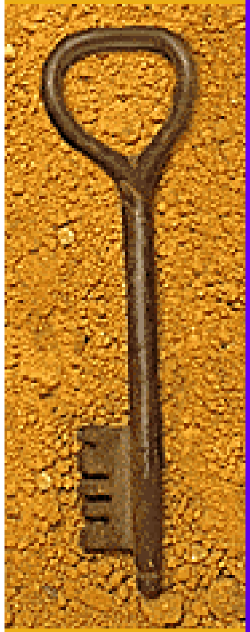
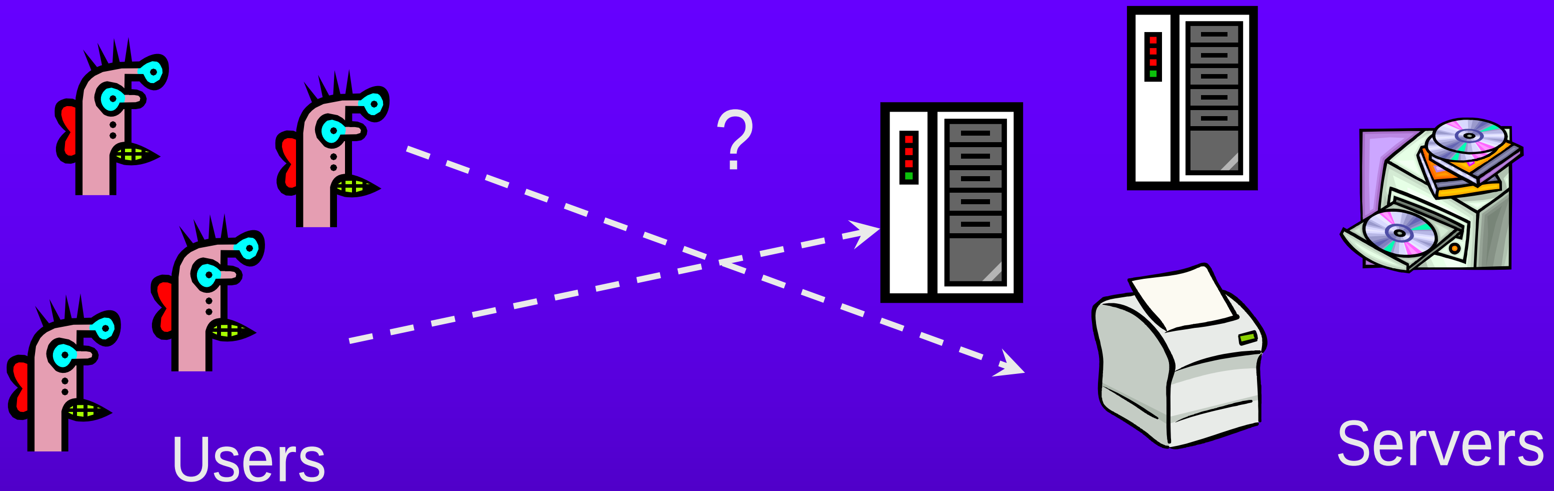
Kerberos & Single-Sign-On



Many-to-Many Authentication



?

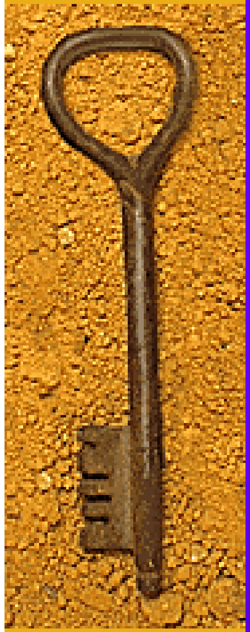
Users

Servers

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

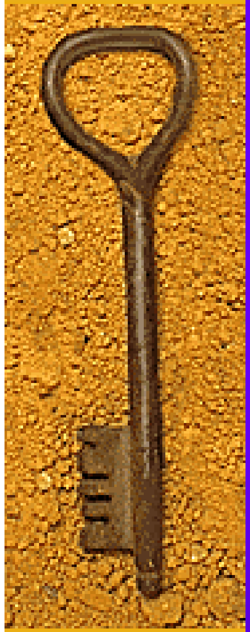
Naïve solution: every server knows every user's password

- Insecure: compromise of one server is enough to compromise all users
- Inefficient: to change his password, user must contact every server



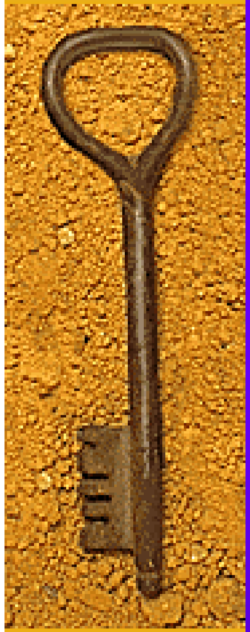
Requirements

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

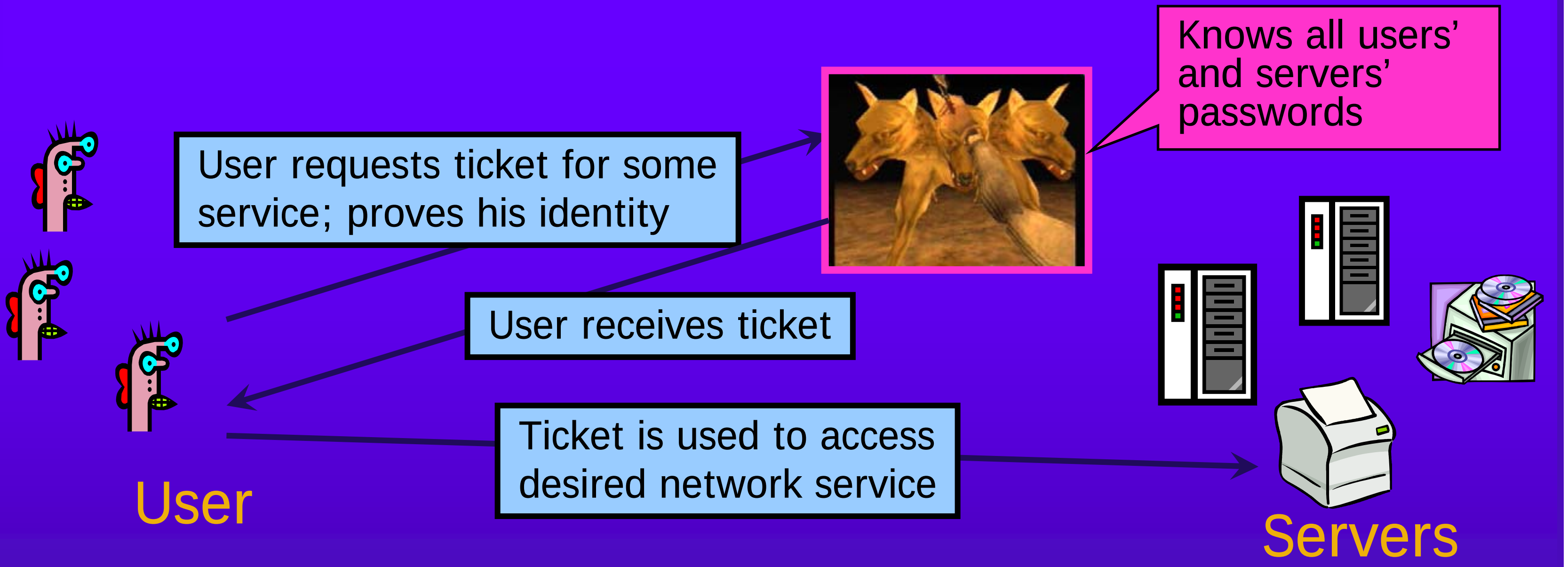


Threats

- 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 on, tampers with or replays other users’ conversations to gain unauthorized access



Solution: Trusted Third Party



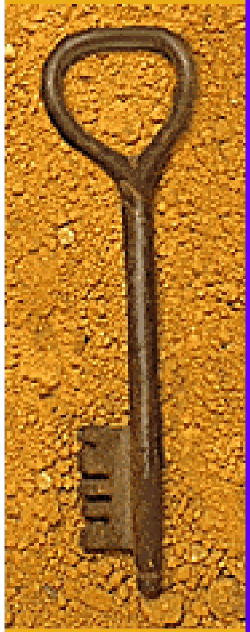
Knows all users' and servers' passwords

User requests ticket for some service; proves his identity

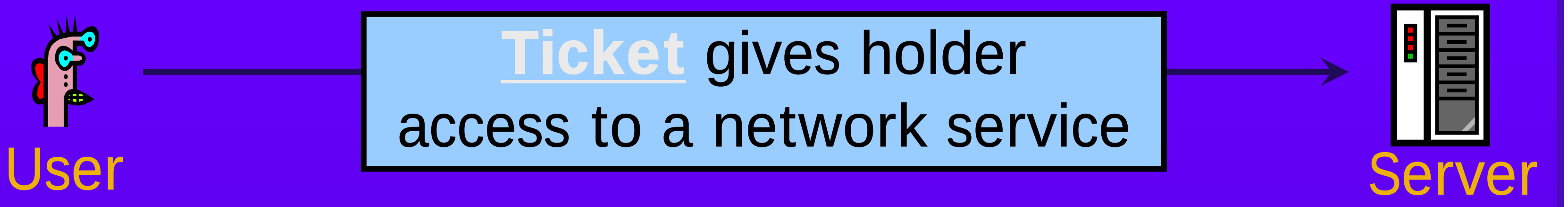
User receives ticket

Ticket is used to access desired network service

- 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



What Should a Ticket Look Like?

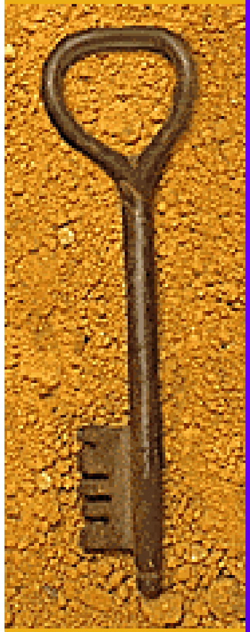


User

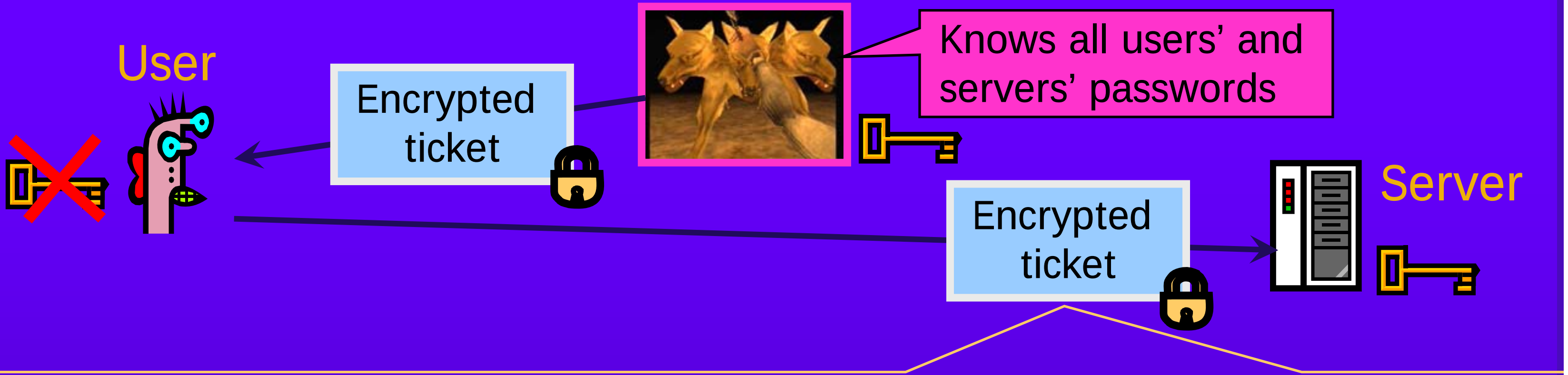
Ticket gives holder access to a network service

Server

- Ticket cannot include server's plaintext password
  - Otherwise, next time user will access server directly without proving his identity to authentication service
- Solution: encrypt some information with a key derived from the server's password
  - Server can decrypt ticket and verify information
  - User does not learn server's password

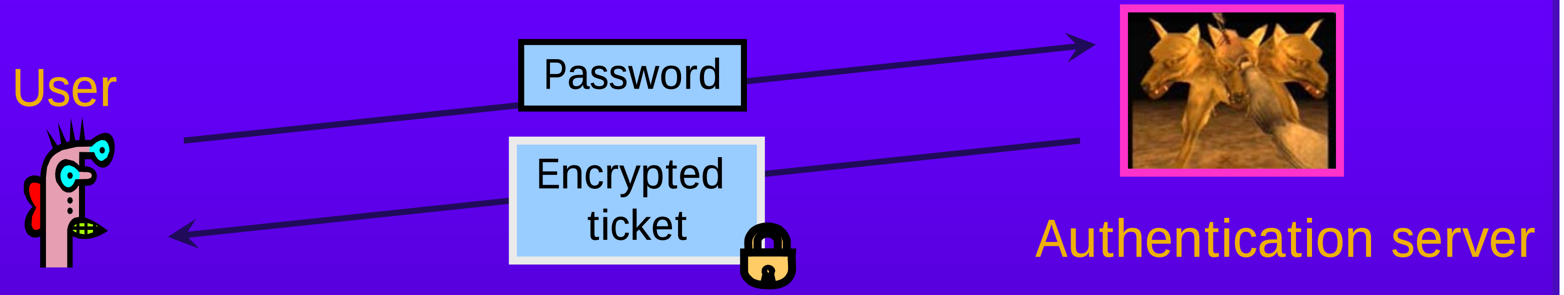


What Should a Ticket Include?

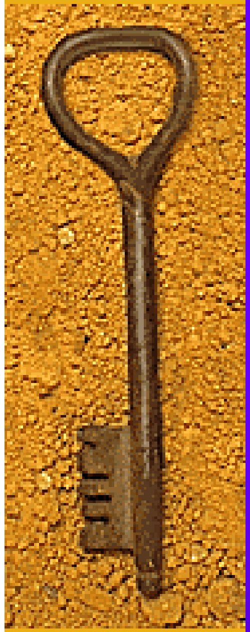


- User name
- Server name
- Address of user's workstation
  - Otherwise, a user on another workstation can steal the ticket and use it to gain access to the server
- Ticket lifetime
- A few other things (e.g., session key)

How Is Authentication Done?

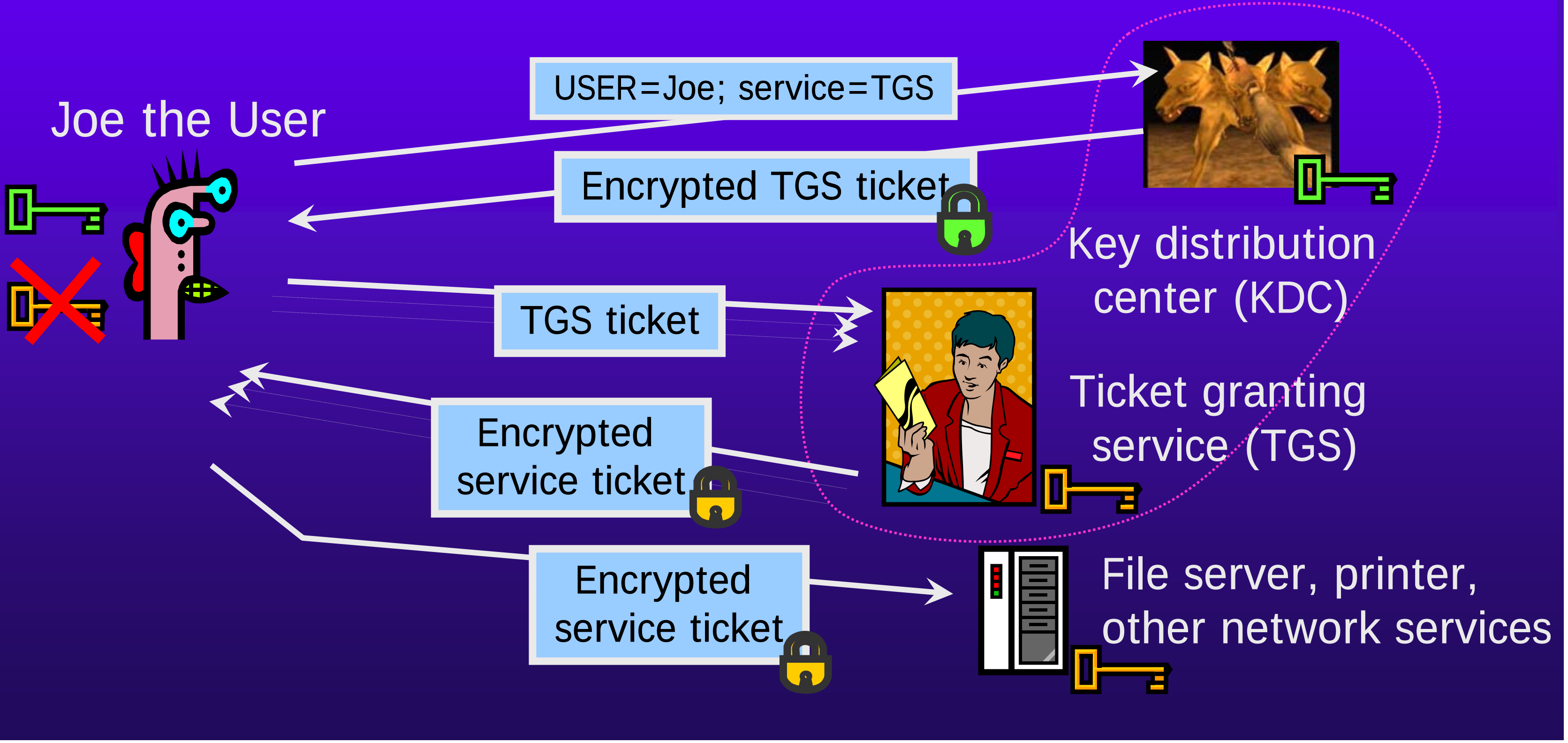


- Insecure: passwords are sent in plaintext
  - Eavesdropper can steal the password and later impersonate the user to the authentication server
- Inconvenient: need to send the password each time to obtain the ticket for any network service
  - Separate authentication for email, printing, etc.



Solution: Two-Step Authentication

- Prove identity **once** to obtain special TGS ticket
  - Instead of password, use key derived from password
- Use TGS to get tickets for many network services



Joe the User

USER=Joe; service=TGS

Encrypted TGS ticket

Key distribution center (KDC)

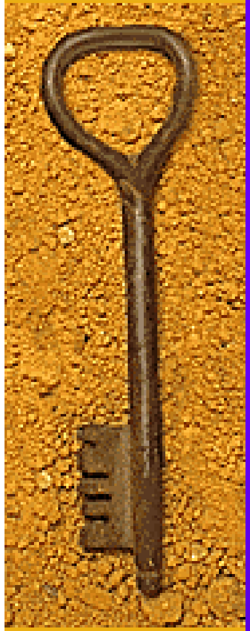
TGS ticket

Encrypted service ticket

Ticket granting service (TGS)

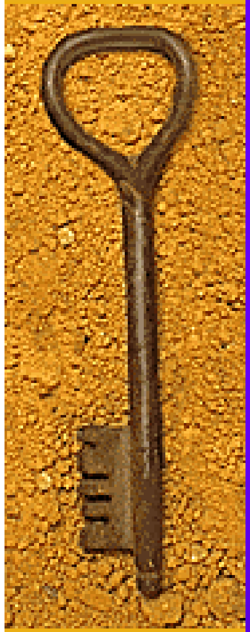
Encrypted service ticket

File server, printer, other network services



Still Not Good Enough

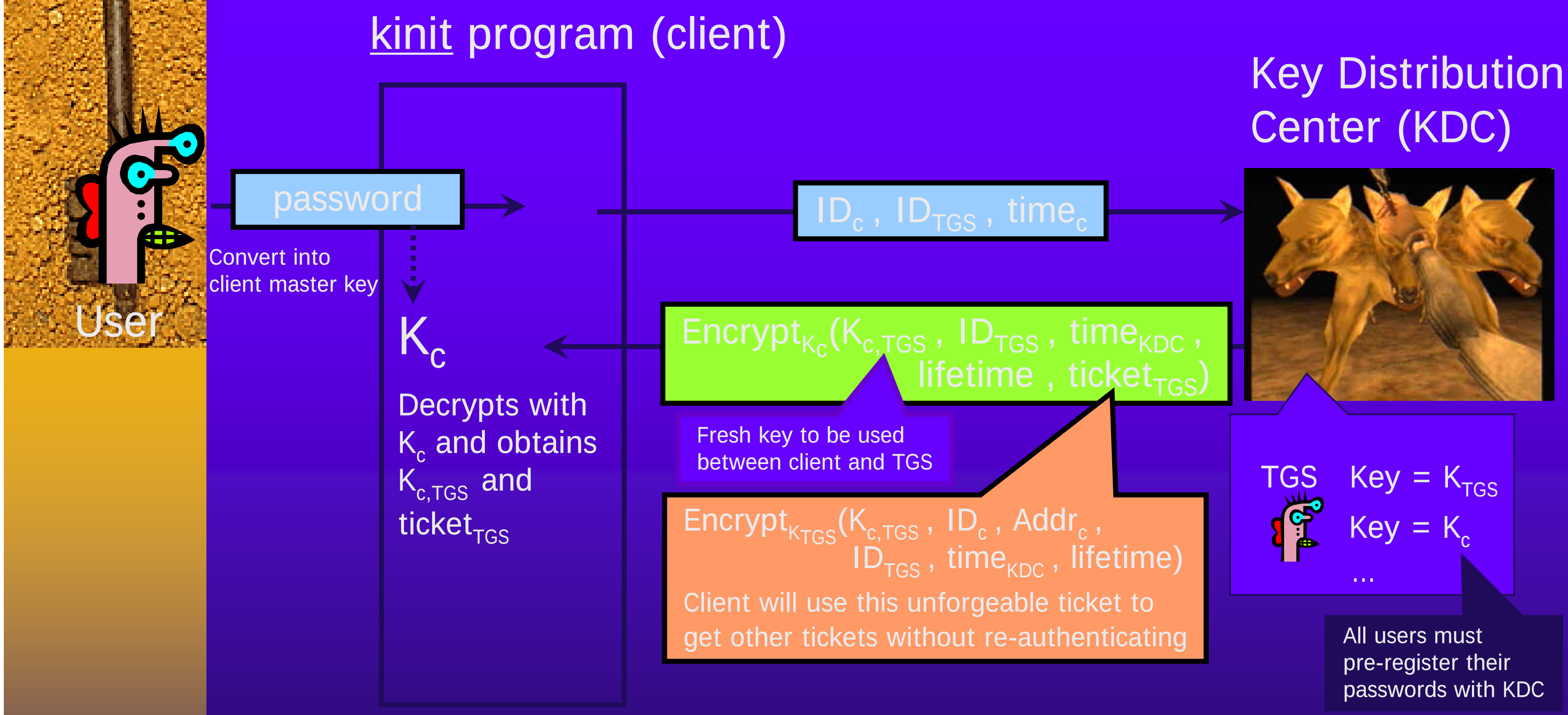
- Ticket hijacking
  - Malicious user may steal the service ticket of another user on the same workstation and use it
    - IP address verification does not help
  - Servers must be able to verify that the user who is presenting the ticket is the same user to whom the ticket was issued
- No server authentication
  - 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



Symmetric Keys in Kerberos

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

“Single Logon” Authentication



kinit program (client)

Key Distribution Center (KDC)

User

password

Convert into client master key

K_c

ID_c , ID_{TGS} , $time_c$

$Encrypt_{K_c}(K_{c,TGS}$, ID_{TGS} , $time_{KDC}$, lifetime , $ticket_{TGS})$

Fresh key to be used between client and TGS

Decrypts with K_c and obtains $K_{c,TGS}$ and $ticket_{TGS}$

$Encrypt_{K_{TGS}}(K_{c,TGS}$, ID_c , $Addr_c$, ID_{TGS} , $time_{KDC}$, lifetime)

Client will use this unforgeable ticket to get other tickets without re-authenticating

TGS Key = K_{TGS}

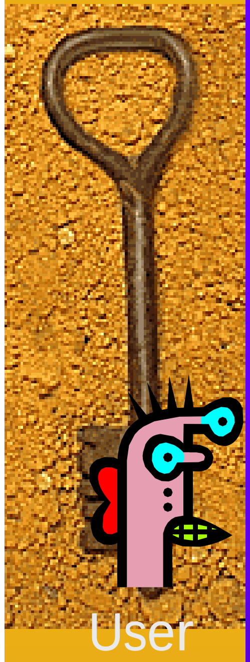
Key = K_c

…

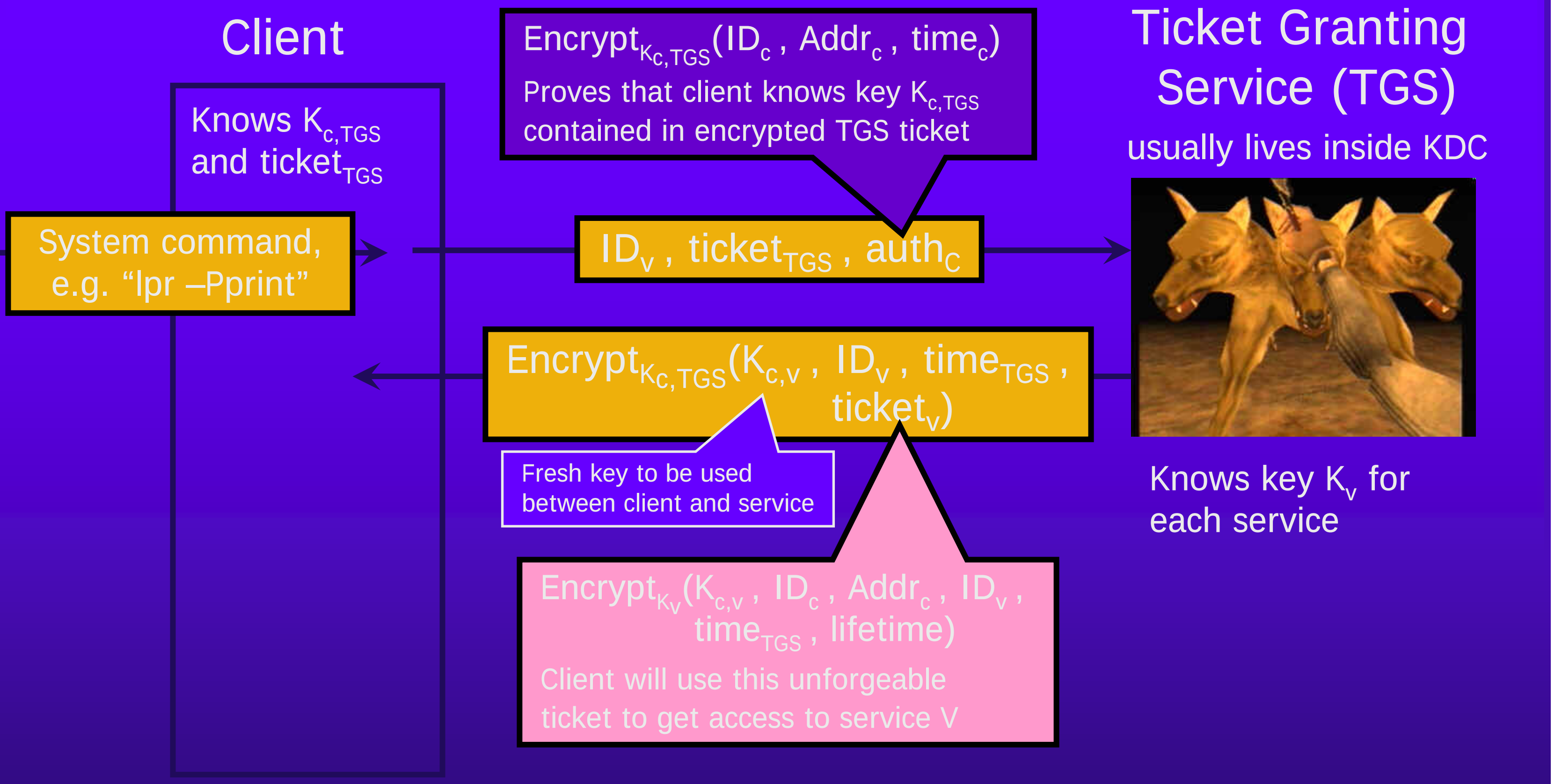
All users must pre-register their passwords with KDC

- 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



User



Client

Knows $K_{c,TGS}$ and $ticket_{TGS}$

System command, e.g. “lpr –Pprint”

$Encrypt_{K_{c,TGS}}(ID_c , Addr_c , time_c)$

Proves that client knows key $K_{c,TGS}$ contained in encrypted TGS ticket

ID_v , $ticket_{TGS}$, $auth_C$

Ticket Granting Service (TGS)

usually lives inside KDC

$Encrypt_{K_{c,TGS}}(K_{c,v} , ID_v , time_{TGS} , ticket_v)$

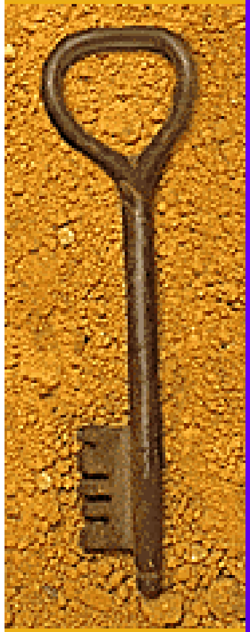
Fresh key to be used between client and service

Knows key K_v for each service

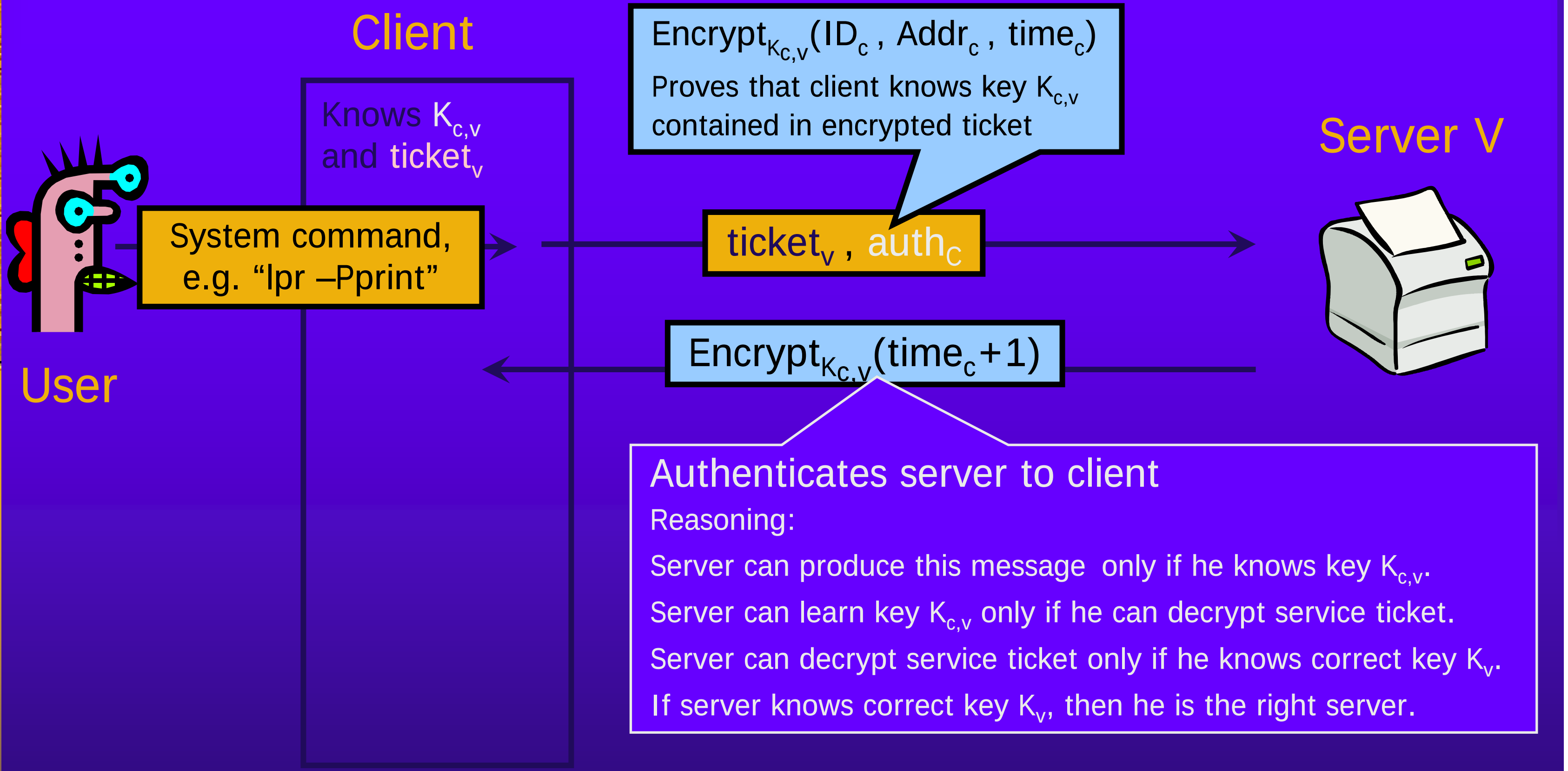
$Encrypt_{K_v}(K_{c,v} , ID_c , Addr_c , ID_v , time_{TGS} , lifetime)$

Client will use this unforgeable ticket to get access to service V

- 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.)



Obtaining Service



Client

Knows $K_{c,v}$ and ticket_v

User

System command, e.g. “lpr –Pprint”

$\text{Encrypt}_{K_{c,v}}(\text{ID}_c , \text{Addr}_c , \text{time}_c)$

Proves that client knows key $K_{c,v}$ contained in encrypted ticket

ticket_v , auth_C

Server V

$\text{Encrypt}_{K_{c,v}}(\text{time}_c+1)$

Authenticates server to client

Reasoning:

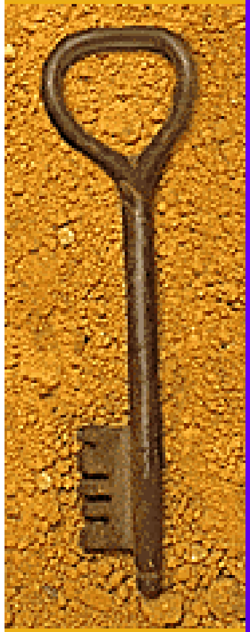
Server can produce this message only if he knows key $K_{c,v}$.

Server can learn key $K_{c,v}$ only if he can decrypt service ticket.

Server can decrypt service ticket only if he knows correct key K_v.

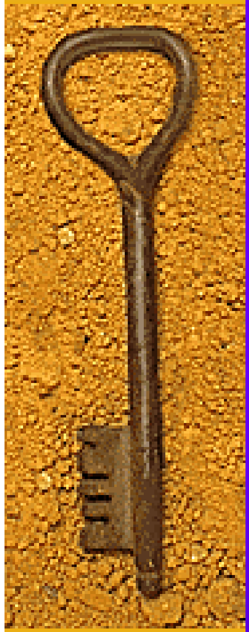
If server knows correct key K_v, then he is the right server.

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



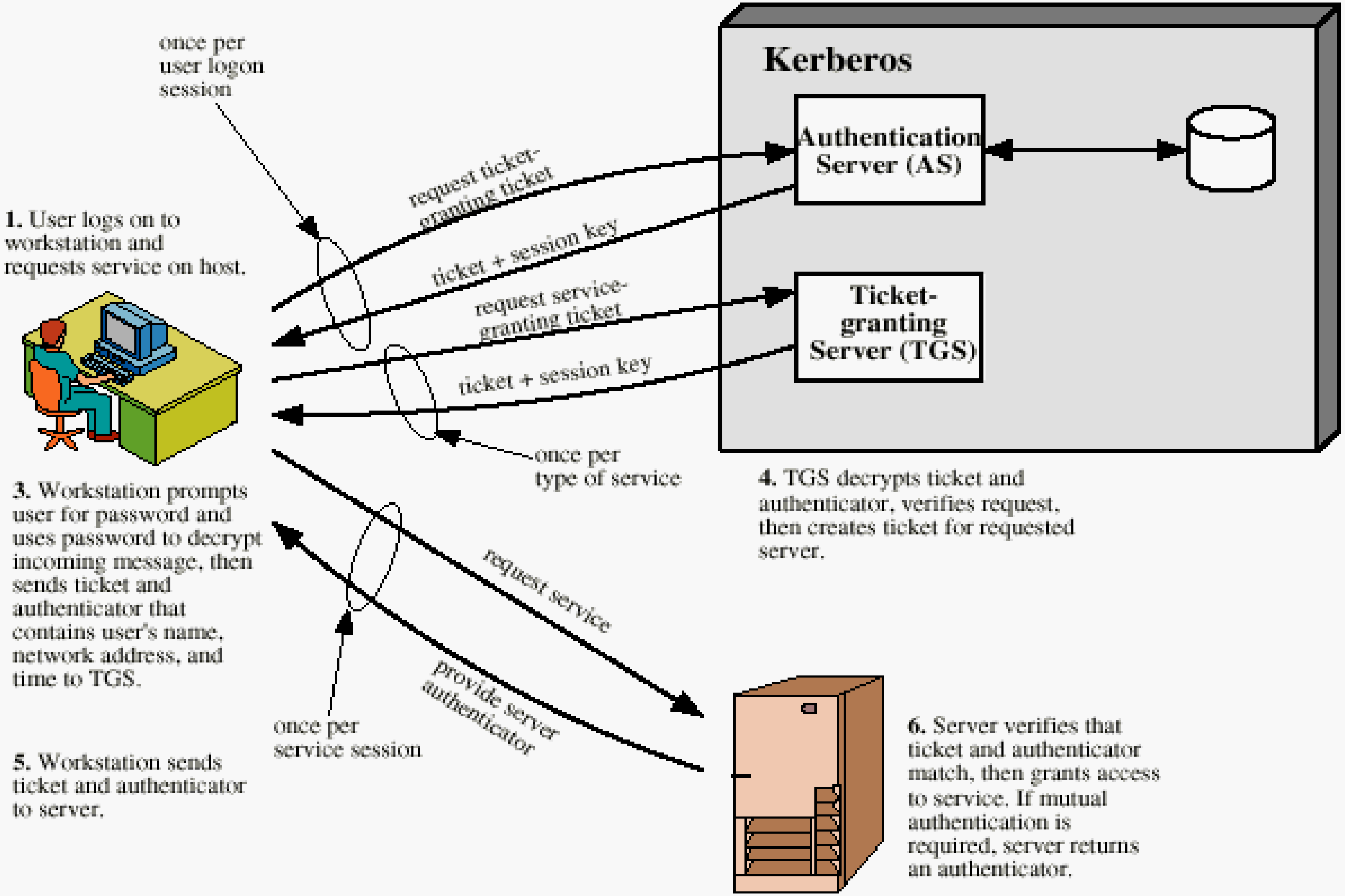
Kerberos in Large Networks

- ♦ 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
  - – N(N-1)/2 key exchanges for full N-realm interoperation



Summary of Kerberos

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.



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

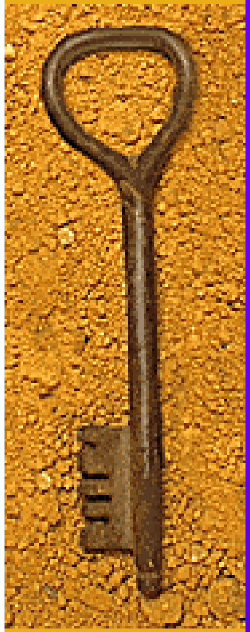
1. User logs on to workstation and requests service on host.

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.

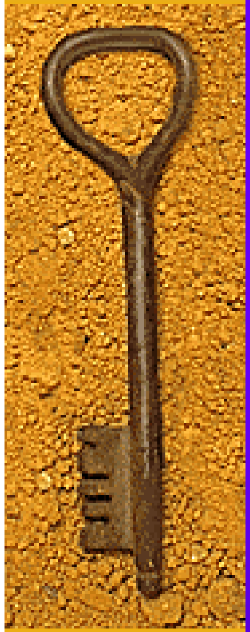
5. Workstation sends ticket and authenticator to server.

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



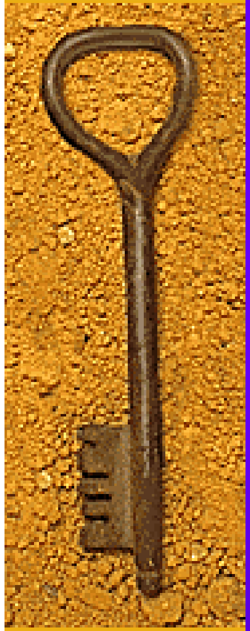
Important Ideas in Kerberos

- Use of short-term session keys
  - Minimize distribution and use of long-term secrets; use them only to derive short-term session keys
  - Separate short-term key for each user-server pair
    - But multiple user-server sessions reuse 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



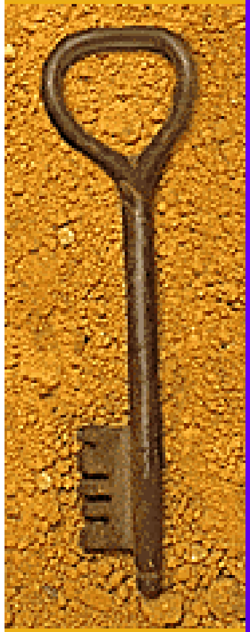
Problematic Issues

- 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 be better
- Same user-server key used for all sessions
- Homebrewed PCBC mode of encryption
  - Tries to combine integrity checking with encryption
- Extraneous double encryption of tickets
- No ticket delegation
  - Printer can’t fetch email from server on your behalf



Kerberos Version 5

- Better user-server authentication
  - Separate subkey for each user-server session instead of re-using the session key contained in the ticket
  - Authentication via subkeys, not timestamp increments
- Authentication forwarding
  - Servers can access other servers on user's behalf
- Realm hierarchies for inter-realm authentication
- Richer ticket functionality
- Explicit integrity checking + standard CBC mode
- Multiple encryption schemes, not just DES



Practical Uses of Kerberos

- Email, FTP, network file systems and many other applications have been kerberized
  - Use of Kerberos is transparent for the end user
  - Transparency is important for usability!
- Local authentication
  - login and su in OpenBSD
- Authentication for network protocols
  - rlogin, rsh, telnet
- Secure windowing systems
  - xdm, kx