Computer Security

OAuth 2: case study
Authorization after Authentication

Authentication:

binding of an identifier to a subject

Authorization:

giving permission to an authenticated subject perform an action on an object

Access Control:

verifying that access to object is in accordance to authorization

OpenID Connect[1]

- focused on upper-level **federated** authentication
  - e.g. allows "Sign in with Google" across many different websites
  - handles trust relationship between a visited site and the *identity provider* that will perform user authentication (possibly, using FIDO2!)
  - "unfortunately", this identity layer is built on top of OAuth 2.0
    - so, terminology gets confusing because: "authorization" keeps appearing in control flow...

Typical OIDC flow

- *User* visits, with browser, website's *App*
- *App* redirects browser to OpenId(entity) Provider, OIdP[2]
- OIdP authenticates *User* and gives an *ID token*[3] to *App*
- (...*now, OAuth's authorization flow takes over for an eventual access to resource*...)

1 aka OIDC
2 trusted by *App*
3 A *token* is a small document, protected against forgery, modification and, sometimes, disclosure.

Entities

- user (if human, "E*nd user"*)
  - "Resource Owner" in OAuth 2.0 terminology
- browser (tool to access web service)
  - "User-Agent" in OAuth 2.0 terminology
- web application (needs users authentication -& authorization!- to access resources)
  - "Client" in OAuth 2.0 terminology
    - Relying Party, trusts Identity Provider for users authentication
- Identity Provider[1] (authenticates users, e.g. using FIDO)
  - "Authorization Server" in OAuth 2.0 terminology
  - issues ID Tokens[2]
    - *assert* who the user (in more or less extent)
    - protected by digital signature and, possibly, encipherment

1 here, OpenID Provider, OP
2 and *authorization grants*...

Terminology confusion

- arises because OpenId Connect uses OAuth 2.0 to handle authentication!
  - technically, the user **authorizes** the Identity Provider to release their identity information to the app!
- e.g. definition of *OpenID Provider (OP):* [1]
  - *OAuth 2.0* ***Authorization Server*** *that is capable of* ***Authenticating*** *the End-User and providing Claims to a Relying Party about the* ***Authentication*** *event and the End-User.*

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

The OpenID Connect protocol, in abstract

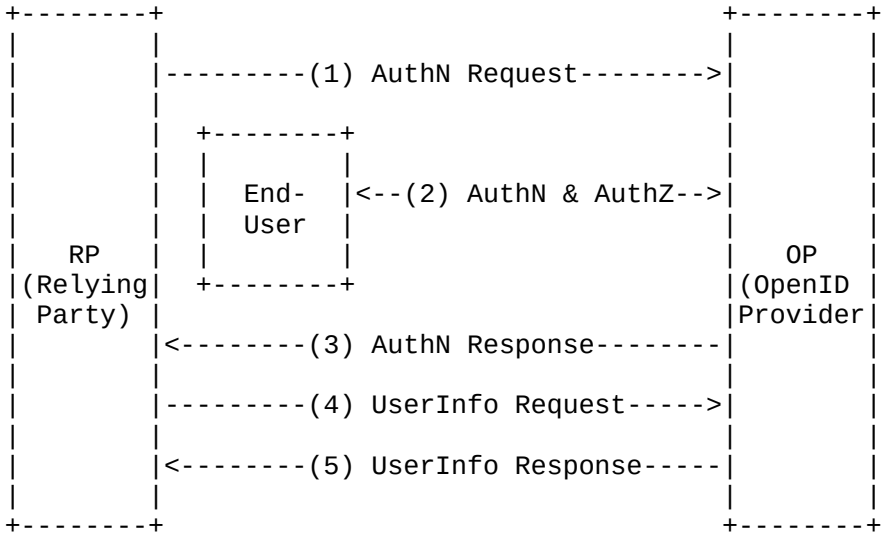


Fig. The OpenID Connect protocol as stated in OpenID Connect Core 1.0 (https://openid.net/specs/openid-connect-core-1_0.html)

...OpenID Connect: protocol...

Legend of previous Figure

- AuthN - Authentication
- AuthZ - Authorization

The OpenID Connect protocol, in abstract, follows the following steps.[1]

1. *The RP (Client) sends a request to the OpenID Provider (OP).*
2. *The OP authenticates the End-User and obtains authorization.*
3. *The OP responds with an ID Token and usually an Access Token.*
4. *The RP **can** send a request with the Access Token to the UserInfo Endpoint.*
5. *The UserInfo Endpoint returns Claims about the End-User.*

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

OAuth 2.0

- open standard to authorization (not authentication)
  - RFC 6749: *The OAuth 2.0 Authorization Framework*
- specifically designed for use with HTTP![1]
  - extensive use of re-directions (imposed on User-Agent, e.g. web browser)
- more *Framework* than *Specification* with optional and undefined parts and many possible extensions, so that
  - is "*likely to produce a wide range of non-interoperable implementations*"!
- uses *authorization grants* and *access tokens* opaque to the applications
- not an authentication protocol, but implicitly includes it:[2]
  - OpenID Connect extends OAuth
    - supplies *identification tokens (*got from Authentication Providers)
      - that represent the user and contain user info (claims)

1 RFC 6749 (1. Introduction): «*This specification is designed for use with HTTP ([RFC2616]). The use of OAuth over any protocol other than HTTP is out of scope*.»

2 more than what one would say by "of course!..."

Example of general usage:

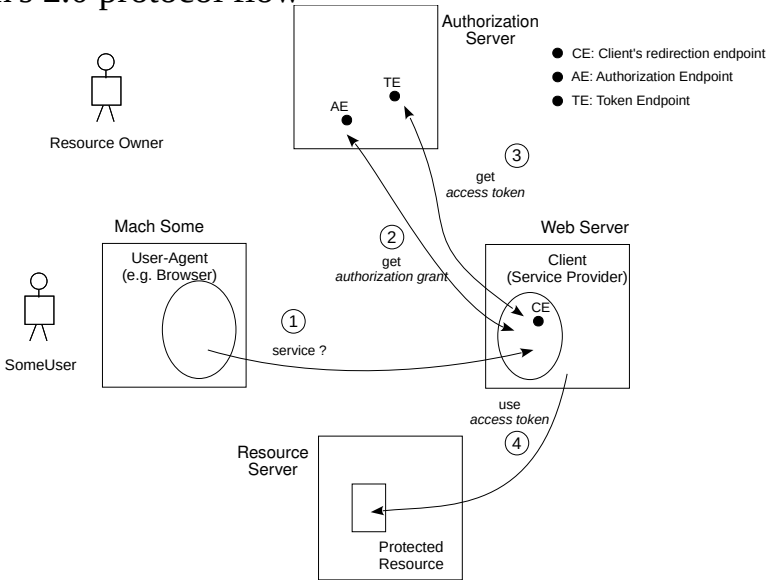
- Alice can grant a Printing service access to her protected photos stored at Bob's Photo-sharing service, without sharing her username and password with the Printing service:
  - *she authenticates herself with a server trusted by Bob's Photo-sharing service, authorizes the terms of the requested access to her photos and the server issues the Printing service delegation-specific credentials*

OAuth 1.0

- published as RFC 5849, was the result of a small *ad hoc* community effort
- OAuth 2.0 builds on OAuth 1.0 deployment experience, but is not backward compatible with it and shares with it very few implementation details.
  - «*When compared with OAuth 1.0, the 2.0 specification is more complex, less interoperable, less useful, more incomplete, and most importantly, less secure.*» (Eran Hammer, initial lead author and editor of OAuth 2.0 and lead author of OAuth 1.0)

Definitions

- **Entities**:[1] participating in OAuth's 2.0 protocol flow
  - Resource Owner
  - Resource Server
  - Client (Application)
  - Authorization Server



1 *Roles* in RFC 6749's parlance

...OAuth 2.0...

- **(Authorization) Grants**: credentials that express consent from Resource Owner to access their Resource
  - authorization code type (preferred)
  - implicit type (not advised)
  - resource owner password credentials type (not advised)
  - client credentials type (preferred for Machine-to-Machine)
- **Token**: data object representing authenticated, authorized or delegated security state
  - Access Token
  - Bearer Token (variant of *access token*)
  - Refresh Token (for getting a new *access token*)
- **Endpoint**: HTTP resource[1] where exchange of security information takes place
  - Authorization endpoint (for geting grants)
  - Token endpoint (for getting tokens)
  - Client endpoint

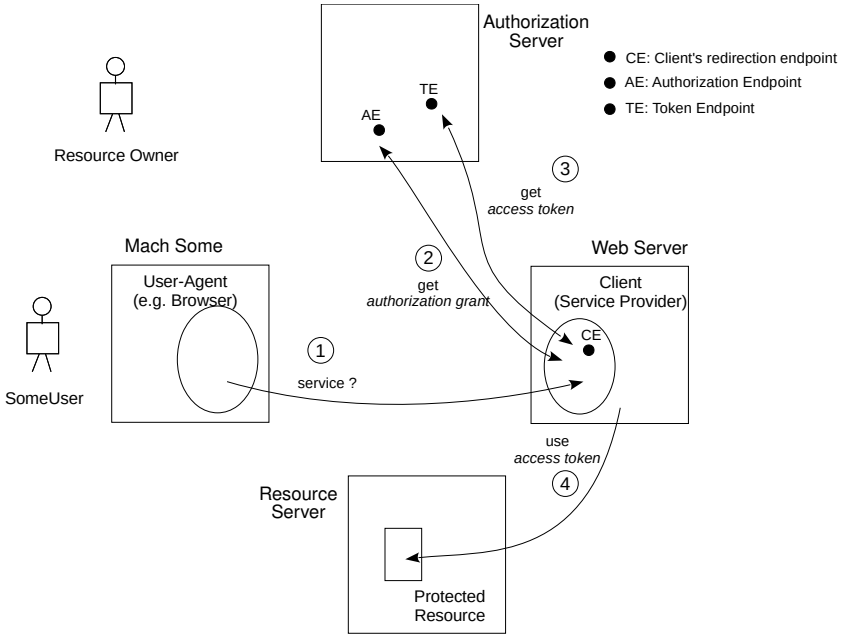
1 URI (Uniform Resource Identifier)

OAuth2's roles

- Resource Owner (if person: User, End-User)
  - determines who and how can access Resource (Object)
  - e.g. Alice has her photos stored in Bob's Photo-sharing service
- Resource Server
  - controls access to Resource by abiding Resource Owner's directives expressed in an access token
  - e.g. Bob's Photo-sharing service grants access to Alice's photos to whom presents a valid access token
- Client (Application)
  - may access protected resources if Resource Owner consents
  - e.g. Printing service accesses Alice's photos (in Bob's Photo-sharing service) only with Alice's consent (expressed in an access token)
  - ... continues...

...OAuth 2.0: roles...

- ... *continued*...
- Authorization Server
  - emits access token after Resource Owner's authentication and consent
  - e.g. Printing service asks Authorization Server for an access token to Alice's photos (stored in Bob's Photo-sharing service); Authorization Server will give the access token only after authenticating Alice[1] and getting Alice's consent



1 possibly in an Authentication Server

Authorization Grants

- Authorization Grant
  - credential representing the Resource Owner's authorization to access their protected resources
  - Client needs it for asking Access Token from Authorization Server
    - with Access Token, Client may access Resource
- Access flow depending on Grant types
  - authorization code:[1] 2 steps [FIG-G:AC]
    - 1st: Client gets *authorization code* from Authorization Server, through User-agent (redirect)
    - 2nd: Client gets *access token* from Authorization Server
  - *... continues...*

1 Preferred, specially when paired with PKCE (Proof Key for Code Exchange) - see ahead.

...OAuth 2.0: access flow depending on Grant types...

- *... continued...*
- implicit:[1] 1 step [FIG-G:I]
  - Client gets *access token* from Authorization Server, through User-agent (redirect)
- resource owner password credentials:[2] 2 steps [FIG-G:ROC]
  - 1st: Client gets *Resource Owner credentials* from Resource Owner through User-agent (redirect)
  - 2nd: Client gets *access token* from Authorization Server
- client credentials:[3] 1 step [FIG-G:CC]
  - Client gets *access token* from Authorization Server, if had previous Grant or works for Resource Owner
- (other, anticipated by extensibility mechanism)

1 Not advised as exposes tokens to unauthorized scripts, browser history logs and interception. Deprecated in OAuth 2.1.
2 Not advised, as the user's credentials are exposed to the client! Deprecated in OAuth 2.1.
3 Preferred for Machine-to-Machine (no User involved).

...OAuth 2.0: grant types…

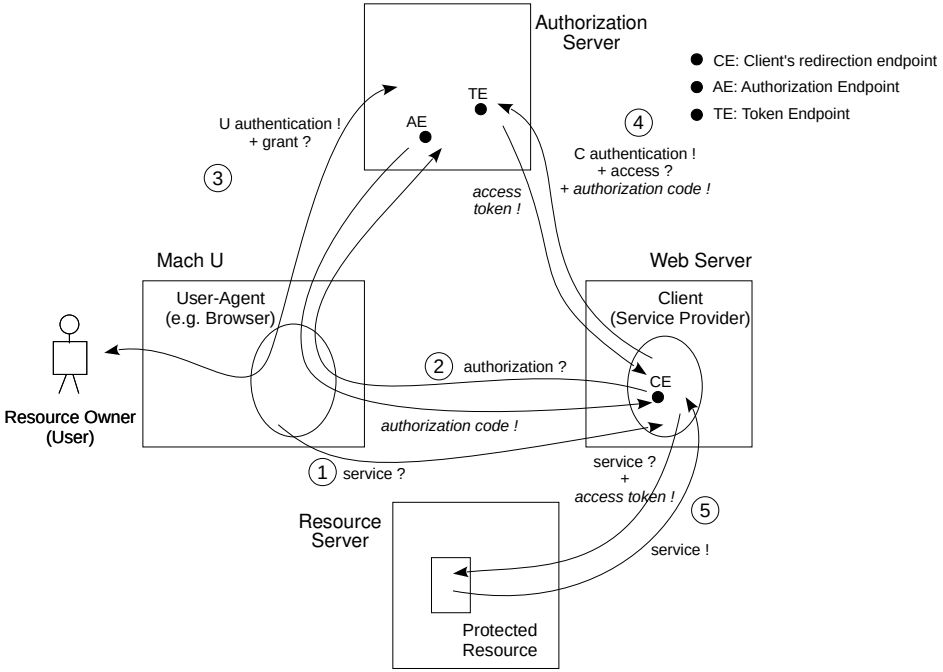


Fig-G:AC. OAuth 2.0 scenario with Grant type *authorization code*.

...OAuth 2.0: grant types…

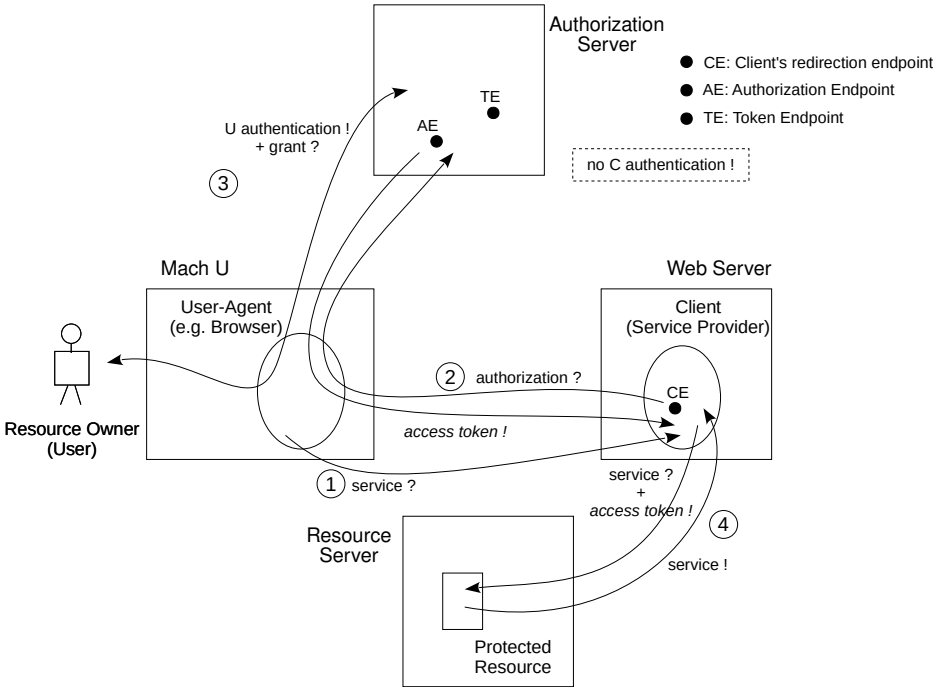


Fig-G:I. OAuth 2.0 scenario with Grant type *implicit*.

...OAuth 2.0: grant types…

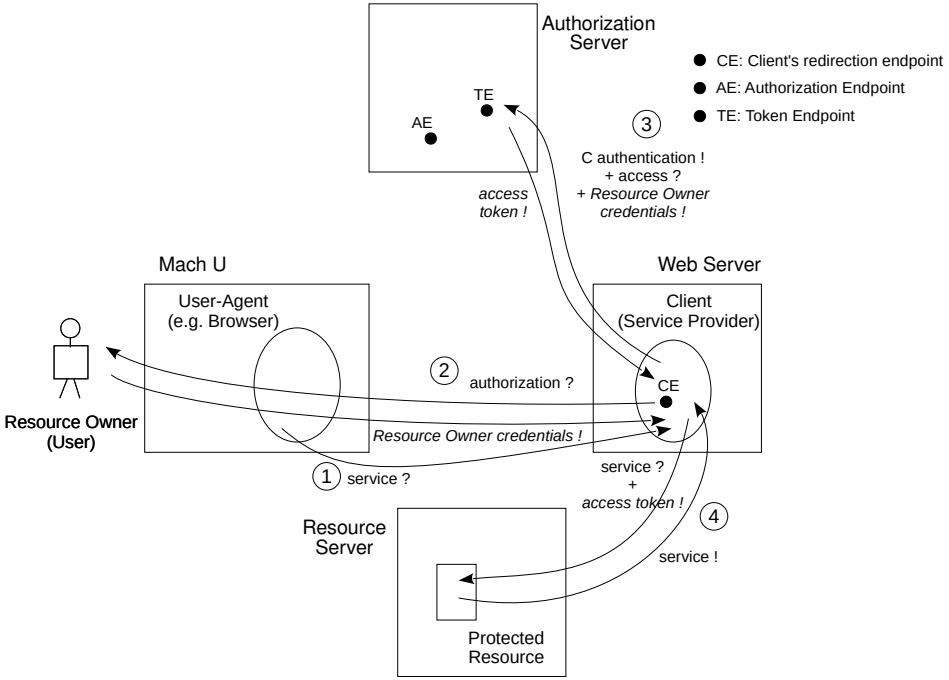


Fig-G:ROC. OAuth 2.0 scenario with Grant type *resource owner password credentials*.

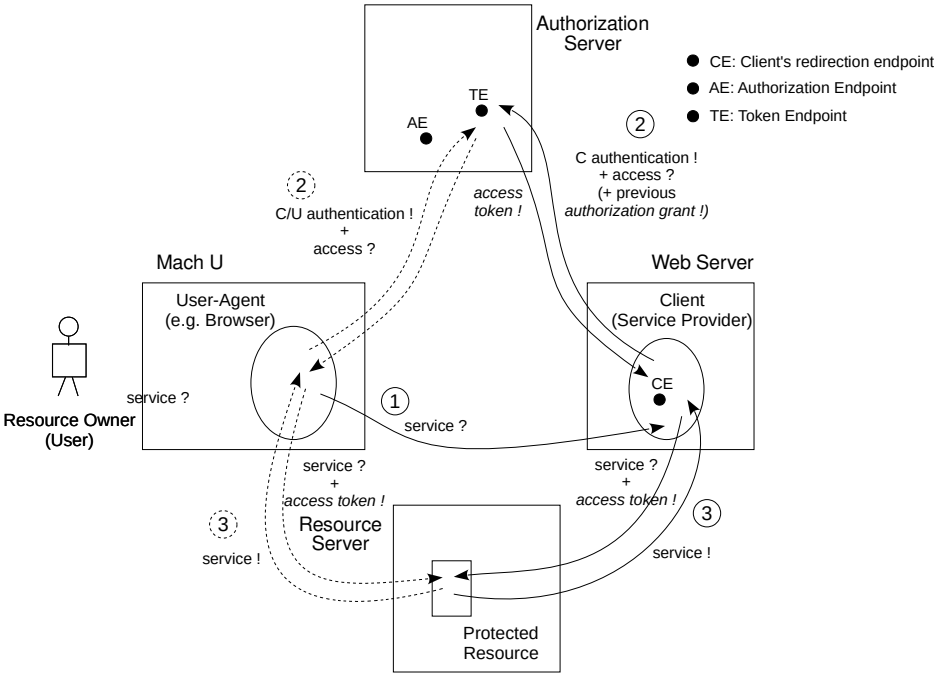


Fig-G:CC. OAuth 2.0 two scenarios with Grant type *client credentials*.

Endpoint

- HTTP resource (URI) where exchange of security information takes place
- Authorization Server endpoints:
  - 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
- Client endpoint:
  - Redirection endpoint
    - used by the Authorization Server to return responses containing authorization credentials to the Client via the Resource Owner's User-agent
    - presented when making the authorization request, OR
    - established during the (optional) client registration process

Token

- data object representing authenticated, authorized or delegated security state
  - portable security item, carrying trust assertions, usable by other entities
- examples: Kerberos tickets, OAuth access tokens, OpenID Connect ID tokens

Token types

- Access Token
  - credential used to access a (protected) resource
    - contains:
      - delegated authorization
      - scopes/permissions of resource usage
      - limited validity
    - generally, is opaque (to Client)
    - may be cryptographically signed

...OAuth 2.0: token types...

- Bearer Token
  - type of Access Token that
  - using it does not require any other proof, besides its possession!
- Refresh Token
  - credential used to obtain a new Access Token[1]
    - useful to extend the life of the initial Access Token
  - generated by the Authorization Server and given with the (initial) Access Token
    - used only with authorization servers (are not sent to Resource Servers)

1 identical or with less privileges than the initial Access Token!

Clients

- applications that want to access protected Resources
  - for that they must get an access token from an Authorization Server
  - which, in general,[1] demands the Client to authenticate
- Client types depend on their ability to perform a secure authentication:[2]
  - confidential type - able to perform secure authentication
  - public type - unable to to perform secure authentication
- Client profiles
  - web application - confidential client running on a web server
  - user-agent-based application - public client running in User-agent[3]
  - native application - public client installed and executed on User device
- each Client gets an Identifier[4] from the Authorization Server upon Client registration

1 an exception arises with the *Implicit Grant* type
2 secure as specified by the Authorization Server
3 e.g. Java applets or JavaScript code
4 is not a secret!

...OAuth 2.0...

Clients' Registration[1]

- is optional, except for
  - public clients or confidential clients using Implicit Grant type
- if not used
  - necessary operational details are beyond the scope of RFC 6749's specification
- if used, is done initially with the Authorization Server
  - Client will specify:
    - type
    - redirection endpoint, for receiving *access token*[2]
    - other information (e.g., application name, website, description...)
  - Authorization Server will return to Client:
    - Identifier - unique string; not a secret
    - Secret – to establish set of client credentials[3] for future authentications with the authorization server

1 with Authorization Server
2 from Authorization Server (via Resource Owner's User-agent) in response to requests with *Authorization Code Grant* or *Implicit Grant*
3 e.g., password, public/private key pair

Proof Key for Code Exchange (PKCE)[1]

- RFC 7636: extension to OAuth 2.0's Authorization Code flow to prevent interception attacks
- basically, connects Step 1, getting grant, to Step2, getting access token, in Authorization Code flow
- in FIG:[2]
  - nonce: *code_verifier*
    - 43-126 char random string
  - ~hash(nonce): *code_challenge*
    - BASE64URL-ENCODE (SHA256 (ASCII (*code_verifier*)))

1 pronounced "*pixie*"
2 RFC 7636's parlance: *code_verifier*, *code_challenge*

...OAuth 2.0: PKCE...

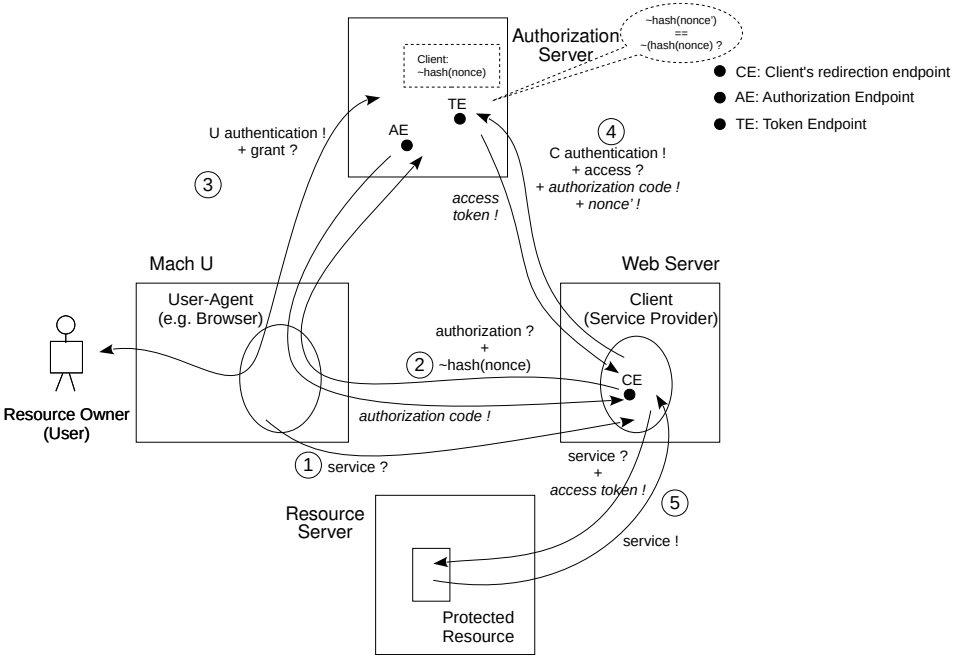


Fig. OAuth 2.0 scenario with Authorization Code grant protected with Proof Key for Code Exchange (PKCE).

Pointers...

- The "**OAuth 2.0 Authorization Framework**", 2012 - D. Hardt (Editor)
  - tools.ietf.org/html/rfc6749
- "**Proof Key for Code Exchange by OAuth Public Clients**", 2015 - N. Sakimura (Editor) et al.
  - tools.ietf.org/html/rfc7636
- "**OpenID Connect Core 1.0**", incorporating errata set 2, 2023 - OpenID Foundation
  - openid.net/specs/openid-connect-core-1_0.html
- "**OpenID Specifications**", 2026 - OpenID Foundation
  - openid.net/developers/specs/
- "**Client Authentication on the Web**", 2001 - Kevin Fu, Emil Sit, Kendra Smith, Nick Feamster
  - pdos.csail.mit.edu/papers/webauth:sec10.pdf