Easily Secure your Microservices with Keycloak

Sébastien Blanc
Red Hat
@sebi2706

Keycloak ?

Keycloak is an open source **Identity** and **Access Management** solution aimed at modern applications and services. It makes it easy to secure applications and services with little to no code.

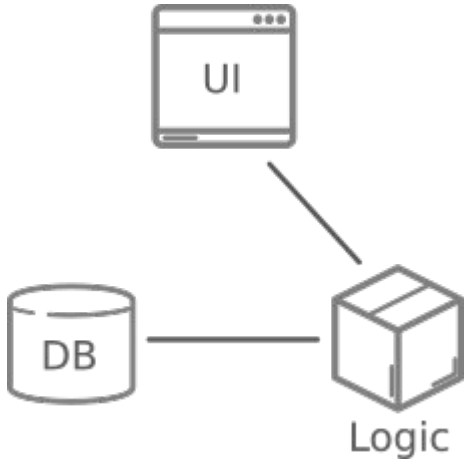
#1

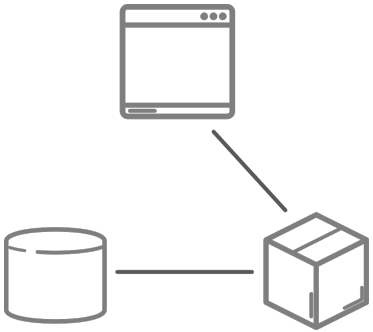
You want to stay DRY
(Don’t Repeat Yourself)

Implementing the security layer yourself

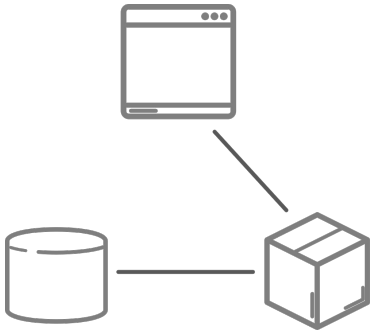
You need to deal with

- Authentication UI：
  - Login Form, Registration Form
  - User Profile Management
- Authentication Backend：
  - Check credentials
  - Store Users
  - Store Passwords
- Glue together the UI and the Backend with authentication flows
- Integrate it with your project.

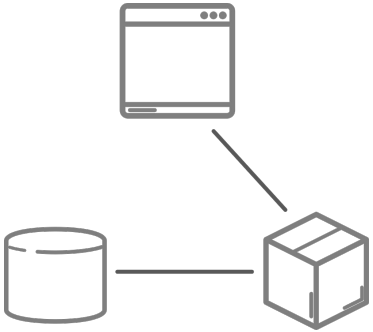




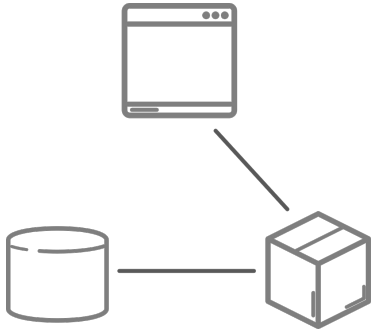
Project 1



Project 2



Project 3



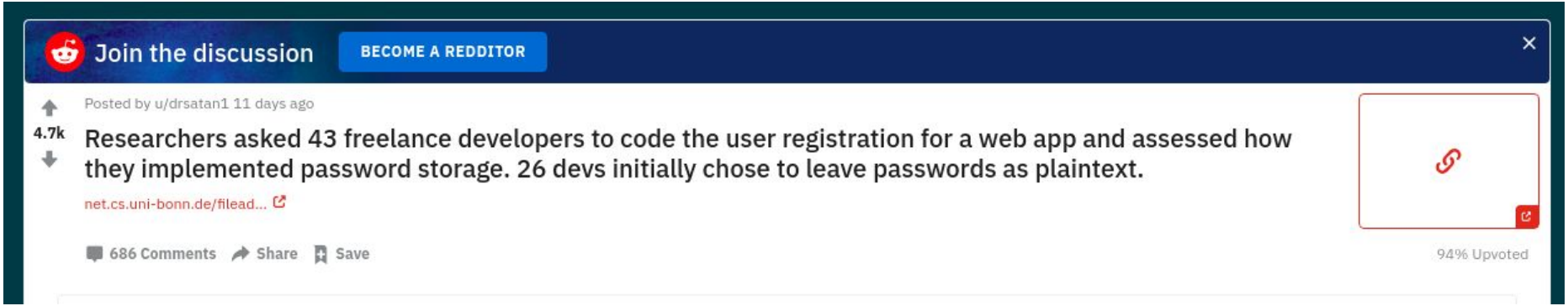
Project 4

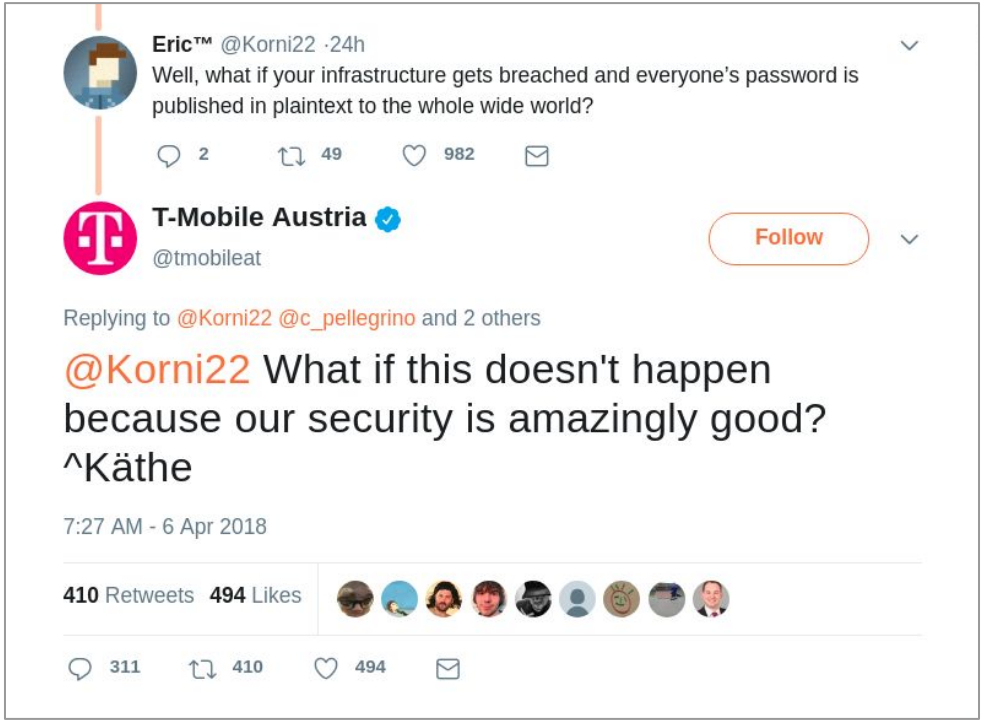
#2

It’s not just a username and a password

#3

You are probably not a security expert





Eric™ @Korni22 ·24h

Well, what if your infrastructure gets breached and everyone's password is published in plaintext to the whole wide world?

2 49 982

T-Mobile Austria

@tmobileat

Follow

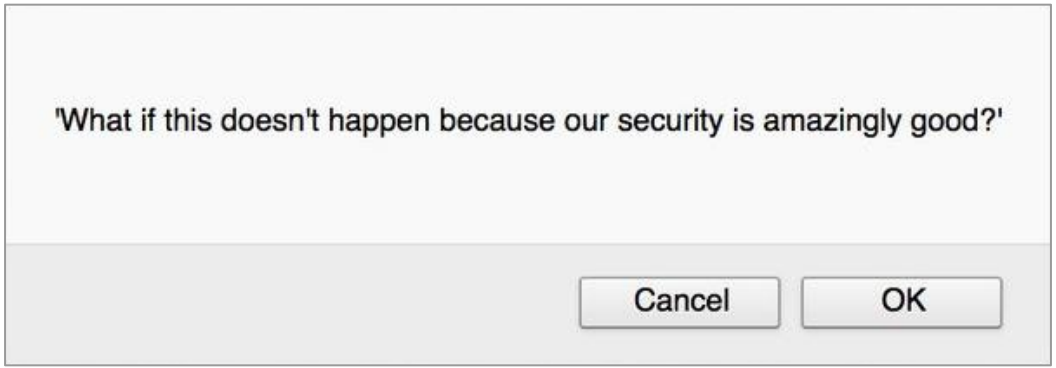
Replying to @Korni22 @c_pellegrino and 2 others

@Korni22 What if this doesn't happen because our security is amazingly good? ^Käthe

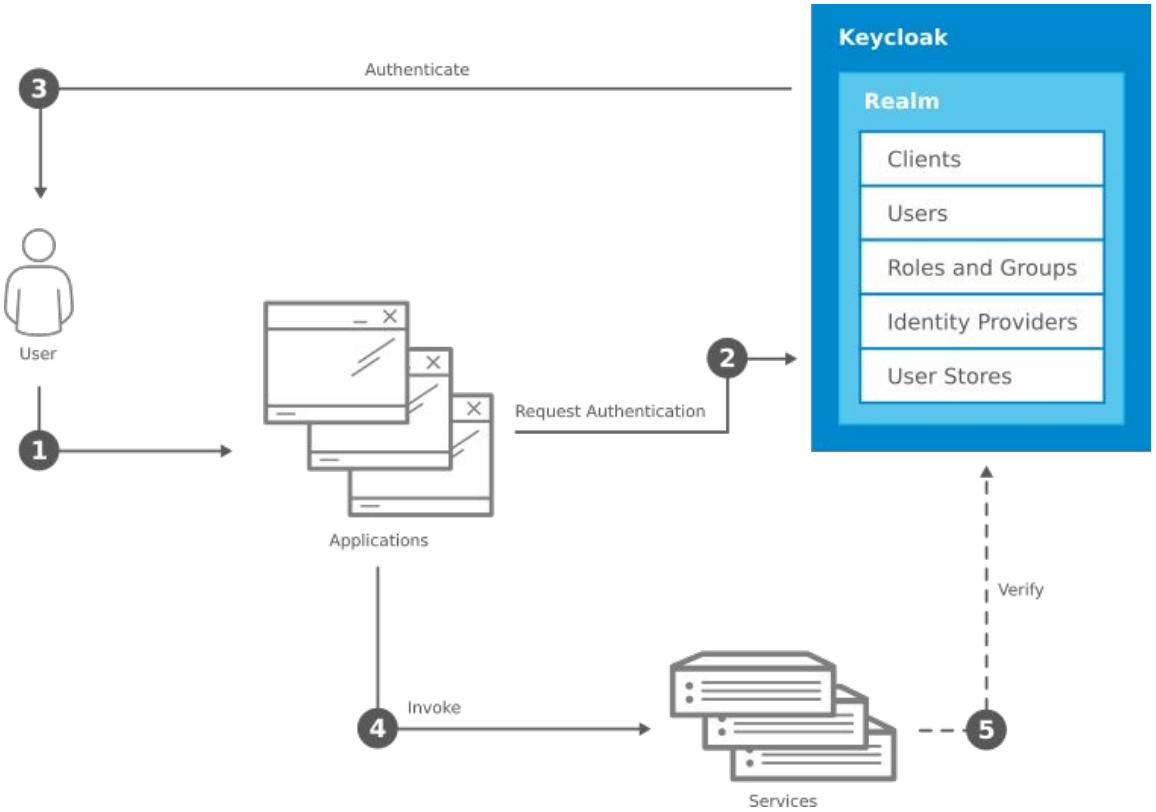
7:27 AM - 6 Apr 2018

410 Retweets **494** Likes

311 410 494


'What if this doesn't happen because our security is amazingly good?'
Cancel
OK

Concepts


Keycloak
Realm
Clients
Users
Roles and Groups
Identity Providers
User Stores
Authenticate
3
User
1
Applications
2
Request Authentication
Verify
4
Invoke
5
Services

Protocols

OpenId Connect

- JSON
- Simpler
- Bearer token

When to use

- Default
- Single-page apps, mobile
- REST services and **Microservices**

SAML

- XML
- More mature

When to use

- Monolithic applications
  - Or you don’t need end-to-end auth
- If your apps already support SAML
- If you have requirements OpenID Connect doesn’t support

You can use both!

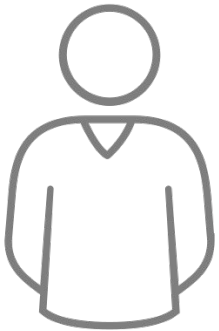
Obtaining the token

Using the OpenId Connect Token endpoint :

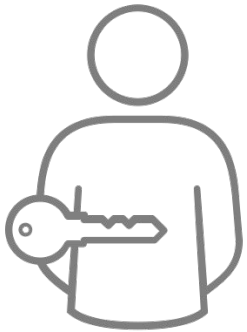
/auth/realms/{myRealm}/protocol/openid-connect/token

- Using Keycloak’s JavaScript Adapter
- Service accounts
- Direct Grant
- Offline Tokens

ID Token

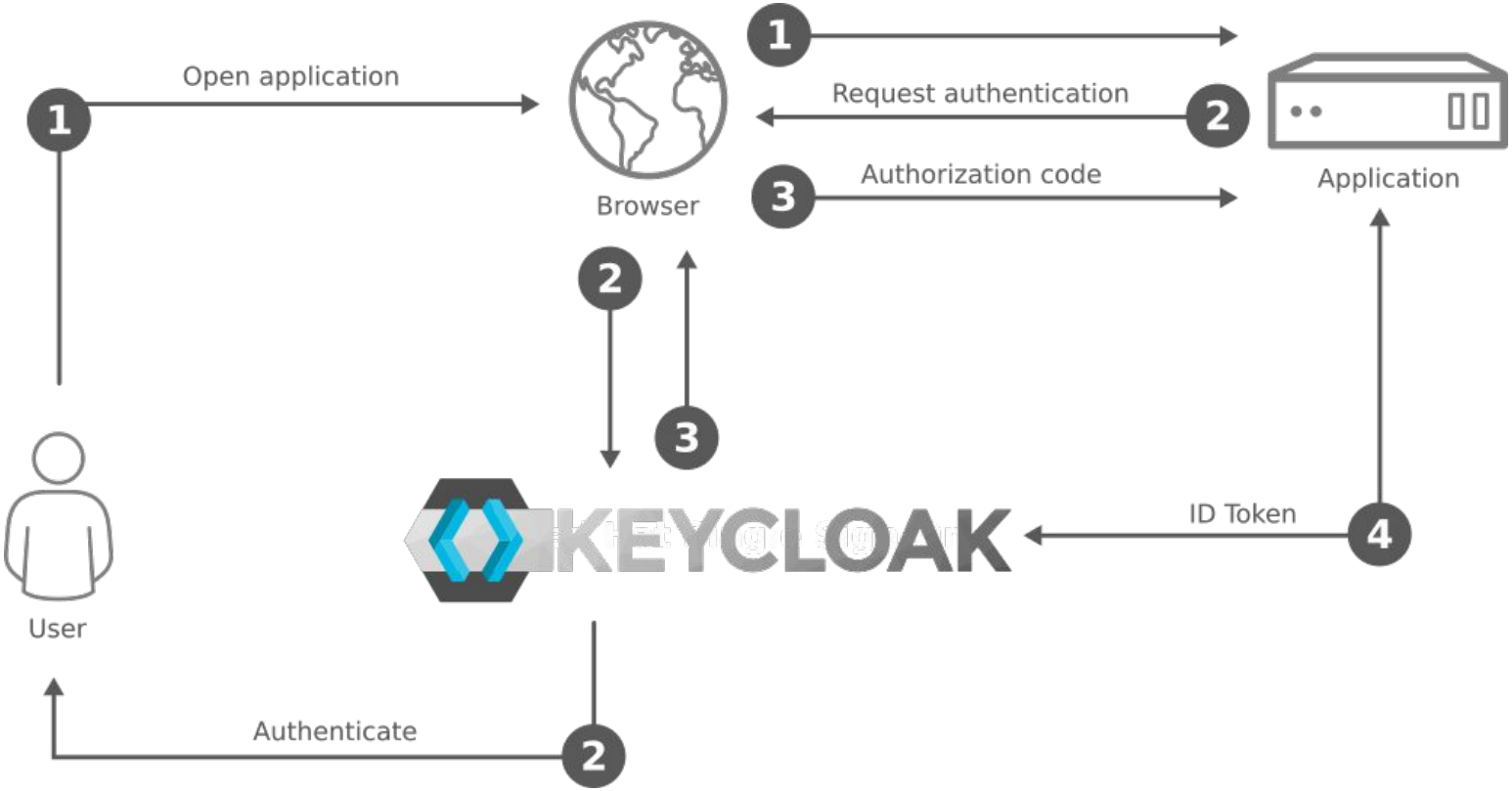


Access Token



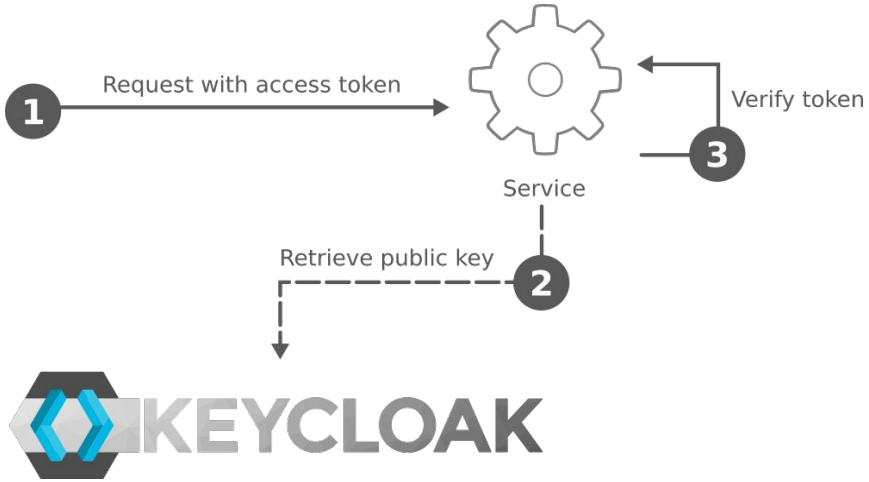
Refresh Token

Monolithic application

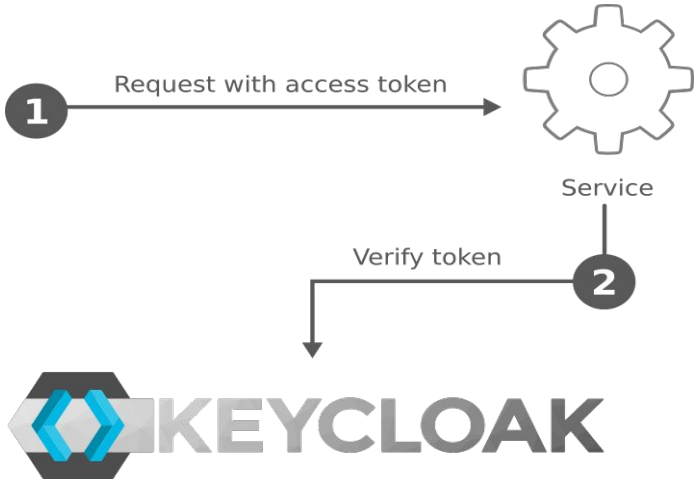


Service

Verify offline with signature



Verify online



Transport

```
var oReq = new XMLHttpRequest();
  oReq.open('GET', "http://localhost:9090/secured.php" , true);
  oReq.setRequestHeader('Authorization', 'Bearer ' + keycloak.token);
  oReq.send();
```

Transport

```
KeycloakSecurityContext session = (KeycloakSecurityContext)
req.getAttribute(KeycloakSecurityContext.class.getName());

HttpClient client = new DefaultHttpClient();

try {

   HttpGet get = new HttpGet(UriUtils.getOrigin(req.getRequestURL().toString()) +
"/database/customers");

   get.addHeader("Authorization", "Bearer " + session.getTokenString());
```

Transport

```
private val restTemplate =  RestTemplate()

fun revertHello(request: HttpServletRequest, principal: Principal) : String {

   val customizer = KeycloakRestTemplateCustomizer()

   customizer.customize(restTemplate)

   if(principal.name.equals("sebi")){

      return restTemplate.getForObject("http://localhost:8082/revert", String::class.java)

   }
```

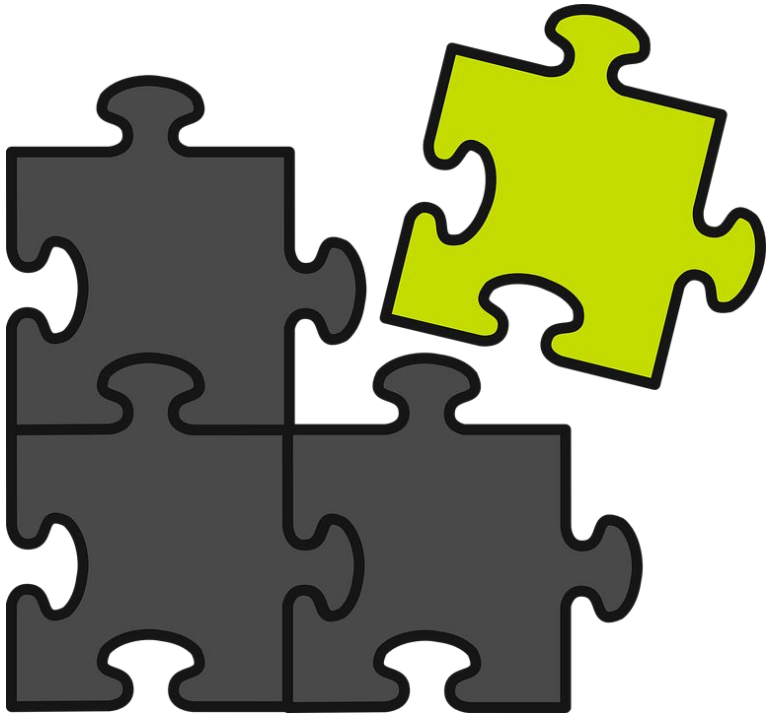
Obtaining the public key

Using the JWKS Endpoint :

/auth/realms/{myRealm}/protocol/openid-connect/certs

- One active key
- 0 or several passive keys
- Will be matched with the KID provided by the token
- Rotate your keys frequently !

The Adapters



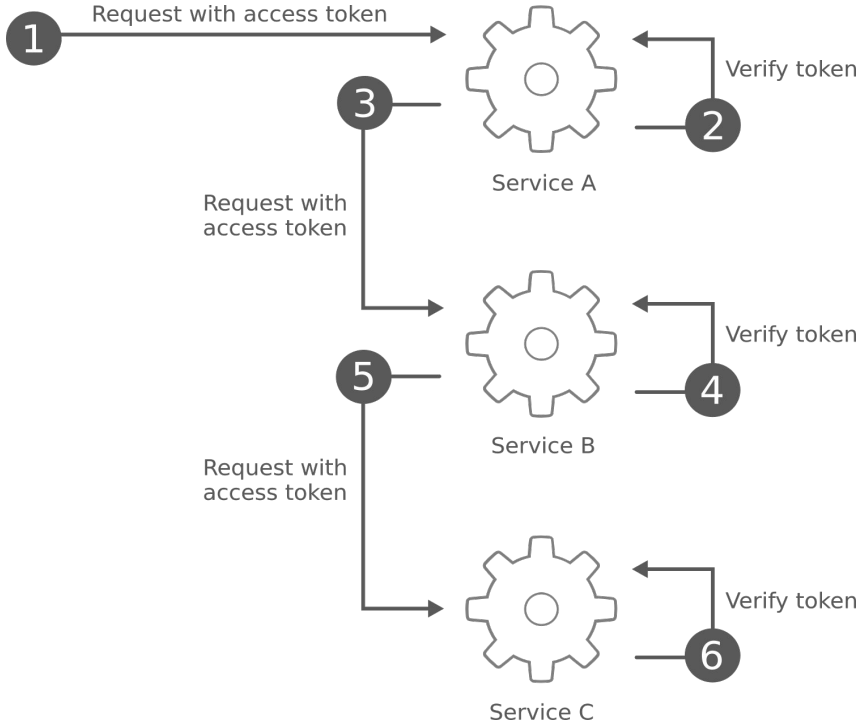
The Adapters

- Handles the Redirects
- Retrieves the public key
- Verifies the token :
  - Issuer
  - Expiration
  - Signature
- Exposes back channel : i.e Revoke Token
- Integrates with the underlying Security Context/System

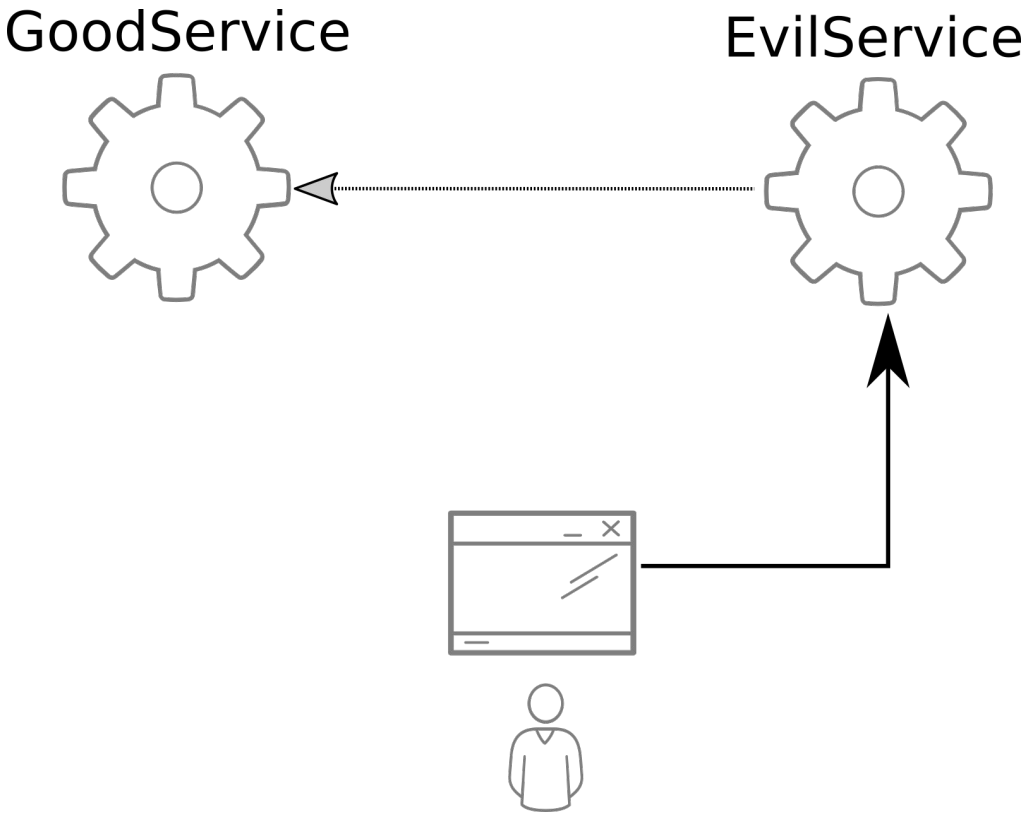
Other options

- Any OpenID Connect library
- Microprofile JWT Extension
- Quarkus adapter (WIP)
- Keycloak Gatekeeper
- Envoy / Istio

Microservices



Limit your Audience

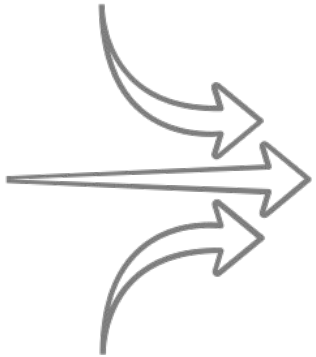


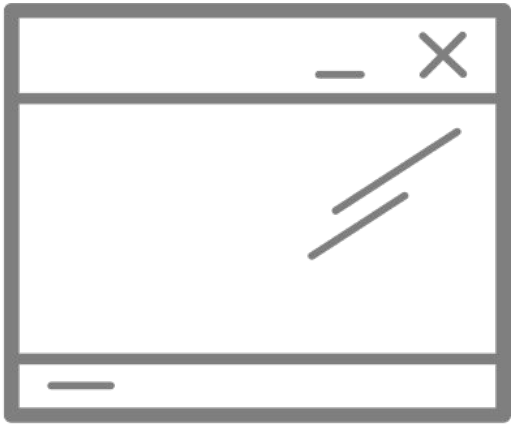
Limit your Audience

- verify-token-audience = true
  - ?scope=evil-service
  - Automatically add audience using client roles

Limit the Roles for a client

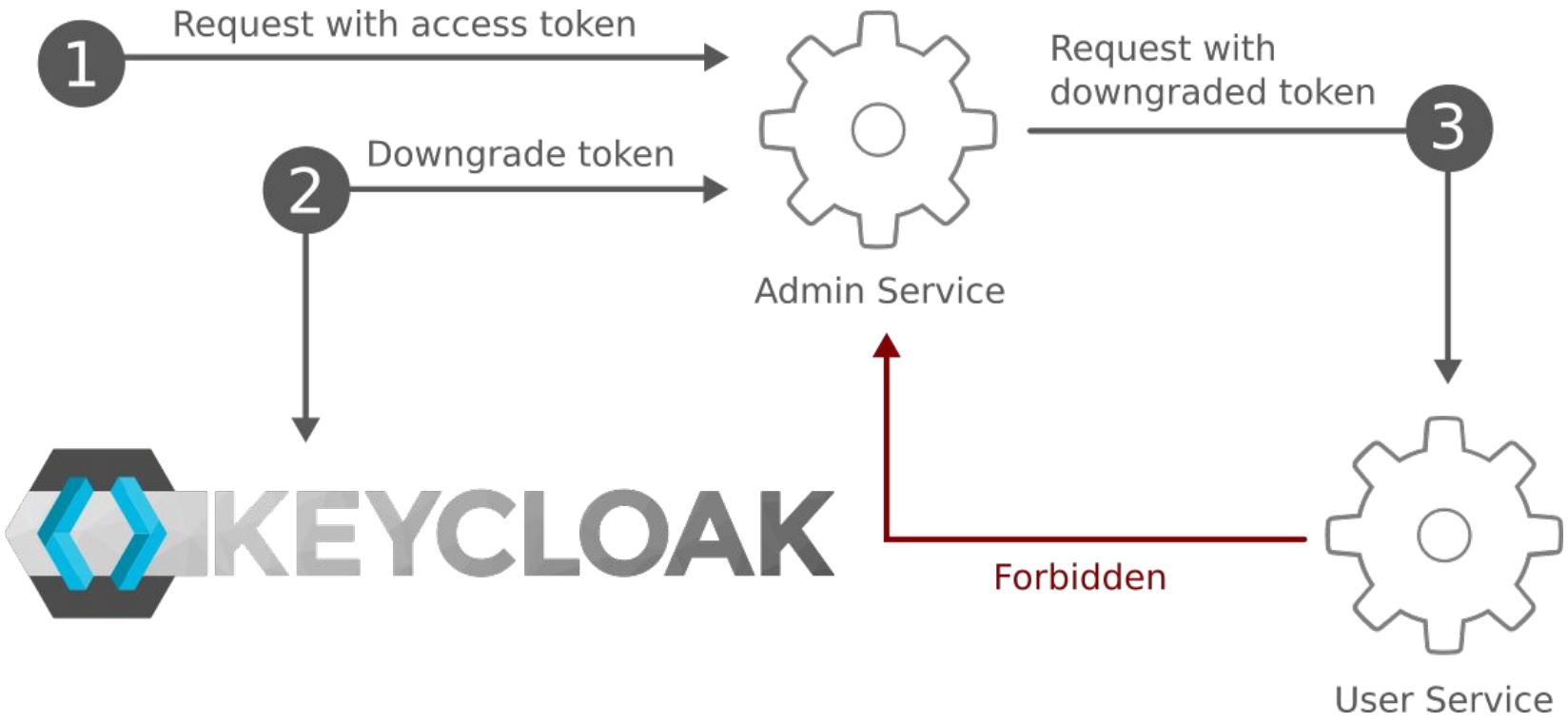
* User
* Admin
* Super Admin
* Uber Admin
* God



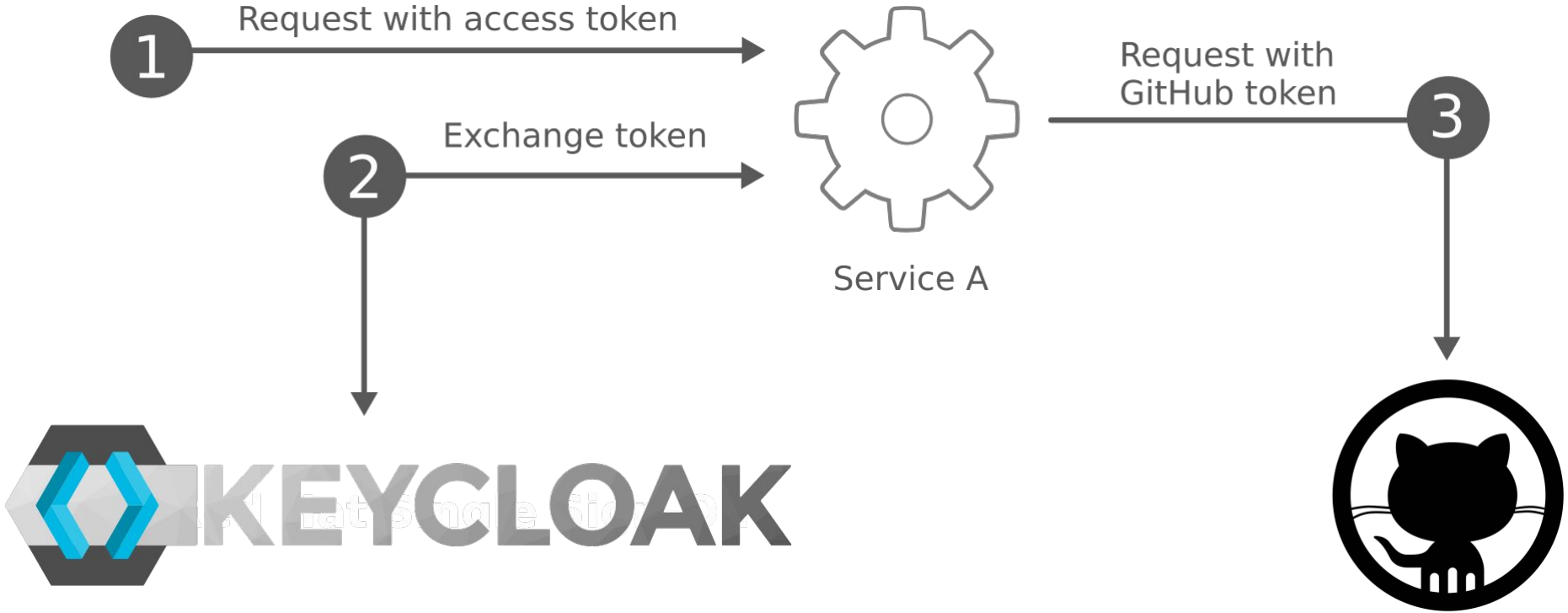


* User
* Admin

Token Exchange - Downgrade



Token Exchange - Exchange for external token

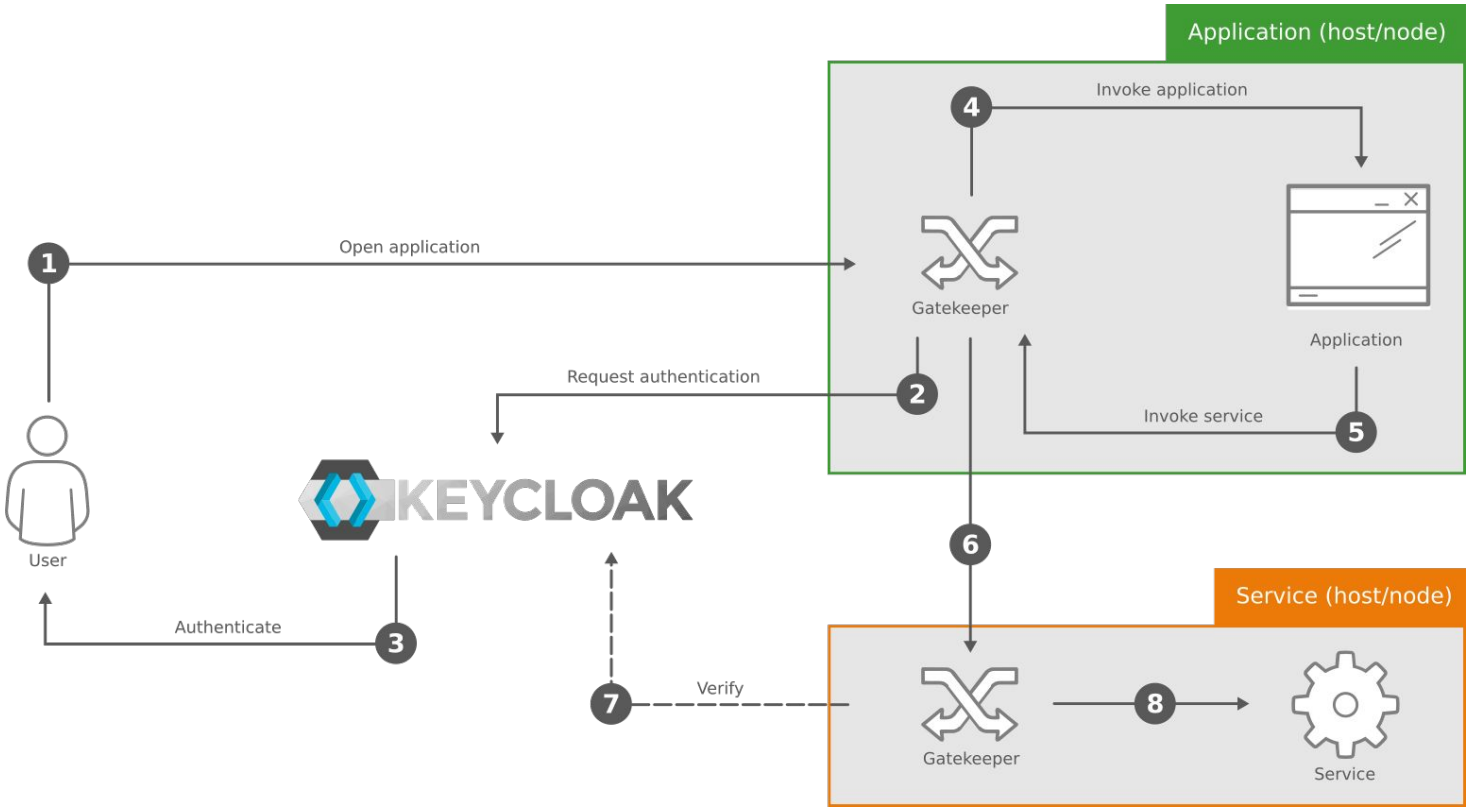


Authorization

- Authentification != Authorization
- Fine grained permissions :
  - Time policy
  - RBAC
  - Protection API
- UMA 2.0 Compliant
- RPT support

Sidecar proxy pattern

Keycloak Gatekeeper



Keycloak Gatekeeper

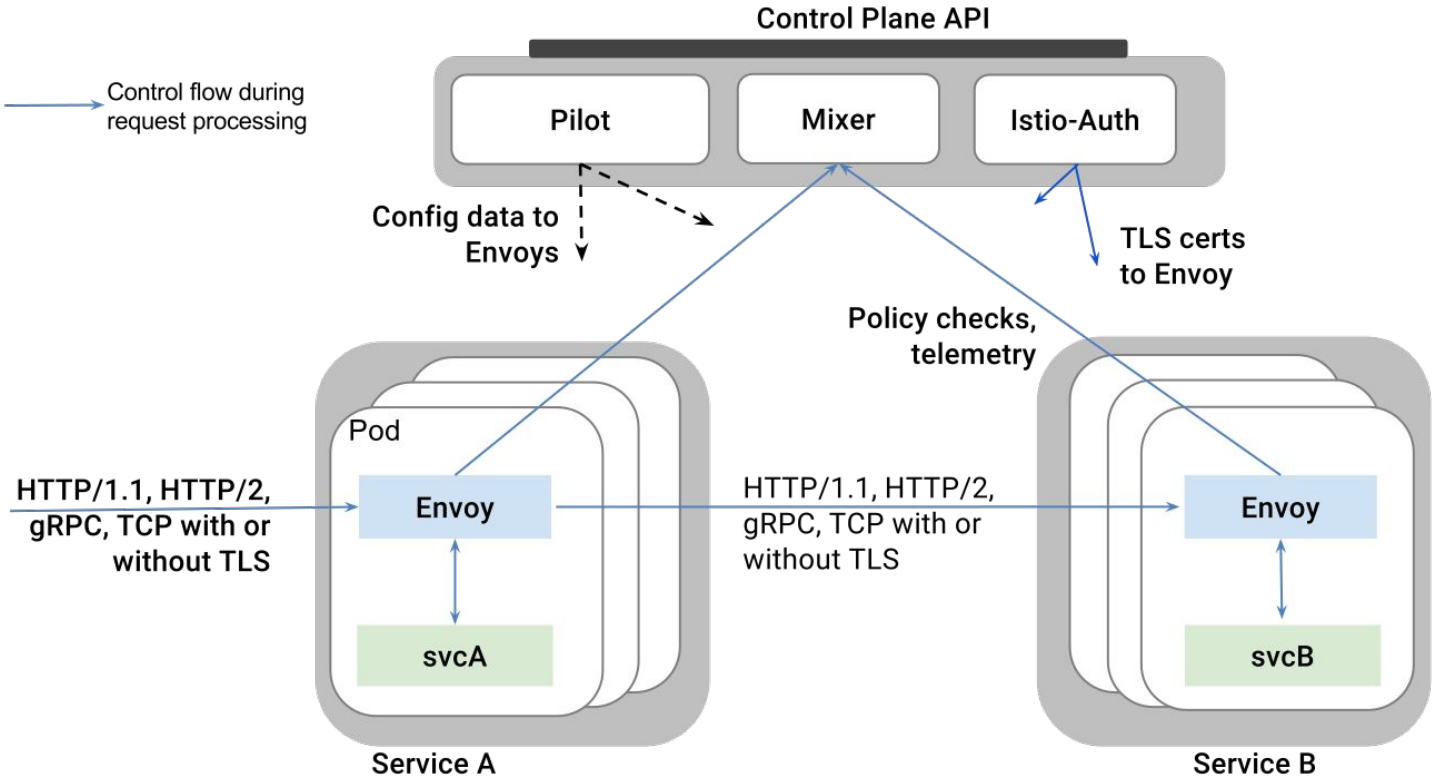
--add-claims = family_name

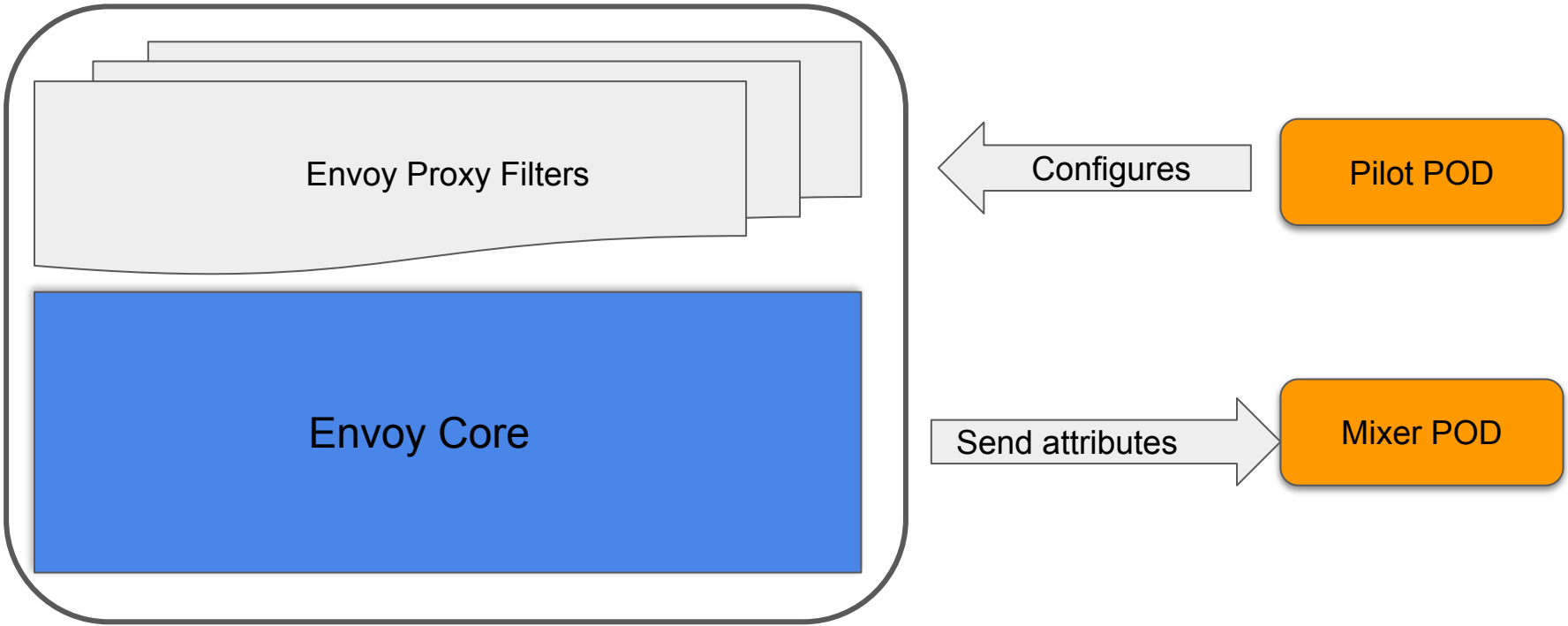
Header => X-Auth-Family-Name

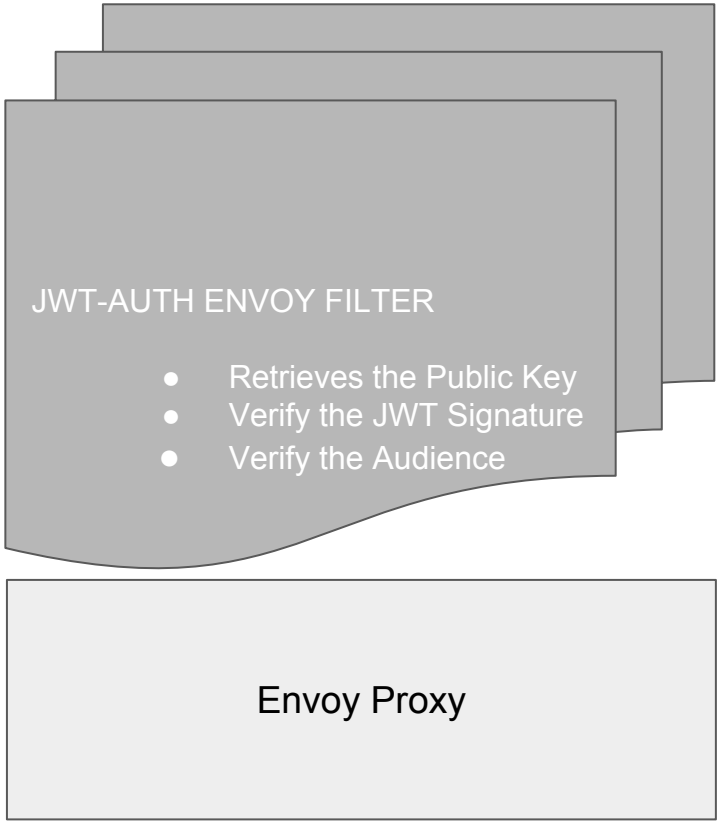
Forward-signing proxy

Let’s Encrypt OOTB support

Istio


Control Plane API
Control flow during request processing
Pilot
Mixer
Istio-Auth
Config data to Envoys
TLS certs to Envoy
Policy checks, telemetry
Pod
HTTP/1.1, HTTP/2, gRPC, TCP with or without TLS
Envoy
svcA
HTTP/1.1, HTTP/2, gRPC, TCP with or without TLS
Envoy
svcB
Service A
Service B


Envoy Proxy Filters
Envoy Core
Configures
Pilot POD
Send attributes
Mixer POD



configures

Pilot Script

```
apiVersion: config.istio.io/v1alpha2
kind: EndUserAuthenticationPolicySpec
metadata:
 name: cars-api-auth-policy
 namespace: myproject
spec:
 jwts:
   - issuer:
http://keycloak.myproject:8080/auth/realms/istio
     jwks_uri:
http://keycloak.myproject:8080/auth/realms/istio/prot
ocol/openid-connect/certs
     audiences:
     - cars-web
```

```
apiVersion: "authentication.istio.io/v1alpha1"
kind: "Policy"
metadata:
  name: "jwt-example"
spec:
  targets:
  - name: httpbin
  origins:
  - jwt:
      issuer: "testing@secure.istio.io"
      jwksUri: https://mykeycloak/auth/realms/admin/certs"
      principalBinding: USE_ORIGIN
```