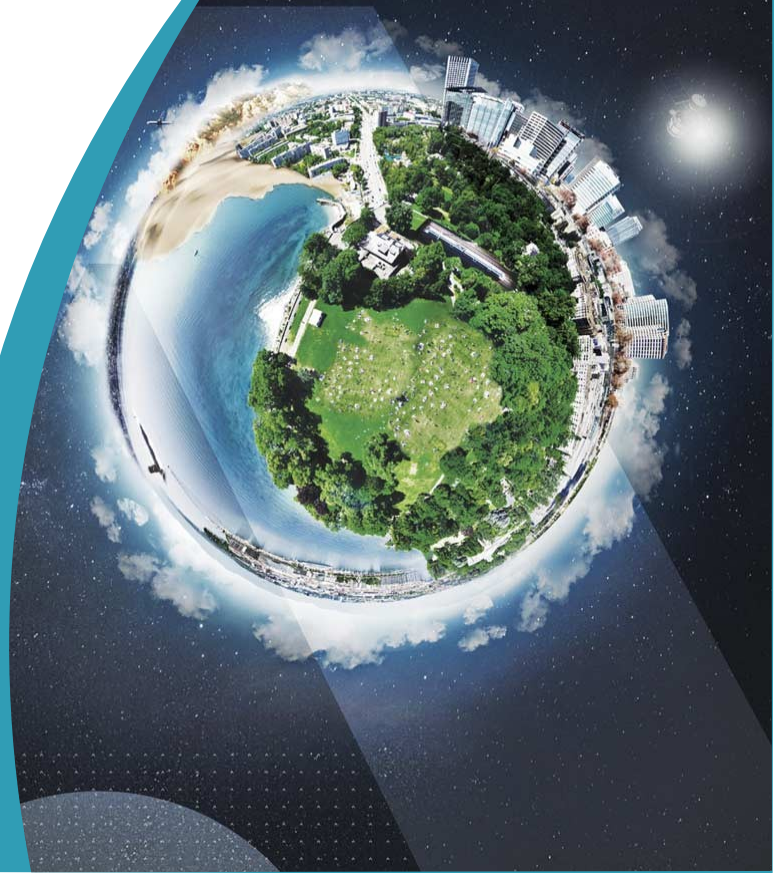
THALES

OpenIdConnect Deep Dive

Milan Jakobi
Thales

2/4/25

milan.jakobi@thalesgroup.com
+33 7 83 25 01 40



Agenda

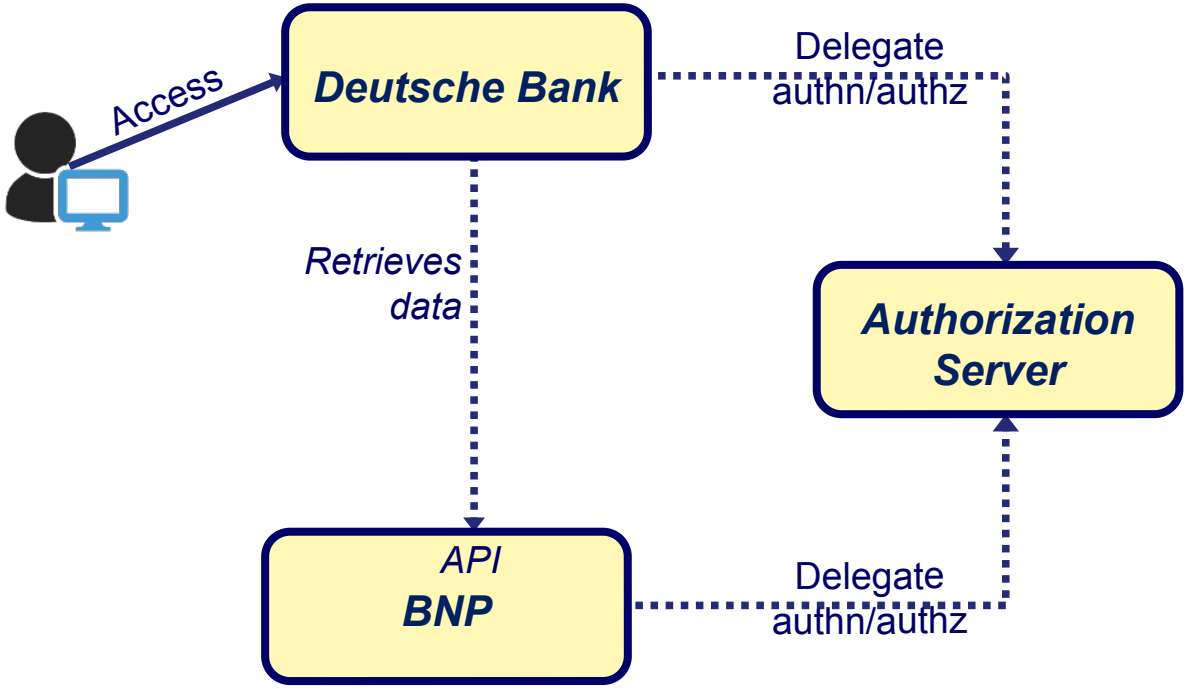
- **Introduction**
- **OAuth2/OIDC concepts**
- **OAuth2/OIDC flows**
- **Some advanced usages**

Authorization: problem statement

User has 2 bank accounts in 2 different banks :

- Deutsche Bank
- BNP

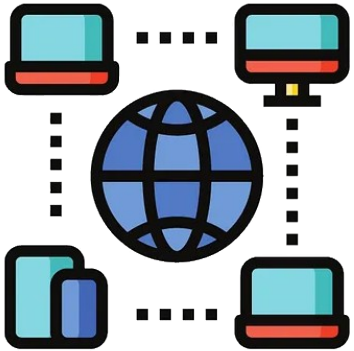
He mostly uses DBs management console but wants it to also display BNP account data, so he can manage everything in one place.



Protocols

- **OAuth2**
  - RFC 6749, onward...
  - Authorization oriented
  - Based on REST APIs and JWTs:
    - Access/Refresh token,
    - Etc.
- **OpenIDConnect**
  - Identity layer ontop of OAuth2
  - Needed for web SSO
  - Additional JWT : identity token

- **SAML**
  - XML/SOAP
  - Complex setup
  - (Was) promoted by Microsoft
  - In decline...



Agenda

- Introduction
- **OAuth2/OIDC concepts**
- OAuth2/OIDC flows
- Some advanced usages

OAuth2 roles

RFC 6749

- **Resource Owner**

An entity/end-user capable of granting access to a protected resource

- **Authorization Server**

The server issuing access tokens to the client

- **Client**

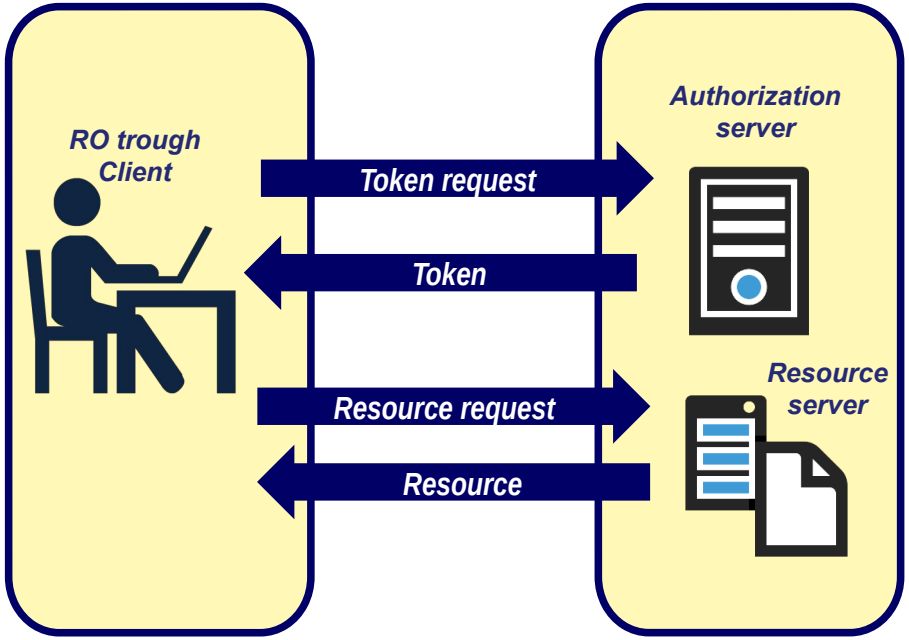
An application making protected resource requests on behalf of the resource owner and with its authorization.

- **Resource Server**

The server hosting the protected resources, capable of responding to protected resource requests using access tokens.

OIDC : Client & Resource Server merged into Relying Party

OAuth2 auth code flow diagram

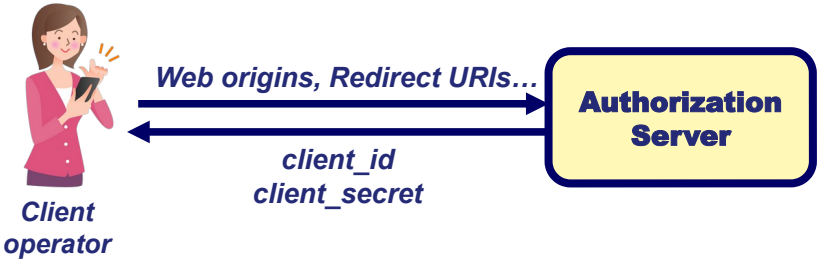


THALES

OAuth2 : registration & validation

Registration

- **Manual**: AS operator delivers client_id & client_secret
- **Dynamic**: automated registration using initial reg. code and /register endpoint
  - Mostly for mobiles



RFC 7951 & 7592

Token validation

- **Local**: relying parties check tokens validity following OAuth2/OIDC spec (signature, expiration…)
- **Introspection**: relying parties submits tokens to AS's /introspection endpoint

OAuth/OIDC JSON Web Tokens

RFC 7519

- **Access Token**
  delivered by AS for a (short) period of time. Enables client access to resource (web page, etc.) **on behalf of end-user.**
- **Refresh Token**
  delivered for a longer period of time in order to renew an (expired) access token (transparently for the end-user).

OAuth2

- **ID Token**
  provides identity oriented claims (gender, age…)
- **Userinfos**
  non standardized token. Carries end-user attributes (gender, work site...)

OIDC

Encoded	Decoded	Parts

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
.eyJpc3MiOiIiLCJpYXQiOjE2NzYyMTc5NTAsImV4cCI6MTcwNzc1Mzk1MCwiYXVkIjoiYWthbWFpLWJsb2ciLCJzdWIiOiIiLCJjb21wYW55IjoiQWthbWFpIiwidXNlciI6IkFrYW1haS1yZWFkZXIiLCJhZG1pbiI6Im5vIn0.kMPz3Z7BSlBTJKijD8bcrpzTZejX7VCZ77w5oQwJO6I
	{
	"typ": "JWT",
	"alg": "HS256"
	}
		Header

	{
	"iss": "",
	"iat": 1676217950,
	"exp": 1707753950,
	"aud": "akamai-blog",
	"sub": "",
	"company": "Akamai",
	"user": "Akamai-reader",
	"admin": "no"
	}
		Payload

	HMACSHA256(base64Encode(header) + "." + base64Encode(payload), secret_key)	Signature

Claims

THALES

Scopes

- **Scopes: collection of claims in tokens**
- **Standard scopes**
  - profile, email, **openid**
- **OpenID scope**
  - Subject + standard claims
- **Profile scope**
  - Name, family_name, given_name, middle_name, nickname, picture, updated_at + standard claims

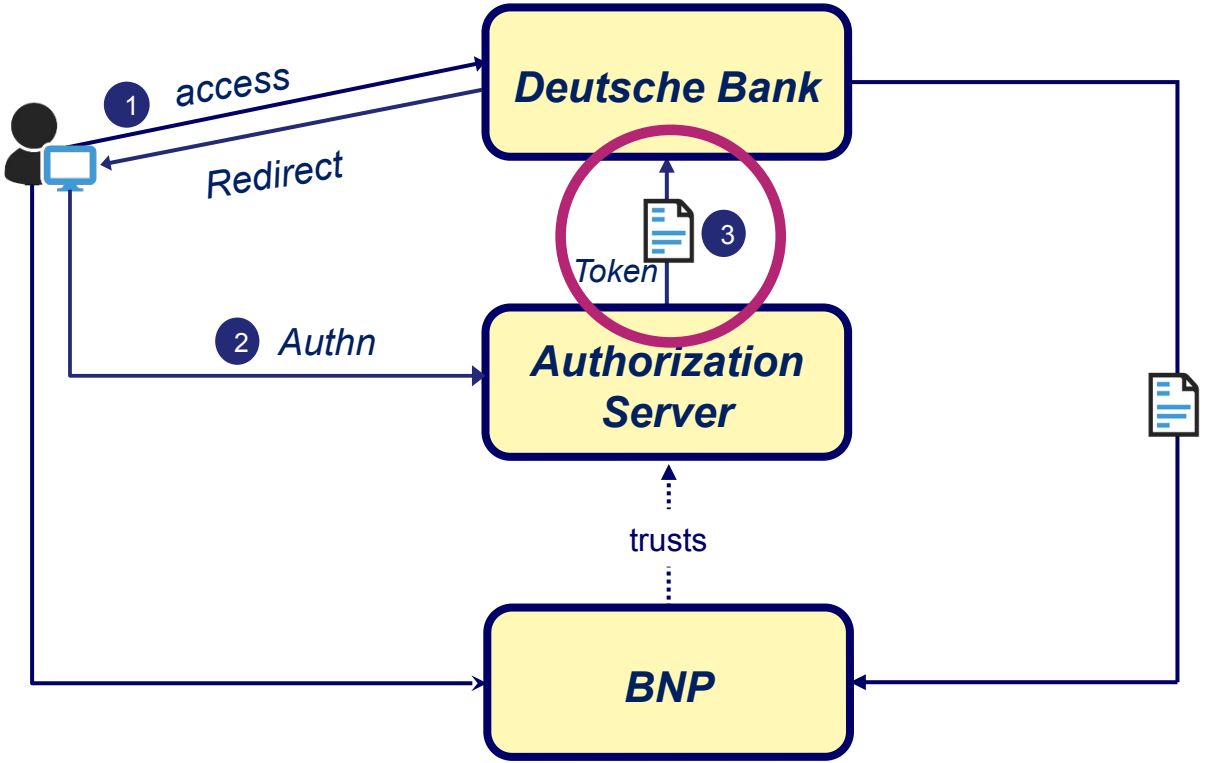
Standard claims:

- **iss**: URL of the AS (issuer)
- **sub**: subject of the token (end-user UUID)
- **aud**: audience, client_id of the token's requester
- **exp**: expiration time of the token
- **iat**: token's time of issuance
- …

Agenda

- Introduction
- OAuth2/OIDC concepts
- **OAuth2/OIDC flows**
- Some advanced usages

OAuth2: Authorization Flow



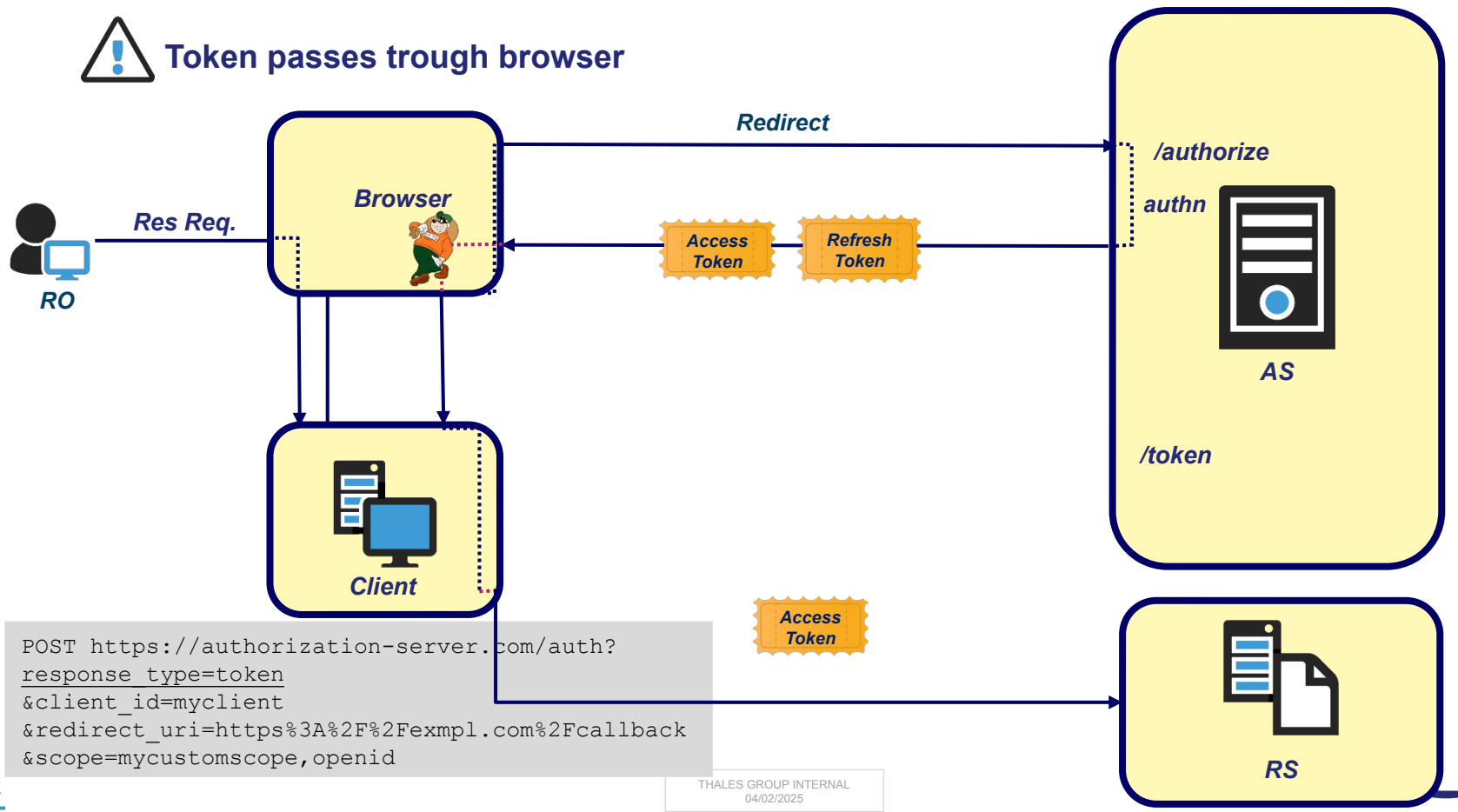
Establish trust between Relying Parties and Authorization Server

- Various ways to retrieve tokens from an Authz server
- Flows determines condition on which tokens are delivered
  - Depends on client types (Web/mobile/IoT…)

THALES

OAuth2 : Implicit Flow

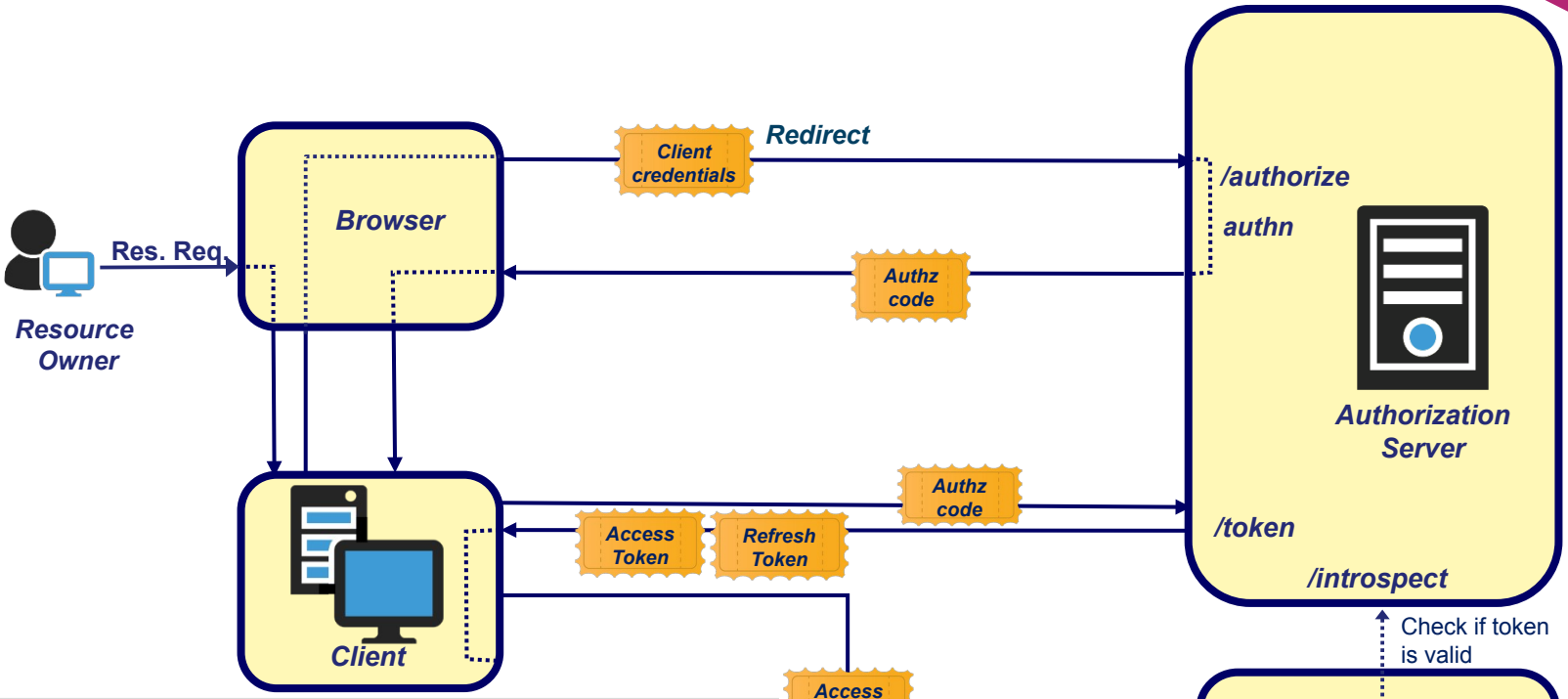
Obsolete




```
POST https://authorization-server.com/auth?
response_type=token
&client_id=myclient
&redirect_uri=https%3A%2F%2Fexmpl.com%2Fcallback
&scope=mycustomscope,openid
```

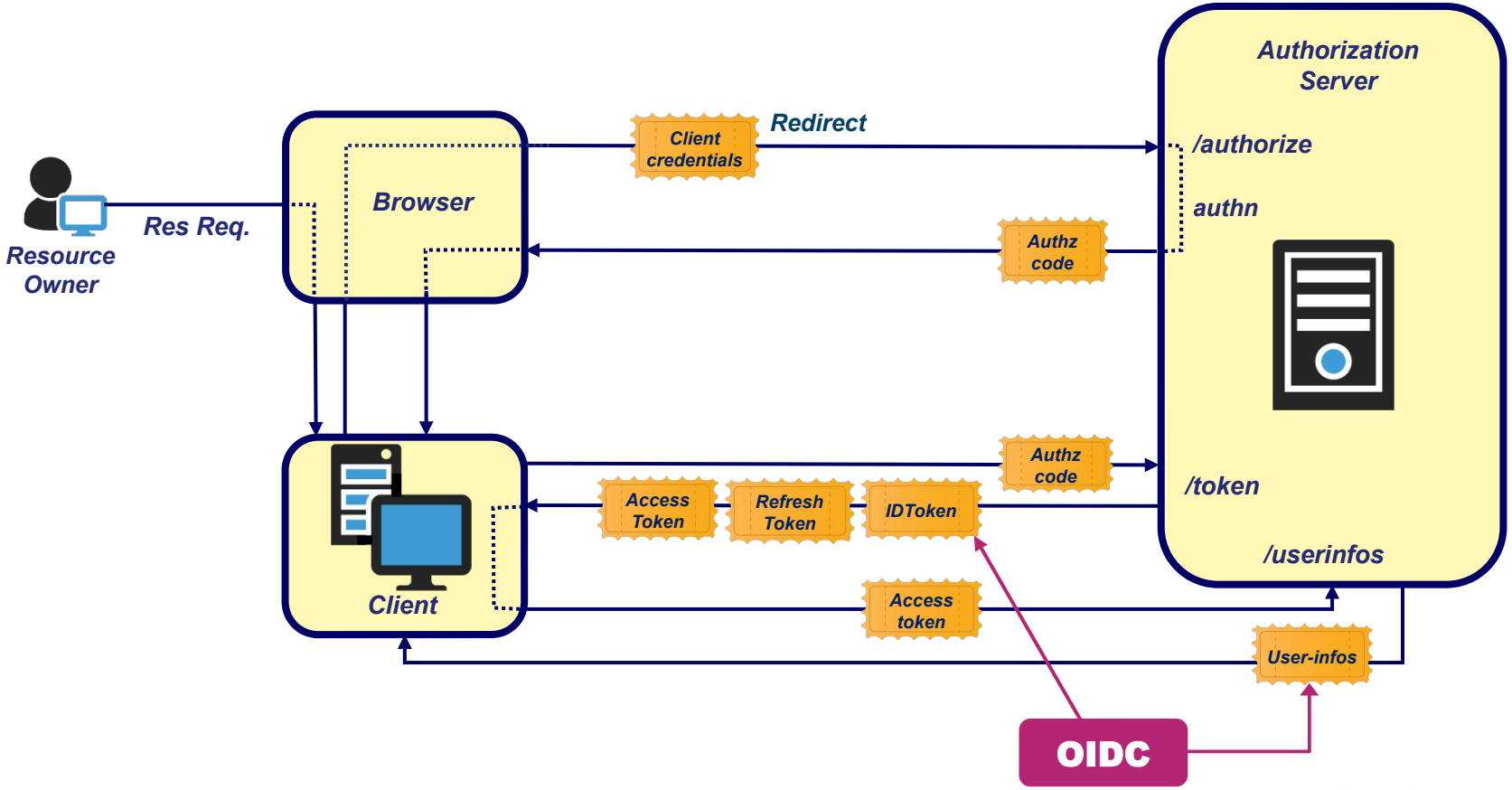
OAuth2: Authorization Code Flow

RFC 6749



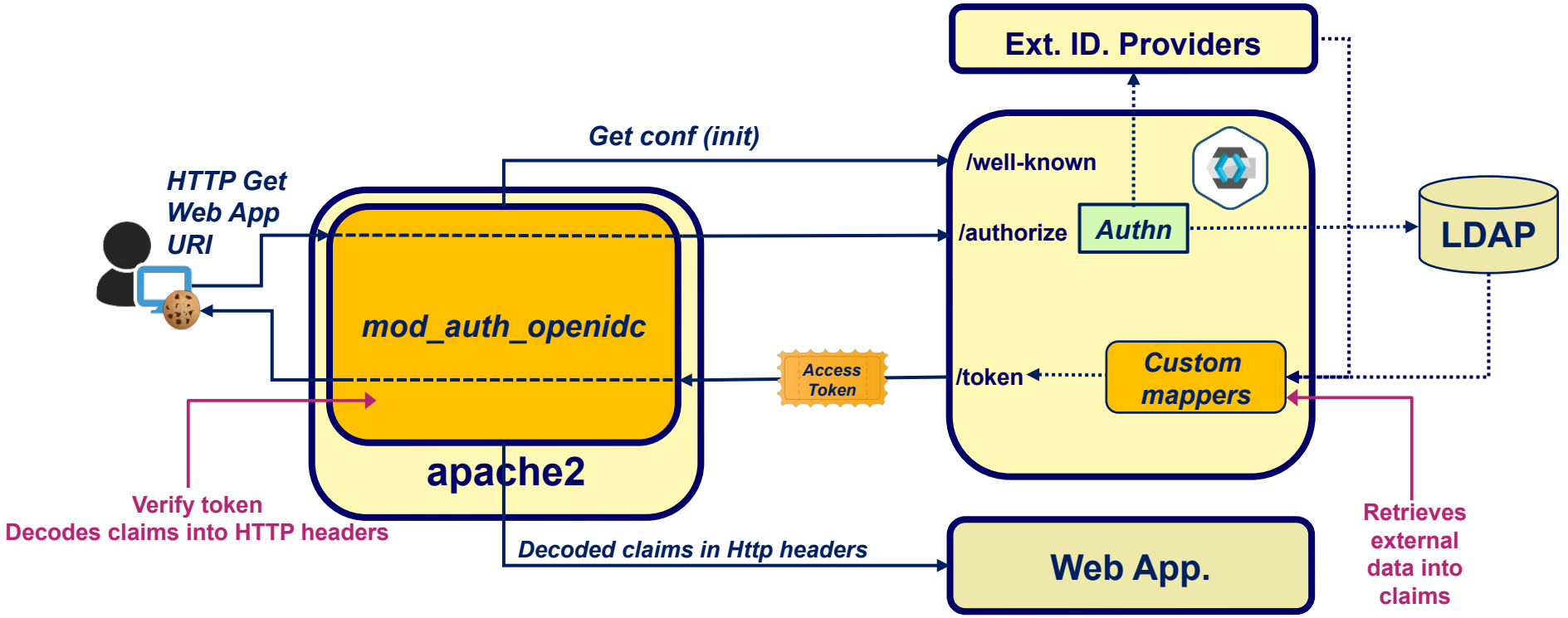

```
https://authorization-server.com/auth?
response_type=code
&client_id=myclient
&client_secret=1234
&redirect_uri=https%3A%2F%2Fexmpl.com%2Fcallback
&scope=mycustomscope,openid
```

OIDC additions



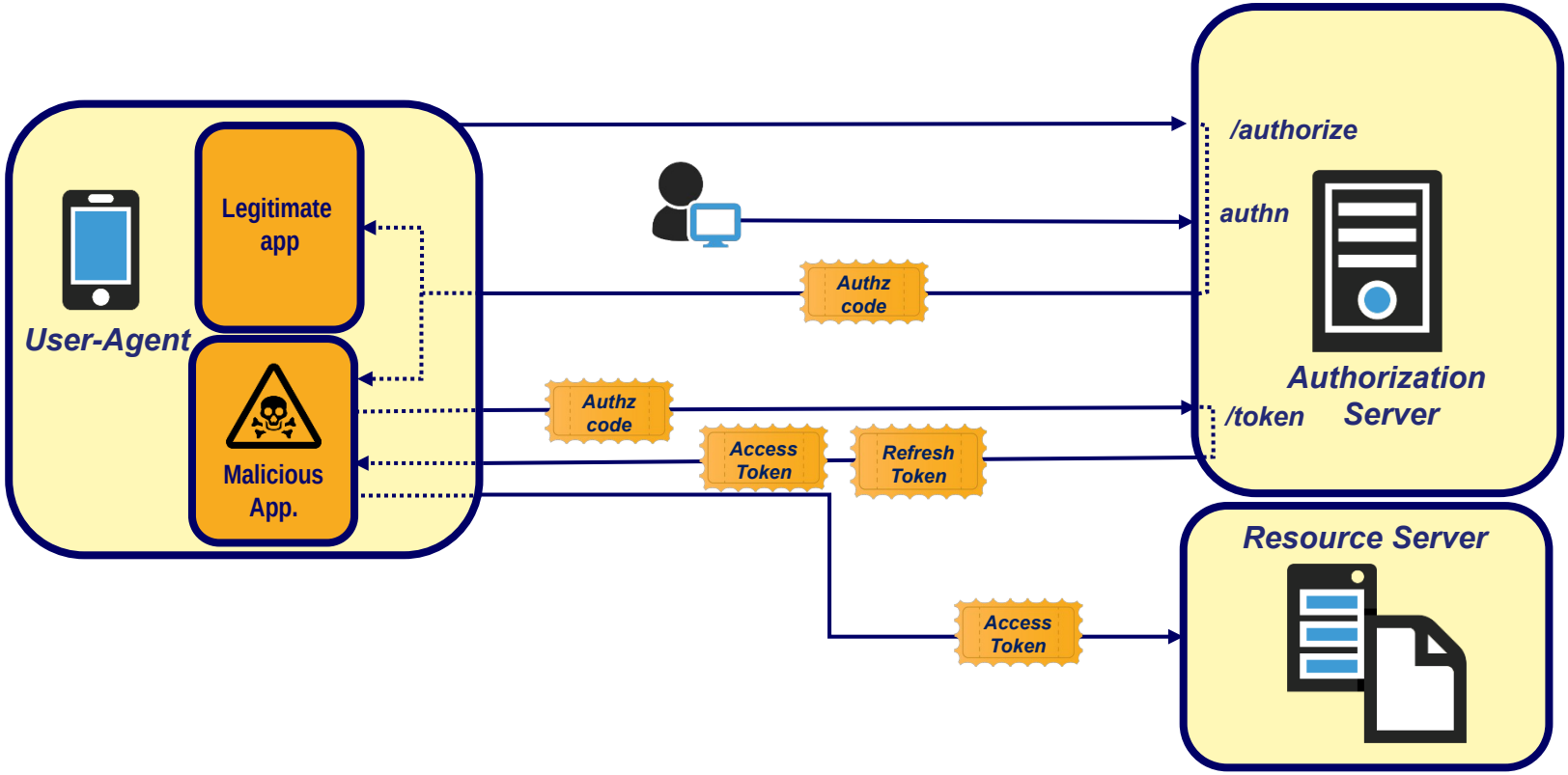
THALES

In practice: web app access



Similar solutions for tomcat, springboot, js, etc.

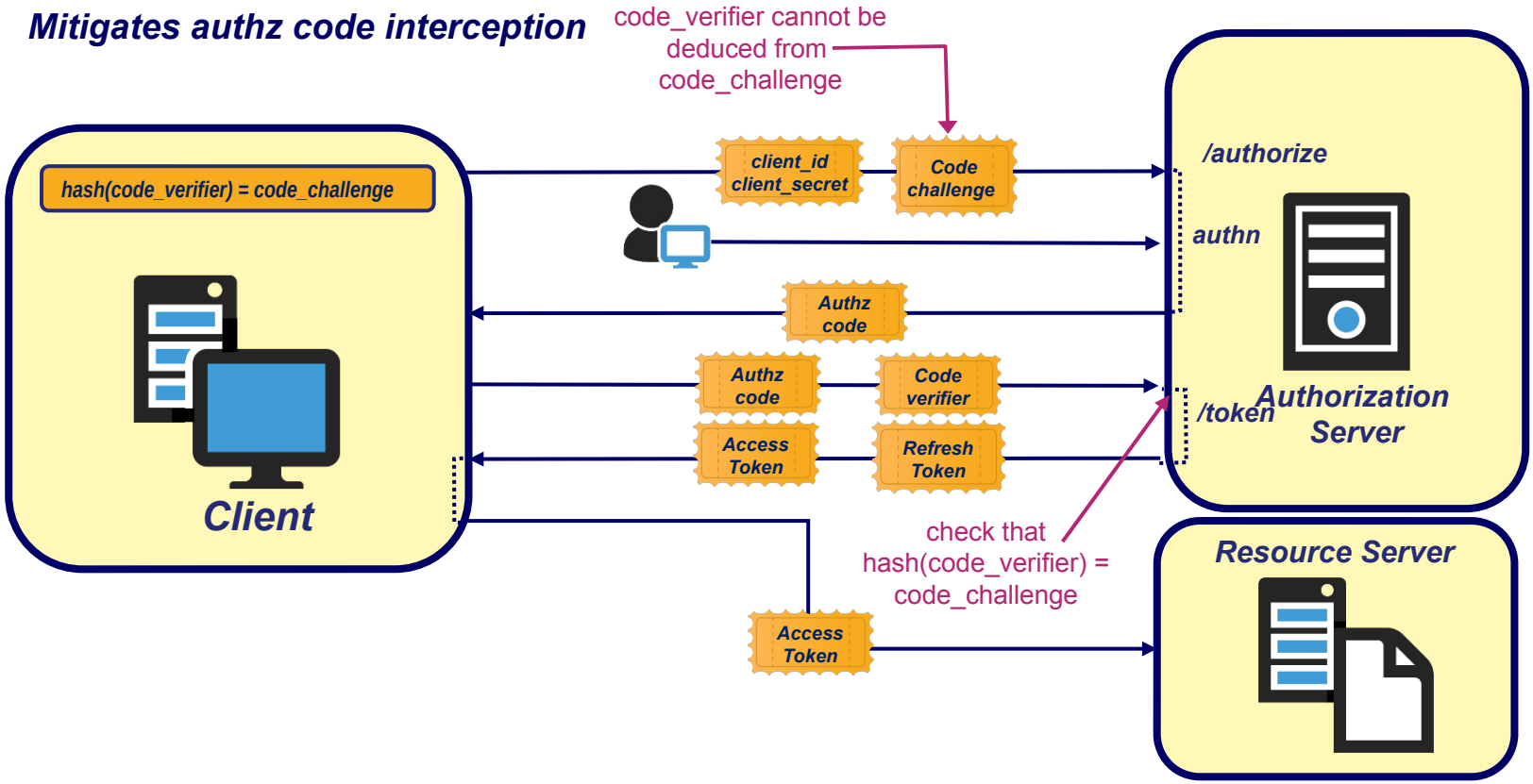
OAuth2: authorization code interception



THALES

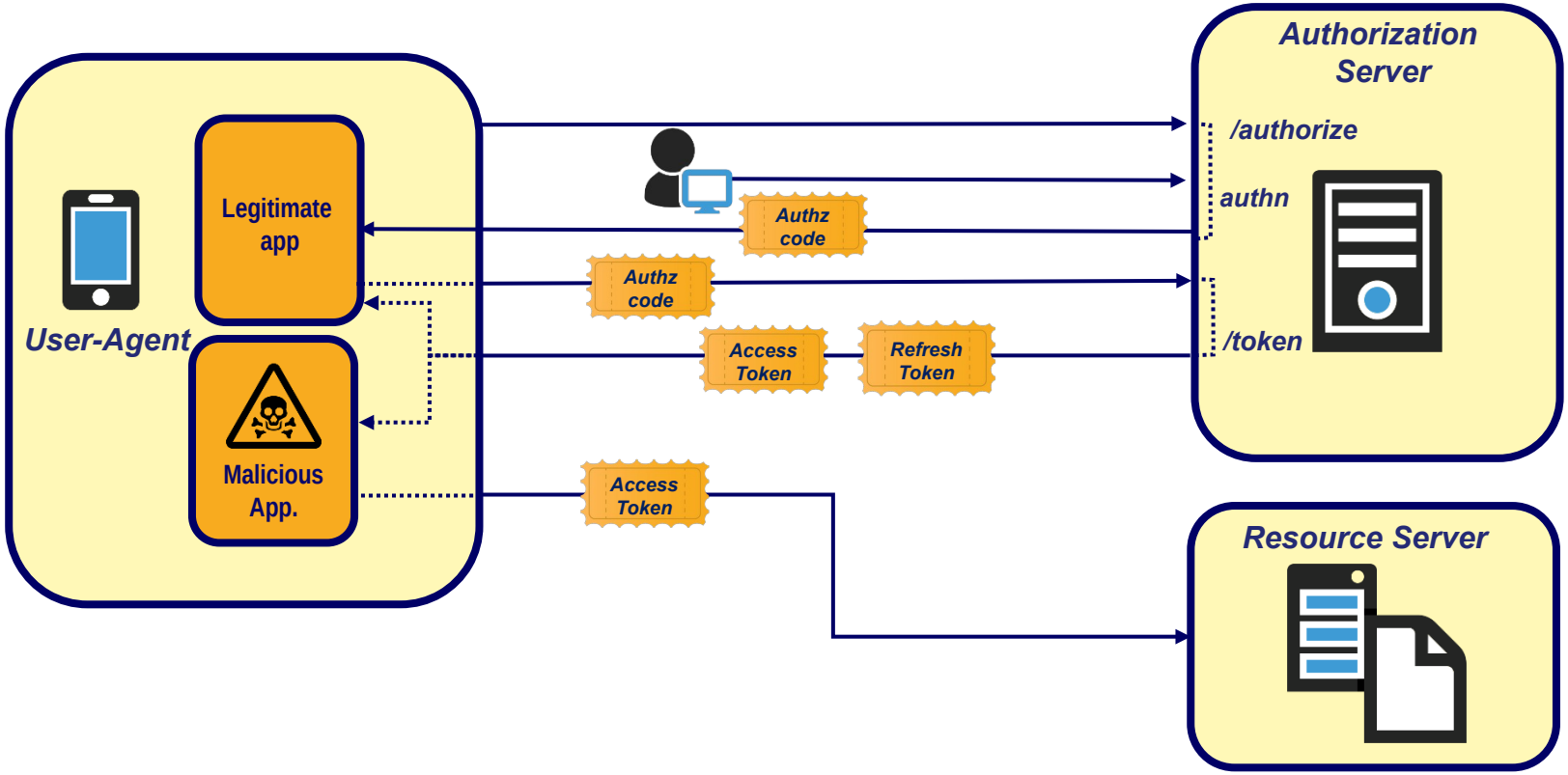
OAuth2: Proof of Key Exchange (PKCE)

RFC 7636



THALES

OAuth2: token interception



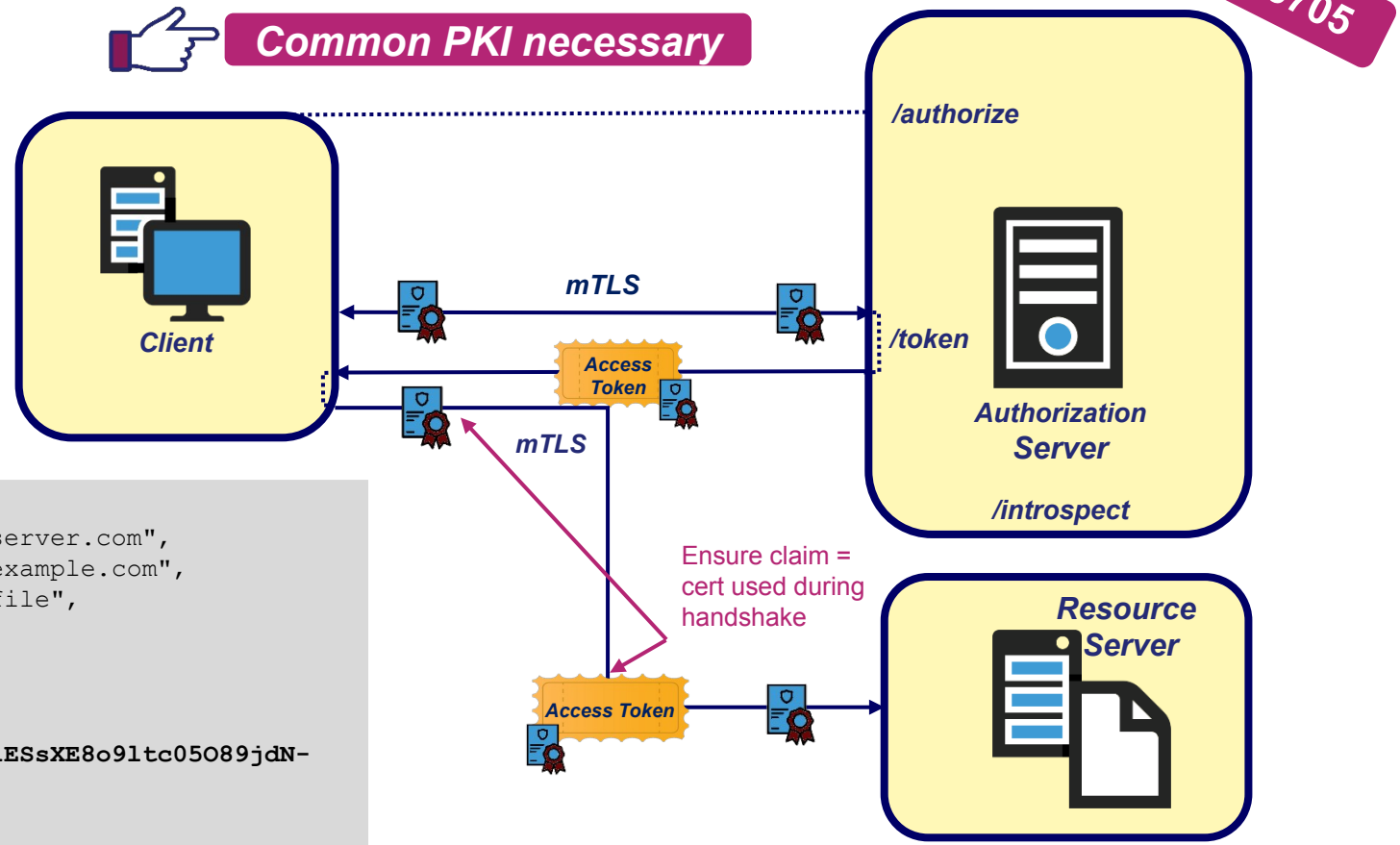
THALES

OAuth2: mTLS flow

RFC 8705

mTLS handshake used to retrieve tokens:

- AS includes hashed client's pubkey in tokens (claim `cnf`)
- AS checks that cert used for TLS and the cnf claim match at each token call




```
{
"iss": "https://authserver.com",
"sub": "random.user@example.com",
"scope": "openid profile",
"exp": 1693699893,
"nbf": 1693699893,
"cnf": {
"x5t#S256":
"bwcK0esc3BDC3DB2Y4_lESsXE8o9ltc05O89jdN-
eb7"
}
}
```


THALES

OAuth2: Demonstrating Proof of Possession

RFC 9449

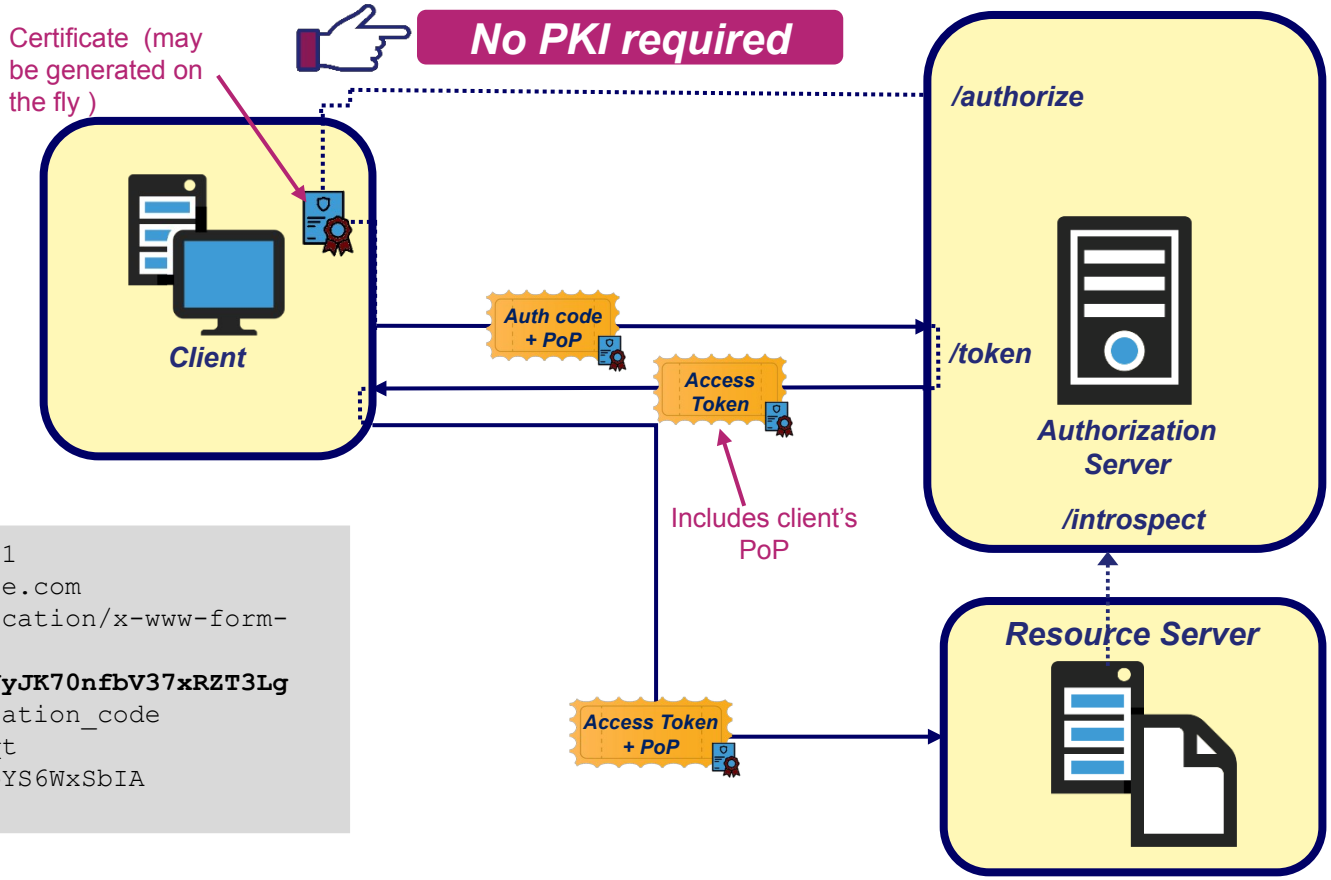
A dedicated key pair is generated (decouple OAuth2 from TLS)

PoP either:

- in dedicated HTTP header as JWT

Or

- In access token `cnf` claim (dpop_jtk)




```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-
urlencoded
DPoP: eyJ0e…4PtFLFNyJK70nfbV37xRZT3Lg
grant_type=authorization_code
&client_id=s6BhdRkqt
&code=SplxlOBeZQQYbYS6WxSbIA
…
```

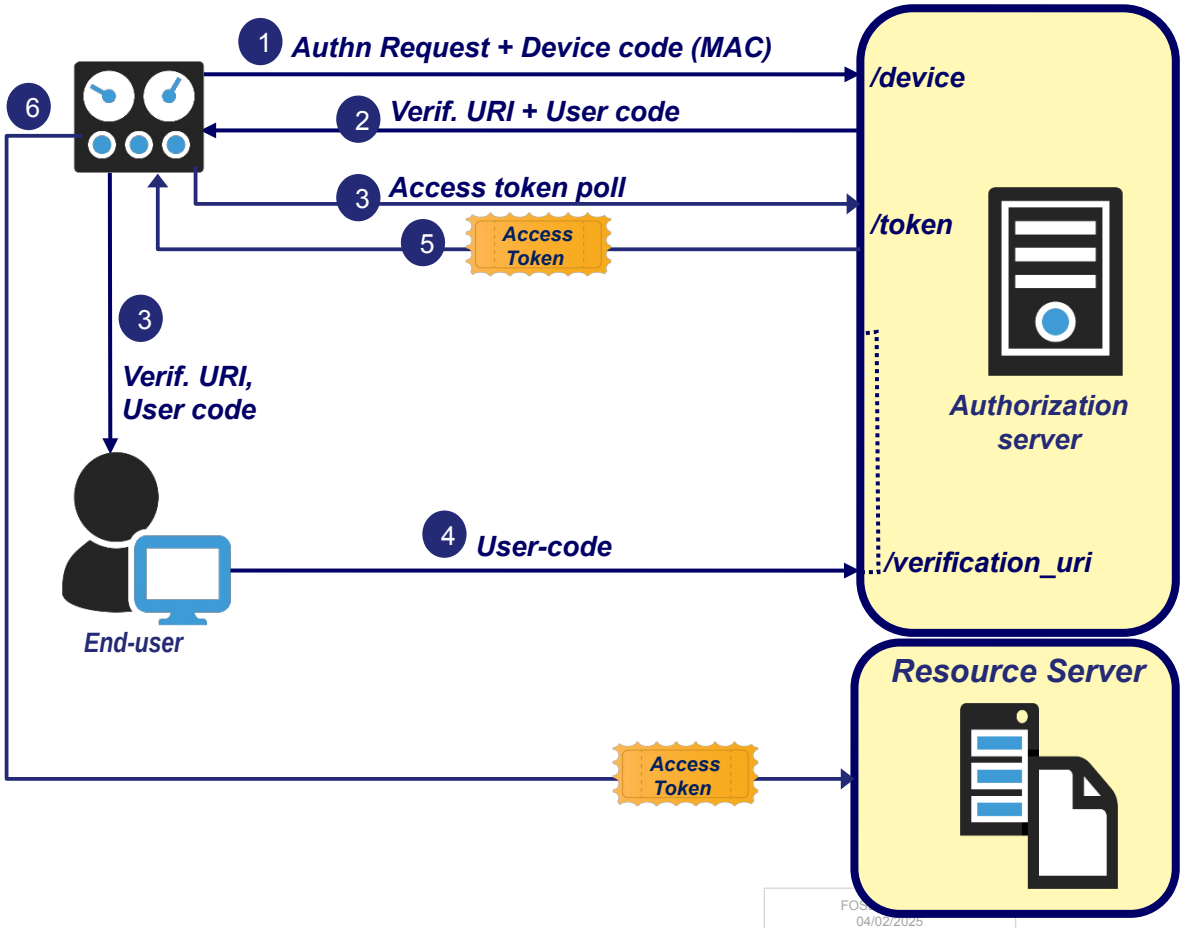

THALES

Agenda

- Introduction
- OAuth2/OIDC concepts
- OAuth2/OIDC flows
- **Some advanced usages**

OAuth2: Device Flow

RFC 8705

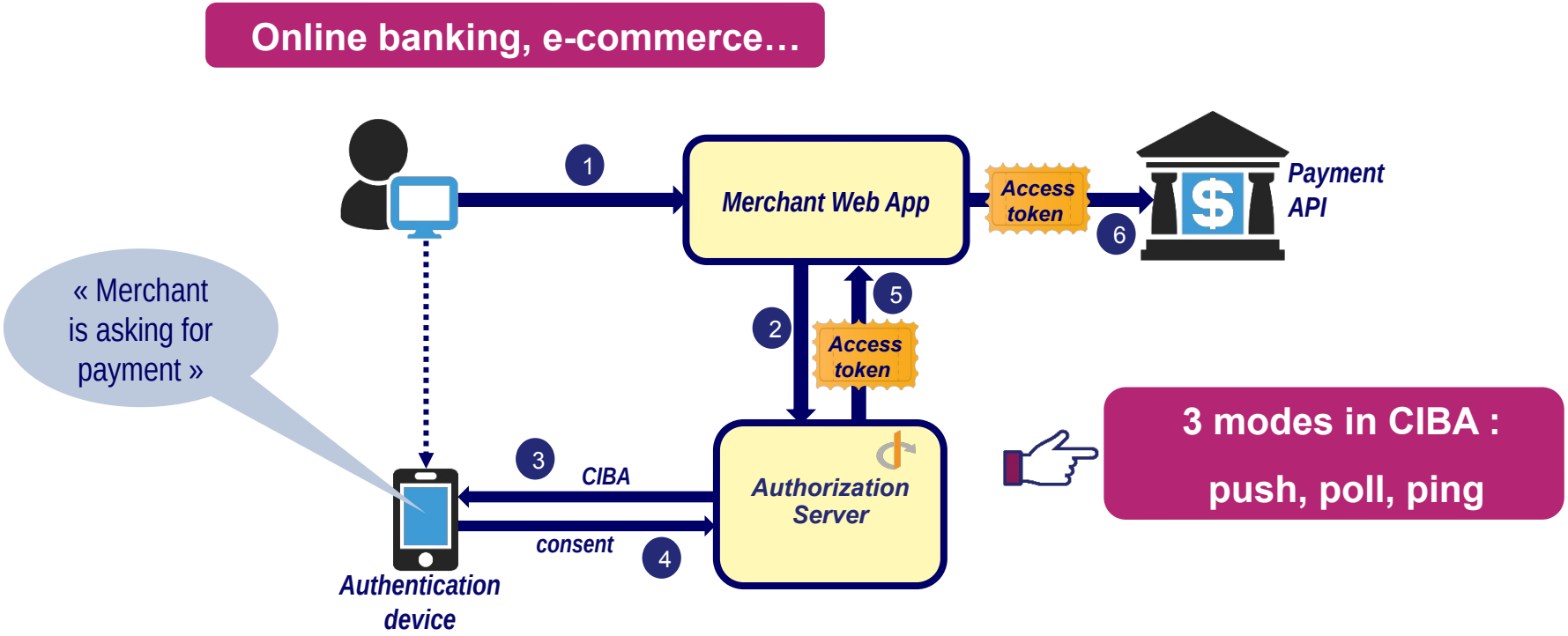


Device : hardware device w/o keyboard (TV set, metering device, etc.).

1. User tries to connect device to RS
2. The device submits its client_id to the OP's device endpoint and gets a user code and validation URL.
3. User code & verif. URI displayed on the device (screen needed); device starts polling for token.
4. End-user enters its code at the verification URL (browser based)
5. Device receives tokens from /token and accesses resource
6. Device requests resource w/token

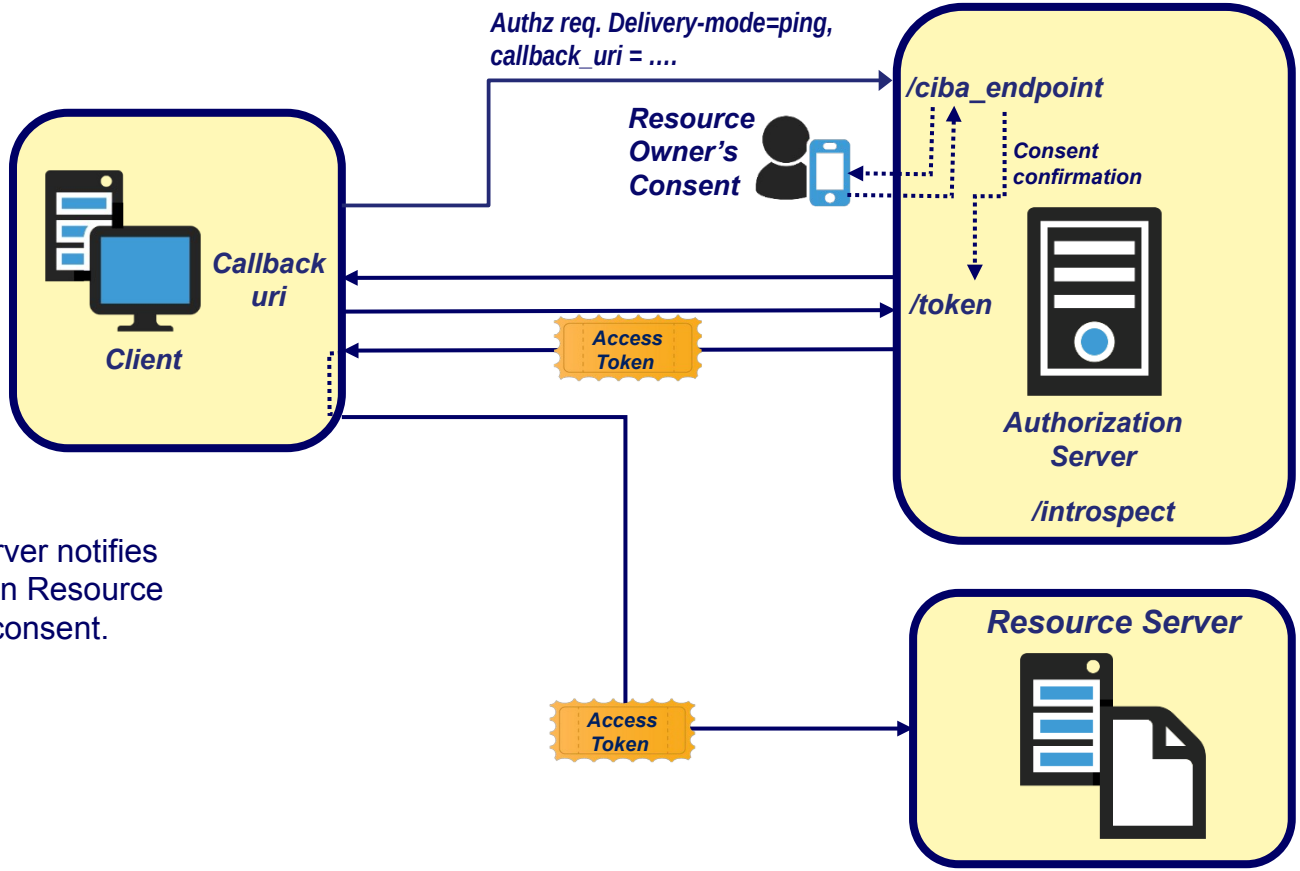
THALES

CIBA: use case



THALES

CIBA: ping mode



Authz Server notifies client upon Resource Owner's consent.

Rich Authorization Requests (RAR)

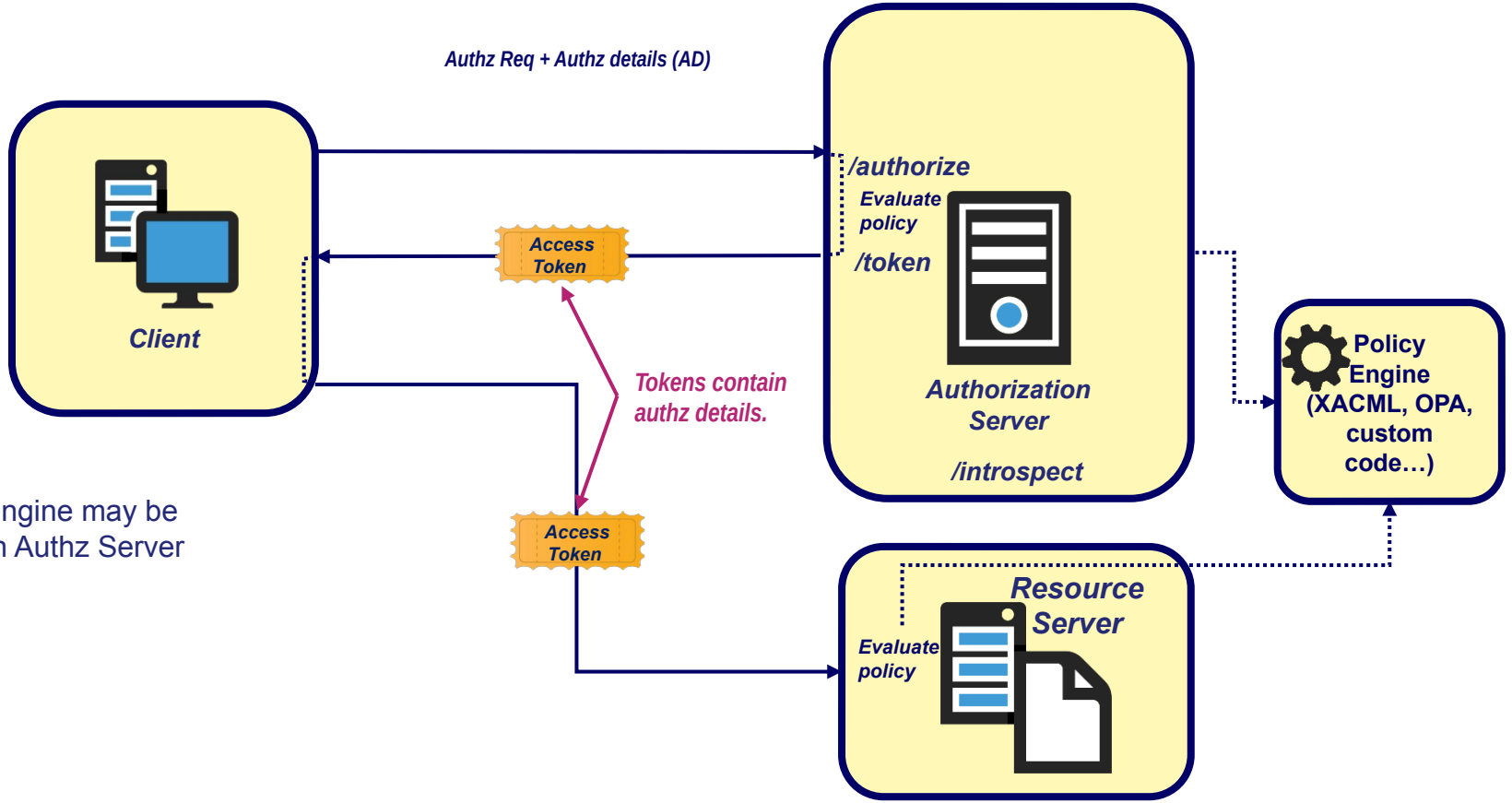
RFC 9396

- “Standard” OAuth2/OIDC access control : based on scopes
  - Communicate only claims included in scope
    - “openid”, “mail”, etc.
  - Rudimentary:
    - Action control ?
    - Timed requests ?
    - RBAC ?
- Solution : add a JSON object in *Authorization Requests*
  - Returned in response
  - Forwarded to RS

```
authorization_details:
{
            "type": "payment_initiation",
            "actions: ["initiate","cancel"],
            "locations": [
            http://example.com/payments
                  ],
            "instructedAmount": {
                "currency": "EUR",
                "amount": "123.50"
            },
                  "creditorName": "Merchant A",
                  "creditorAccount": {
                "iban":"DE02100100109307118603"
            },
"remittanceInformationUnstructured:
                        "Ref Number Merchant«
}
```


THALES

RAR



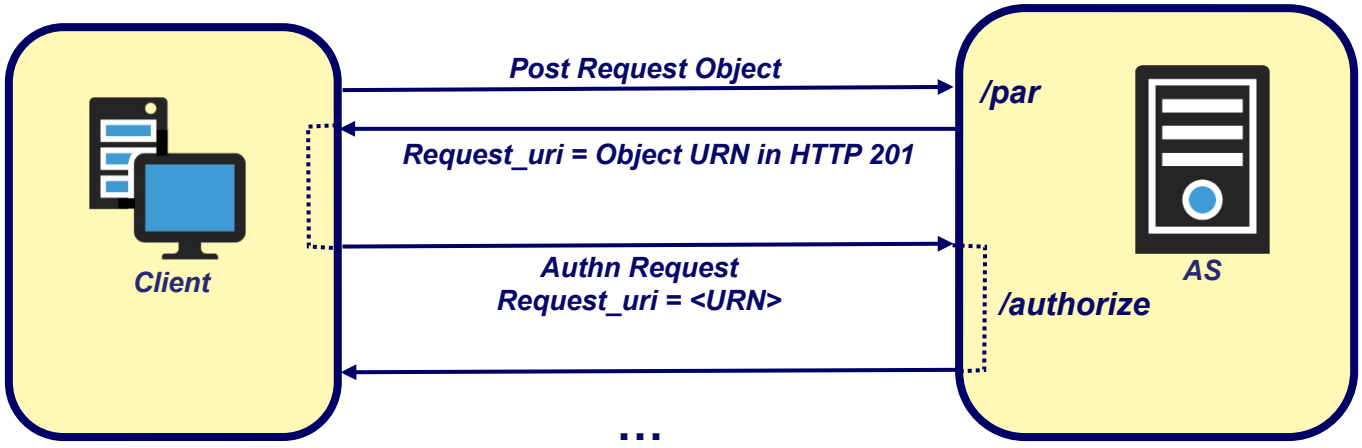
Policy engine may be native in Authz Server

THALES

Pushed Authorization Requests (PAR)

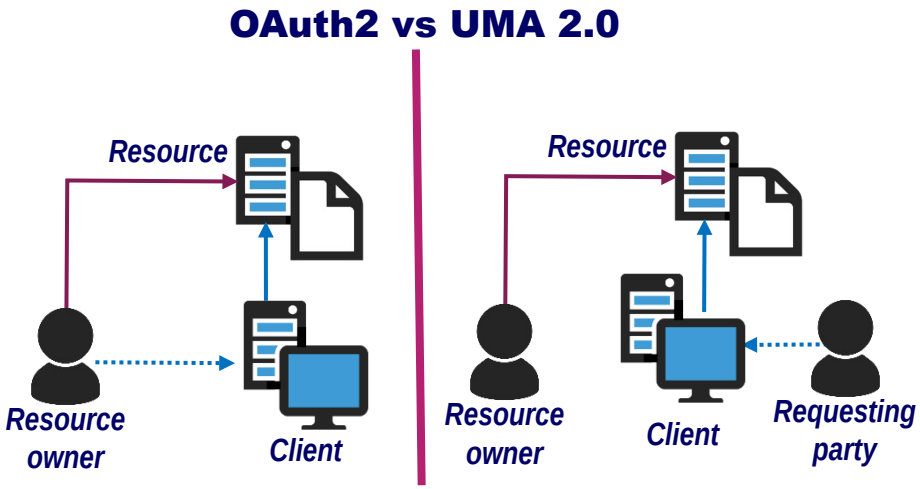
RFC 9126

- **Issue with « standard » authz requests (URI params)**
- No crypto integrity protection
- Requests may be too long for browsers
- No confidentiality (encryption) possible
- **=> Solution : create a dedicated JSON Request object**



THALES

User-Managed Access (UMA)



UMA

- Built on top of OAuth2
- Additional entity : requesting party (RqP)
- Access delegation: ie. makes resources shareable
- Allows asynchronous management of resources access

Use case

User1 wants User2 to access his bank account balance, but he but doesn't want her to see his transaction history.

UMA : additional objects

Tokens

- **Access API token** (AAT): UMA specific access token used to retrieve RqP permissions
- **Permission Ticket** (PT): token representing permissions of a given RqP
- **Requesting Party Token** (RPT): resource request token

Endpoints

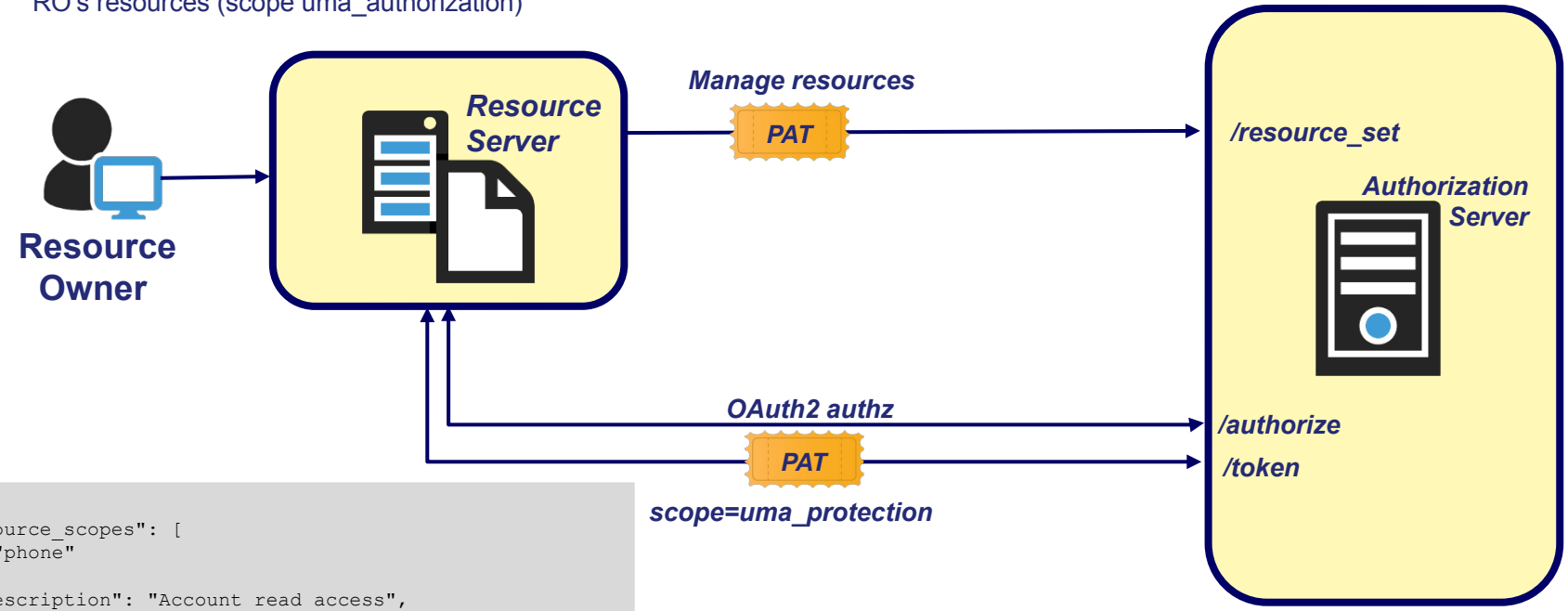
- **/.well-known/uma2-configuration** : configuration discovery
- **/permission** : used by RS to retrieve PTs using PATs
- **/resource_registration** : used by RS to create resources on RO's behalf using PATs

Params

- dedicated scope: **uma_protection**
- dedicated grant_type: urn:ietf:params:oauth:grant-type:**uma-ticket**

UMA : Resource Owner's workflow

- **Protection API token** (PAT): service token specifically delivered to RS to manage RO's resources (scope uma_authorization)

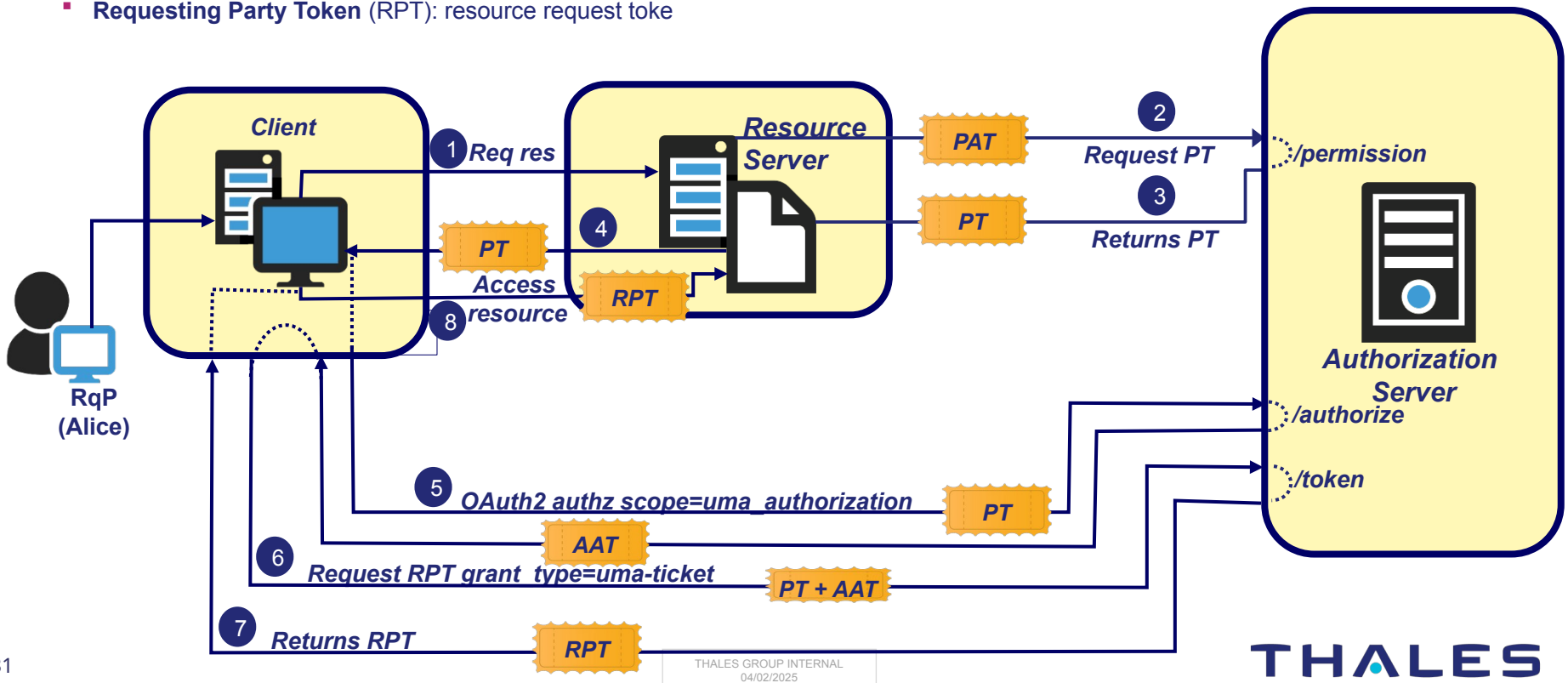



```
{
"resource_scopes": [
    "phone"
  ],
  "description": "Account read access",
  "iconUri": "http://www.example.com/icons/picture.png",
  "name": " Bank Account access",
  "type": "http://www.example.com/resource/bank-
account",
  "user_access_policy_uri": https://myauthserver.fr/{dom
ain}/uma/protection/resource_set/{resource_id}/policies
}
```


THALES

UMA : Authorization workflow

- **Access API token** (AAT): UMA specific access token used to retrieve RqP permissions
- **Permission Ticket** (PT): token representing permissions of a given RqP
- **Requesting Party Token** (RPT): resource request toke



THALES

Any questions ?