Intro to Authentication

Is the process of proving that you are who you say you are.

Applies to humans, bots, systems...

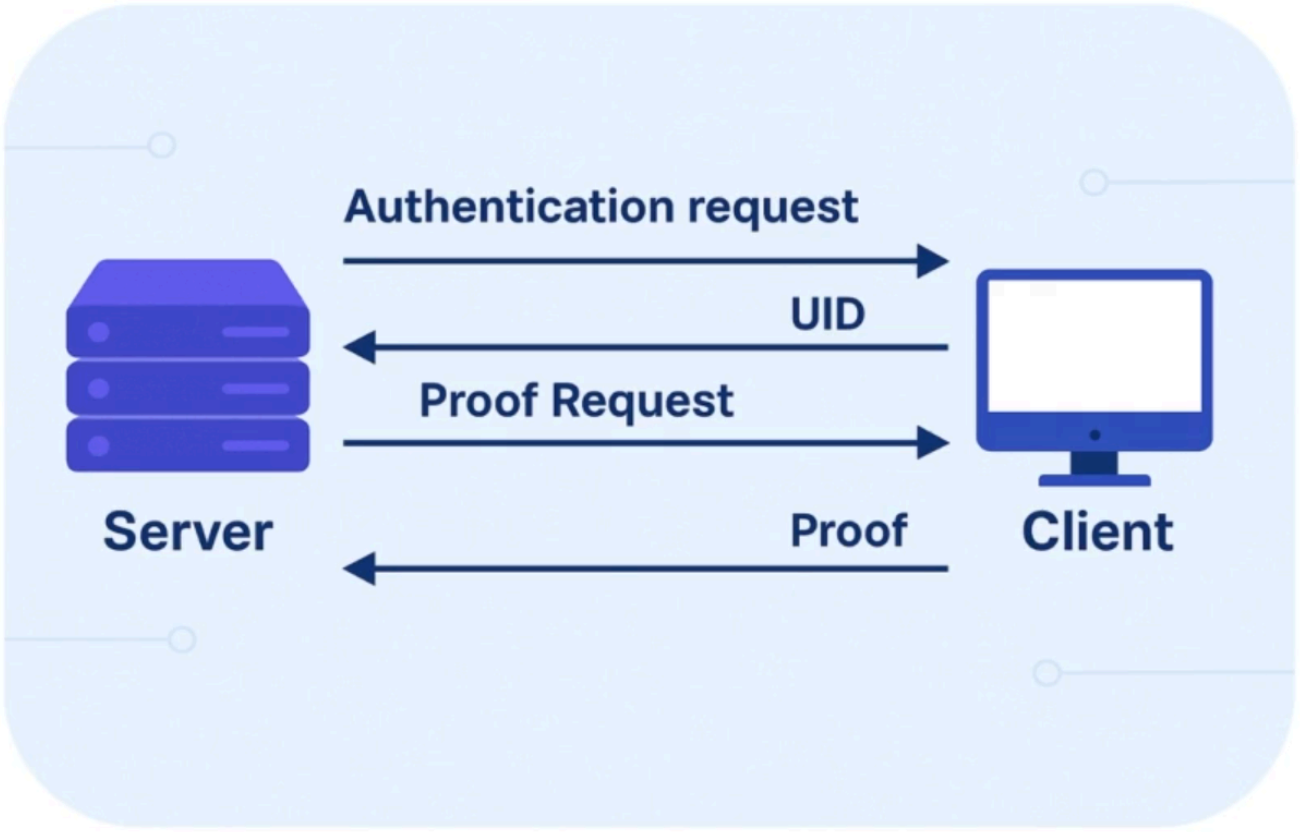
Auth Factors

Auth factors could be:

- Knowledge that only the user know (passwords, pins...)
- Ownership (id cards and such)
- Inherence (biometrics)

> We do not share biometrics over network bcz if they get leaked they can't be changed like a password (unless ur fine with cutting ur fingers off :>)

Generic Authentication



Password Based Authentication

- Client transmits proof in clear text
- Server:
  - Case 1:
    - Server save as clear text.
  - Case 2:
    - Server saves hashed passwords.

Make sure to use good passwords, and reset them each once in a while.

Dictionary Attack

Brute forcing, some databases were made for known hashes, and for the most common passwords.

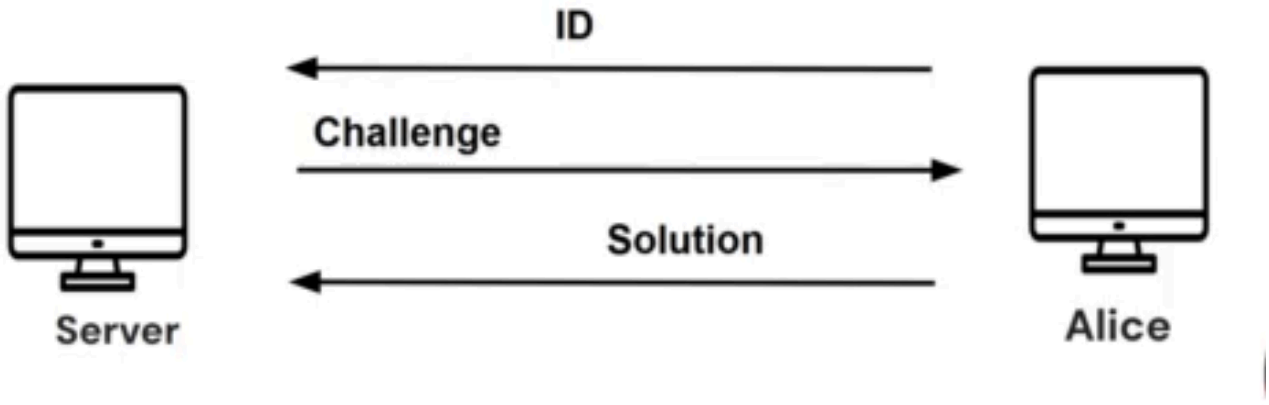
A problem with hashes is that if two people are using the same password, the hash would be the same for both of them, which might lead the attacker into thinking about using dictionary attacks as the password doesn't seem to be unique for a person, so it might be a common password that might be alr on some database.

Salt

An additive to the password, u use to make distinct hashes for the same password.

Why it works, basically bcz it makes dictionaries useless, u gotta redo all hashes with the salt for it to work.

Challenge Response Authentication



Challenge must be:

- Non repeatable
- Non invertable

Symmetric CRA

1. Share secret key over some secure channel/using encryption (diffie-hellman and such)
2. Server send nonce to user
3. User sends the solution back to server

Symmetric CRA Attack

1. Intruder sends a Nonce to Bob to solve
2. Bob sends the solution and sends a nonce for the intruder to solve
3. Intruder starts a new session and sends the nonce he got back to Bob
4. Bob solves the nonce and sends back to second session
5. Intruder takes this answer and sends it back to Bob
6. viola, intruder is in ggs.

Asymmetric CRA

1. User sends certificate with their public key
2. Server sends encrypted nonce will the user's public key, only the owner of the secret key can get back the actual thing, thus intruders can't use this
3. User sends the response in plain text back, yes can be sniffed, but it is not repeatable so is fine

---

MFA/2FA

Apply another layer of security.

One of the layers should be non reusable (just like email verification pins and such)

OTP

One time password

1. Finish first auth
2. Server sends request for OTP
3. User sends OTP and gets access

S/Key System

1. Secret is generated

2. Secret is hashed P1=hash(secret)
3. $$P_n = Hash(P_{n-1})$$
4. Server would store Pn at first
5. Auth would happen by sending Pn-1 (the one before last)
6. Server then verifies the identity if hash works, and discards Pn, and sets the hash value to Pn-1
7. Now we need to send the one just before that until we get back to the secret, then we redo the entire thing

---

Time-Based OTP

One time password made using both secret and time.

Requires:

- Time sync between server and client
- Window time slot

> Messing with the time can suffer different attacks, like DoS by changing the time, or even guessing the password by moving the time forwards and storing the generated OTP.

OOB OTP

Out of bound OTP

1. Server auth req
2. UID + Puid
3. OTP on a different channel (just like using sms)

Authorization

Role-Based Access Control

Depends on roles, admins, mods, members...

Attribute-Based Access Control

Like age restrictions, if under a certain age, remove some permissions

Relationship-Based Access Control

Just like Instagram private accounts, only followers can view

---