Dr. Ammar Haider
Assistant Professor
School of Computing

CS3002 Information Security

User Authentication

Reference: Stallings SPP chap 3, 23

Authentication

- Verifying the identity of the user connecting to a system (authenticator)

- Once authenticated, system will proceed to check **authorization**
  - what resources user is authorized to access
  - what privileges they have

Means of Authentication

I. **Something you know (knowledge)**
   - password, pin, security Q/A

II. **Something you have (possession)**
   - mobile number, token (code generator), smartcard

III. **Someone you are (biometrics)**
   - fingerprint, facial features, voice

IV. **Something you do (behaviour)**
   - typing rhythm, computer usage pattern, mouse movement, touch interaction, handwriting

- **Multifactor authentication**: combining more than one modes from above list

Authentication Types

- **Repudiable Authentication**
  - involves factors, “what you know” and “what you have,”
  - the information presented can be unreliable because such factors suffer from several well-known problems
  - e.g. passwords can be leaked, possessions can be lost, forged, or easily duplicated.

- **Non-repudiable Authentication**
  - involves characteristics whose proof of origin cannot be denied.
  - include biometrics like iris patterns, retinal images, hand geometry
  - they positively verify the identity of the individual.

Implementing Authentication

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

Password-based Authentication

- secret = user's password

- **User:** provides their identity *uid* and proof (password)
- **System:** verifies the proof
  - case #1:
    - system knows all user's secrets in cleartext (!!!)
    - to check: proof == $\text{secret}_{\text{uid}}$?
  - case #2: use one-way hash (digest)
    - system knows the digests of all user's secrets
    - to check: hash(proof) == $\text{digest}_{\text{uid}}$?

Passwords Pros

For users

- Easy to remember (if only for one system)

For system administrators

- Easiest to implement, compared to other auth methods
- Users are familiar with the concept, do not require training
- Lowest cost: no specialized hardware needed

Passwords Cons

For users

- can't remember too many passwords
  - either use same password for multiple services !!!
  - or use simple easy to remember passwords
    - but that makes them guessable (son's name!)
  - or use some form of password storage
    - post-it notes !!!
    - client-side password manager app (aka vault)

- vulnerable to shoulder surfing, phishing, social engineering

Passwords Cons

For system administrators

- password files are very frequently a stealing target for hackers. So, server-side password storage remains a challenge
  - hashing with a strong hash function is must
  - even hashed passwords are vulnerable to dictionary attacks

- password readable during transmission
  - encrypted channel is a must

Offline Password Cracking

- When hackers manages to steal the **password file** from a service's database, they immediately get a big dataset of hashed passwords.

- Since hash is a one-way function, attacker needs guesswork and a lot of computation to work out the actual passwords.
  1. make a guess g_i
  2. compute $h_i = \text{hash}(g_i)$
  3. search h_i in the file. If found, a hash has been cracked, g_i is the revealed password.
  4. loop back to step 1 until all hashes cracked

Offline Password Cracking

- The most naïve way of guessing is to **brute-force**.
  - try every possible combination of letters, numbers, and symbols to guess a target password
  - quickly becomes infeasible when target passwords are longer
- For a more selective brute forcing, attackers can use **mask attacks** – it assumes that target passwords follow a specific pattern
  - e.g. a common pattern is name and year (like sana2024). To crack such passwords, one can use a mask like ****#### where * is a lower case letter and # is a digit.
- Mask attacks greatly speed up the password search if certain characteristics of the password are known.

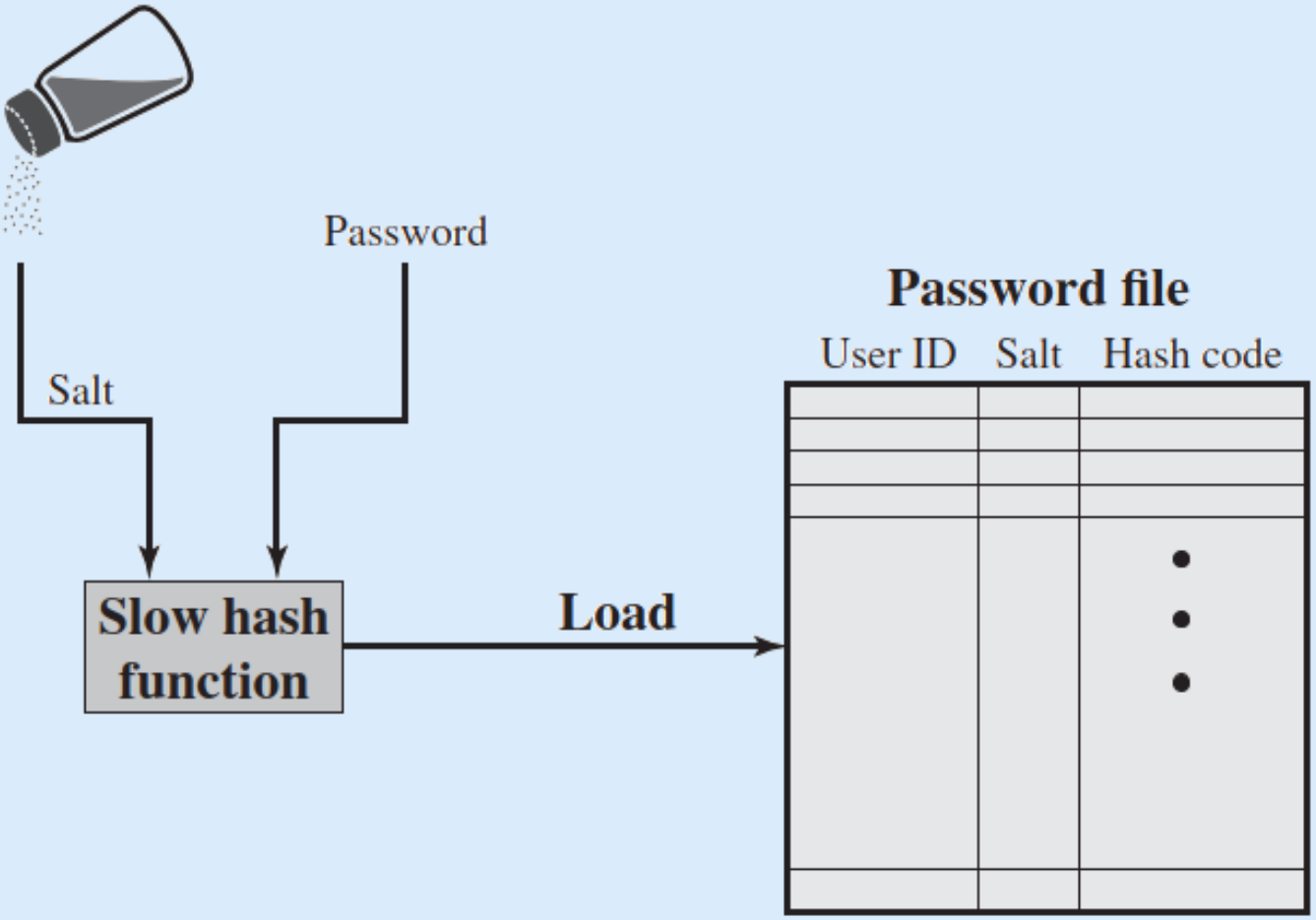
Offline Password Cracking

- Other type of offline cracking is **dictionary attack**, where the password guesses are drawn from a list of commonly-used or previously-leaked passwords.
  - Such a list is called *dictionary*
- These attacks require far less computing effort than brute forcing, but will only reveal passwords from the dictionary

Passwords: Salted storage

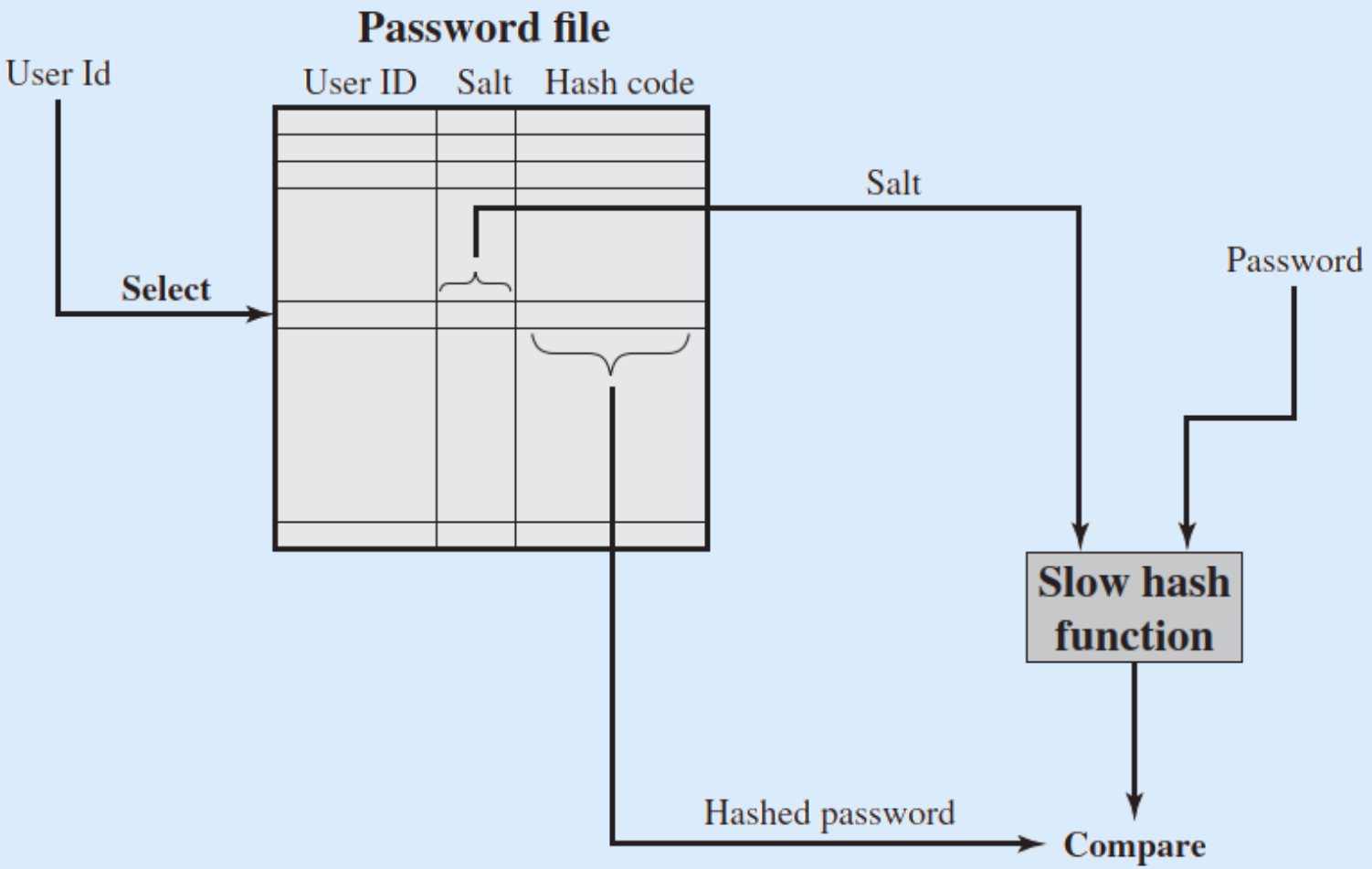
- To thwart dictionary attacks, passwords can be **salted**.
- At registration time, for each user UID:
  - create / ask the password
  - generate a salt (a new random value for each user)
- compute HP = hash (password | salt)
- store the triples { UID, HP, $salt_{UID}$ } in file

Passwords: Salted storage



(a) Loading a new password

Passwords: Salted storage



(b) Verifying a password

Passwords: Salted storage

Advantages of salting

- 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, since a unique salt is used for each user.
- Nearly impossible to tell if a person used the same password on multiple systems.

Remote User Authentication

- Local authentication is safer
  - Login to personal computer
  - PIN input at ATM machine

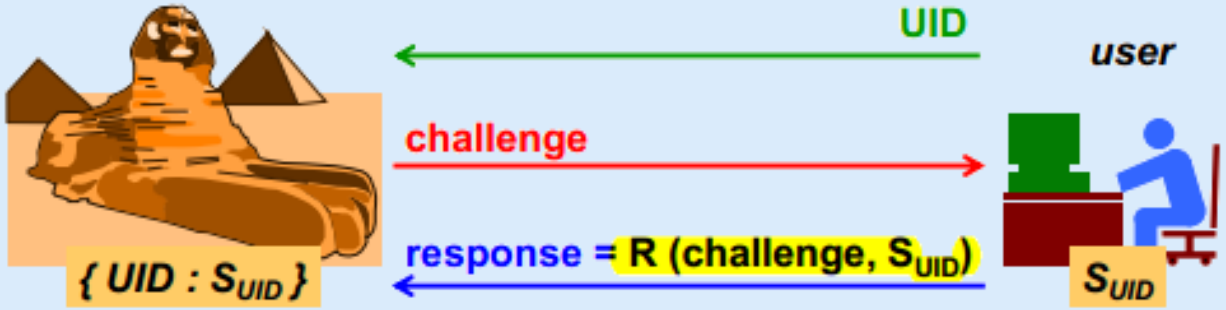
- Remote authentication problem
  - Eavesdroppers listening, looking for passwords
  - Solutions

  1) transmit password over an encrypted channel
     - i.e. manage the overhead of creating a secure channel first
  2) use a challenge-response strategy

Challenge-response authentication

Using Symmetric Crypto

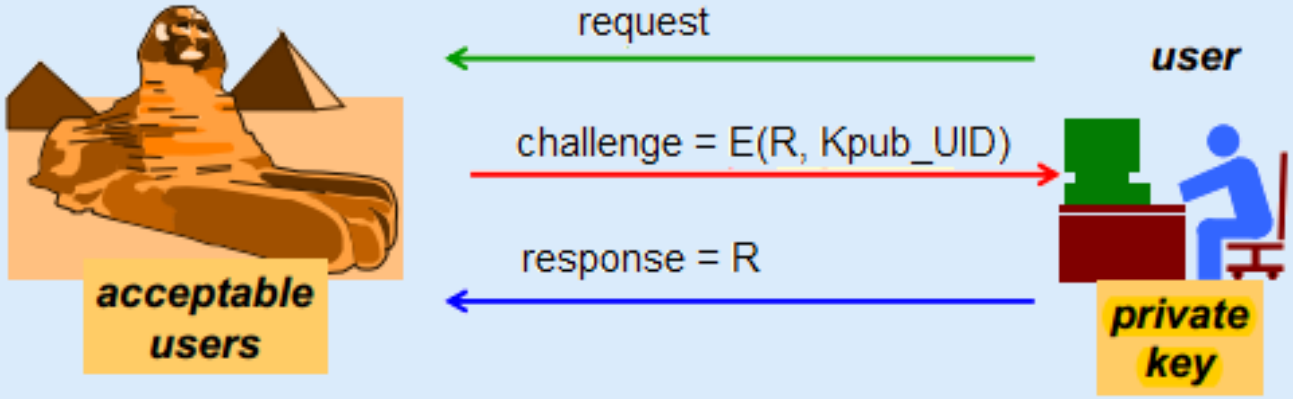
- server sends a challenge (typically a random nonce) to the user …
- … who replies with the solution after a computation involving the shared secret and the challenge
  - e.g. encrypt the nonce using shared secret as the key
  - the server should know the secret in clear!
- $R()$ is a non-secret mathematical function



Challenge-response authentication

Using Asymmetric Crypto

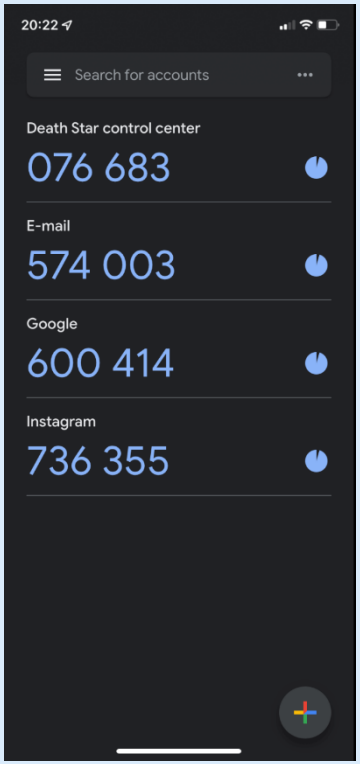
- server knows the public keys of all users
- server sends a challenge to user: a random number R encrypted with the user's public key ...
- ... and the users replies by sending R in clear thanks to its knowledge of the private key



Possession-based authentication

Mostly implemented via One-Time Passcode (OTP), which is generated from a **secret (seed).**

- The secret could be held by server only.
  - OTP generated by server and sent by call/text to mobile number
- Or the secret could be shared between client and server. So OTP can be created at client side:
  - generated and shown by a hardware device
  - generated and shown by an app on smartphone
  - emitted by smart card or hardware keys (connected over USB, Bluetooth or NFC)



Code generator app (google authenticator)

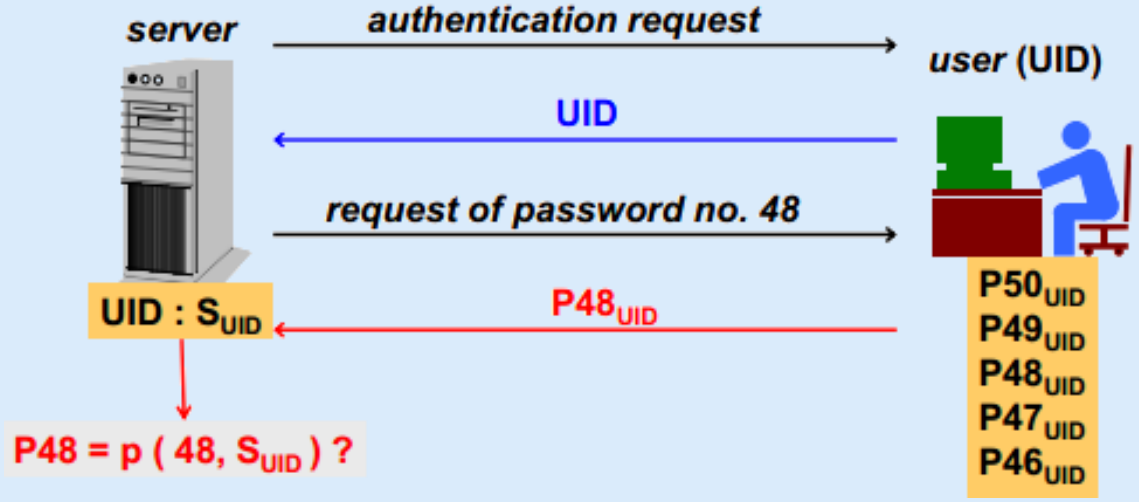
Hardware token

YubiKey

Possession-based authentication

Counter based one-time passwords

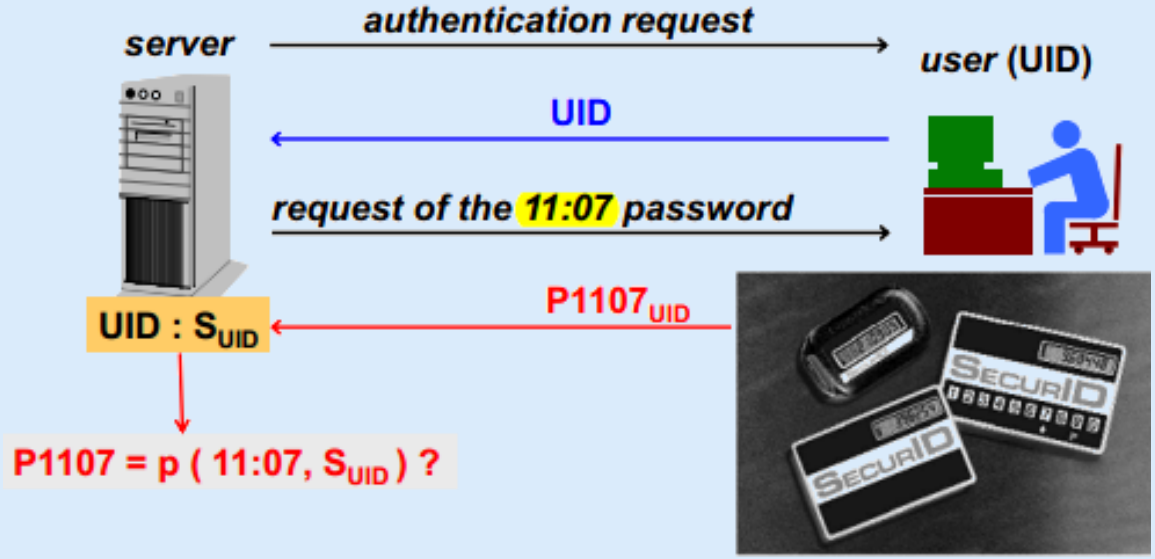
- A secret value (seed) is securely handed over to client
- At authentication time, OTP is calculated at the client using **an increasing counter** and seed
- A one-way function is used for OTP calculation (why?)



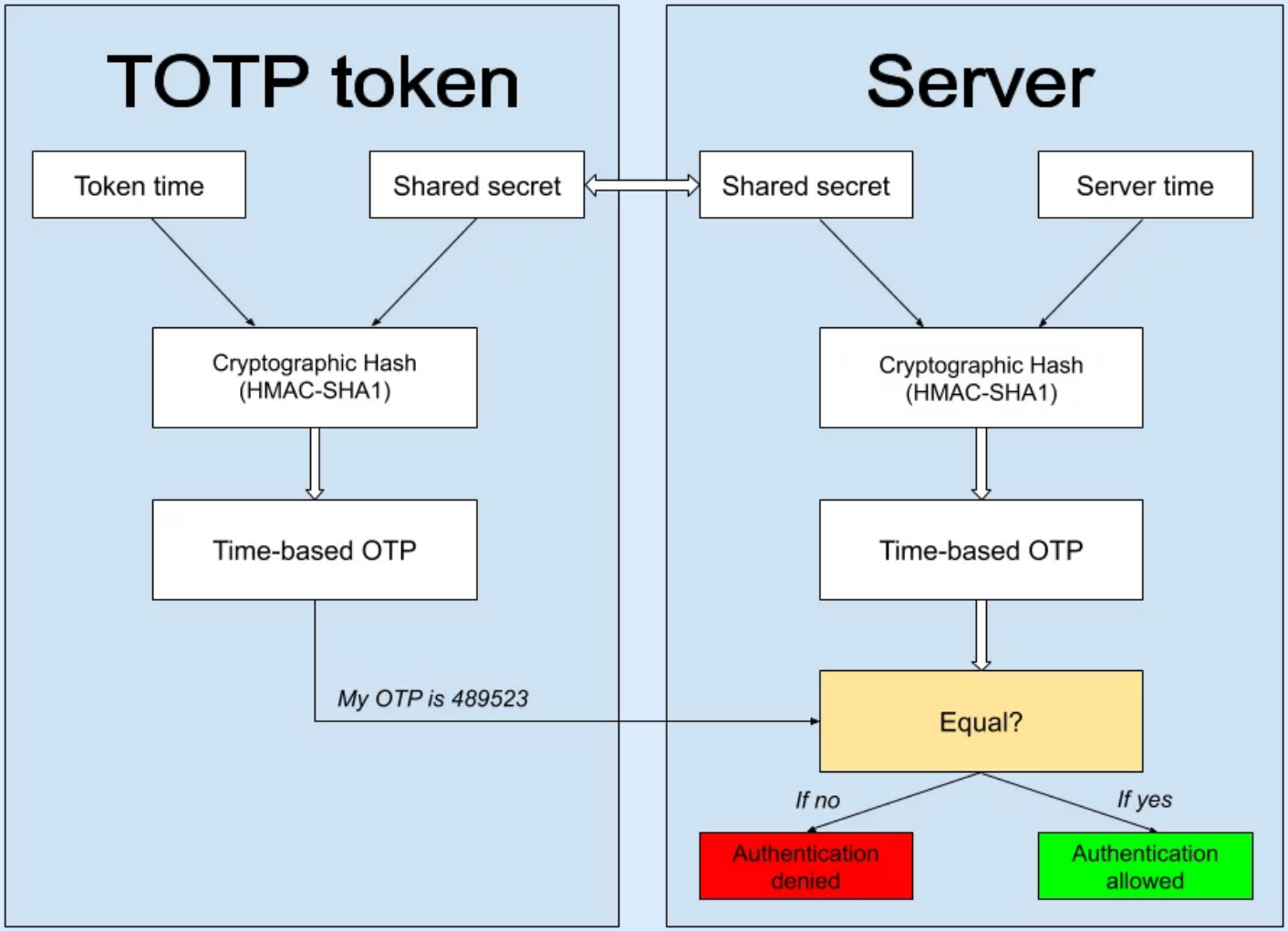
Possession-based authentication

Time based one-time passwords (TOTP)

- A secret value (seed) is securely handed over to client
- At authentication time, OTP is calculated at the client using **the current time** and seed
- A one-way function is used for OTP calculation (why?)



Possession-based authentication



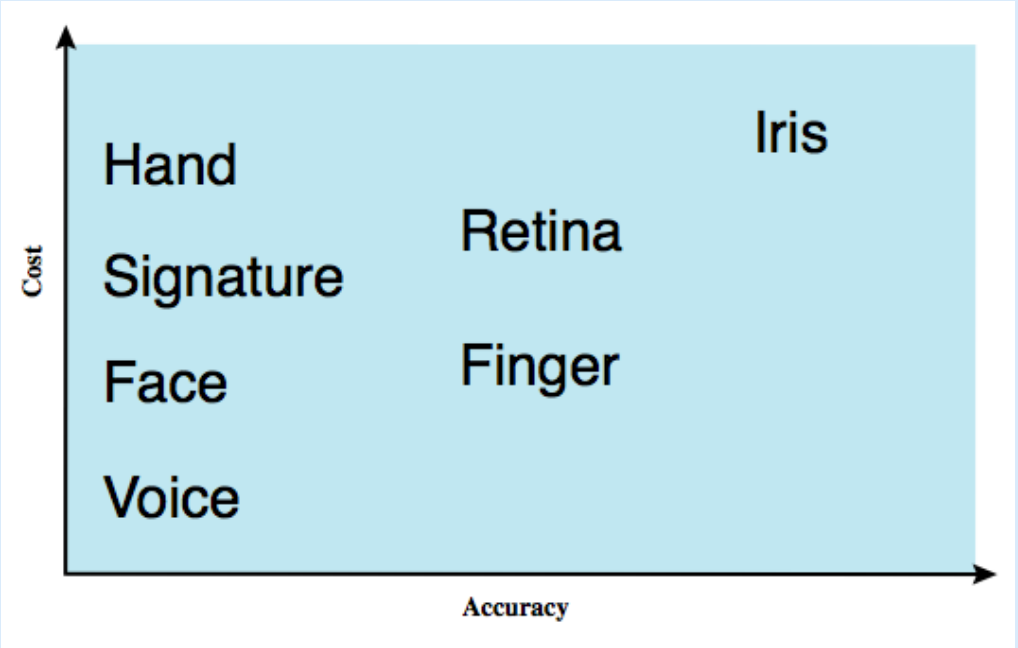
SecurID: Architecture

- invented and patented by RSA security
- time-based synchronous OTP technique
  - P_{UID} (t) = f(SUID, t)

- the client sends:
  - user, PIN, token-code (computed from seed and current time)
- based on user name and PIN the server verifies against three possible token-codes:
  - TC-1, TC-0, TC+1
- will fail if there is a drift of more than one minute
- wrong authentication attempts limited

Biometric Authentication

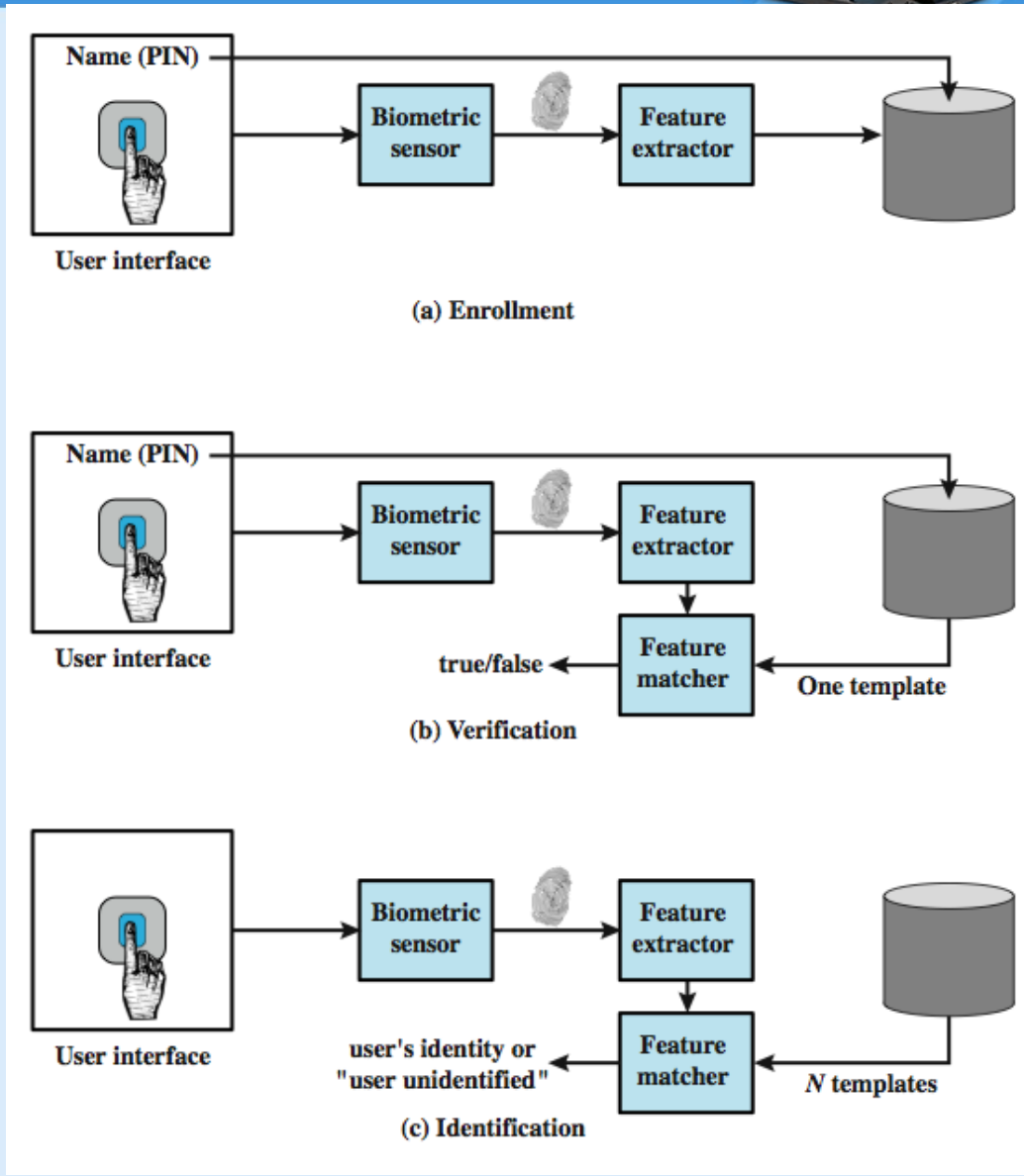
- Authenticate user based on one of their physical characteristics:
  - facial features
  - fingerprint
  - hand geometry
  - palm vein pattern
  - retina vessel pattern
  - iris pattern
  - signature dynamics
  - voice



Operation of a biometric system

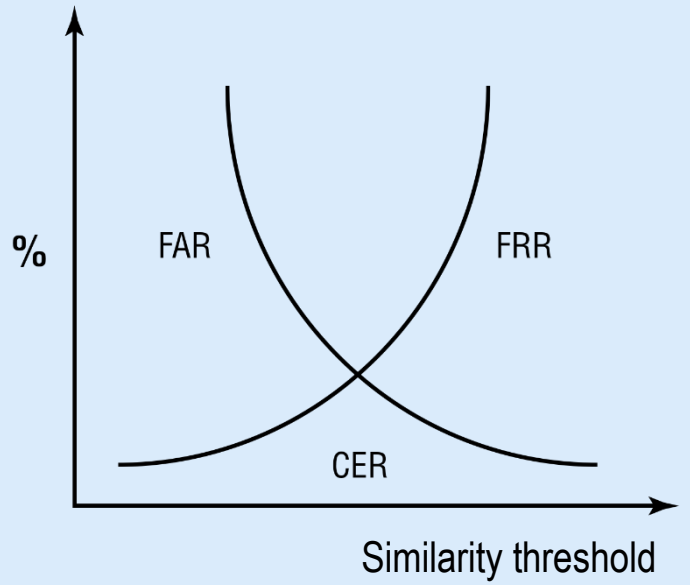
Verification is analogous to user login via a smart card and a PIN

Identification is biometric info but no IDs; system compares with stored templates



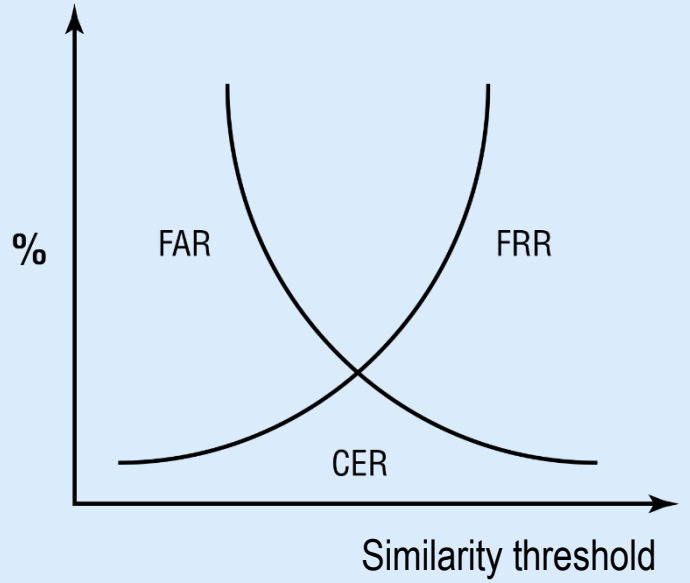
Biometric template matching issue

- Biometric information is digitized, then feature-extracted. Due to sensor noise and variability in human usage, there won't be an exact match between any two biometric scans
  - So, system computes a **similarity score** (say 0 to 100). If the score is above some **threshold**, the user is accepted otherwise rejected
  - FAR = False Acceptance Rate
  - FRR = False Rejection Rate
  - Using a higher threshold will reduce FAR (unauthorized users gaining access), at the cost of more genuine users getting blocked (high FRR)!



Biometric template matching issue

- The crossover error rate (CER) is the point at which false reject and false accept rates intersect.
- CER can be used as a measure of quality of biometric hardware. A higher-quality and expensive system will have a low CER (say 1-2%).



Other challenges in biometric auth

- Biological characteristics are variable due to aging, wounds, swelling, wetness etc.
  - voice altered due to emotion or injury: https://youtu.be/iYhpbph4sLc
  - retinal blood pattern altered due to alcohol or drug
- Persons with disability unable to use the system, so a fallback is needed
- Extra hardware & logistical costs of sensors
- No anonymous access possible

Comparison of biometric methods

Biometrics	Universality	Uniqueness	Permanence	Collectability	Performance	Acceptability	Circumvention
Face	H	L	M	H	L	H	L
Facial Thermogram	H	H	L	H	M	H	H
Fingerprint	M	H	H	M	H	M	H
Hand Geometry	M	M	M	H	M	M	M
Hand Vein	M	M	M	M	M	M	H
Eye: Iris	H	H	H	M	H	H	H
Eye: Retina	H	H	M	L	H	L	H
DNA	H	H	H	L	H	L	L
Odor & Scent	H	H	H	L	L	M	L
Voice	M	L	L	M	L	H	L
Signature	L	L	L	H	L	H	L
Keystroke	L	L	L	M	L	M	M
Gait	M	L	L	H	L	H	M

Table 6-1 Ranking of Biometric Effectiveness and Acceptance

Note: In the table, H = High, M = Medium, and L = Low.

Network authentication with multiple services

Multi-Service Authentication

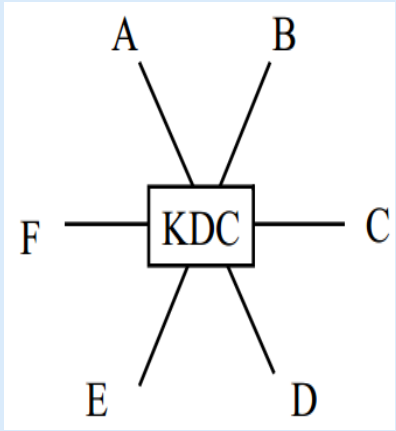
- A typical enterprise environment has several application servers
  - e.g. sales management, knowledgebase, wiki, accounts, HR management, technical support
- It will be too complex (and insecure) to store separate user credentials or keys on each application server

- Solution: designate a **single authentication server** for all auth requests

Key Distribution Center

A solution for multi-party secure communication using **symmetric cryptography**

- Each node is configured with a secret key, shared with KDC.
- KDC has all the keys.
- To initiate A ↔ B communication,
  - KDC sends a session key K_{AB} encrypted with A's key to A and encrypted with B's key to B.



Issues:

- if KDC is compromised, all systems are compromised
- single point of failure or performance bottleneck
- KDC has to be online all the time. Replication!

Kerberos

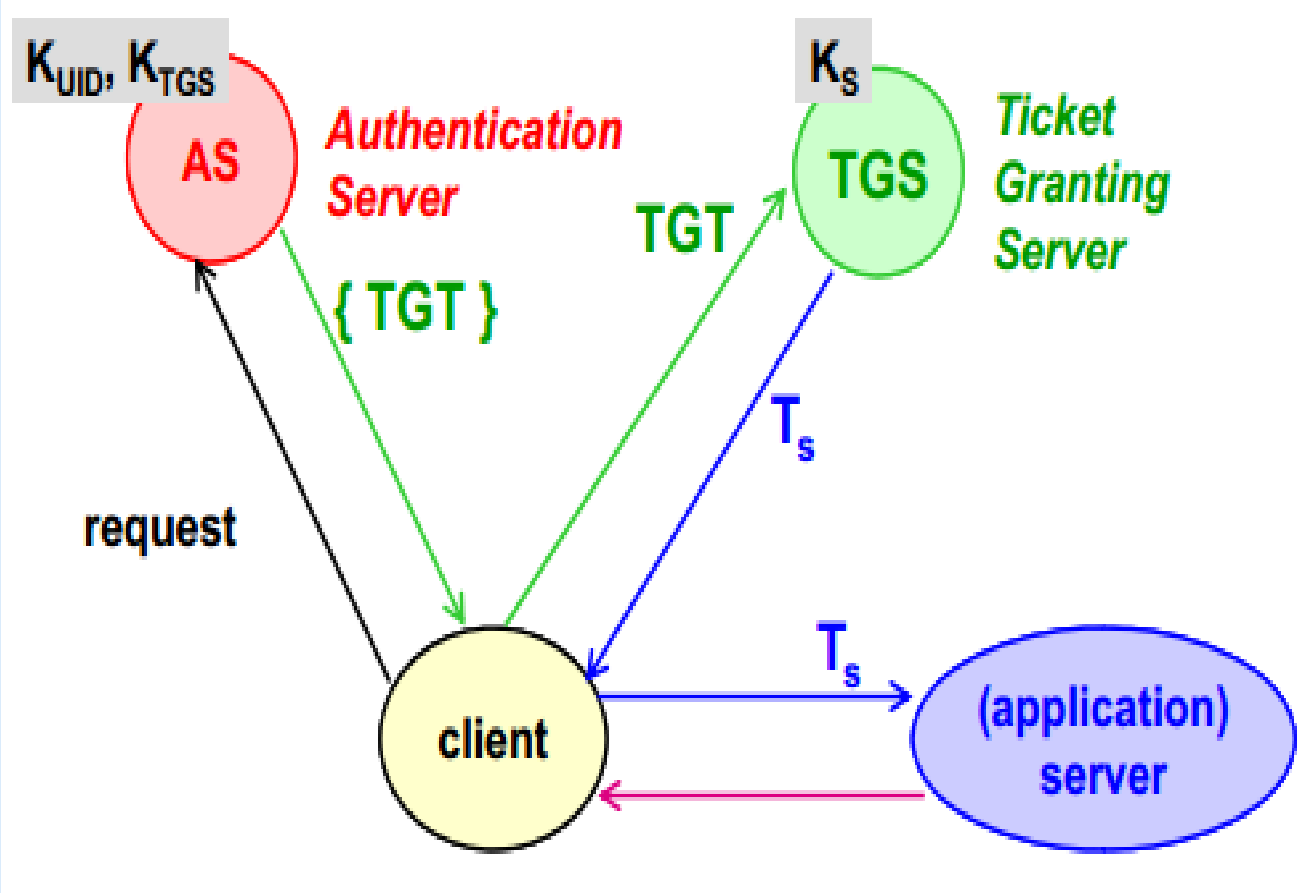
- Network authentication protocol
- Based on idea of using symmetric crypto
  - Maintain a KDC
- Developed at MIT for project Athena

- Named after Greek mythological character “Cerberus” – the three headed dog
- Used by popular operating systems and servers
- Protect against eavesdropping and replay attacks

Kerberos overview

- Authentication server authenticates a user
- TGS, Ticket Granting Server, grants ticket to the user, for a specific service in the network
  - 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, the KDC (auth. server + TGS) and the Application server are the 3 heads of kerberos

Kerberos High Level Working

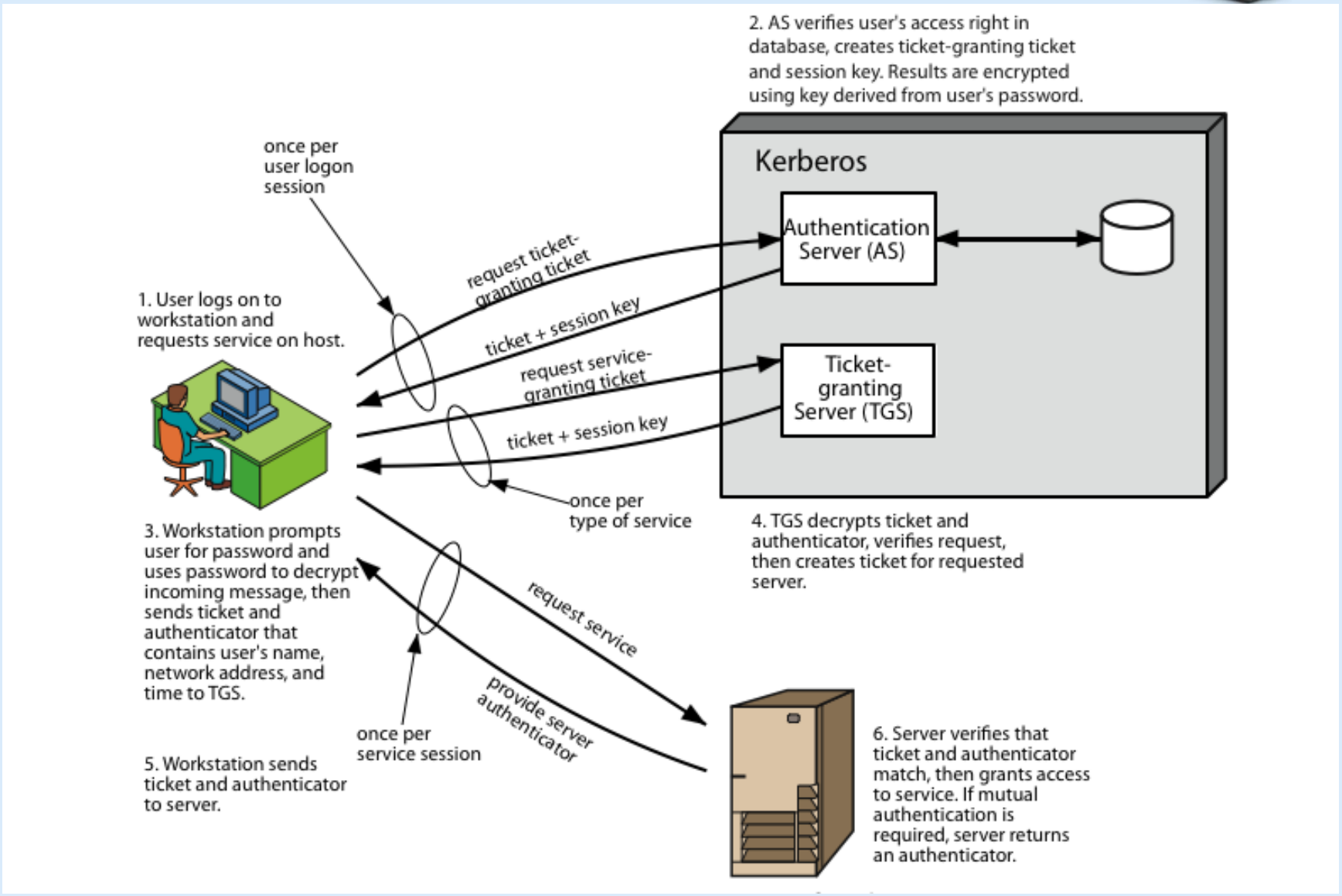


TGT = ticket granting ticket T_s = service ticket

K_{UID} = secret key of user, pre-shared with AS

K_S = secret key of application server, pre-shared with TGS

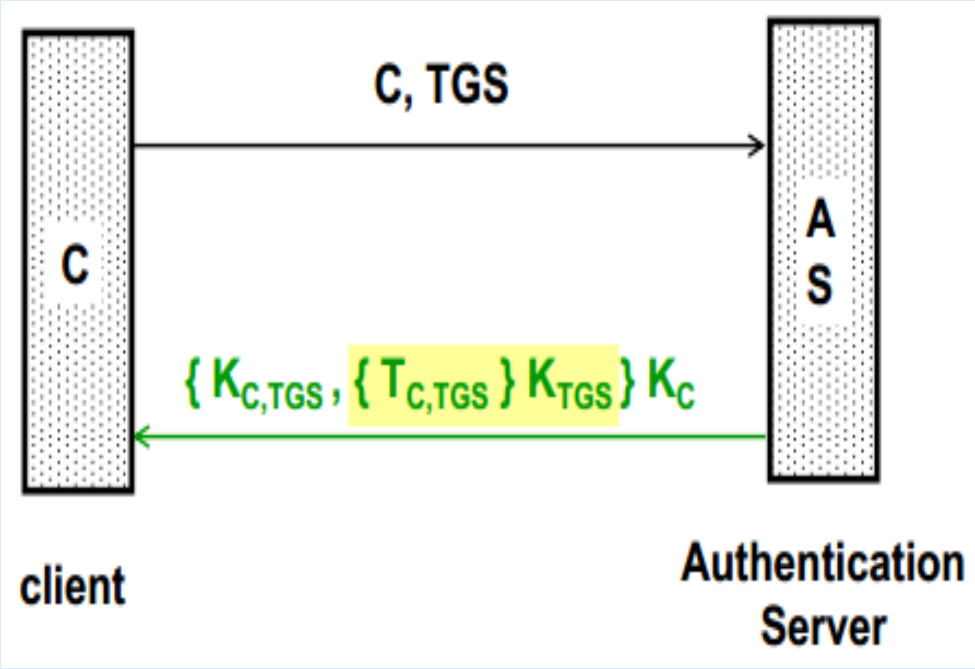
Kerberos Protocol



1-2. Client Authentication (Issue TGT)

Request

- client send their id C, and ask for a ticket that will help them connect to ticket granting server TGS
- this message is in plaintext



Response contains

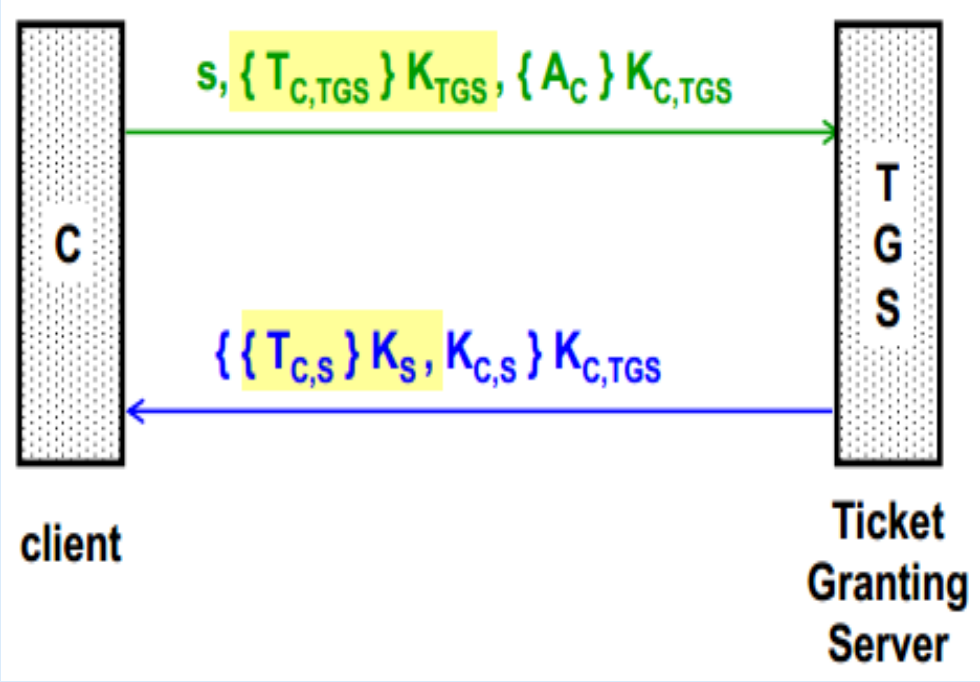
- $K_{C,TGS}$: the session key to connect to ticket granting server TGS
- $T_{C,TGS}$: the ticket-granting ticket. It is encrypted with K_{TGS} (the secret key of TGS), not known to client

Whole response is encrypted with the secret key of client K_C (derived from their password)

3-4. Client Authorization (Issue Service Ticket)

Request

- s : id of the service that client wants to use
- Encrypted TGT (yellow highlighted)
- A_C : an authenticator message from client, proving his identity to TGS. It is encrypted by this session's key $K_{C,TGS}$



Response

- $T_{C,S}$: service ticket, encrypted with key of application server K_S (not known to client)
- $K_{C,S}$: A session key for use between client and application server

Whole message is encrypted by this session's key $K_{C,TGS}$

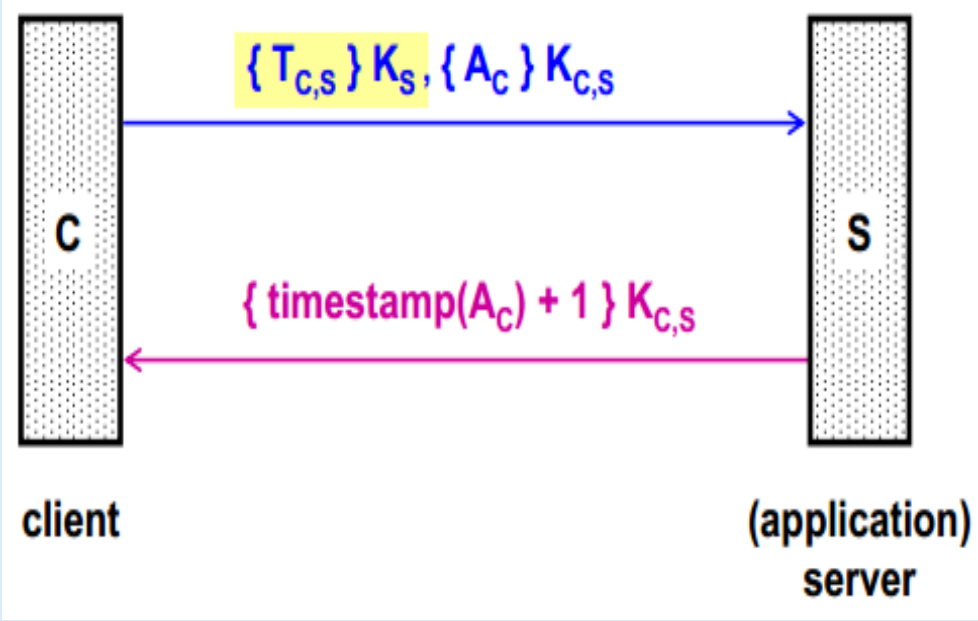
5-6. Client Service Request

Request

- the encrypted service ticket (yellow highlighted)
- A_C : an authenticator message containing client ID and timestamp. It is encrypted by this session's key $K_{C,S}$.

Response

- Timestamp in client's message plus 1, encrypted by this session's key $K_{C,S}$. It is a confirmation to client that server is ready to serve it.



For reference

Kerberos message exchange in full detail

https://www.youtube.com/watch?v=5N242XcKAsM

Other Authentication Systems

- SSO (single sign-on)
  - Single credential set, multiple services
- OAuth (open authentication)
  - Allows “access tokens” to be issued to third party clients