Using Keycloak

From Zero to Hero - Keycloak as your BaaS for IDM!

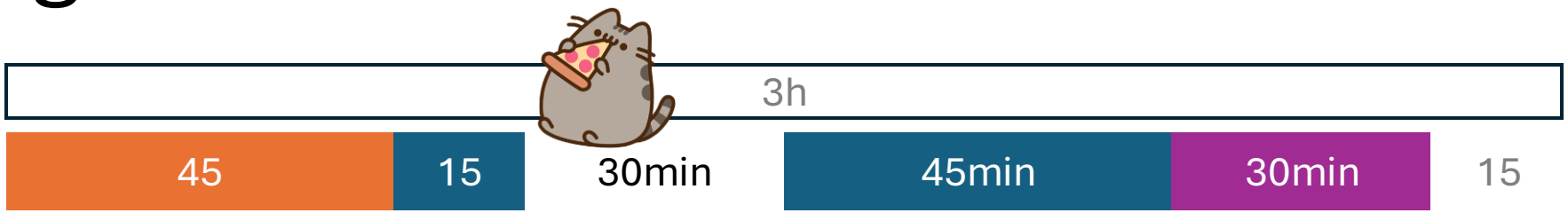
Java Vienna Meetup

Michael Riedmann

@mriedmann
michael.riedmann@riit.at

- Trying to tame computers since 1999 (VBA, VB6)
- Started 2010 as IT Support Engineer and Junior Developer
- Infrastructure Engineer / Dev at BM.I till 2018
  working mainly with Windows (C#, AD, ADFS, PKI, IIS)
- Site Reliability Engineer / Consultant at Cloudflight until 2024
  specialized on Java (mainly Spring Boot, some JavaEE), Linux (RHEL, Ubuntu, CoreOS), Kubernetes (Openshift) and Keycloak with strong Focus on Agile Projectmanagement and Devops
- Now self-employed currently working as Freelancer
  Java Software-Architect, SRE Consultant, but basically trying to solve people's problems with computers.

Our Agenda



Basics+

- Minimal Crypto Intro
- AuthN / AuthZ
- OAuth2
- OIDC

IdPs & Keycloak

- Why?
  - IdP
  - Self-Hosting
  - Keyclaok
- How?

Tips, Tricks & FAQ

- Deployment
- Security
- Pitfalls
- Your questions

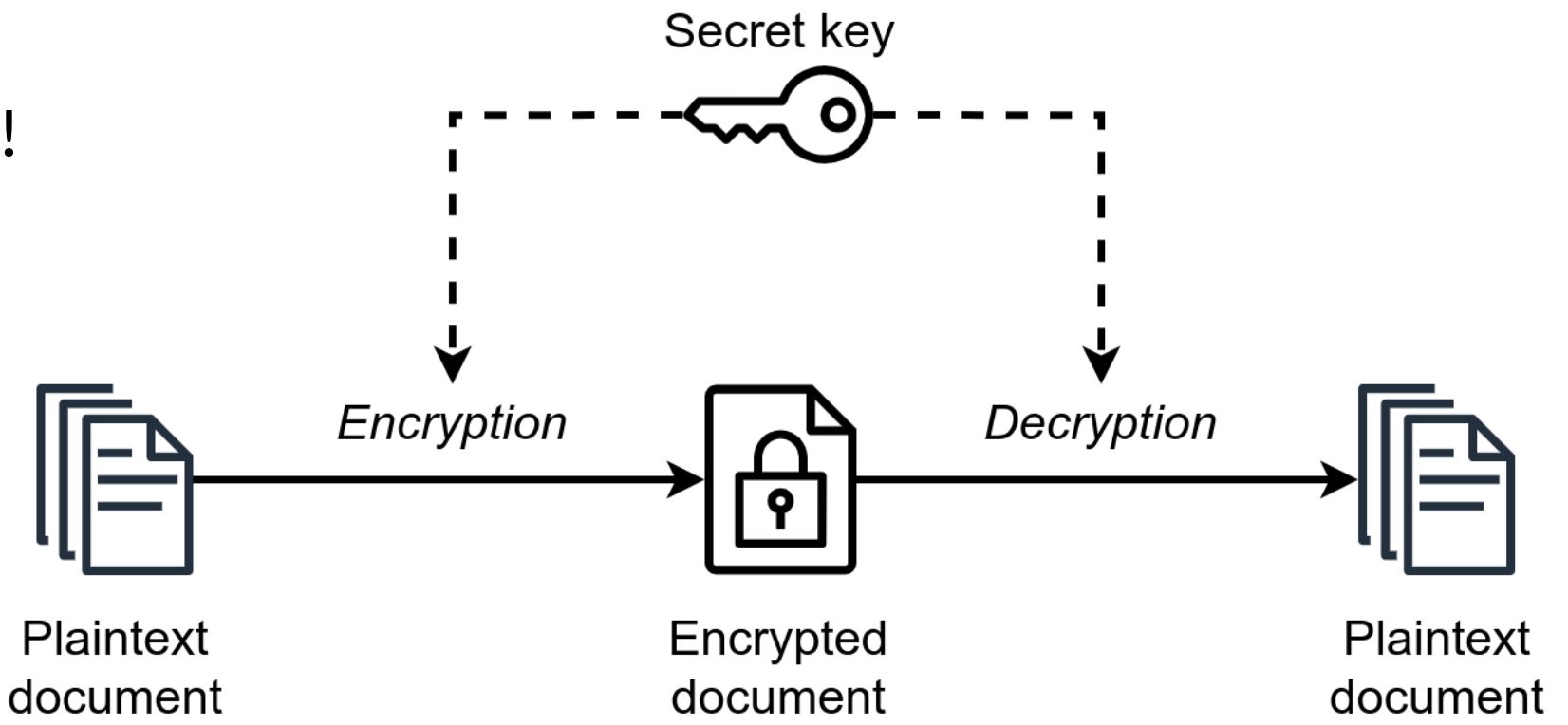
Let’s get started!

Hold on a moment ...

some questions first.

Symmetrical Cryptography

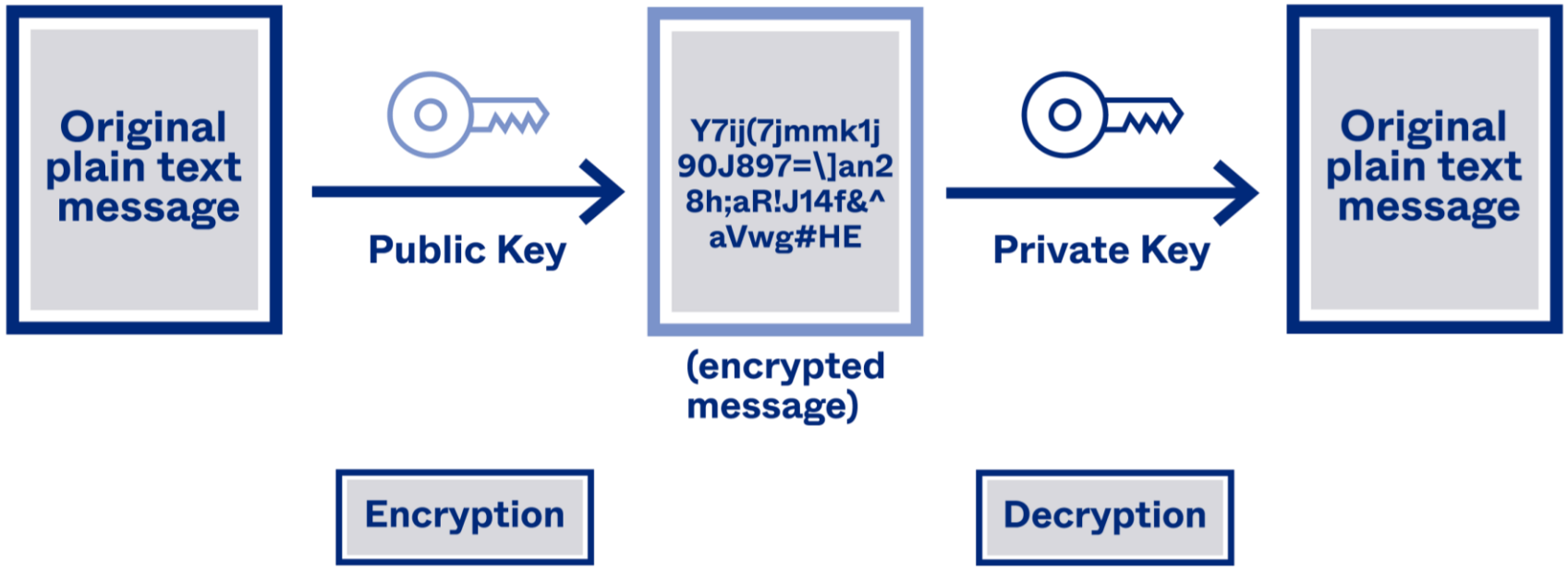
- Problem?
- => Key Exchange!



https://en.wikipedia.org/wiki/File:Simple_symmetric_encryption.png

Asymmetrical Cryptography

- Problem?
- => Slow & Weak



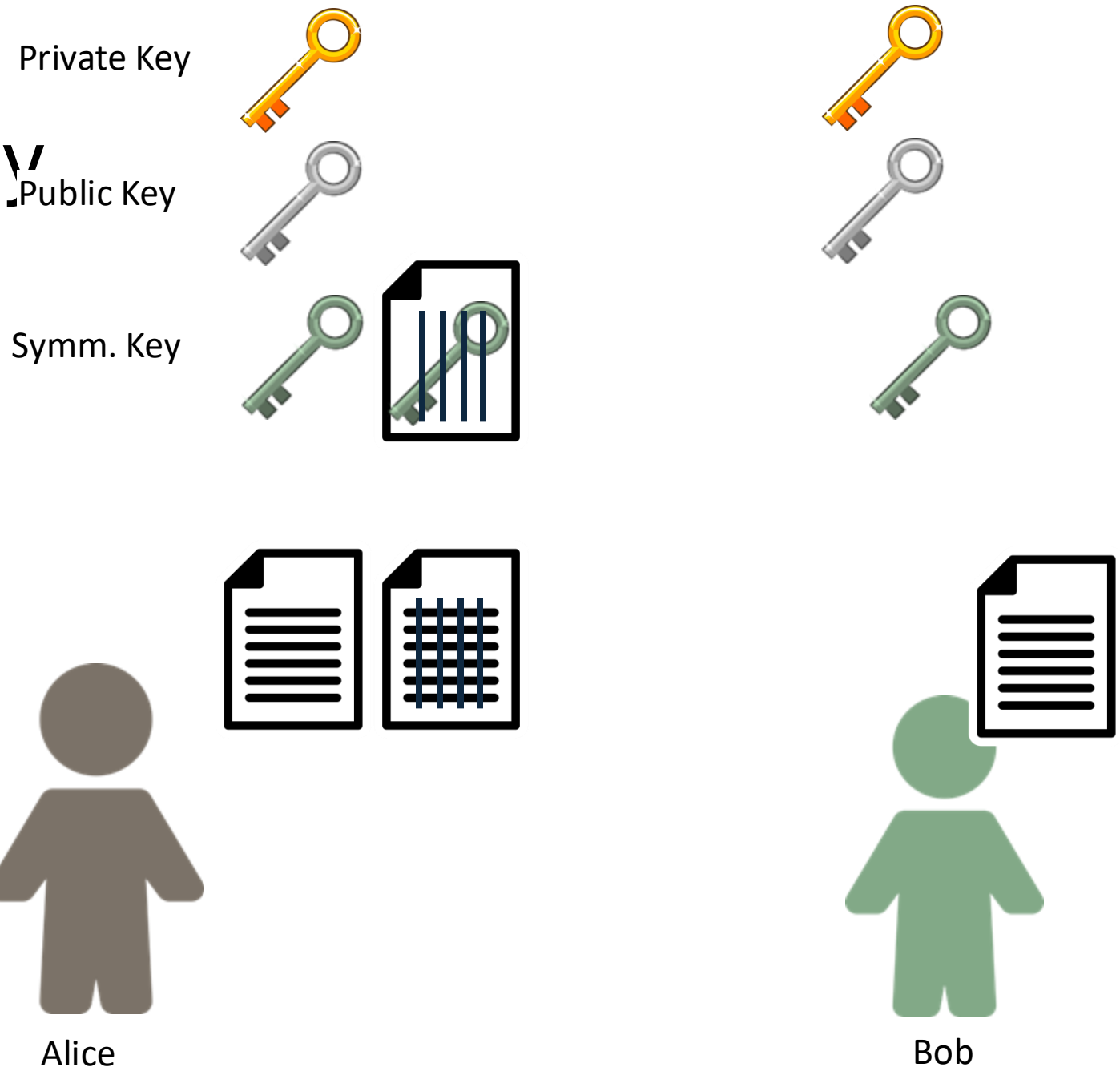
https://www.okta.com/identity-101/public-key-encryption/

“Hybrid” Cryptography

- Use asym. crypto to generate a shared key
- Sym. crypto to send payload data

- Benefits
  - Secure
    (no key can be captured on the way)
  - Fast / Strong
    (because symm. crypto)

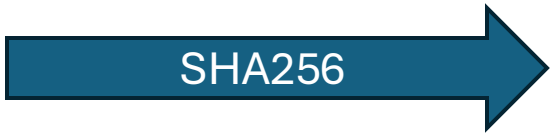
- Problem?
  - How do I know if Bob is Bob and not Eve?



Authenticity
How to know if something is identical?

Hash Functions

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

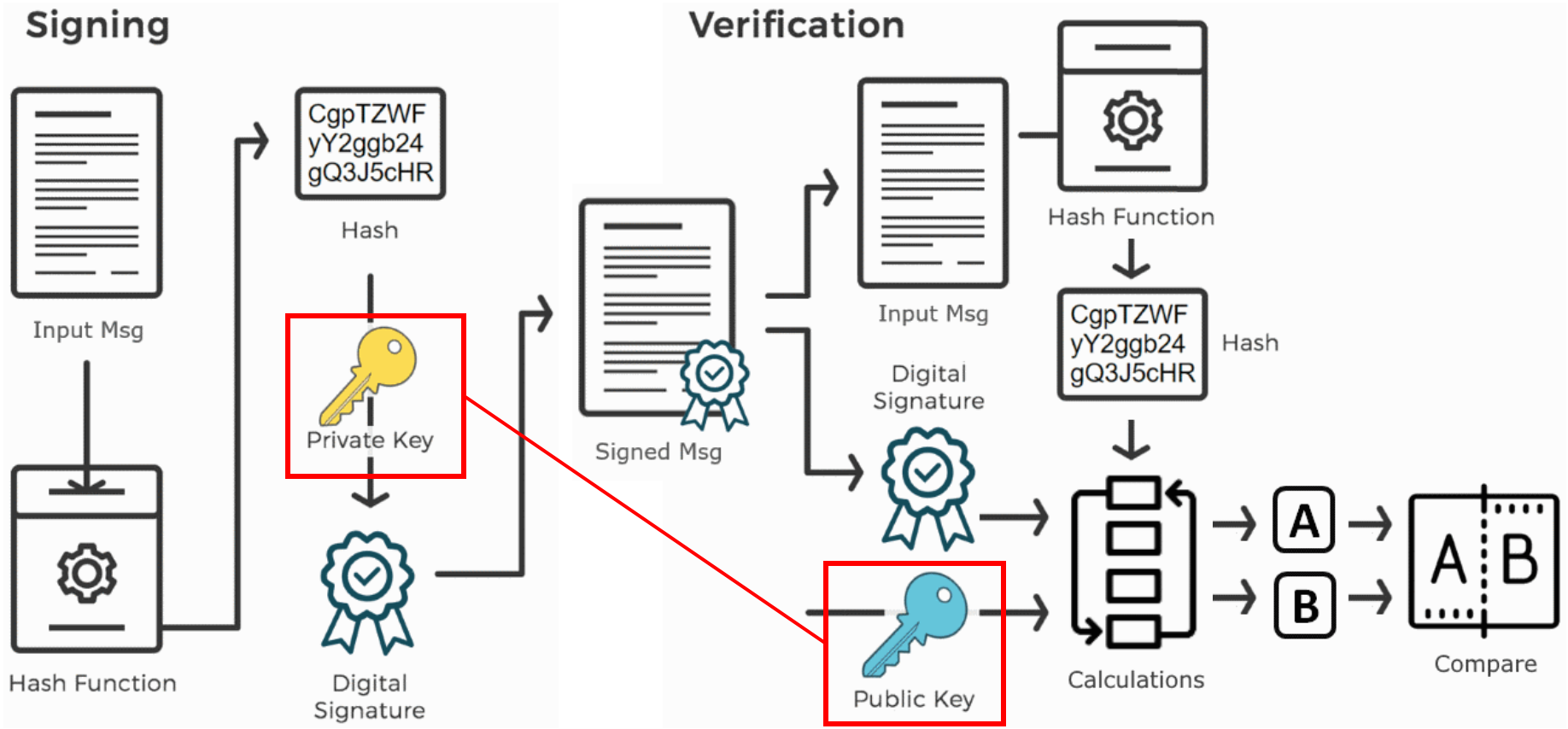


FF4EF4245DA5B09786E3D3DE8B430292F
A081984DB272D2B13ED404B45353D28

- Hash(arbritary length input) = fixed length output
- Same Input = Same Output
- No simple *reversing* possible
- No shortcuts to find a *collision*

Authenticity

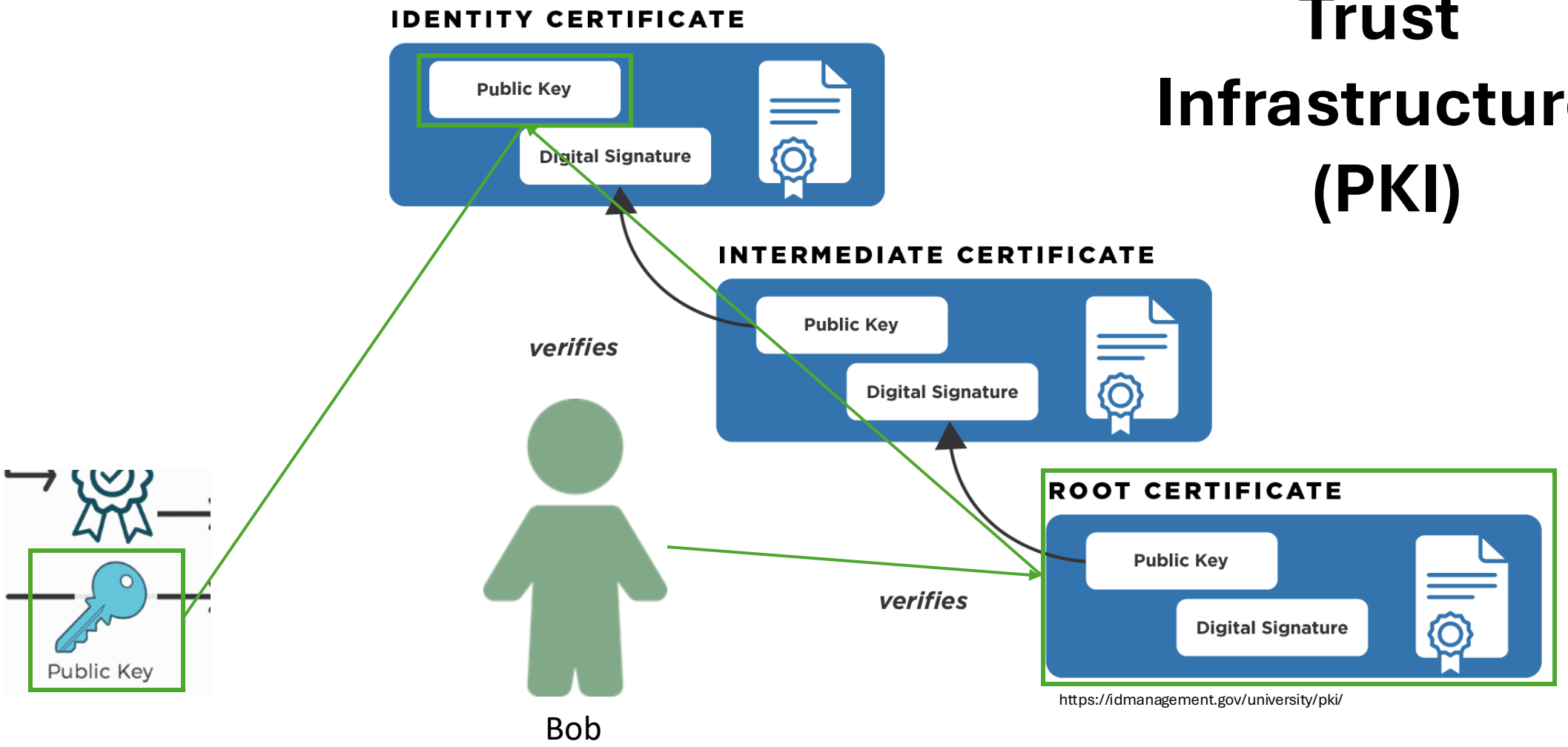
How to know if bob is the sender? **Digital Signatures**



https://cryptobook.nakov.com/digital-signatures

Authenticity
How to know if bob's key is bob's key?

Trust Infrastructure (PKI)



https://idmanagement.gov/university/pki/

Basic basics done ...

continue with basics.

Authentication – AuthN
Who are you?

SOMETHING THE CUSTOMER KNOWS

(e.g., password or PIN)

SOMETHING THE CUSTOMER HAS

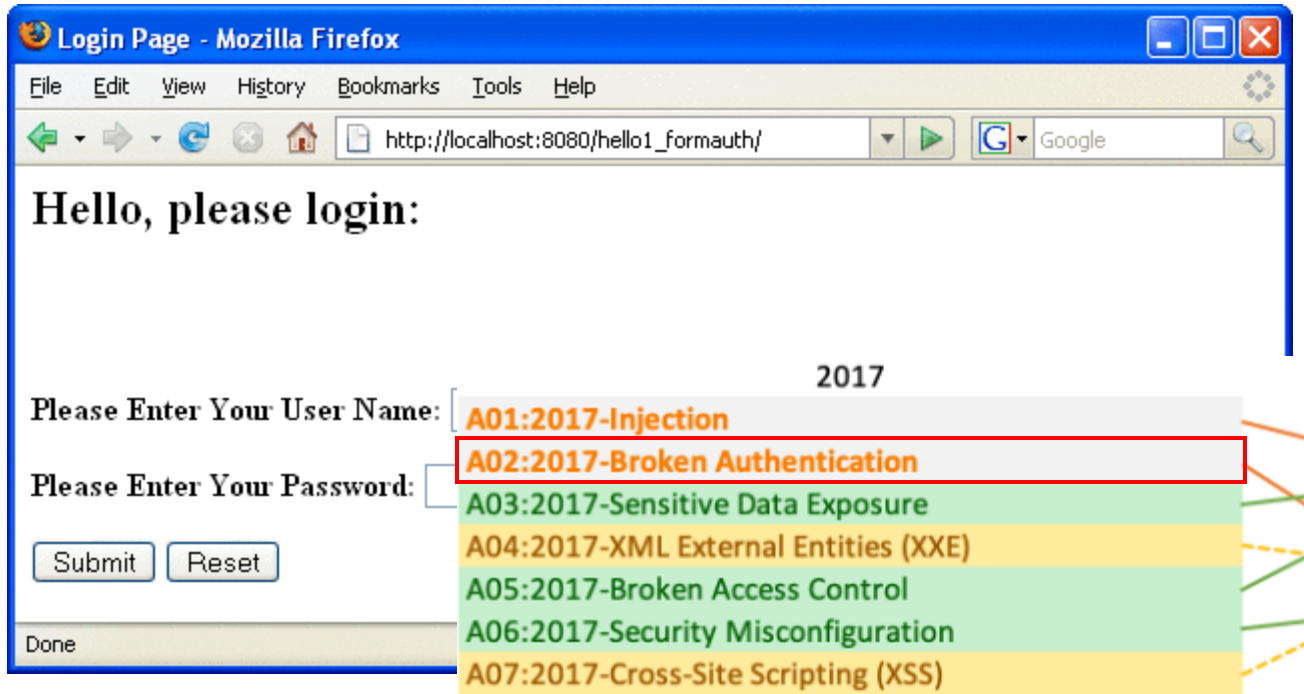
(e.g., phone or hardware token)

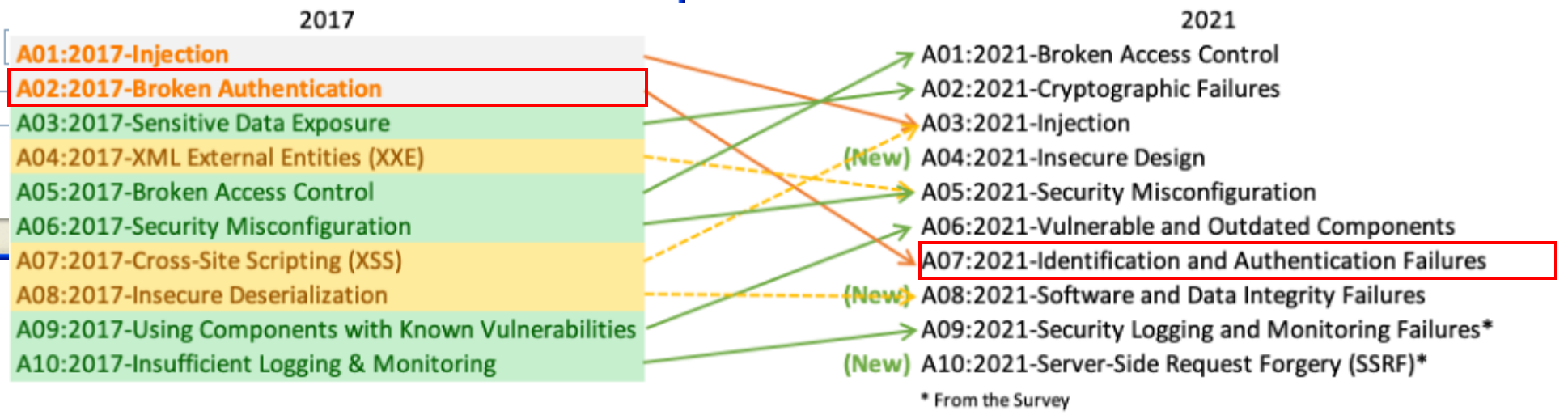
SOMETHING THE CUSTOMER IS

(e.g., fingerprint or face recognition)

https://www.xorlogics.com/tag/psd2/

AuthN – Form-based Authentication



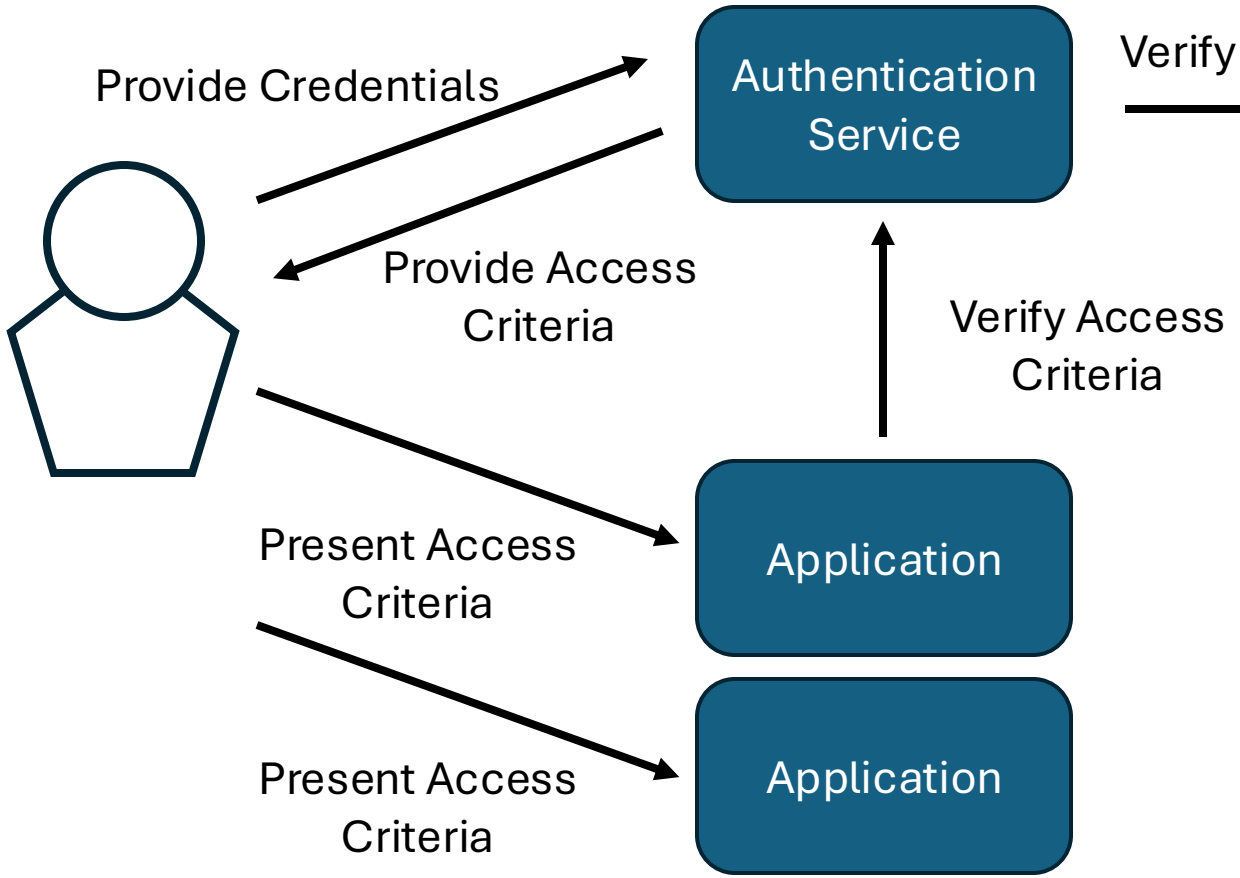


Challenges:

- storage
- Transport
- validation

https://owasp.org/www-project-top-ten/

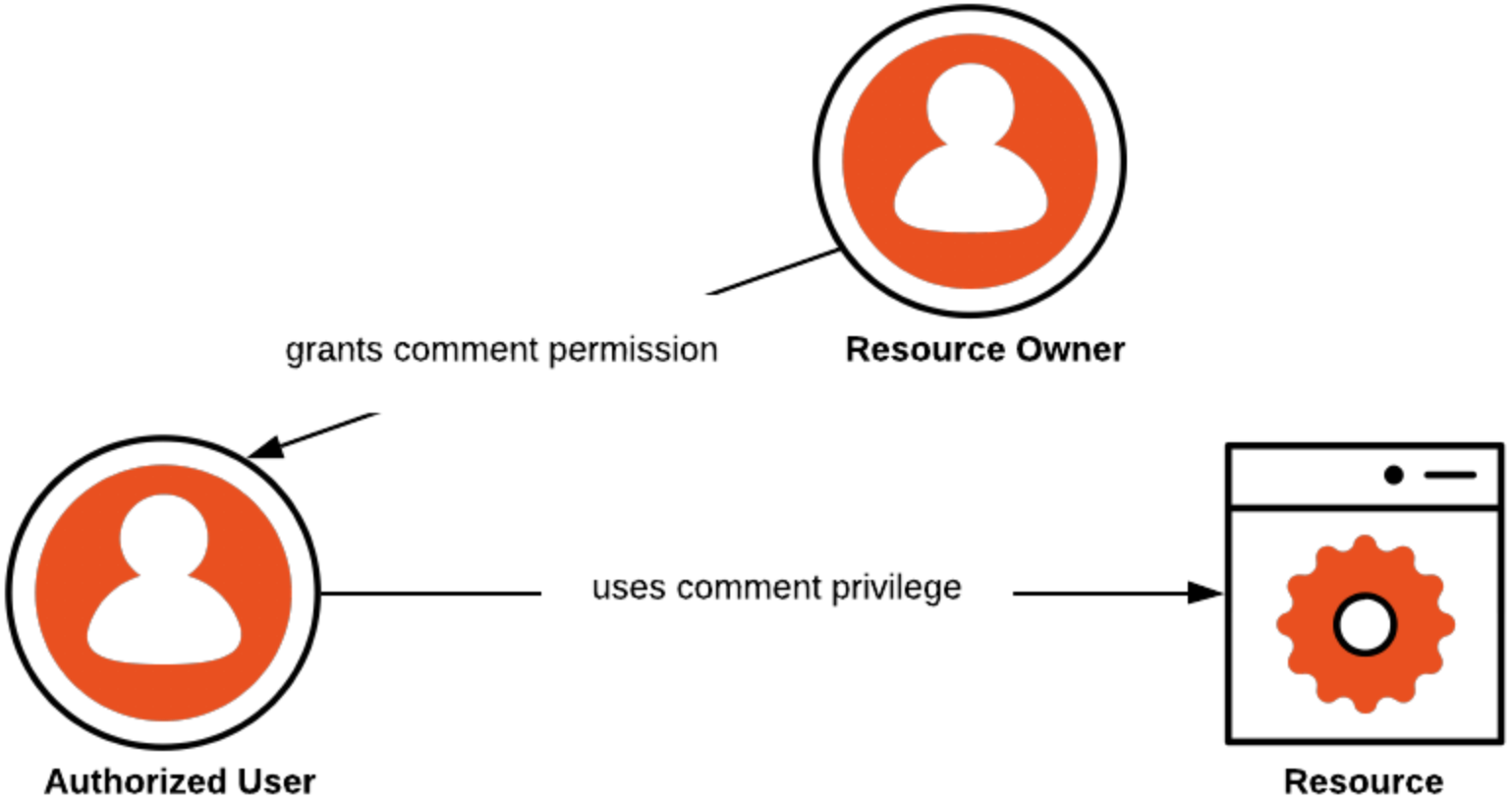
AuthN – Delegated Authentication (SSO)



Challenges:

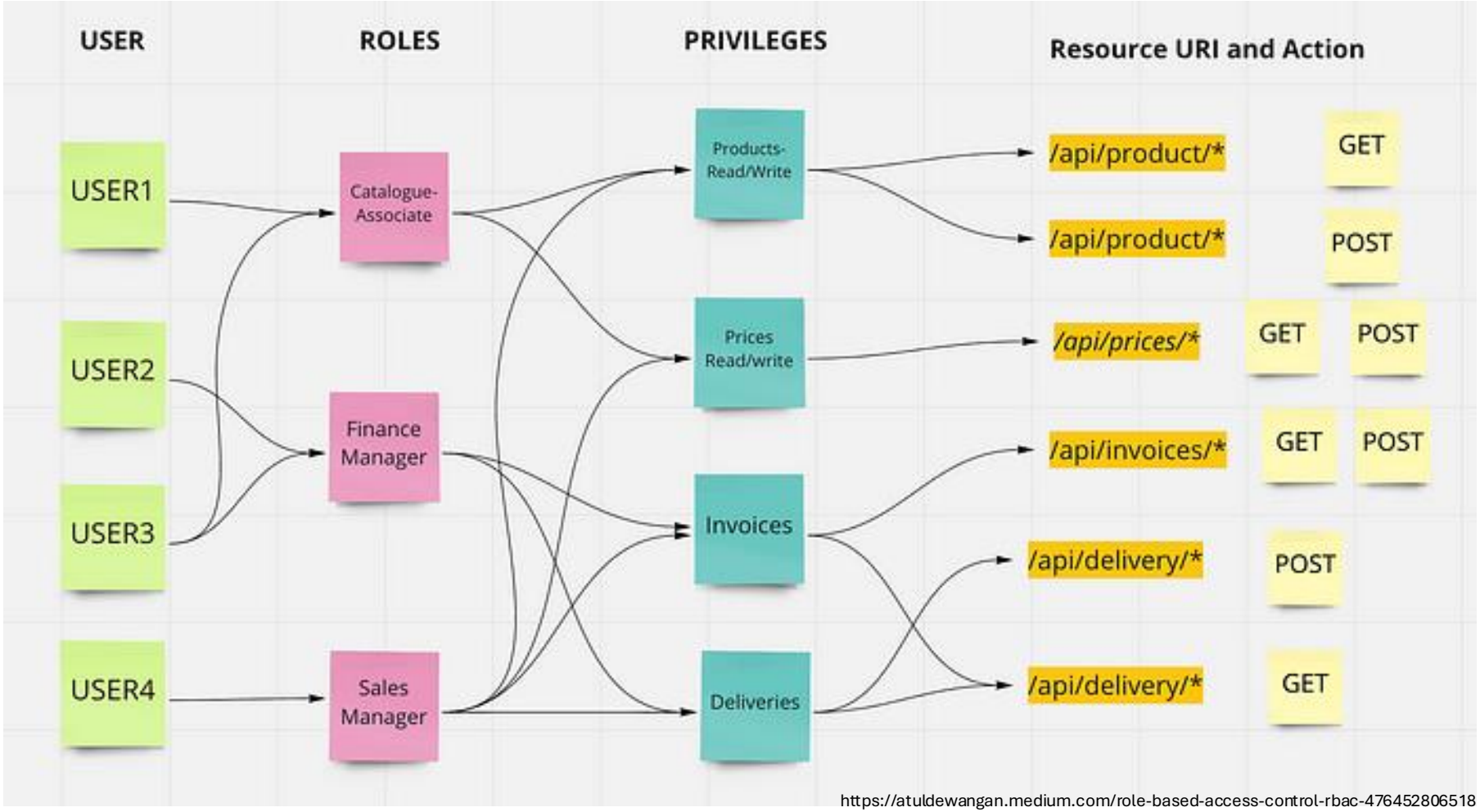
- AC Formats
- AC Validation / AS bottleneck
- App -> AS communication
- Timeouts / Logout

Authorization – AuthZ
Are you allowed to do that?



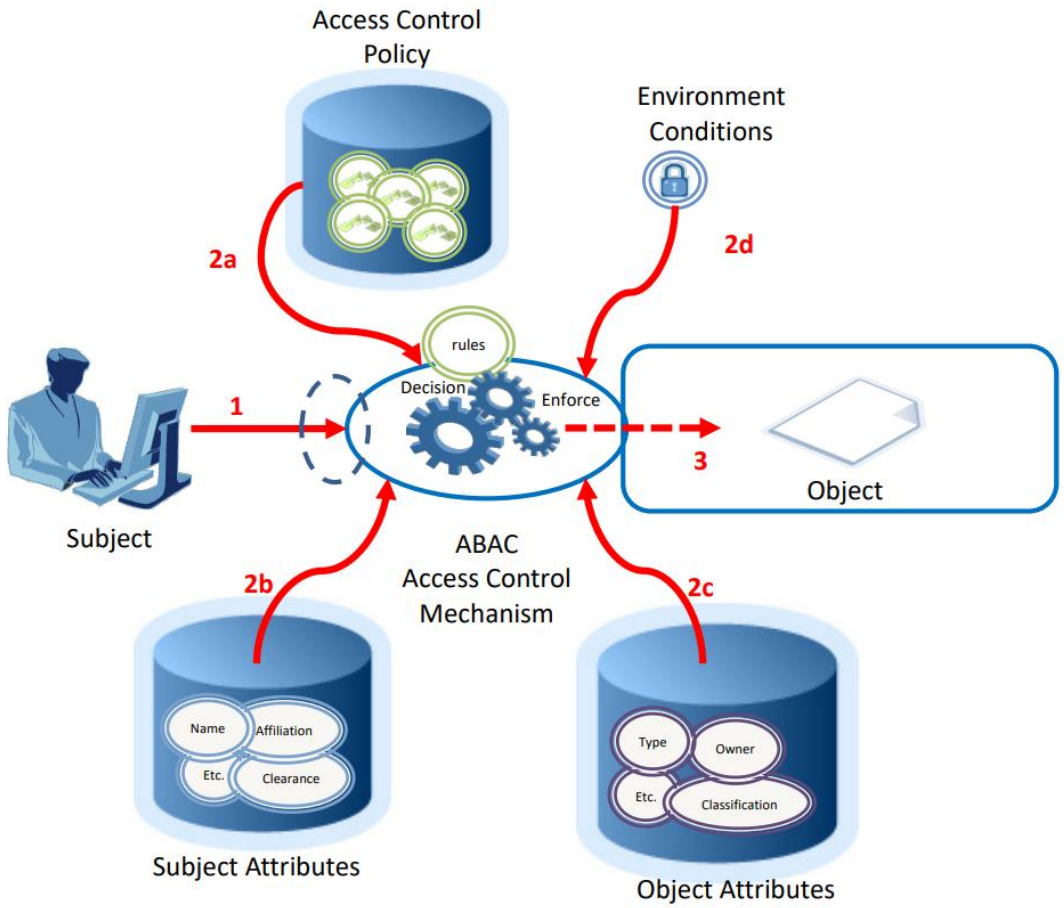
https://auth0.com/intro-to-iam/what-is-authorization

AuthZ - RBAC
Role Based Access Control



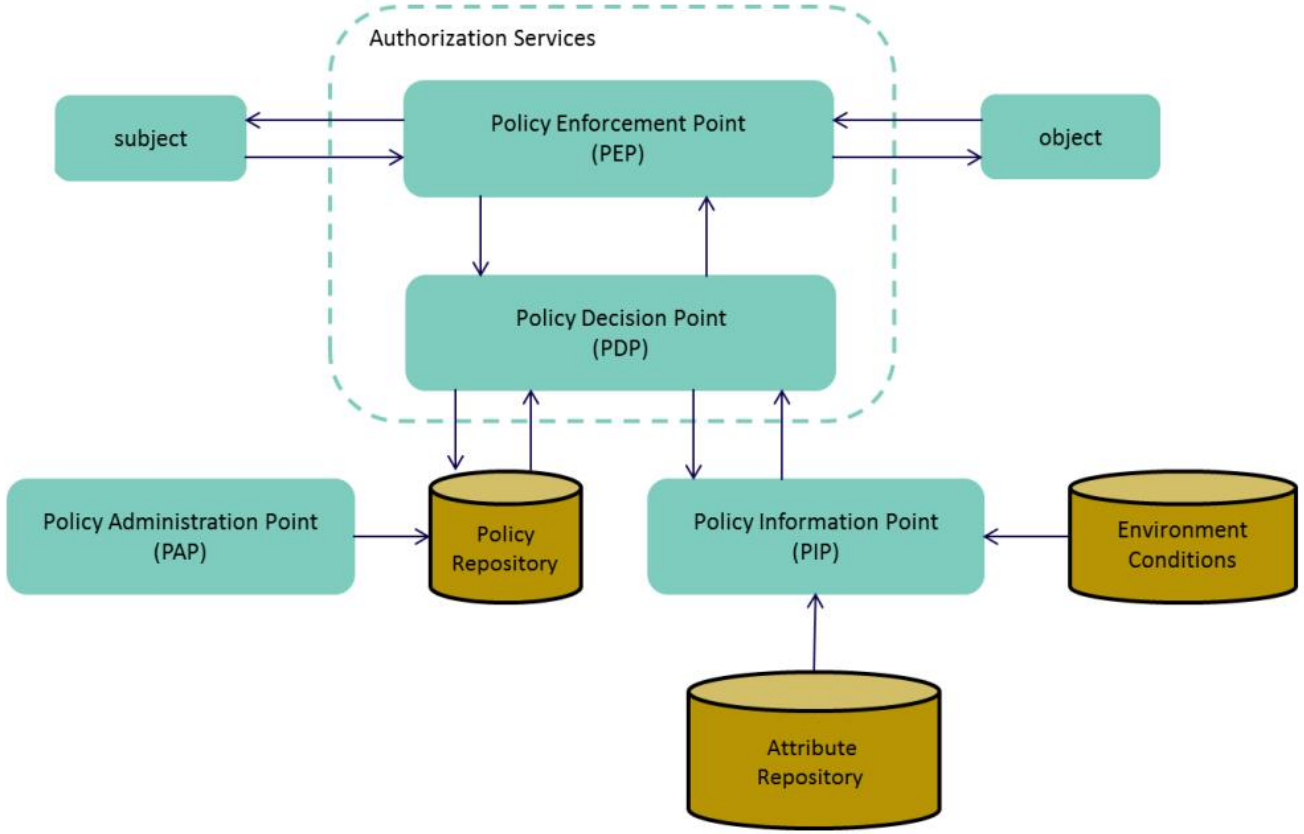
https://atuldewangan.medium.com/role-based-access-control-rbac-476452806518

AuthZ - ABAC
Attribute Based Access Control



https://doi.org/10.6028/NIST.SP.800-162

AuthZ - ABAC NIST
Attribute Based Access Control



https://doi.org/10.6028/NIST.SP.800-162

AuthZ - PBAC
Policy Based Access Control

See ABAC

No seriously, it’s the same 😆

Basics done ...

let‘s talk about the meat in the soup.

OAuth2 - What is it?

“OAuth2 is an **authorization framework** that enables applications to obtain **limited access** to user [resources…] on an HTTP service, such as Facebook, GitHub, or Google.
It works by **delegating** user authentication to the service that hosts the user account and authorizing third-party applications to access the user[‘s resources …].”

– ChatGPT 4

OAuth2 - Nomenclature

Resource Owner (RO)

The resource owner is the user who authorizes an application to access their account.

Client

The client is the application that wants to access the user’s account.

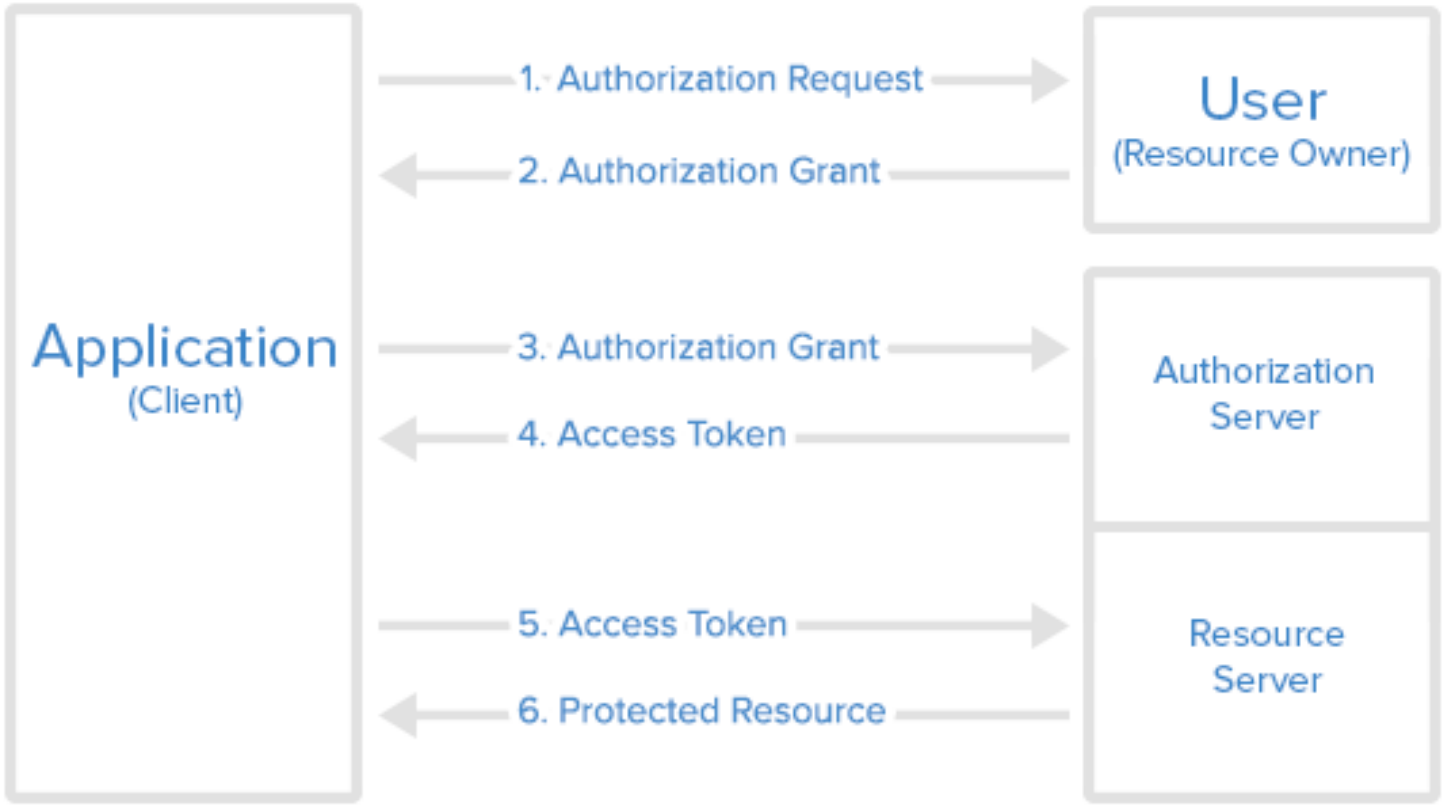
Resource Server (RS)

The resource server hosts the protected user accounts.

Authorization Server (AS)

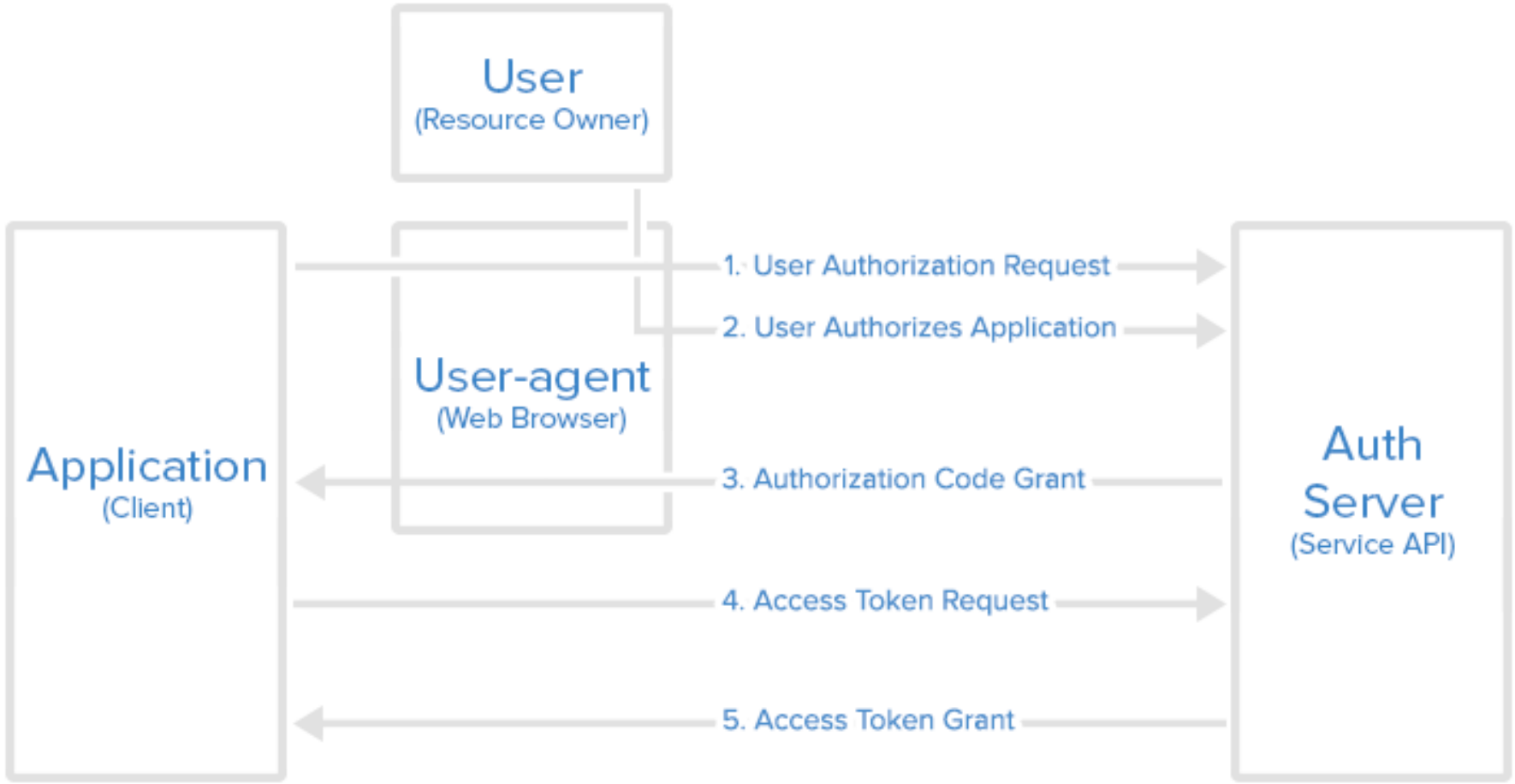
The authorization server verifies the identity of the user then issues access tokens to the application.

OAuth2 - General Concept



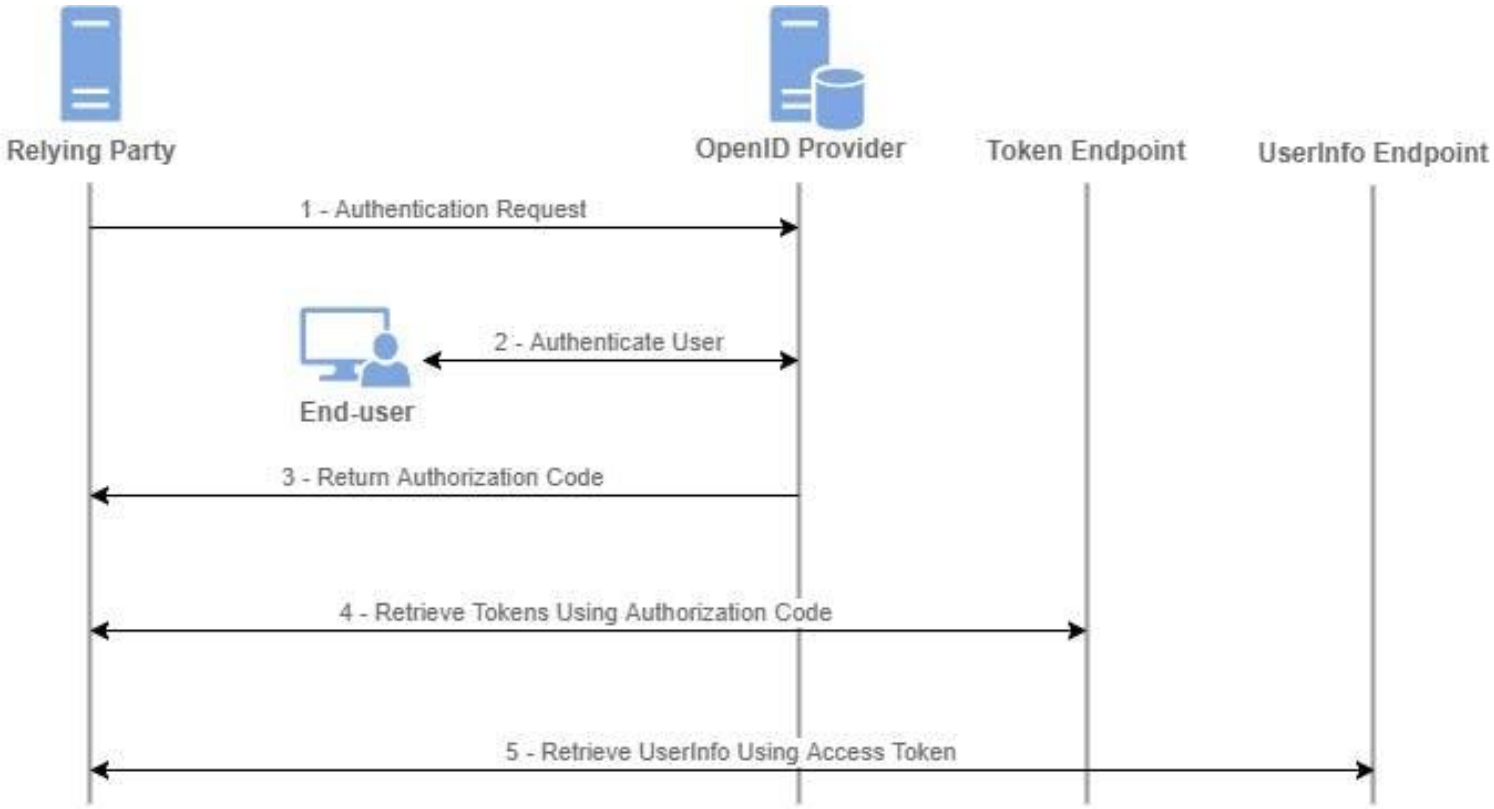
https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2

OAuth2 - Authorization Code Flow



https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2

OpenID Connect (OIDC) (not OpenID)



Looks familiar right? OIDC is based on OAuth2 😉

OAuth2 vs OIDC - What's the difference?

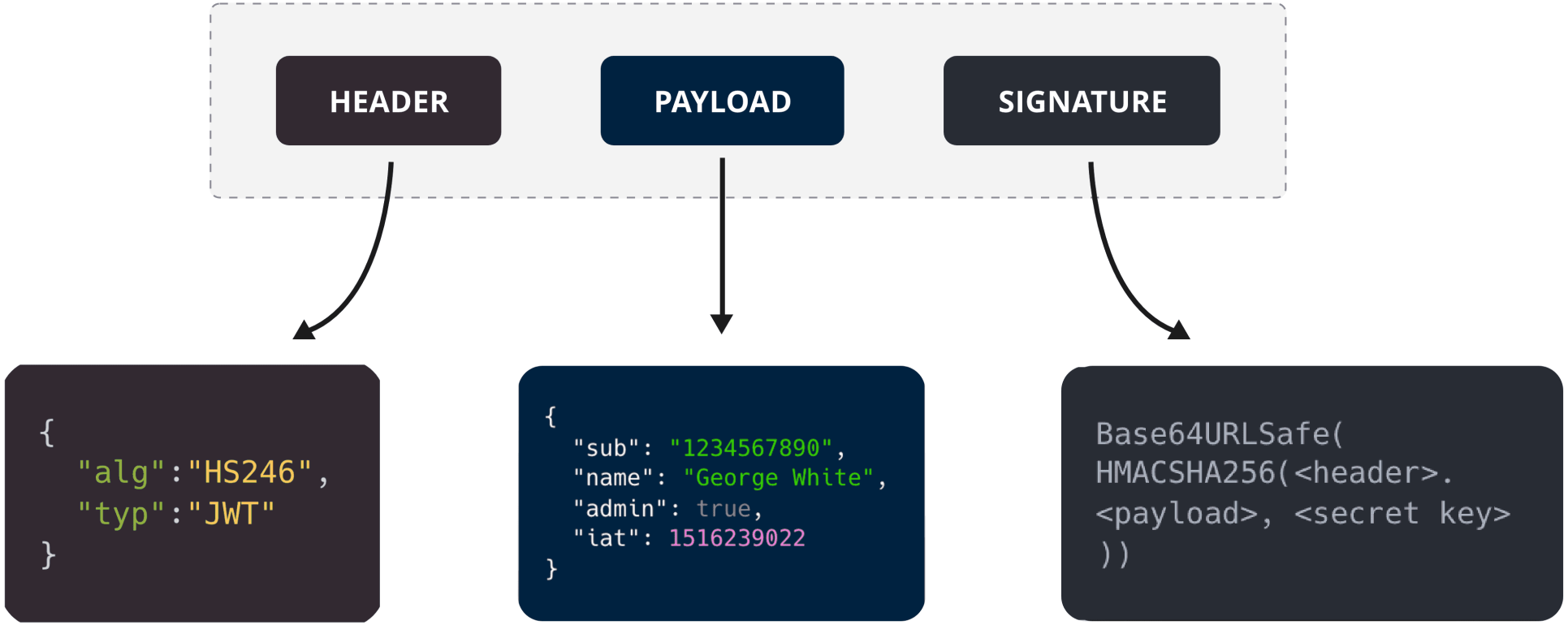
- OIDC is based on OAuth2 => Better question: Why OIDC?
- OAuth2 is for authorization (AuthZ), but not authentication (AuthN)
- You know that the client is allowed to do it, but not who allowed it to do that. (as RS)
- Why? OAuth2 (Access-)Token are opaque => Just strings => No context.
- There was a "general habit" to call an API Endpoint to gather that information (/userinfo)
- OIDC is mainly the standardization of this old habit + a well-defined Token-Format (JWT)
- OIDC Tokens are called "ID Token" => Json Web Token (JWT) formatted.

Json Web Token (JWT)

pronounced: jot

For german-speakers please don't call it Jay-Wee-Tee. It's like "handy" or "vee-LAN" 😅

base64(Header<Json> + Payload<Json> + Signature)



Basics+ done!

Thank you so far – Questions?

OIDC IdPs & Keycloak

What, why and how?

What is an IdP?

An **Identity Provider (IdP)** like **Keycloak** is a system that manages user identities and provides authentication and authorization services to applications. It helps centralize user management, allowing multiple applications to delegate authentication to a single trusted service.

- ChatGPT 4o

Why using an IdP?

- Secure Authentication (2FA, OTP, …)
- Credential-Storage offloading
- Advanced Authorization (RBAC, ABAC)
- Single Sign-On (SSO)
- Federation & Identity Brokering
- User Management & Self-Service
- Protection against attacks (pattern detection, anti-bruteforce, …)
- Auditing / Information-Security

Why Keycloak?

- Open-Source (no “premium” features)
- Supports SAML, OAuth2/OIDC, LDAP, KRB5 and IdP Federation
- Supports modern MFA (OTP, WebAuthN, Passkeys)
- Highly customizable using Java/Quarkus
- HA/Load-Balancing via Infinispan
- Automation: K8s Operator, Terraform, Ansible, Admin CLI
- Mature Project (started 2013)
- Big community (especially in Austria)
- Enterprise Support available via RedHat (RHBK)
- Multiple Databases supported (Postgres, MariaDB, MySql, Oracle, MSSQL)

So far so good ...

Demo Time? Not yet!

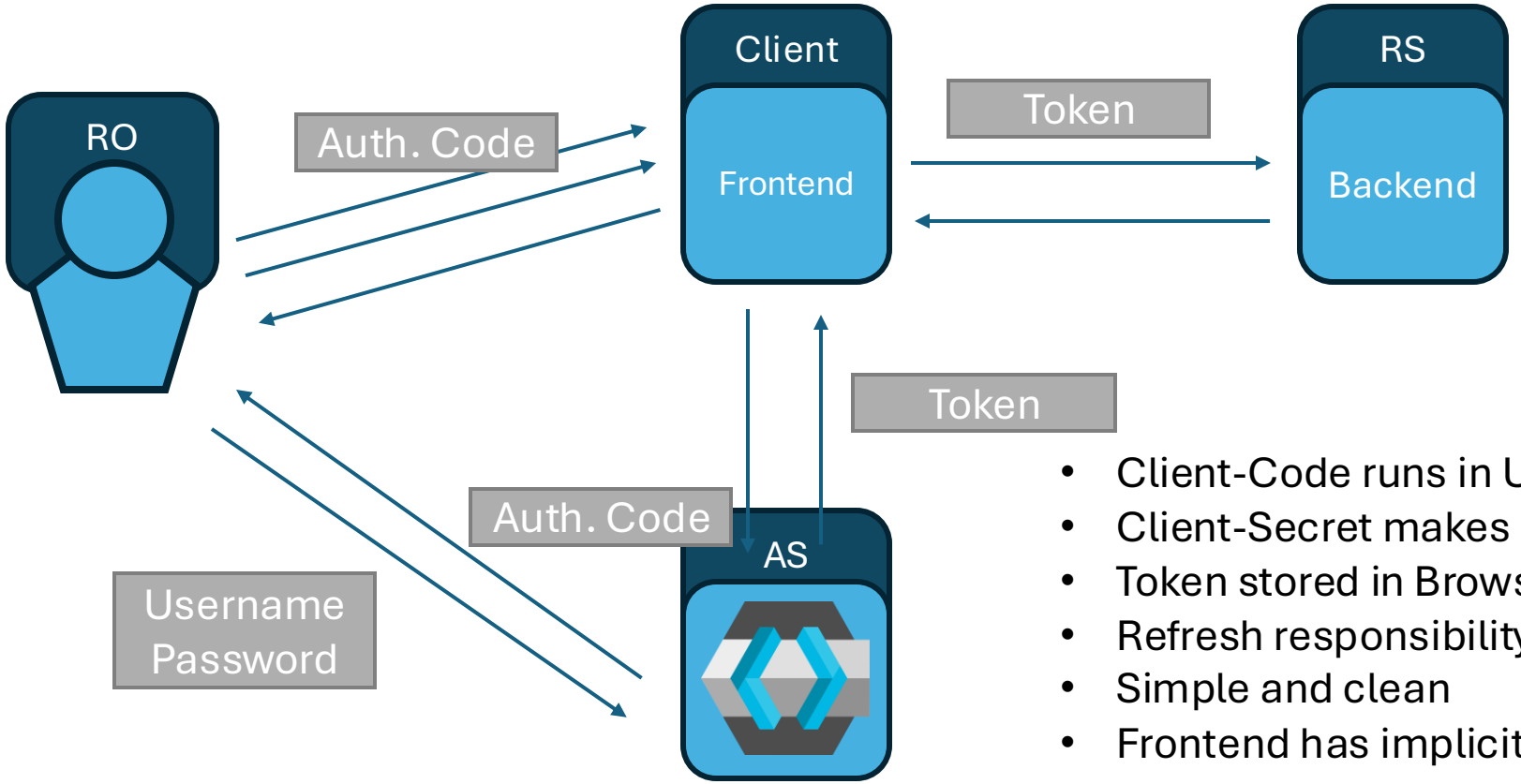
Challenges integrating Keycloak
or any other IdP

- Where are is our user-store?
  LDAP? Database? Other IdP like ADFS/EntraID?
- Where and who are our clients?
  Internal Applications, Mobile Apps, B2B Customer API?
- Do we need HA and Load-Balancing?
  SLAs? expected load? How many users, clients, logins/min, etc.
- What Database can/should we use?
- What is the Environment? Windows, Linux, K8s, VMs, Bare-Metal?
- What about Timeouts, User-Locking and other Policies?

Challenges integration **with** Keycloak

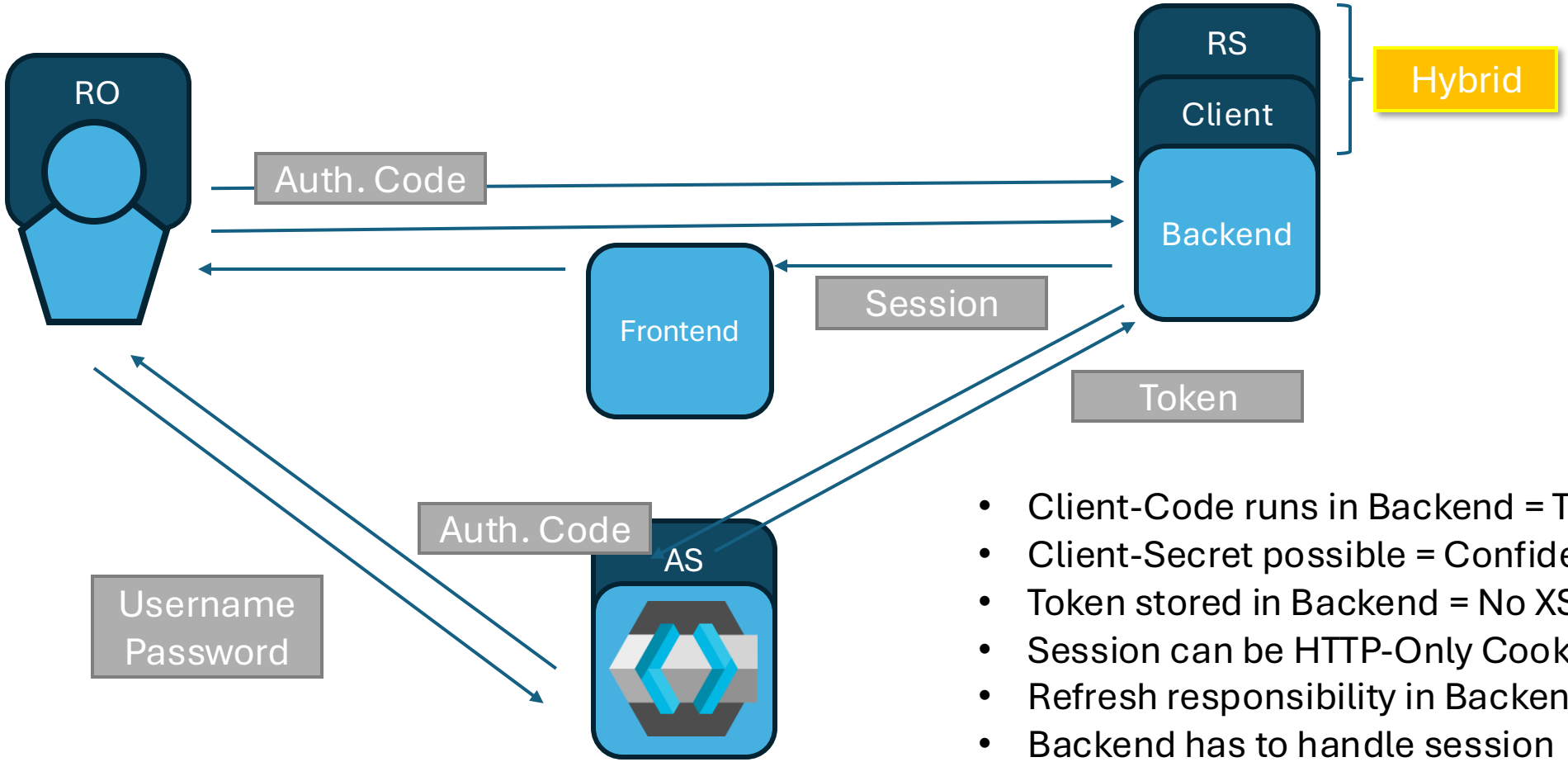
- What and where is the actual client (component that holds the token)? Frontend, Backend, Proxy, Mobile-App, Desktop-App?
- How can I communicate with the IdP?
  Only on the back-channel/front-channel? Not at all? HTTP/HTTPS?
- What's about logout? Is it essential? Single-Logout needed?
- Does the client act in-behalf-of the user? Is impersonation needed?
- Does the client need access to services on its own? (Service Account, Offline-Token, etc.)
- How does the authorization work? Who decides on what?
- Do I have technical constraints on the clients? Memory, CPU, etc.

Example Case: SPA



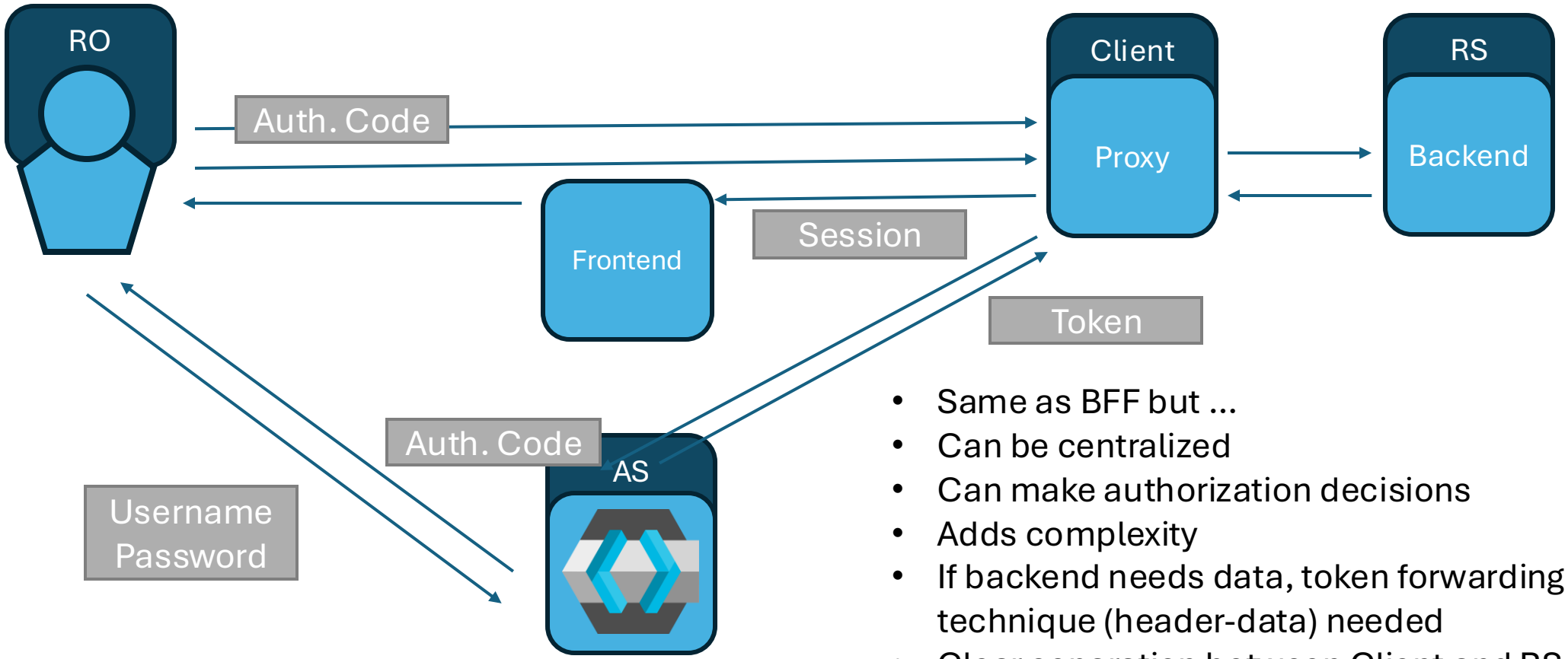
- Client-Code runs in User-Browser = Untrusted
- Client-Secret makes no sense = Public Client
- Token stored in Browser = XSS Attacks possible
- Refresh responsibility in Browser
- Simple and clean
- Frontend has implicit access to Token-Info

Example Case: Backend-For-Frontend



- Client-Code runs in Backend = Trusted
- Client-Secret possible = Confidential Client
- Token stored in Backend = No XSS Attacks
- Session can be HTTP-Only Cookie
- Refresh responsibility in Backend
- Backend has to handle session
- No implicit user-info in frontend

Example Case: Authenticating Proxy



- Same as BFF but …
- Can be centralized
- Can make authorization decisions
- Adds complexity
- If backend needs data, token forwarding other technique (header-data) needed
- Clear separation between Client and RS

Finally ...

Demo Time!

Demos

- Setup
- Keycloak
- Quarkus App
- Quarkus App SwaggerUI
- Single Page Application (SPA)
- Custom User Attributes with custom validator
- Custom Themes
- Magic Link
- Webhook
- Tracing
- Config As Code

Tips, Tricks & FAQ

How to not crash and burn with Keycloak in Production

Tip 01: Protect your Master-Realm (!!)

- Never ever use the master-realm to serve enduser-facing clients in production! Create a separate realm and use this one
- Reason 1: Exposing the master-realm is a risk that you do not have to take. If you have any reverse-proxy or WAF in place, protect the master realm urls. If you have used it for customer-application you cannot do that.
- Reason 2: DoS Lockout Attacks. If someone bruteforces your master-realm and you have any lockout policy in place (ldap or bruteforce detection) you get locked-out on your master realm.
- Reason 3: Authentication Flow mess-ups. If you mess-up your authentication flow on the master realm you can lock yourself out pretty hard. I would recommend, don’t touch the authentication flows on your master realm.
- Reason 4: Making future multi-tendency changes hard. If you have a dedicated realm for your customers, adding another realm for another group of customers is easy. If you have used your master-realm rights-management might get tricky if your old admins should not get access to the new realm. Master-Realm Admins are global admins.

Tip 02: Pick the correct version

- There are roughly 4 minor releases each year and a major release every 2-3 years. (new)
- Red Hat Build of Keycloak (RHBK) is the enterprise-supported version, built from even releases. If you plan to use RHBK check that you are using the right Version of Keycloak.
- Security-Patches for Keycloak (not RHBK) are generally “latest minor only”. So, you should update regularly!
- If you have to freeze your version, at least do it on a RHBK compatible version. Backports from there are more likely and you can buy a subscription of push comes to shove.

Tip 03: Know your Database

- Postgres is by far the best supported and mostly used DB for Keycloak. Stick with it if possible.
- MSSQL and Oracle have shown some quirks in the past. If you have to use them, make sure you have a solid test-environment (e2e- and load-tests).
- There are some “optional” indices one could set if needed. In general, that is not necessary anymore but knowing the schema helps if performance issues come up.
- Especially if you are using a HA setup make sure your DB connection is fast enough. A latent or faulty connection makes Keycloak very slow.

Tip 04: Bruteforce Detection

- There is a built-in Bruteforce detection feature available.
- If you are using LDAP or another User-Store that has a lock-out policy on password retries to should enable this feature.
- You have to set the bruteforce detection slightly stricter than the lookout policy to avoid a DoS attack.
- Be careful with permanent lockout, especially if your master-realm is reachable from outside.

Tip 05: Check your Timeout Settings

- There is a bunch of different Timeout-Settings.
- One set is related to oauth tokens (access token, refresh token) and one set is modifying the keycloak session timeouts.
- Do not confuse them, your access-tokens should be short-lived (<10min) because you cannot revoke an access-token without extensive support on client and RS side.
- Access-Token should be timed out before the SSO-Session idles out. There are some confusing things happening if it’s the other way round.

Tip 06: Time Synchronization is essential

- As with all other cryptographical authentication systems, OIDC and JWT need time-synchronization between all components (Client, AS, RS). It is not always possible to guarantee this on clients, but at least the AS and RS should be using NTP to compensate time-drift.
- Timeout Settings are also affected by time-drift. There is no classical “drift” setting so you have to think about possible positive and negative time-drift when setting timeouts.
- 5 minutes is a conservative setting, 1 minute is ok if you need more security, below 30sec it might get unreliable.

Tip 07: Oh my Infinispan

- Infinispan is used as distributed temporary data storage and cache in clustered setups.
- This is needed because it is possible (and likely) that the user has initiated the authorization code flow on a different keycloak node than the client will access to swap the code for a token.
- There is some data that has to be shared: auth.codes, tokens, session-states, etc. besides that Keycloak uses Infinispan as cache.
- Because it is a in-memory solution that is also configured with a redundancy of 1 by default, data gets lost if a node restarts. So be careful with node-restarts and read the HA guide in the official docs.

Tip 08: ABAC / Permission API / UMA

- Keycloak offers a complex ABAC solution based on the UMA specification.
- You can activate it only on confidential clients (client authentication enabled), also direct access grants have to be allowed, and service accounts are activated if “Authorization” (=ABAC) is enabled.
- This feature DOES NOT affect token issuing. Authorization is only possible if clients actively call Keycloak and enforce the configured policy decisions on their side.
- There are some client implementations that provide support for this feature out-of-the-box.

Tip 09: Understand your DNS

- In default configuration (v19.0+) Keycloak needs a static hostname because it is written as issuer into tokens and used for auto-discover urls.
- Before that and if you disable strict_hostname modern versions keycloak uses the sent *host* (or x*forwarded*) header to determine its own hostname, using that for base-urls and issuer claims.
- This gives some kind of “multi-homed” behavior but is source of trouble most of the time.
- Usually this makes waves if your BFF Client is “seeing” the Keycloak under a different hostname then the client, generating wrong redirect urls.
- The other way round is also possible: If the Client sees the Keycloak differently from the RS, there is an issuer mismatch, leading to token-verification errors on the RS.

Tip 10: Don’t abuse features and endpoints

- Don’t use Keycloak session cookies as authentication, use token.
- Don’t reuse tokens between multiple clients, use token-exchange.
- Don’t use IDToken as access_token and vice versa.
- Don’t intercept endpoints to add features in production.
- Don’t use the Database as interface, use the API.
- Don’t build critical usecases on preview features.
- Don’t use scopes and groups as authorization, use roles.
- Don’t use groups as user-attribute-replacement, use attributes.
- Don’t call external/slow services from token mappers.

Stories to share?
Questions to ask?

Thank You!