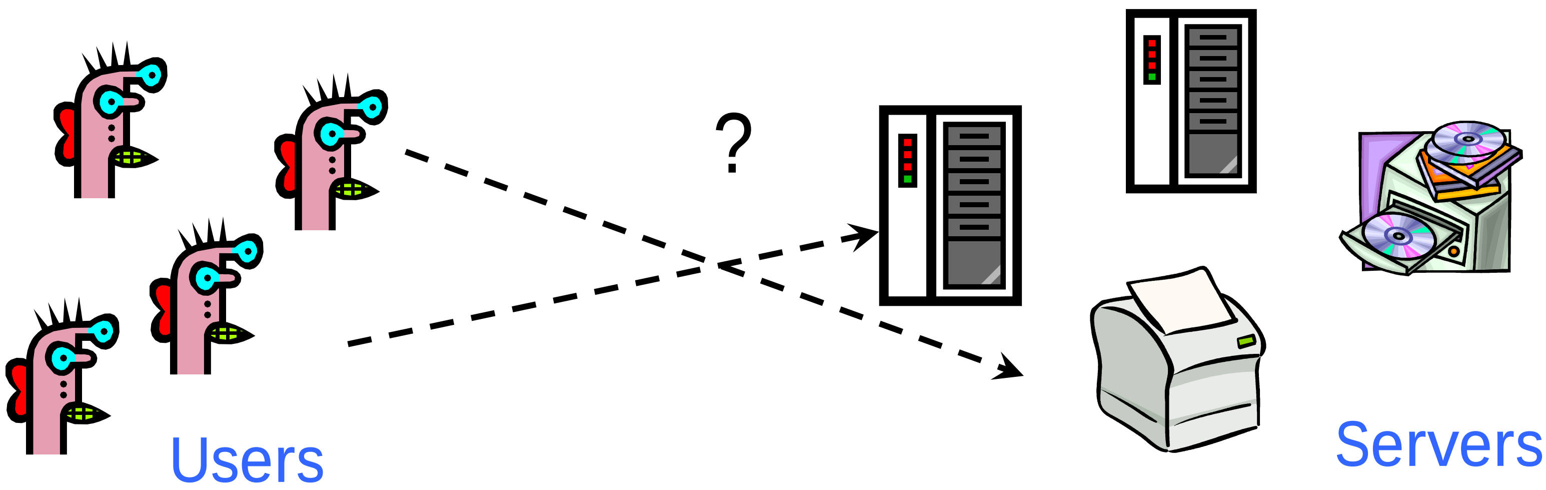
Authentication: Kerberos

CSC 154

Many-to-Many Authentication



How do users prove their identities when requesting services from machines on the network?

Naïve solution: every server knows every user's password

- Insecure: break into one server $\Rightarrow$ compromise all users
- Inefficient: to change password, user must contact every server

Observation

- When the enterprise is **large**:
  - Letting servers store passwords is a **bad** idea
    - –Vulnerable to password cracking
    - –Administrators' nightmare
  - Letting users remember fancy one-time passwords is also a **bad** idea

Dream authentication service

- Let users use only passwords → simple
- Do not let servers store passwords

  - While achieving strong security against
    - Password cracking
    - Eavesdropping
    - Impersonating attack
    - Replay attack

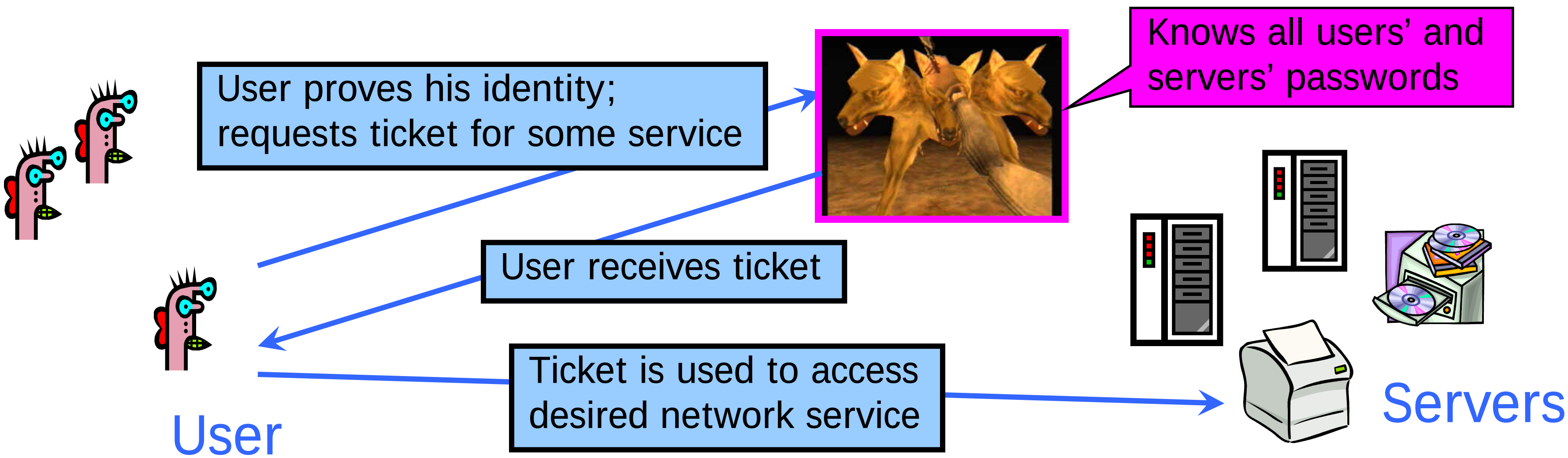
Requirements

- Security
  - ... against attacks by passive eavesdroppers and actively malicious users
- Reliability
- Transparency
  - Users shouldn’t notice authentication taking place
  - Entering password is Ok, if done rarely
- Scalability
  - Large number of users and servers

Threats

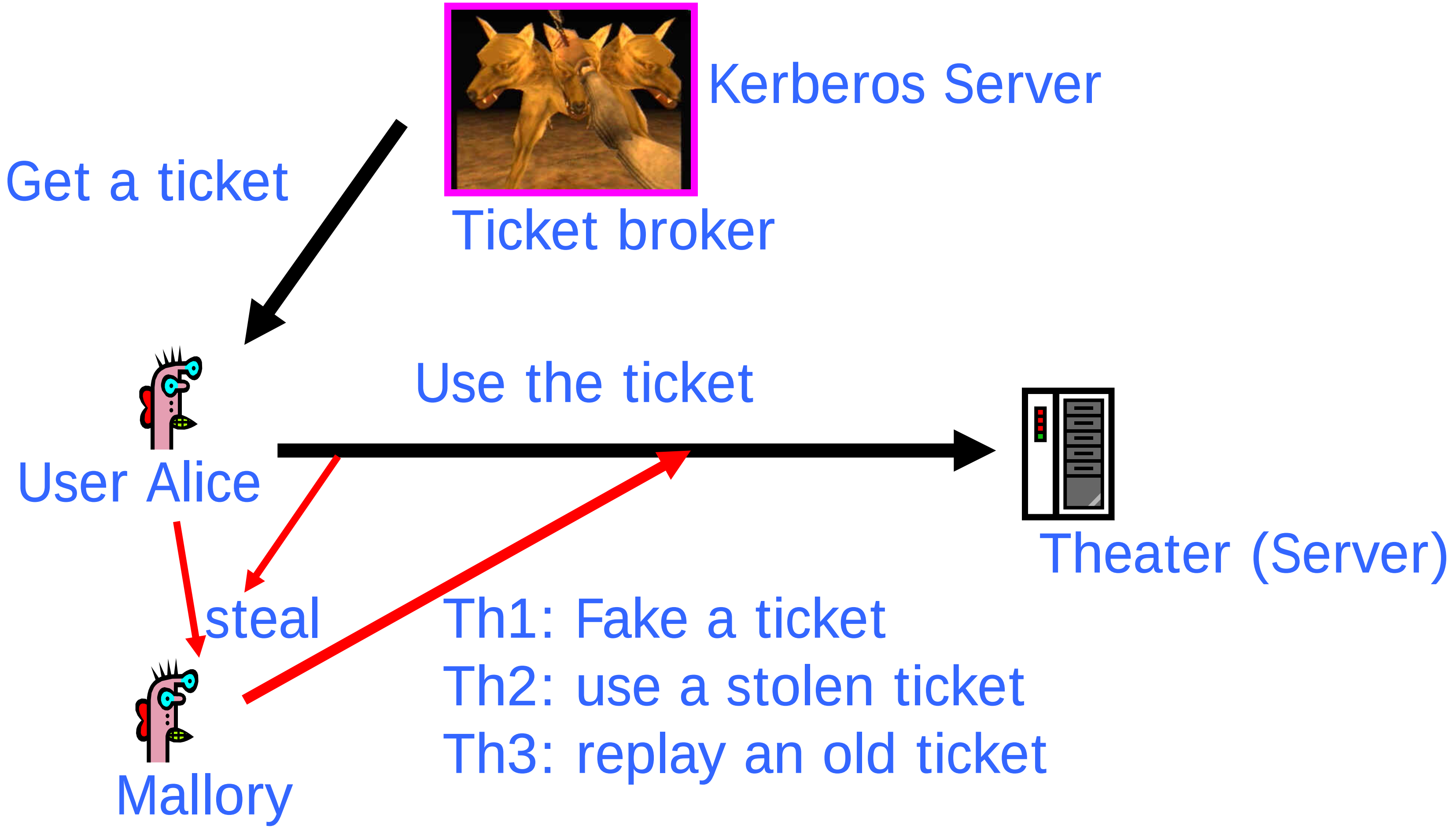
- User impersonation
  - Malicious user with access to a workstation pretends to be another user from the same workstation
    - Can’t trust workstations to verify users’ identities
- Network address impersonation
  - Malicious user changes network address of his workstation to impersonate another workstation
- Eavesdropping, tampering and replay
  - Malicious user eavesdrops, tampers or replays other users’ conversations to gain unauthorized access

Solution: Trusted Third Party



- Trusted authentication service on the network
  - Knows all passwords, can grant access to any server
  - Convenient, but also the single point of failure
  - Requires high level of physical security

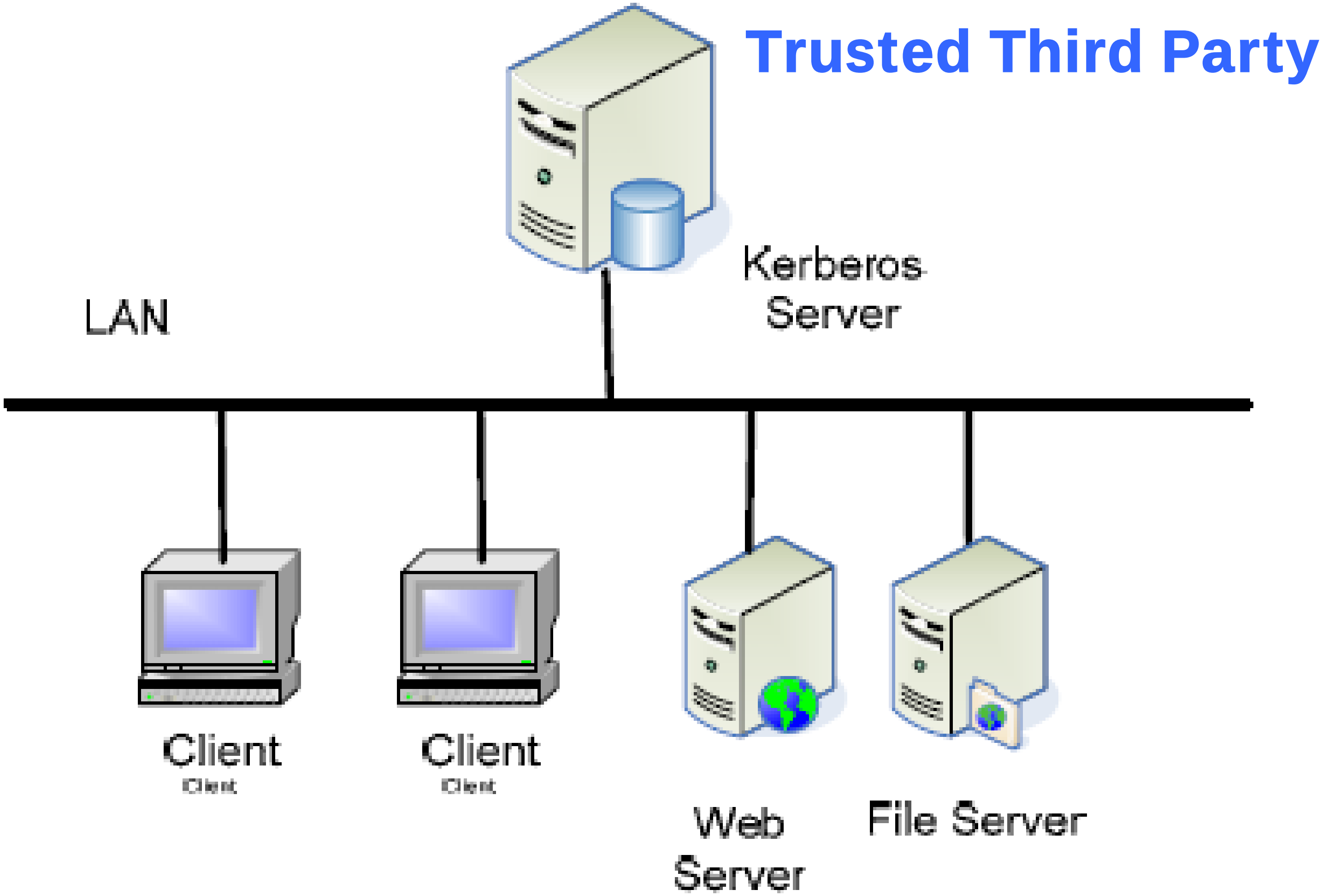
Kerberos Idea: ticket-based service access



The Ticket Analogy

- People need a ticket to access a service: a Broadway show
  - The ticket proves that you **earned** the service
- Each server is a theater
- The trusted third party (i.e., the Kerberos server) is the **ticket broker**
  - One broker handles all Broadway shows
- Servers do not store passwords → theaters do not sell tickets

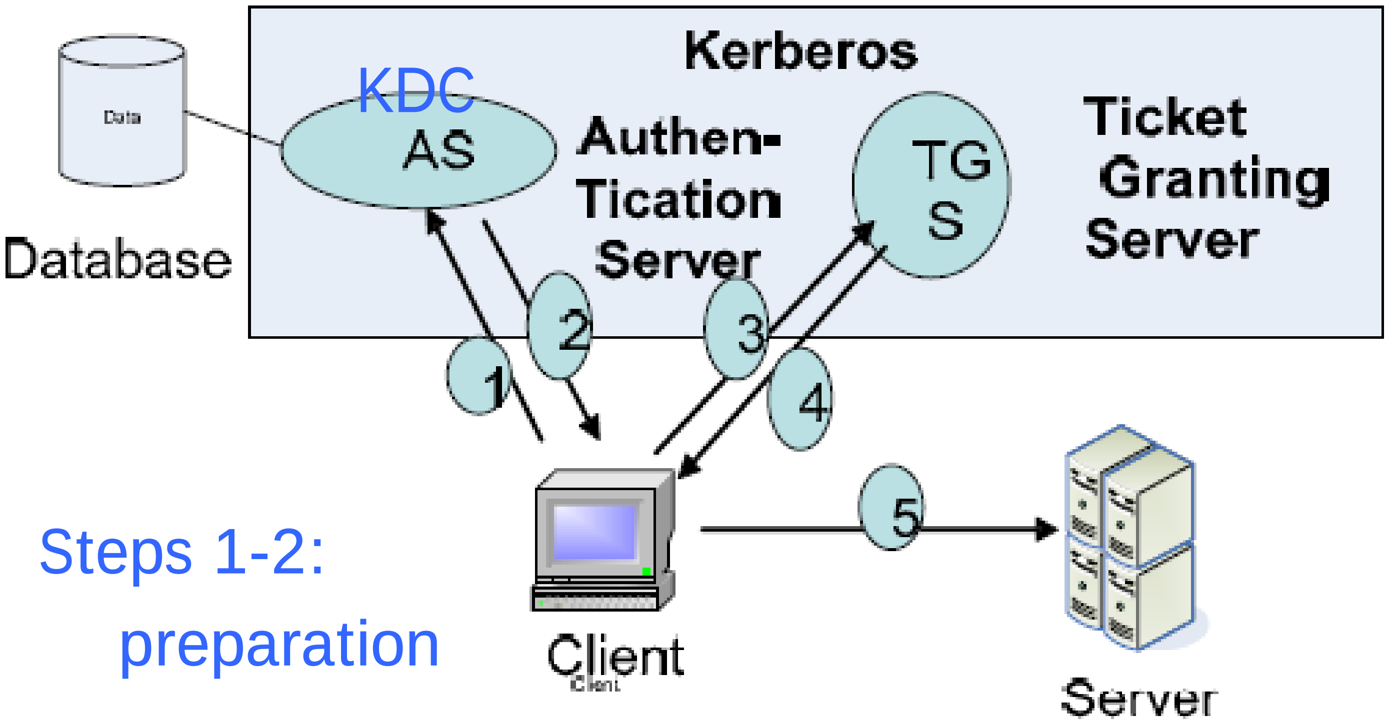
Network deployment



Threats to ticket-based service access

- **TH1: Fake a ticket →** (fix #1) encrypt it with a key **only** the server knows
- **TH2: Steal a ticket →**
  - (fix #2) Let the user **prove** his identity when buying the ticket: then put user name on the ticket
  - (fix #3) Check **identify** before granting service
- **Th3: Replay an old ticket →** (fix #4) using *timestamp*

Kerberos in a nutshell



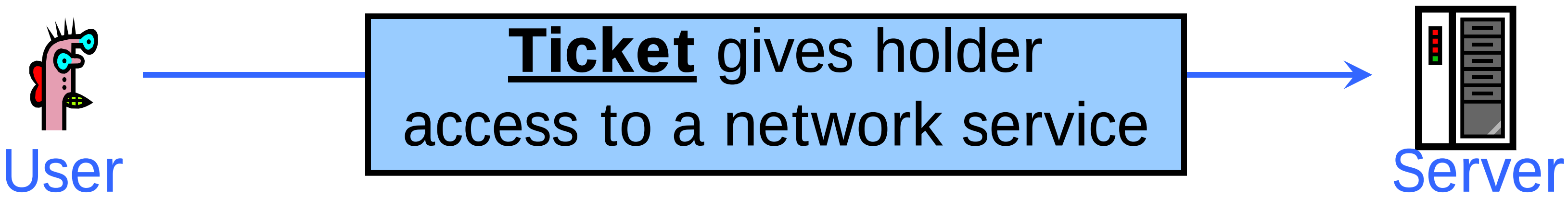
Steps 1-2:
preparation

Step 3: fix #2 (prove identity) and fix #4 (timestamp)

Step 4: fix #1 (encrypt the ticket)

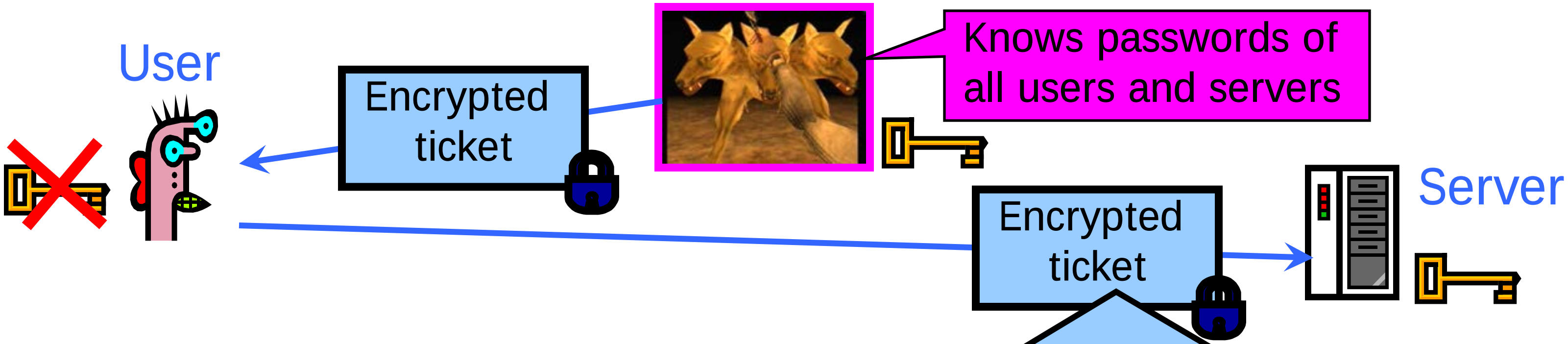
Step 5: fix #3 (check identity) and fix #4 (timestamp)

What Should a Ticket Look Like?

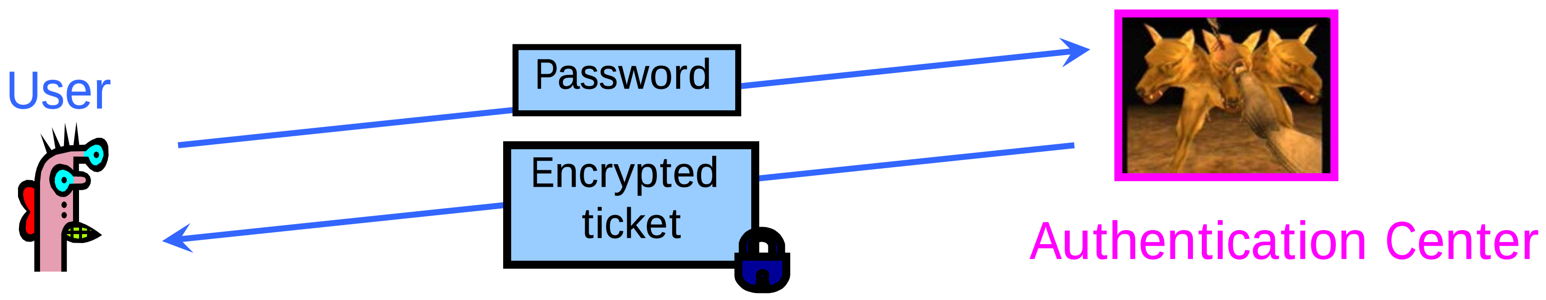


- Ticket could be faked
- Solution: encrypt some information with a key known to the server (but not the user!)
  - Server can decrypt ticket and verify information
  - User does not learn server's key

What Should a Ticket Include?



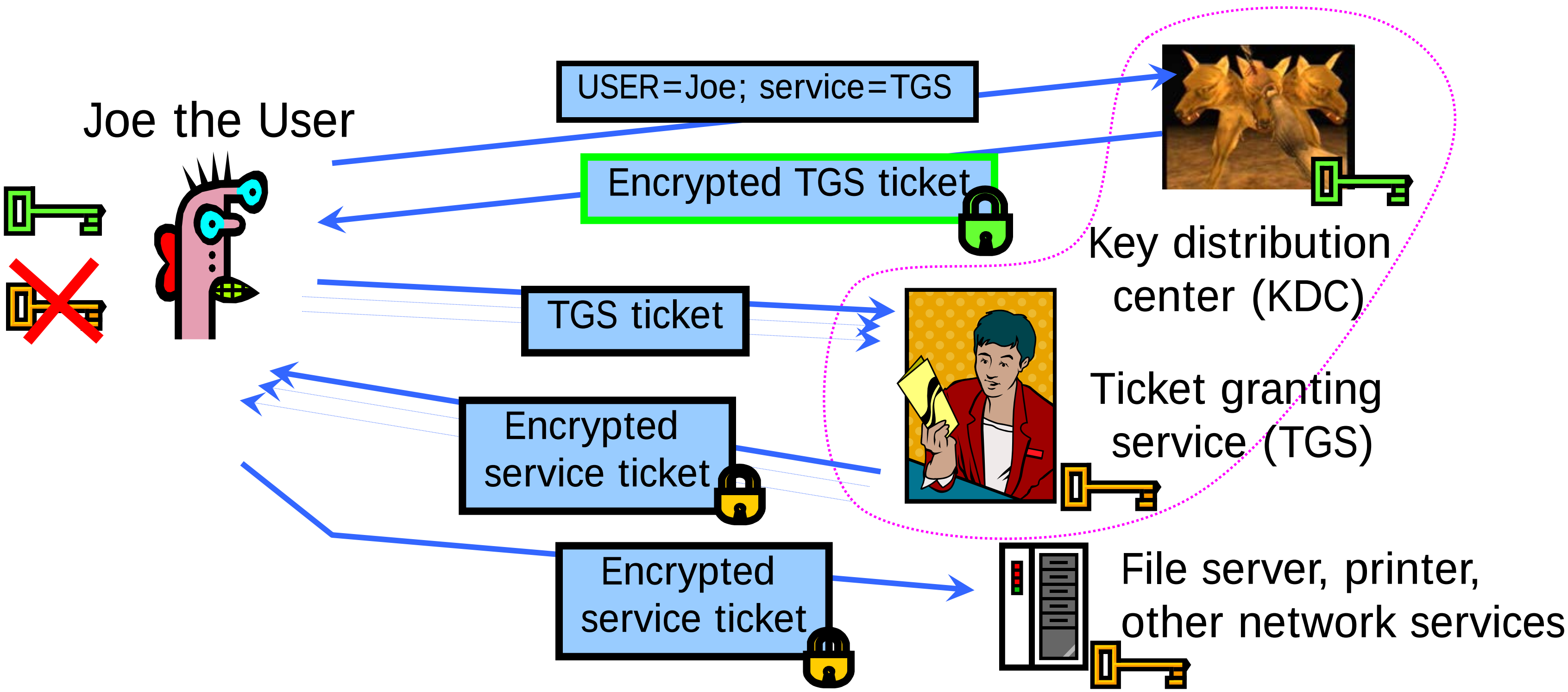
- User name
- Server name
- Address of user's workstation
  - Otherwise, a user on another workstation can steal the ticket and use it to gain access to the server
- Ticket lifetime
- A few other things (e.g., session key)



- **Insecure:** passwords are sent in plaintext
  - Eavesdropper can steal the password and later impersonate the user to the authentication server
- **Inconvenient:** need to send the password each time to obtain the ticket for any network service
  - Separate authentication for email, printing, etc.

Two-Step Authentication

- Prove identity **once** to obtain special TGS ticket
- Use TGS to get tickets for any network service



4 players in Kerberos

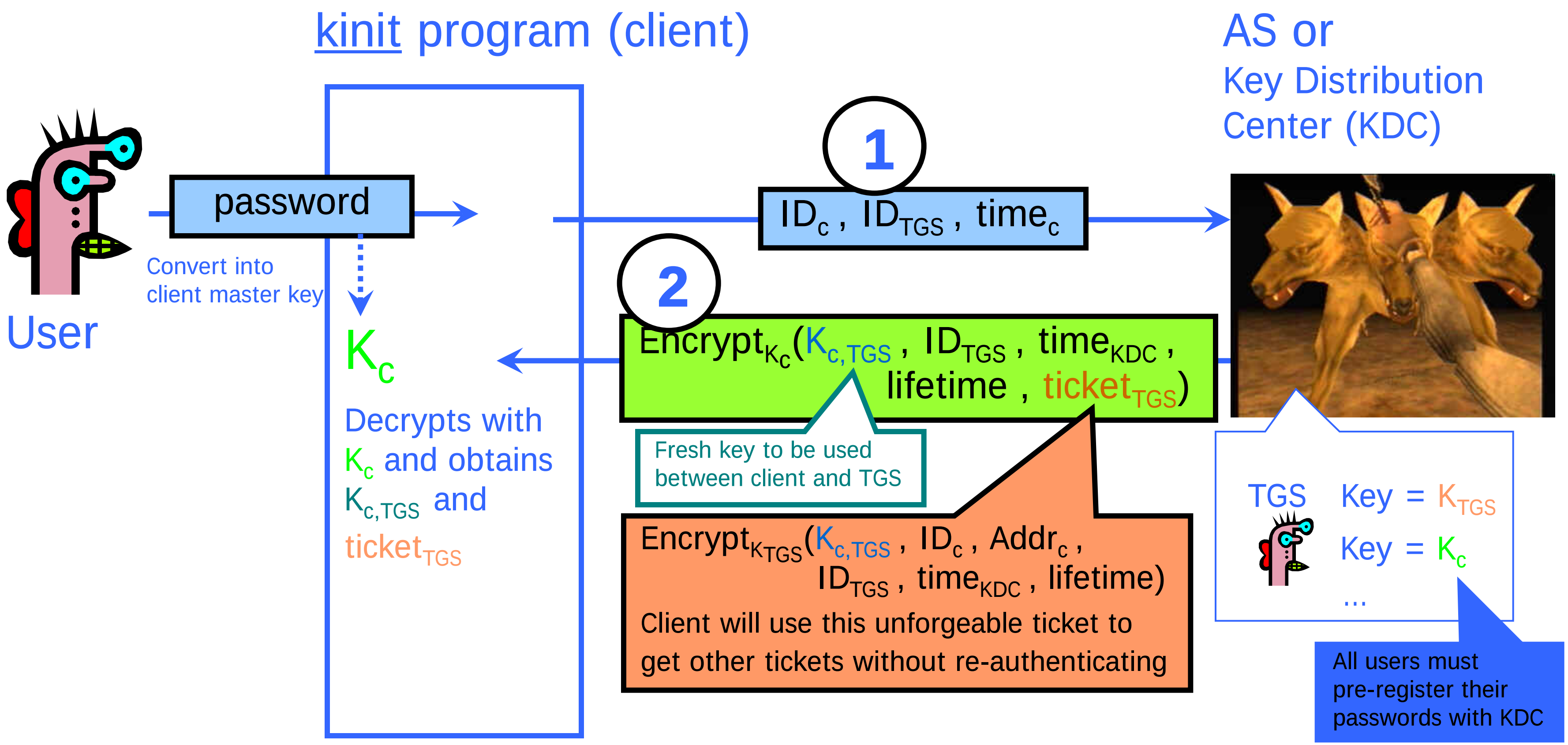
- TGS
- KDC (AS)
-
- Service V – an email server; a printing server
  - Note: in our reading material, V is denoted as W
- Client C

- The other players are NOT important

Symmetric Keys in Kerberos

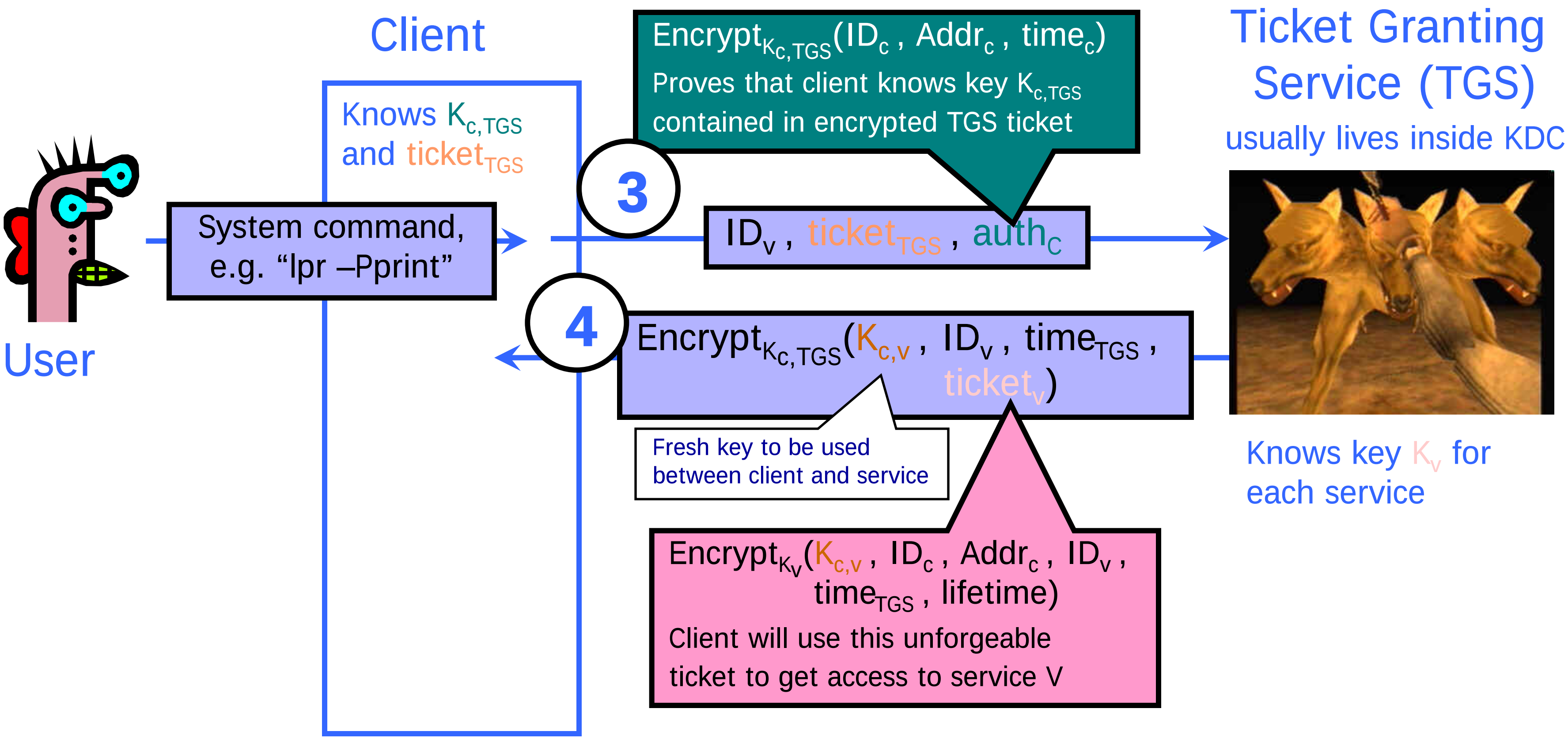
- K_c is <u>long-term</u> key of client C
  - Derived from user's password
  - Known to client and key distribution center (KDC)
- K_{TGS} is <u>long-term</u> key of TGS
  - Known to KDC and ticket granting service (TGS)
- K_v is <u>long-term</u> key of network service/server V
  - Known to V and TGS; separate key for each service
- $K_{c,TGS}$ is <u>short-term</u> key between C and TGS
  - Created by KDC, known to C and TGS
- $K_{c,v}$ is <u>short-term</u> key between C and V
  - Created by TGS, known to C and V

"Single Logon" Authenticatio: steps 1-2



- Client only needs to obtain TGS ticket **once** (say, every morning)
  - Ticket is encrypted; client cannot forge it or tamper with it

Obtaining a Service Ticket: steps 3-4

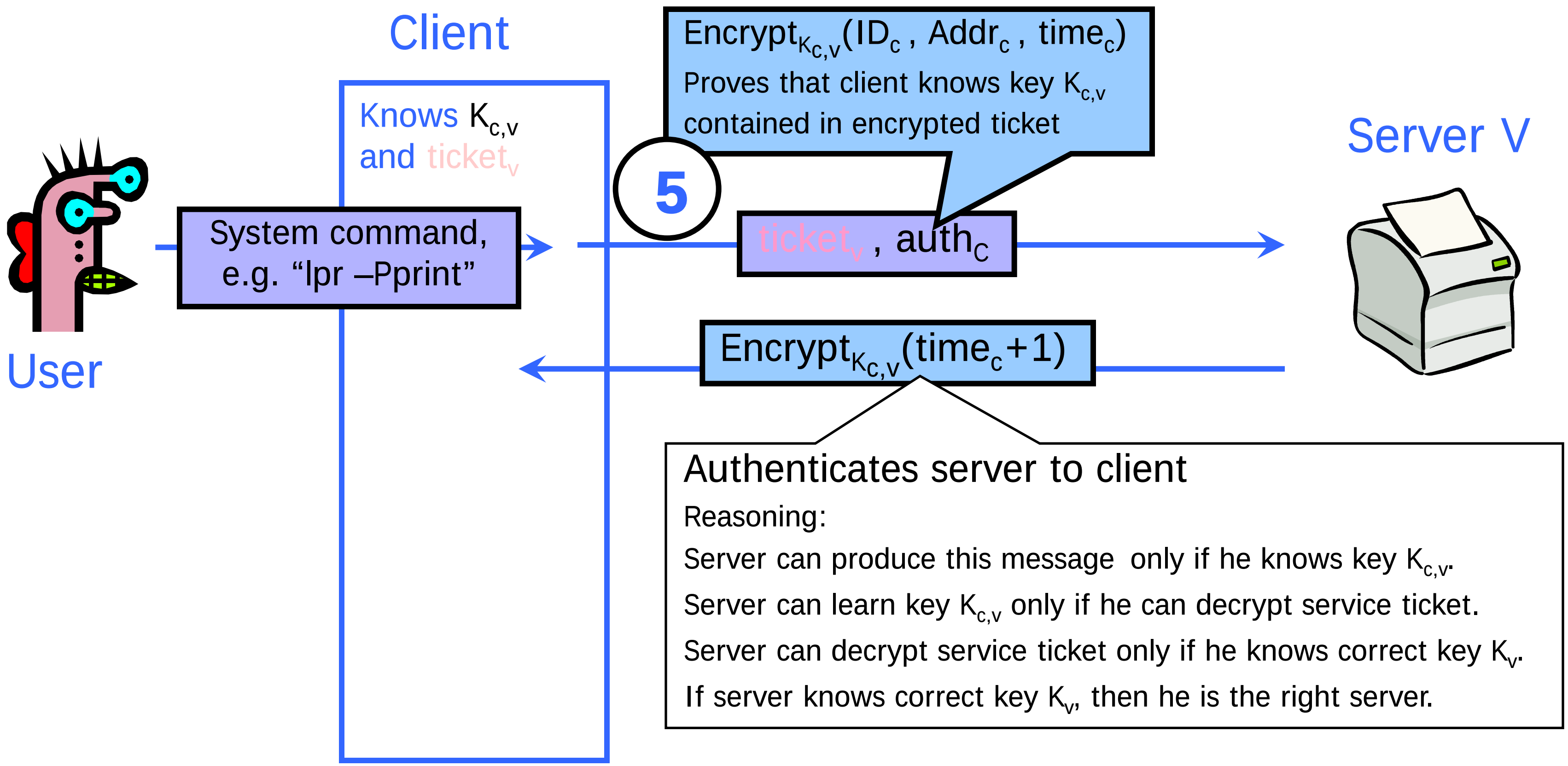


- Client uses TGS ticket to obtain a service ticket and a short-term key for each network service
  - One encrypted, unforgeable ticket per service (printer, email, etc.)

Steps 3-4: summary

- Goal: get the service ticket
- To avoid ticket stealing, TGS will stamp the user's name on every ticket → hence, TGS needs to authenticate the user's identity
- To prove her identity, Alice will send two things to TGS: TGT + authenticator
  - Both are encrypted
- TGS can decrypt the TGT in which TGS will find the authenticator encryption key → then TGS can decrypt the authenticator
- TGS compares the info items contained in the two things → **if they match**, the user is **authenticated**!

Obtaining Service: Step 5



- For each service request, client uses the short-term key for that service and the ticket he received from TGS

Step 5: summary

- Goal: Alice proves her identity to the server; then the server will provide the service
- To avoid ticket stealing, the ticket contains the user’s name
- To avoid attack faking, the ticket is encrypted by TGS and only the server can decrypt it
- To avoid both ticket stealing and replay attack, Alice needs to send another authenticator to the server
- The server decrypts the ticket in which it will find the authenticator encryption key → the server decrypts the authenticator → the server compares XXX with YYY → do they match?

Questions to discuss

- Has __5__ messages
- Has __4__ players
- Uses _5__ keys
- Uses _2__ authenticators
- Uses __2__ tickets (two types)
- Who knows what?
- Which keys are used for which purposes?
- Why replay attack will fail?
- Why stolen tickets will fail?

Who knows what? Who creates what?

[False] The user knows what is inside a TGT (permission)

[True] TGS knows what is inside a server ticket

[False] KDC knows what is inside a server ticket

[False] TGS knows the password of the user

[True] KDC knows the long term key of the user

[False] KDC knows the long term key of the server

[True] Each authenticator is known to two players

[True] Authenticators are always created by user

[True] Short term keys are always created by Kerberos

Which keys are used for which purposes?

- K_c – used by KDC and client
- K(c, TGS) – short term comm. Between client and TGS
- K_TGS – used to encrypt permission tickets
- K_v – used to encrypt real tickets
- K(c, v) – between client and server
- How about authenticators?
  - Only involve short term keys
  - All authenticators contain a timestamp → suppose to be a short term use

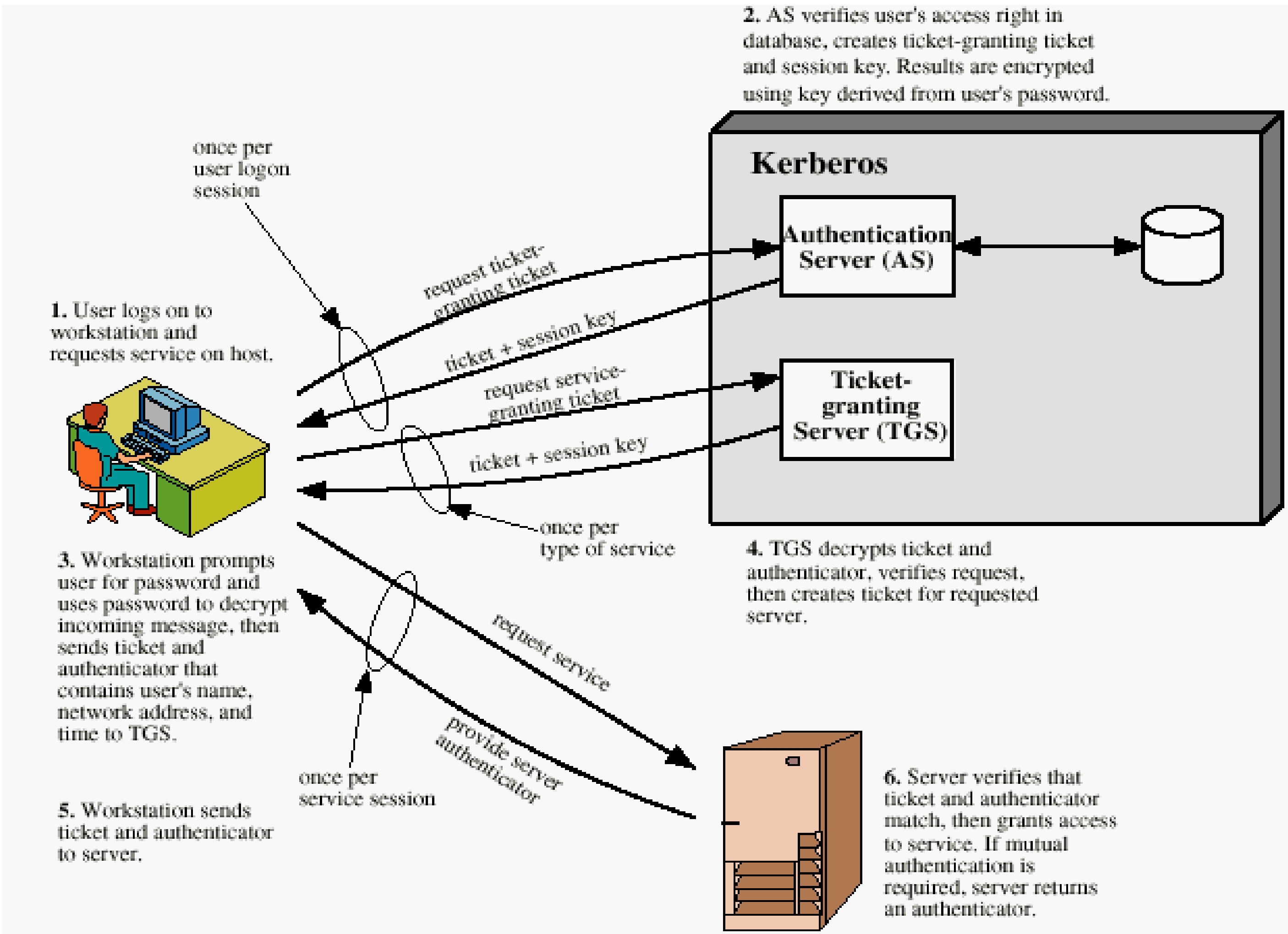
Why replay attack will fail?

- Timestamp is contained in authenticators
- Authenticators are required to access TGS and Server
- What if the attacker tries to modify timestamp?
  - Can he do this? Why hard to do?
  - If the user replays, he will succeed because he creates every authenticator
  - If the user machine is compromised, the attacker will succeed
  - Otherwise, the attacker cannot fake the timestamp

Why stolen tickets will fail?

- Because we use authenticators to check identity

Summary of Kerberos



Kerberos in Large Networks

- One KDC isn't enough for large networks (why?)
- Network is divided into realms
  - KDCs in different realms have different key databases
- To access a service in another realm, users must...
  - Get ticket for home-realm TGS from home-realm KDC
  - Get ticket for remote-realm TGS from home-realm TGS
    - As if remote-realm TGS were just another 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 derive short-term keys
  - Separate session key for each user-server pair
    - ... but multiple user-server sessions re-use the same key
- Proofs of identity are based on authenticators
  - Client encrypts his identity, address and current time using a short-term session key
    - Also prevents replays (if clocks are globally synchronized)
  - Server learns this key separately (via encrypted ticket that client can't decrypt) and verifies user's identity
- Symmetric cryptography only

Problematic Issues

- Password dictionary attacks on client master keys
- Replay of authenticators
  - 5-minute lifetimes long enough for replay
  - Timestamps assume global, secure synchronized clocks
  - Challenge-response would have been better
- Same user-server key used for all sessions
- Extraneous double encryption of tickets
- No ticket delegation
  - Printer can’t fetch email from server on your behalf

- Ticket hijacking
  - Malicious user may steal the service ticket of another user on the same workstation and use it
    - IP address verification does not help
  - Servers must verify that the user who is presenting the ticket is the same user to whom the ticket was issued
- No server authentication
  - Attacker may misconfigure the network so that he receives messages addressed to a legitimate server
    - Capture private information from users and/or deny service
  - Servers must prove their identity to users