (finished) sessions.
e.g., by using Diffie-Hellman key exchange.

A

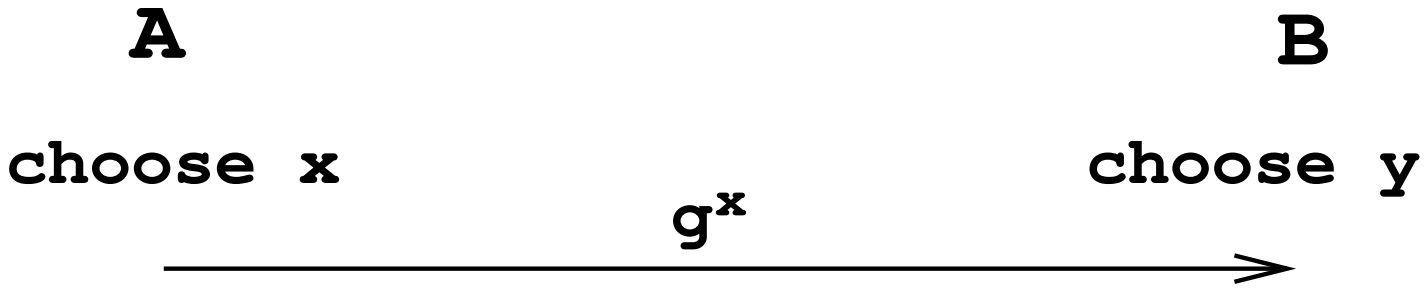
B

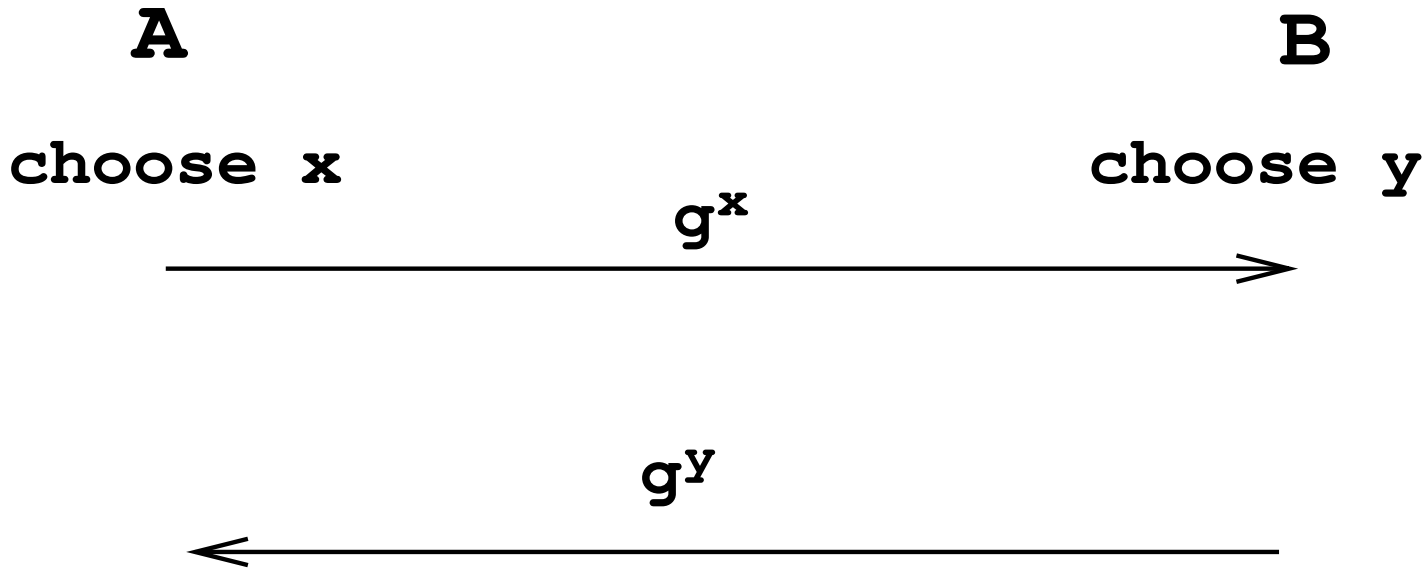
A

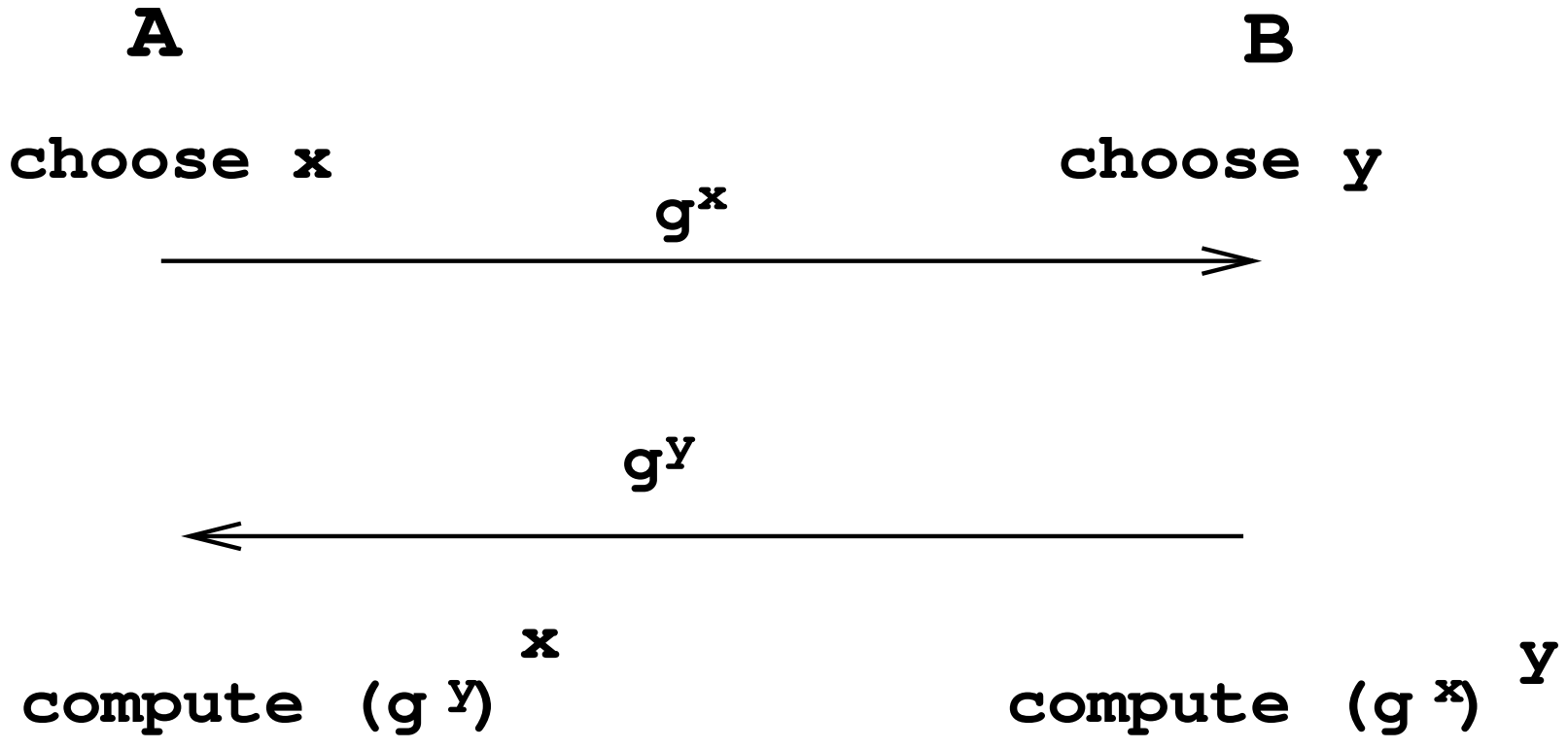
```
choose x
```

B

```
choose y
```

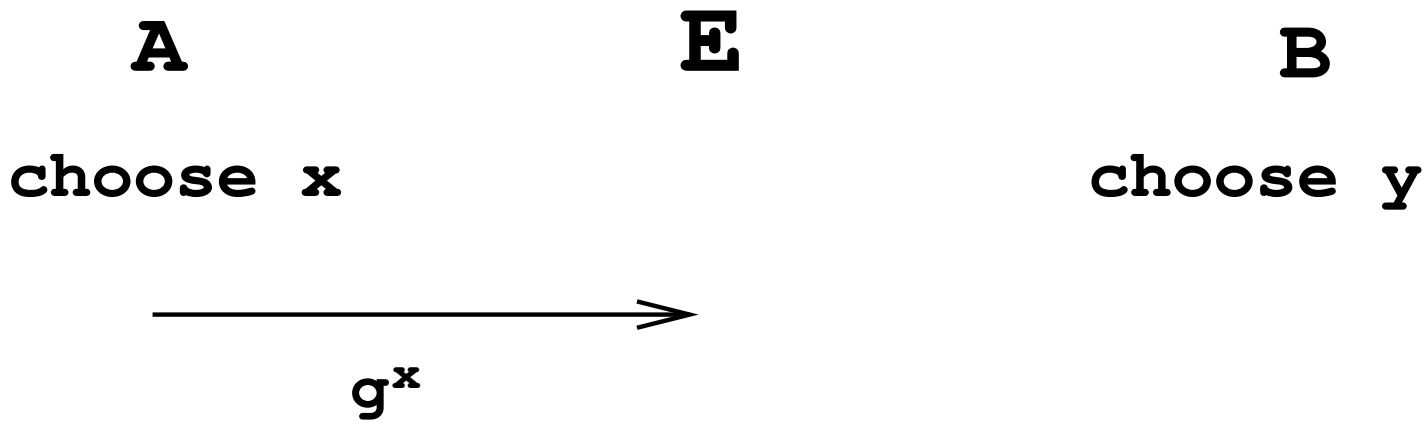

A
B
choose x
choose y
g^x

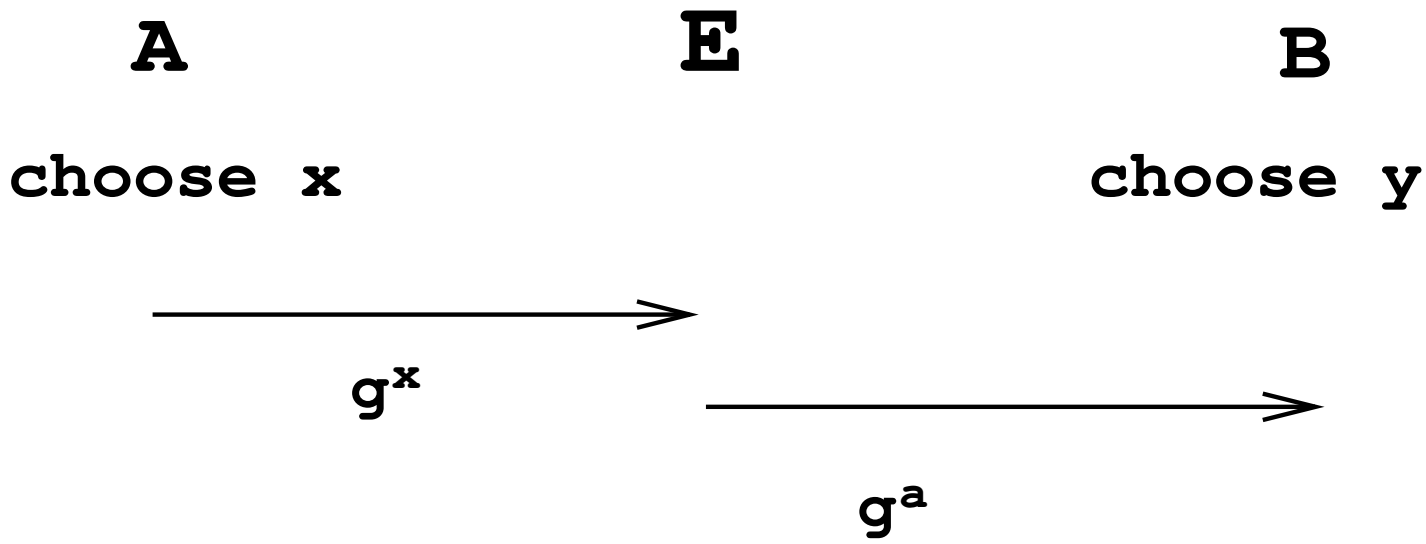

A
B
choose x
choose y
g^x
g^y

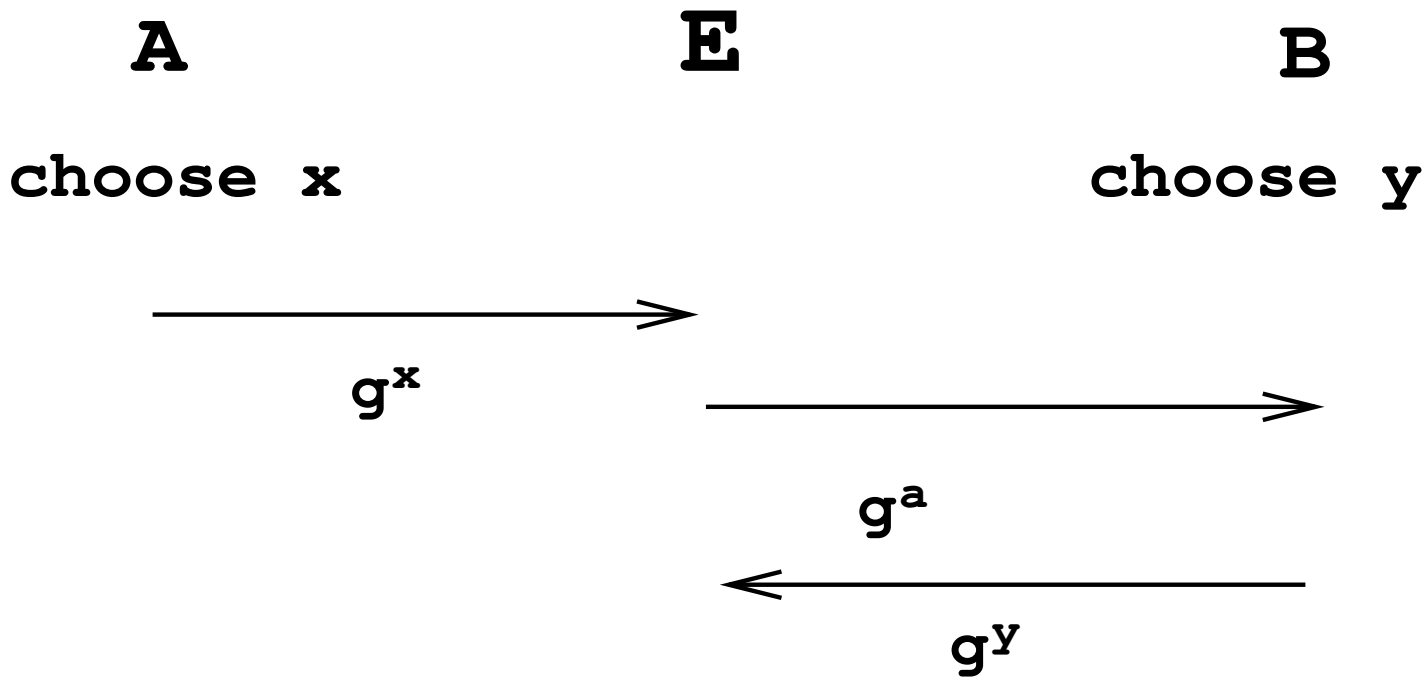

A
B
choose x
choose y
g^x
g^y
compute $(g^y)^x$
compute $(g^x)^y$

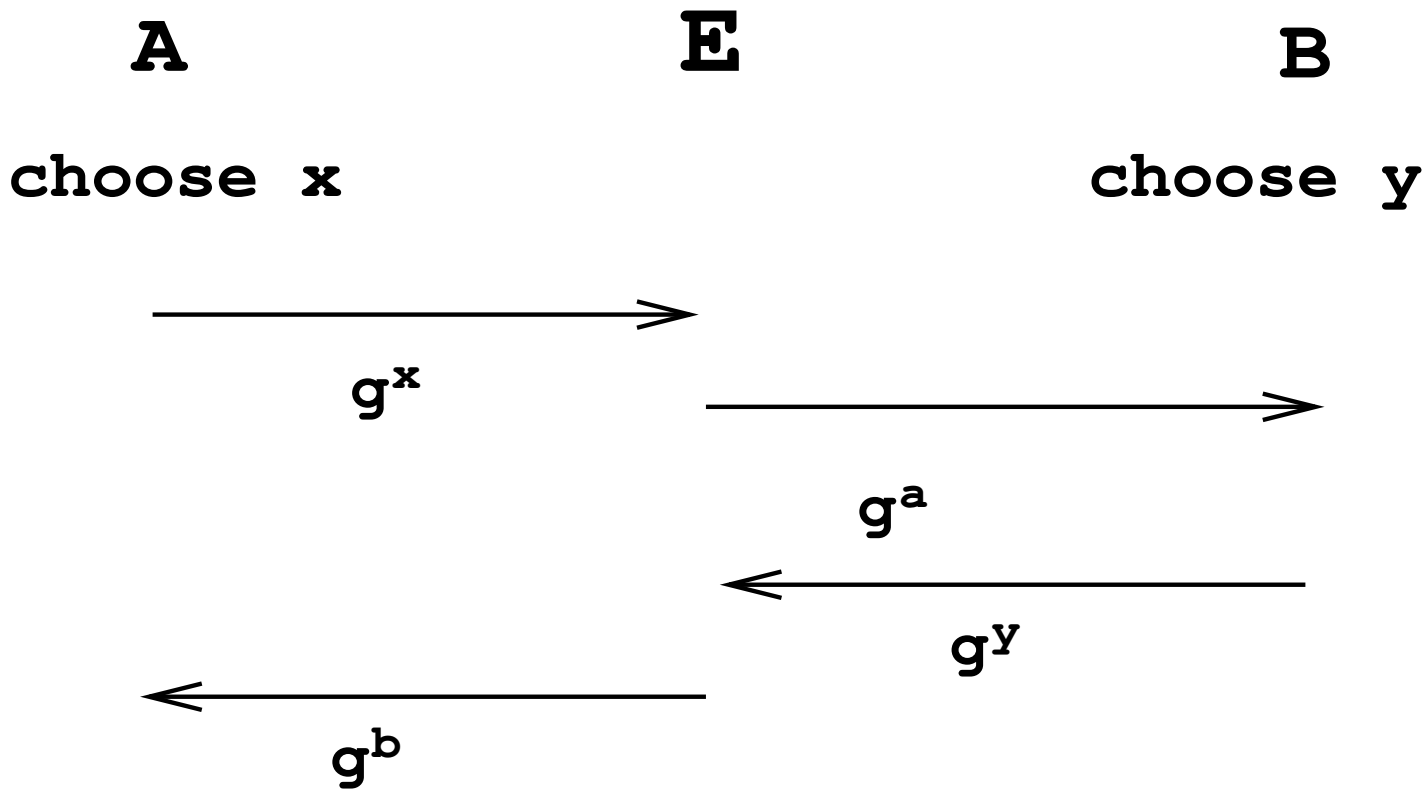
A **E** **B**

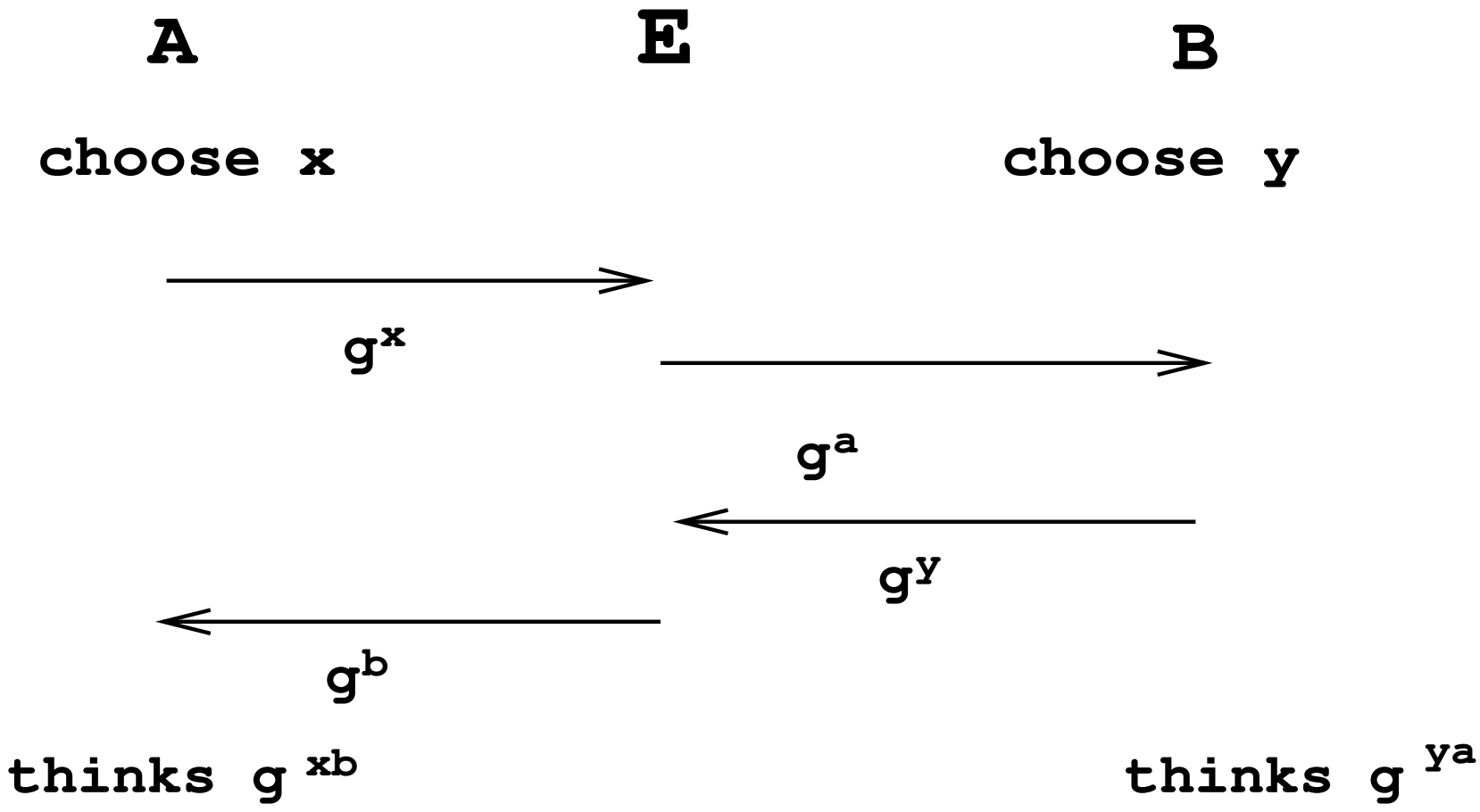
choose x **choose y**

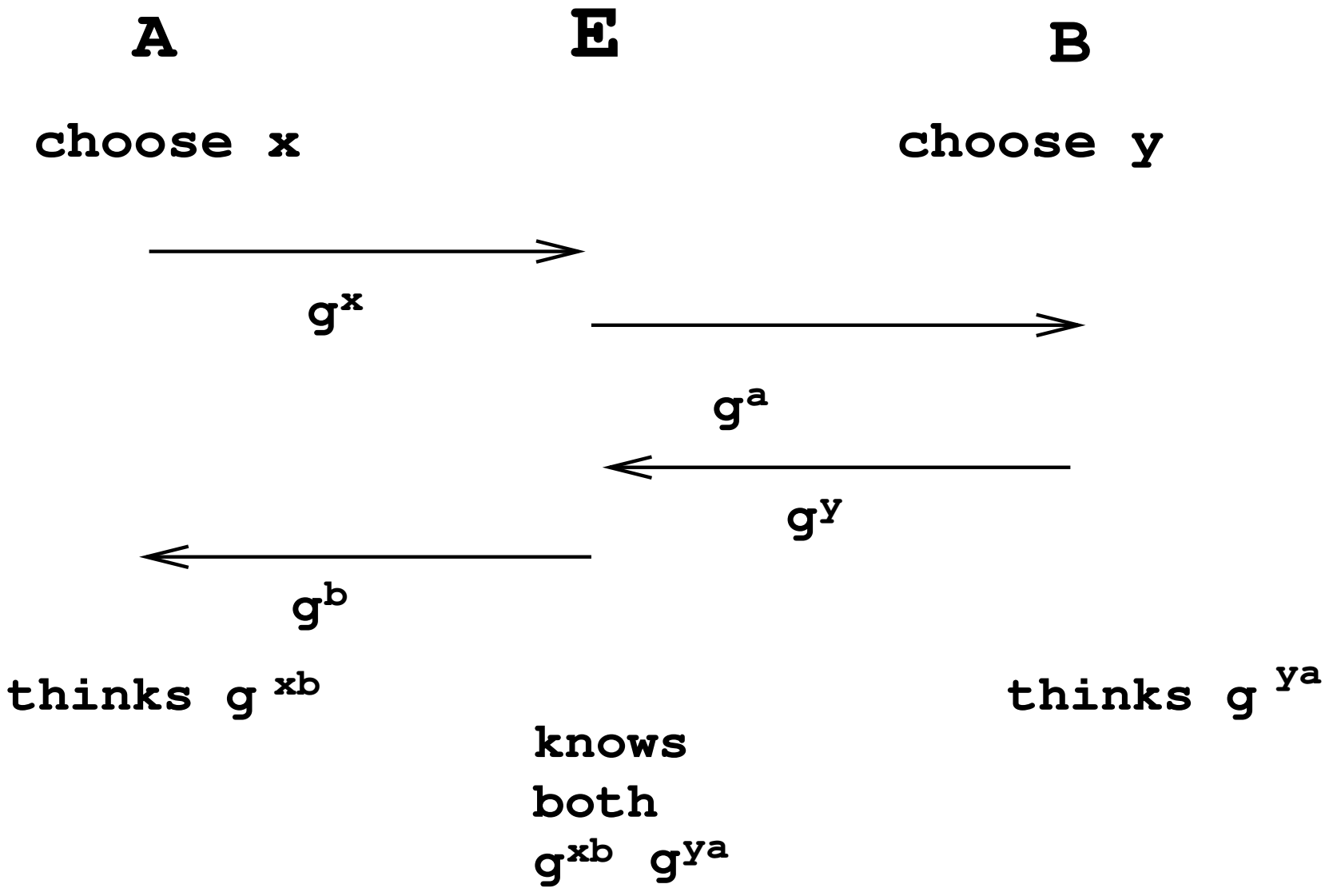

A
E
B
choose x
choose y
g^x

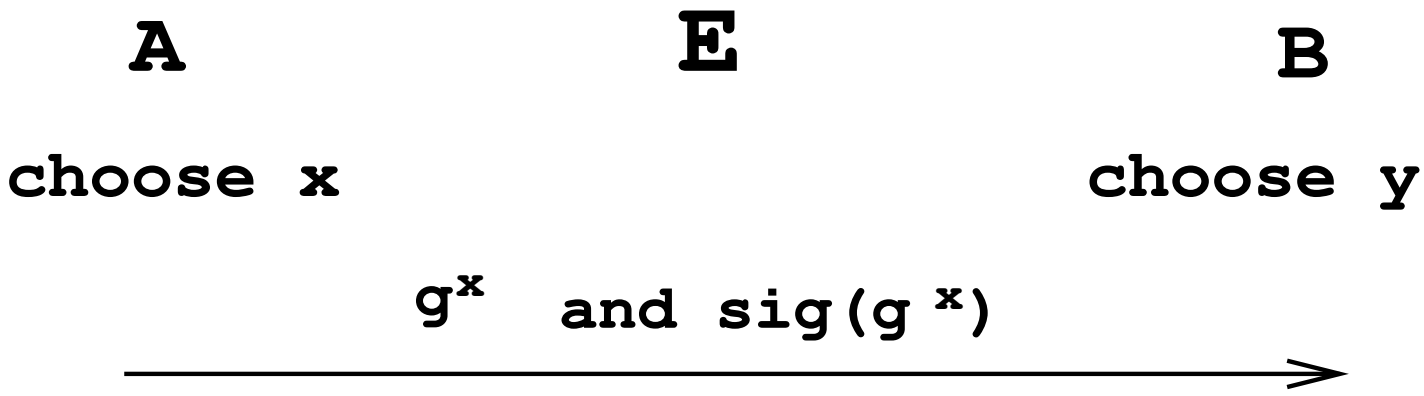

A
E
B
choose x
choose y
g^x
g^a

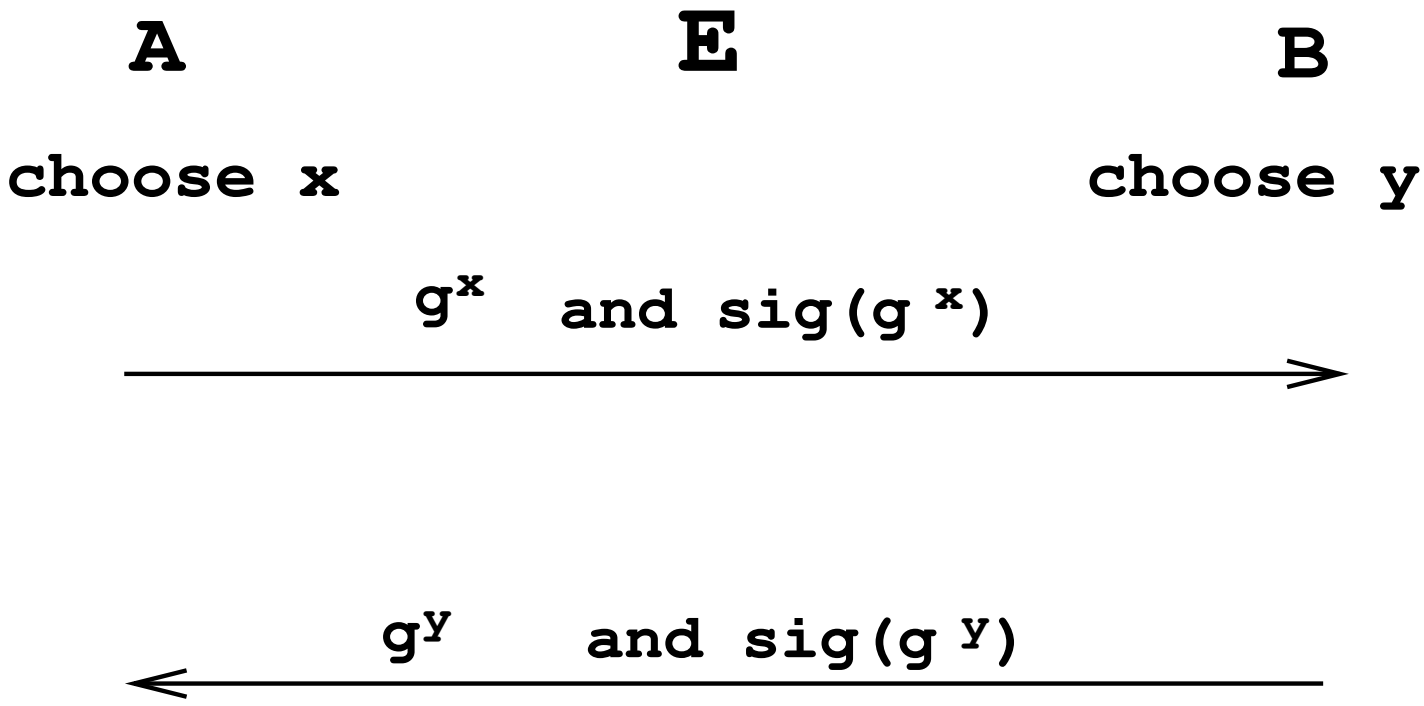

A
E
B
choose x
choose y
g^x
g^a
g^y

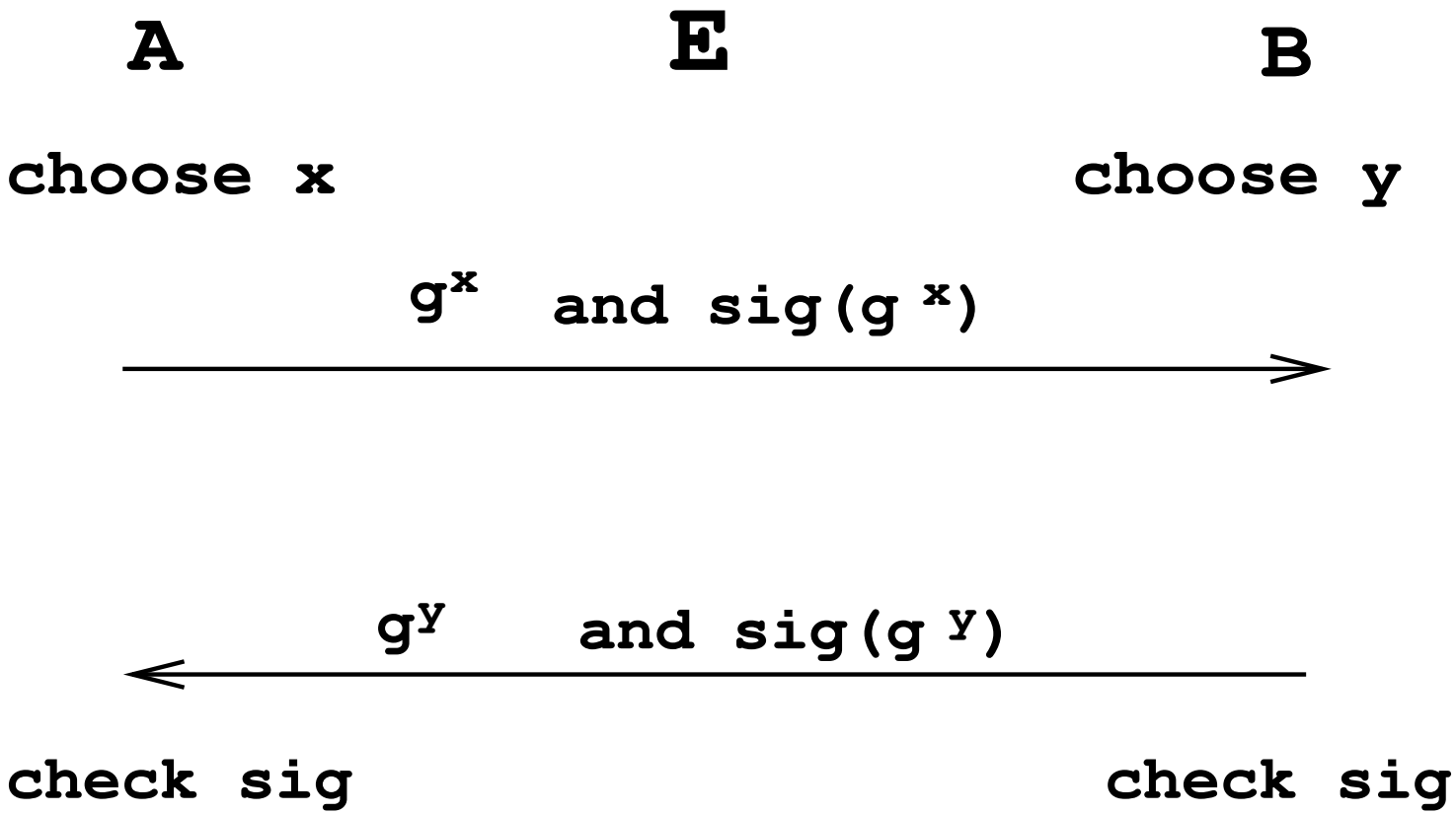

A
E
B
choose x
choose y
g^x
g^a
g^y
g^b


A
E
B
choose x
choose y
g^x
g^a
g^y
g^b
thinks g^{xb}
thinks g^{ya}


A
E
B
choose x
choose y
g^x
g^a
g^y
g^b
thinks g^{xb}
thinks g^{ya}
knows
both
g^{xb} g^{ya}


A
E
B
choose x
choose y
g^x and sig(g^x)


A
E
B
choose x
choose y
g^x and sig(g^x)
g^y and sig(g^y)


A
E
B
choose x
choose y
g^x and sig(g^x)
g^y and sig(g^y)
check sig
check sig

A **E** **B**

choose x **choose y**

replay attack

A

choose x

E

somehow knows a and $sig_A(g^a)$

B

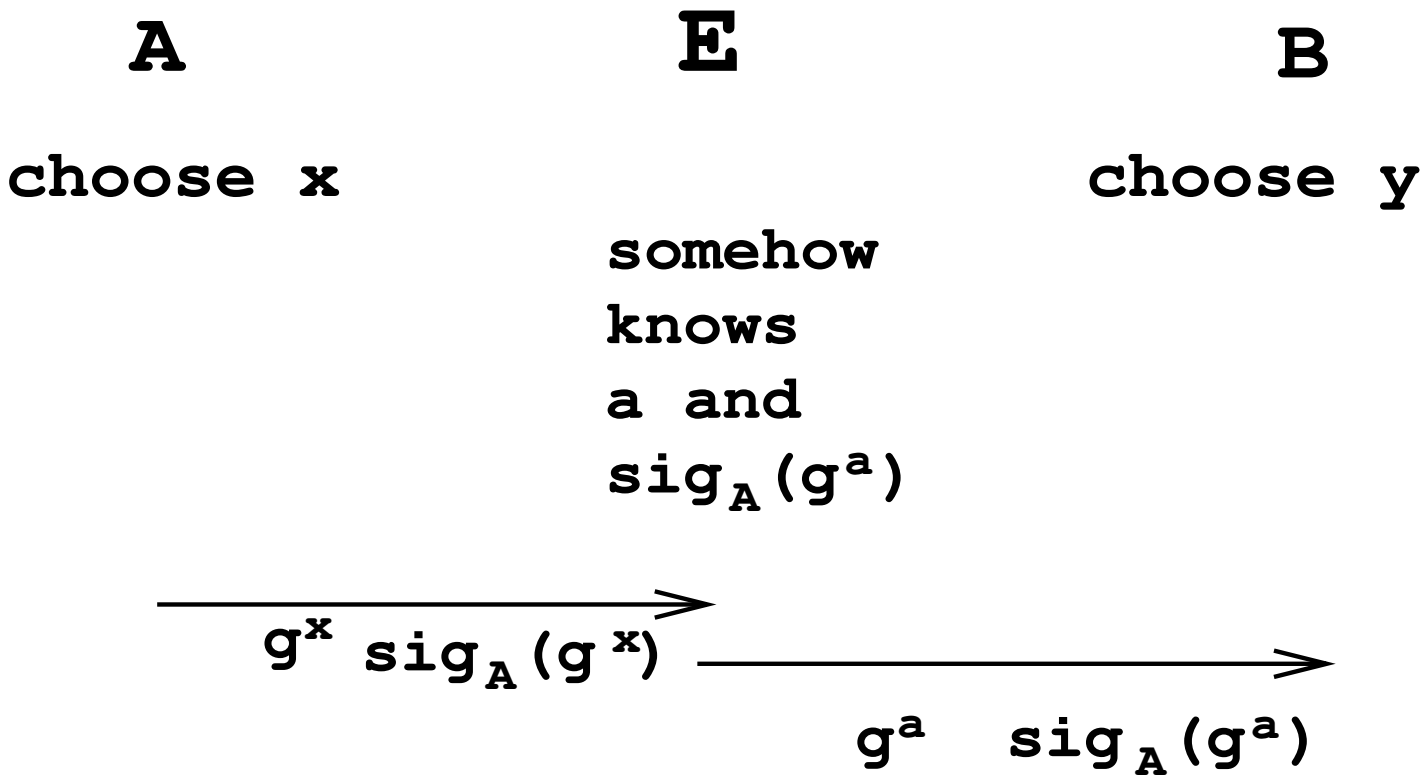
choose y

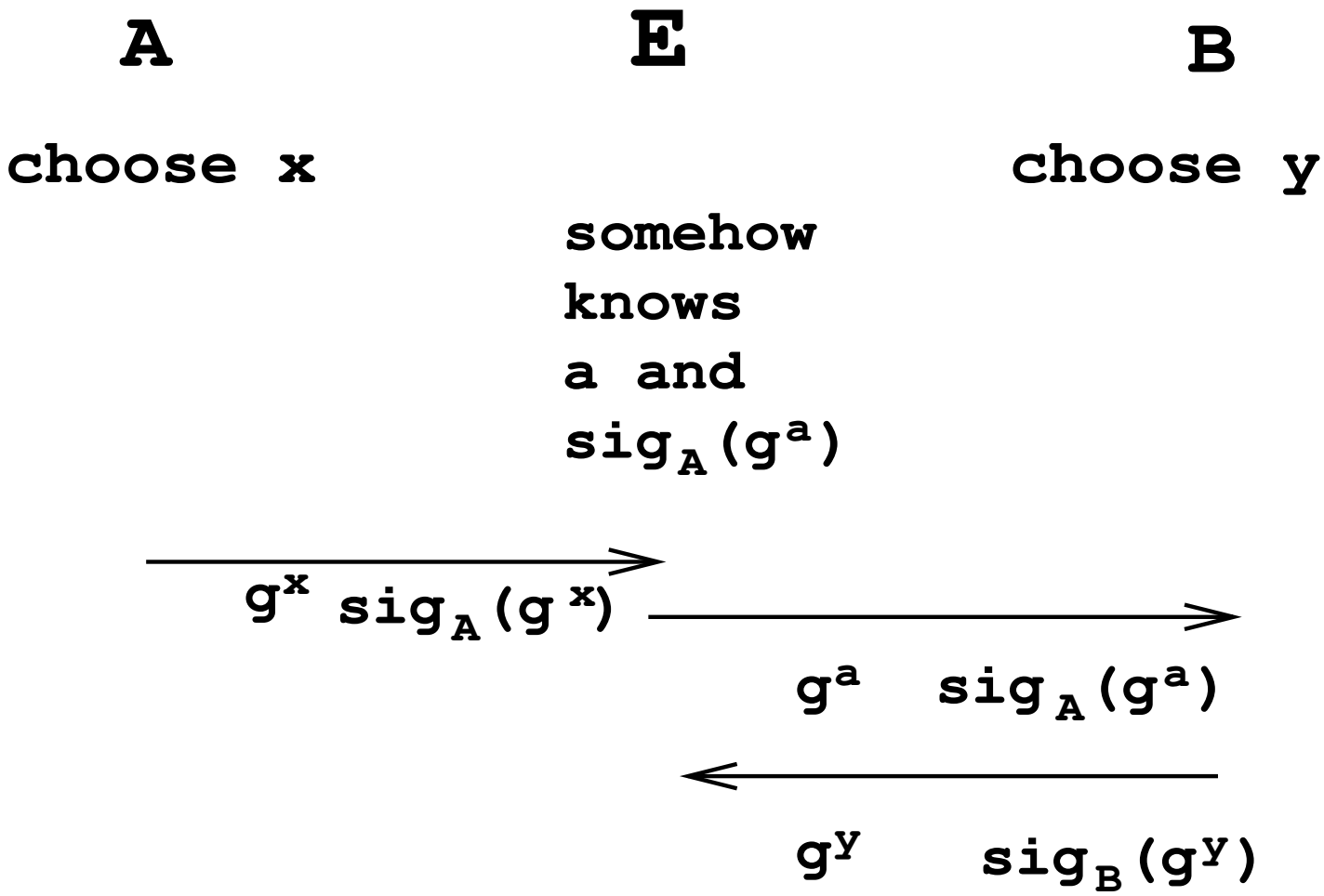
A **E** **B**

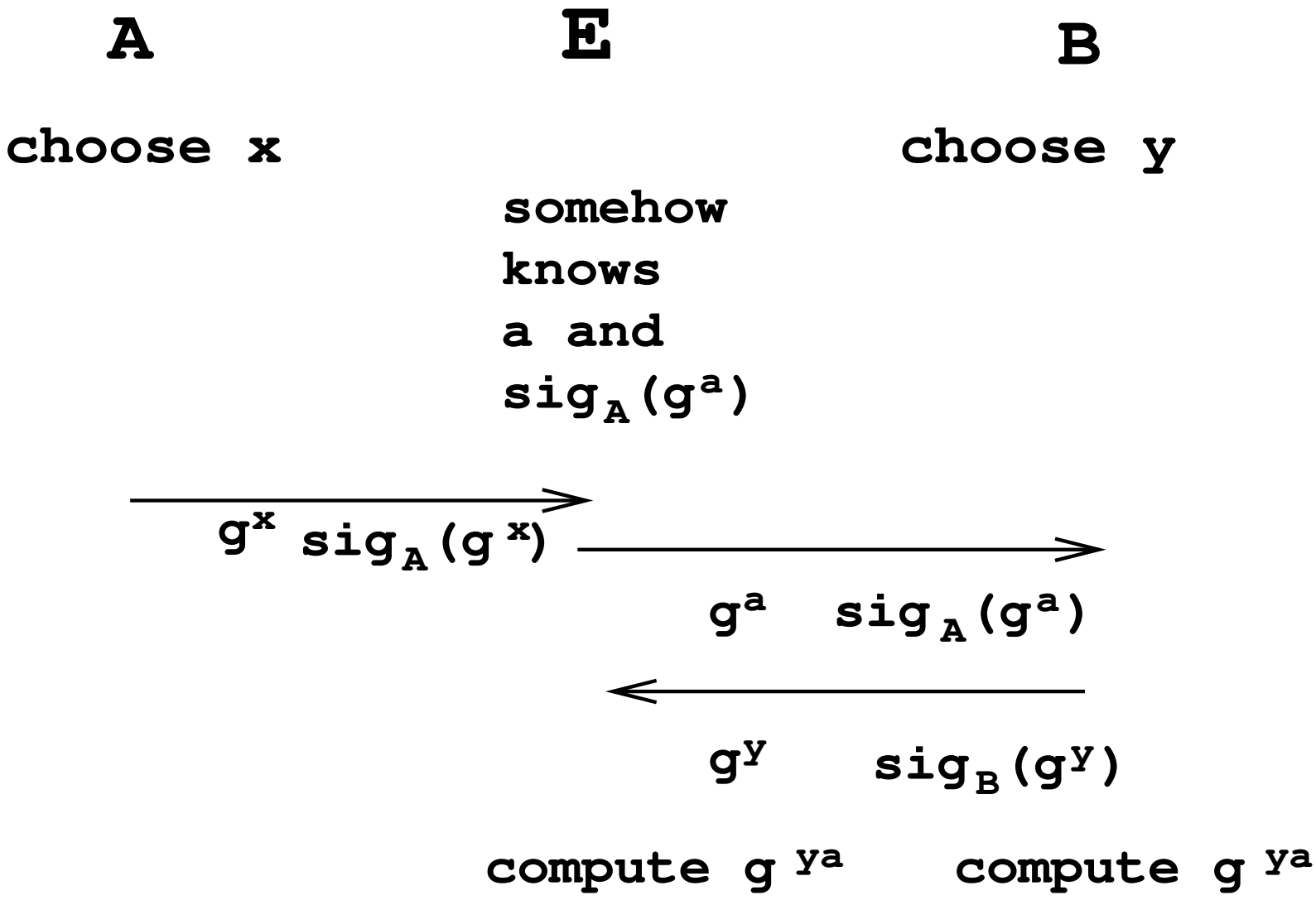
choose x **choose y**

somehow knows a and $sig_A(g^a)$

g^x $sig_A(g^x)$


A
E
B
choose x
choose y
somehow
knows
a and
$sig_A(g^a)$
g^x $sig_A(g^x)$
g^a $sig_A(g^a)$


A
E
B
choose x
choose y
somehow
knows
a and
$sig_A(g^a)$
g^x $sig_A(g^x)$
g^a $sig_A(g^a)$
g^y $sig_B(g^y)$


A
E
B
choose x
choose y
somehow
knows
a and
sig_A(g^a)
g^x sig_A(g^x)
g^a sig_A(g^a)
g^y sig_B(g^y)
compute g^ya
compute g^ya

Kerberos and Mediated Key Exchange

Key Distribution Centre (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?

KDC Operation (in principle)

- $A \rightarrow$ KDC: "I want to talk to Bob"
- KDC invents a random key K_{AB}
- KDC $\rightarrow A$: $\{$ use K_{AB} for Bob $\}_{K_A}$
- KDC $\rightarrow B$: $\{$ use K_{AB} for Alice $\}_{K_B}$
- what can go wrong?

KDC Operation (in practice)

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

Needham-Schroeder Protocol

- goal is key transport on insecure networks
  - e.g., you print a document at the University of Calgary
- 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
  - $\{data\}_{K_{XY}}$ data is encrypted with K_{XY}
- Protocol:
  - $A \rightarrow S : A, B, N_A$
  - $S \rightarrow A : \{N_A, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
  - $A \rightarrow B : \{K_{AB}, A\}_{K_{BS}}$
  - $B \rightarrow A : \{N_B\}_{K_{AB}}$
  - $A \rightarrow B : \{N_B - 1\}_{K_{AB}}$
- where is the flaw?

One fix

- amend the first two lines to:
  - $A \rightarrow B : A$
  - $B \rightarrow A : \{A, N'_B\}_{K_{BS}}$
  - $A \rightarrow S : \{A, B, N_A, \{A, N'_B\}_{K_{BS}}\}_{K_{AS}}$
  - $S \rightarrow A : \{N_A, K_{AB}, B, \{K_{AB}, A, N'_B\}_{K_{BS}}\}_{K_{AS}}$
- 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})
  - $\{\cdot\}_{PX}$ - encrypted for X
  - $\{\cdot\}_X$ - signed by X
- Protocol:
  - $A \rightarrow S : A, B$
  - $S \rightarrow A : \{K_{PB}, B\}_{K_S}$
  - $A \rightarrow B : \{N_A, A\}_{K_{PB}}$
  - $B \rightarrow S : B, A$
  - $S \rightarrow B : \{K_{PA}, A\}_{K_S}$
  - $B \rightarrow A : \{N_A, N_B\}_{K_{PA}}$
  - $A \rightarrow B : \{N_B\}_{K_{PB}}$
- where is the flaw?

Mafia fraud

- Alice connects to Eve willingly
  - Eve runs some website (like an illegal downloading site)
- Eve then connects to Bob pretending to be Alice
- Bob issues “Alice” (really Eve) a challenge to sign
- Eve uses that as a challenge for Alice pretending it’s Eve’s website

This is also called the chess grandmaster's problem:
how was a young girl named Anne-Louise
able to defeat a grandmaster in chess?

In Mafia fraud Eve is acting as a man-in-the-middle for authentication, and Anne-Louise as a man-in-the-middle for the chess grandmasters.

So why is it called Mafia fraud and not MitM?

Mafia fraud / grandmaster problem the victim
willingly and knowingly communicates
with the attacker and
unknowingly communicates
with the other victim.

Mafia fraud / grandmaster problem the victim
willingly and knowingly communicates
with the attacker and
unknowingly communicates
with the other victim.
MitM involves the victims unknowingly
communicating with the attacker.

N-S Mafia Fraud

- $A \rightarrow S : A, E$
- $S \rightarrow A : \{K_{PE}, E\}_{K_S}$
- $A \rightarrow E : \{N_A, A\}_{K_{PE}}$
- $E \rightarrow B : \{N_A, A\}_{K_{PB}}$
  - E can decrypt this and so know N_A
- $B \rightarrow E : \{N_A, N_B\}_{K_{PA}}$
  - E cannot decrypt this so does not learn N_B
- $E \rightarrow A : \{N_A, N_B\}_{K_{PA}}$
  - E can just relay it verbatim
- $A \rightarrow E : \{N_B\}_{K_{PE}}$
  - E learns N_B by design
- $E \rightarrow B : \{N_B\}_{K_{PB}}$
  - success

There I fixed it (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})
  - $\{\cdot\}_{PX}$ - encrypted for X
  - $\{\cdot\}_X$ - signed by X
- Protocol:
  - $A \rightarrow S : A, B$
  - $S \rightarrow A : \{K_{PB}, B\}_{K_S}$
  - $A \rightarrow B : \{N_A, A\}_{K_{PB}}$
  - $B \rightarrow S : B, A$
  - $S \rightarrow B : \{K_{PA}, A\}_{K_S}$
  - $B \rightarrow A : \{N_A, N_B, \mathbf{B}\}_{K_{PA}}$
  - $A \rightarrow B : \{N_B\}_{K_{PB}}$

Now for Kerberos, based on symmetric N-S.

Many-to-Many Authentication

- how to prove identity when requesting services on network
  - 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

Requirements

- security
  - against attacks by passive eavesdroppers
  - against attacks by actively malicious users
- transparency
  - users shouldn't notice authentication 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 pretends 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 replays other users' conversations to gain unauthorized access

Solution: Trusted Third Party

- user proves identity to trusted third party (TTP), requests a ticket for service
- TTP knows all users and servers, 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

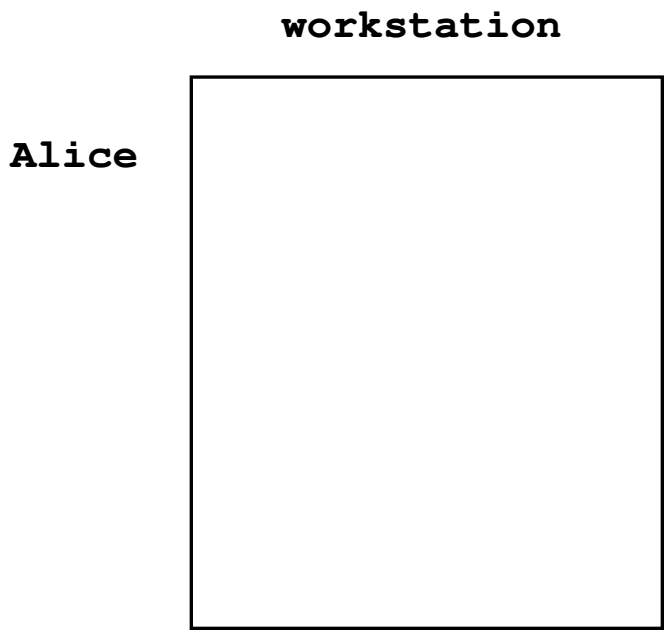
- protocol:
  - user authenticates to the key distribution centre (KDC)
  - gets a special ticket granting service (TGS) ticket
  - user gives TGS ticket to TGS server when needed
  - gets encrypted service ticket (e.g., for printer)
  - user gives ticket to printer

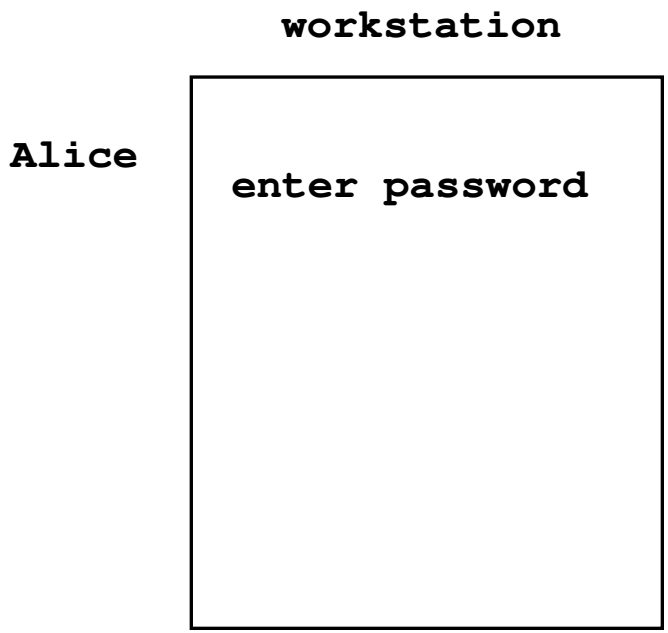
Threats to Two Step

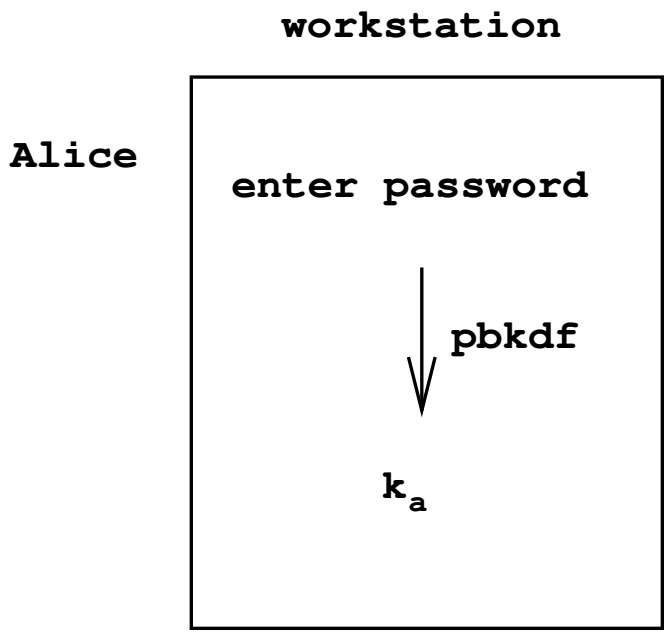
- ticket hijacking
  - malicious user steals service ticket
  - uses it on the same workstation
    - network address verification doesn't help
  - server must verify that the user who gives the ticket is the same who was issued
- no server authentication
  - attacker may misconfigure the network so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

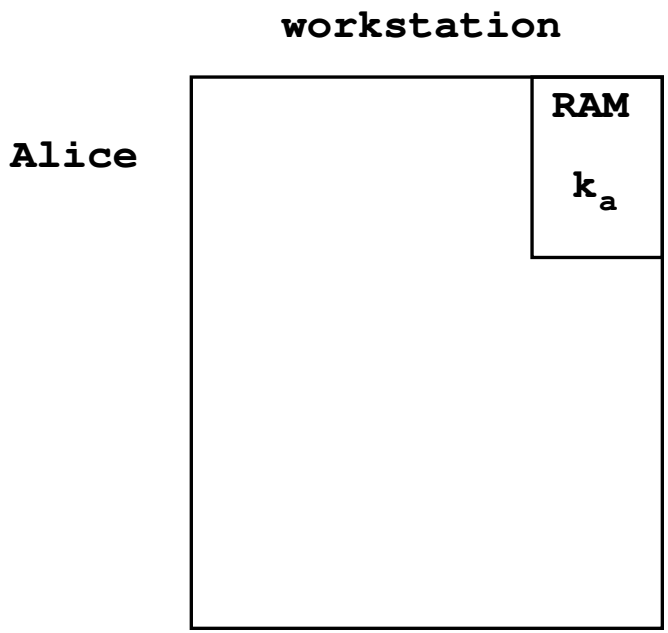
Kerberos

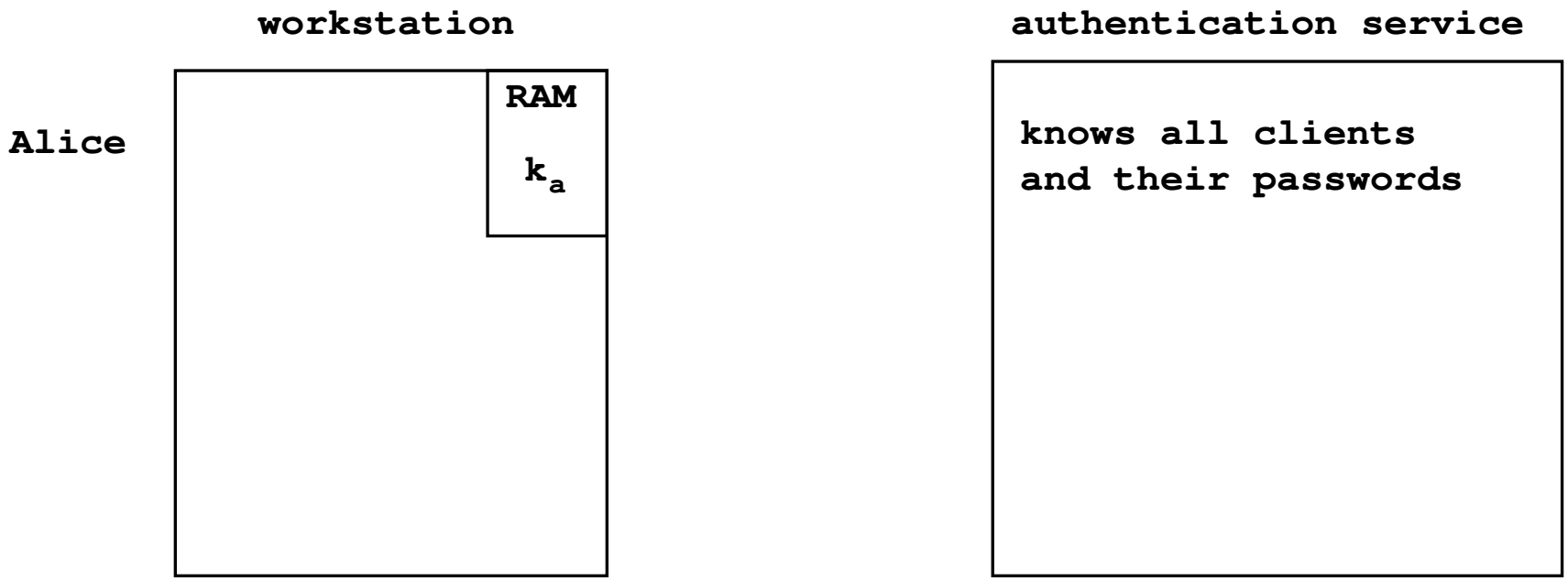
- K_C is a long-term key of client C
  - derived from the user's password
- K_{TGS} is a long-term key of the TGS
  - known by KDC and TGS
- K_V is a long-term key of network service V
  - known to V and TGS; each V has its own key
- $K_{C,TGS}$ is a short-term session key b/w C and TGS
  - created by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

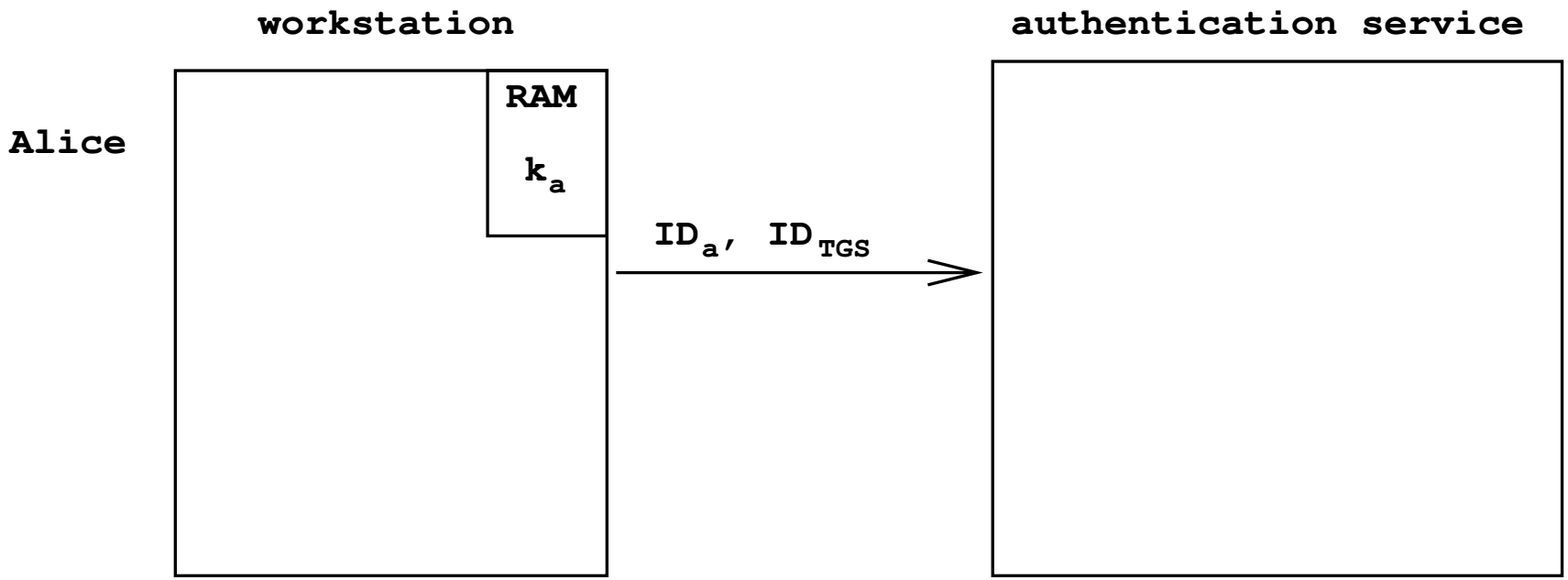

workstation
Alice

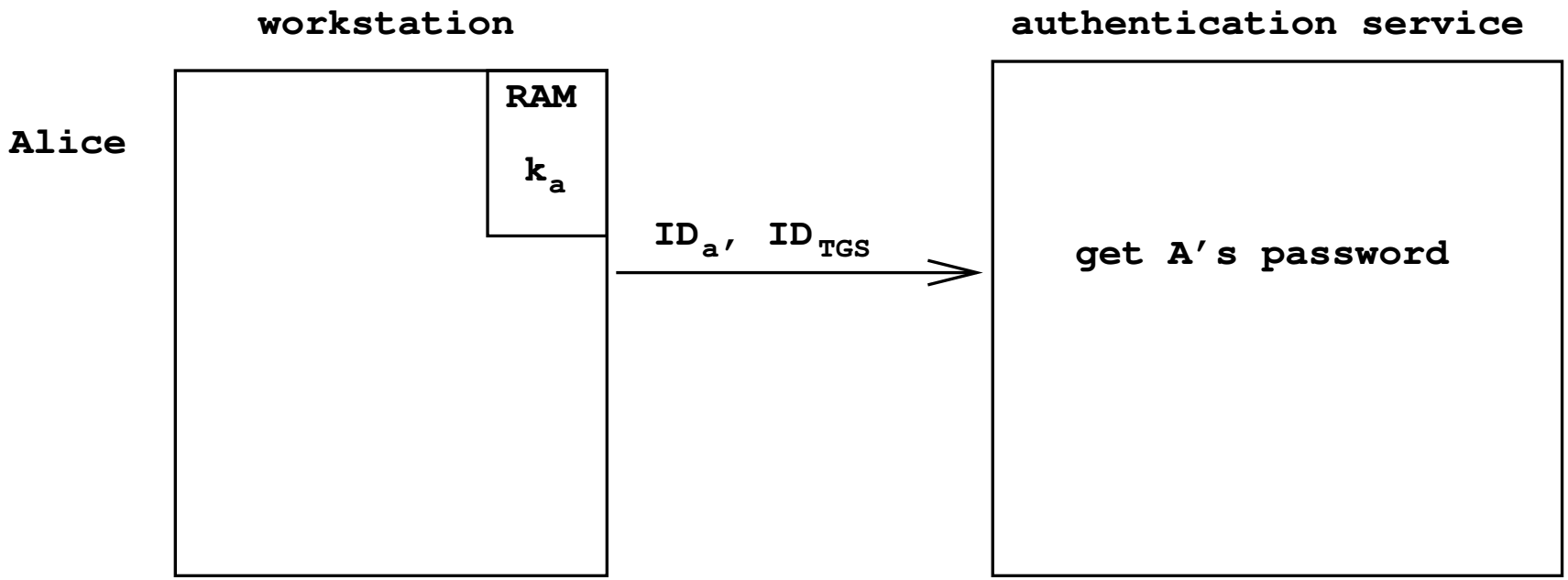

workstation
Alice
enter password

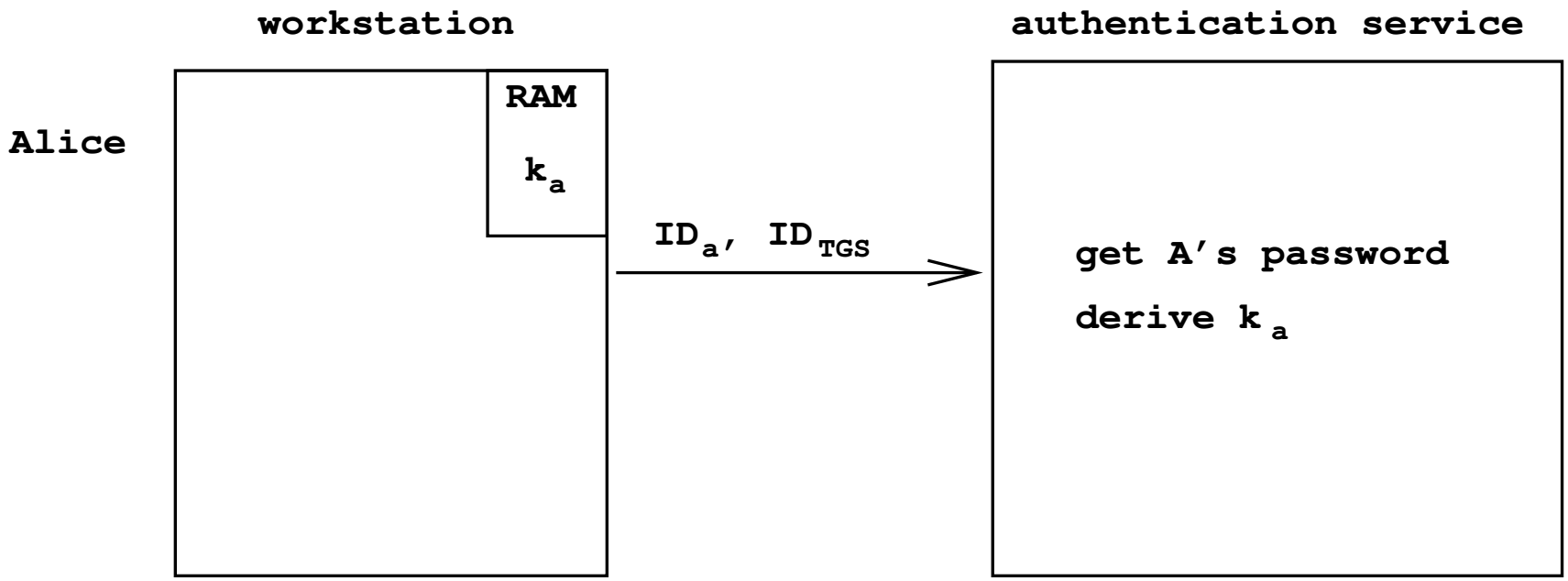

workstation
Alice
enter password
pbkdf
k_a

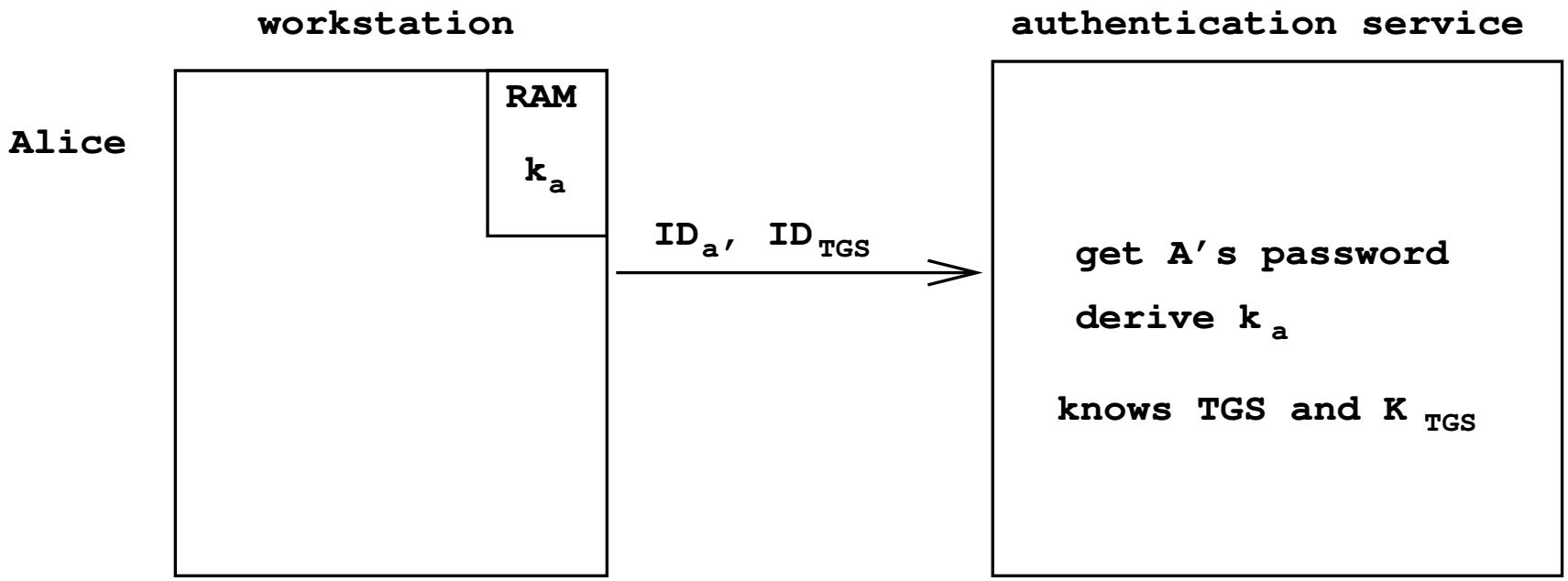

workstation
RAM
k_a
Alice

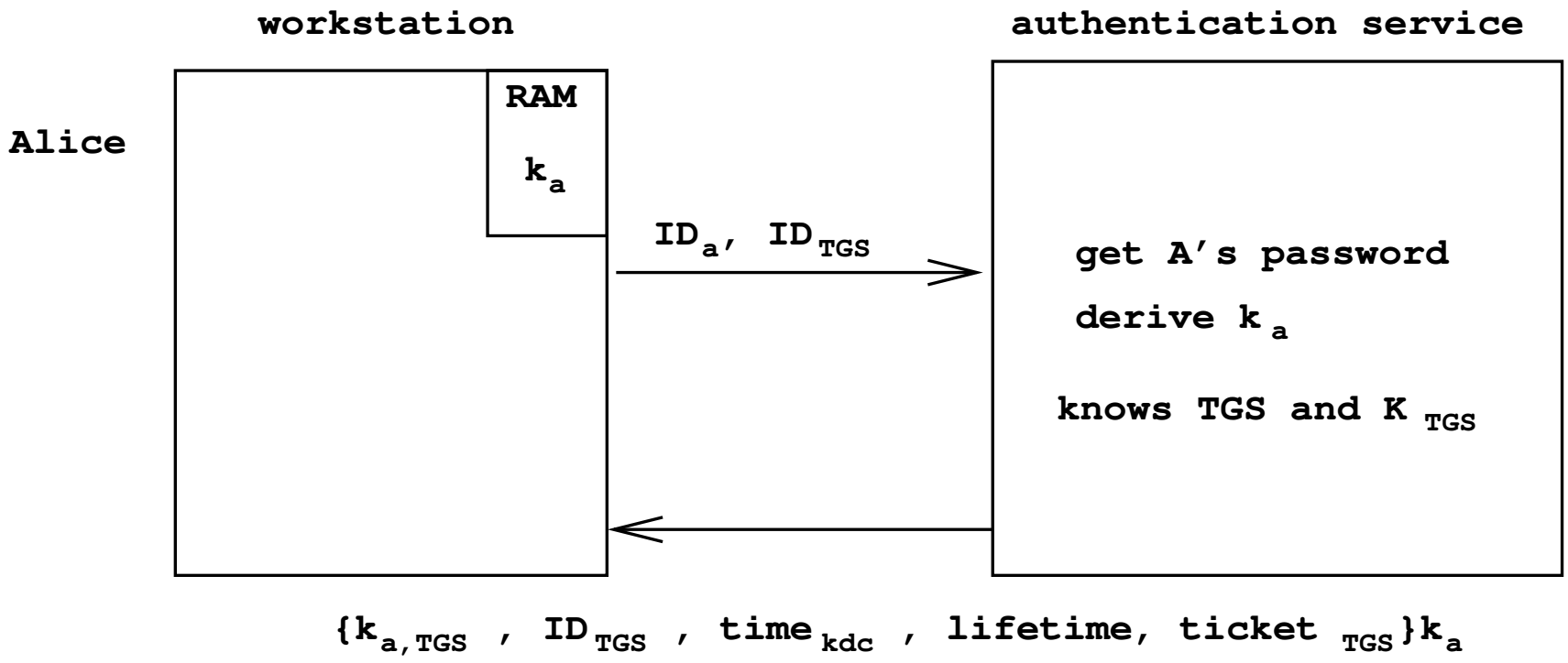

workstation
authentication service
Alice
RAM
k_a
knows all clients
and their passwords

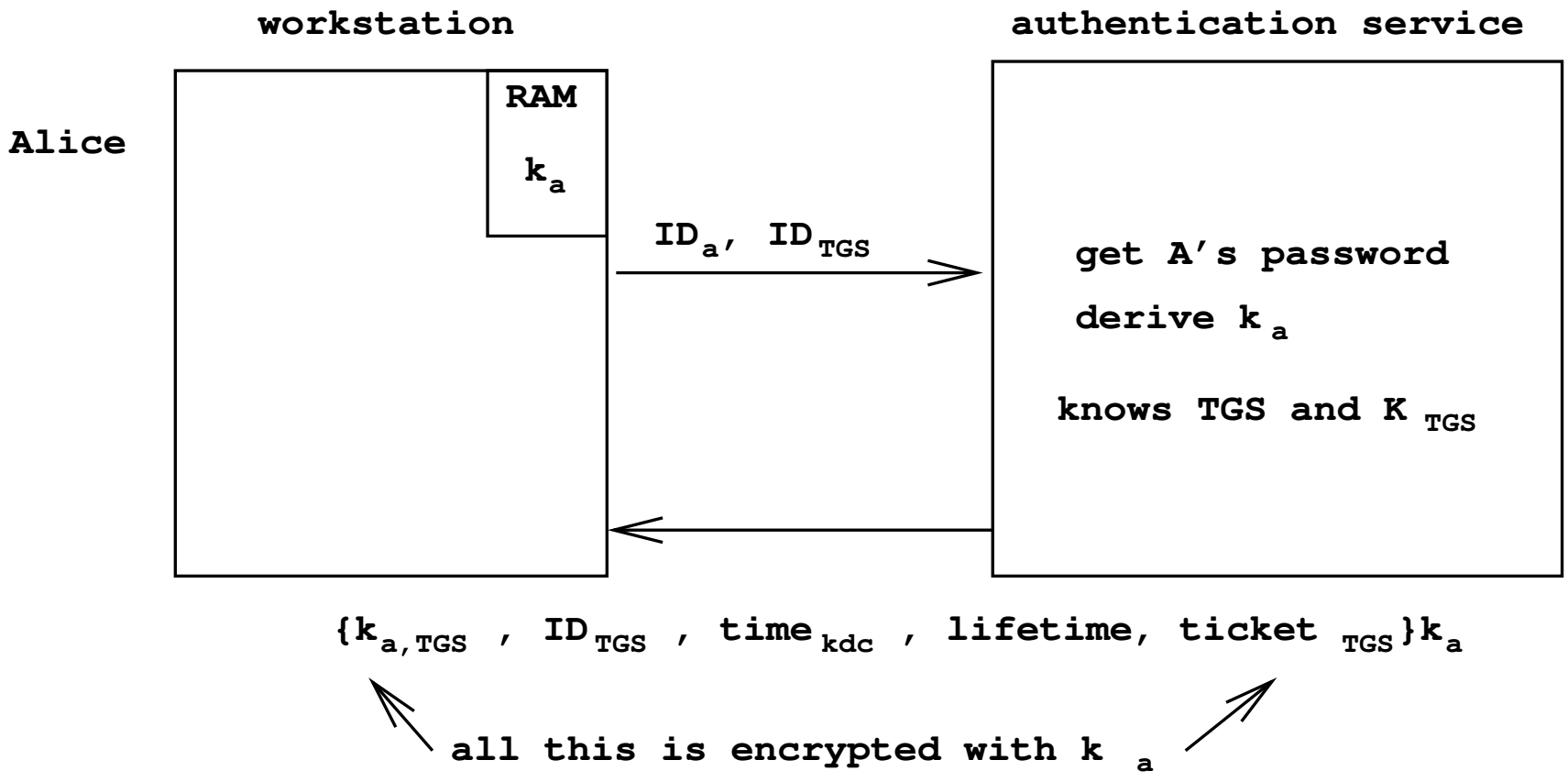

workstation
authentication service
Alice
RAM
k_a
ID_a, ID_{TGS}

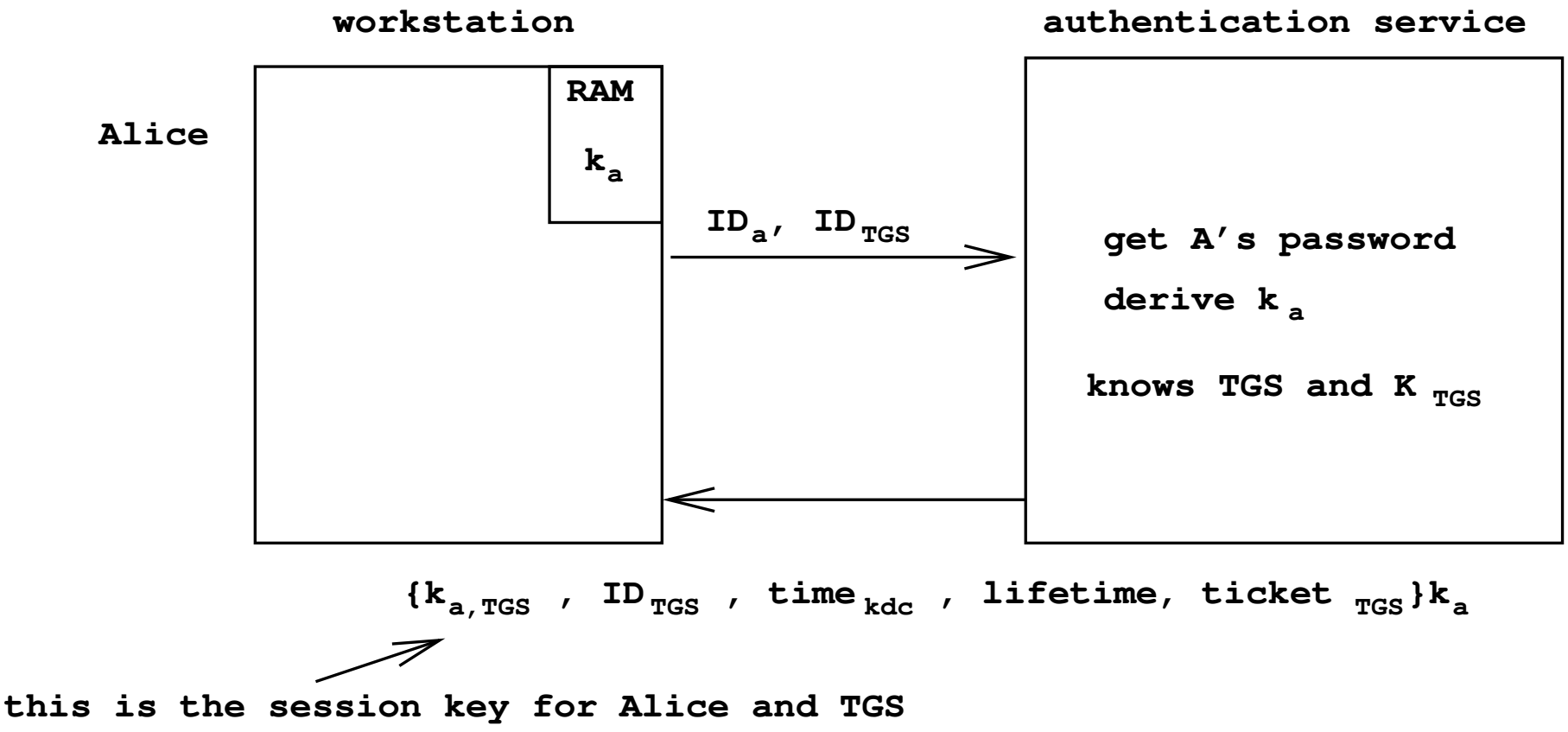

workstation
authentication service
Alice
RAM
k_a
ID_a, ID_{TGS}
get A's password

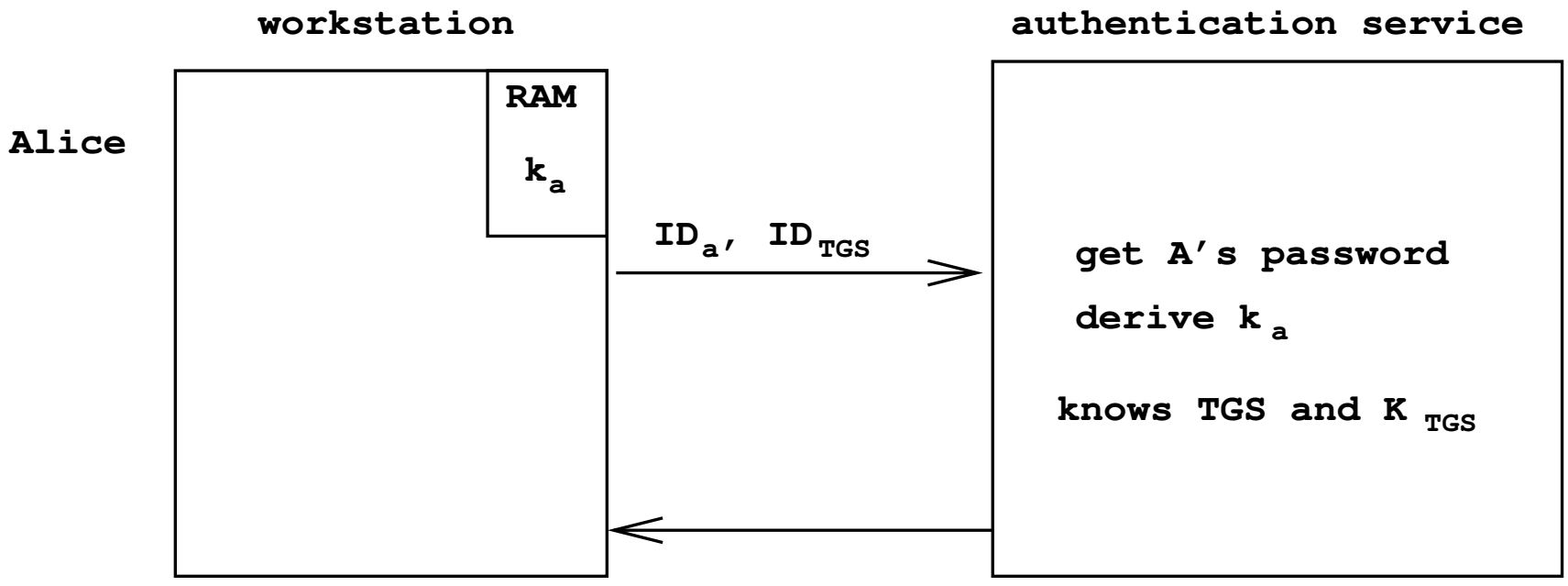

workstation
authentication service
Alice
RAM
k_a
ID_a, ID_{TGS}
get A's password
derive k_a


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

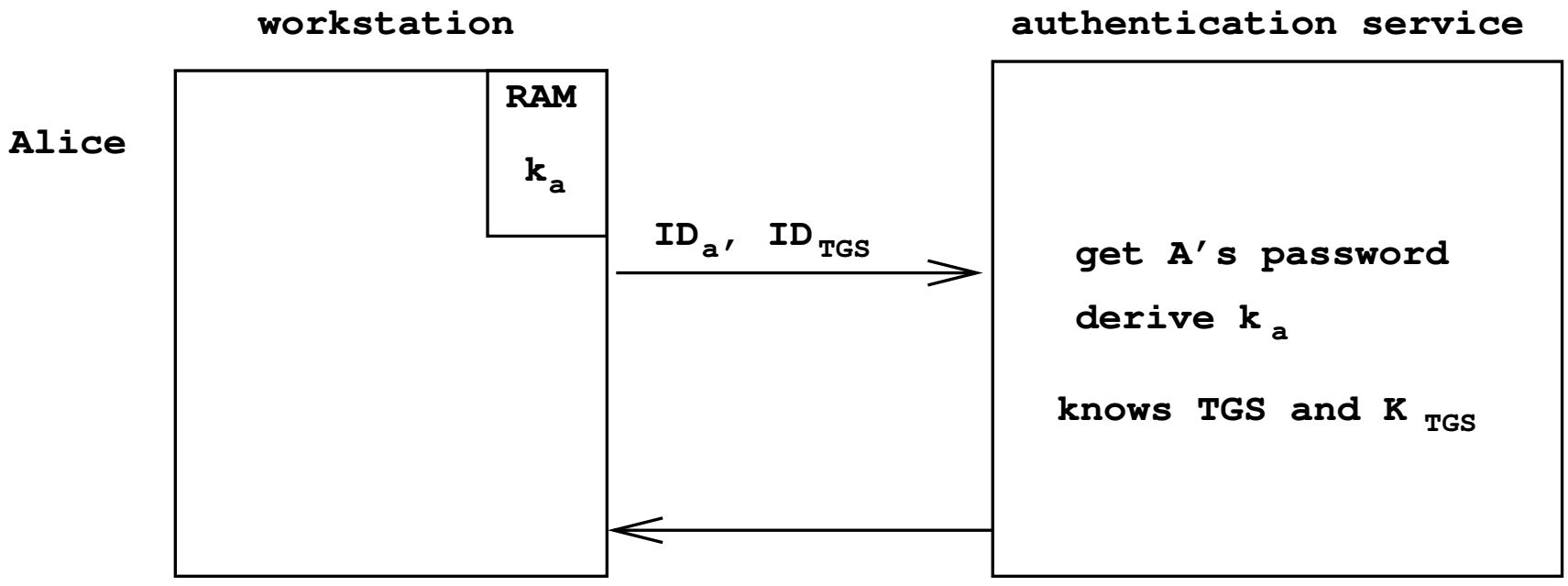

workstation
authentication service
Alice
RAM
k_a
ID_a, ID_{TGS}
get A's password
derive k_a
knows TGS and K_{TGS}
$\{k_{a,TGS}$, ID_{TGS} , $time_{kdc}$, lifetime, $ticket_{TGS}\}k_a$


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

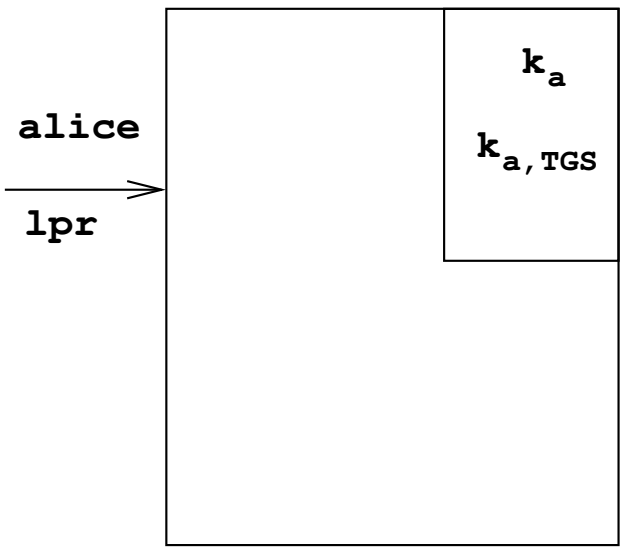

workstation
authentication service
Alice
RAM
k_a
ID_a, ID_{TGS}
get A's password
derive k_a
knows TGS and K_{TGS}
$\{k_{a,TGS}$, ID_{TGS} , $time_{kdc}$, lifetime, $ticket_{TGS}\}k_a$
this is the session key for Alice and TGS

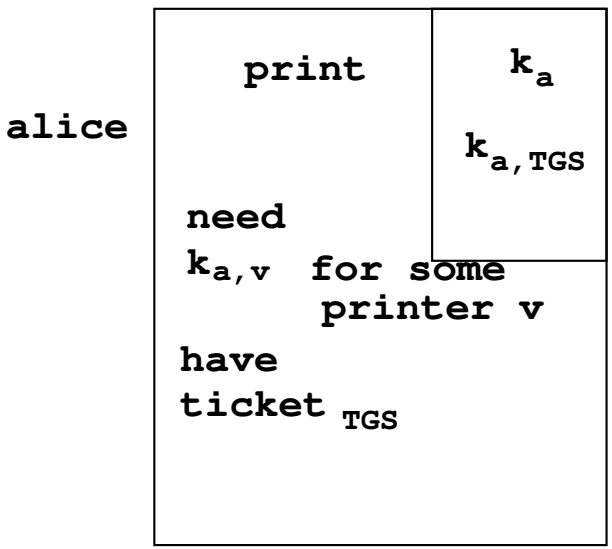


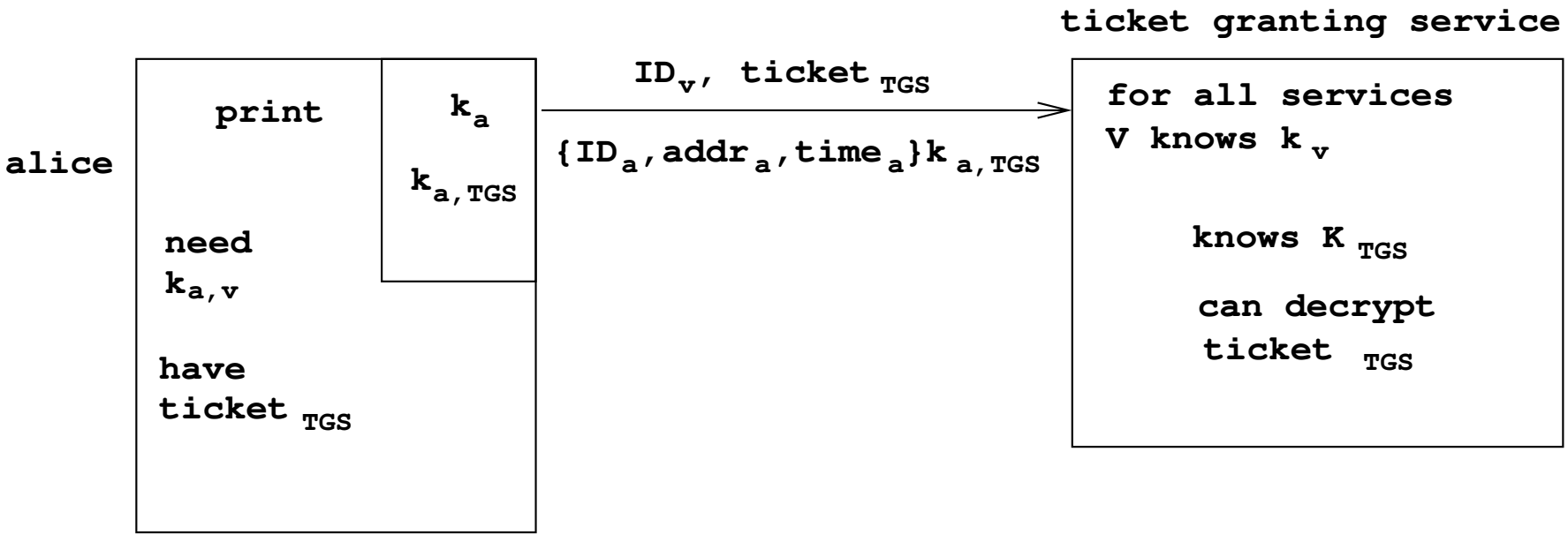
$$ticket_{TGS}=\{k_{A,TGS}, ID_A, time_{KCD}, lifetime, ID_{TGS}, addr_A\}k_{TGS}$$

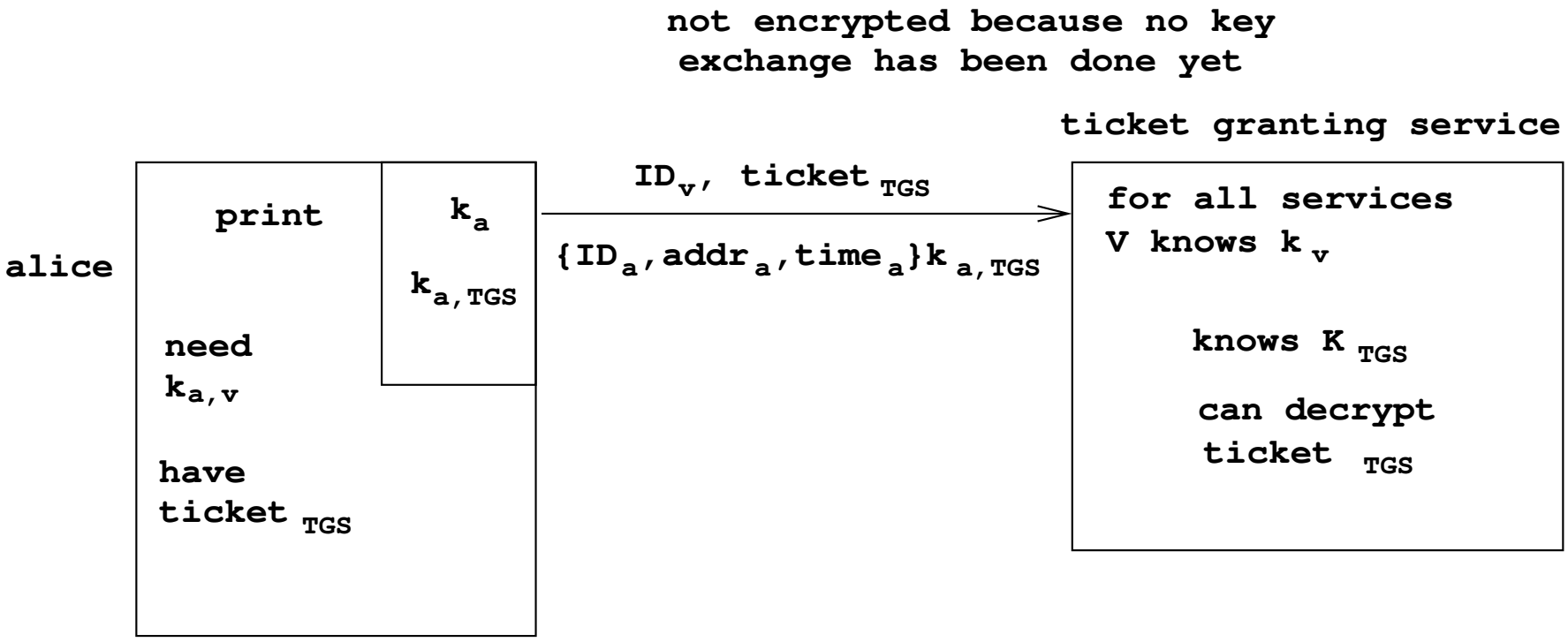

workstation
authentication service
Alice
RAM
k_a
ID_a, ID_{TGS}
get A's password
derive k_a
knows TGS and K_{TGS}
$\{k_{a,TGS}, ID_{TGS}, time_{kdc}, lifetime, ticket_{TGS}\}k_a$
$ticket_{TGS}=\{k_{A,TGS}, ID_A, time_{KCD}, lifetime, ID_{TGS}, addr_A\}k_{TGS}$
Alice can't understand this, but is expected to deliver it to TGS

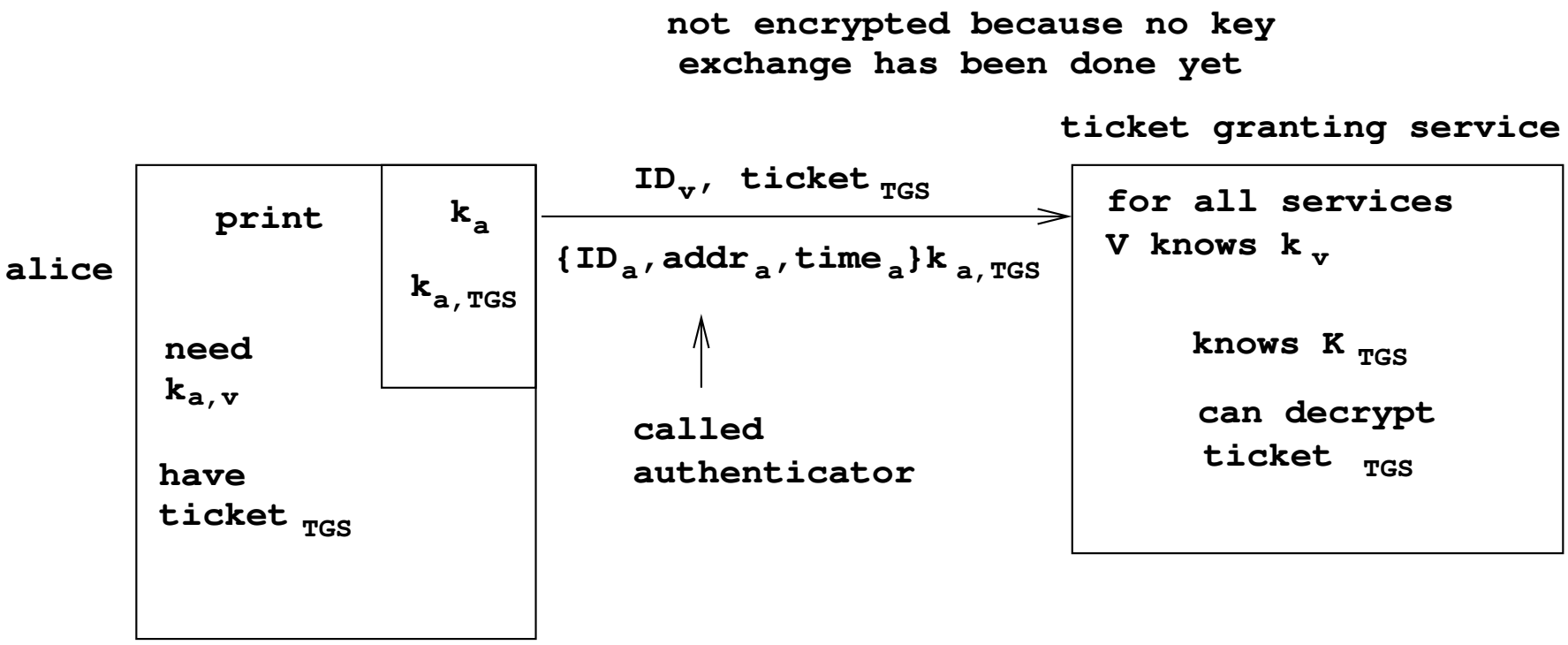
alice
wants
to
print

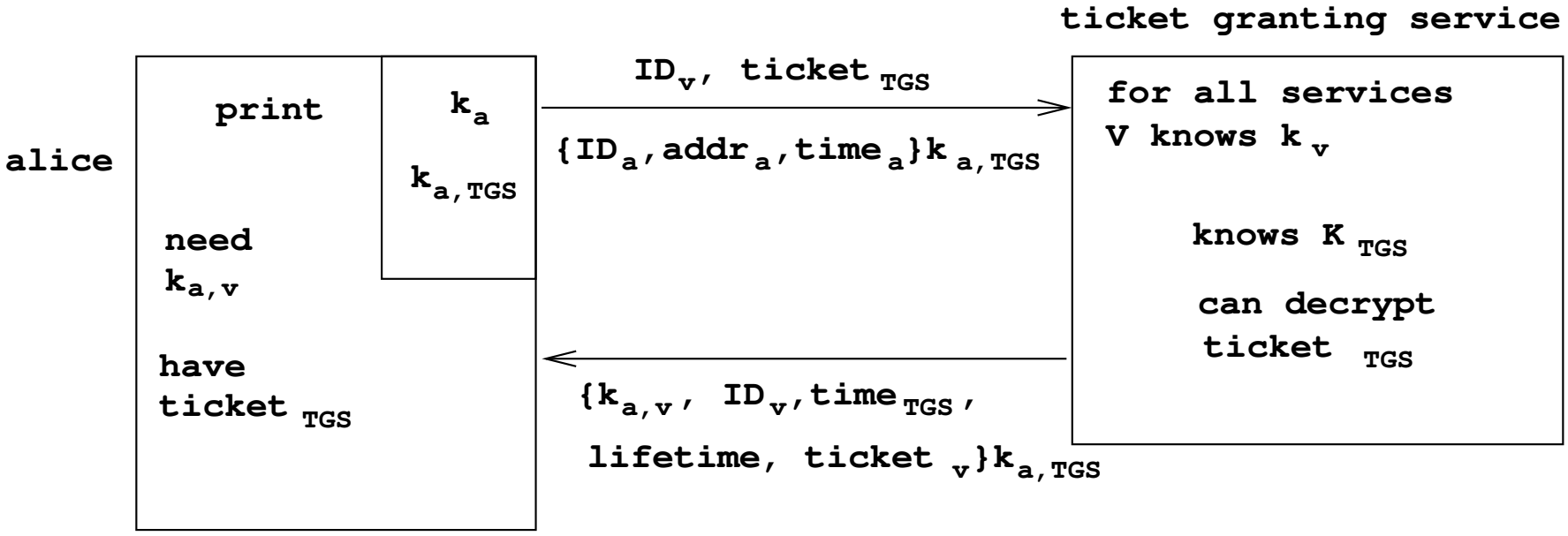

alice
lpr
k_a
$k_{a,TGS}$

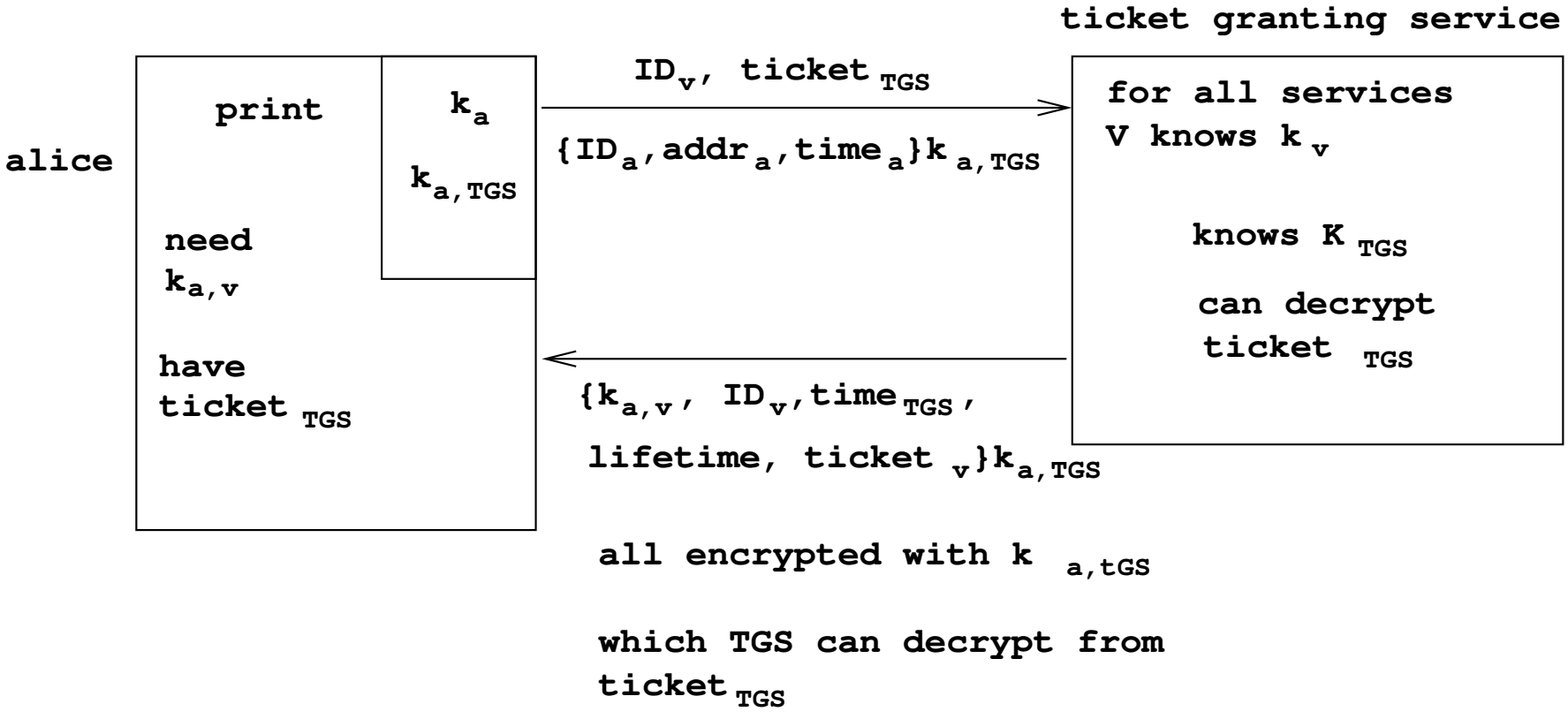

alice
print
k_a
$k_{a,TGS}$
need
$k_{a,v}$ for some
printer v
have
ticket $_{TGS}$

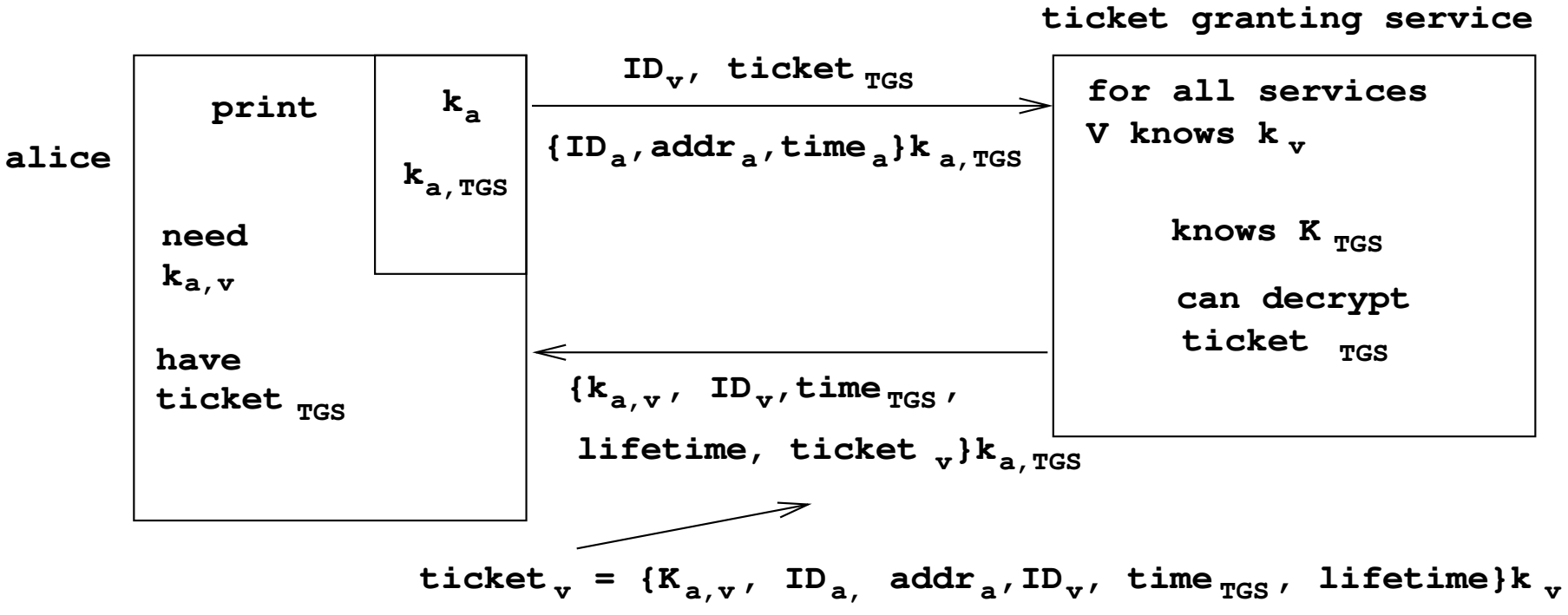

ticket granting service
alice
print
need
$k_{a,v}$
have
ticket $_{TGS}$
k_a
$k_{a,TGS}$
ID_v, ticket $_{TGS}$
$\{ID_a, addr_a, time_a\}k_{a,TGS}$
for all services
V knows k_v
knows K_{TGS}
can decrypt
ticket $_{TGS}$

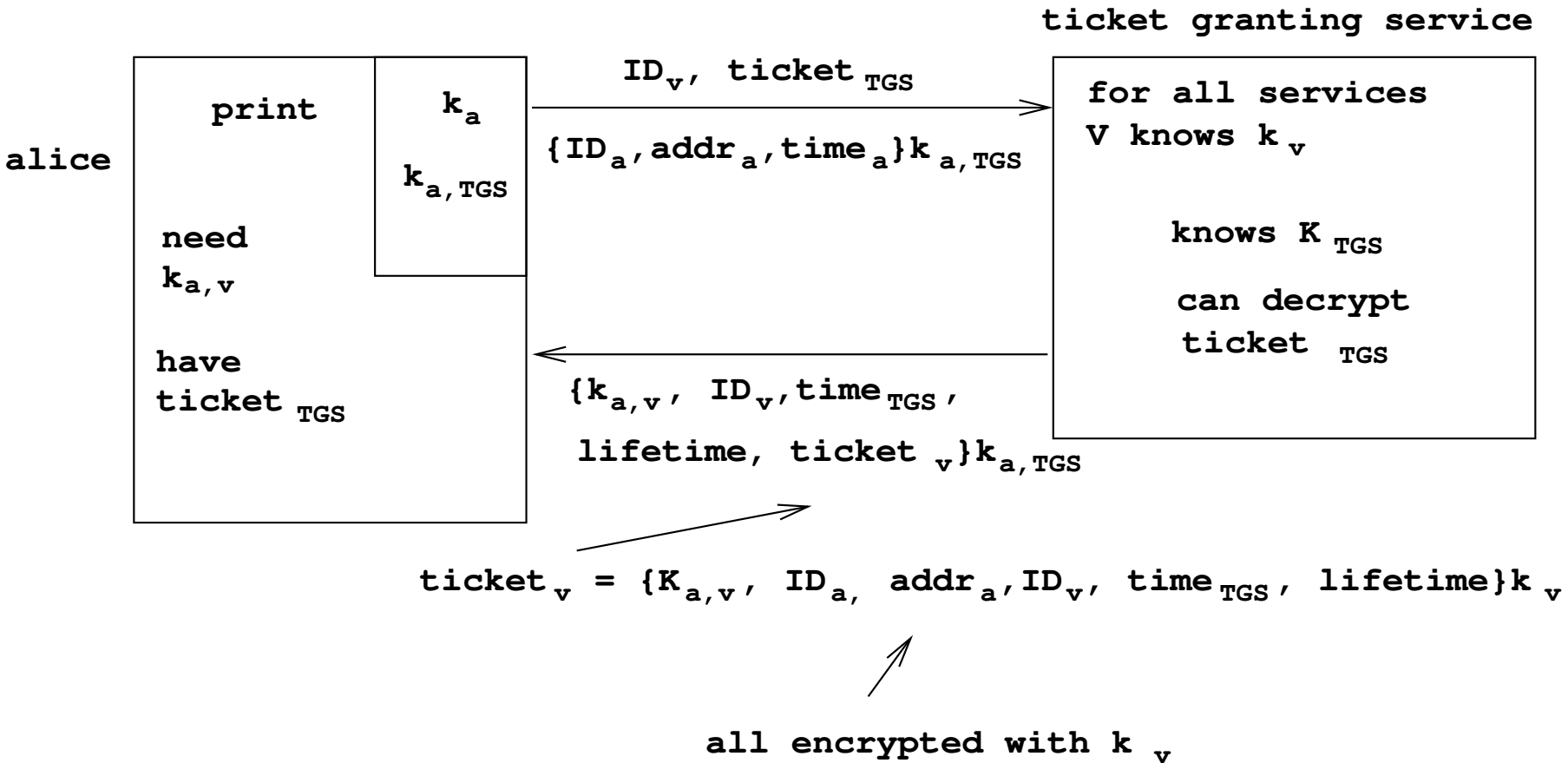

not encrypted because no key
exchange has been done yet
ticket granting service
alice
print
need
$k_{a,v}$
have
ticket $_{TGS}$
k_a
$k_{a,TGS}$
ID_v, ticket $_{TGS}$
$\{ID_a, addr_a, time_a\}k_{a,TGS}$
for all services
V knows k_v
knows K_{TGS}
can decrypt
ticket $_{TGS}$

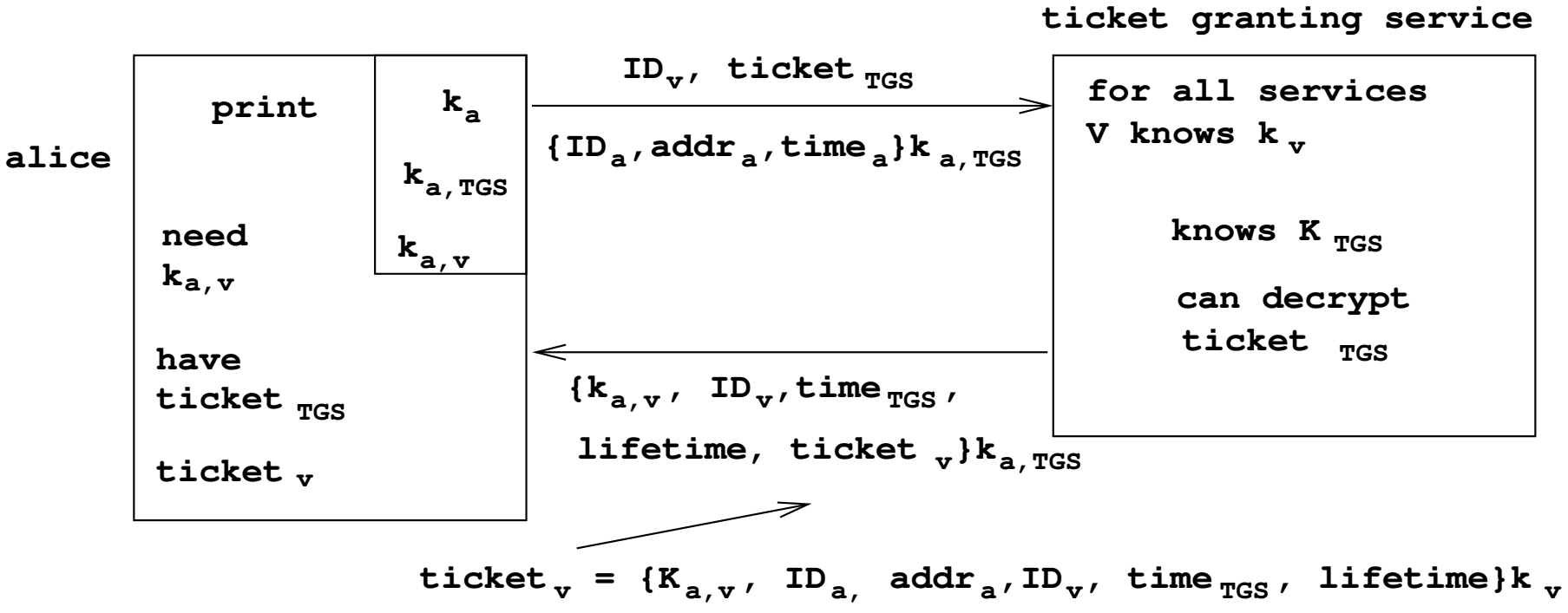

not encrypted because no key
exchange has been done yet
ticket granting service
alice
print
need
$k_{a,v}$
have
ticket $_{TGS}$
k_a
$k_{a,TGS}$
ID_v, ticket $_{TGS}$
$\{ID_a, addr_a, time_a\}k_{a,TGS}$
called
authenticator
for all services
V knows k_v
knows K_{TGS}
can decrypt
ticket $_{TGS}$

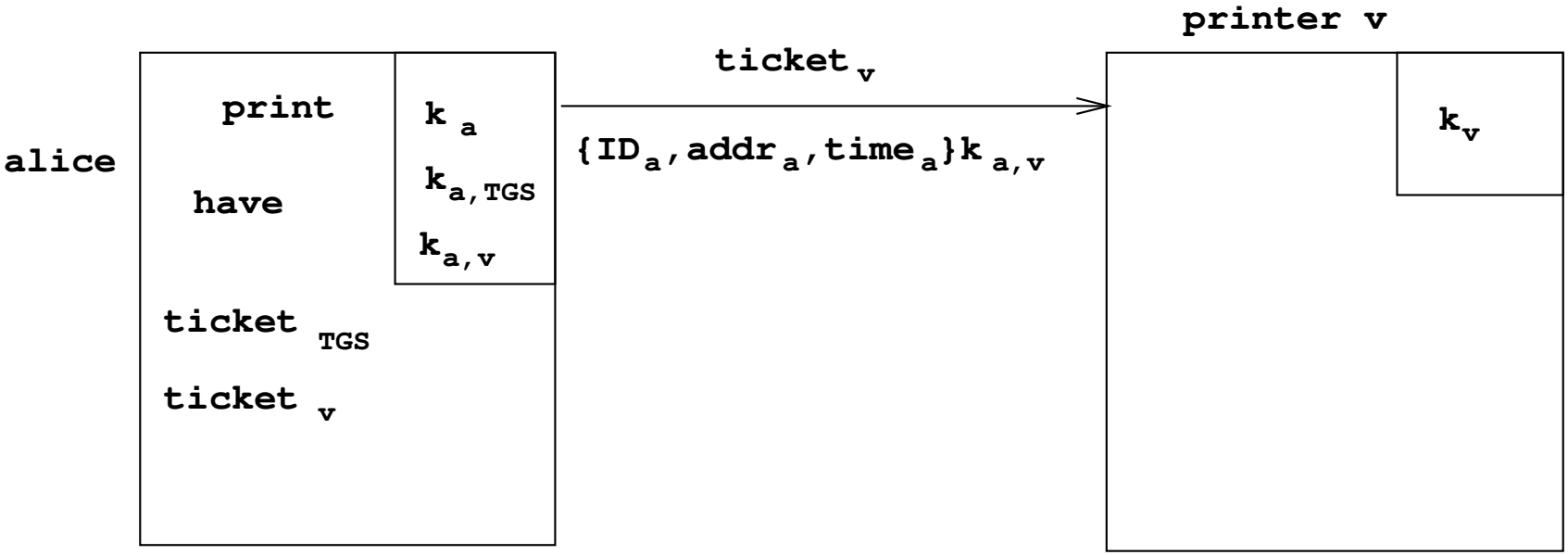

ticket granting service
alice
print
k_a
$k_{a,TGS}$
need
$k_{a,v}$
have
ticket $_{TGS}$
ID_v, ticket $_{TGS}$
$\{ID_a, addr_a, time_a\}k_{a,TGS}$
$\{k_{a,v}$, ID_v, $time_{TGS}$,
lifetime, ticket $_v\}k_{a,TGS}$
for all services
V knows k_v
knows K_{TGS}
can decrypt
ticket $_{TGS}$

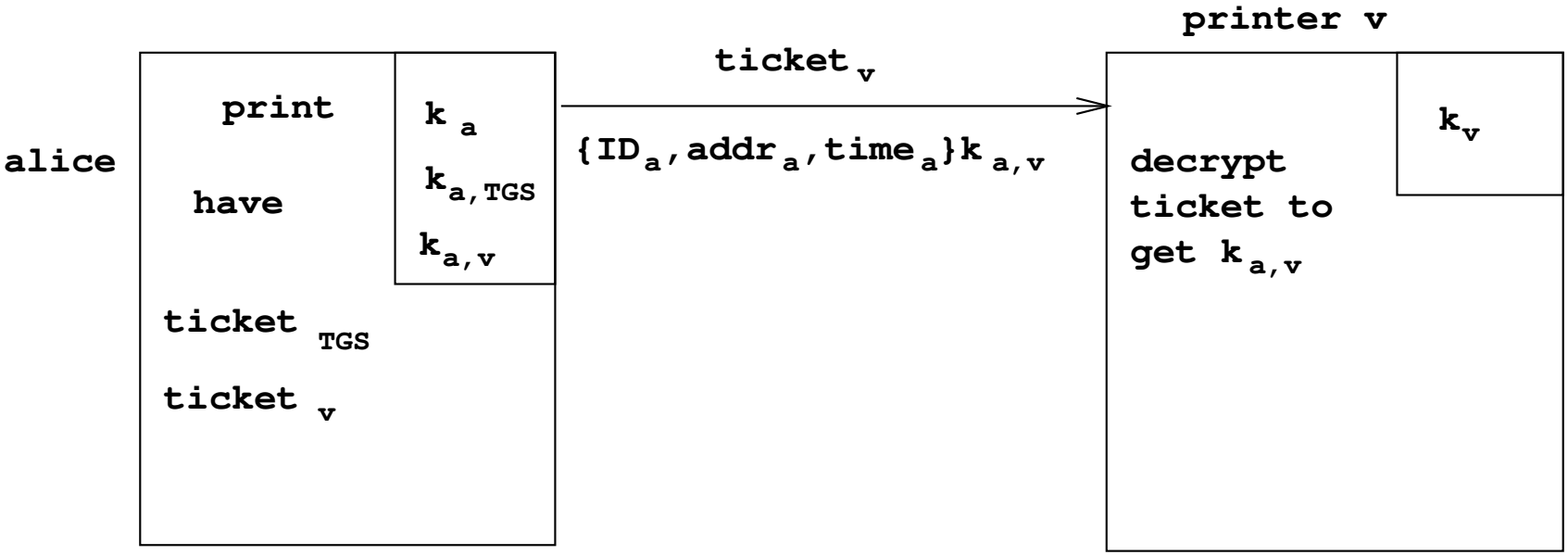

ticket granting service
alice
print
k_a
$k_{a,TGS}$
need $k_{a,v}$
have ticket $_{TGS}$
ID_v, ticket $_{TGS}$
$\{ID_a, addr_a, time_a\}k_{a,TGS}$
for all services V knows k_v
knows K_{TGS}
can decrypt ticket $_{TGS}$
$\{k_{a,v}$, ID_v, $time_{TGS}$,
lifetime, ticket $_v\}k_{a,TGS}$
all encrypted with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

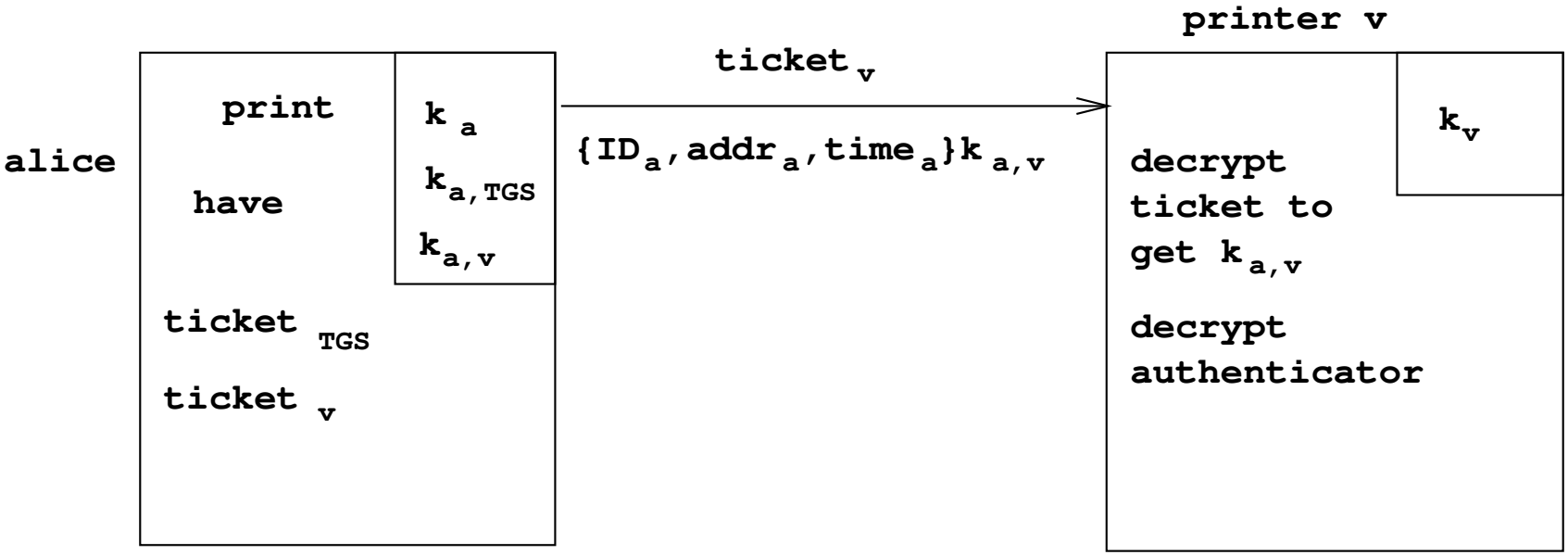

ticket granting service
alice
print
need $k_{a,v}$
have ticket $_{TGS}$
k_a
$k_{a,TGS}$
ID_v, ticket $_{TGS}$
$\{ID_a, addr_a, time_a\}k_{a,TGS}$
for all services V knows k_v
knows K_{TGS}
can decrypt ticket $_{TGS}$
$\{k_{a,v}$, ID_v, time $_{TGS}$,
lifetime, ticket $_v\}k_{a,TGS}$
ticket $_v$ = $\{K_{a,v}$,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

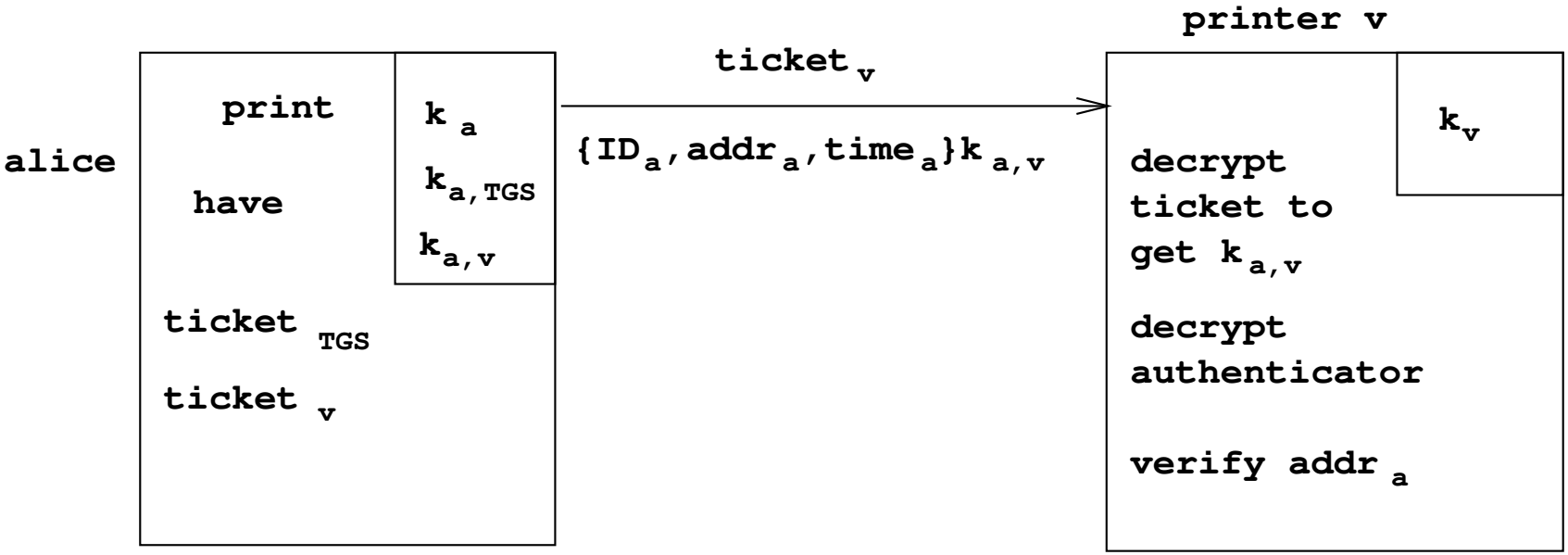

ticket granting service
alice
print
need $k_{a,v}$
have ticket $_{TGS}$
k_a
$k_{a,TGS}$
ID_v, ticket $_{TGS}$
$\{ID_a, addr_a, time_a\}k_{a,TGS}$
for all services V knows k_v
knows K_{TGS}
can decrypt ticket $_{TGS}$
$\{k_{a,v}$, ID_v, time $_{TGS}$,
lifetime, ticket $_v\}k_{a,TGS}$
ticket $_v$ = $\{K_{a,v}$,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

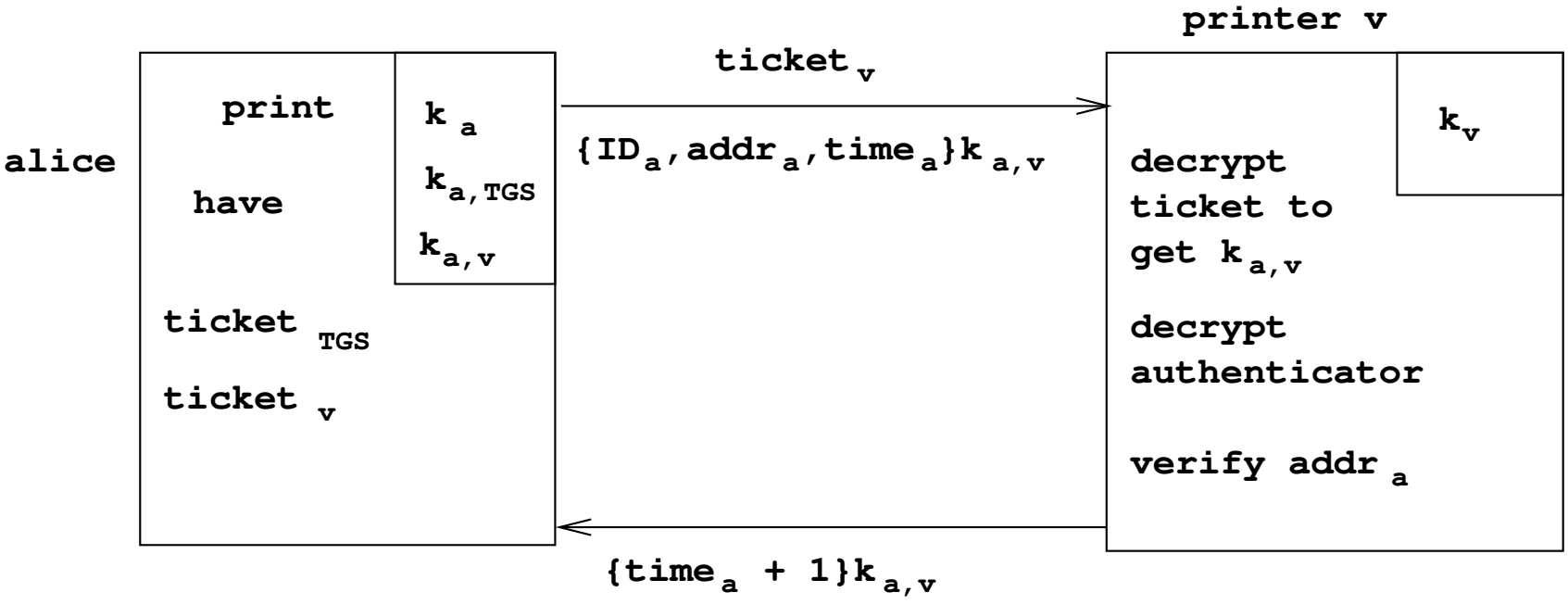

ticket granting service
alice
print
k_a
$k_{a,TGS}$
$k_{a,v}$
need $k_{a,v}$
have ticket $_{TGS}$
ticket $_v$
ID_v, ticket $_{TGS}$
$\{ID_a, addr_a, time_a\}k_{a,TGS}$
for all services
V knows k_v
knows K_{TGS}
can decrypt ticket $_{TGS}$
$\{k_{a,v}, ID_v, time_{TGS},$
lifetime, ticket $_v\}k_{a,TGS}$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

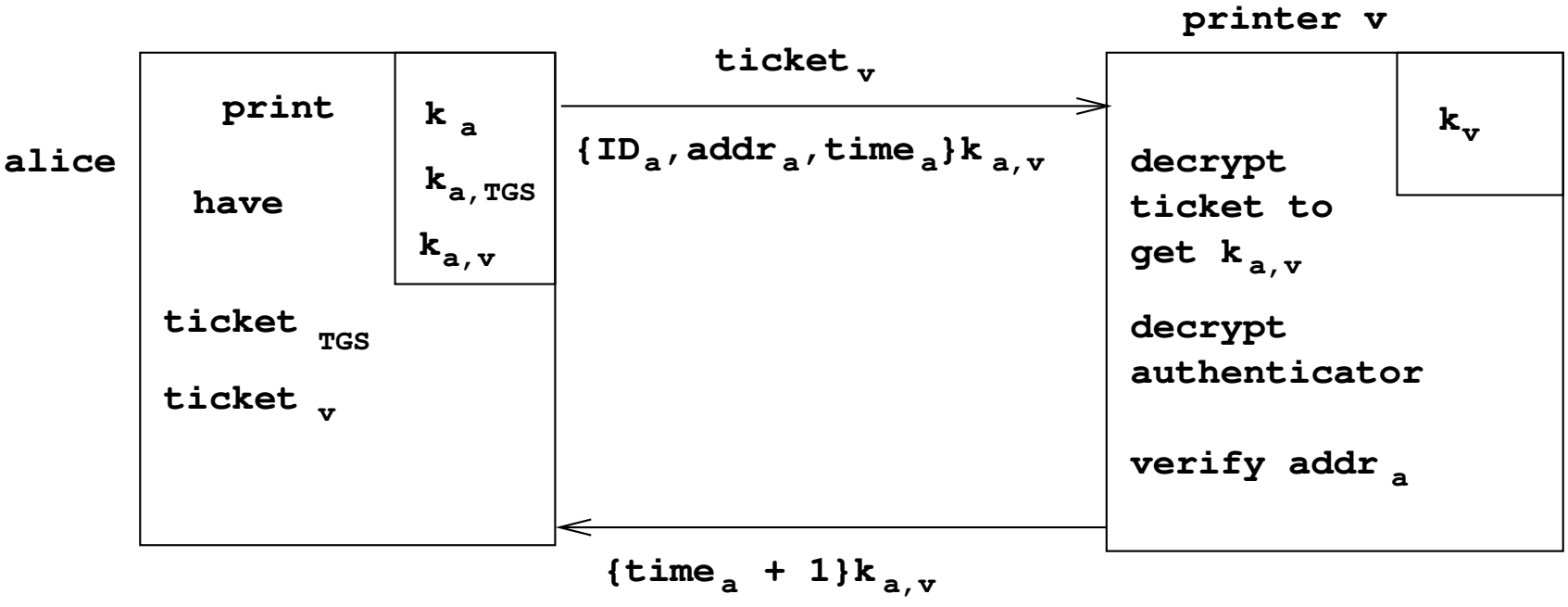

printer v
ticket v
alice
print
have
ticket TGS
ticket v
k a
k a,TGS
k a,v
{ID a, addr a, time a}k a,v
k v


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


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


printer 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
k v
decrypt ticket to get k a,v
decrypt authenticator
verify addr a


printer 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
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