

# Information Security

## CS3002

### (Sections BDS-7A/B)

## Lecture 21

Instructor: Dr. Syed Mohammad Irteza

Assistant Professor, Department of Computer Science

30 October, 2024

# Types of Authentication

- There are two basic types of authentication: non-repudiable and repudiable.
- *Repudiable Authentication* – involves factors , “what you know” and “what you have, ”that can present problems to the authenticator because the *information presented can be unreliable* because such factors suffer from several well-known problems including the fact that possessions can be lost, forged, or easily duplicated.
- *Non-repudiable Authentication* – involves *characteristics whose proof of origin cannot be denied*. Such characteristics include biometrics like iris patterns, retinal images, and hand geometry and they positively verify the identity of the individual

# Authentication Mechanisms

In general, authentication takes one of the following three forms:

- *Basic authentication involving a server:*
  - The server maintains a user file of either passwords and usernames or some other useful piece of authenticating information. This information is always examined before authorization is granted.
- *Challenge-response:*
  - In which the server or any other authenticating system generates a challenge to the host requesting for authentication and expects a response.
- *Centralized authentication:*
  - In which a central server authenticates users on the network and in addition also authorizes and audits them

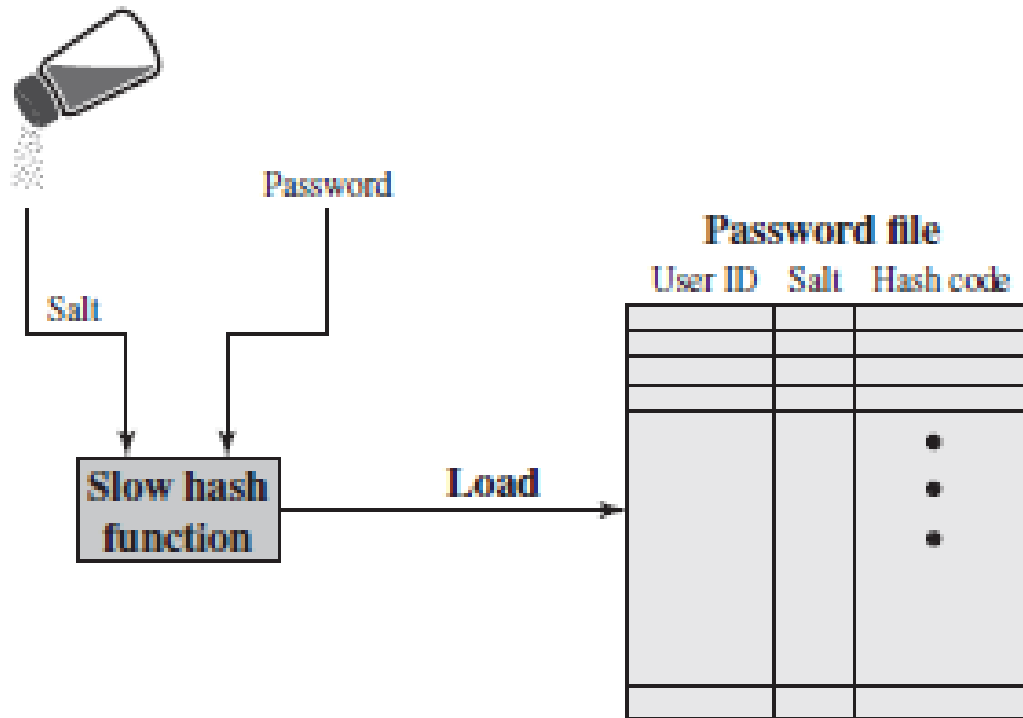
# Password-based Authentication

- secret = the user password
- (client) create and transmit proof
  - $F = I$  (the identity function)
  - i.e. proof = password (cleartext!)
- (server) verify the proof:
  - case #1:  $F = I$  (the identity function)
    - server knows all passwords in cleartext(!)
    - access control: proof = password ?
  - case #2:  $F =$  one-way hash (that is a digest)
    - server knows the passwords' digests, HUID
    - access control:  $f(\text{proof}) = \text{HUID}$  ?

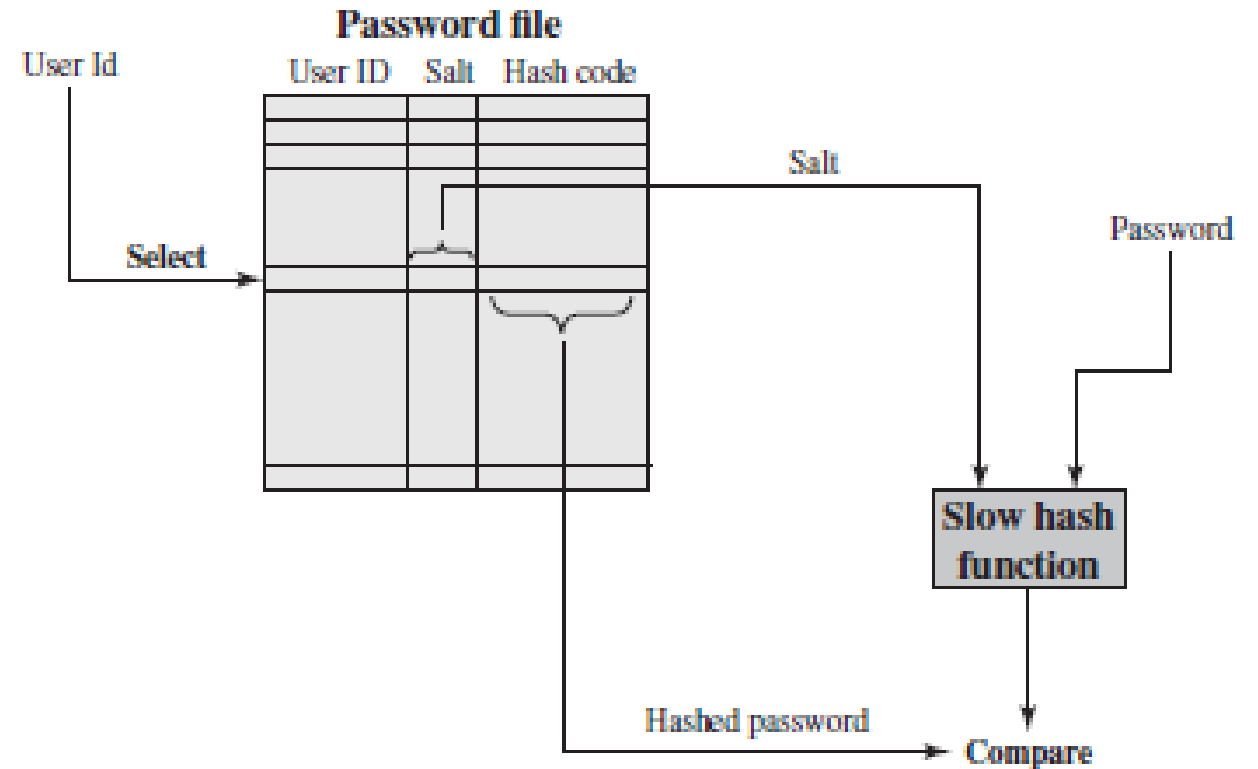
# Passwords

- Authentication based on alphanumeric characters or numbers
  - *Benefits*
    - Easy to remember (if only for one system)
  - *Disadvantages*
    - User-side password storage:
      - Post-It!
      - Client-side password manager or wallet
    - Password guessable (my son's name)
    - Password readable during transmission
    - Server-side password storage issues (hashing is must)
      - 35% passwords identified using dictionary attack
      - Use "salt"
    - Shoulder surfing
    - Using same password in multiple places

# Using “salt” and hash: Fig 3.3, UNIX Password Scheme



(a) Loading a new password



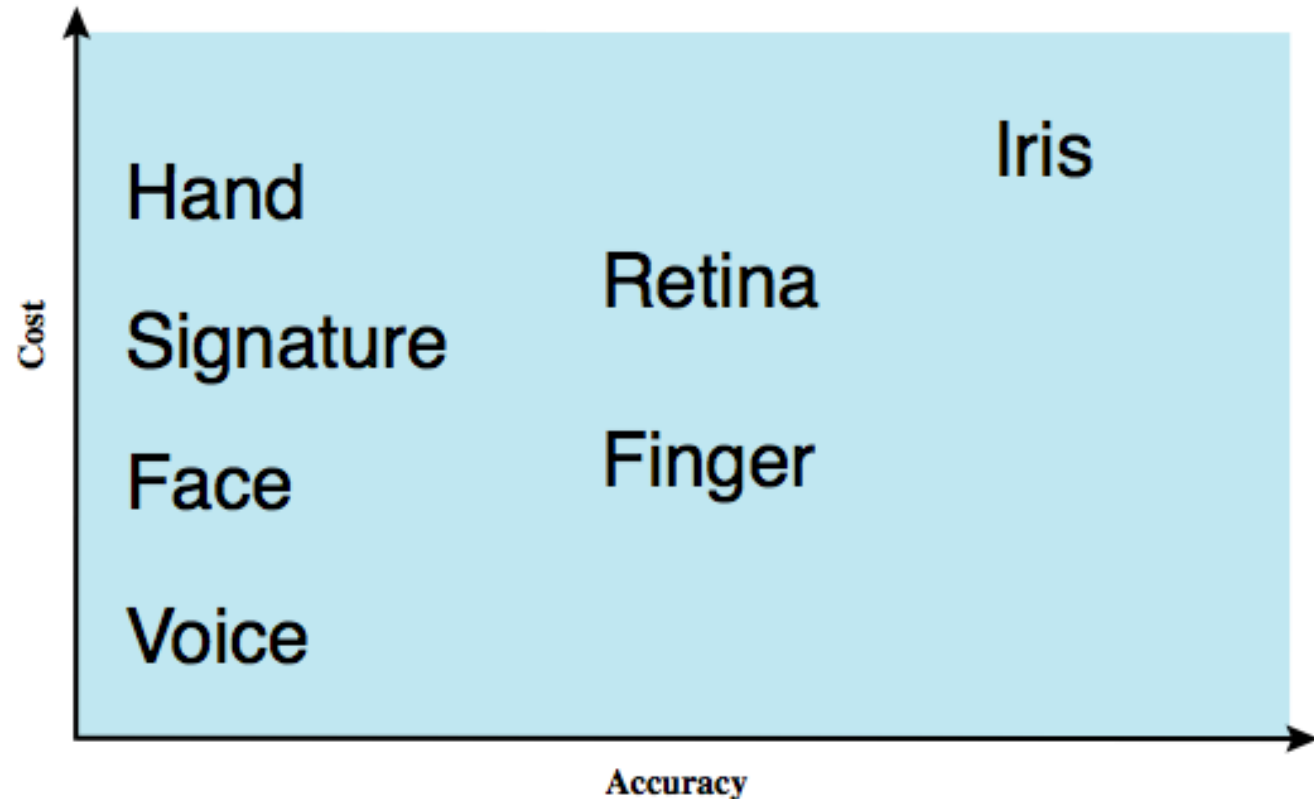
### (b) Verifying a password

# Using “salt” and hash

- For each user ID:
  - Create/ask the password
  - Generate a **random salt** (should contain rarely used or control characters)
- Compute HP = hash(password | salt)
- Store the triples {UID, HP, salt<sub>UID</sub>}
- Advantages:
  - Prevents duplicate passwords from being visible in the password file (*different HP for users having the same password*)
  - Increases the difficulty of *offline dictionary attacks*
  - Nearly impossible to tell if a person used the *same password on multiple systems*

# Biometric Authentication

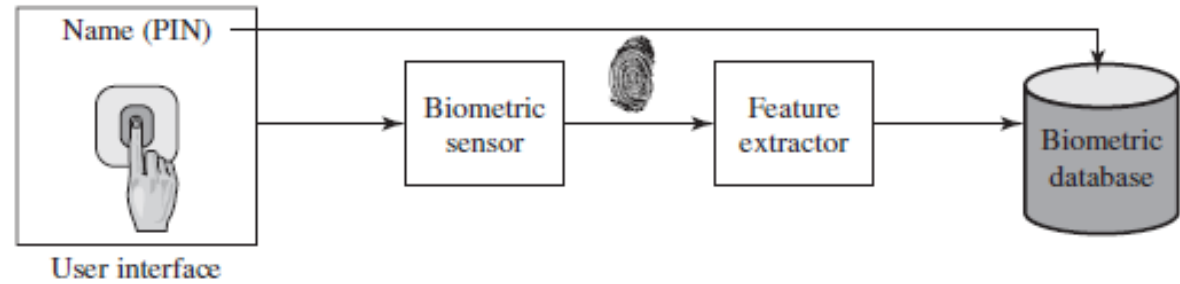
- Authenticate user based on one of their physical characteristics:
  - Facial
  - Fingerprint
  - Hand geometry
  - Retina pattern
  - Iris
  - Signature
  - Voice



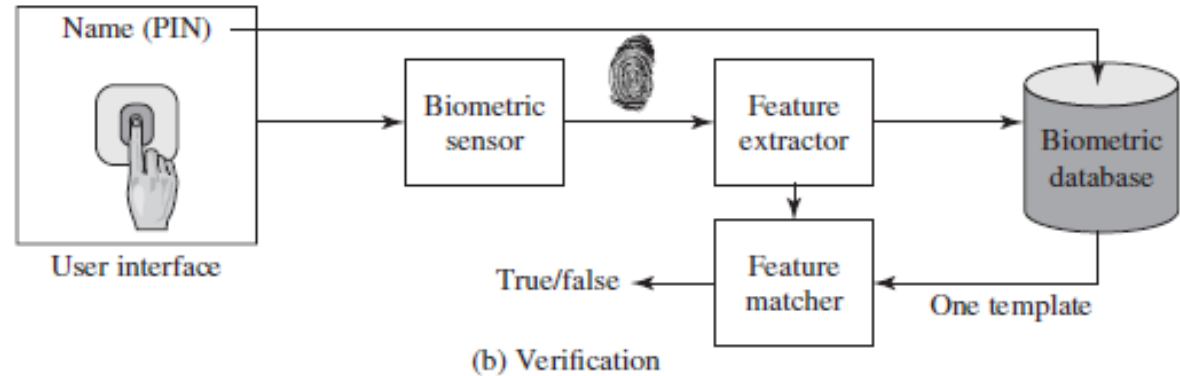


# Operation of a biometric system

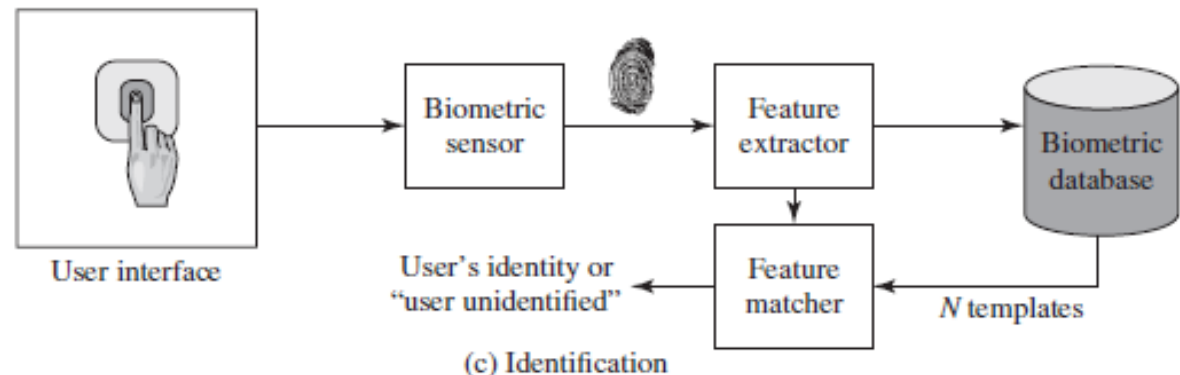
- *Verification* is analogous to user login via a smart card and a PIN
- *Identification* is biometric info but no IDs; system compares with stored templates



(a) Enrollment



(b) Verification



(c) Identification

# Problems with biometric systems

- FAR = *False Acceptance Rate*
- FRR = *False Rejection Rate*
- FAR and FRR may be partly tuned but they heavily depend on the cost of the device
- variable biological characteristics:
  - Finger wound
  - Voice altered due to emotion or injury
  - retinal blood pattern altered due to alcohol or drug

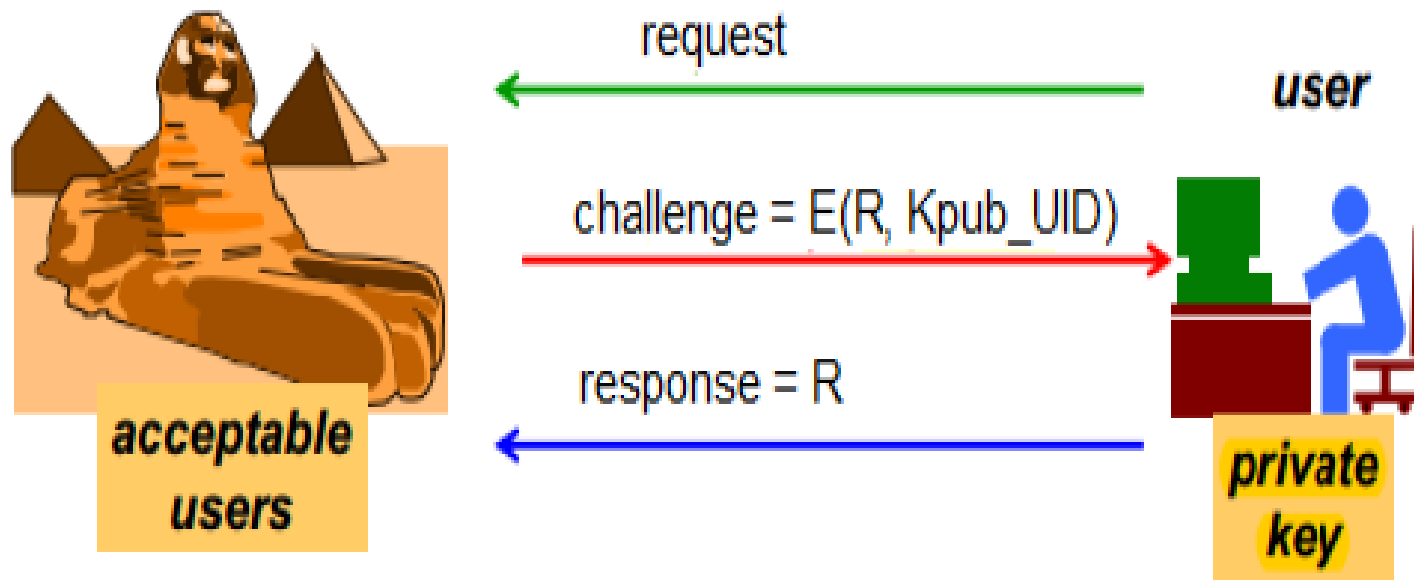
# Symmetric challenge-response authentication

- a challenge (typically a random nonce) is sent to the user ...
- ... who replies with the solution after a computation
- involving the shared secret and the challenge
- the server must know the secret in clear
- often  $R$  is a hash function (can't be encryption)

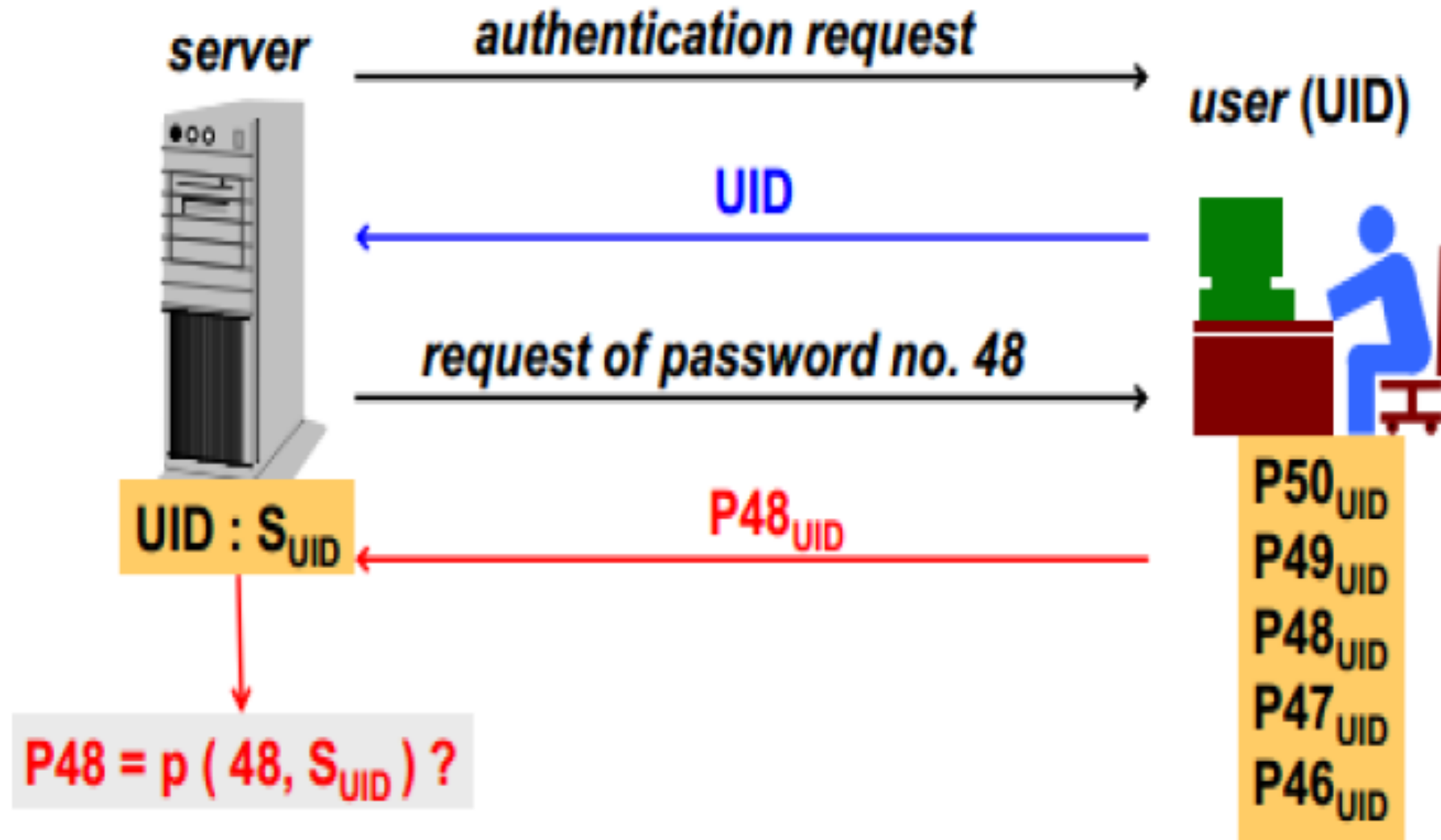


# Asymmetric challenge response system

- a random number  $R$  is encrypted with the user's public key ...
- and the user replies by sending  $R$  in clear thanks to its knowledge of the private key



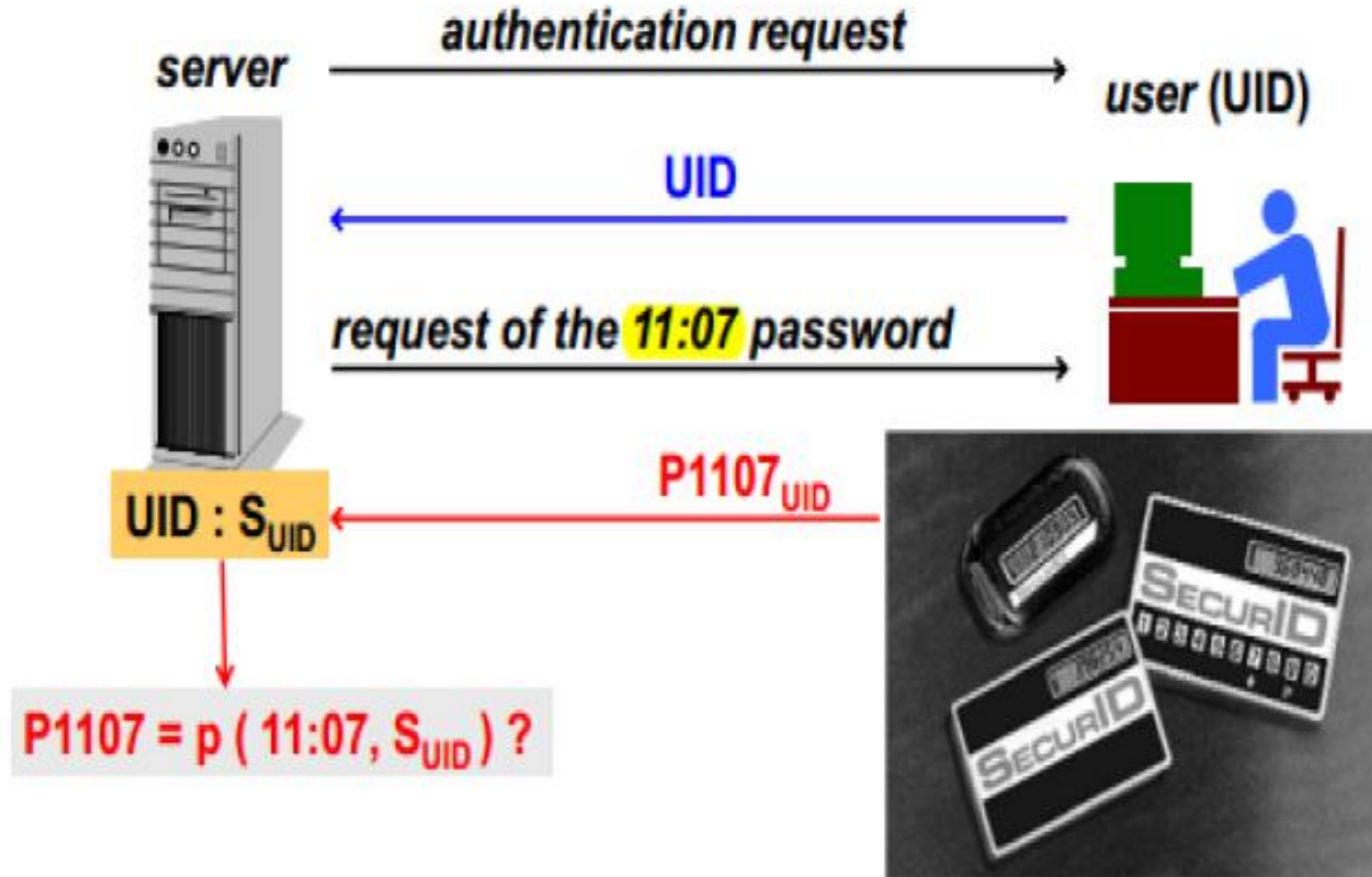
# One-time Passwords



# One-time Passwords



# Passwords (one-time token based)

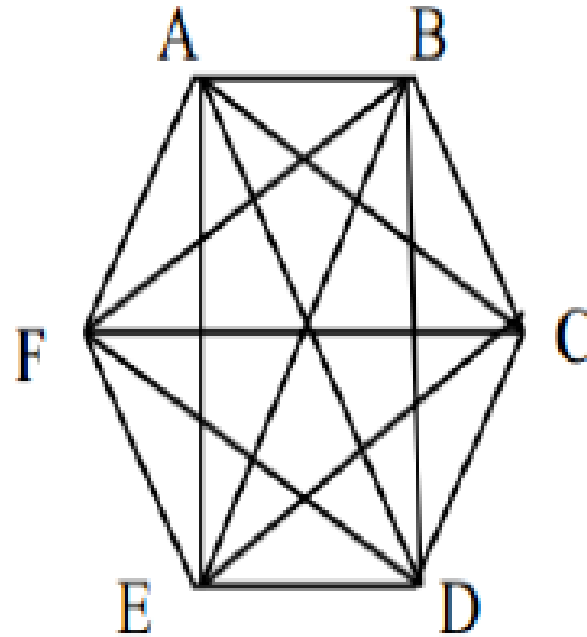


# SecurID: architecture

- Invented and patented by Security Dynamics
- Time-based synchronous OTP technique:
  - $P_{UID}(t) = h(S_{UID}, t)$
- The client sends:
  - user , PIN , token-code (seed, time)
- Based on user name and PIN the server verifies against three possible token-codes:
  - TC-1, TC-0, TC+1
- More than once if there is a drift of more than one minute
- Wrong authentication attempts limited



# Secret keys for N-system network



- n system need  $n(n-1)/2$  pairs of secret keys
- Each system remembers  $n-1$  keys
- If a new system comes in n new key are generated
- If a system leaves,  $n-1$  keys are removed

# PKI and Certificate Authorities

- Certificate consists of:
  - A public key plus a User ID of the key owner
  - Signed by a third party trusted by community
  - Often govt/bank certificate authority (CA)
- Users obtain certificates from CA
  - Create keys & unsigned certificate, gives to CA, CA signs certificate & attaches signature, returns to user
- Other users can verify certificate
  - Checking signature on certificate using CA's public key

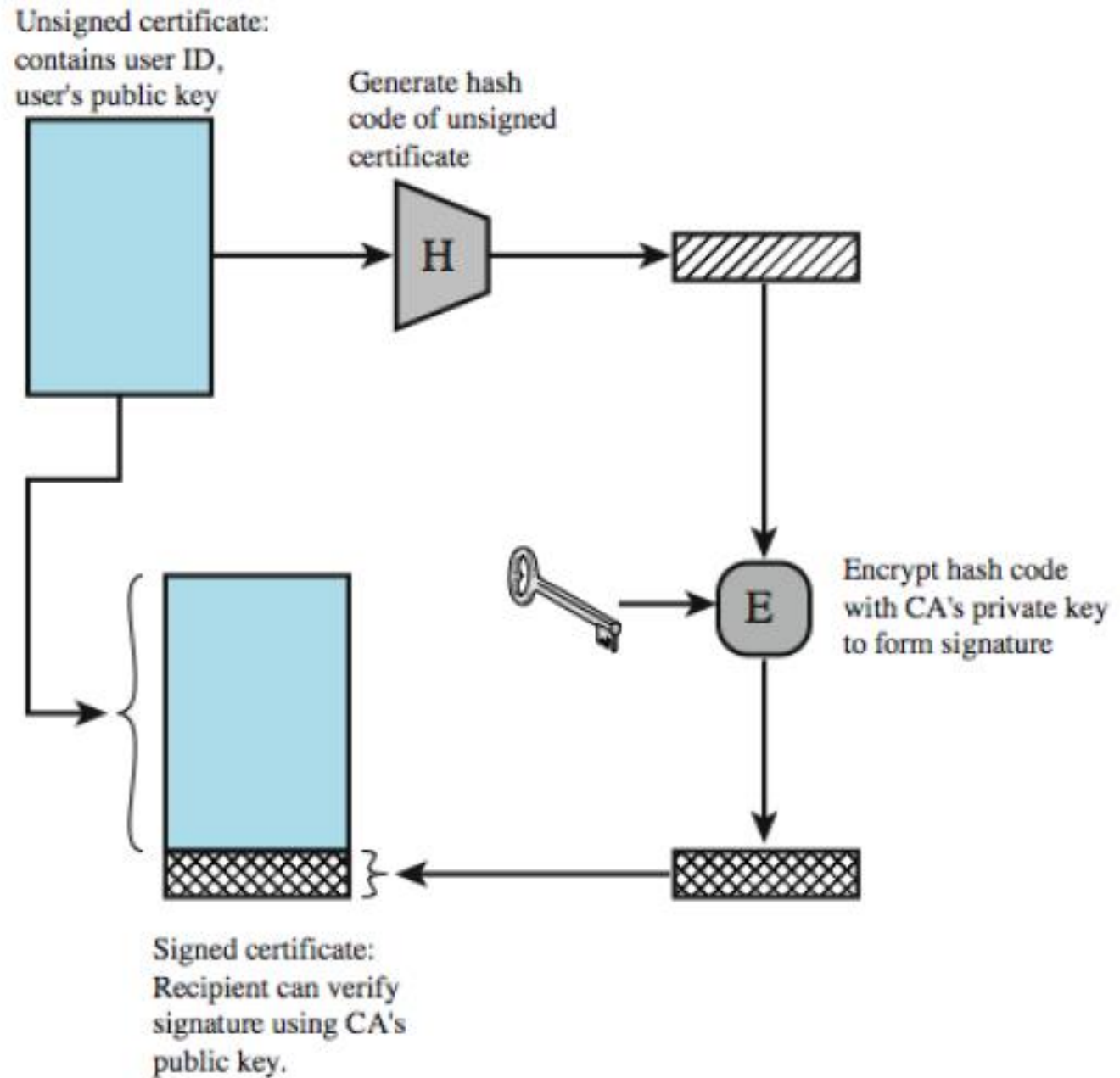
# Common Key Steps

1. User software creates a pair of keys: private and public
2. Clients prepares unsigned certificate that includes user ID and public key
3. User provides unsigned certificate to a CA
4. CA creates a signature:
  - I. Creates a hash code of the unsigned certificate
  - II. Encrypts the hash code with the CA's private key
5. CA attaches the signature to unsigned certificate to make signed certificate

# Key Steps (continued)

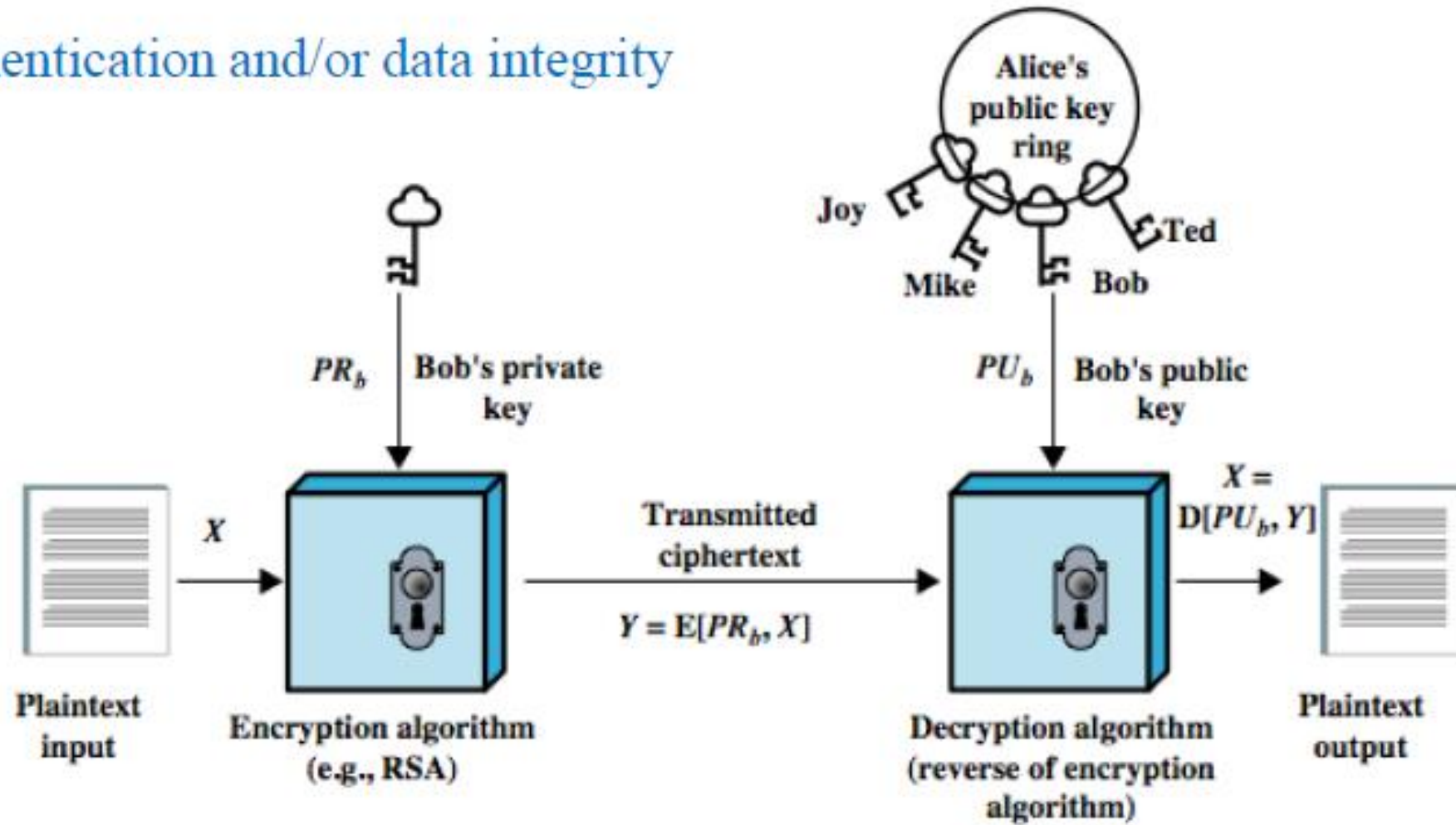
6. CA returns the signed certificate to the client
7. Client may provide signed signature to other users
8. Any user may verify the certificate
  - i. Calculate the hash code of certificate (exclude signature)
  - ii. Decrypt signature using CA's public key
  - iii. Compare the two

# Public Key Certificates



# Public Key Authentication

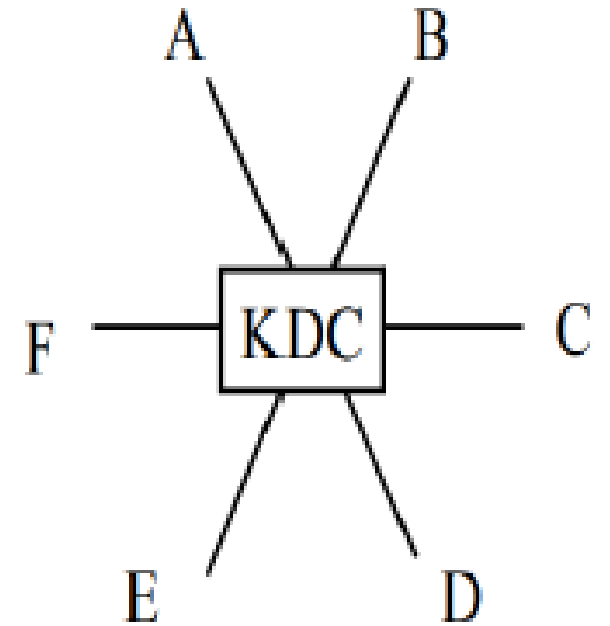
Authentication and/or data integrity



(b) Authentication

# Key Distribution Center (KDC)

- Each node is configured with KDC's key.
- KDC has all the keys.
- $A \leftrightarrow B$  communication?
  - KDC sends a key  $K_{AB}$  encrypted with A's key to A and B's key to B.
- Issues:
  - If KDC is compromised, all systems are compromised
  - KDC is single point of failure or performance bottleneck
  - KDC has to be on-line all the time. Replication!



# Kerberos



- Network authentication protocol
- Based on Trusted Third Party (TTP) – KDC
- invented by MIT for project Athena

[Kerberos: The Network Authentication Protocol](#)



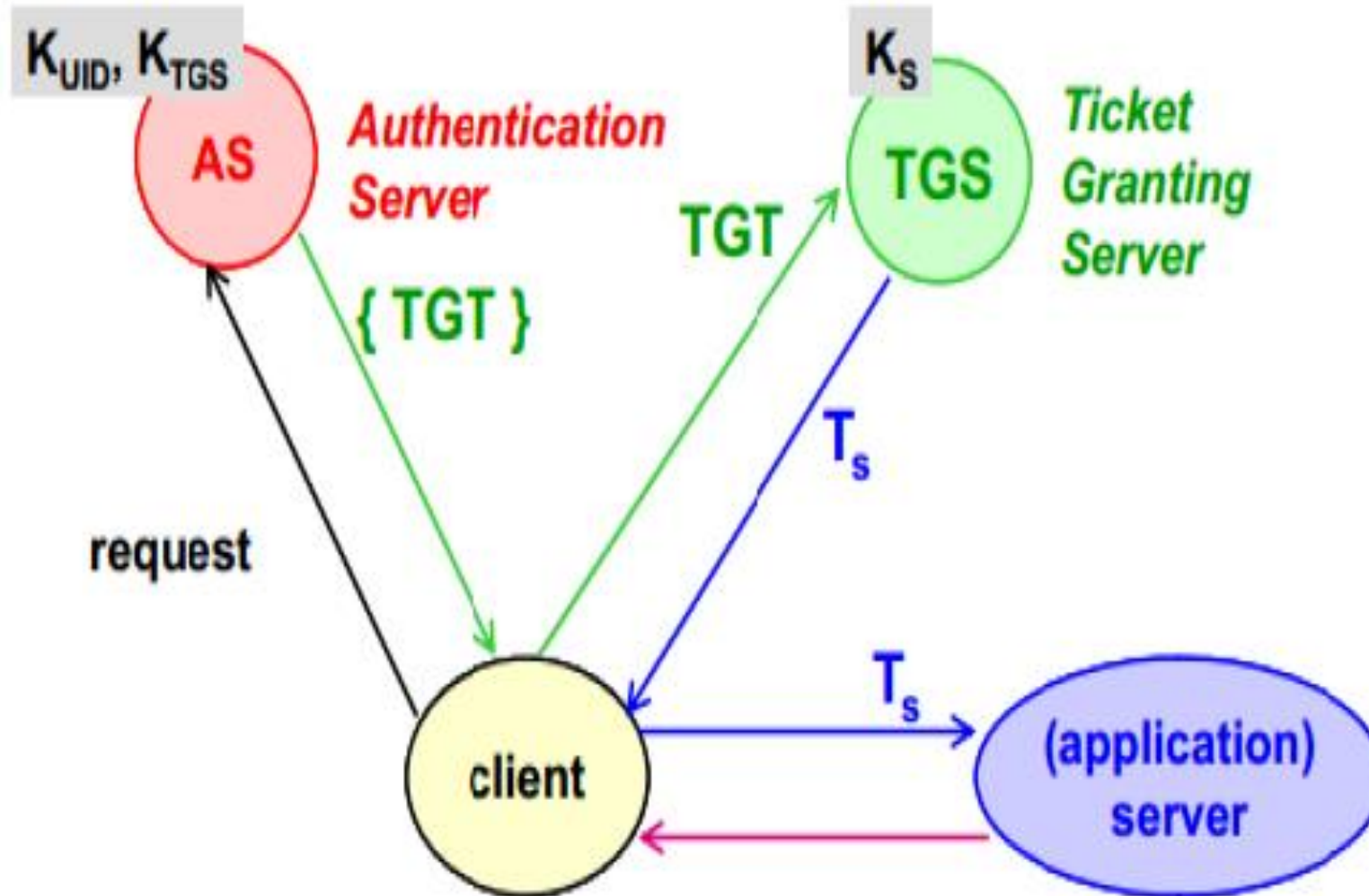
- Named after Greek mythological character “Cerberus” Three headed dog protecting the entrance of Hades
- Used by popular operating systems and servers
- Protect against eavesdropping and firewall limitation to users and replay attacks



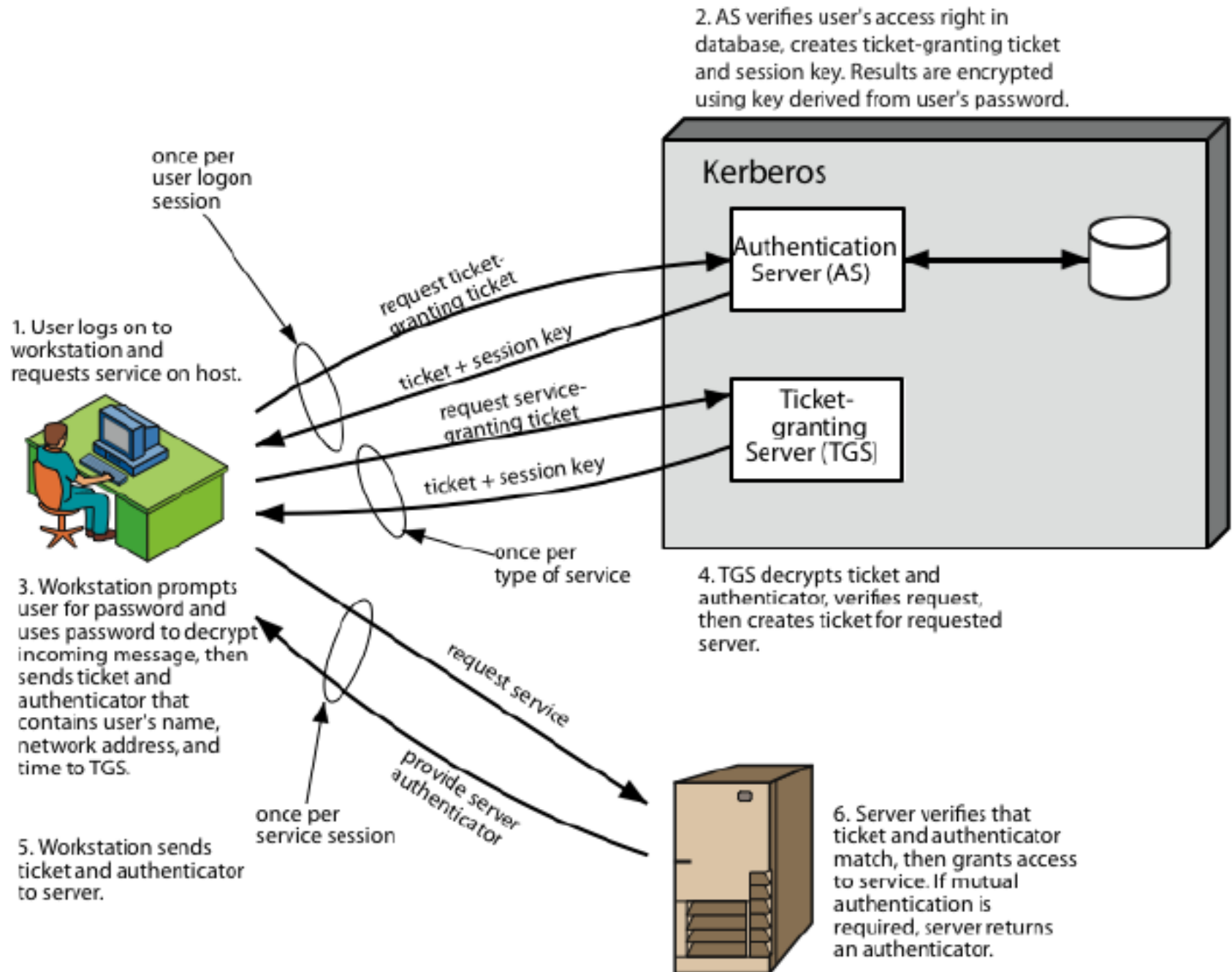
# Kerberos overview

- **Authentication server** authenticates a user to a specific service in the network
- TGS: **Ticket Granting Server**, grants ticket to the user
- **Authentication server** and **TGS** can be the same system. They work as a single unit.
- **Application Server** provides the service to the user
- The *client/user*, *Auth. Server & TGS*, *Application server* are the 3 heads of Kerberos!

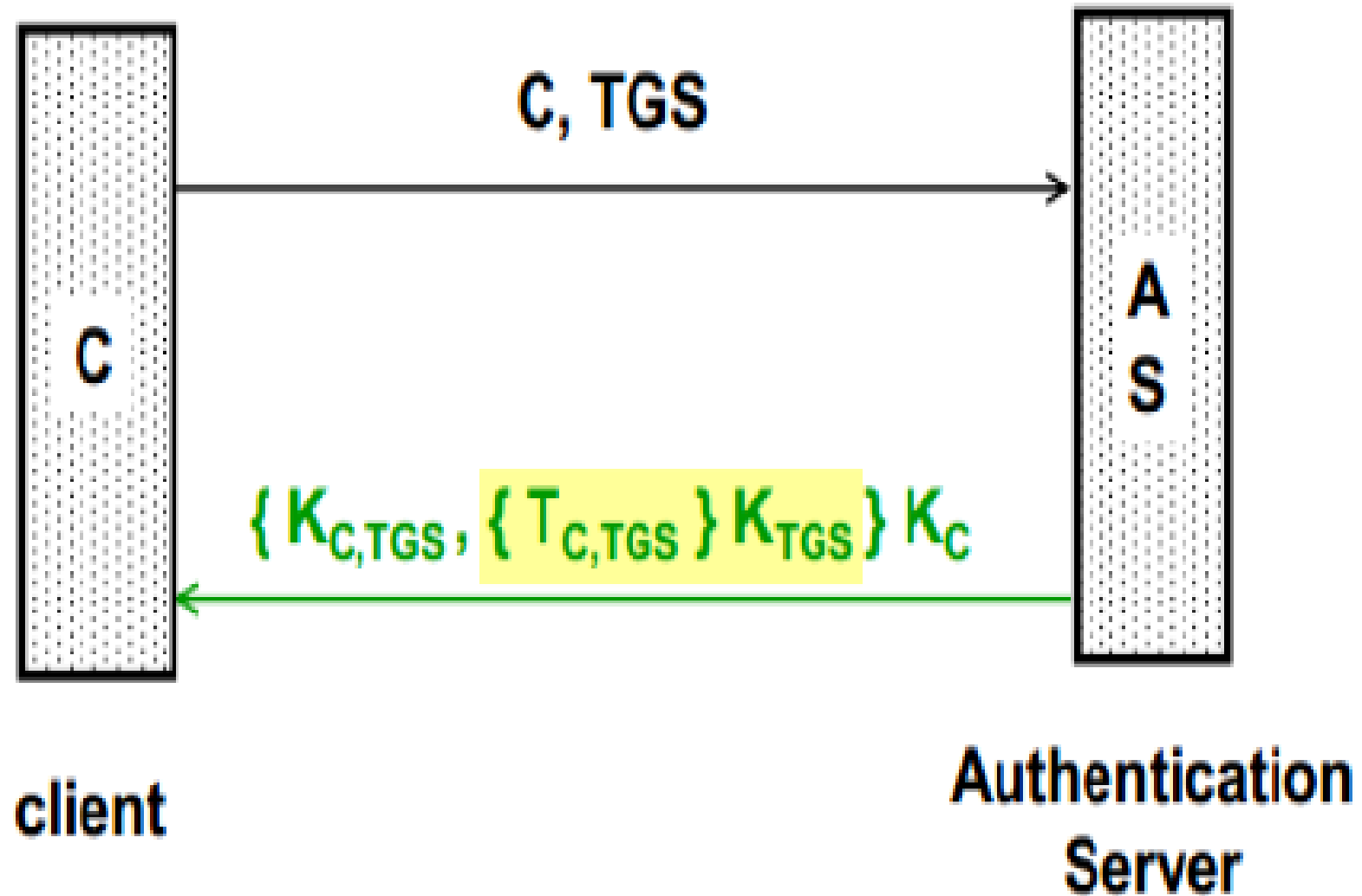
# Kerberos high-level view



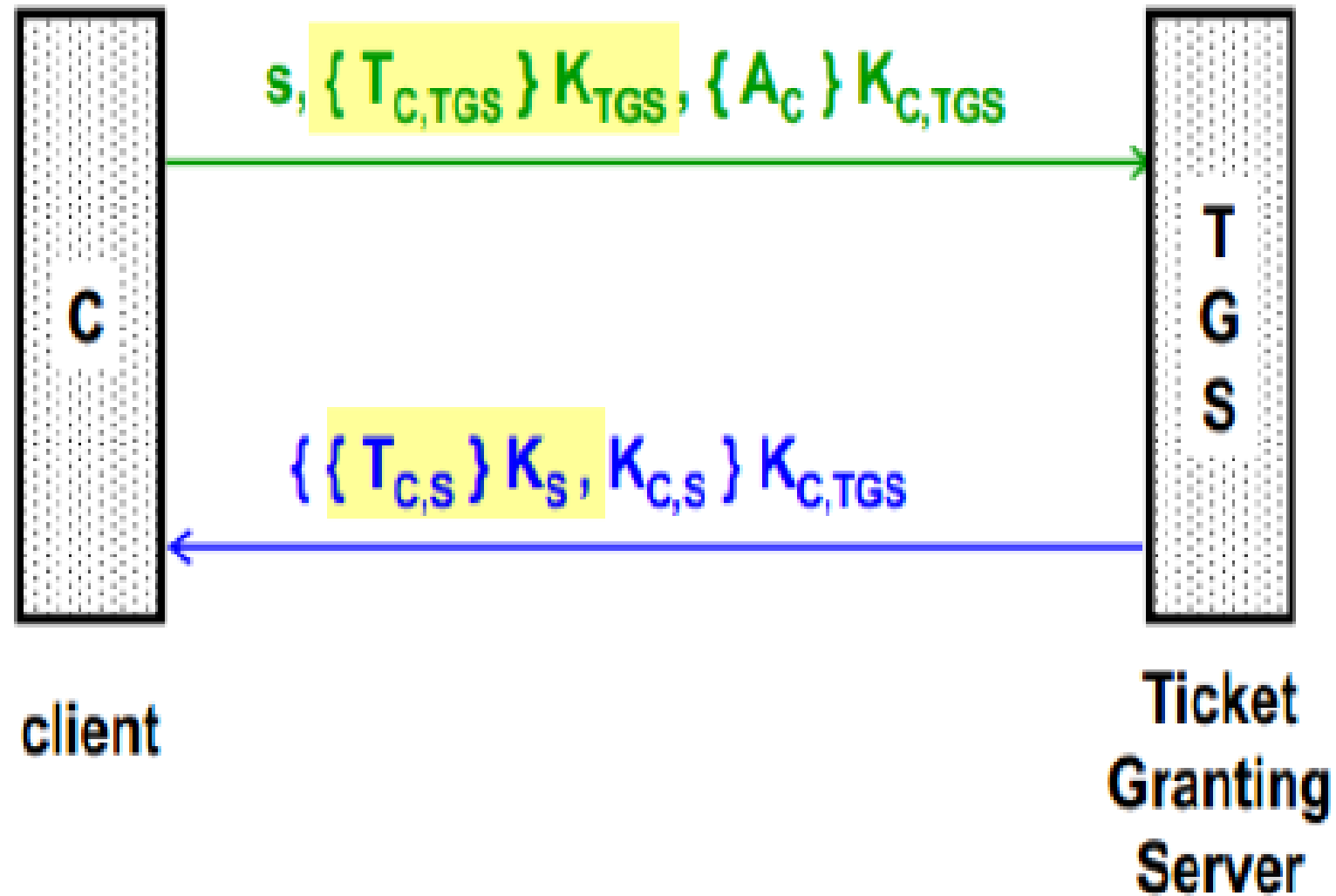
# Kerberos Protocol



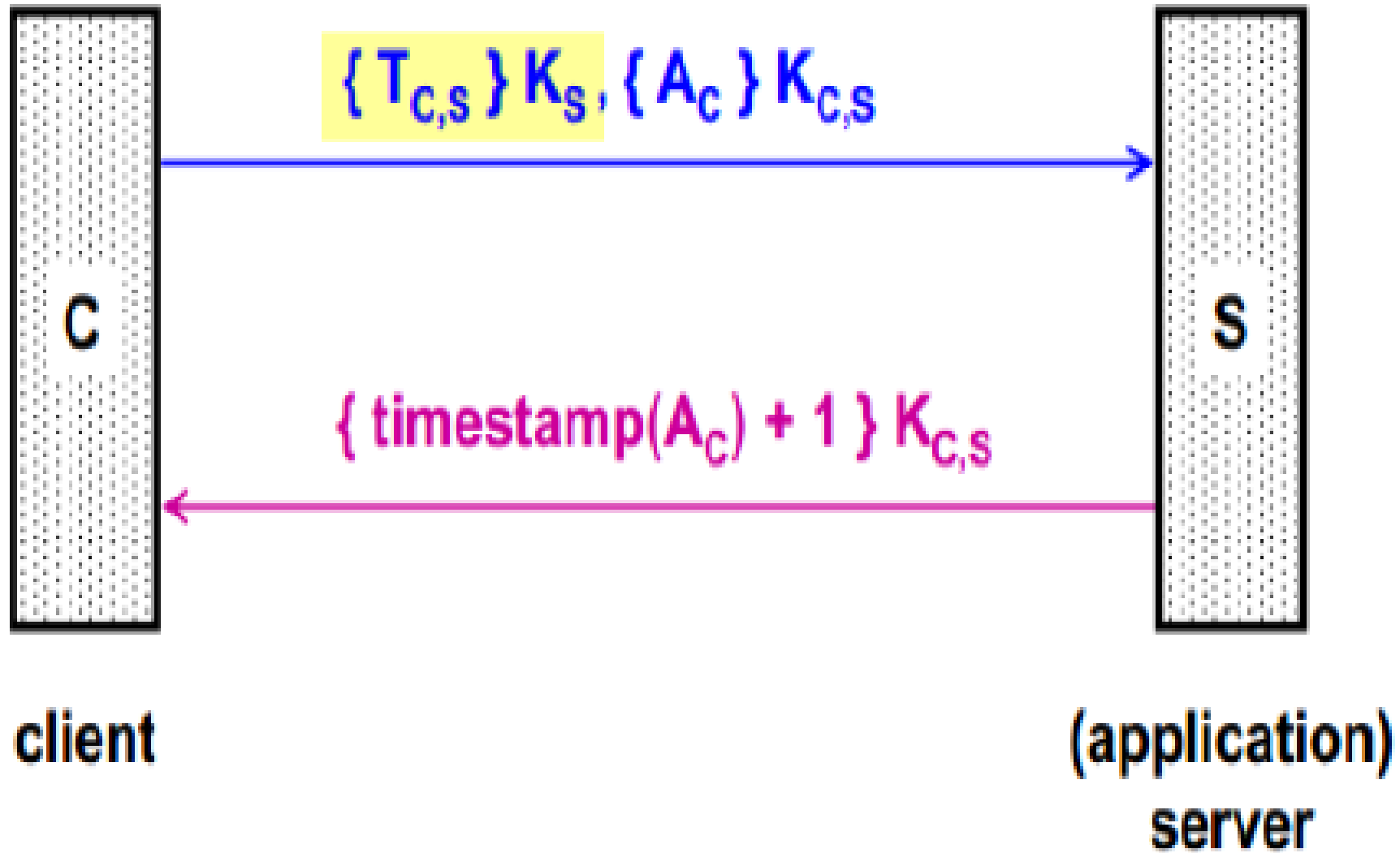
# TGT Request



# Ticket Request



# Ticket use



# Kerberos: Component Breakdown

- $K_{C,TGS}$ :
  - This is the session key that is shared between the client and the TGS. It is generated by the Authentication Server (AS) and included in the Ticket-Granting Ticket (TGT) that the client receives during the initial authentication process.
- $\{T_{C,TGS}\} K_{TGS}$ :
  - This is the encrypted message that contains the client's identity (TC) and the name of the desired service (TGS), encrypted using the TGS's secret key ( $K_{TGS}$ ). The TGS's secret key is known only to the TGS and the Kerberos Key Distribution Center (KDC).
- $\{K_{C,TGS}, \{T_{C,TGS}\} K_{TGS}\}$ :
  - This is the message that combines the session key and the encrypted ticket information. It is encrypted using the session key  $K_{C,TGS}$  that is shared between the client and the TGS.
- $K_C$ :
  - This is the final layer of encryption that is applied to the entire message using the session key  $K_C$  that is shared between the client and the service. The final encrypted message is then sent to the TGS.

# Other Authentication Systems

- OATH (open authentication)
  - Interoperability of authentication systems based on OTP, both symmetric and asymmetric
- SSO (single sign-on)
  - Single credential multiple services

