OAuth 2.1 & OpenID Connect in Action

What’s New, What’s Secure, and What You Need To Know

JAVAFORUM stuttgart 2025

Stuttgart – 10.7.2025

CGI

About Me

Andreas Falk

andreas.falk@cgi.com

LinkedIn

https://www.linkedin.com/in/andifalk

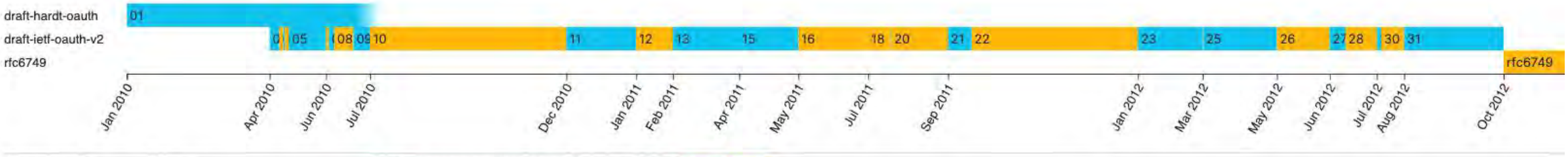
OAuth 2 and OpenID Connect have been around for Ages!

The OAuth 2.0 Authorization Framework

RFC 6749

So Why a Presentation on Them?

Status | Email expansions | History



Document	Type	
		RFC - Proposed Standard (October 2012) Errata IPR

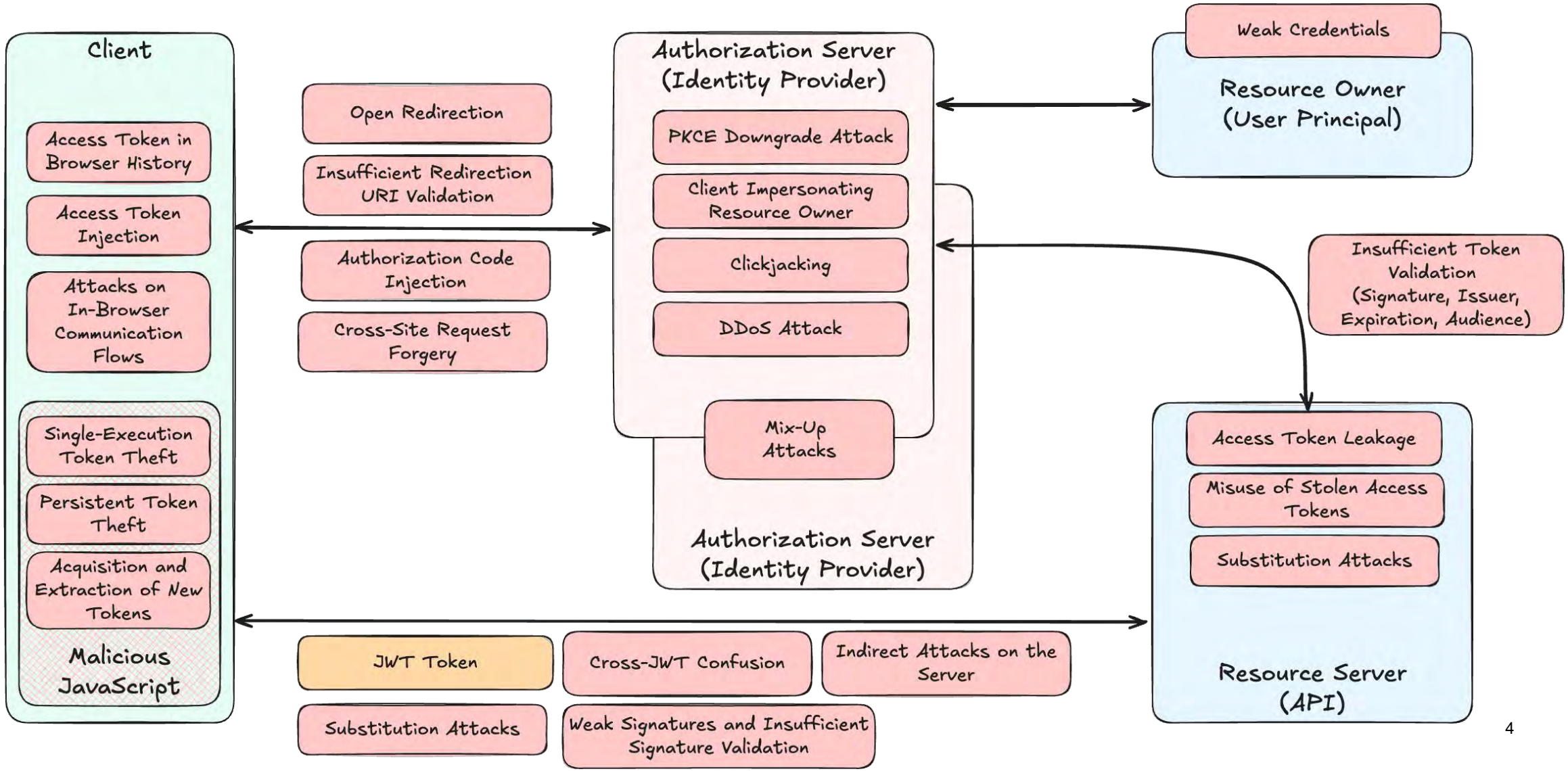
The OpenID Foundation Launches the OpenID Connect Standard

Published February 26, 2014

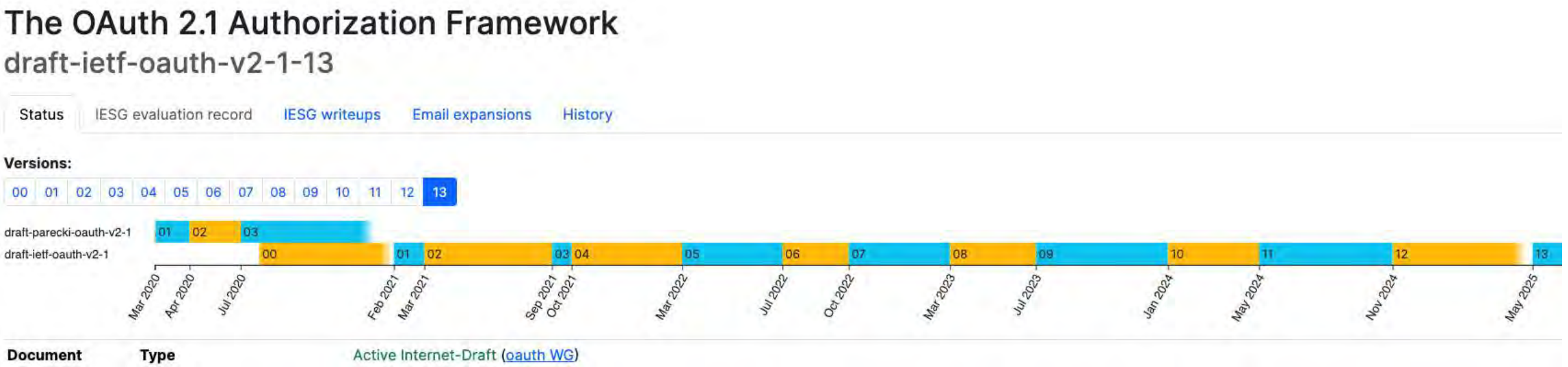
Providing Increased Security, Usability, and Privacy on the Internet

RSA 2014 and Mobile World Congress- San Francisco, CA, and Barcelona, Spain – Feb. 26, 2014 – The OpenID Foundation announced today that its membership has ratified the OpenID Connect standard. Organizations and businesses can now use OpenID Connect to develop secure, flexible, and interoperable identity Internet ecosystems so that digital identities can be easily used across websites and applications via any computing or mobile device. OpenID Connect has been implemented worldwide by Internet and mobile companies, including

Known OAuth 2 / OIDC / JWT Attacks – Constantly Evolving!



Known OAuth 2 / OIDC / JWT Attacks – Mitigations are on the way



https://medium.com/oauth-2/why-you-should-stop-using-the-oauth-implicit-grant (2018)
The OAuth 2.1 Authorization Framework (Draft)
Best Current Practice for OAuth 2.0 Security (RFC 9700)
OAuth 2.0 for Browser-Based Applications (Draft)

+ a whole lot more Specifications 

OAuth 2.1 / OIDC 1.0 in OWASP ASVS 5.0

Application Security Verification Standard (ASVS) V5.0 released at OWASP AppSec EU 2025 in Barcelona

https://github.com/OWASP/ASVS/releases/tag/v5.0.0_release

V10.1 Generic OAuth and OIDC Security

This section covers generic architectural requirements that apply to all applications using OAuth or OI

#	Description	Level
10.1.1	Verify that tokens are only sent to components that strictly need them. For example, when using a backend-for-frontend pattern for browser-based JavaScript applications, access and refresh tokens shall only be accessible for the backend.	2
10.1.2	Verify that the client only accepts values from the authorization server (such as the authorization code or ID Token) if these values result from an authorization flow that was initiated by the same user agent session and transaction. This requires that client-generated secrets, such as the proof key for code exchange (PKCE) 'code_verifier', 'state' or OIDC 'nonce', are not guessable, are specific to the transaction, and are securely bound to both the client and the user agent session in which the transaction was started.	2

V10.3 OAuth Resource Server

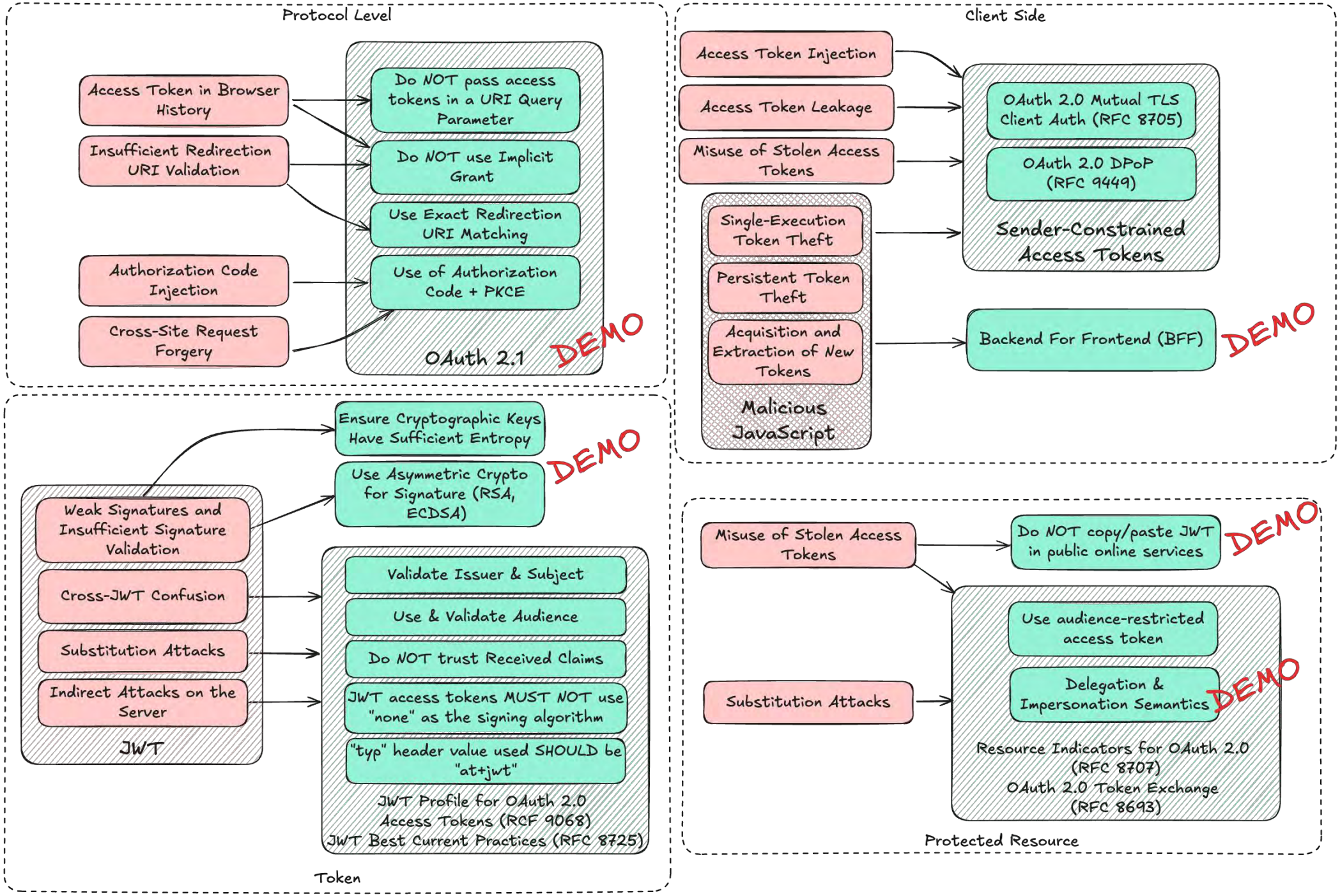
In the context of ASVS and this chapter, the resource server is an API. To provide secure access, the resource server must:

- Validate the access token, according to the token format and relevant protocol specifications, e.g., JWT-validation or OAuth token introspection.
- If valid, enforce authorization decisions based on the information from the access token and permissions which have been granted. For example, the resource server needs to verify that the client (acting on behalf of RO) is authorized to access the requested resource.

Therefore, the requirements listed here are OAuth or OIDC specific and should be performed after token validation and before performing authorization based on information from the token.

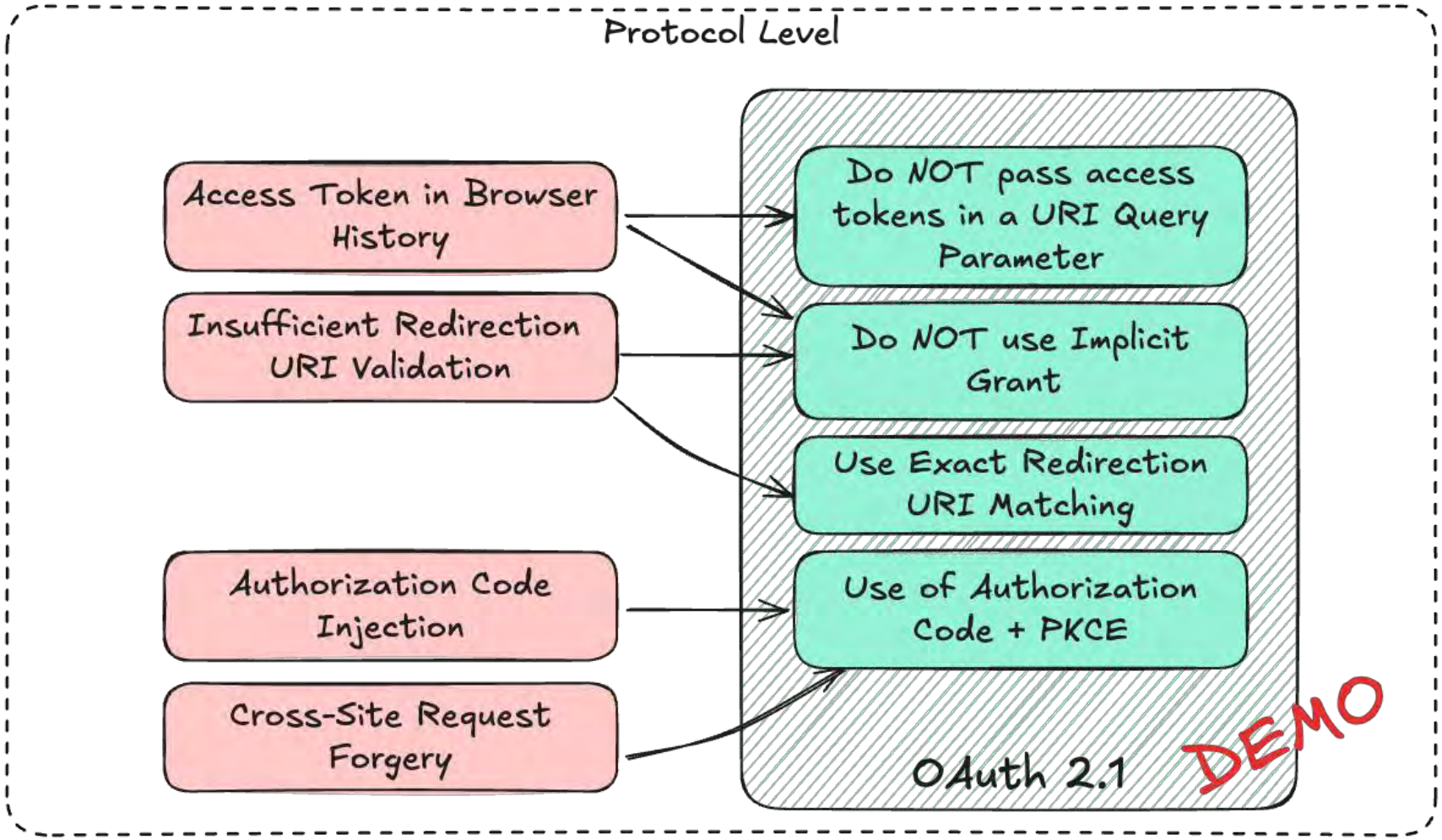
#	Description	Level
10.3.1	Verify that the resource server only accepts access tokens that are intended for use with that service (audience). The audience may be included in a structured access token (such as the 'aud' claim in JWT), or it can be checked using the token introspection endpoint.	2
10.3.2	Verify that the resource server enforces authorization decisions based on claims from the access token that define delegated authorization. If claims such as 'sub', 'scope', and 'authorization_details' are present, they must be part of the decision.	2
10.3.3	Verify that if an access control decision requires identifying a unique user from an access token (JWT or related token introspection response), the resource server identifies the user from claims that cannot be reassigned to other users. Typically, it means using a combination of 'iss' and 'sub' claims.	2
10.3.4	Verify that, if the resource server requires specific authentication strength, methods, or recentness, it verifies that the presented access token satisfies these constraints. For example, if present, using the OIDC 'acr', 'amr' and 'auth_time' claims respectively.	2
10.3.5	Verify that the resource server prevents the use of stolen access tokens or replay of access tokens (from unauthorized parties) by requiring sender-constrained access tokens, either Mutual TLS for OAuth 2 or OAuth 2 Demonstration of Proof of Possession (DPoP).	3

Hitchikers Guide to the Talk

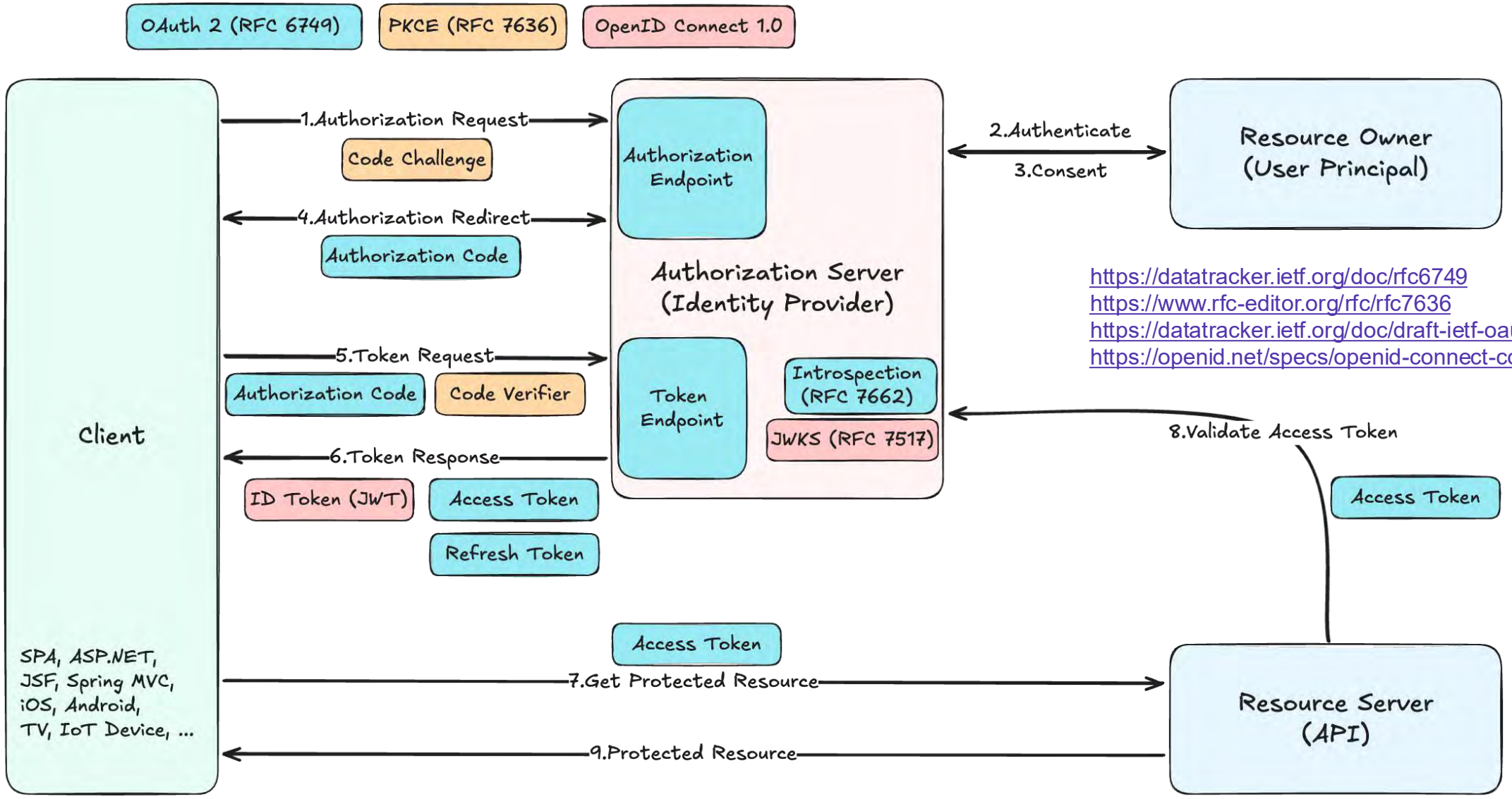


Part 1

Protocol Layer



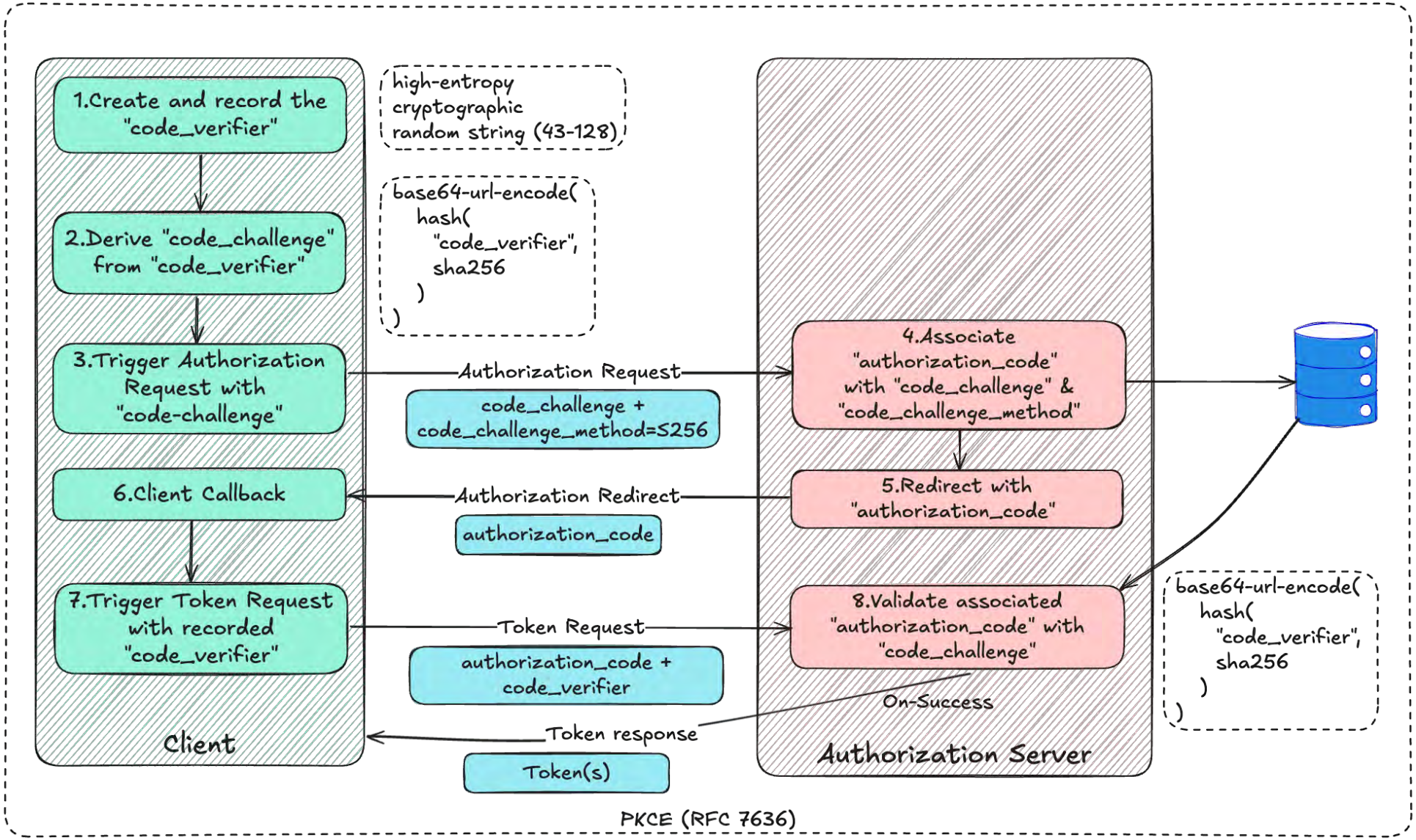
OAuth 2.1 & OpenID Connect 1x1 – Authorization Code + PKCE



https://datatracker.ietf.org/doc/rfc6749
https://www.rfc-editor.org/rfc/rfc7636
https://datatracker.ietf.org/doc/draft-ietf-oauth-v2-1
https://openid.net/specs/openid-connect-core-1_0.html

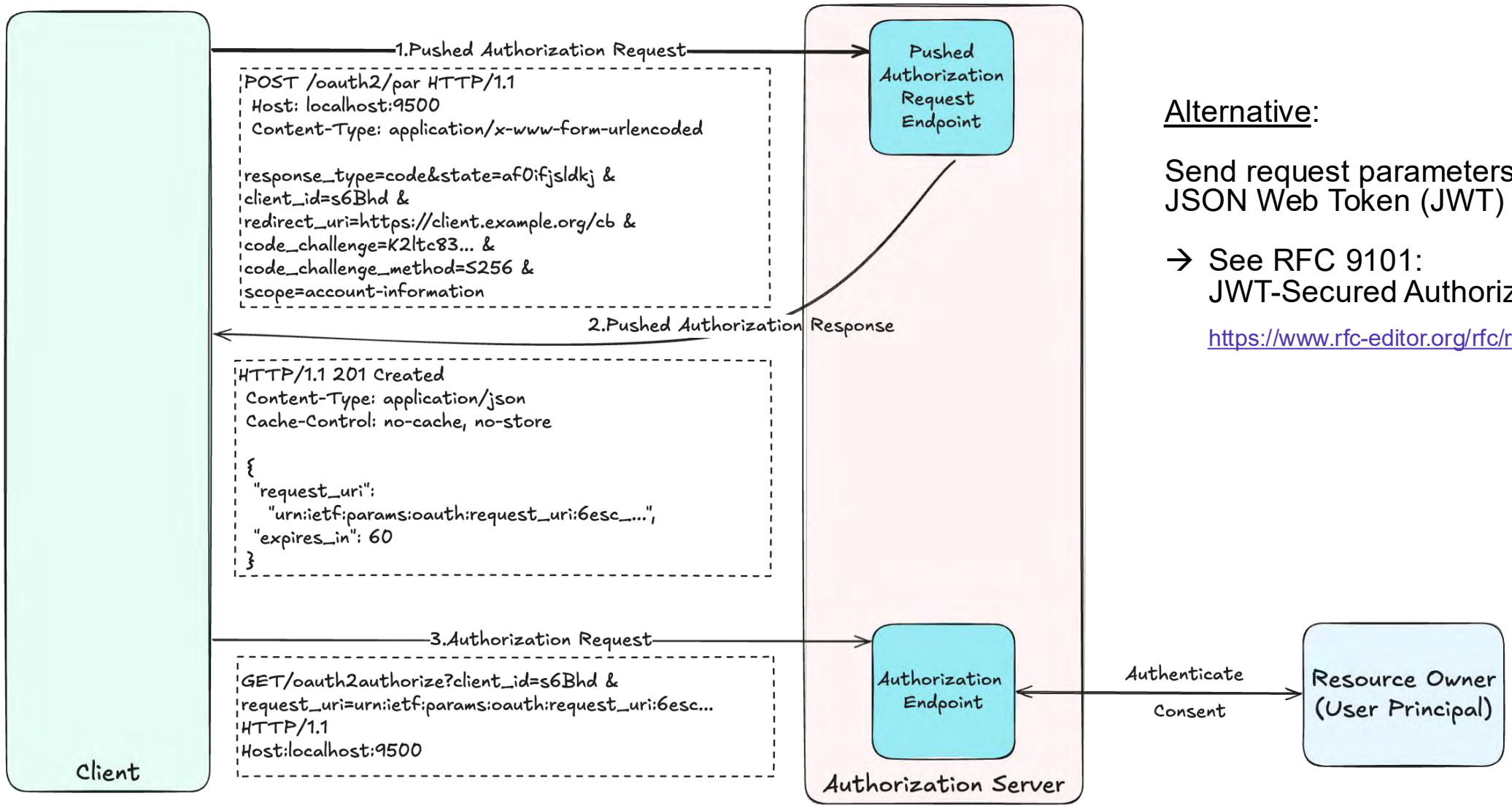
Proof Key for Code Exchange (PKCE)

https://www.rfc-editor.org/rfc/rfc7636



Dynamic secrets replacing static "*client_secret*" for token request

OAuth 2.0 Pushed Authorization Requests (PAR) - RFC 9126



Alternative:

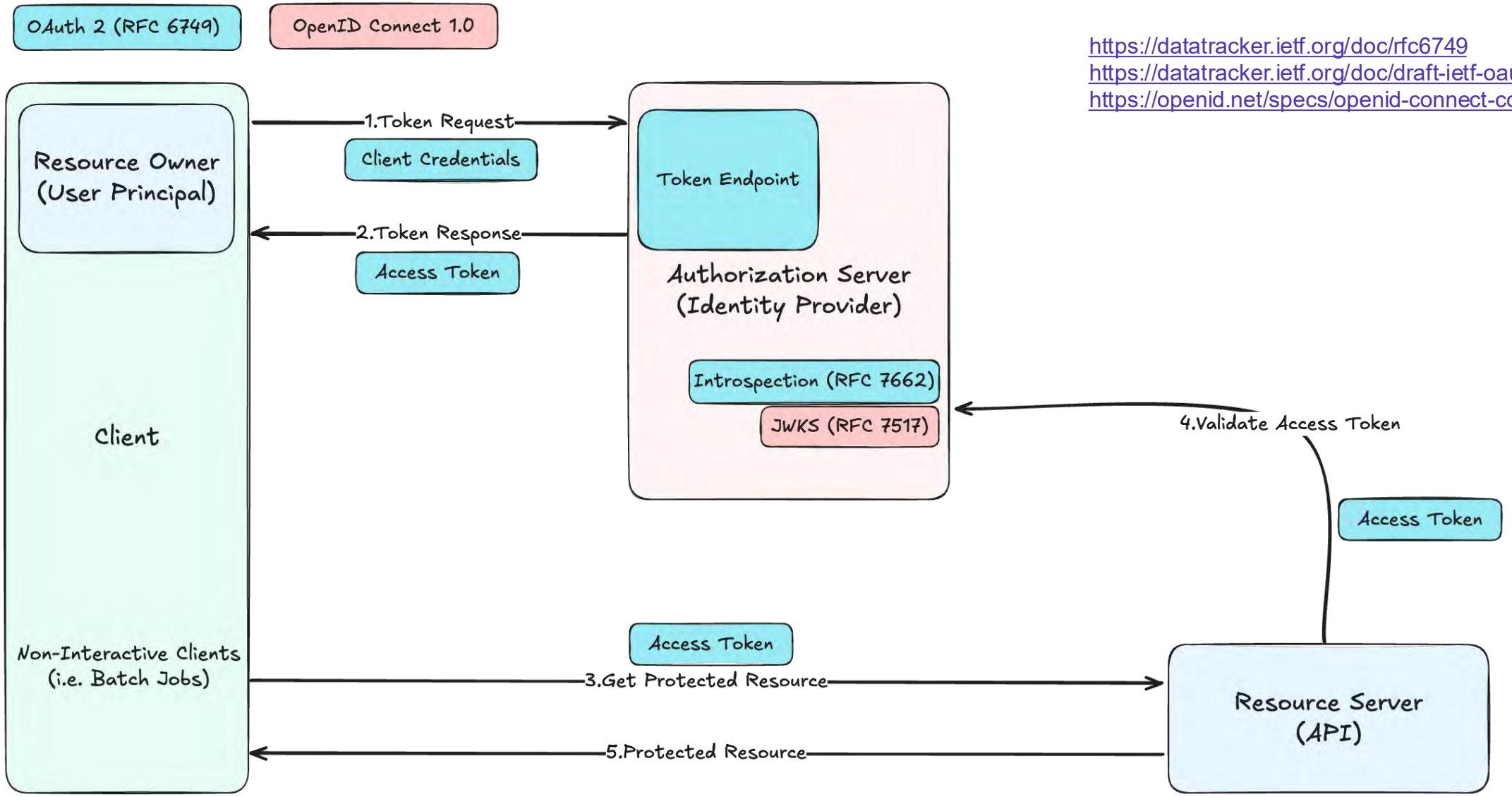
Send request parameters in a JSON Web Token (JWT)

→ See RFC 9101: JWT-Secured Authorization Request

https://www.rfc-editor.org/rfc/rfc9101.html

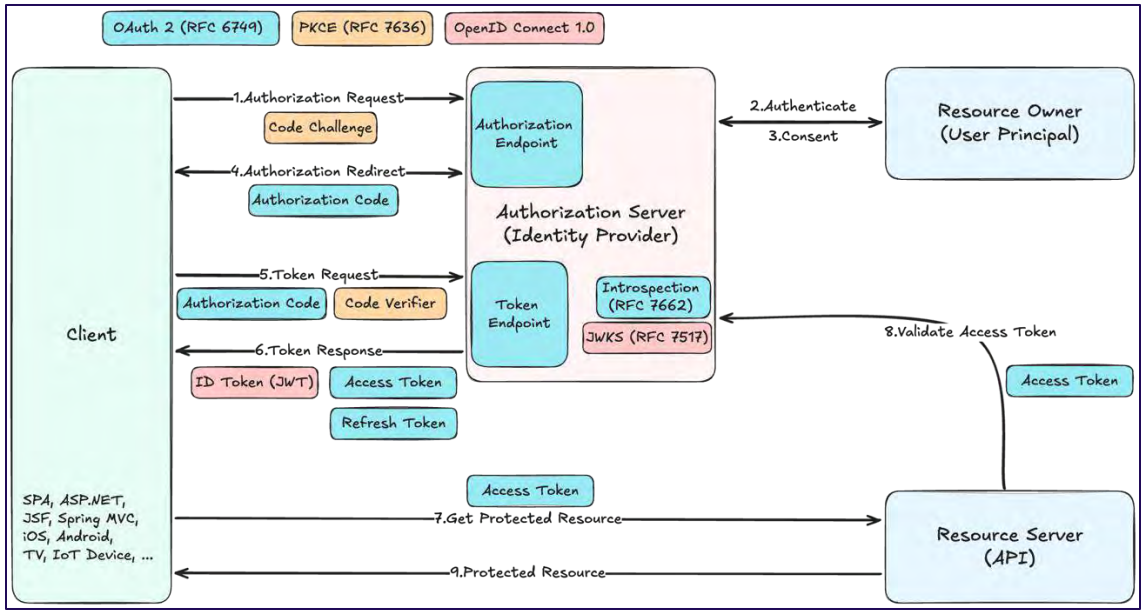
https://www.rfc-editor.org/rfc/rfc9126.html

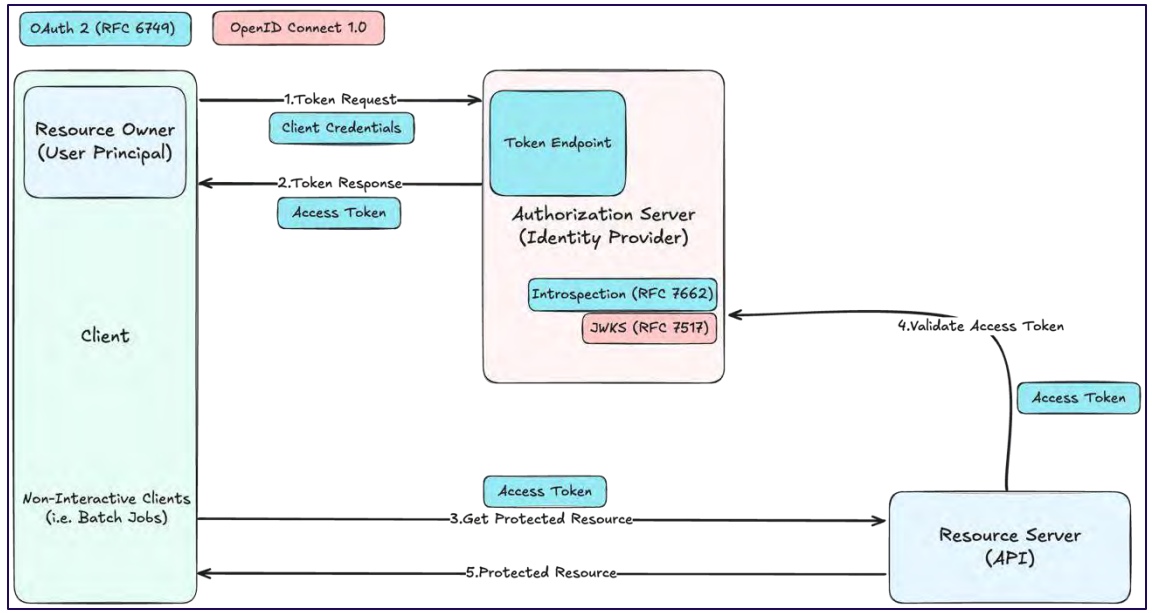
OAuth 2.1 & OpenID Connect 1x1 – Client Credentials



https://datatracker.ietf.org/doc/rfc6749
https://datatracker.ietf.org/doc/draft-ietf-oauth-v2-1
https://openid.net/specs/openid-connect-core-1_0.html

OAuth 2.1 & OpenID Connect 1x1 – Demo

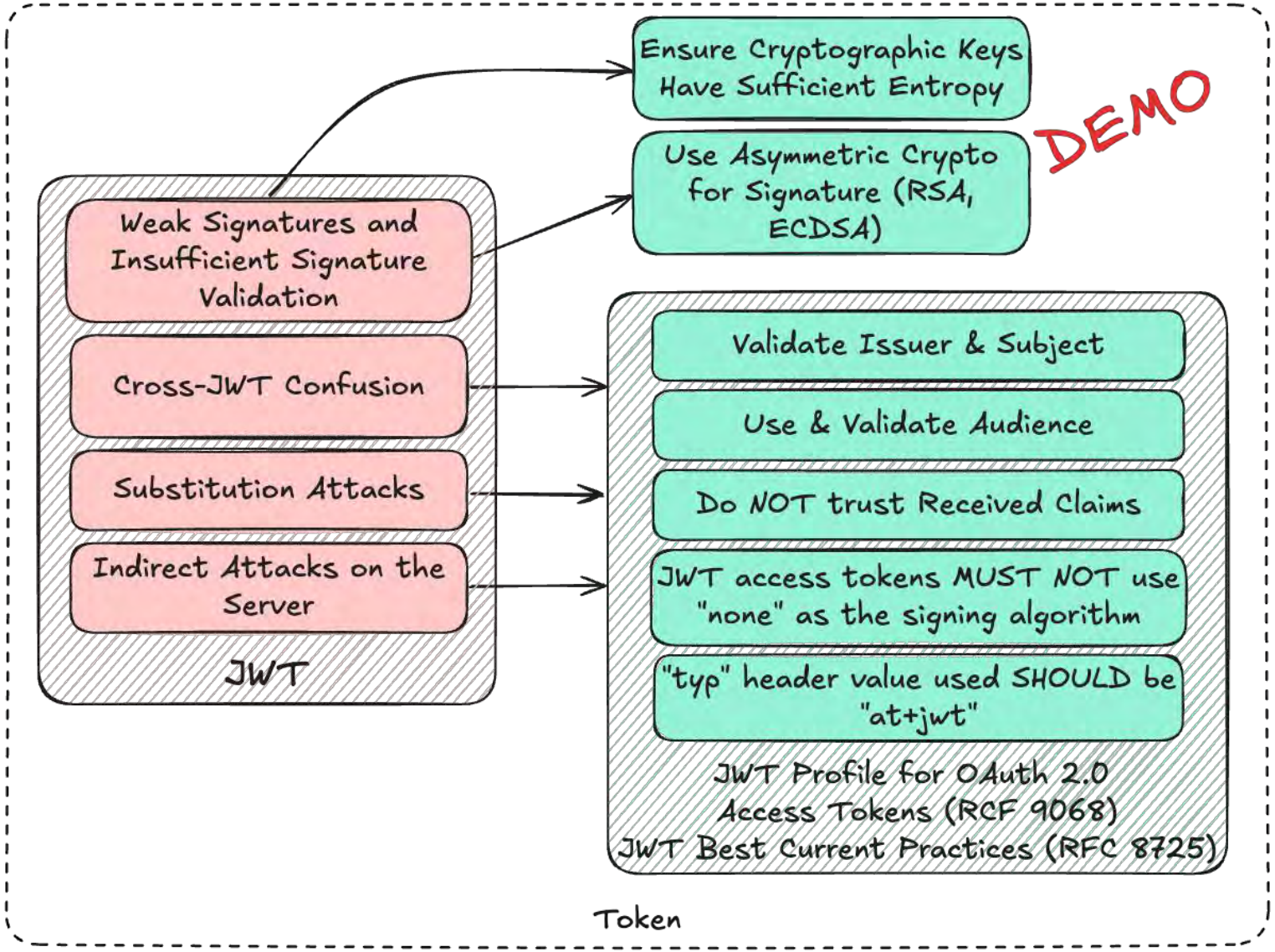




https://github.com/andifalk/federated-identity-demos

Part 2

Tokens



JWT Brute Forcing - Demo

https://github.com/ticarpi/jwt_tool

https://github.com/andifalk/federated-identity-demos →

JWT Profile for OAuth 2.0 Access Tokens - RFC 9068


```
{"typ":"at+JWT","alg":"RS256","kid":"RjEwOwOA"}
.
{
 "iss": "https://authorization-server.example.com/",
 "sub": "5ba552d67",
 "aud":   "https://rs.example.com/",
 "exp": 1639528912,
 "iat": 1618354090,
 "jti" : "dbe39bf3a3ba4238a513f51d6e1691c4",
 "client_id": "s6BhdRkqt3",
 "scope": "openid profile reademail",
 "roles": [],
 "groups": [
  {
    "value": "fc348aa8-3835-40eb-a20b-c726e15c55b5",
    "$ref": "https://example.com/v2/Groups/fc348aa8-3835-40eb-a20b-c726e15c55b5",
    "display": "Employees"
  }
 ],
 "entitlements": []
}
```


Indicates that the token is an Access Token

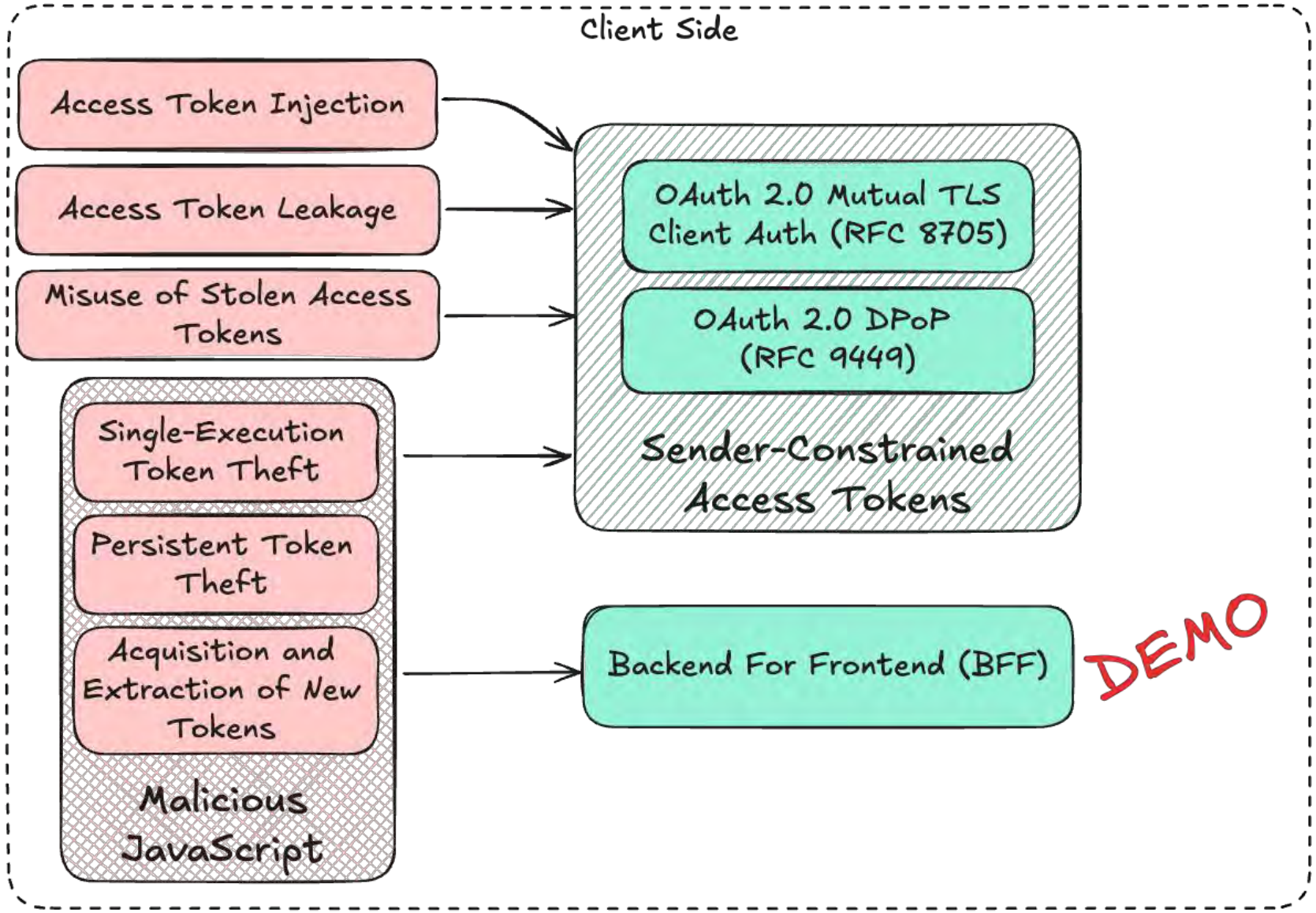
Must NOT use “none”

System for Cross-domain Identity Management (SCIM)

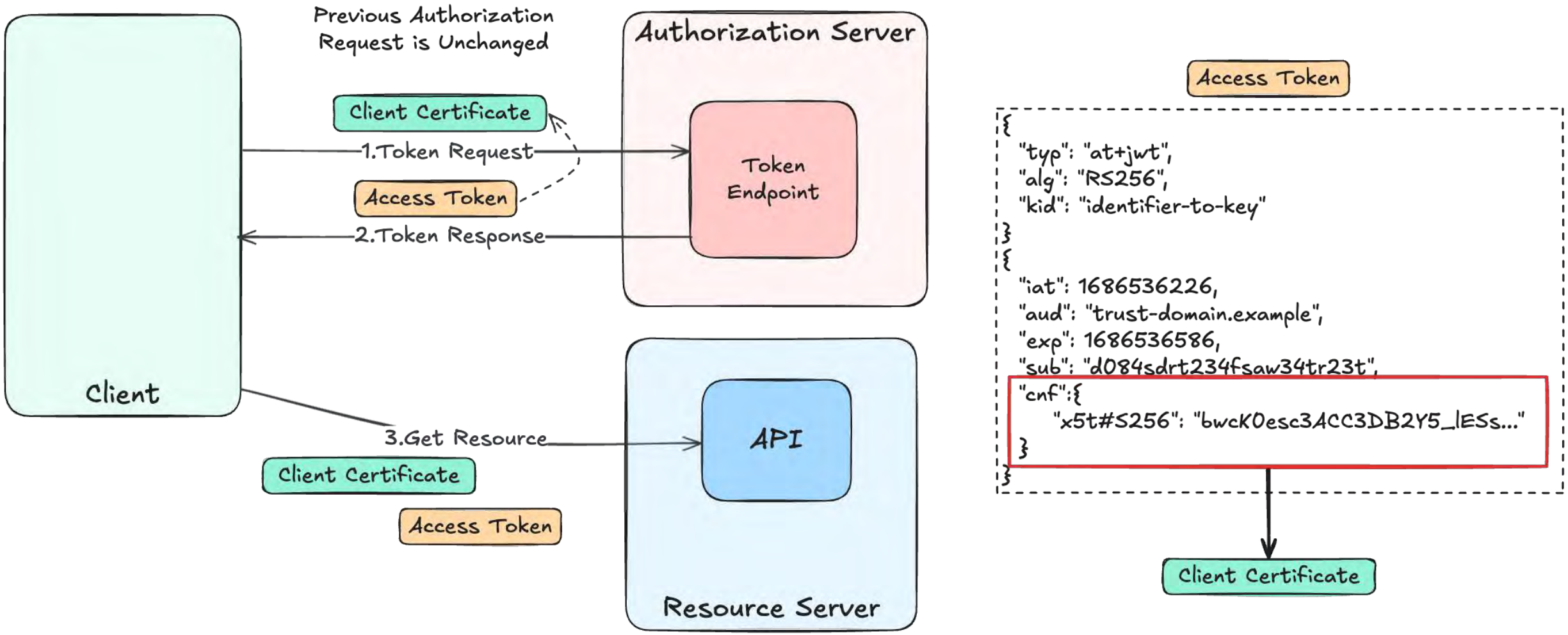
https://www.rfc-editor.org/rfc/rfc9068.html
https://www.rfc-editor.org/rfc/rfc7643.html (SCIM)

Part 3

The Client-Side

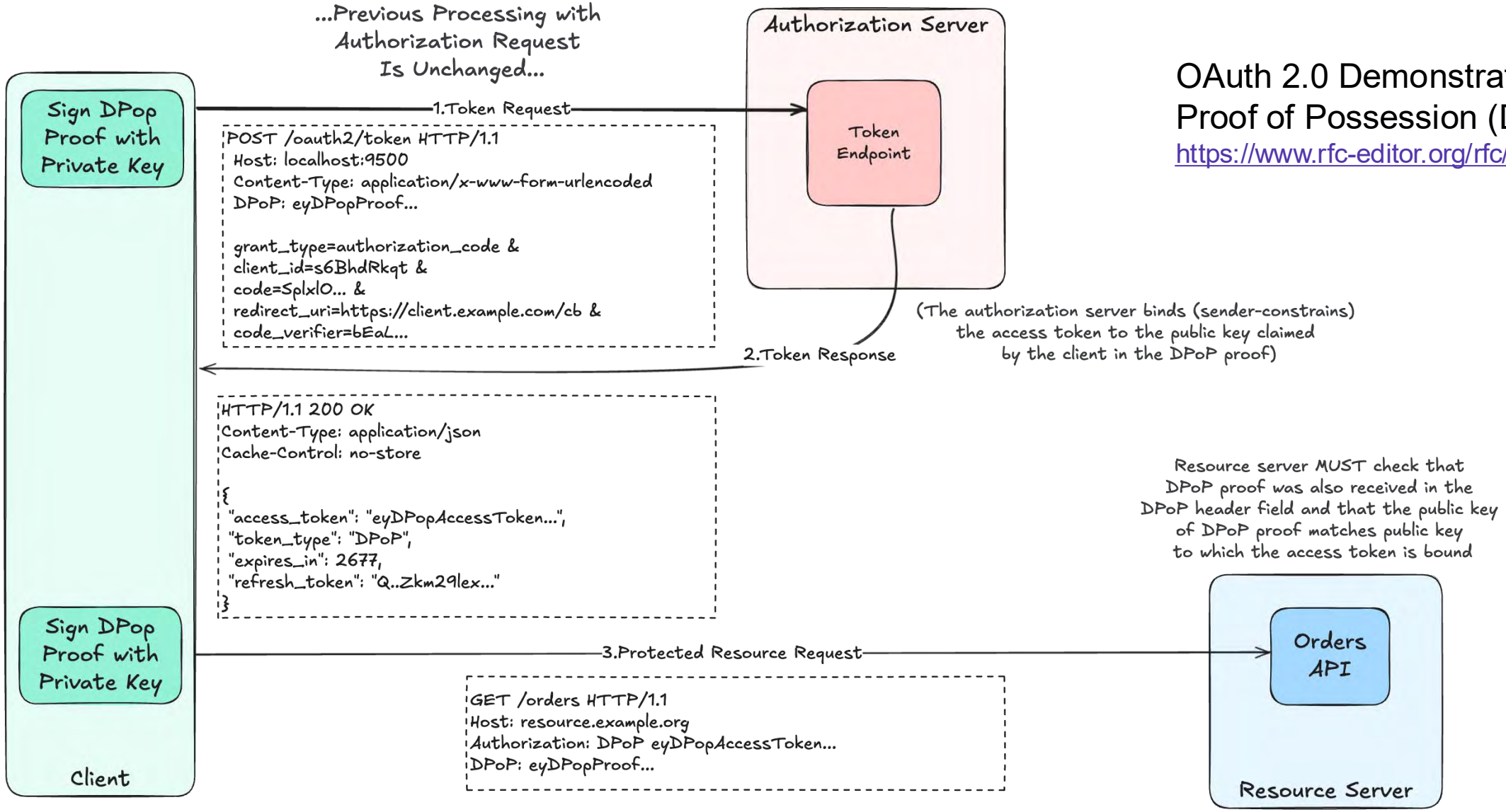


OAuth 2.0 Sender-Constrained Tokens with RFC 8705 (MTLS)



https://www.rfc-editor.org/rfc/rfc8705.html

Sender-Constrained Tokens – RFC 9449: DPoP



OAuth 2.0 Demonstrating Proof of Possession (DPoP)
https://www.rfc-editor.org/rfc/rfc9449.html

https://www.w3.org/TR/2017/REC-WebCryptoAPI-20170126/#SubtleCrypto-method-generateKey

DPoP Proof JWTs & Public Key Confirmation of Access Tokens

DPoP Proof JWT


```
{
  "typ":"dpop+jwt",
  "alg":"ES256",
  "jwk": {
    "kty":"EC",
    "x":"l8tFrhx-34tV3hRICRDY9zCkDlpBhF42UQUfWVAWBFs",
    "y":"9VE4jf_Ok_o64zbTTlcuNJajHmt6v9TDVrU0CdvGRDA",
    "crv":"P-256"
  }
}
.
{
  "jti":"e1j3V_bKic8-LAEB",
  "htm":"GET",
  "htu":"https://resource.example.org/protectedresource",
  "iat":1562262618,
  "ath":"fUHyO2r2Z3DZ53EsNrWBb0xWXoaNy59IiKCAqksmQEo"
}
```

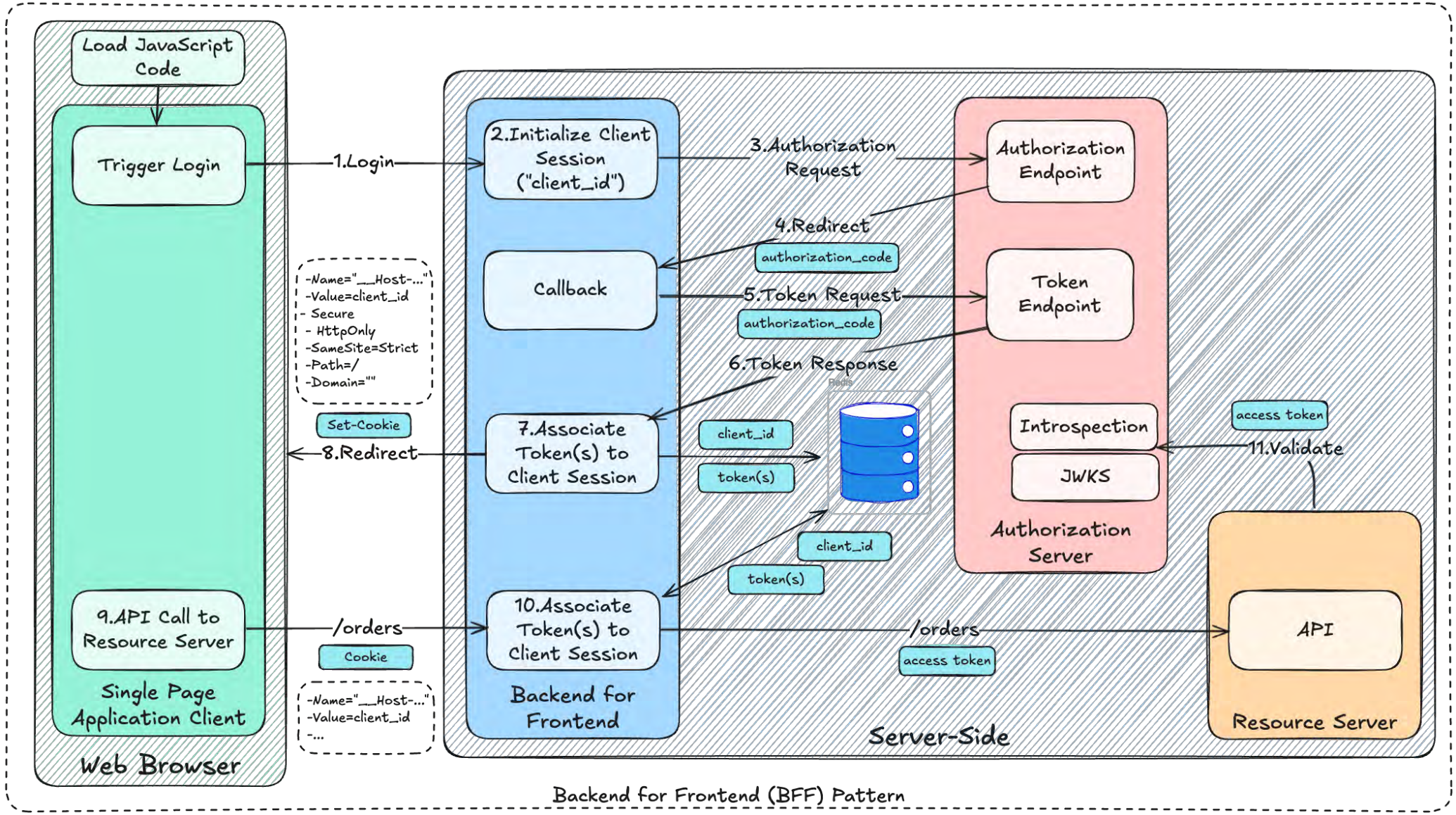

DPoP Bound Access Token


```
{
  "typ":"at+jwt",
  "alg":"ES256",
}
.
{
  "sub":"someone@example.com",
  "iss":"https://server.example.com",
  "nbf":1562262611,
  "exp":1562266216,
  "cnf":
  {
    "jkt":"0ZcOCORZNYy-DWpqq30jZyJGHTN0d2HglBV3uiguA4I"
  }
}
```

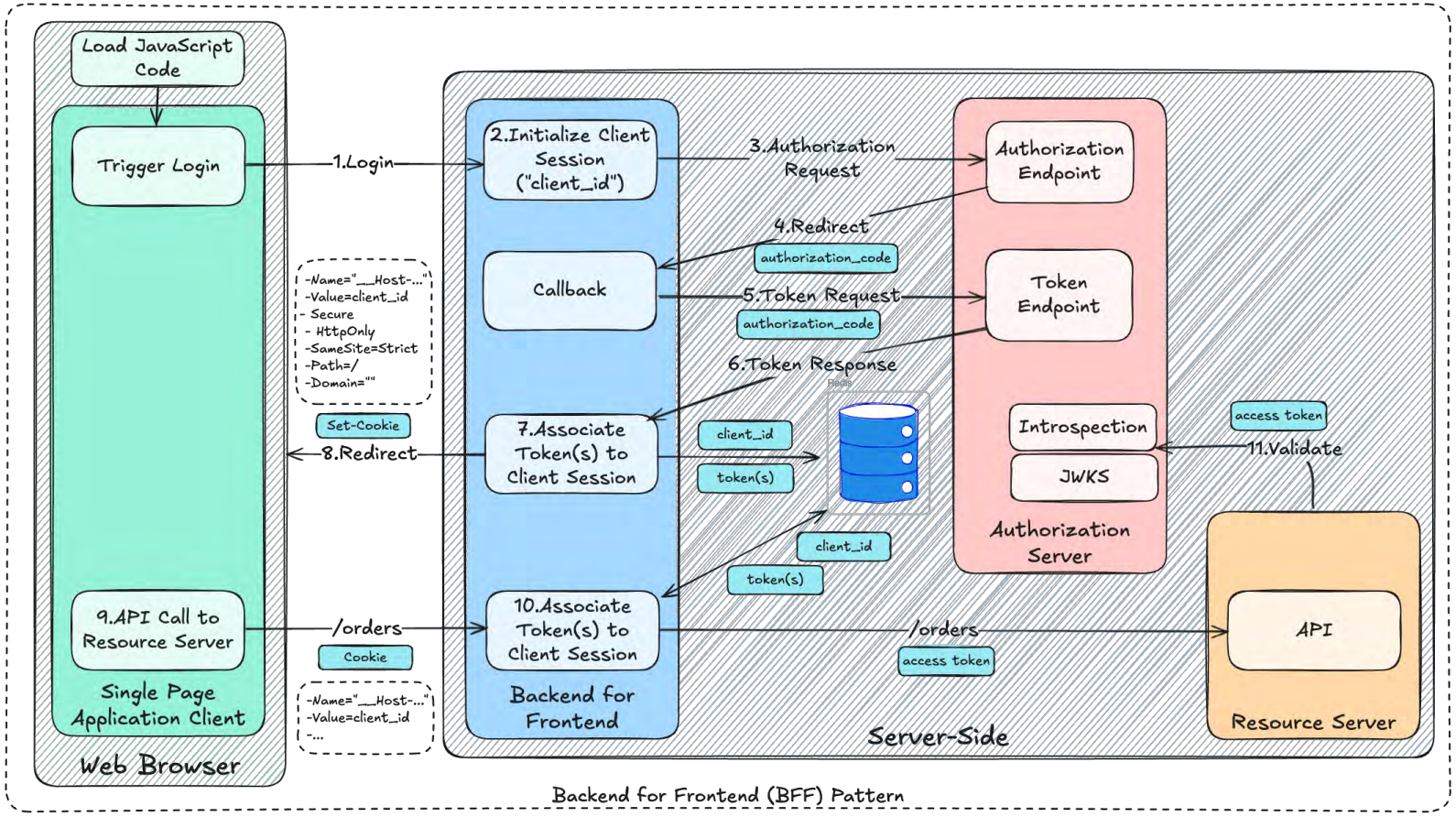

Base64Url encoding of the JWK SHA-256 Thumbprint

Claim	Description
jti	Unique identifier for the DPoP proof JWT
htm	HTTP method of the request to which the JWT is attached
htu	The HTTP target URI of the request to which the JWT is attached, without query and fragment parts.
iat	Creation timestamp of the JWT
ath	Hash of the access token (if used in conjunction with an access token)

Free Browser from Tokens – Backend for Frontend (BFF)



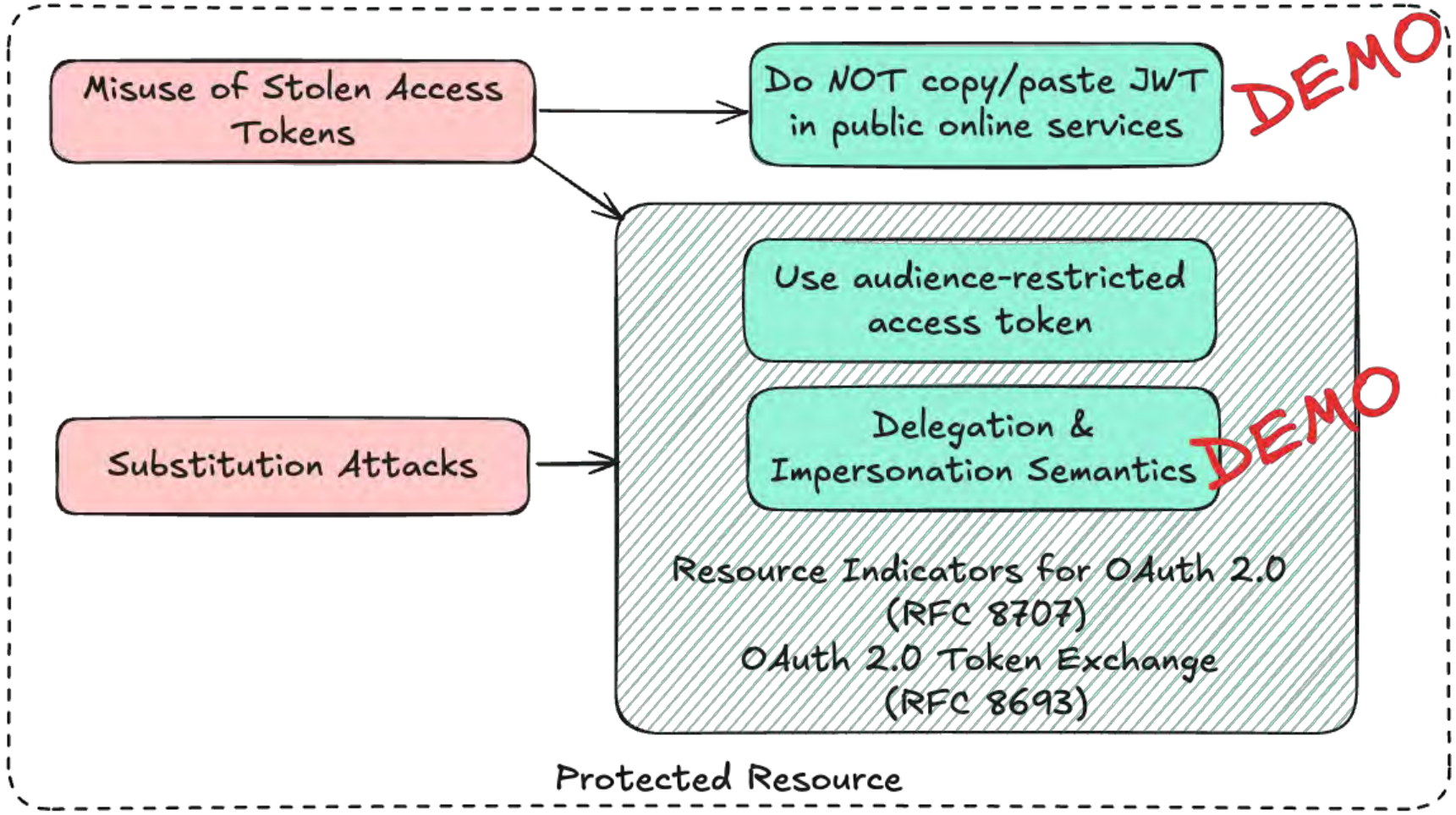
Backend for Frontend (BFF) – Demo



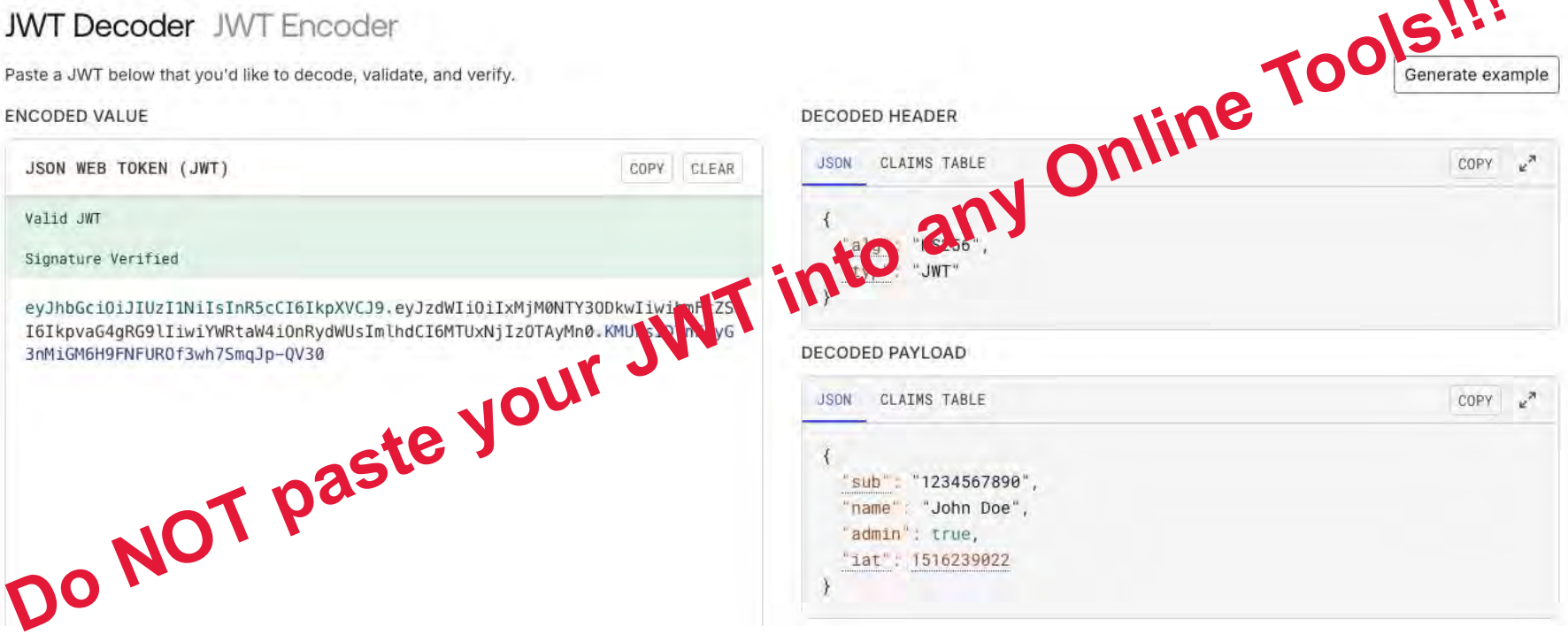
https://github.com/spring-projects/spring-authorization-server/tree/1.5.1/samples

Part 4

Protected Resources



Misuse of Stolen Access Tokens



Do NOT paste your JWT into any Online Tools!!!

https://github.com/smallstep/cli

https://gchq.github.io/CyberChef

https://github.com/ticarpi/jwt_tool (Python)

vscode-jwt-decoder (VS Code Extension)

JWT Analyzer (IntelliJ Plugin)

https://www.pgrs.net/2022/06/02/simple-command-line-function-to-decode-jwts

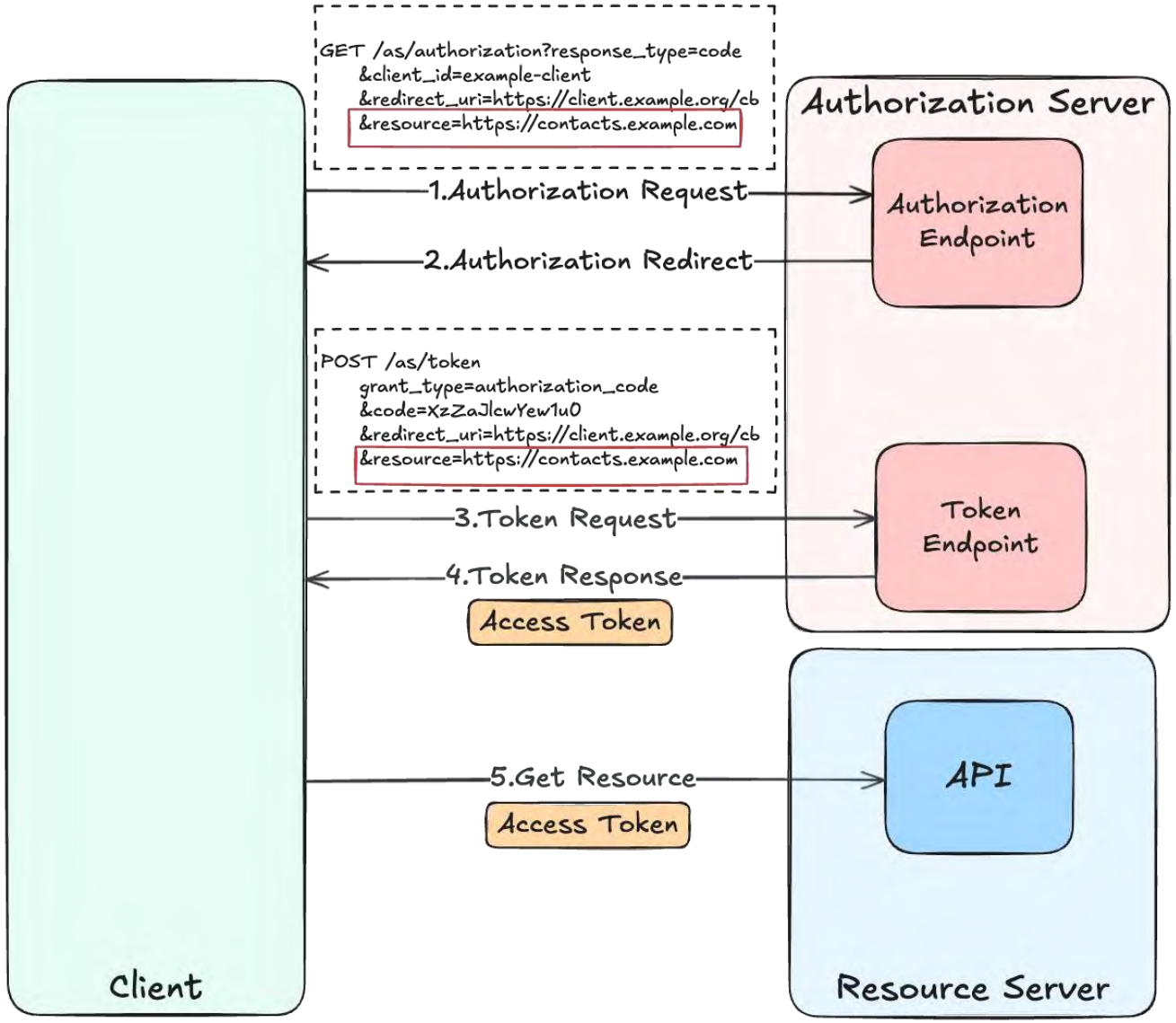
```
jwt-decode() {

  jq -R 'split(".") |.[0:2] | map(gsub("-"; "+") | gsub("_"; "/")
    | gsub("%3D"; "=") | @base64d) | map(fromjson)' <<< $1

}
```

JWT Decoding and Verification - Demo

```
$ TOKEN=$(step oauth --bare --oidc)
$ GOOG="https://www.googleapis.com/oauth2/v3/certs"
$ echo $TOKEN | step crypto jwt verify --jwks $GOOG --subtle
{
  "header": {
    "alg": "RS256",
    "kid": "9a33b5edb49d0867a8672d9573b1e0d2375886e1"
  },
  "payload": {
    "azp": "10816048420-8qt7bavg36it824il.apps.googleusercontent.com",
    "aud": "10816048420-8qt7bavg36it824il.apps.googleusercontent.com",
    "sub": "107273808323495259178",
    "hd": "smallstep.com",
    "email": "bob@smallstep.com",
    "email_verified": true,
    "at_hash": "1tQuEWqaCtRzKp6doePLyQ",
    "exp": 1533599080,
    "iss": "https://accounts.google.com",
    "iat": 1533595480
  },
  "signature": "jbmrziGKXtcWwdXeWf2bIX4hB9m4q4mQsTjtmBOcqsVfblkqvdI_hyA
d4wqJsGw2qMg2RjLi6T8GPtjvs2z1rPbzbsEEr00QvjO-93VNKeGa6DcU9F5AET-gyDEYda
1w-Sg-I35zgBGhDllb0rha6MNmJTzEZ8NXNQJOnaoK82lNarzp0A4sd0STb4vv5dUAJ4"
}
```

https://github.com/andifalk/federated-identity-demos →

Audience-Restricted Tokens – Resource Indicators (RFC 8707)

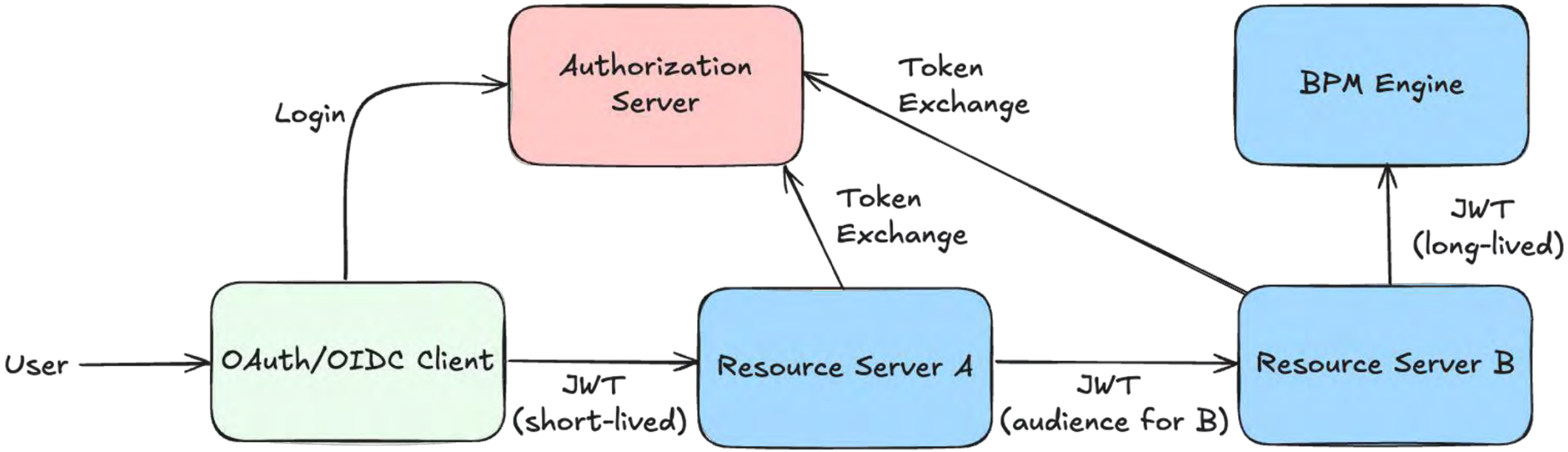


Access Token


```
{
  "typ": "at+jwt",
  "alg": "RS256",
  "kid": "identifier-to-key"
}
{
  "iat": 1686536226,
  "aud": "https://contacts.example.com",
  "exp": 1686536586,
  "sub": "d084sdrt234fsaw34tr23t",
  ...
}
```


https://www.rfc-editor.org/rfc/rfc8707.html

OAuth Token Exchange – RFC 8693



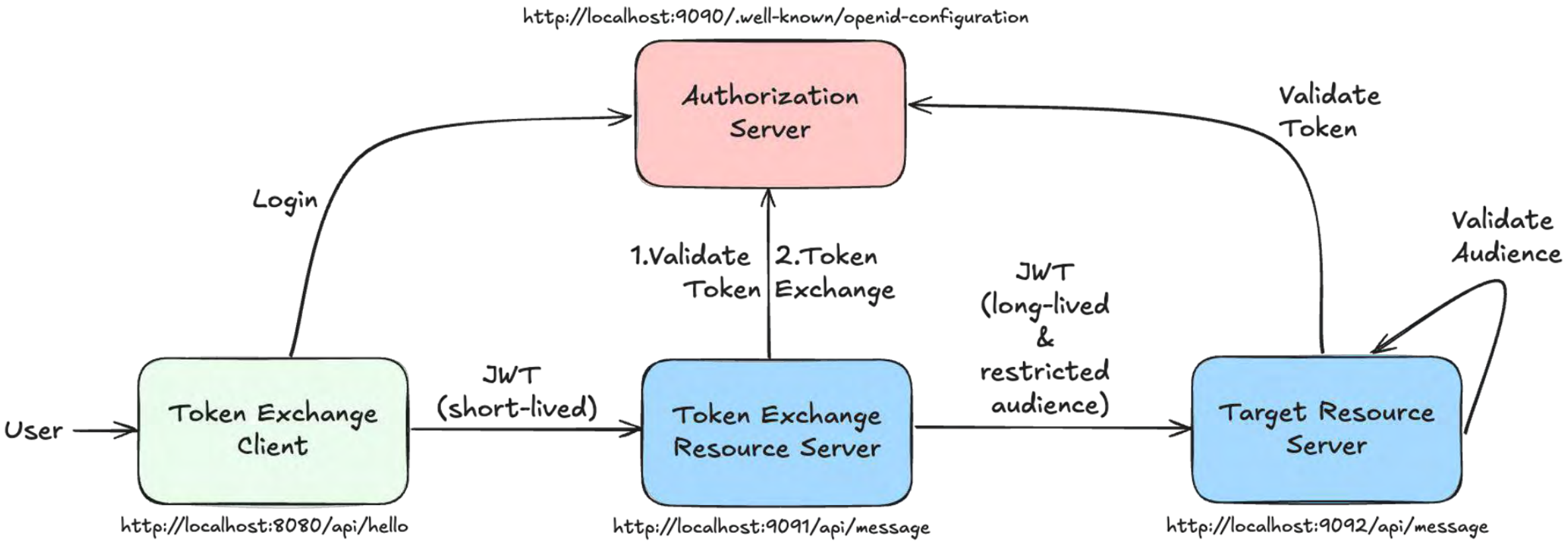
https://www.rfc-editor.org/rfc/rfc8693.html
https://docs.spring.io/spring-security/reference/6.3/whats-new.html - _oauth_2_0_token_exchange_grant_5199

Impersonation vs. Delegation in OAuth Token Exchange

⚖️ Comparison Table (OAuth Token Exchange)

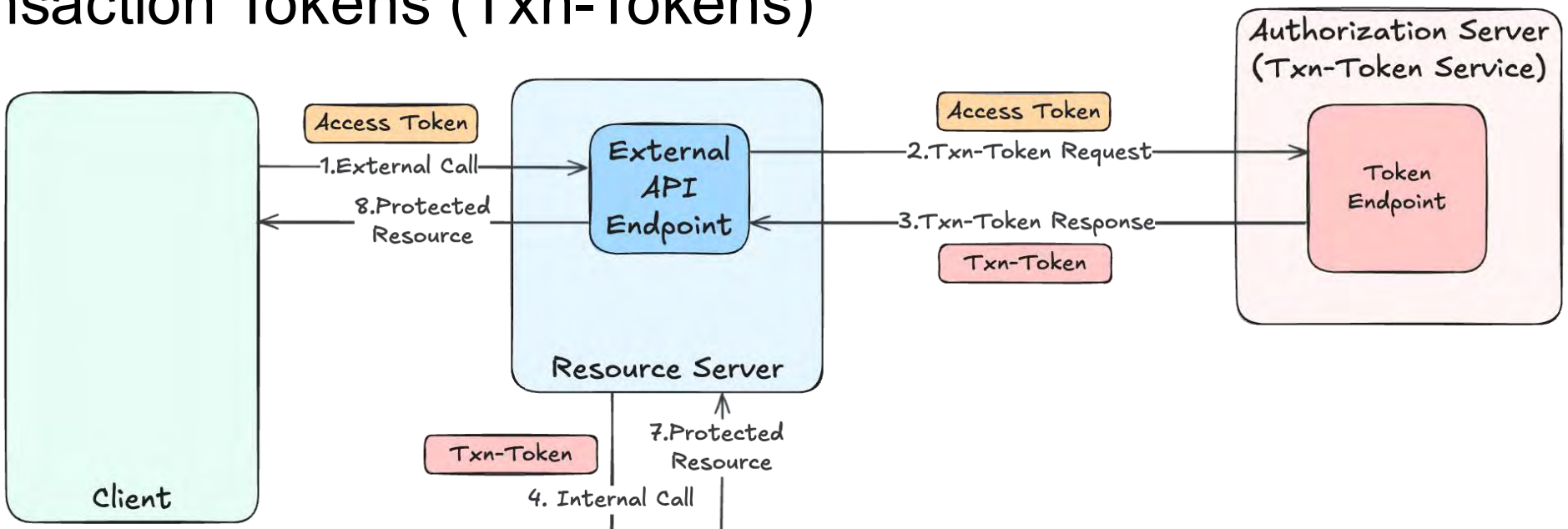
Feature	Impersonation	Delegation
RFC 8693 Usage	Only `subject_token`	`subject_token` + `actor_token`
Identity in access token	Subject only (`sub=user`)	Subject + Actor (`sub=user`, `act=caller`)
Use case example	Login as user / Support admin	Microservice acting on behalf of a user
Auditability	Limited – appears as user only	Full – includes caller identity
Security risk	Higher – hides real caller	Lower – maintains trust chain

OAuth Token Exchange – Demo Scenario



https://github.com/andifalk/federated-identity-demos →

Transaction Tokens (Txn-Tokens)

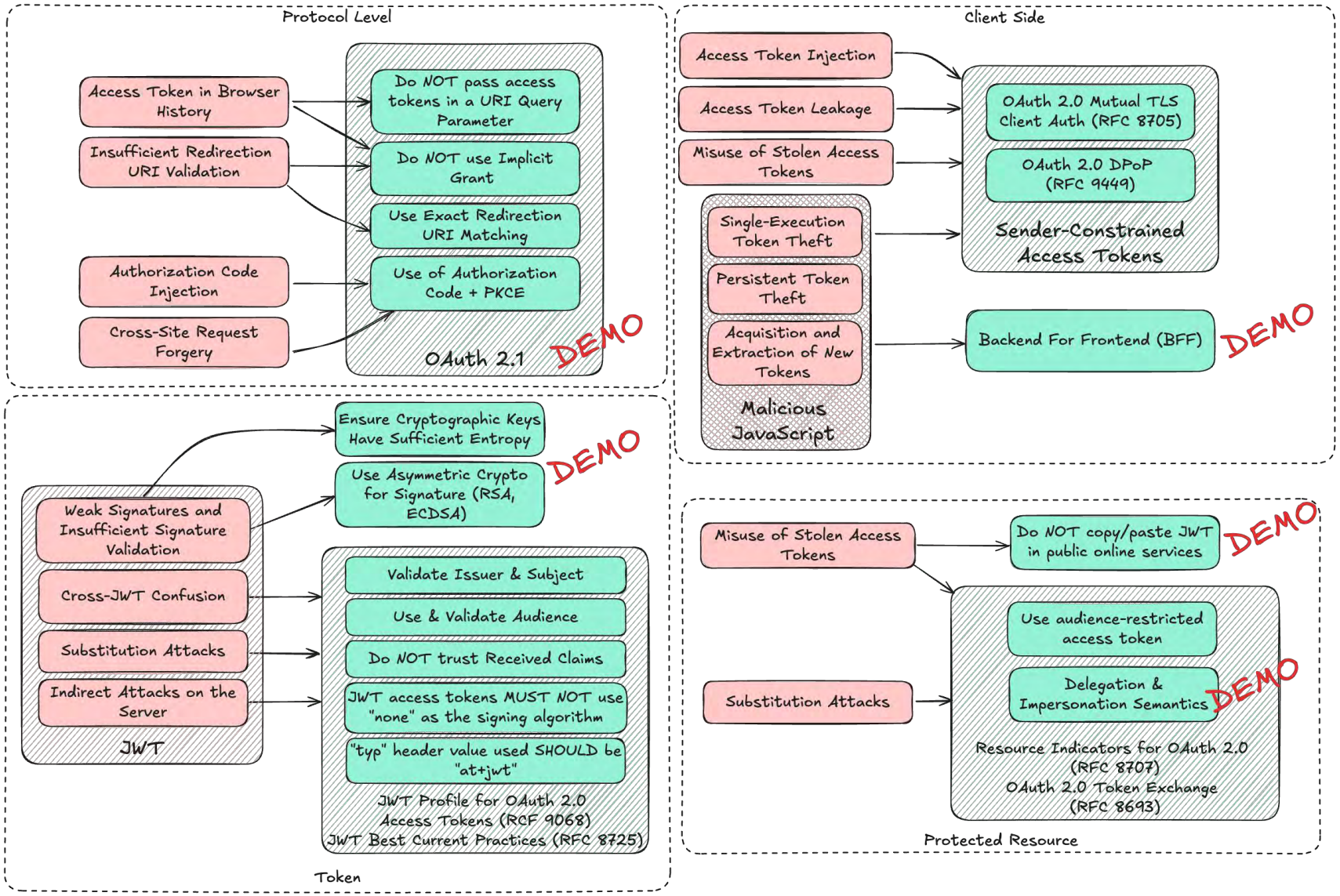



```
{
  "typ": "txntoken+jwt",
  "alg": "RS256",
  "kid": "identifier-to-key"
}
{
  "iat": 1686536226,
  "aud": "trust-domain.example",
  "exp": 1686536586,
  "txn": "97053963-771d-49cc-a4e3-20aad399c312",
  "sub": "d084sdrt234fsaw34tr23t",
  "rctx": {...},
  "purp" : "trade.stocks",
  "tctx": {...}
}
```


https://www.ietf.org/archive/id/draft-ietf-oauth-transaction-tokens-05.html

Summary

Key Take Aways




NEVER implement the
OAuth 2.1 Protocol Yourself !!

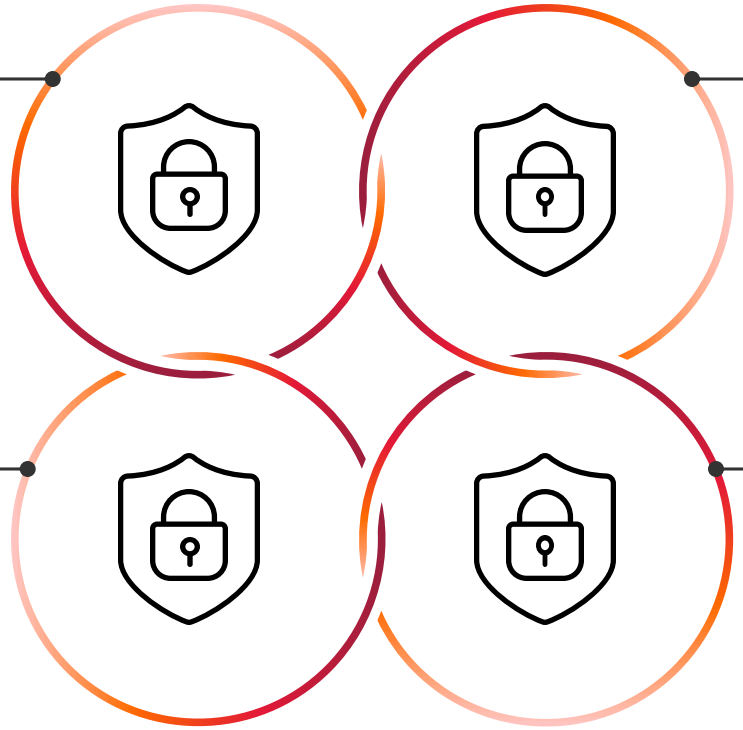
Summary of Mitigations for OAuth 2.1 Attacks

Protocol Layer

- Use OAuth 2.1 Grants Only
- Use PKCE
- Exact Redirect URI Matching
- Do not use Access Token in URI Query Parameters

Token

- Never Copy/Paste Token into Online Services
- Prefer Asymmetric Signing Algorithms (RSA, DHDCE)
- Follow JWT Profile for OAuth 2.0 Access Tokens (RFC 9068)



The Client-Side

- Use Sender-Constraint Tokens (MTLS, DPoP)
- Free Browser from Tokens with Backend for Frontend (BFF)
- Protect SPA from XSS Attacks

Protected Resources

- Audience-Restricted Tokens with *Resource Bundles* (RFC 8707)
- Impersonation & Delegation Patterns with *Token Exchange* (RFC 8693)
- Look for *Transaction Tokens* (Draft)

Provider Support

Provider	OAuth 2.1 (Draft)	PKCE (RFC 7636)	RFC 9126 (OAuth Security BCP)	RFC 8705 (Mutual TLS)	RFC 9449 (DPoP)	RFC 8725 (JWT BCP)	RFC 9068 (JWT Profile for Access Tokens)	RFC 8693 (Token Exchange)
Auth0	✅	✅	🔶 Partial	🔶 Enterprise Add-on	✅ (Beta)	✅	🔶 (Experimental)	🔶 (Beta via Rules/Hooks)
MS Entra ID	✅	✅	✅	🔶 Confidential Client + Certs	❌	✅	❌	🔶 (Entra ID - Limited)
Google Identity	✅	✅	🔶 Partial	❌	❌	✅	❌	❌
Okta	✅	✅	✅	✅ (with Workflows)	✅ (Preview)	✅	🔶 (Preview for APIs)	🔶 (Some API Gateways only)
Keycloak	✅	✅	✅	✅	✅ (v24+)	✅	✅ (via config)	✅ (v24+)
ForgeRock	✅	✅	✅	✅	✅	✅	✅	✅
Ping Identity	✅	✅	✅	✅	✅	✅	✅	✅
Curity	✅	✅	✅	✅	✅	✅	✅	✅
AWS Cognito	✅	✅	🔶 Partial	❌	❌	✅	❌	❌
Spring Authorization Server	✅	✅	✅	✅	✅	✅	✅	✅

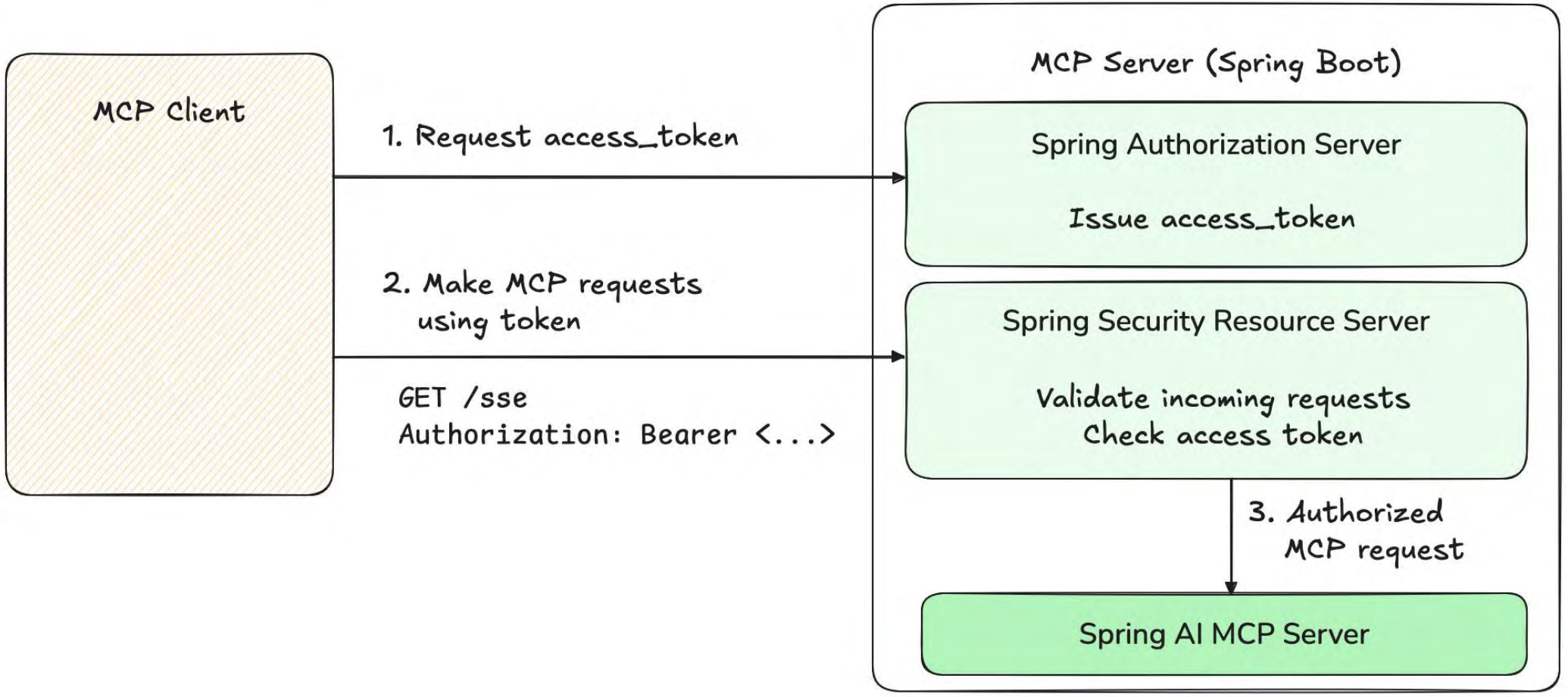
Legend:

- ✅ = Fully supported
- 🔶 = Partially supported / Preview / Requ
- ❌ = Not supported or not documented

https://github.com/andifalk/federated-identity-demos

ONE MORE THING…

AI Model Context Protocol (MCP) Security with OAuth2



https://modelcontextprotocol.io/specification/2025-03-26
https://spring.io/blog/2025/04/02/mcp-server-oauth2
https://aaronparecki.com/2025/04/03/15/oauth-for-model-context-protocol

https://github.com/andifalk/federated-identity-demos

Thank You! Any Questions?

Andreas Falk

andreas.falk@cgi.com

Connect with me

https://www.linkedin.com/in/andifalk

References

- https://github.com/OWASP/ASVS (Application Security Verification Standard)
- https://datatracker.ietf.org/wg/oauth/documents (IETF OAuth Working Group)
- https://datatracker.ietf.org/doc/draft-ietf-oauth-v2-1 (OAuth 2.1)
- https://datatracker.ietf.org/doc/draft-ietf-oauth-browser-based-apps (SPA Best Practice)
- https://datatracker.ietf.org/doc/draft-ietf-oauth-transaction-tokens (Transaction Tokens)
- https://datatracker.ietf.org/doc/rfc9700 (Best Current Practice for OAuth 2.0 Security)
- https://datatracker.ietf.org/doc/rfc9449 (DPoP)
- https://datatracker.ietf.org/doc/rfc9126 (OAuth 2.0 Pushed Authorization Requests)
- https://datatracker.ietf.org/doc/rfc9068 (JWT Profile for OAuth 2.0 Access Tokens)
- https://datatracker.ietf.org/doc/rfc8693 (OAuth 2.0 Token Exchange)
- https://datatracker.ietf.org/doc/rfc8725 (JSON Web Token Best Current Practices)
- https://openid.net/wg/connect/specifications (OpenID Connect Specifications)
- https://github.com/spring-projects/spring-authorization-server/tree/1.5.1/samples