Identity & Access Control

(Past) Present & Future

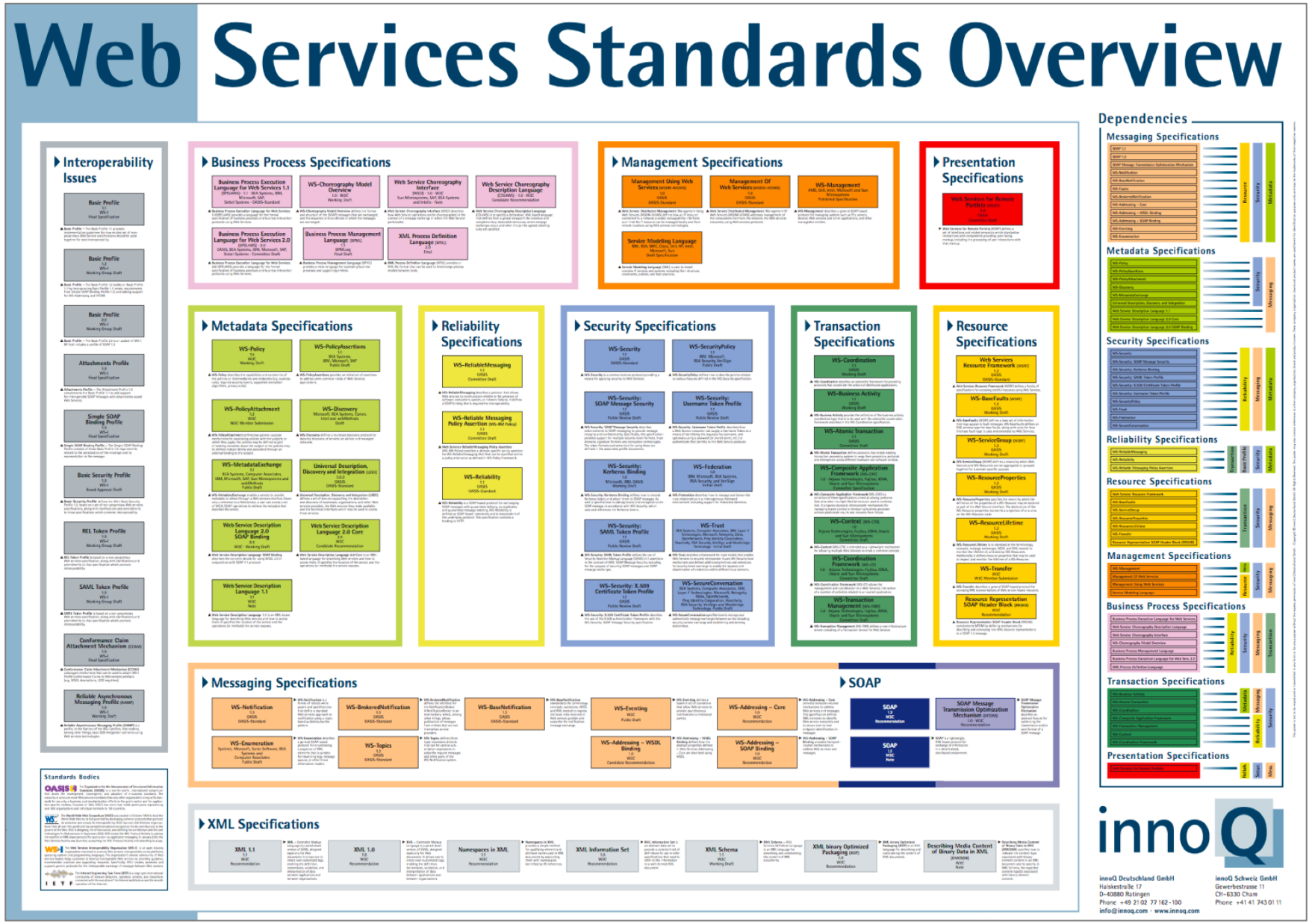
Dominick Baier
@leastprivilege

Me

- **Independent Consultant**
  - Specializing on Application Security Architectures
  - Working with Software Development Teams (ISVs and in-house)

- **Creator and Maintainer of IdentityServer OSS Project**
  - Certified OpenID Connect & OAuth 2.0 Implementation for .NET
  - https://identityserver.io

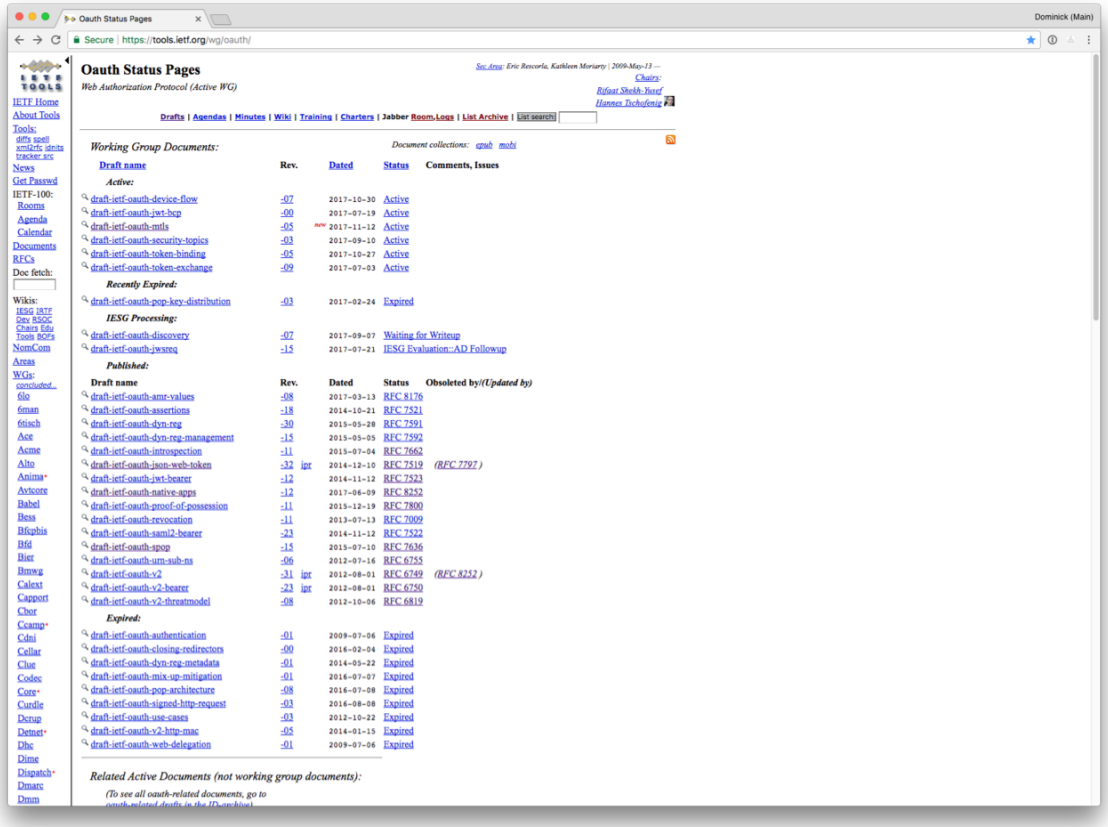
email **dominick.baier@leastprivilege.com**
blog **http://leastprivilege.com**
twitter **@leastprivilege**
slides **https://speakerdeck.com/leastprivilege**



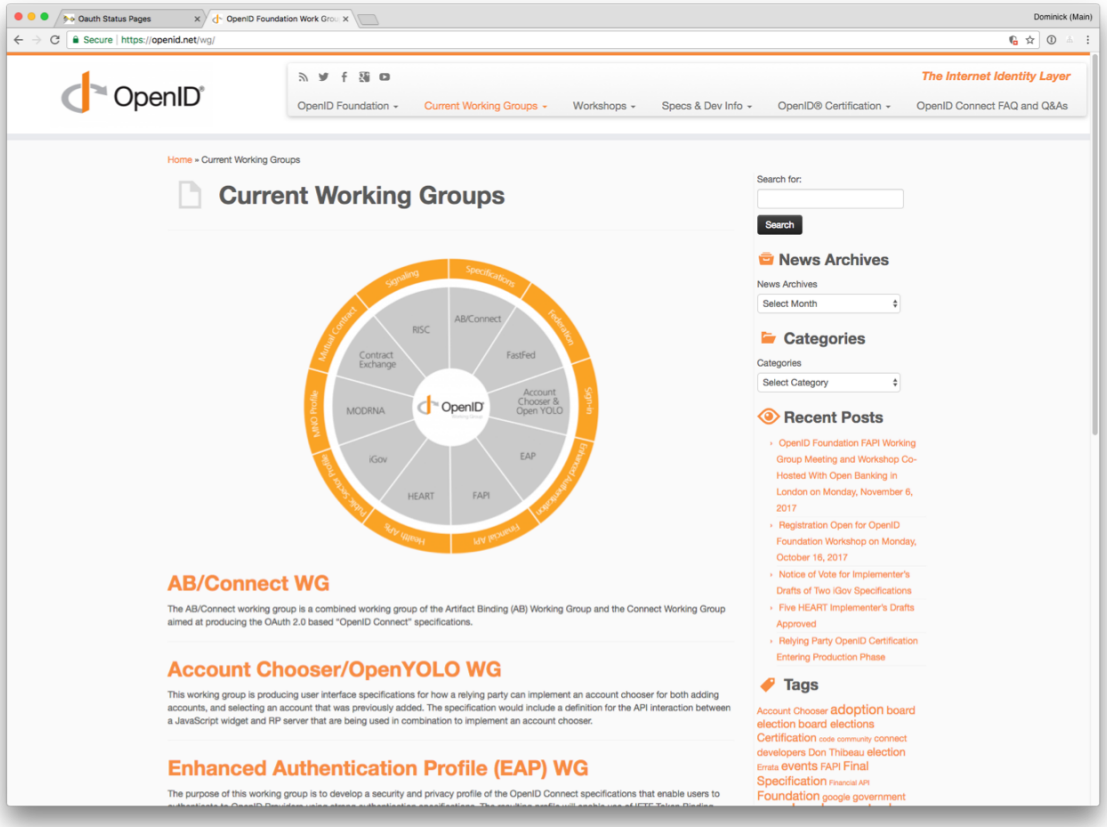
https://www.innoq.com/soa/ws-standards/poster/innoQ%20WS-Standards%20Poster%202007-02.pdf

Where to look?

https://tools.ietf.org/wg/oauth/



https://openid.net/wg/



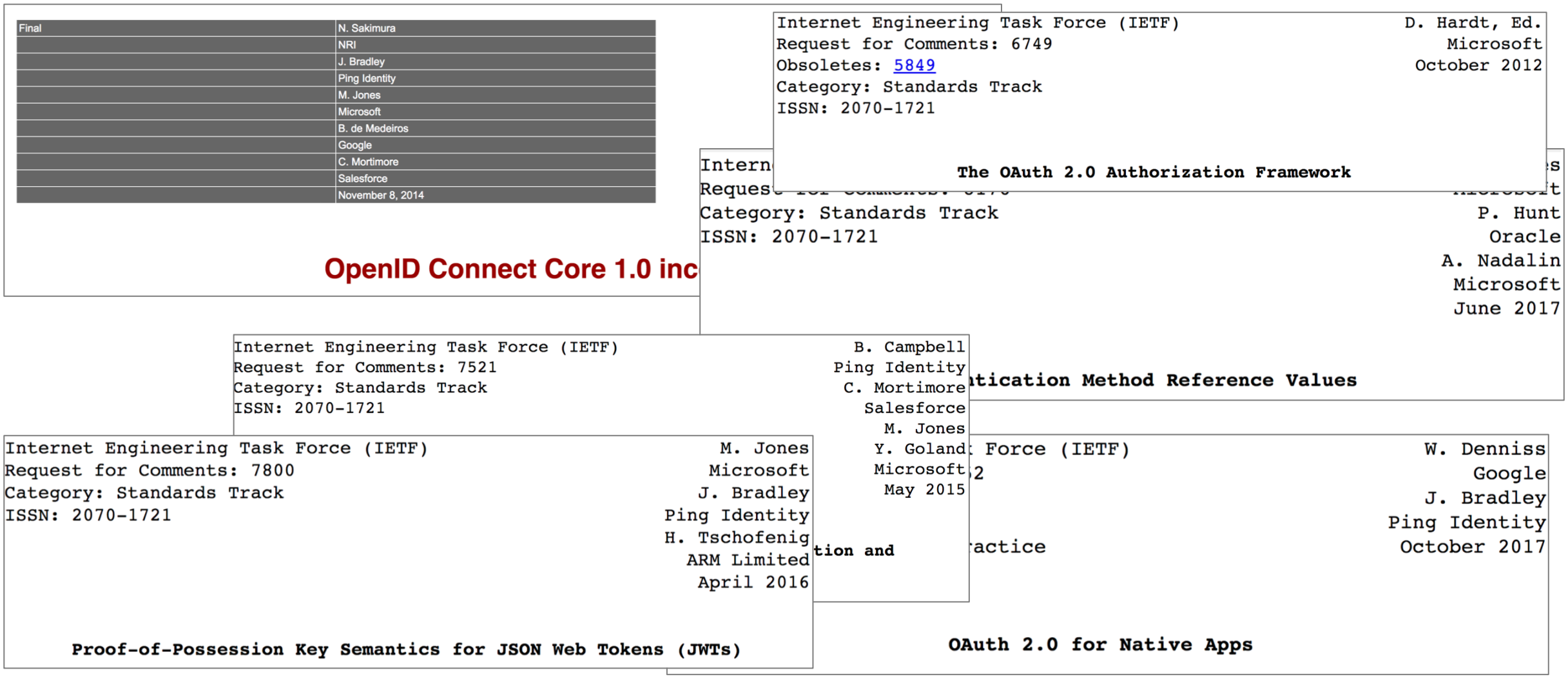
IETF

- **Done**
  - RFC6749: The OAuth 2.0 Authorization Framework
  - RFC6750: The OAuth 2.0 Authorization Framework: Bearer Token Usage
  - RFC6819: OAuth 2.0 Threat Model and Security Considerations
  - RFC7009: OAuth 2.0 Token Revocation
  - RFC7519: JSON Web Tokens (JWT)
  - RFC7521: Assertion Framework for Client Authentication and Authorization Grants
    - RFC7522: SAML Profile
    - RFC7523: JWT Profile
  - RFC7591: OAuth 2.0 Dynamic Client Registration Protocol
  - RFC7592: OAuth 2.0 Dynamic Client Registration Management Protocol
  - RFC7636: Proof Key for Code Exchange by OAuth Public Clients
  - RFC7662: OAuth 2.0 Token Introspection
  - RFC7800: Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)
  - RFC8176: Authentication Method Reference Values
  - RFC8252: OAuth 2.0 for Native Apps
- **Processing**
  - OAuth 2.0 Device Flow for Browserless and Input Constrained Devices
  - OAuth 2.0 Authorization Server Metadata
  - JWT Secured Authorization Request
  - OAuth 2.0 Token Exchange
- **Active**
  - JSON Web Token Best Current Practices
  - OAuth Security Topics
  - Mutual TLS Profile for OAuth 2.0
  - OAuth 2.0 Token Binding
- **Recently expired**
  - OAuth 2.0 Proof-of-Possession: Authorization Server to Client Key Distribution

OpenID Foundation

- **OpenID Connect**
  - Core
  - Discovery
  - Dynamic Registration
  - Session Management
    - Front-Channel Logout
    - Back-Channel Logout
  - Federation
- **Other Working Groups**
  - Enhanced Authentication Profiles (EAP)
  - Financial APIs (FAPI)
  - Mobile Operator Discovery, Registration & autheNticAtion (MODRNA)
  - Health Relationship Trust (HEART)
  - International Government Assurance (iGov)
  - more…

Spec Authors



Final	N. Sakimura
	NRI
	J. Bradley
	Ping Identity
	M. Jones
	Microsoft
	B. de Medeiros
	Google
	C. Mortimore
	Salesforce
	November 8, 2014

OpenID Connect Core 1.0 inc

Internet Engineering Task Force (IETF) — D. Hardt, Ed.
Request for Comments: 6749 — Microsoft
Obsoletes: 5849 — October 2012
Category: Standards Track
ISSN: 2070-1721

The OAuth 2.0 Authorization Framework

Intern…
Reques… for Comments: 8176
Category: Standards Track — P. Hunt, Oracle
ISSN: 2070-1721 — A. Nadalin, Microsoft
June 2017

…tication Method Reference Values

Internet Engineering Task Force (IETF) — B. Campbell, Ping Identity
Request for Comments: 7521 — C. Mortimore, Salesforce
Category: Standards Track — M. Jones
ISSN: 2070-1721 — Y. Goland, Microsoft
May 2015

Internet Engineering Task Force (IETF) — M. Jones, Microsoft
Request for Comments: 7800 — J. Bradley, Ping Identity
Category: Standards Track — H. Tschofenig, ARM Limited
ISSN: 2070-1721 — April 2016

Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)

…Force (IETF) — W. Denniss, Google
…2 — J. Bradley, Ping Identity
…ractice — October 2017

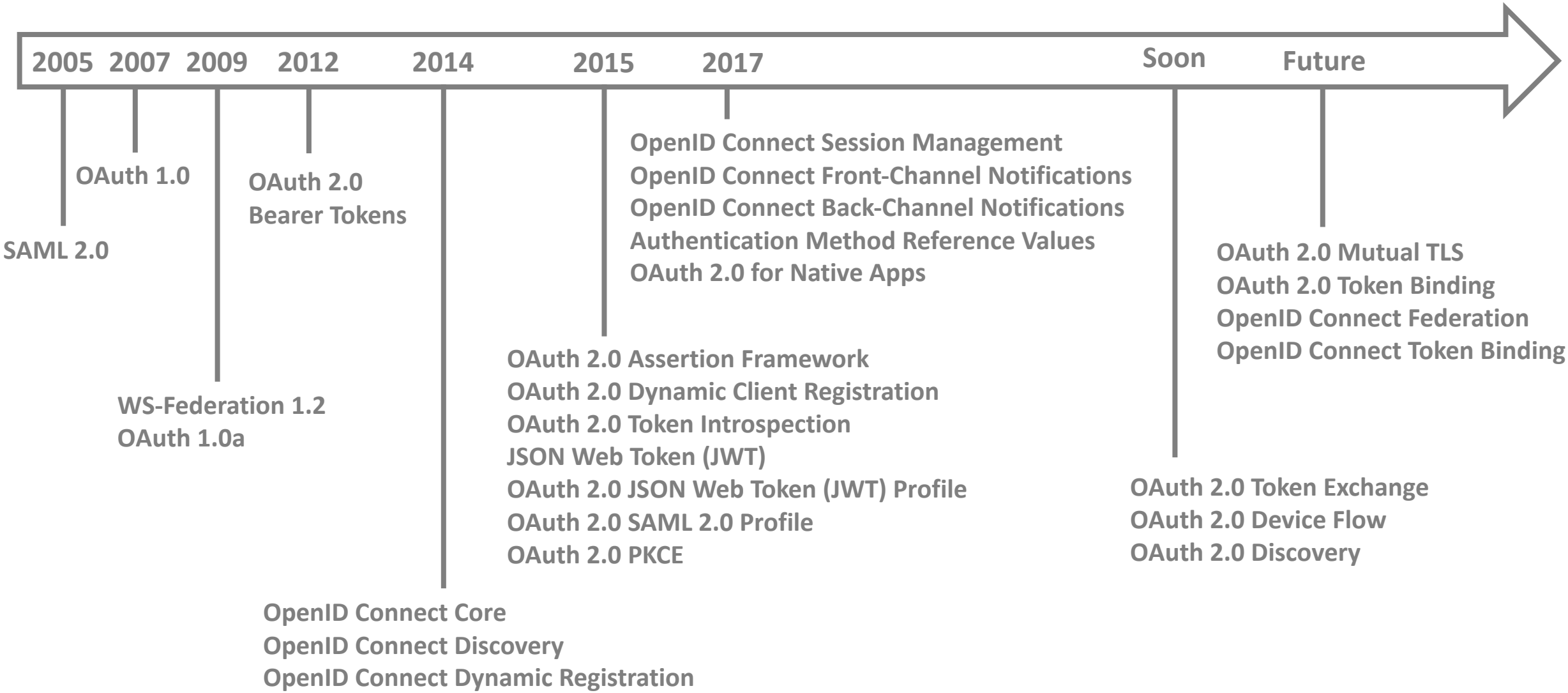
…tion and

OAuth 2.0 for Native Apps

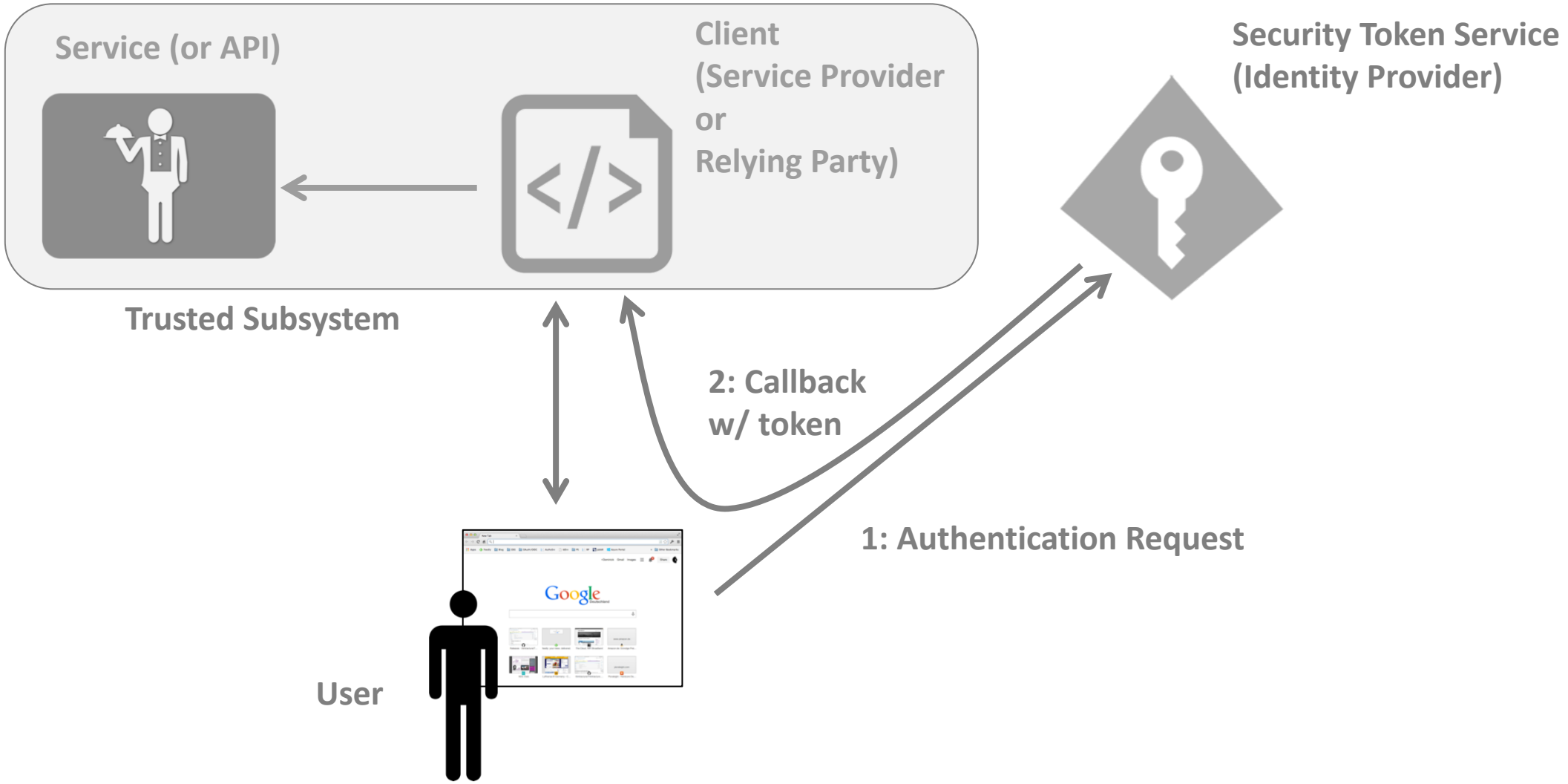
Blogs

- **Mike Jones**
  - http://self-issued.info/
- **John Bradley**
  - http://www.thread-safe.com/
- **Nat Sakamura**
  - https://nat.sakimura.org/

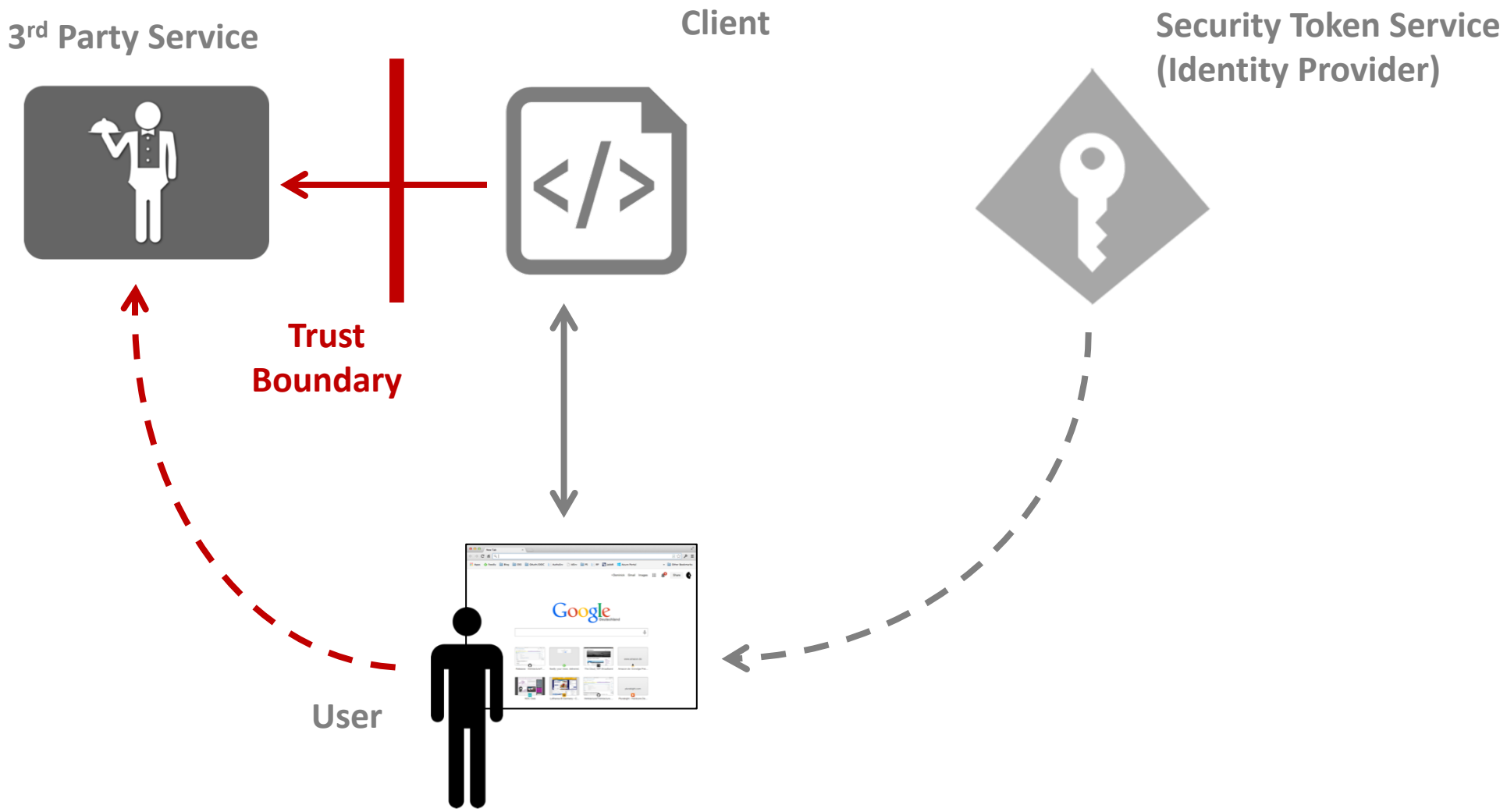
Timeline


2005
2007
2009
2012
2014
2015
2017
Soon
Future
SAML 2.0
OAuth 1.0
OAuth 2.0
Bearer Tokens
WS-Federation 1.2
OAuth 1.0a
OpenID Connect Core
OpenID Connect Discovery
OpenID Connect Dynamic Registration
OAuth 2.0 Assertion Framework
OAuth 2.0 Dynamic Client Registration
OAuth 2.0 Token Introspection
JSON Web Token (JWT)
OAuth 2.0 JSON Web Token (JWT) Profile
OAuth 2.0 SAML 2.0 Profile
OAuth 2.0 PKCE
OpenID Connect Session Management
OpenID Connect Front-Channel Notifications
OpenID Connect Back-Channel Notifications
Authentication Method Reference Values
OAuth 2.0 for Native Apps
OAuth 2.0 Token Exchange
OAuth 2.0 Device Flow
OAuth 2.0 Discovery
OAuth 2.0 Mutual TLS
OAuth 2.0 Token Binding
OpenID Connect Federation
OpenID Connect Token Binding

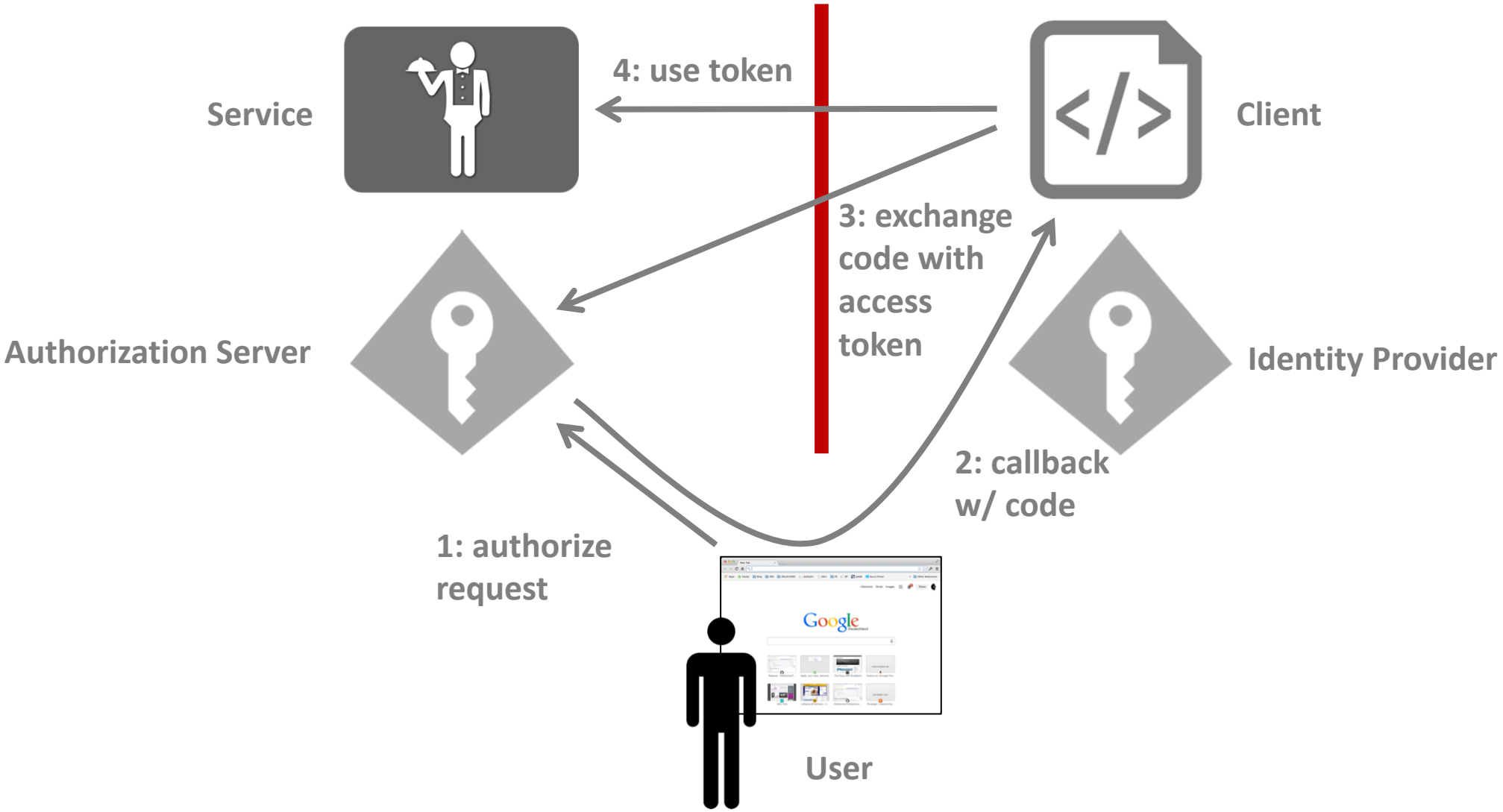
How it all began: Web-based SSO



3rd Parties / Identity Delegation



OAuth


Service
4: use token
Client
3: exchange code with access token
Authorization Server
Identity Provider
2: callback w/ code
1: authorize request
Google
User

Public Clients

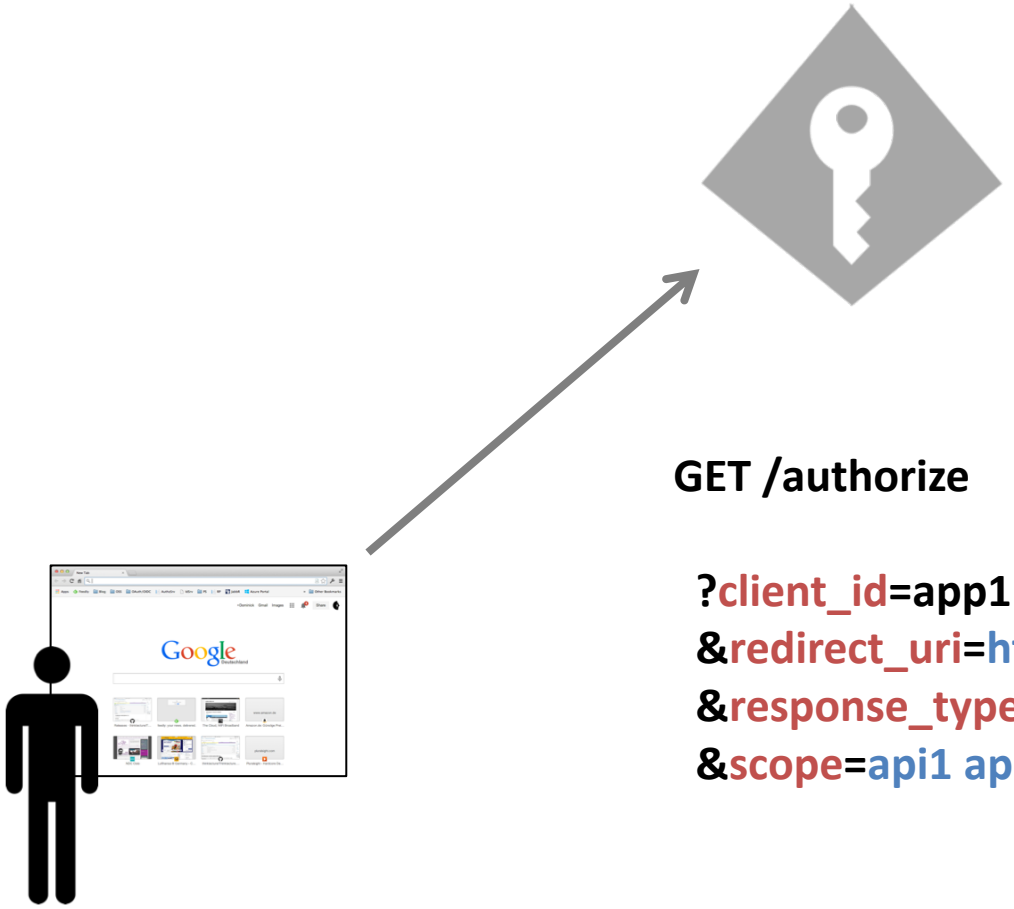
OAuth 2

- **Simplification**
  - bearer tokens

- **Support for public clients**
  - client secret optional
  - implicit grant type specifically made for JavaScript

  - client credentials grant for server-to-server communication
  - password grant type for legacy applications

Access Token Request



GET /authorize

?client_id=app1
&redirect_uri=https://app.com/cb.html
&response_type=token
&scope=api1 api2

Response

GET /callback.html

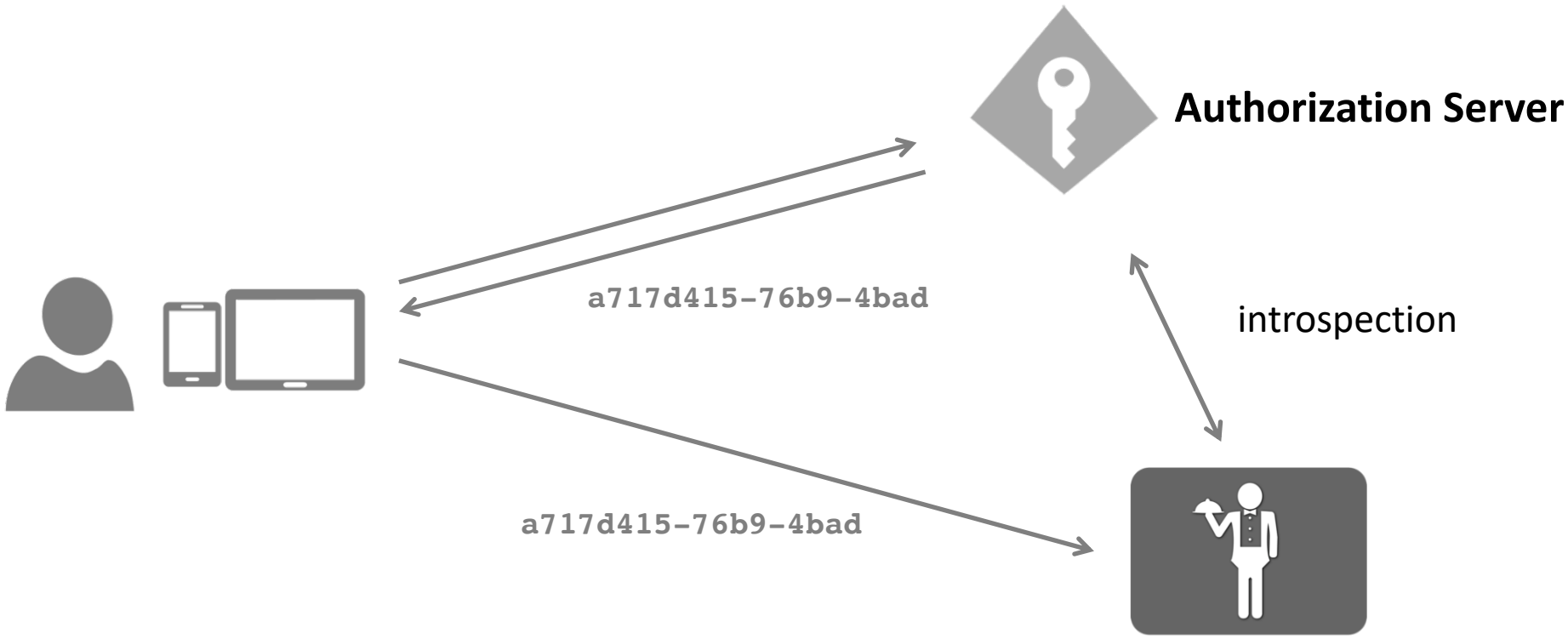
#token=32x…133
&expires_in=3600
&token_type=Bearer

The token problem (Part 1)

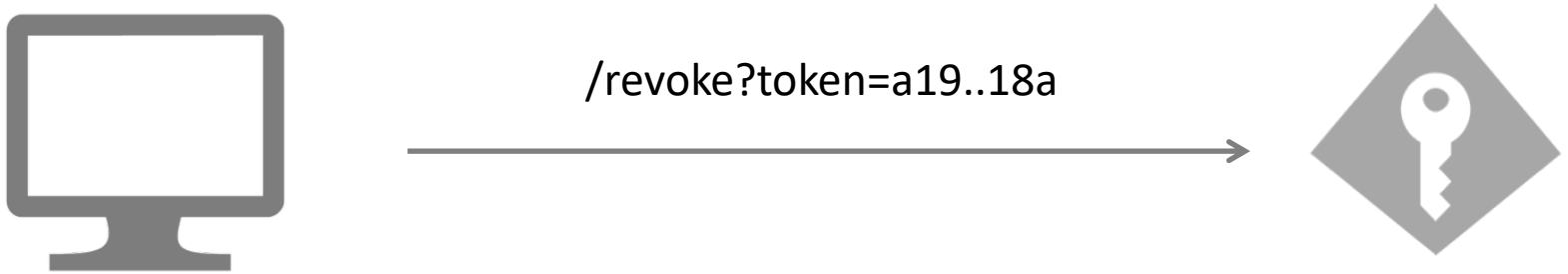
- **OAuth 2.0 did not define a token format**
  - many homegrown (and thus incompatible) implementations

- **Token introspection**
  - turn opaque tokens into claims
- **Token revocation**
  - get rid of tokens

Token Introspection (RFC 7662)



Token Revocation (RFC 7009)



JSON Web Tokens (JWT)

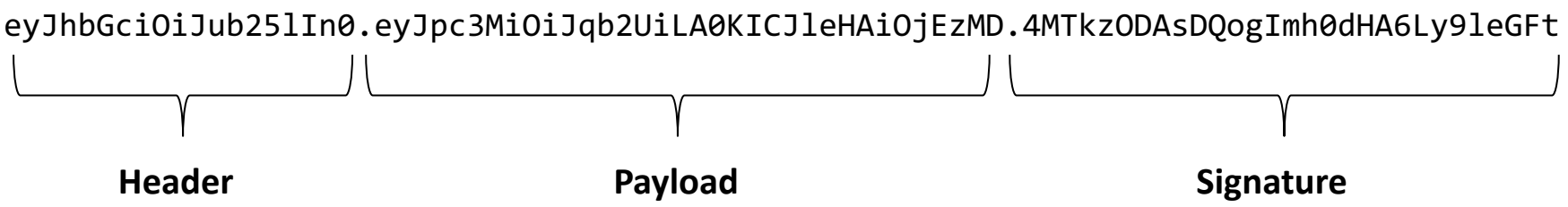
- **Family of RFCs dealing with structure, signatures, encryption, key material..**
- **Standard claim types**
  - https://www.iana.org/assignments/jwt/jwt.xhtml

Header

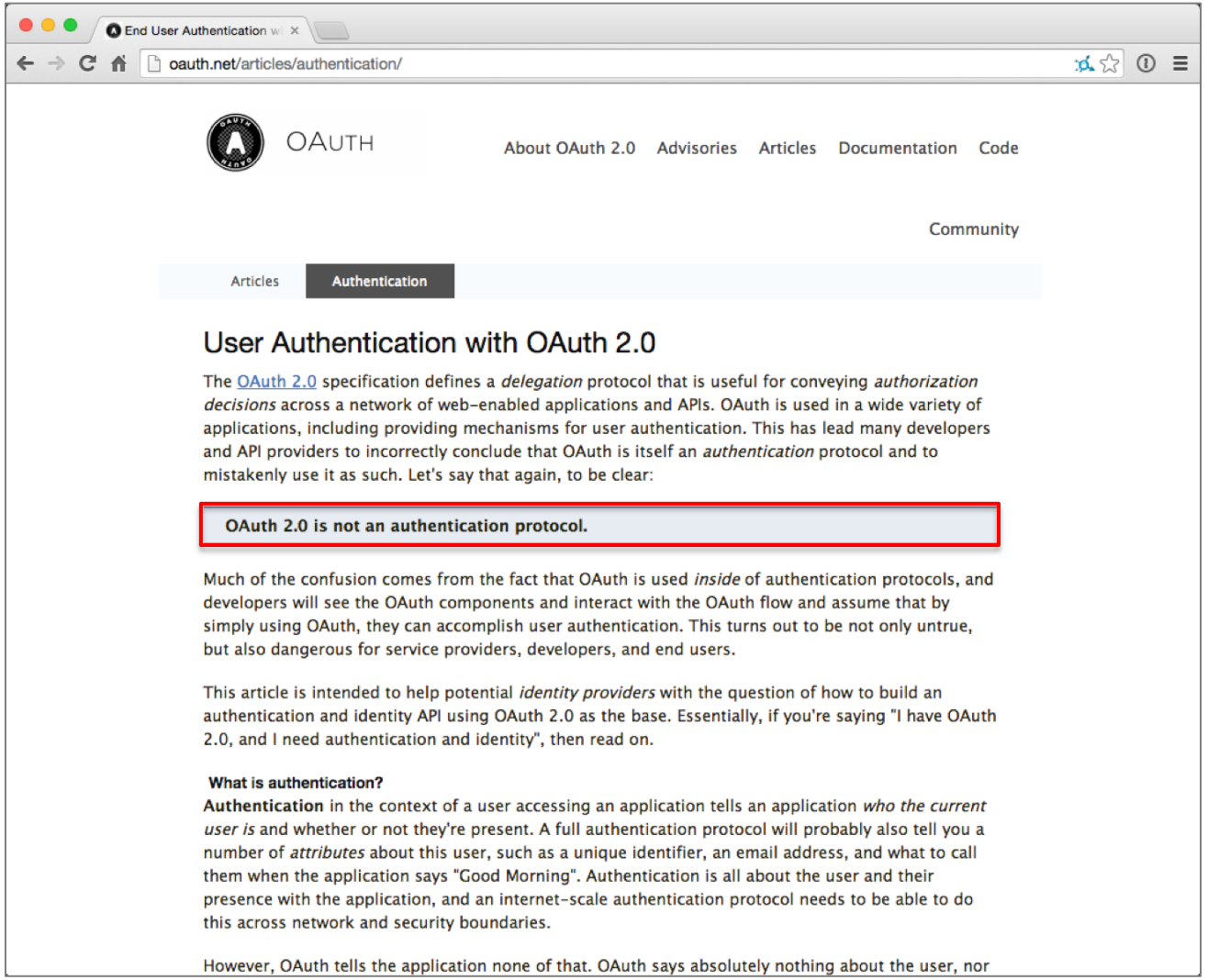
```
{
  "typ": "JWT",
  "alg": "RS256"
  "kid": "1"
}
```

Payload

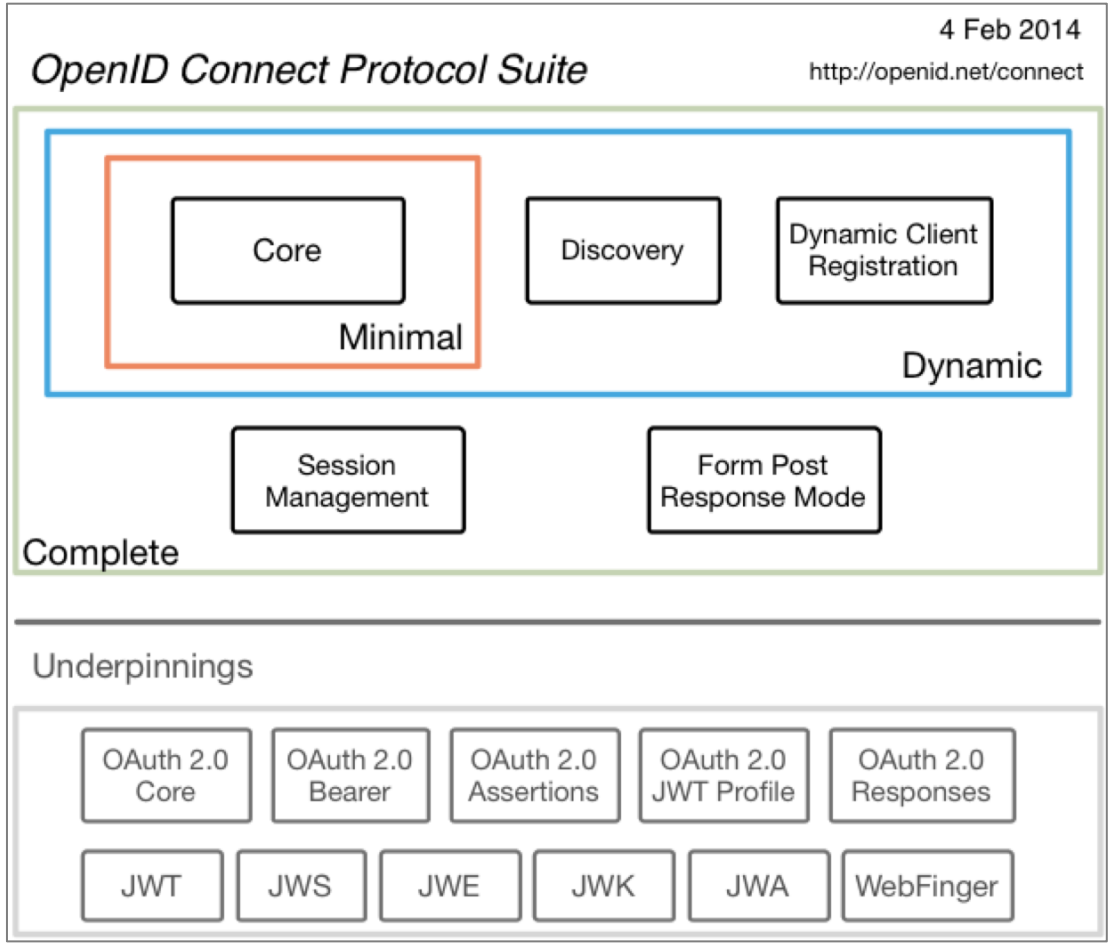
```
{
  "iss": "http://myIssuer",
  "exp": "1340819380",
  "aud": "http://myResource",

  "client_id": "client1",
  "user_id": "bob"
}
```



The "success" of OAuth 2



OpenID Connect



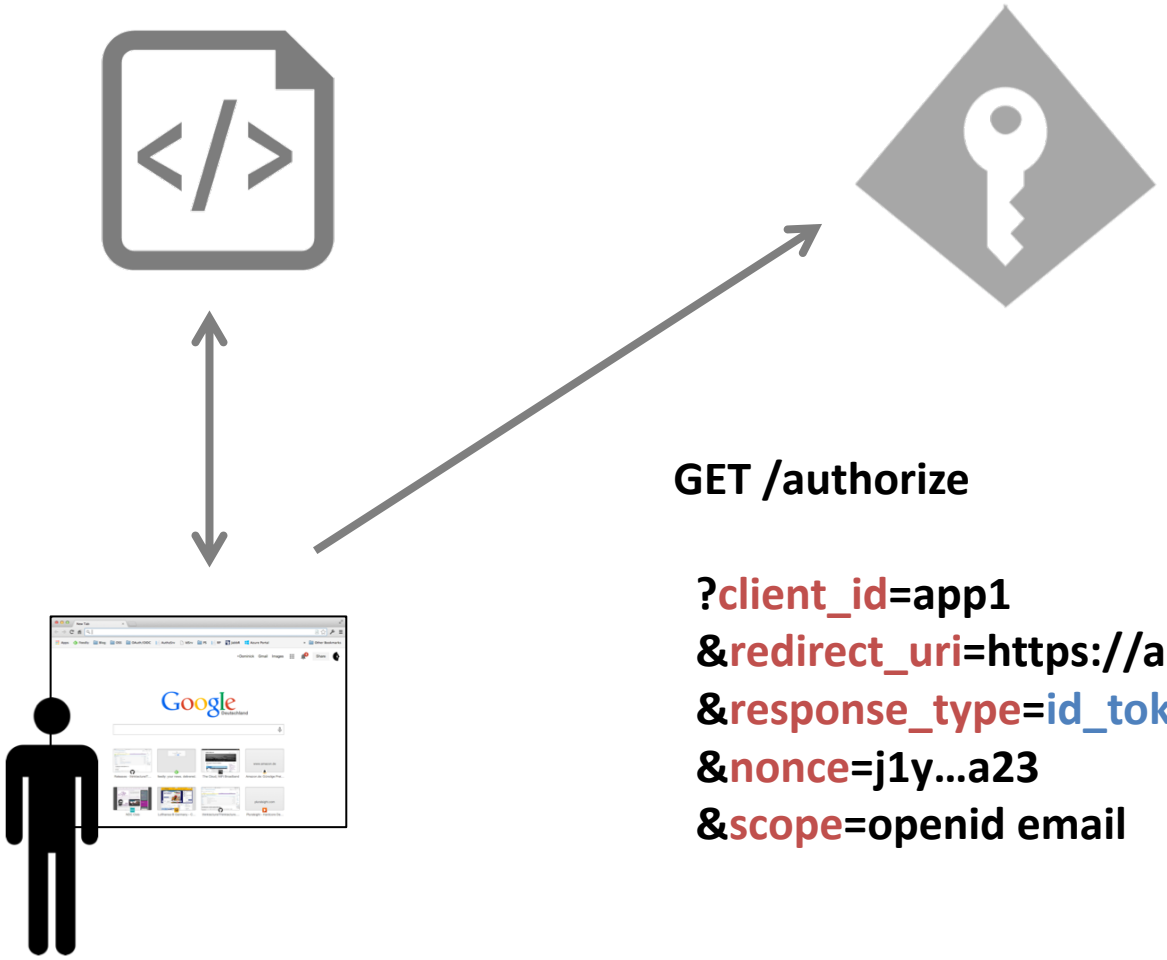
http://openid.net/connect/

Key OpenID Connect Features

- **Identity tokens**
  - signed protocol response
  - authentication metadata
  - helps mitigate various attacks
- **Discovery (metadata)**
- **Multiple response types**
  - allows requesting identity and access tokens in a single protocol interaction
- **Session management**

- **Interop**
  - https://openid.net/certification/

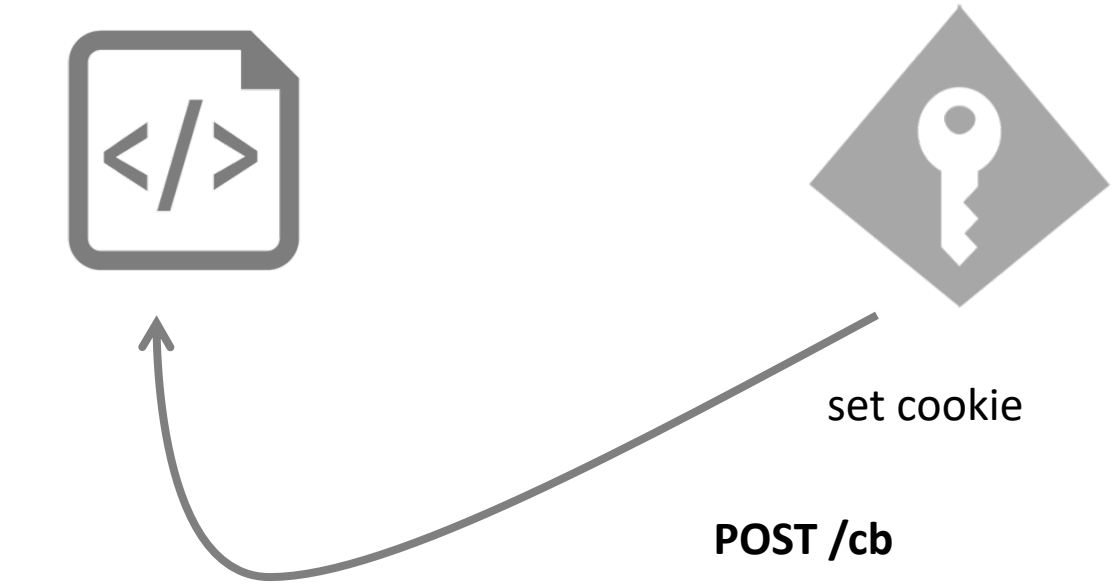
Authentication-only



GET /authorize

?client_id=app1
&redirect_uri=https://app.com/callback
&response_type=id_token
&nonce=j1y…a23
&scope=openid email

Response



POST /cb

```
<form action="https://app.com/cb">
   <input type="hidden"
          name="id_token"
          value="xjsj…aas" />
</form>

<script>document.forms[0].submit()</script>
```

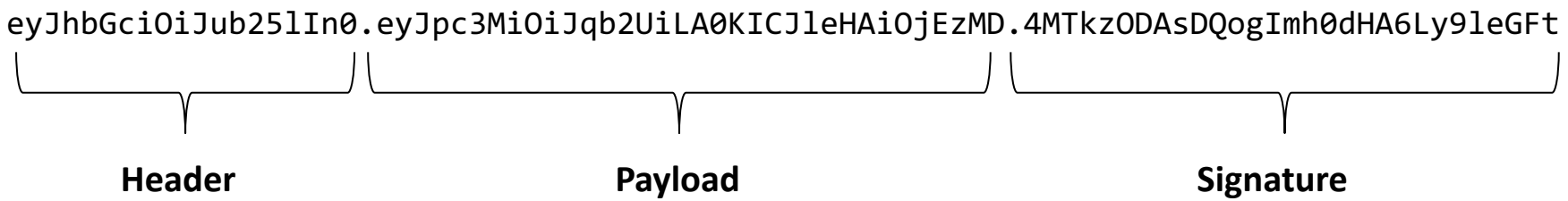
Identity Token

Header

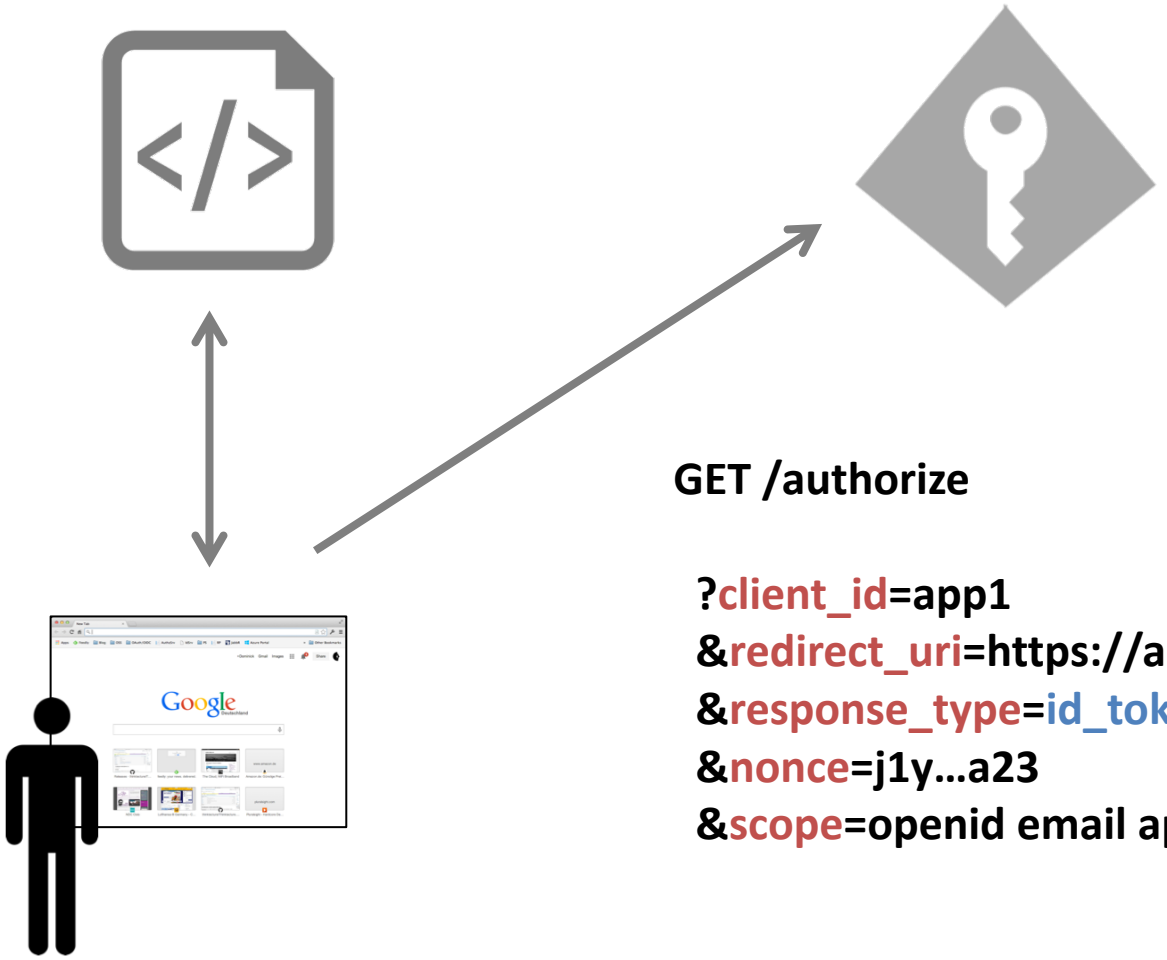
```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "mj399j…"
}
```

Payload

```
{
  "iss": "https://issuer",
  "exp": 1340819380,
  "iat": 1340818761,
  "aud": "app1",
  "nonce": "j1y…a23",
  "amr": [ "pwd" ],
  "auth_time": 12340819300

  "sub": "182jmm199",
  "name": "Alice",
}
```



Authentication and API Access



GET /authorize

?client_id=app1
&redirect_uri=https://app.com/callback
&response_type=id_token token
&nonce=j1y…a23
&scope=openid email api1 api2

Response

GET /callback.html

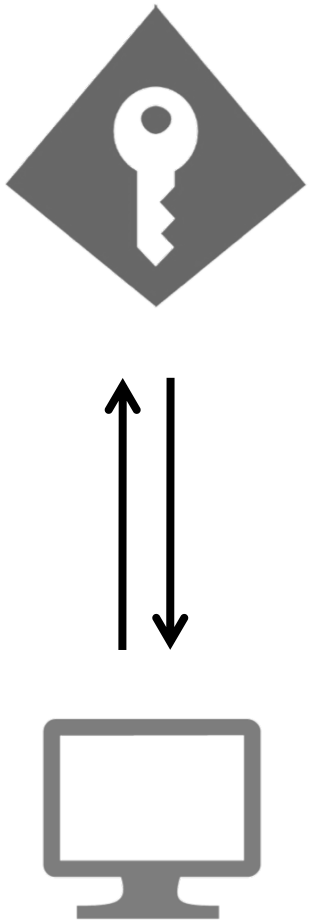
#id_token=x12f…zsz
&token=32x…133] **cryptographically linked**
&expires_in=3600
&token_type=Bearer

Session Management

- **Logout is hard – thus three specs**
  - JavaScript-based sessions
  - Front-channel notifications
  - Back-channel notifications

- **A full logout means**
  - logout from local client
  - logout from identity provider
  - logout from potential upstream identity provider
  - notify all other clients in same session

Front-Channel Notifications

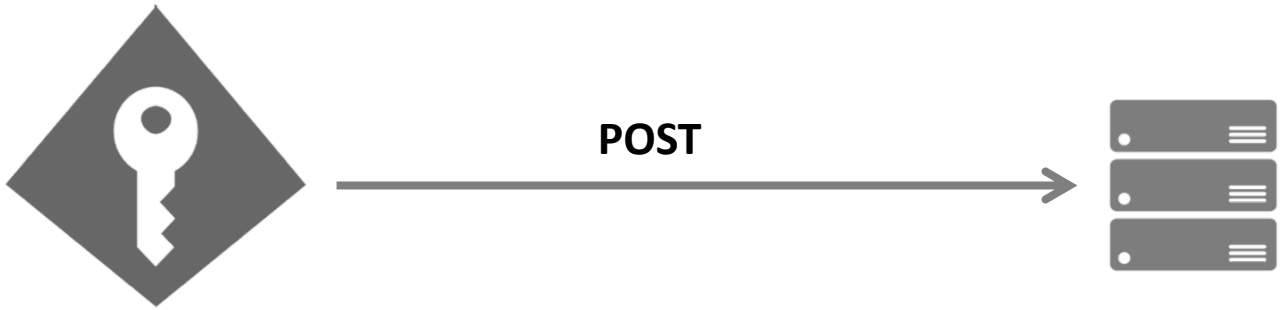


Client

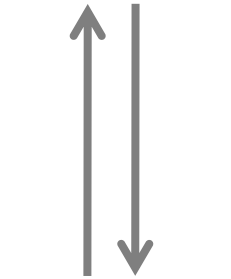
GET /end_session

```
<iframe style="visibility:hidden"
     src="https://client1/signout?sid=123">
</iframe>
<iframe class="visibility:hidden"
     src="https://client2/signout?sid=123">
</iframe>
<iframe class="visibility:hidden"
     src="https://client3/signout?sid=123">
</iframe>

<a href="https://client1">return</a>
```

Back-Channel Notifications



GET /end_session



Client

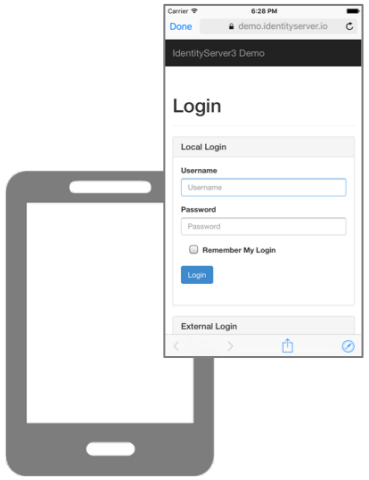
```
{
  "iss": "https://demo.identityserver.io",
  "sub": "248289761001",
  "aud": "client1",
  "iat": 1471566154,
  "jti": "bWJq",
  "sid": "8u09jejd099",
  "events": {
    "http://schemas.openid.net/event/backchannel-logout": {}
  }
}
```

The Public Client Problem

- **JavaScript applications**
  - no place to hide secrets
  - Content Security Policy (CSP) improves the situation
- **Native Mobile/Desktop Clients**
  - slightly better due to access to native APIs
  - targeted attacks in the past
- **OAuth 2.0 (and OpenID Connect) for native apps**
  - https://tools.ietf.org/search/rfc8252

Proof Key for Code Exchange (PKCE)

nonce = random_number
code_verifier = random_number
code_challenge = hash(code_verifier)

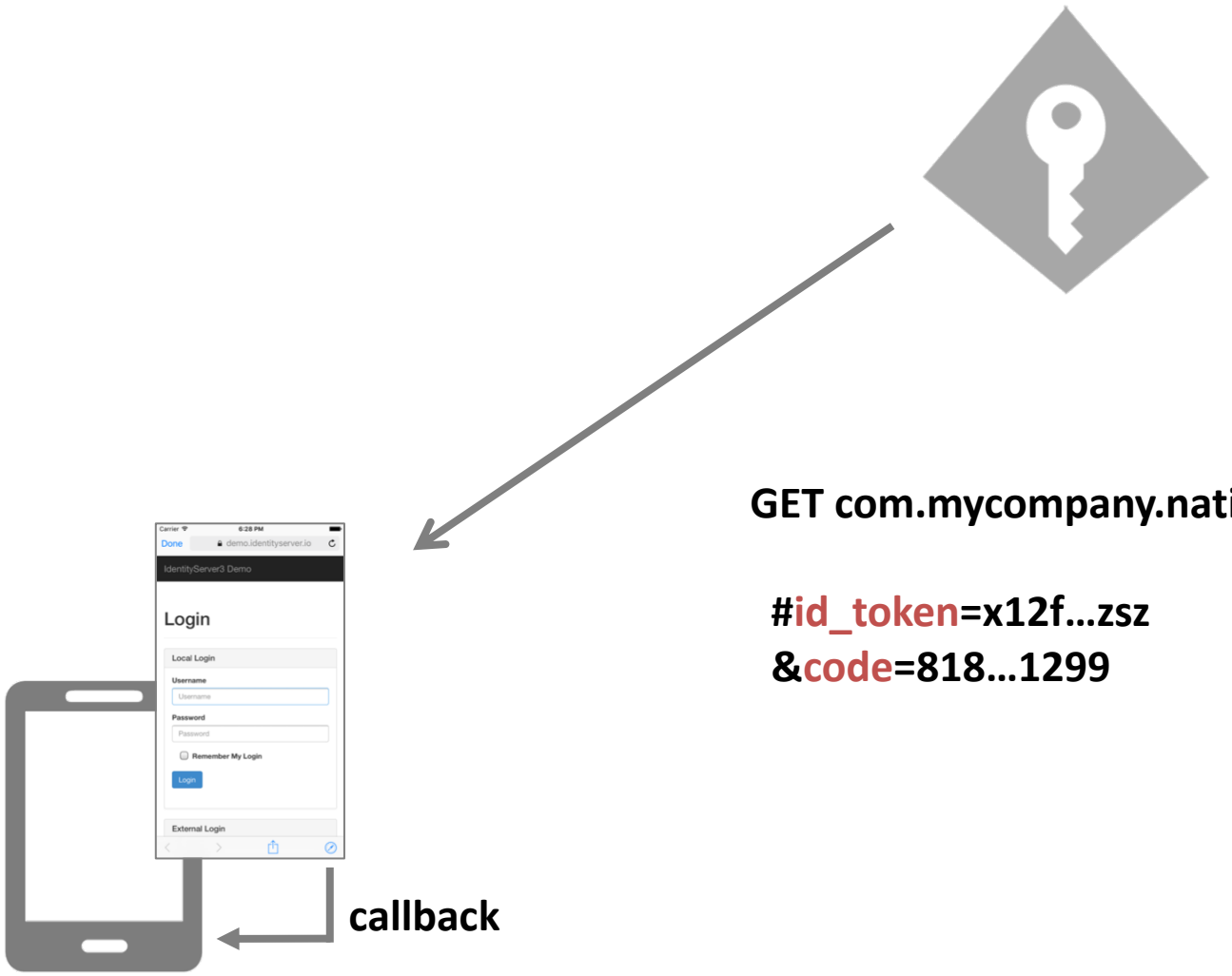


GET /authorize

?client_id=nativeapp
&scope=openid profile api1 api2 *offline_access*
&redirect_uri=com.mycompany.nativeapp://cb
&response_type=*code id_token*
&nonce=j1y…a23
&code_challenge=x929..1921

https://tools.ietf.org/html/rfc7636

Receiving the response

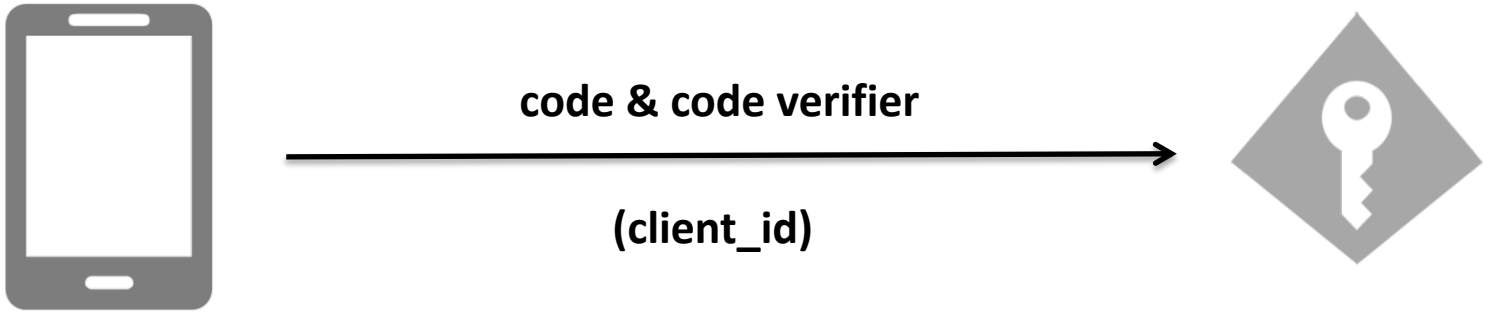


GET com.mycompany.nativeapp://cb

#id_token=x12f…zsz
&code=818…1299

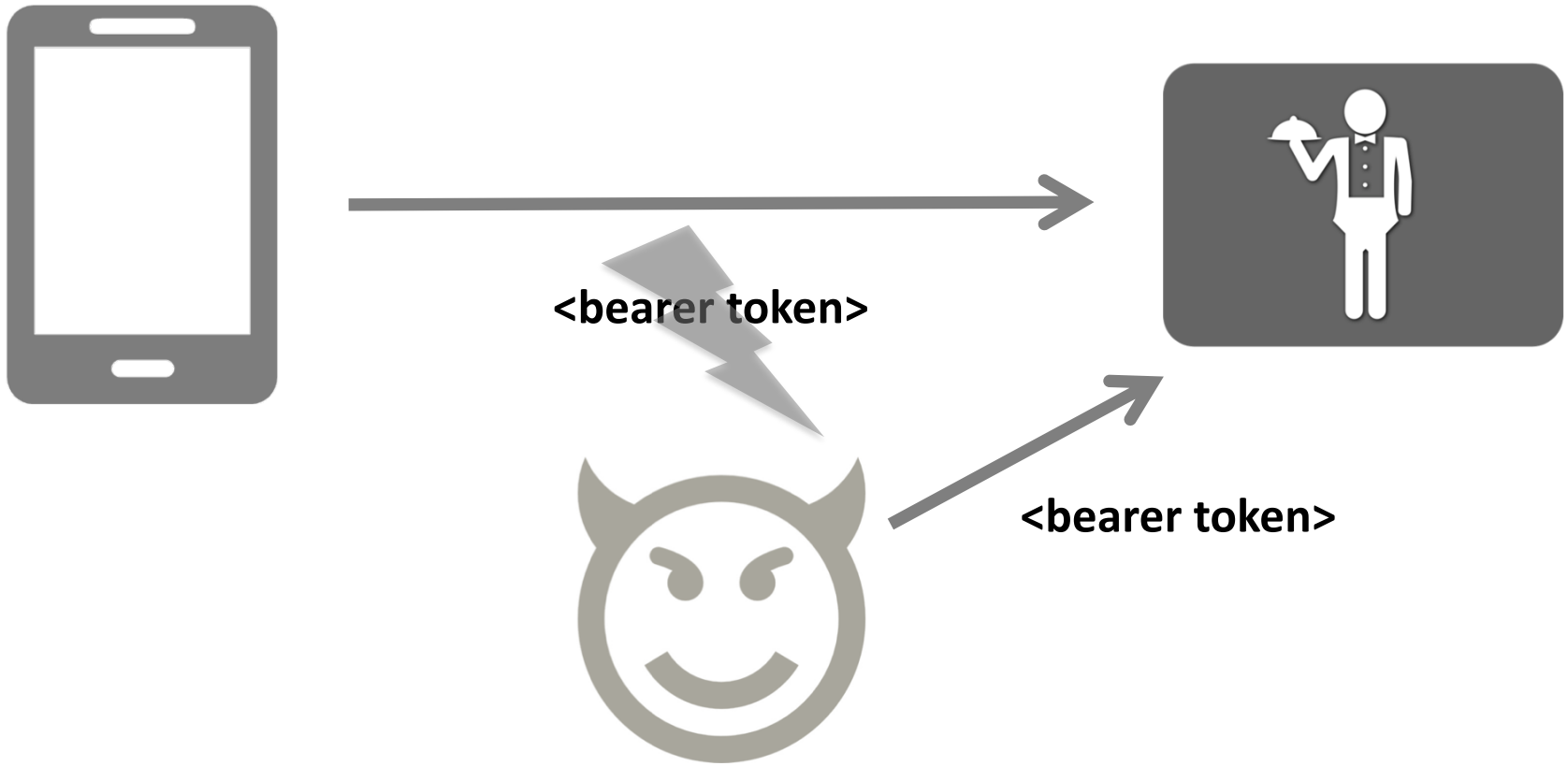
Requesting the access token

- **Exchange code for access token**
  - using client id and code verifier




```
{
  access_token: "xyz…123",
  refresh_token: "dxy…103"
  expires_in: 3600,
  token_type: "Bearer"
}
```

The Token Problem (Part 2): Bearer Tokens



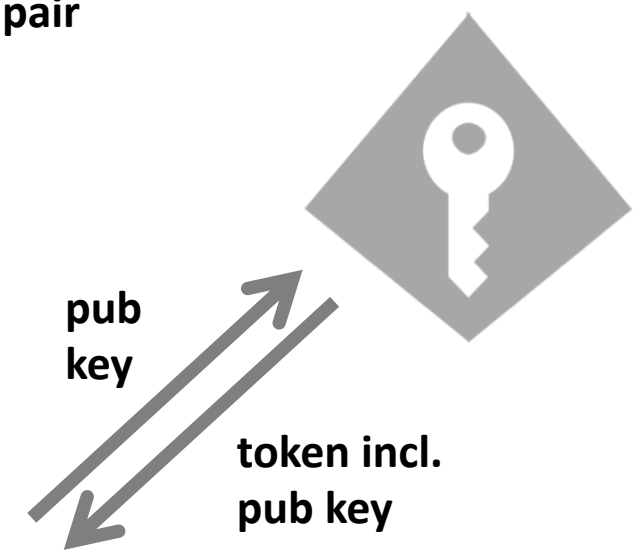
General Approach

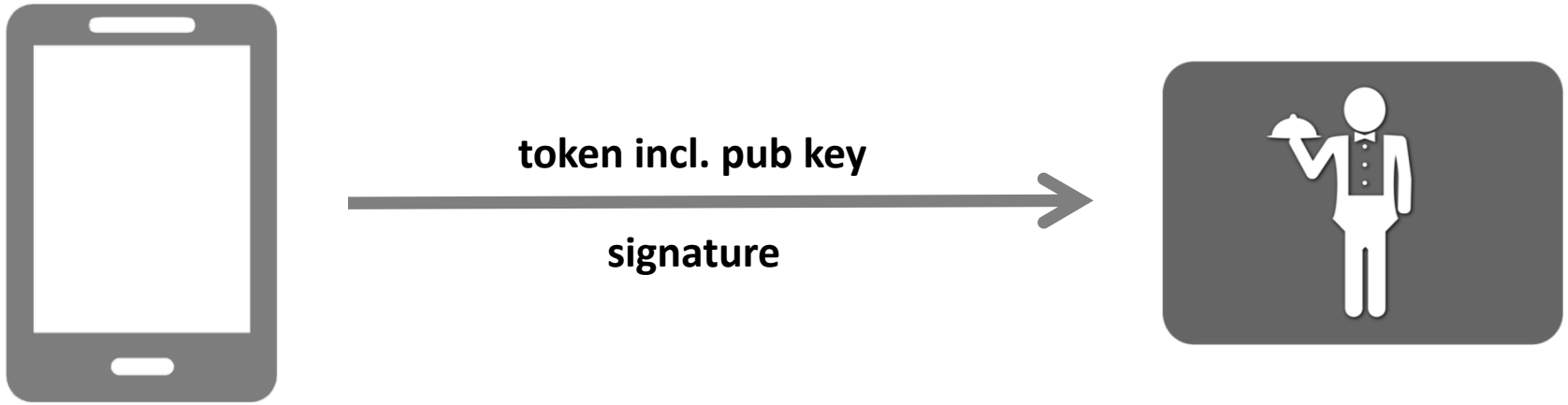
1) client generates pub/priv key pair

2) sends public key to STS during token request

3) STS embeds pub key in access token

4) client uses private key to sign HTTP request





5) API validates access token

6) extracts proof key & validates the HTTP signature

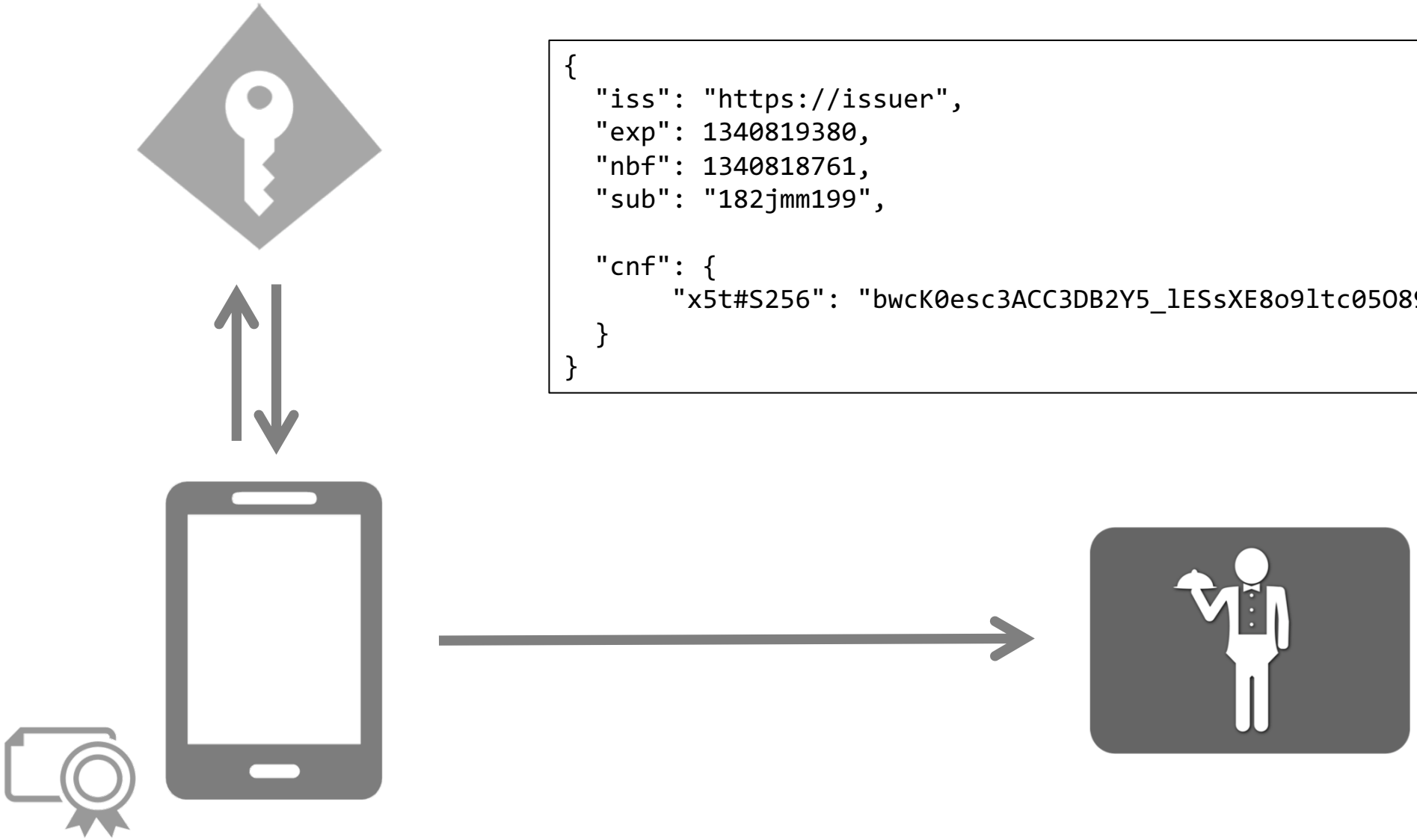
Current Opinion

- **Standardization of *cnf* (confirmation) claim**

```
{
  "cnf": "JSON web key",
}
```

- **Specialized scenarios**
  - Mutual TLS Profile for OAuth 2.0
- **Consumer scenarios**
  - HTTPS Token Binding for OAuth 2 and OpenID Connect

Example: Mutual TLS



```
{
  "iss": "https://issuer",
  "exp": 1340819380,
  "nbf": 1340818761,
  "sub": "182jmm199",

  "cnf": {
       "x5t#S256": "bwcK0esc3ACC3DB2Y5_lESsXE8o9ltc05O89jdN-dg2"
  }
}
```

Token Binding

- **IETF**
  - https://tools.ietf.org/html/draft-ietf-tokbind-protocol
  - https://tools.ietf.org/html/draft-ietf-tokbind-https

Abstract

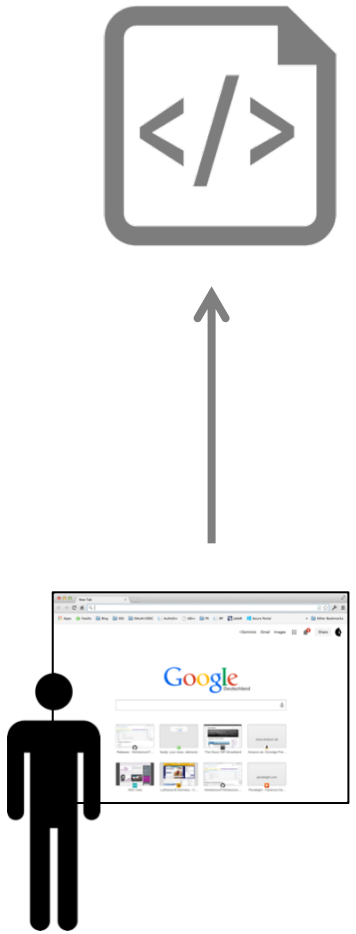
This document specifies Version 1.0 of the Token Binding protocol. The Token Binding protocol allows client/server applications to create long-lived, uniquely identifiable TLS bindings spanning multiple TLS sessions and connections. Applications are then enabled to cryptographically bind security tokens to the TLS layer, preventing token export and replay attacks. To protect privacy, the Token Binding identifiers are only conveyed over TLS and can be reset by the user at any time.

Using Token Binding for PoP

- **OpenID Connect Token Bound Authentication**
  - https://openid.net/specs/openid-connect-token-bound-authentication-1_0.html

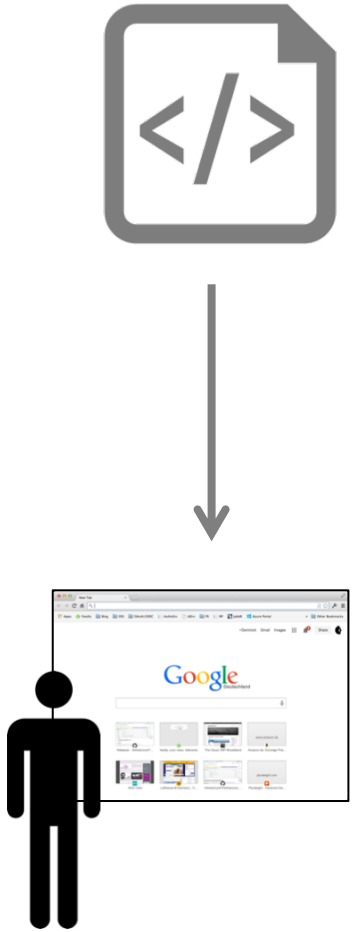
- **OAuth 2.0 Token Binding**
  - https://tools.ietf.org/wg/oauth/draft-ietf-oauth-token-binding/

Example: Request to Client




```
GET / HTTP/1.1

Host: client.example.io
Sec-Token-Binding: AIkAAgBBQKzyIrmcYKTZfFJv …1_610h0h-IX-
```

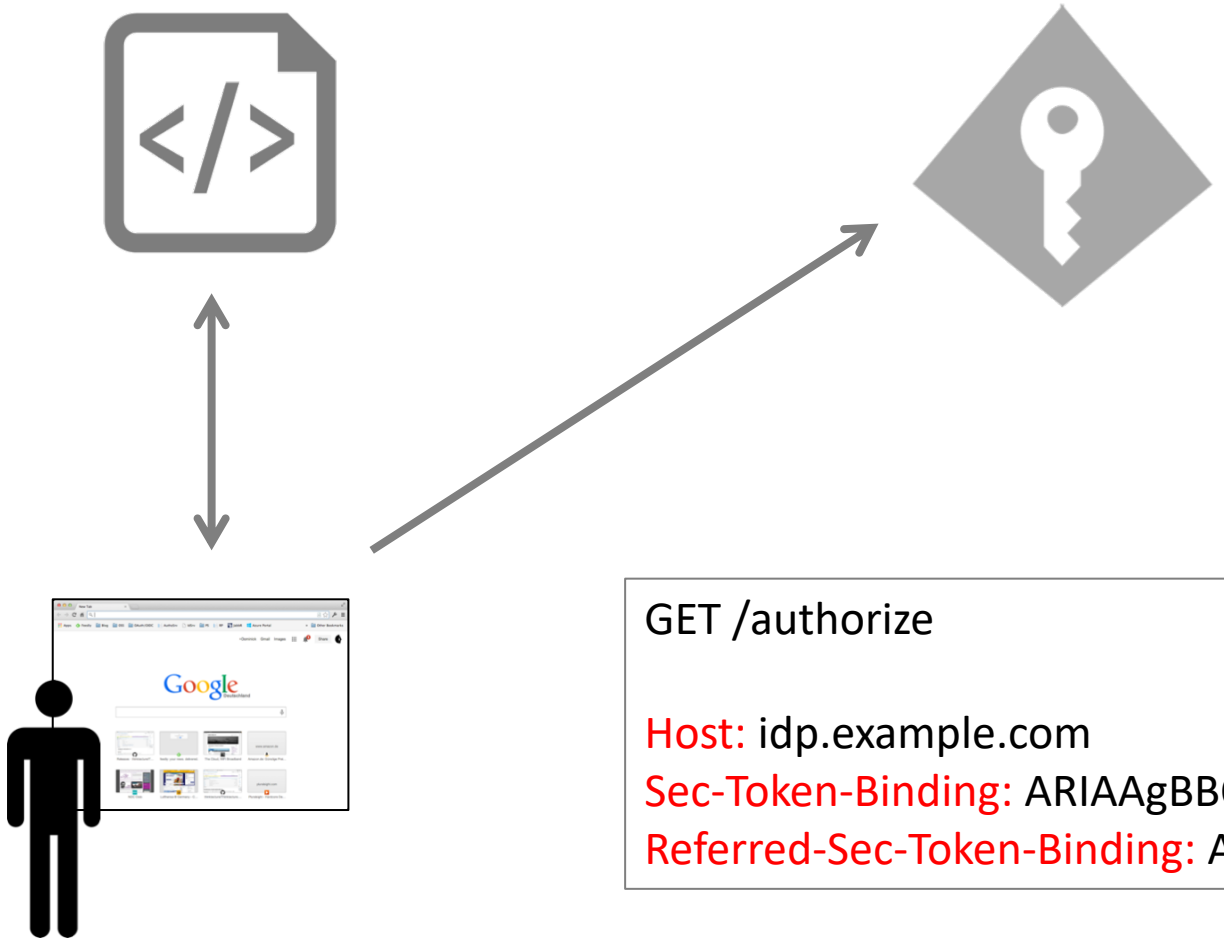
Example: Redirect to OpenID Connect Provider



HTTP/1.1 302

Location: https://idp.example.io/authorize?....
Include-Referred-Token-Binding-ID: true

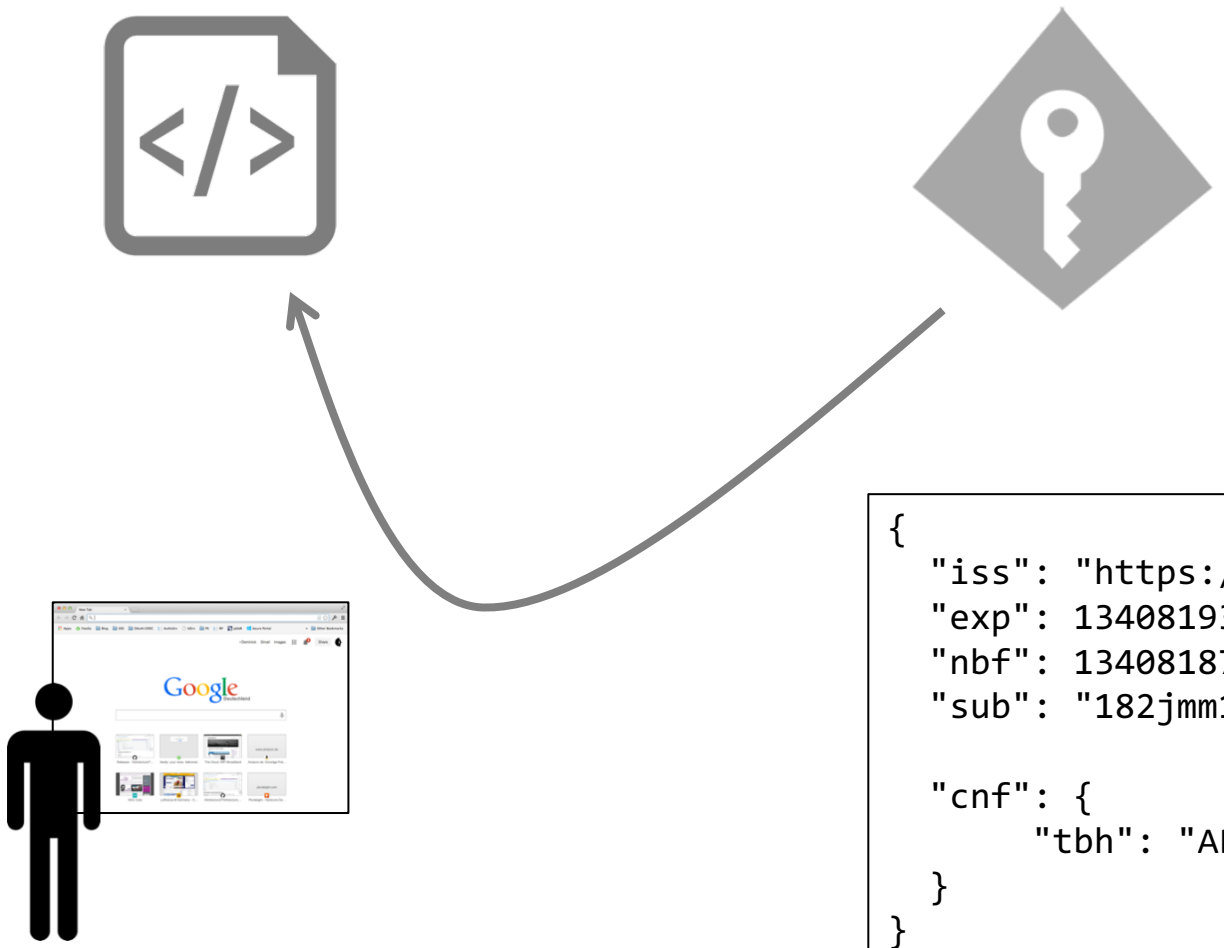
Example: Authentication Request



GET /authorize

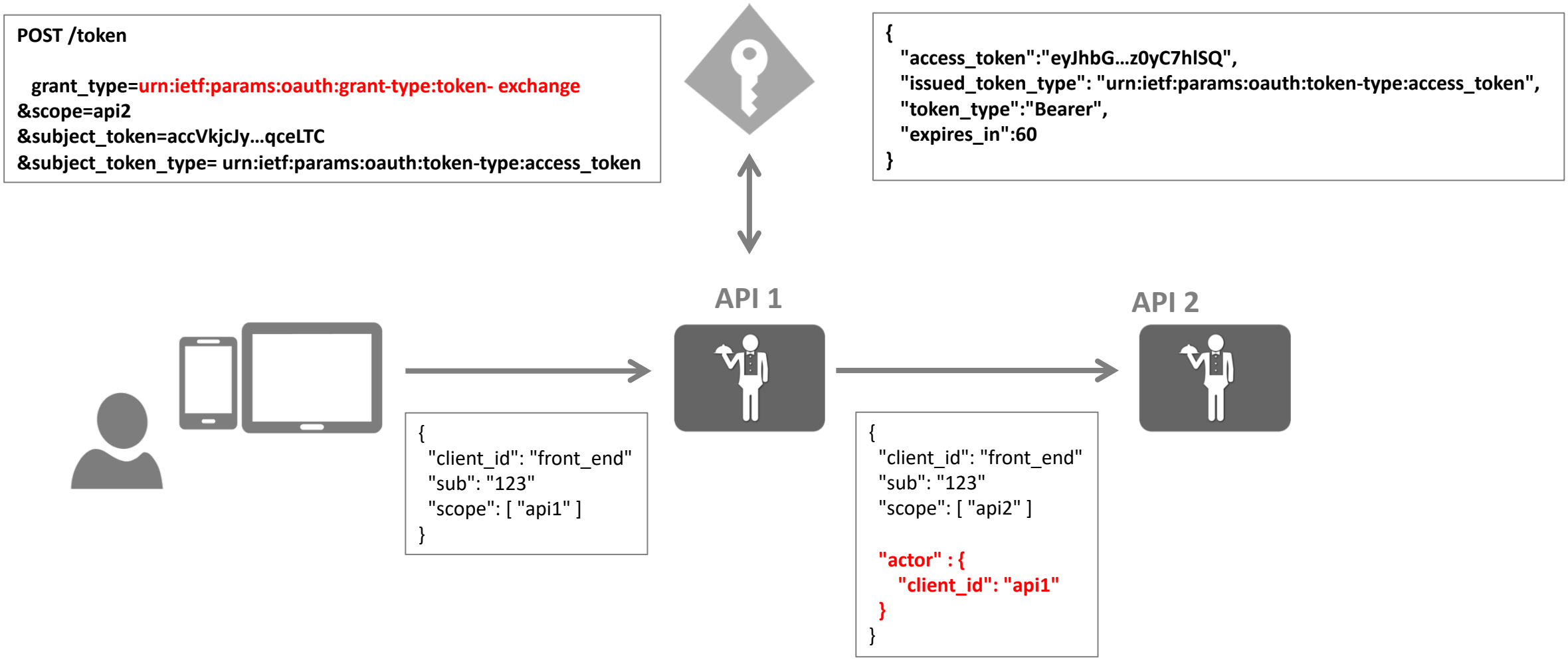
Host: idp.example.com
Sec-Token-Binding: ARIAAgBBQC…fsI1D1sTq5mvT_2H_dihNIvu
Referred-Sec-Token-Binding: ARIAAgBBQC…fsI1D1sTq5mvNIvu

Example: Authentication Response



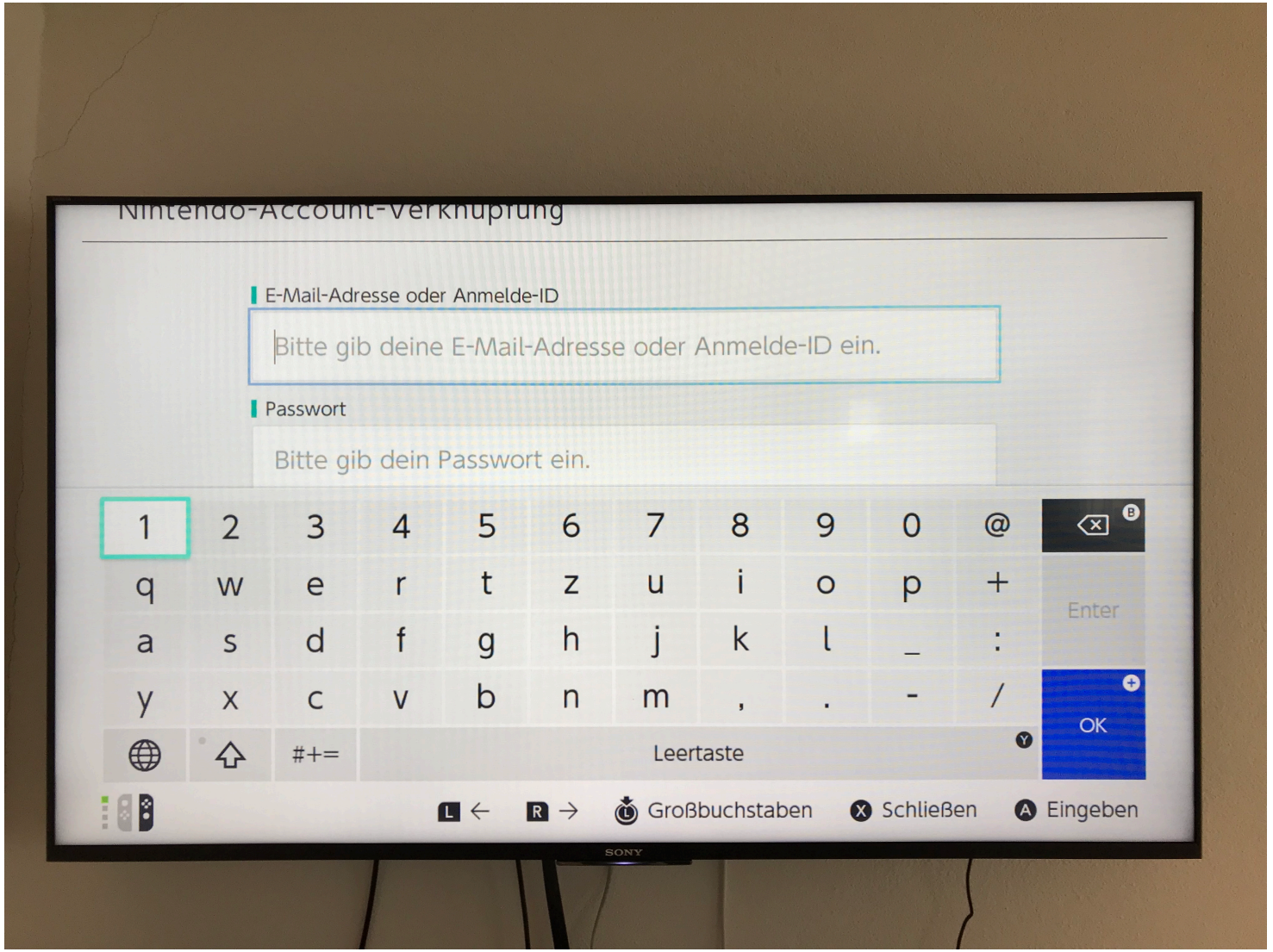

```
{
  "iss": "https://issuer",
  "exp": 1340819380,
  "nbf": 1340818761,
  "sub": "182jmm199",

  "cnf": {
       "tbh": "ARIAAgBBQC…fsI1D1sTq5mvNIvu"
  }
}
```

Multi-Hop Delegation

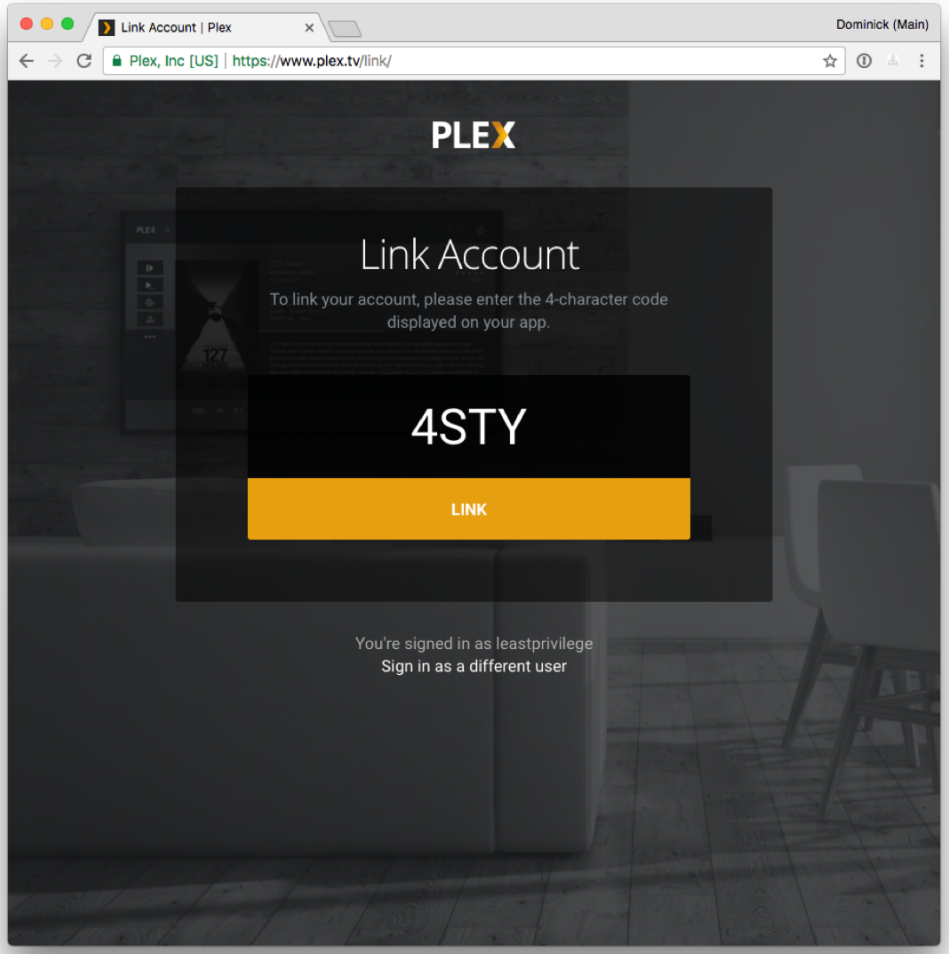


https://tools.ietf.org/wg/oauth/draft-ietf-oauth-token-exchange/

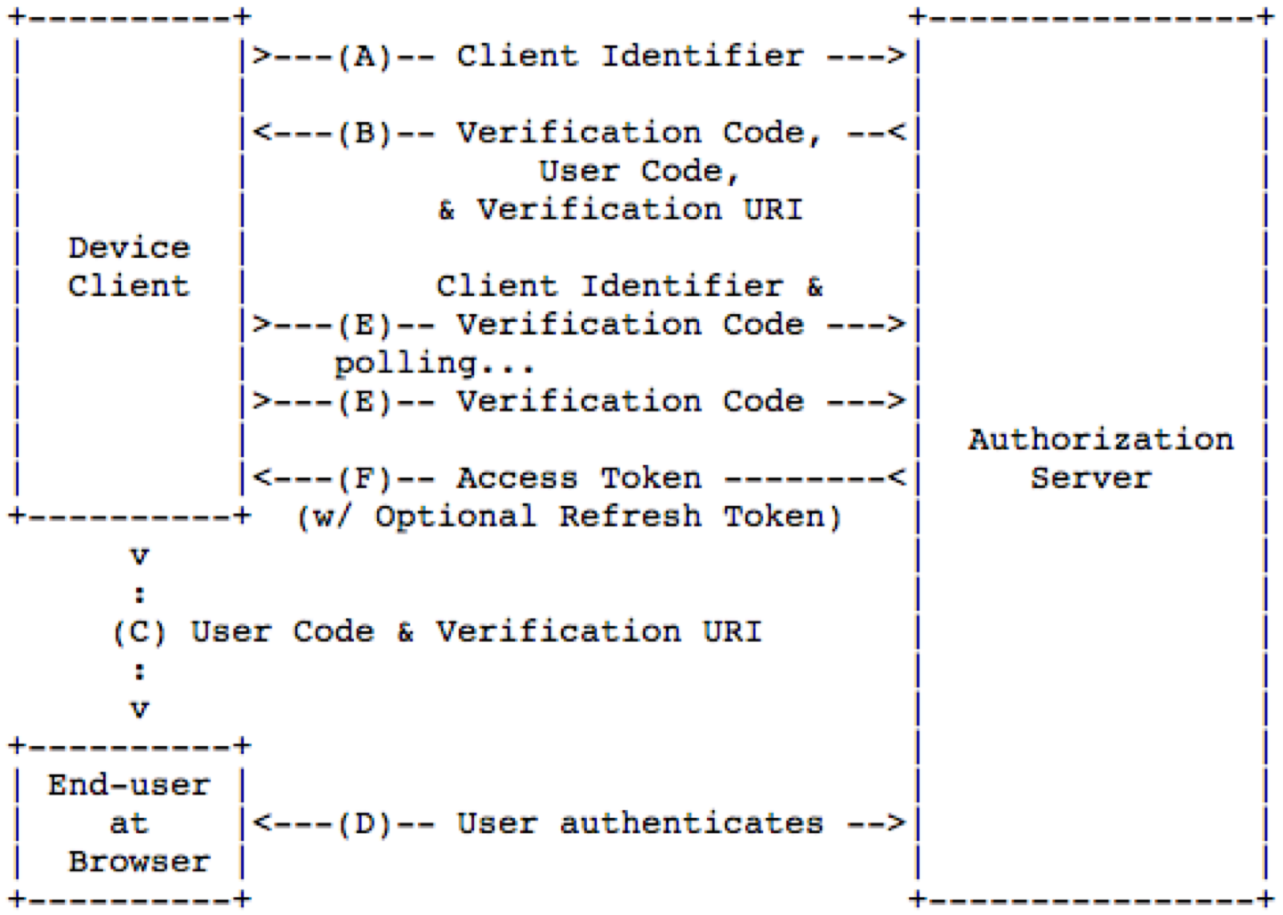
"Constrained Input Devices"



OAuth 2.0 Device Flow
for Browserless and Input Constrained Devices



…or in ASCII Art



What's going on with OpenID Connect?

- **Some important working groups (IMO)**
  - Enhanced Authentication
  - Financial APIs
  - Federation

- **Some OIDC features back-ported to OAuth 2.0**
  - JWT secured authorization requests
  - discovery
  - client management

IoT

- **Concise Binary Object Representation (CBOR)**
  - https://tools.ietf.org/html/rfc7049
  - http://cbor.io/
- **CBOR Object Signing and Encryption (COSE)**
  - https://tools.ietf.org/html/rfc8152
- **CBOR Web Token (CWT)**
  - https://tools.ietf.org/html/draft-ietf-ace-cbor-web-token-10

> The suggested pronunciation of CWT is the same as the English word "cot".

Summary

- **Easier than WS* ?**
  - well, at least it's not XML
  - no easy solutions for hard problems

- **Pick the features you need**
  - if you require interop, a (final) spec would be good

- **Good coverage of base specs in standard libraries/products**
  - OpenID Connect only spec so far with conformance tests

Thanks!