Security in Internet Applications
(Specific protocols)

2024/25 Q2

Jaime Delgado

DAC - UPC

DMAG
DISTRIBUTED MULTIMEDIA APPLICATIONS GROUP

Security in Internet Applications

- Security in application layer protocols
- XML and security
- Specific security protocols for the Web

Security in Internet Applications

- Security in application layer protocols
- XML and security
- **Specific security protocols for the Web**

Specific security web protocols

Specific security protocols:

SAML

OAuth

OpenID Connect

JSON Web Tokens (JWT)

SAML (Security Assertion Markup Language)

- v2.0 March 2005. OASIS
  (Organization for the Advancement of Structured Information Standards)
  https://www.oasis-open.org/standards#samlv2.0
- Interchange of authentication and authorization data
- Based in XML
- Security tokens: “Assertions” (user info)
- Interchange between an “identity provider” and a “service provider”
- Useful for authentication and authorization in web environments

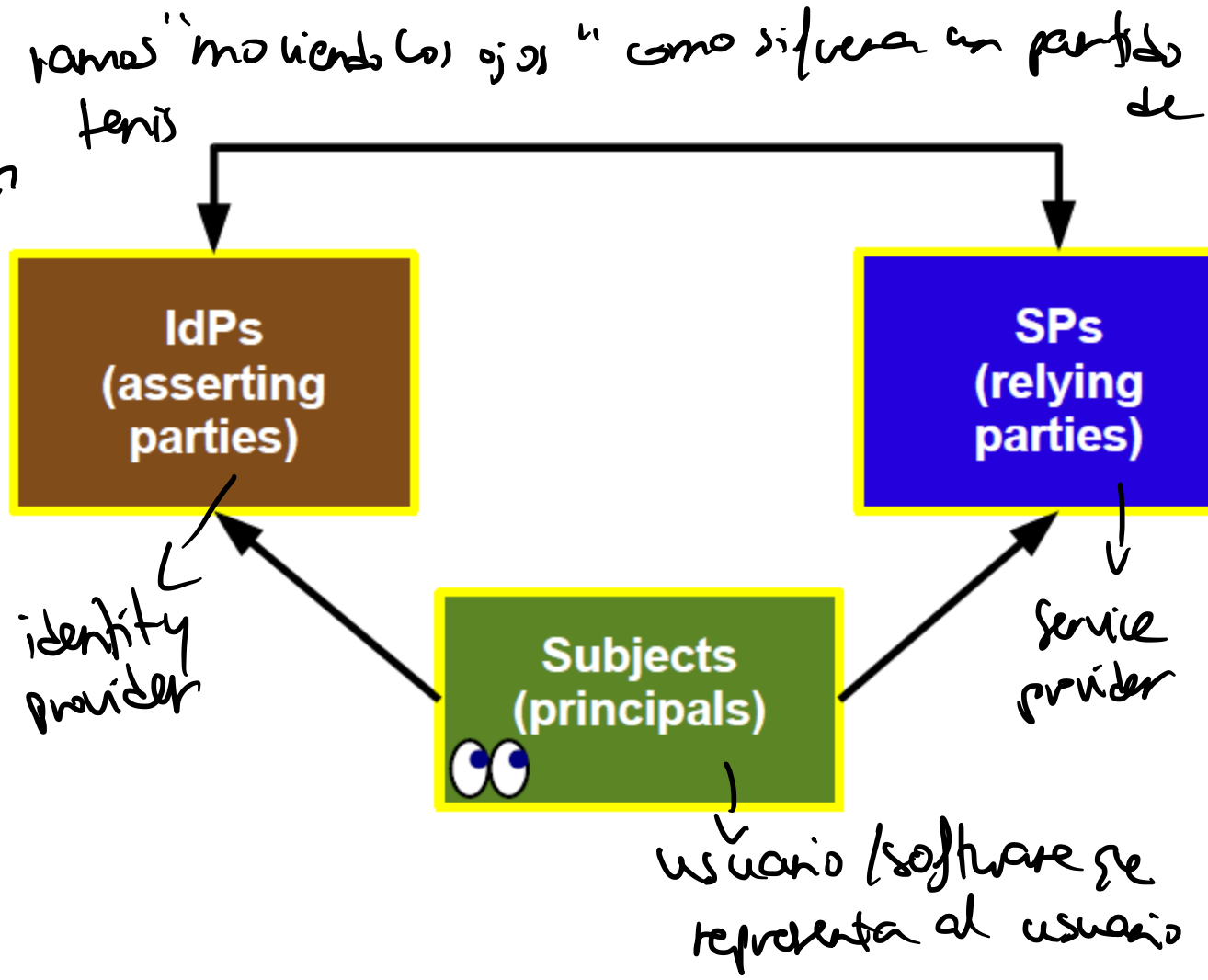
Protocolo para autorizar y autenticar

SAML (Security Assertion Markup Language)

- v2.0 March 2005. OASIS
  (Organization for the Advancement of Structured Information Standards)
  https://www.oasis-open.org/standards#samlv2.0
- Interchange of authentication and authorization data
- Based in XML
- Security tokens: “Assertions” (user info)
- Interchange between an “**identity provider**” and a “**service provider**”
- Useful for authentication and authorization in web environments

How these entities interrelate

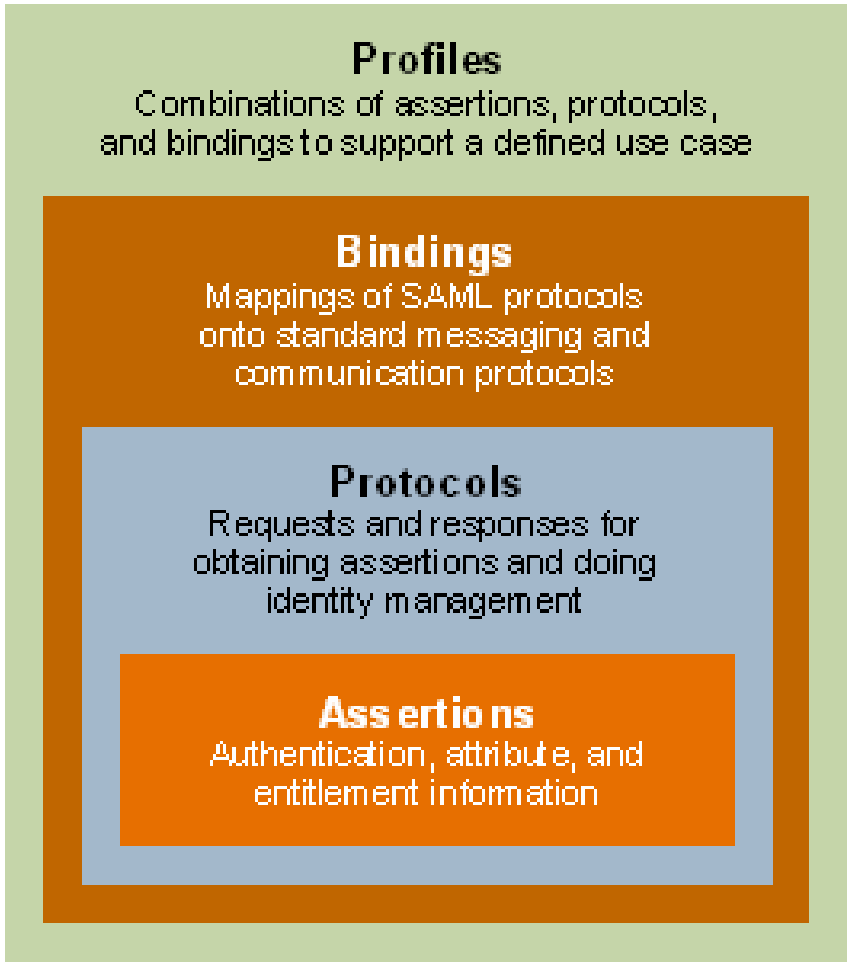
- Most of the SAML and ID-FF* use cases are eyeball-oriented
- But some back-channel (SOAP and other) communication takes place in service of this



* ID-FF (origin of SAML): Liberty Identity Federation Framework

SAML basic concepts

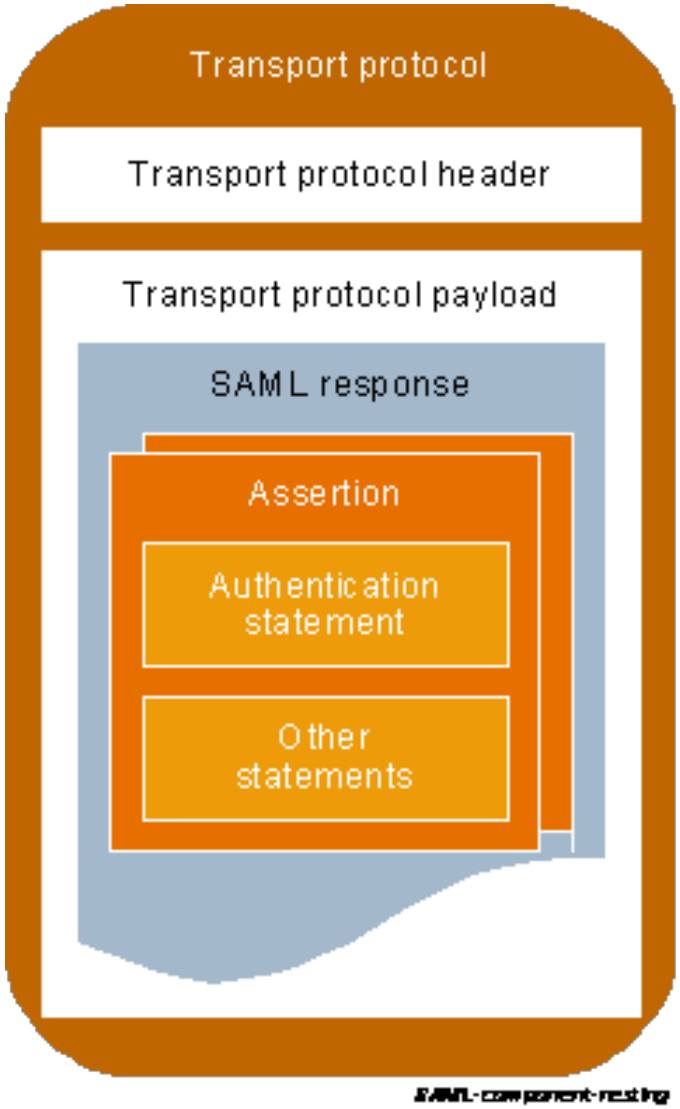
Se puede implementar SAML sobre lo que queramos, pero normalmente lo hacemos sobre HTTP.



Authentication Context
Detailed data on types and strengths of authentication

Metadata
Configuration data for identity and service providers

SAML components

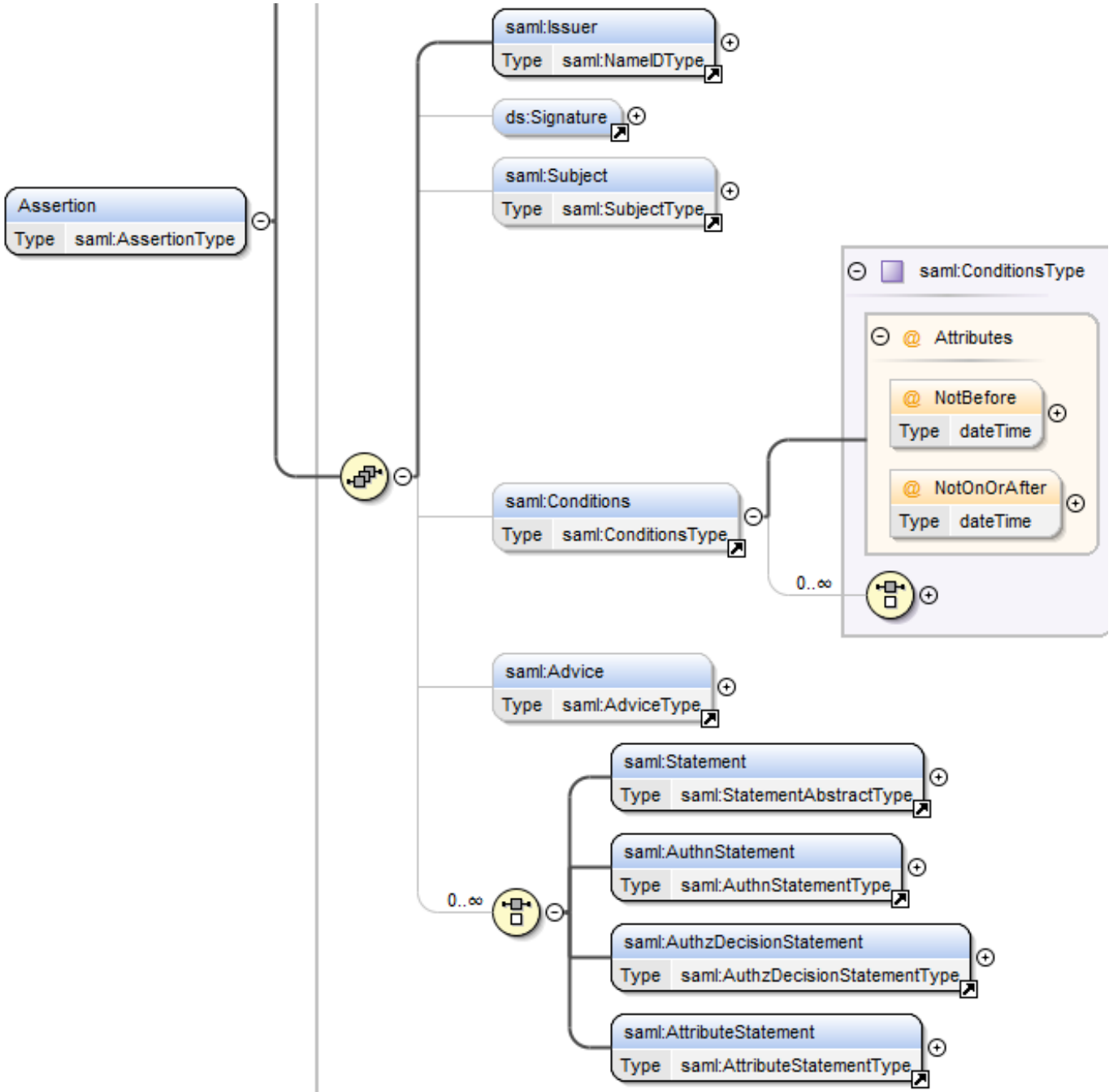


SAML assertion structure

```
 1: <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
 2:   Version="2.0"
 3:   IssueInstant="2005-01-31T12:00:00Z">
 4:   <saml:Issuer Format=urn:oasis:names:SAML:2.0:nameid-format:entity>
 5:     http://idp.example.org
 6:   </saml:Issuer>
 7:   <saml:Subject>
 8:     <saml:NameID
 9:       Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
10:         j.doe@example.com
11:     </saml:NameID>
12:   </saml:Subject>
13:   <saml:Conditions
14:     NotBefore="2005-01-31T12:00:00Z"
15:     NotOnOrAfter="2005-01-31T12:10:00Z">
16:   </saml:Conditions>
17:   <saml:AuthnStatement
18:     AuthnInstant="2005-01-31T12:00:00Z" SessionIndex="67775277772">
19:     <saml:AuthnContext>
20:       <saml:AuthnContextClassRef>
21:         urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
22:       </saml:AuthnContextClassRef>
23:     </saml:AuthnContext>
24:   </saml:AuthnStatement>
25: </saml:Assertion>
```

Figure 6: Assertion with Subject, Conditions, and Authentication Statement

SAML: Assertion structure



SAML attribute statement structure

```
 1: <saml:AttributeStatement>
 2:    <saml:Attribute
 3:       xmlns:x500="urn:oasis:names:tc:SAML:2.0:profiles:attribute:X500"
 4:       NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
 5:       Name="urn:oid:2.5.4.42"
 6:       FriendlyName="givenName">
 7:       <saml:AttributeValue xsi:type="xs:string"
 8:         x500:Encoding="LDAP">John</saml:AttributeValue>
 9:    </saml:Attribute>
10:    <saml:Attribute
11:      NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic"
12:      Name="LastName">
13:      <saml:AttributeValue
14:        xsi:type="xs:string">Doe</saml:AttributeValue>
15:    </saml:Attribute>
16:    <saml:Attribute
17:      NameFormat="http://smithco.com/attr-formats"
18:      Name="CreditLimit">
19:      xmlns:smithco="http://www.smithco.com/smithco-schema.xsd"
20:      <saml:AttributeValue xsi:type="smithco:type">
21:        <smithco:amount currency="USD">500.00</smithco:amount>
22:      </saml:AttributeValue>
23:    </saml:Attribute>
24: </saml:AttributeStatement>
```

Figure 7: Attribute Statement

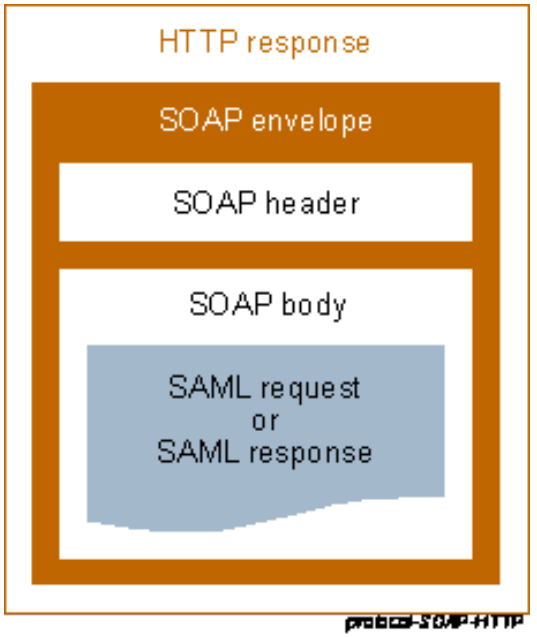
SAML: SOAP binding example

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <env:Envelope
3.    xmlns:env="http://www.w3.org/2003/05/soap/envelope/">
4.    <env:Body>
5.      <samlp:AttributeQuery
6.        xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
7.        xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
8.        ID="aaf23196-1773-2113-474a-fe114412ab72"
9.        Version="2.0"
10.       IssueInstant="2006-07-17T20:31:40Z">
11.       <saml:Issuer>http://example.sp.com</saml:Issuer>
12.       <saml:Subject>
13.           <saml:NameID
14.           Format="urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName">
15.           C=US, O=NCSA-TEST, OU=User, CN=trscavo@uiuc.edu
16.           </saml:NameID>
17.       </saml:Subject>
18.       <saml:Attribute
19.           NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
20.           Name="urn:oid:2.5.4.42"
21.           FriendlyName="givenName">
22.       </saml:Attribute>
23.     </samlp:AttributeQuery>
24.   </env:Body>
25. </env:Envelope>
```

Figure 9: Attribute Query in SOAP Envelope

```
 1: <?xml version="1.0" encoding="UTF-8"?>
 2: <env:Envelope  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
 3:   <env:Body>
 4:     <samlp:Response
 5:       xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
 6:       xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
 7:       Version="2.0"
 8:       ID="i92f8b5230dc04d73e93095719d191915fdc67d5e"
 9:       IssueInstant="2006-07-17T20:31:41Z"
10:       InResponseTo="aaf23196-1773-2113-474a-fe114412ab72 ">
11:       <saml:Issuer>http://idp.example.org</saml:Issuer>
12:       <samlp:Status>
13:         <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
14:       </samlp:Status>
15:                   ...SAML assertion...
16:     </samlp:Response>
17:   </env:Body>
18: </env:Envelope>
```

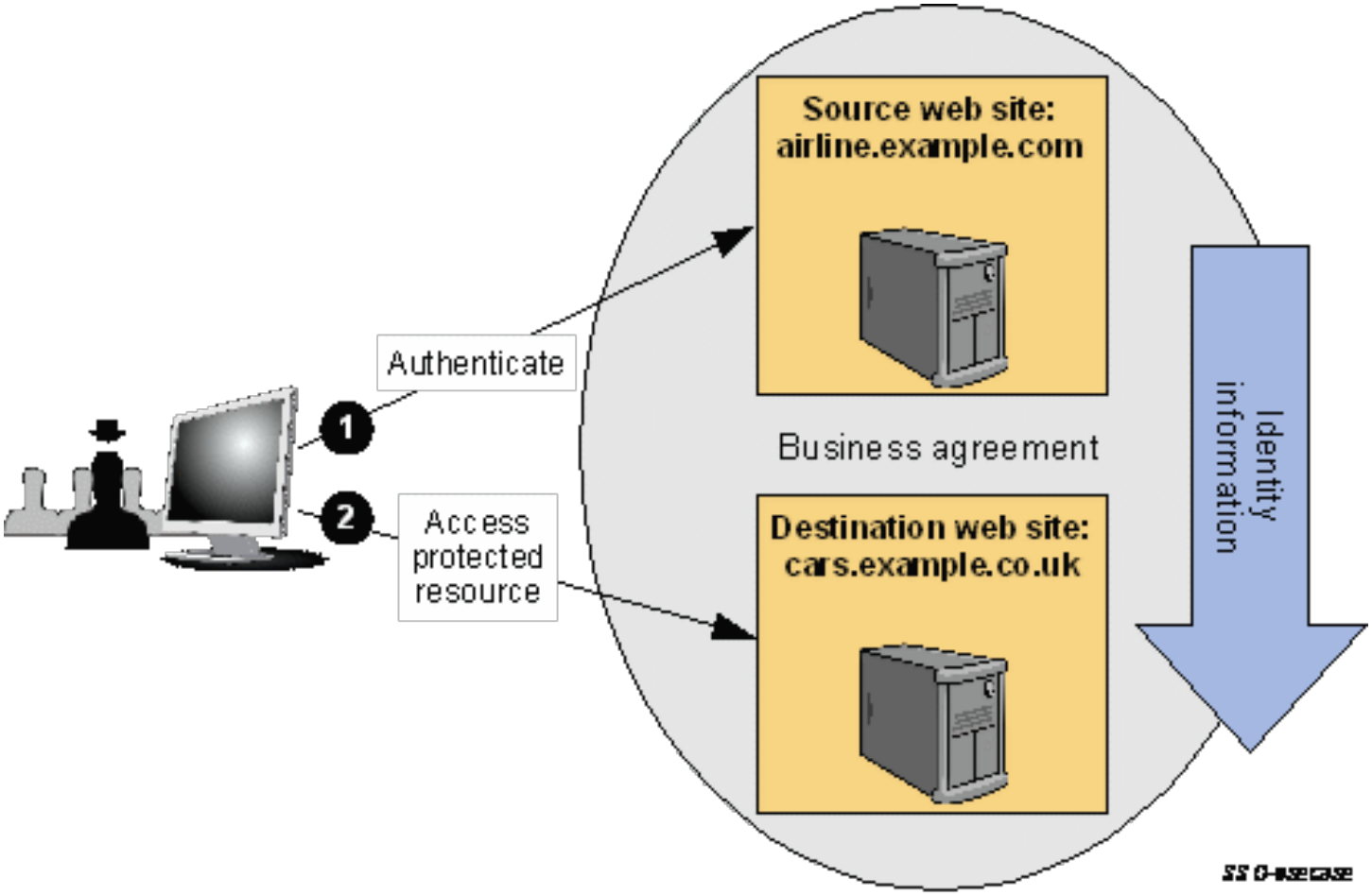
Figure 10: Response in SOAP Envelope



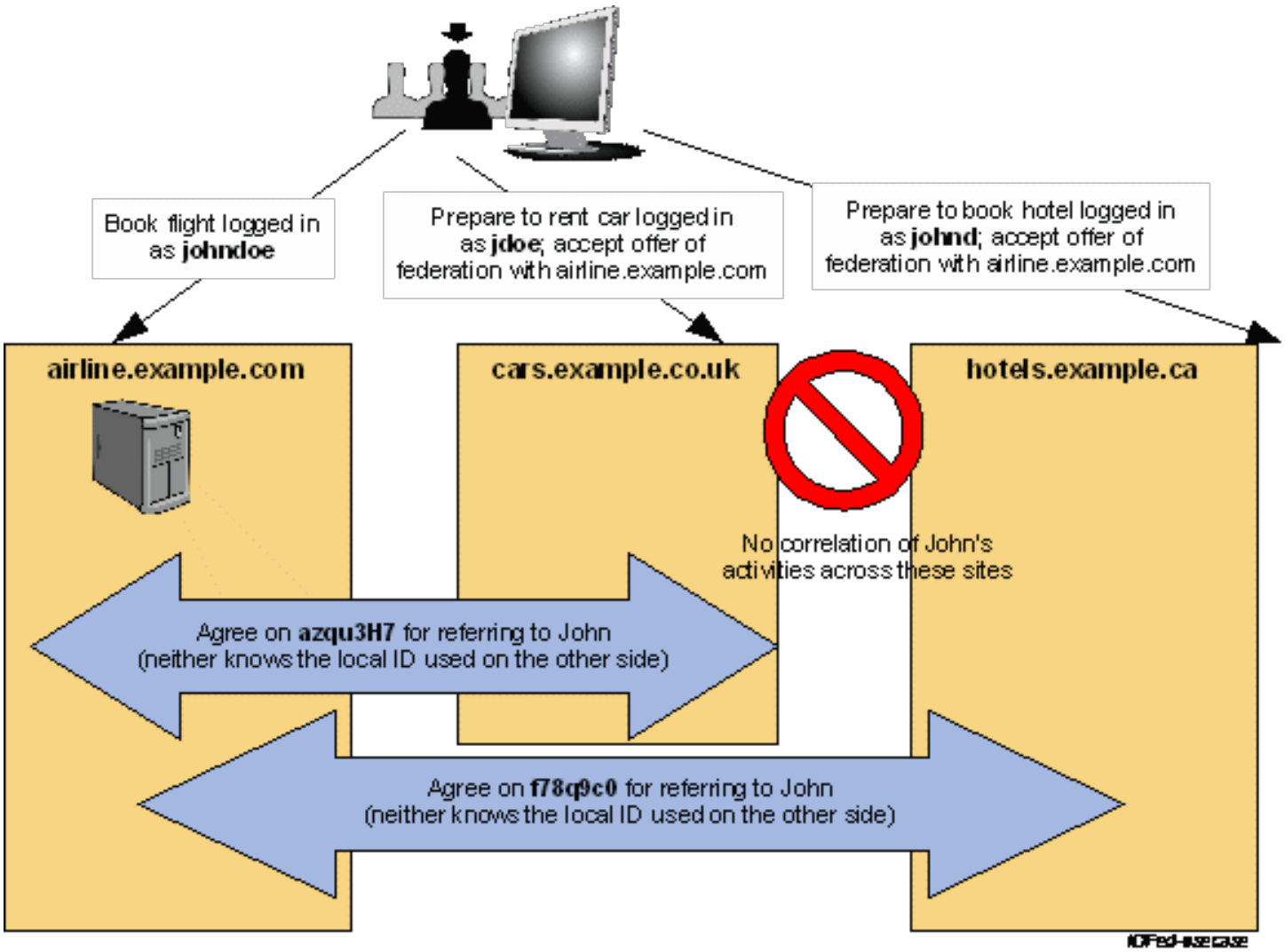
SAML: Examples of uses

- **Single sign-on**
  - Standard browsers
  - Enhanced HTTP clients
- **Identity federation**
  - Using a well-known name or attribute
  - For anonymous users by means of attributes
  - Using a privacy-preserving pseudonym

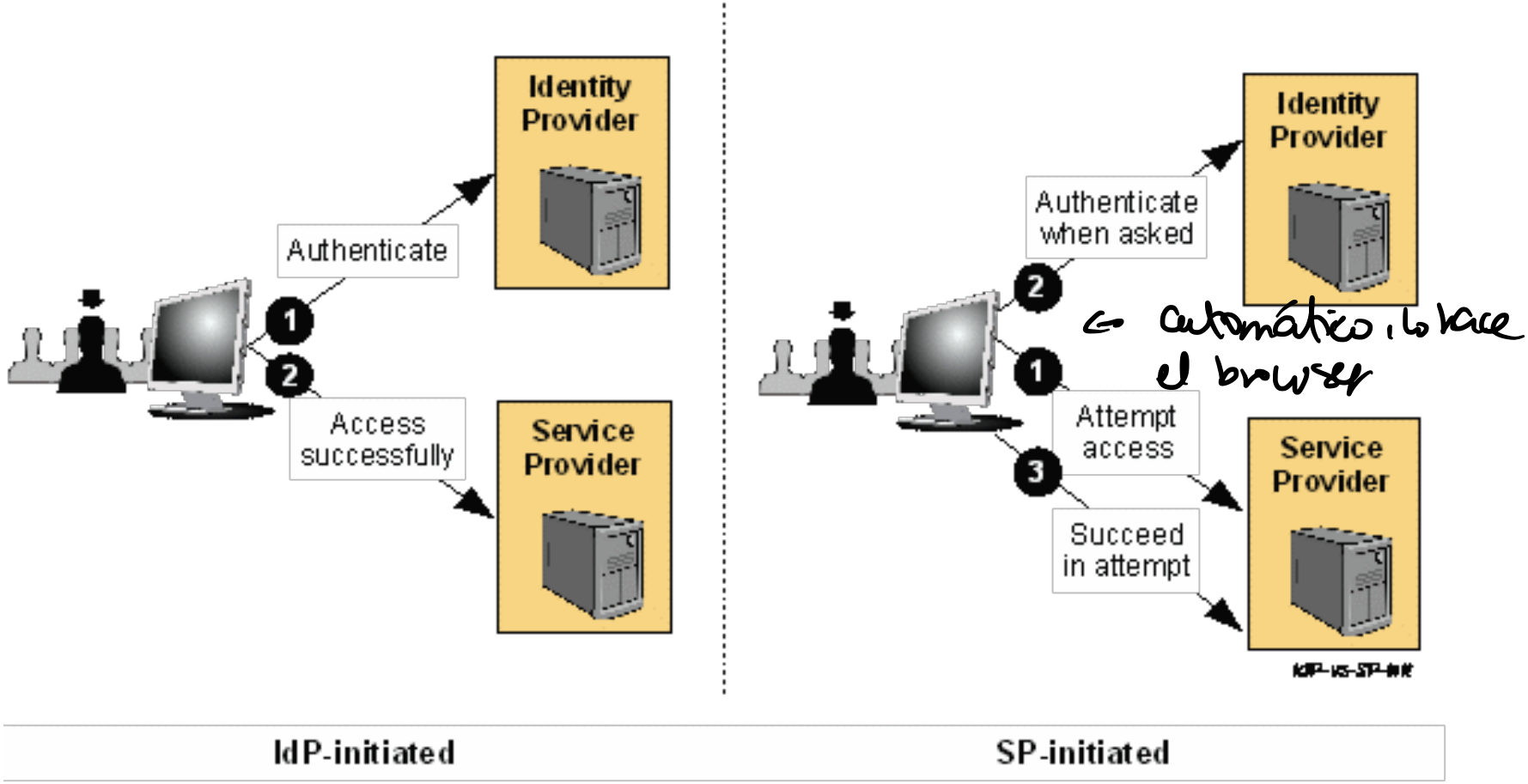
Single Sign-On (SSO) use case



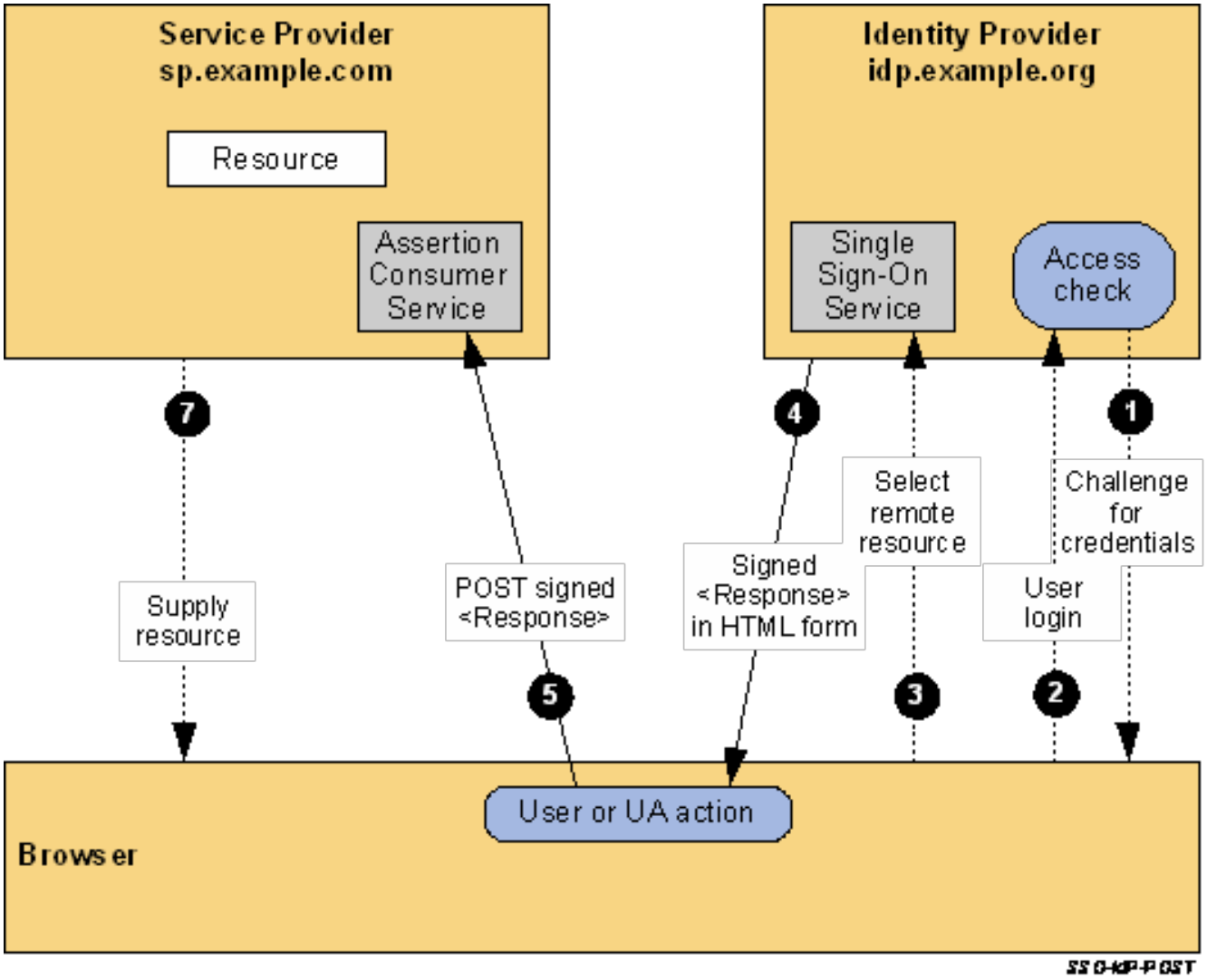
Identity Federation use case



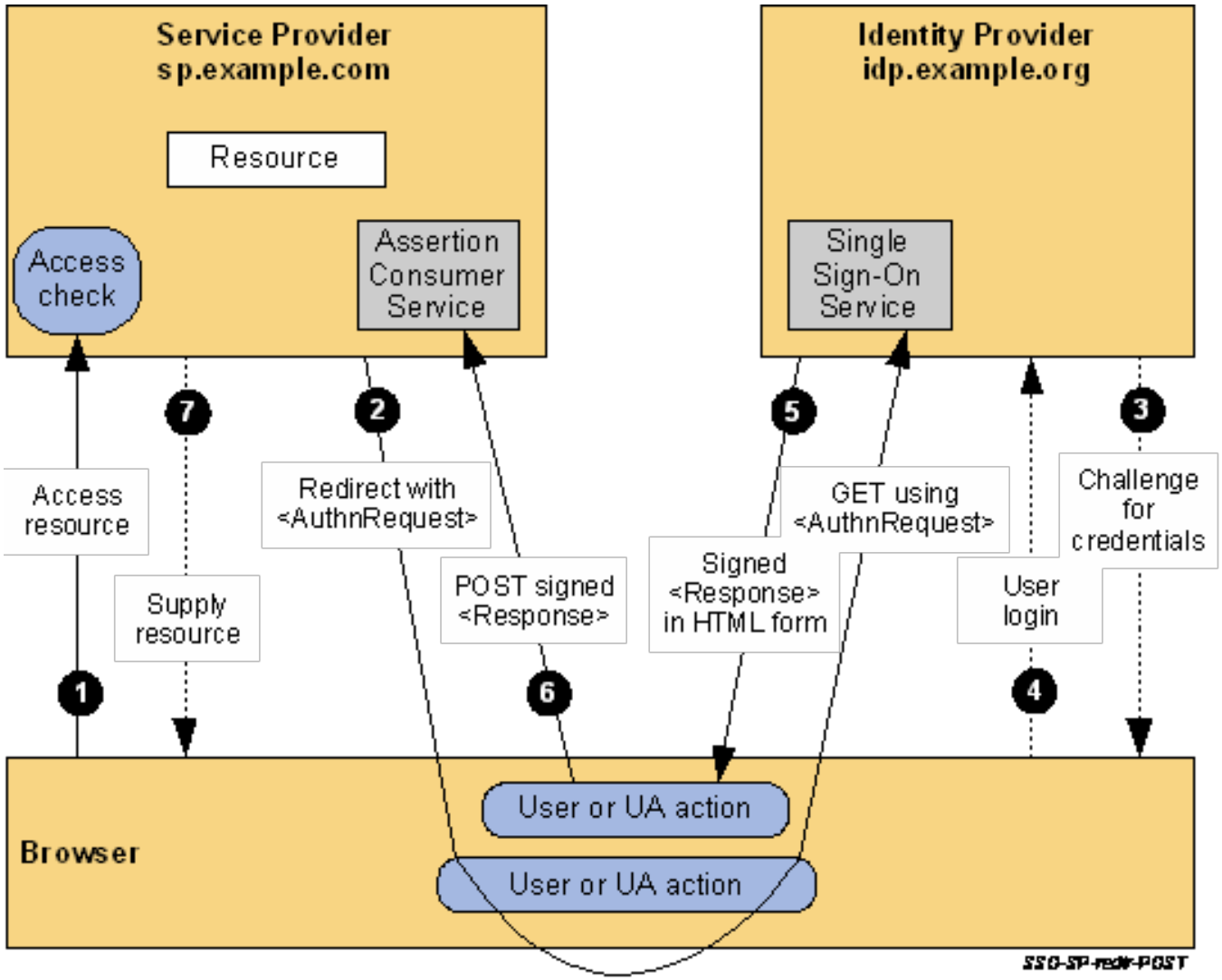
Initiation of web browser SSO



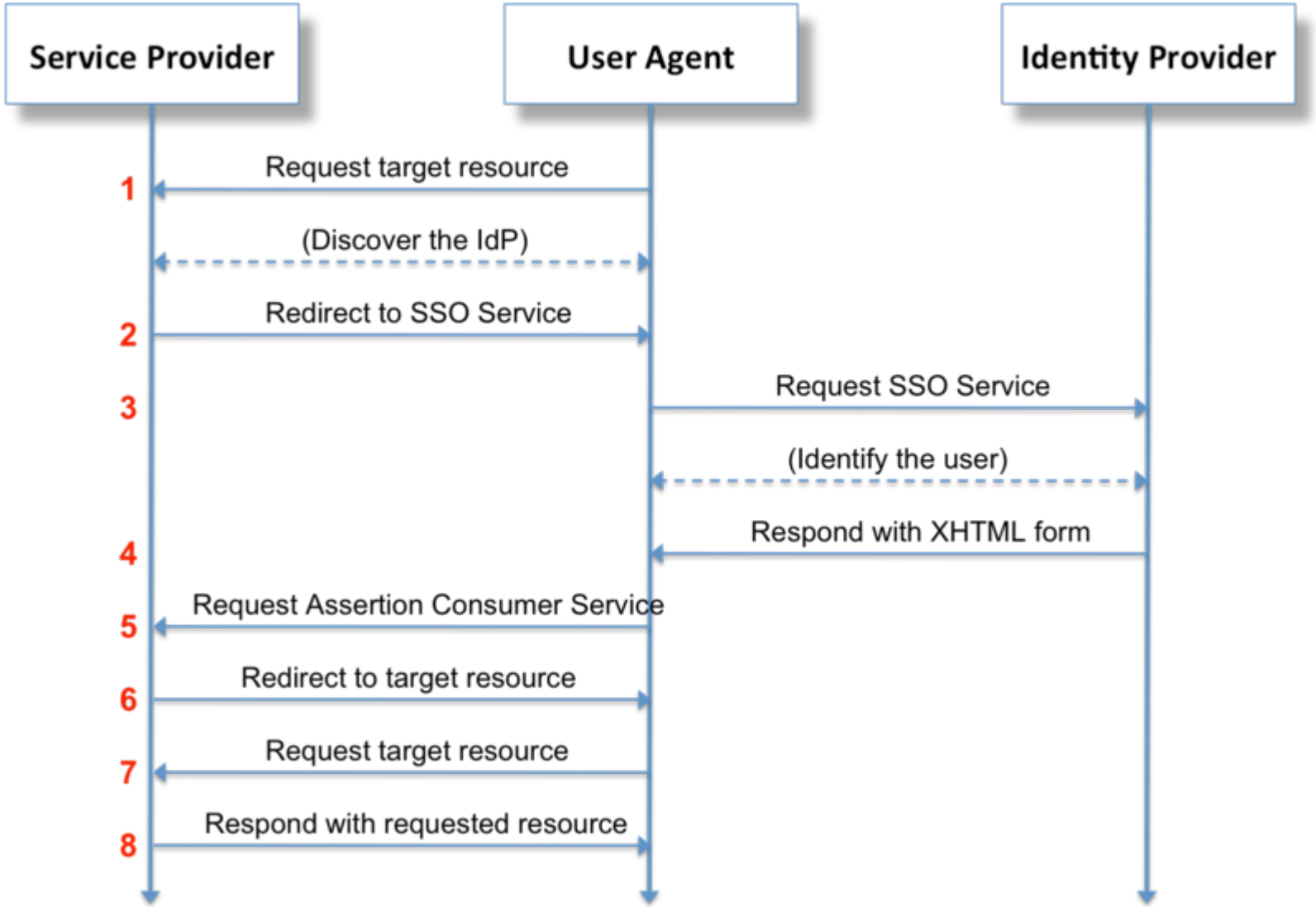
Example of IdP-Initiated SSO: POST Bindings



Example of SP-Initiated SSO: Redirect/POST Bindings



SP-Initiated web browser SSO example



Specific security web protocols

Specific security protocols:

- SAML
- **OAuth**
- **OpenID Connect**
- **JSON Web Tokens (JWT)**

OAuth 2.0 (Open Authorization)

- Developed by IETF (Internet Engineering Task Force)
- RFC **6749** (+ 6750 & 6819) first published in 2012, updates in RFC 8252 (’17) [+ m*any related standards*]
  - Not backwards compatible with OAuth 1.0 !!

- OAuth 1.0 published in RFC 5849 in 2010
  - Implements different authorization flows
  - No longer used in many services (e.g. Google APIs)

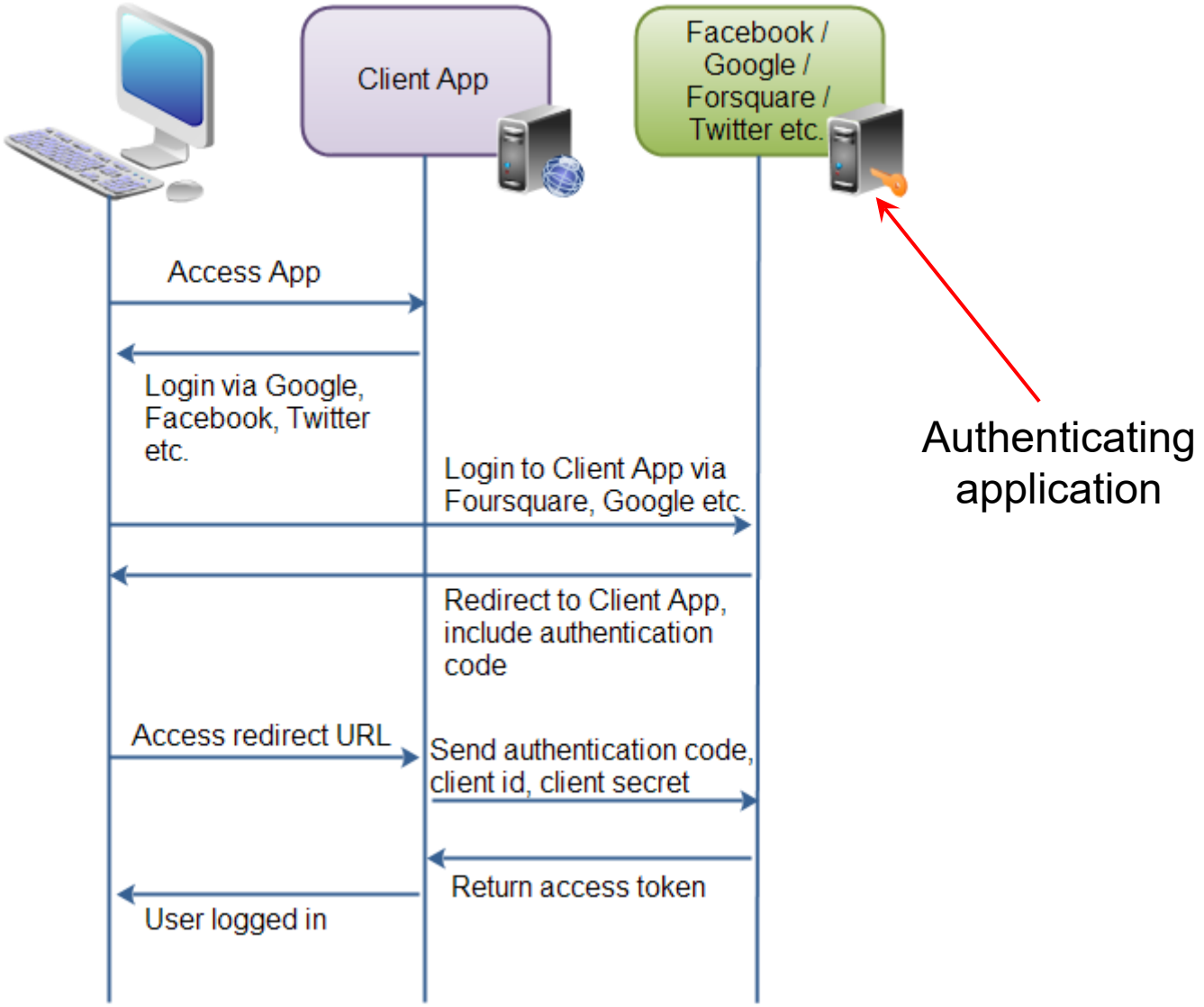
OAuth 2.0: Features

- Authorization protocol
- Allows users to approve an application to act on their behalf
- No password share with the application
- Text protocol
- Over HTTP(S)

Google is the only one that has your password

Example: Log in with Google

OAuth 2.0: Protocol example

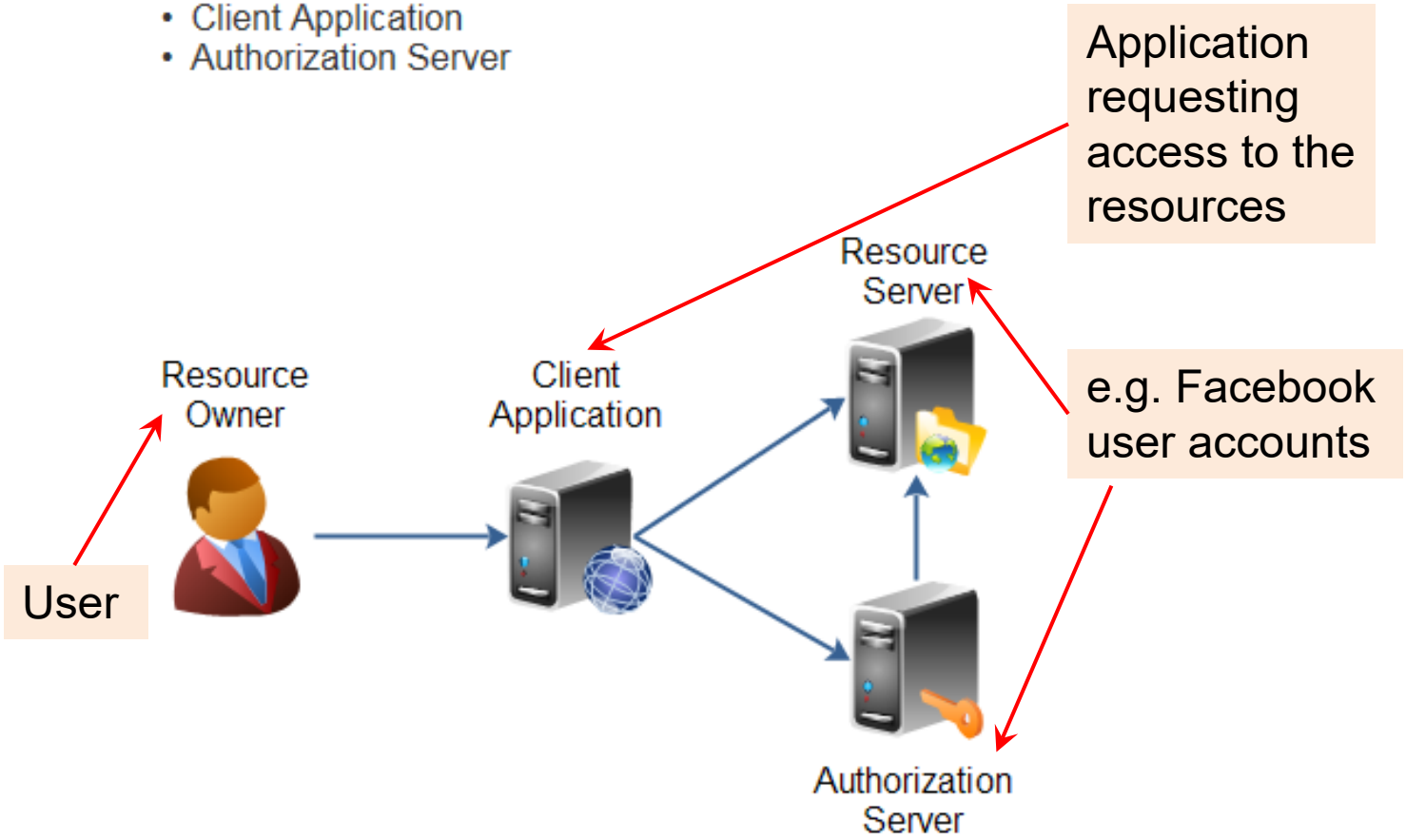


http://tutorials.jenkov.com/oauth2/overview.html

OAuth 2.0: Roles

OAuth 2.0 defines the following roles of users and applications

- Resource Owner
- Resource Server
- Client Application
- Authorization Server



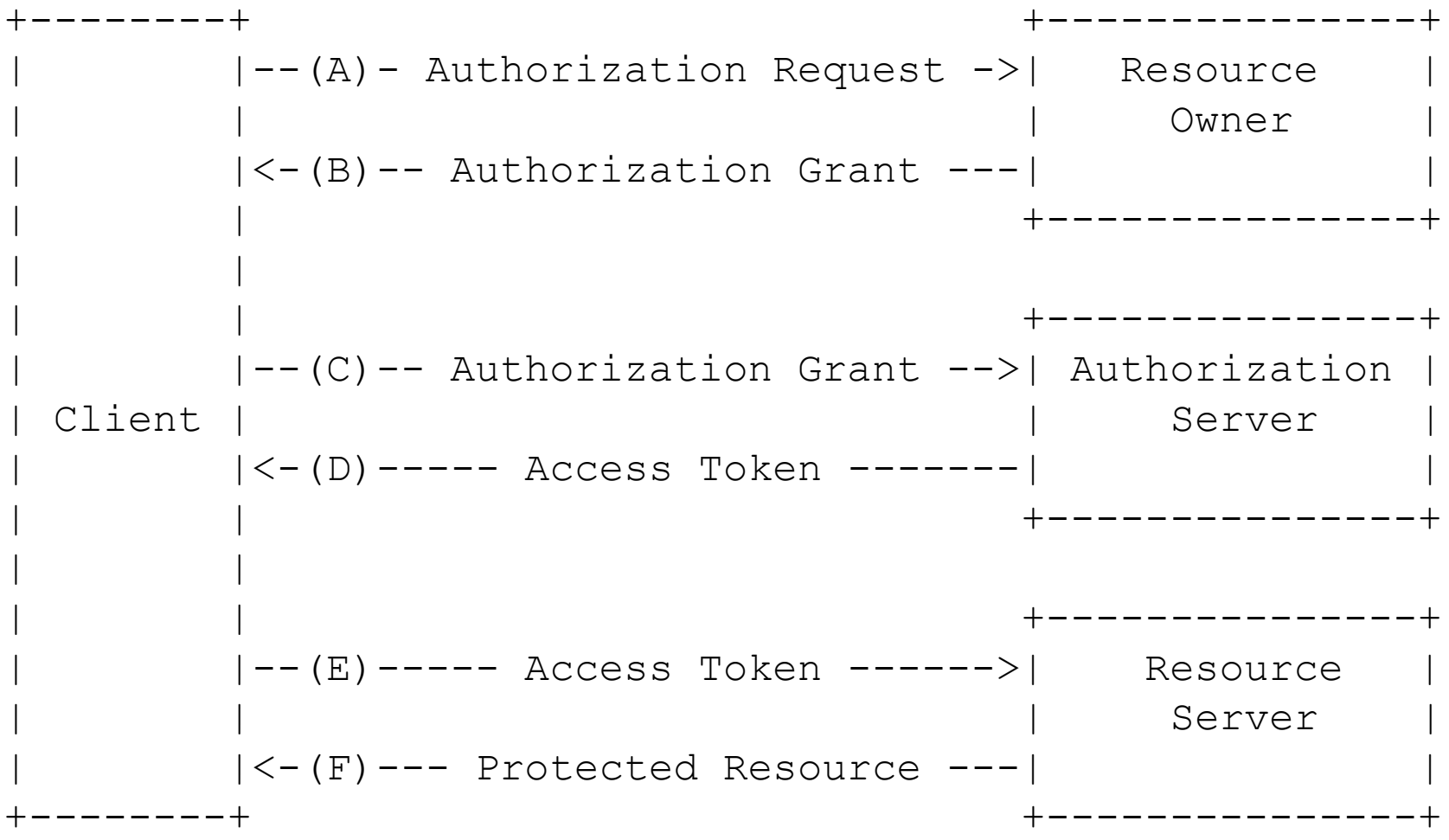
OAuth 2.0 roles as defined in the specification.

http://tutorials.jenkov.com/oauth2/roles.html

OAuth 2.0: Roles and users

- **Resource Owner**: Grants access to protected resource (typically the end-user)
- **Resource Server**: Hosts protected resources (API to be accessed)
- **Client (application)**: Requests access to protected resource on behalf of the Resource Owner
- **Authorization Server**: Authenticates the Resource Owner and issues Access Tokens after authorization
- ***User Agent****: Used by the Resource Owner to interact with the Client (for example a browser or a native application)*

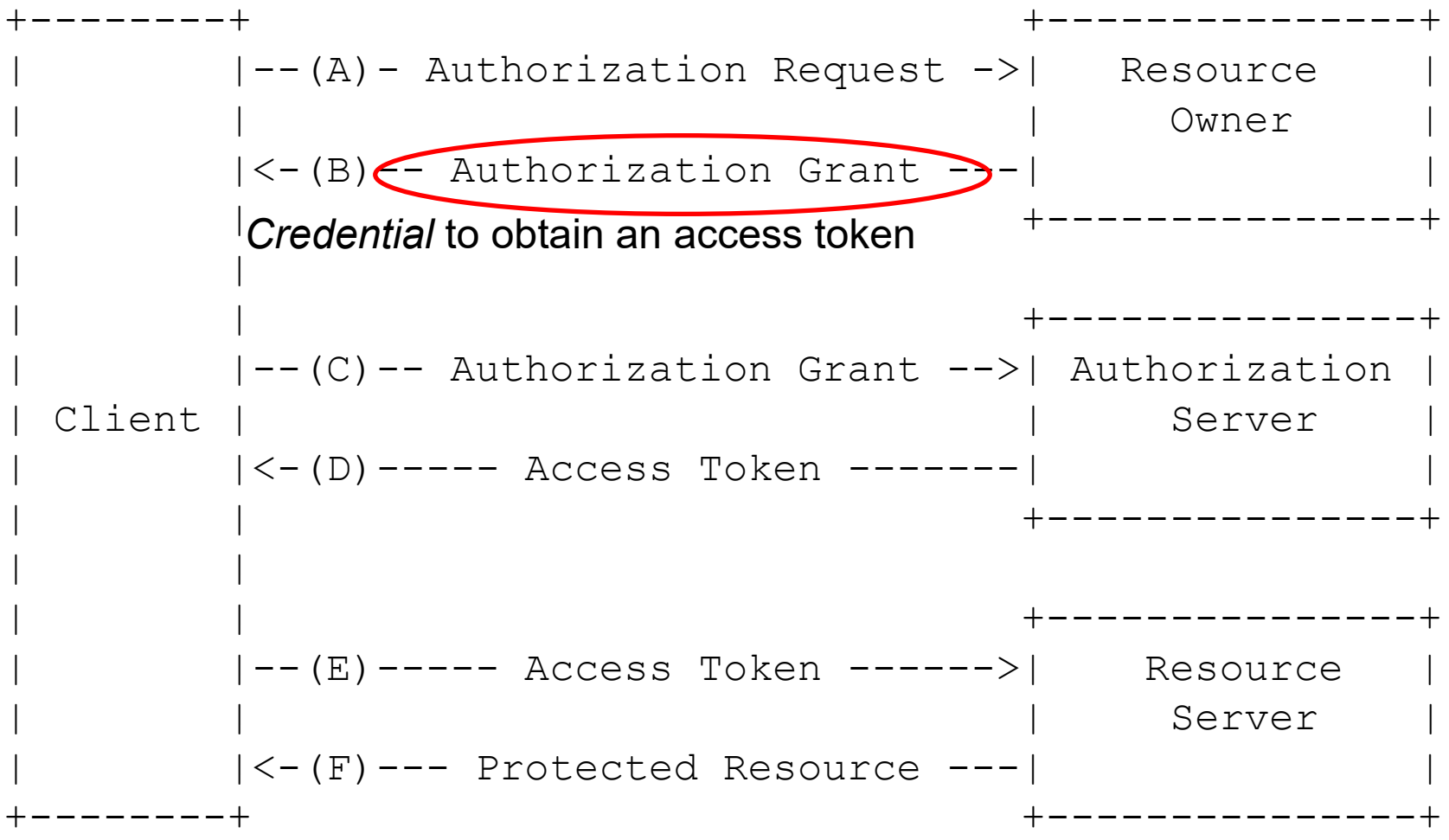
OAuth 2.0 Framework



https://tools.ietf.org/html/rfc6749

Abstract Protocol Flow

OAuth 2.0 Framework



https://tools.ietf.org/html/rfc6749

Abstract Protocol Flow

Authorization Grant types

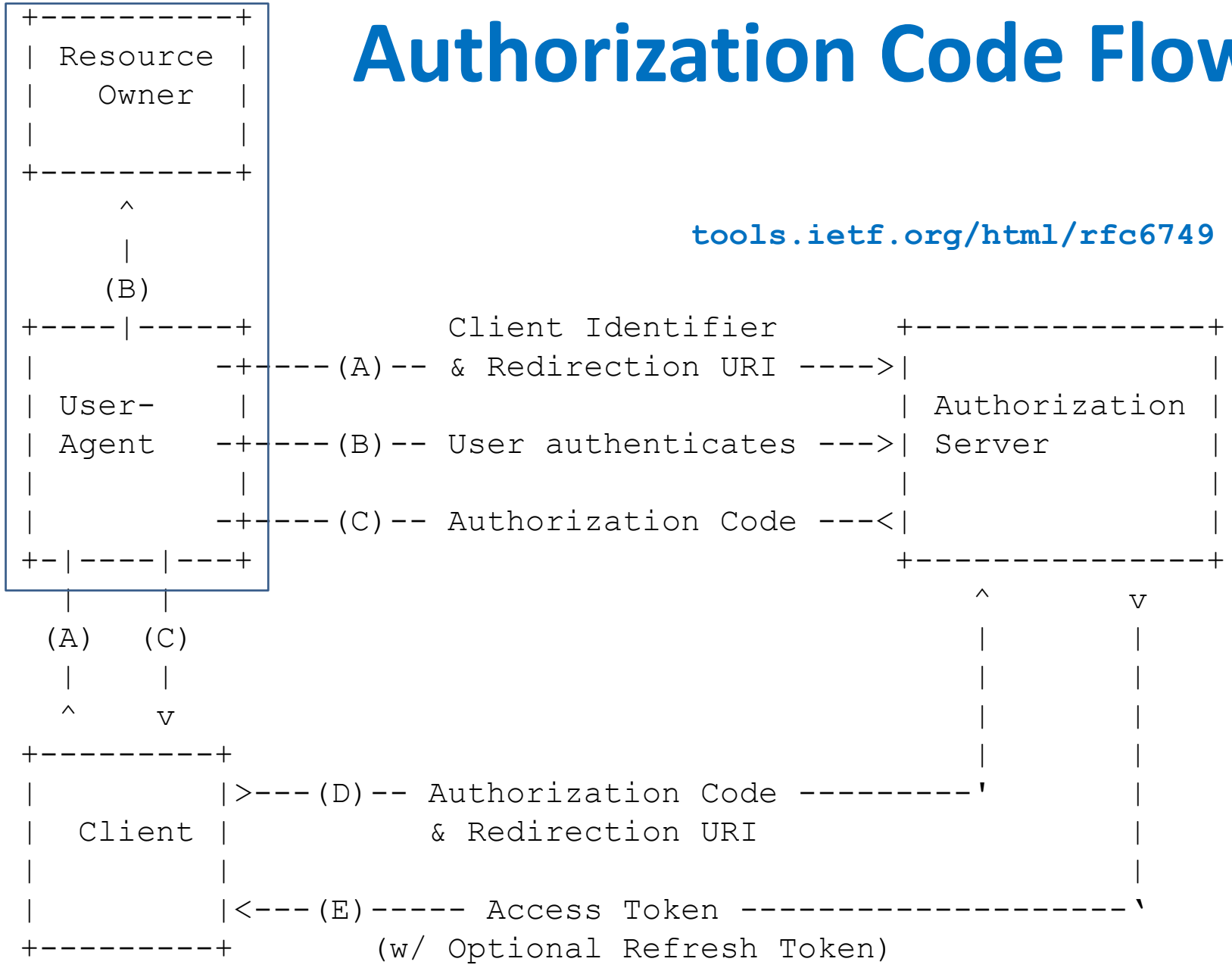
- Authorization code:
  - For server-side applications *(most commonly used)*
- Implicit:
  - Applications running on user's device
- Resource Owner password credentials:
  - Trusted applications (owned by the service, …)
- Client credentials:
  - Applications API access

Authorization Grant types

- **Authorization code:**
  - **For server-side applications *(most commonly used)***
- Implicit:
  - Applications running on user's device
- Resource Owner password credentials:
  - Trusted applications (owned by the service, …)
- Client credentials:
  - Applications API access

Authorization Code Flow

tools.ietf.org/html/rfc6749



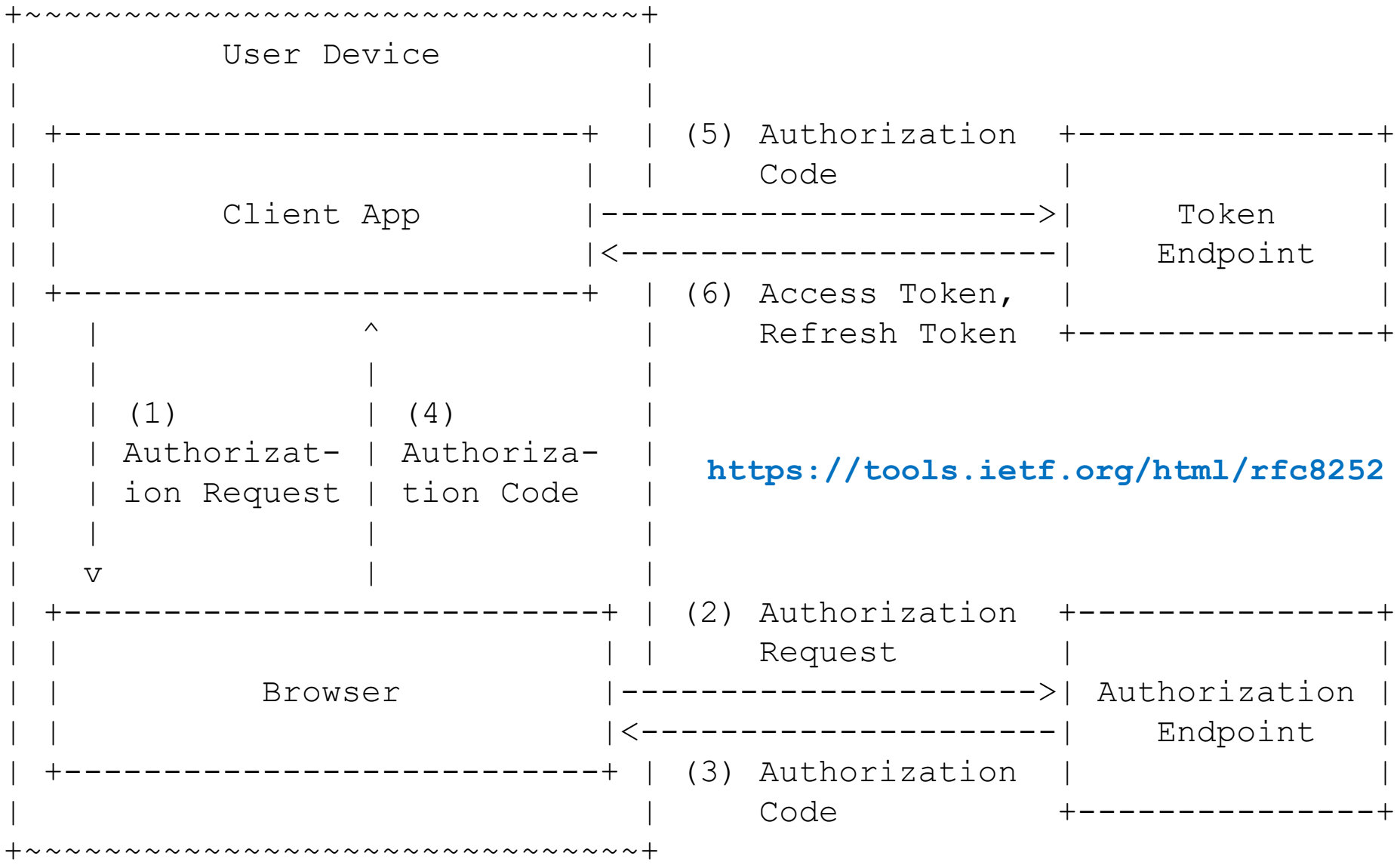
Lines illustrating steps (A), (B), and (C) are broken into two parts as they pass through the user-agent

Protocol Endpoints (RFC6749)

Authorization server endpoints (HTTP resources):

- Used by the authorization process
- Two types:
  - **Authorization endpoint** - used by the client to obtain authorization from the resource owner via user-agent redirection
  - **Token endpoint** - used by the client to exchange an authorization grant for an access token, typically with client authentication

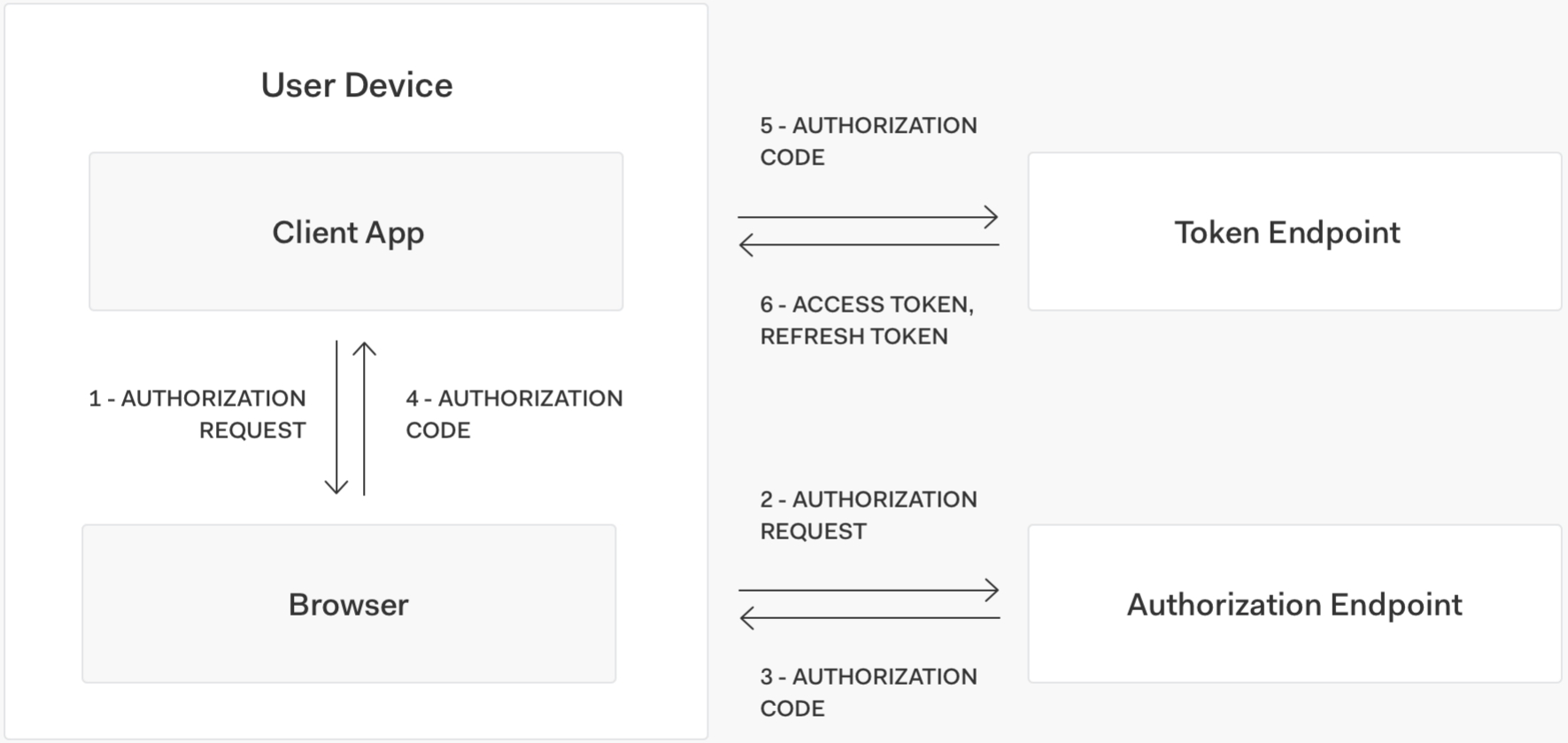
OAuth 2.0 for Native Apps (RFC8252)



Authorization Flow for Native Apps Using the Browser

OAuth 2.0 for Native Apps

Authorization Flow for Native Apps Using the Browser



Native App Authorization via an External User-Agent

https://auth0.com/blog/oauth-2-best-practices-for-native-apps/

OAuth 2.0: Authorization request

`response_type`	*Required*. Depends on grant type
`client_id`	*Required*. Id of the application that asks for authorization *(issued by authorization server)*
`redirect_uri`	*Optional*. The redirect URI registered by the client
`scope`	*Optional*. List of operations that the application requires. *Examples in https://oauth.net/2/scope/*
`state`	*Recommended*. To maintain state. Returned to the application in the redirect_uri. To prevent CSRF (Cross-site request forgery) attack

OAuth 2.0: Authorization response

`code`	*Required*. Depends on grant type. The authorization code, for example. Must expire (10' f.e.). Used once by the client
`state`	*Required, if present in request*. The same value as sent by the client in the state parameter

OAuth 2.0: Access token request

`client_id`	*Required*. Application's id
`client_secret`	*Required*. The client application's client secret
`grant_type`	*Required*. For example `authorization_code`
`code`	*Required*. The authorization code received, if it is the case
`redirect_uri`	*Required, if request URI included in authorization request*. Must be identical

OAuth 2.0: Access token response

```
{ "access_token" : "...",
  "token_type" : "...",
  "expires_in" : "..."
}
```

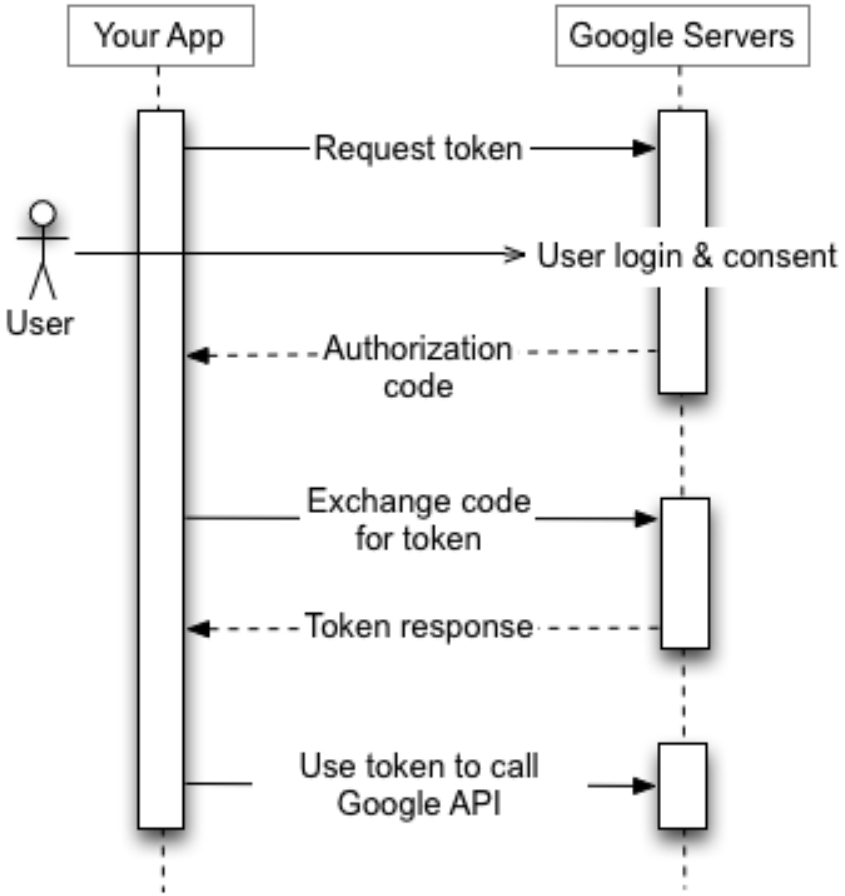
JSON array/string option

Properties:

- `access_token` - as assigned by the authorization server
- `token_type` - assigned by the authorization server
- `expires_in` - number of seconds after which the access token expires. *Expiration of access* tokens is optional
- …

OAuth 2.0: Example

Google APIs: Web server & installed applications



https://developers.google.com/identity/protocols/OAuth2

OAuth 2.0: REST example

Google APIs example (Mobile & Desktop Apps)

Obtain the authorization code

[*Authorization Request*] (URL)

```
https://accounts.google.com/o/oauth2/v2/auth?
 scope=email%20profile&
 response_type=code&
 state=security_token%3D138r5719ru3e1%26url%
3Dhttps%3A%2F%2Foauth2.example.com%2Ftoken&
 redirect_uri=com.example.app%3A/oauth2redirect&
 client_id=client_id
```

https://developers.google.com/identity/protocols/oauth2/native-app

OAuth 2.0: REST example

Google APIs example (Mobile & Desktop Apps)

Authorization code → access token

[*Access token Request*] (HTTP)

```
POST /token HTTP/1.1
Host: oauth2.googleapis.com
Content-Type: application/x-www-form-urlencoded

code=4/P7q7W91a-oMsCeLvIaQm6bTrgtp7&
client_id=your_client_id&
client_secret=your_client_secret&
redirect_uri=https://oauth.example.com/code&
grant_type=authorization_code
```

https://developers.google.com/identity/protocols/oauth2/native-app

OAuth 2.0: REST example

Google APIs example (Mobile & Desktop Apps)

Authorization code → access token

[*Access token Response*] (JSON)

```
{
  "access_token":"1/fFAGRNJru1FTz70BzhT3Zg",
  "expires_in":3920,
  "token_type":"Bearer"
  "scope": "https://www.googleapis.com/auth/
drive.metadata.readonly",
  "refresh_token":"1/xEoDL4iW3cxlI7yDbSRFYNG01
kVKM2C-259HOF2aQbI"
}
```

Duración del token suele ser menor que el de SAML, así que se necesita el parámetro de refresh token.

https://developers.google.com/identity/protocols/OAuth2InstalledApp

OAuth 2.0: REST example

Google APIs example (Mobile & Desktop Apps)

[*Refresh token Request*] (HTTP)

```
POST /token HTTP/1.1
Host: oauth2.googleapis.com
Content-Type: application/x-www-form-urlencoded

client_id=your_client_id&
client_secret=your_client_secret&
refresh_token=refresh_token&
grant_type=refresh_token
```

https://developers.google.com/identity/protocols/oauth2/native-app

OAuth 2.0: REST example

Google APIs example (Mobile & Desktop Apps)

[*Refresh token Response*] (JSON)

```
{
  "access_token":"1/fFAGRNJru1FTz70BzhT3Zg",
  "expires_in": 3920,
  "scope": "https://www.googleapis.com/auth/
drive.metadata.readonly",
  "token_type": "Bearer"
}
```

https://developers.google.com/identity/protocols/OAuth2InstalledApp

OAuth 2.0: REST example

Google APIs example (Mobile & Desktop Apps)

[*Refresh token Response*] (JSON)

```
{
  "access_token":"1/fFAGRNJru1FTz70BzhT3Zg",
  "expires_in": 3920,
  "scope": "https://www.googleapis.com/auth/
drive.metadata.readonly",
  "token_type": "Bearer"
}
```

Bearer token: grants access to the party in possession of the token

https://developers.google.com/identity/protocols/OAuth2InstalledApp

Specific security web protocols

Specific security protocols:

- SAML
- OAuth
- **OpenID Connect**
- **JSON Web Tokens (JWT)**

OpenID Connect

- Simple identity layer on top of OAuth 2.0
- Enables Clients (web-based, mobile, JavaScript, …) to:
  - verify identity of end-user based on authentication performed by Authorization Server
  - obtain basic profile information (end-user) in REST-like manner
  - …
- Specification developed by OpenID Foundation (1.0, 2014) (errata set 2 in Dec. 2023)

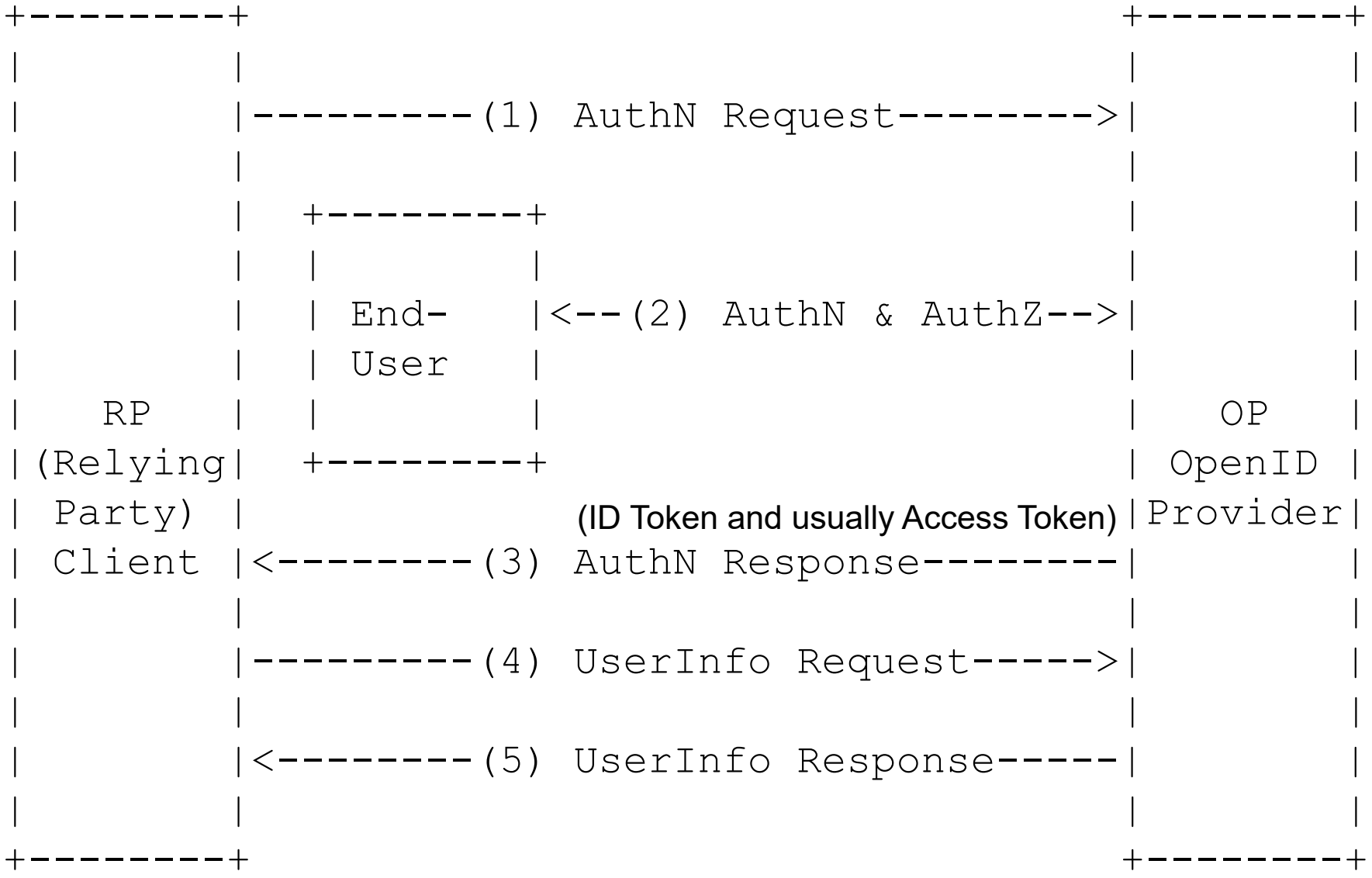
OpenID Connect

For developers:

- Provides a secure and verifiable answer to the question:
  "What is the identity of the person currently using the browser or mobile app that is connected?"
- Removes responsibility of setting, storing, and managing passwords
  → avoid credential-based data breaches

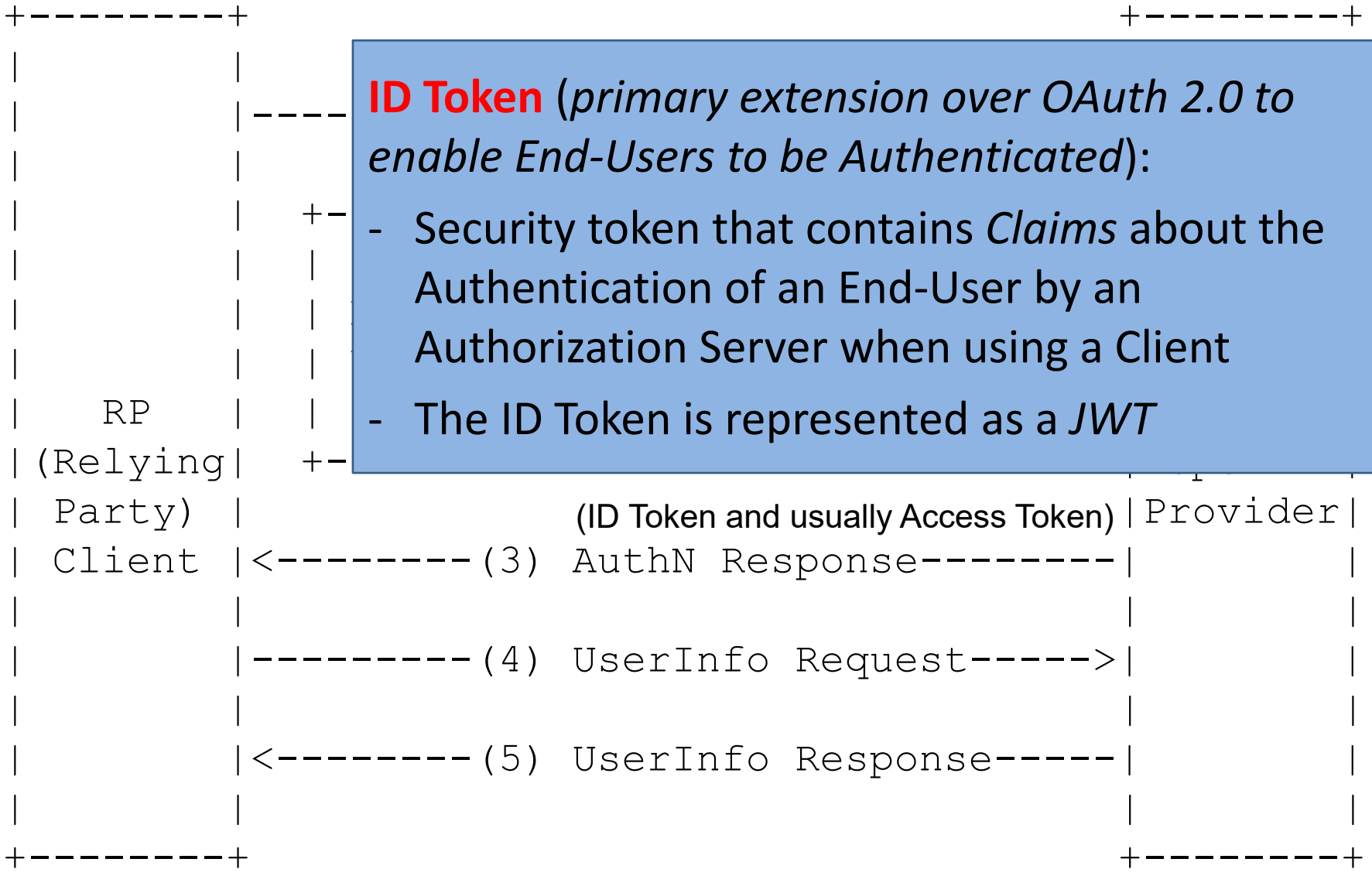
Añade peso sobre OAuth.

OpenID Connect protocol



https://openid.net/specs/openid-connect-core-1_0.html

OpenID Connect protocol

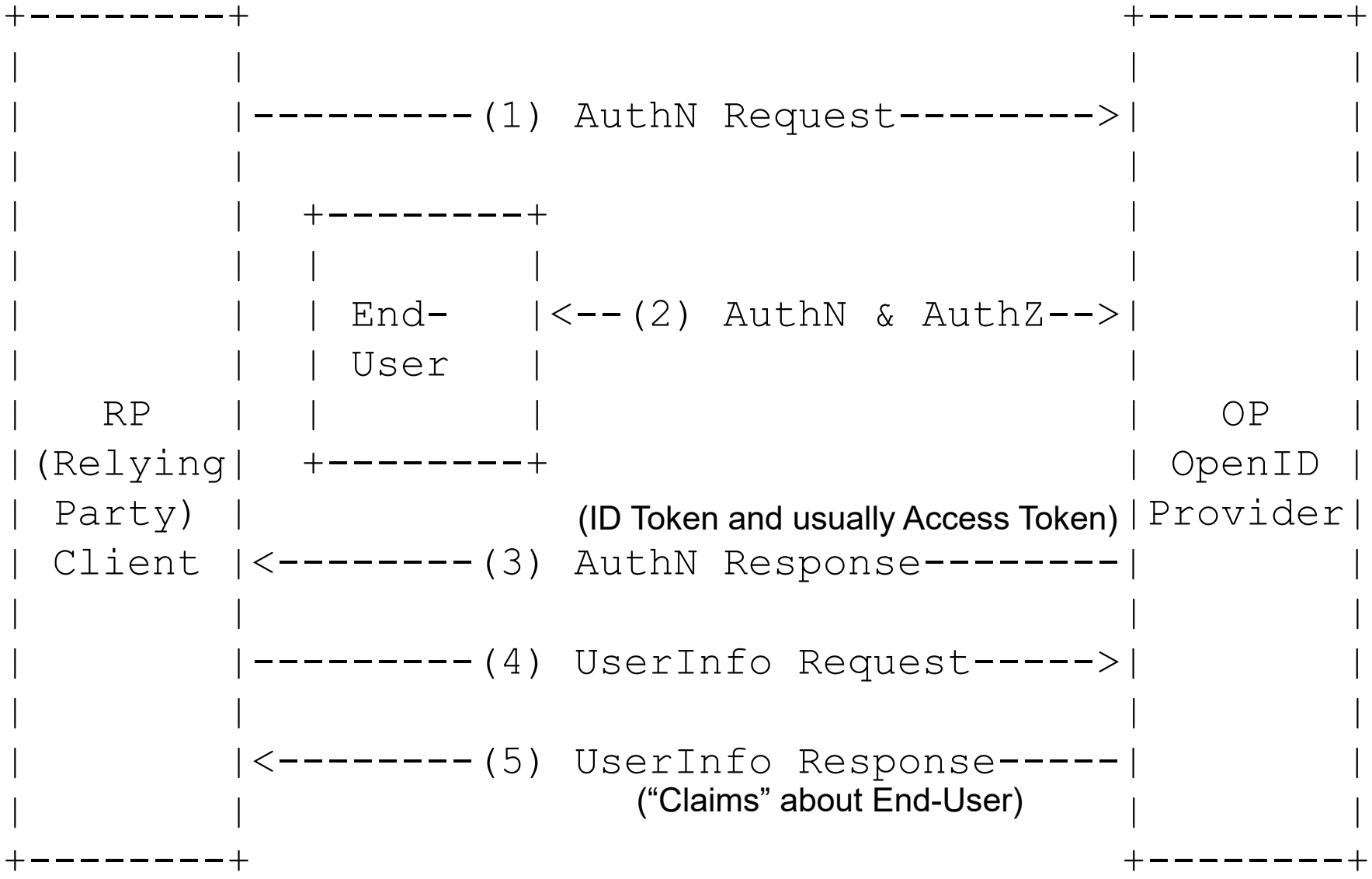


ID Token (*primary extension over OAuth 2.0 to enable End-Users to be Authenticated*):

- Security token that contains *Claims* about the Authentication of an End-User by an Authorization Server when using a Client
- The ID Token is represented as a *JWT*

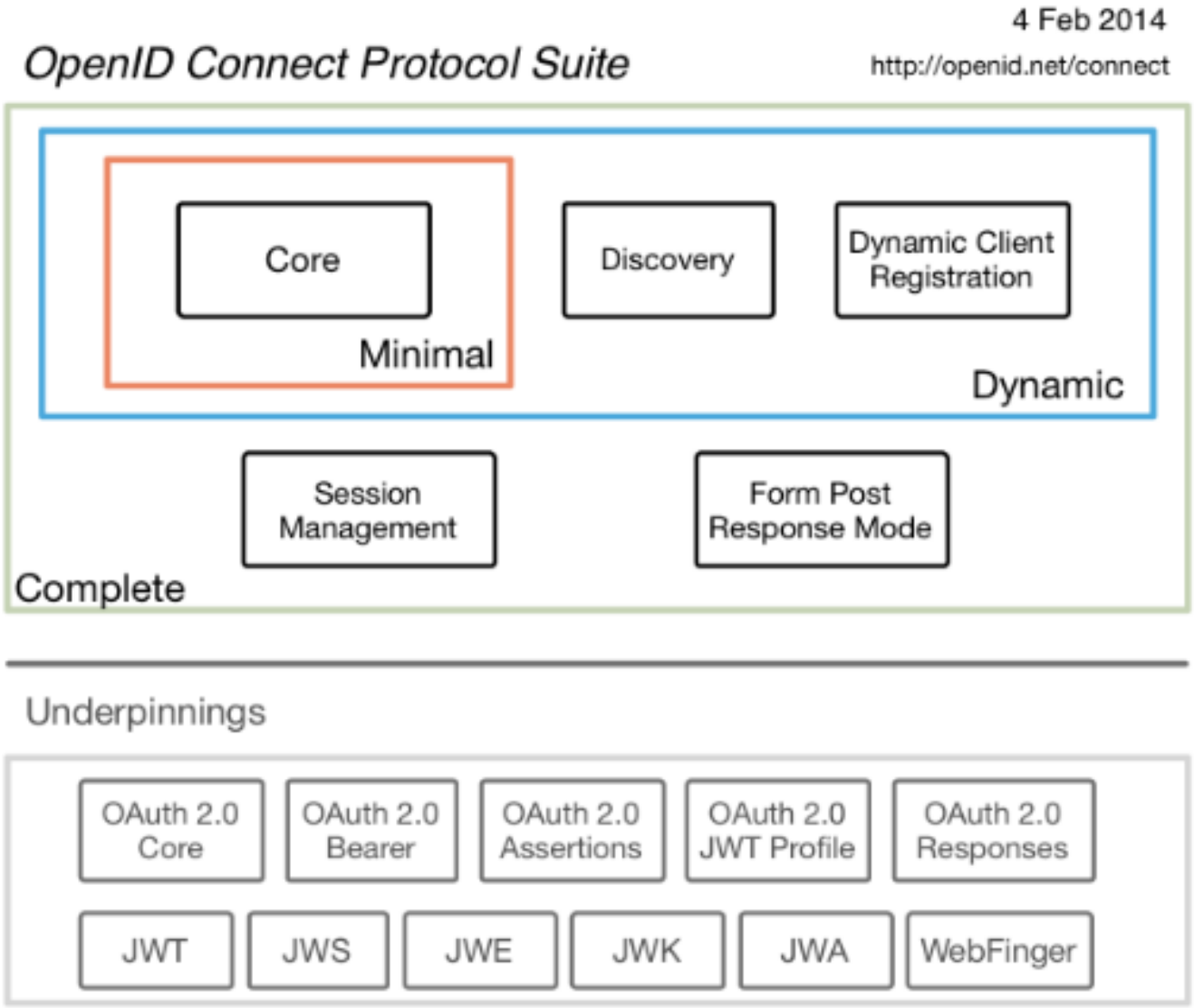
https://openid.net/specs/openid-connect-core-1_0.html

OpenID Connect protocol



https://openid.net/specs/openid-connect-core-1_0.html

OpenID Connect



OpenID Connect: Example

- Google APIs (OpenID Connect authentication URI):

```
https://accounts.google.com/o/oauth2/v2/auth?
response_type=code&
client_id=424911365001.apps.googleusercontent.com&
scope=openid%20email&
redirect_uri=https%3A//oauth2.example.com/code&
state=security_token%3D138r5719ru3e1%26url%3Dhttps%
  3A%2F%2Foauth2-login-demo.example.com%2FmyHome&
login_hint=jsmith@example.com&
nonce=0394852-3190485-2490358&
hd=example.com
```

https://developers.google.com/identity/openid-connect/openid-connect

Specific security web protocols

Specific security protocols:

- SAML
- OAuth
- OpenID Connect
- **JSON Web Tokens (JWT)**

JSON Web Tokens (JWT)

Se tiene que usar con otro protocolo distinto, no tiene su propio como OAuth.

- Tries to "simplify" SAML
- "JWT ("jot") is a standard for *safely* passing *claims* in space constrained environments" (*)
- "Container" format : estructura de datos
- Use: user authentication, secure info. exchange
- "JSON Web Token" RFC7519 (2015) *[updates later]*
- Also:
  - JSON Web Signature (JWS) RFC7515 (2015)
  - JSON Web Encryption (JWE) RFC 7516 (2015)

* *"The JWT Handbook", S.E. Peyrott*

JWT structure

- 3 parts (separated by "."):
  - Header
  - Payload
  - Signature (optional)
- *Base64Url* encoded
  - As base64 but changing 2 characters:
    62: From + to - (minus)
    63: From / to _ (underline)

no están en la tabla de mapeo

JWT structure

- **Header**
  - Signing/decrypting **alg**orithm
  - Token **typ**e (equal to “JWT” when present)
  - **c**ontent **ty**pe (for, rarely, nested JWTs)
  - …

JWT structure

- **Payload**
  - Registered claims:
    - **iss**uer, **exp**iration time, **sub**ject (principal),
    - **aud**ience (recipients), **n**ot **b**e**f**ore,
    - **i**ssued **at** (time), **j**W**t** **i**D,
  - Public claims:
    - IANA JSON Web Token Registry or URI
  - Private claims:
    - Custom

JWT structure – Public claims

- https://www.iana.org/assignments/jwt/jwt.xhtml
- Example: OpenID Connect Core 1.0
  - Claims defined in IANA JSON Web Token Registry:

OpenID Connect defined claims (1/2)

- acr Authentication Context Class Reference
- amr Authentication Methods References
- at_hash Access Token hash value
- auth_time Time when the authentication occurred
- azp Authorized party - the party to which the ID Token was issued
- nonce Value used to associate a Client session with an ID Token
- c_hash Code hash value
- address Preferred postal address
- birthdate Birthday
- email Preferred e-mail address
- email_verified True if the e-mail address has been verified; otherwise false
- family_name Surname(s) or last name(s)
- gender Gender
- given_name Given name(s) or first name(s)
- locale Locale

OpenID Connect defined claims (2/2)

- middle_name Middle name(s)
- name Full name
- nickname Casual name
- phone_number Preferred telephone number
- phone_number_verified True if the phone number has been verified
- picture Profile picture URL
- preferred_username Shorthand name to refer to End-User
- profile Profile page URL
- updated_at Time the information was last updated
- website Web page or blog URL
- zoneinfo Time zone
- _claim_names JSON object whose member names are the Claim Names for the Aggregated and Distributed Claims
- _claim_sources JSON object whose member names are referenced by the member values of the _claim_names member
- sub_jwk Public key used to check the signature of an ID Token

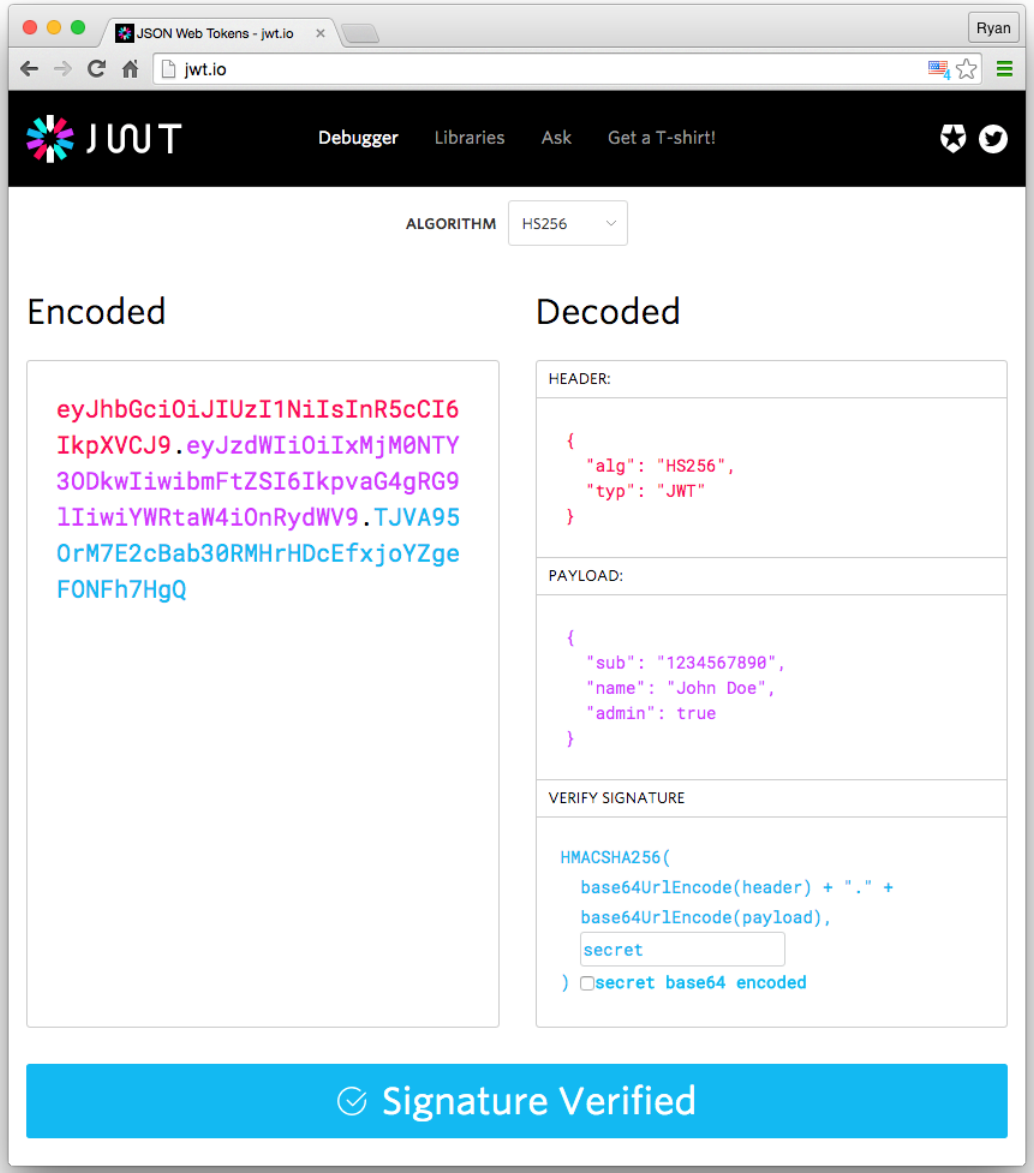
JWT structure

- **Signature (JWS)**
- Example (with the HMACSHA256 algorithm)

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

- Other recommended algorithms:
  - RSASSA PKCS1 v1.5 using SHA-256
  - ECDSA using P-256 and SHA-256
- Other algorithms supported

JWT structure



Encrypted JWT (JWE)

- **Structure**
  - Protected header
  - Encrypted key (symmetric)
  - Initialization vector *(optional)*
  - Encrypted data (ciphertext)
  - Authentication tag *(optional)*

Encrypted JWT (JWE)

- **Recommended encryption algorithms (key encryption)**
  - RSAES-PKCS1-v1_5 *(to be removed)*
  - RSAES-OAEP with defaults *(to be required)*
  - AES-128 Key Wrap
  - AES-256 Key Wrap
  - Elliptic Curve Diffie-Hellman Ephemeral Static (ECDH-ES) using Concat KDF *(to be required)*
  - ECDH-ES + AES-128 Key Wrap
  - ECDH-ES + AES-256 Key Wrap

Encrypted JWT (JWE)

- **Recommended encryption algorithms (content encryption)**
  - RSAES-PKCS1-v1_5 *(to be removed)*
  - AES CBC + HMAC SHA: AES 128 to 256-bits with Cipher Block Chaining and HMAC + SHA-256 to 512 for validation.
  - AES GCM: AES 128 to 256 using Galois Counter Mode.
  - **REQUIRED:**
    - AES-128 CBC + HMAC SHA-256
    - AES-256 CBC + HMAC SHA-512

Security in Internet Applications

- ~~Security in application layer protocols~~
- ~~XML and security~~
- ~~Specific security protocols for the Web~~