Recent evolutions in the OAuth 2.0 and OpenID Connect landscape

Dr. Philippe De Ryck

https://Pragmatic Web Security.com

Dr. Philippe De Ryck

- Deep understanding of the web security landscape
- Google Developer Expert (not employed by Google)
- Course curator of the SecAppDev course
  (https://secappdev.org)

Pragmatic Web Security

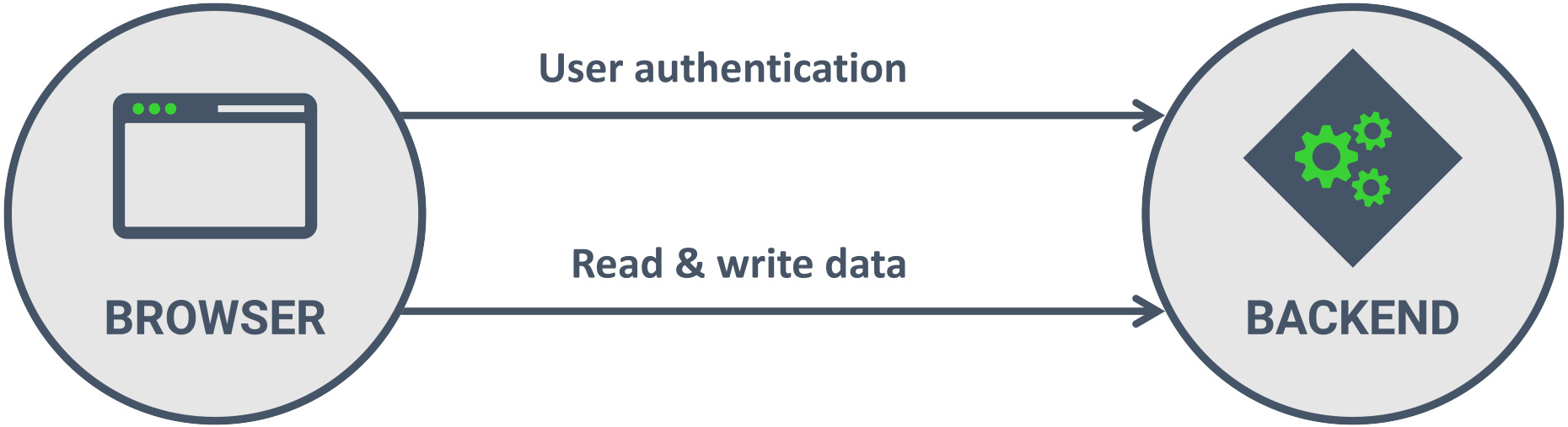
High-quality security training for developers and managers

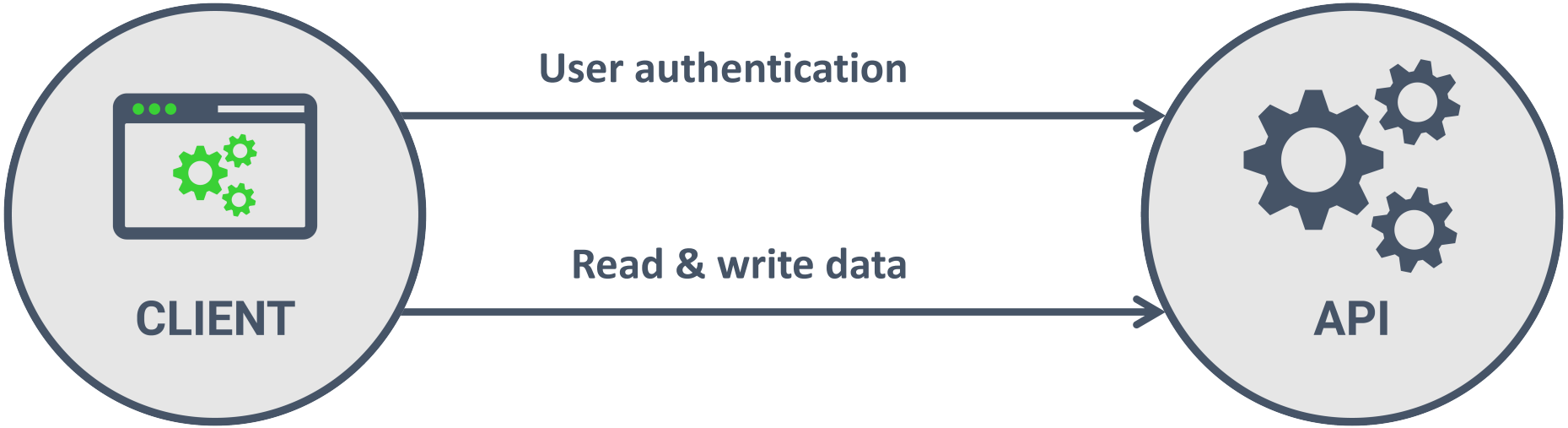
Custom courses covering web security, API security, Angular security, …

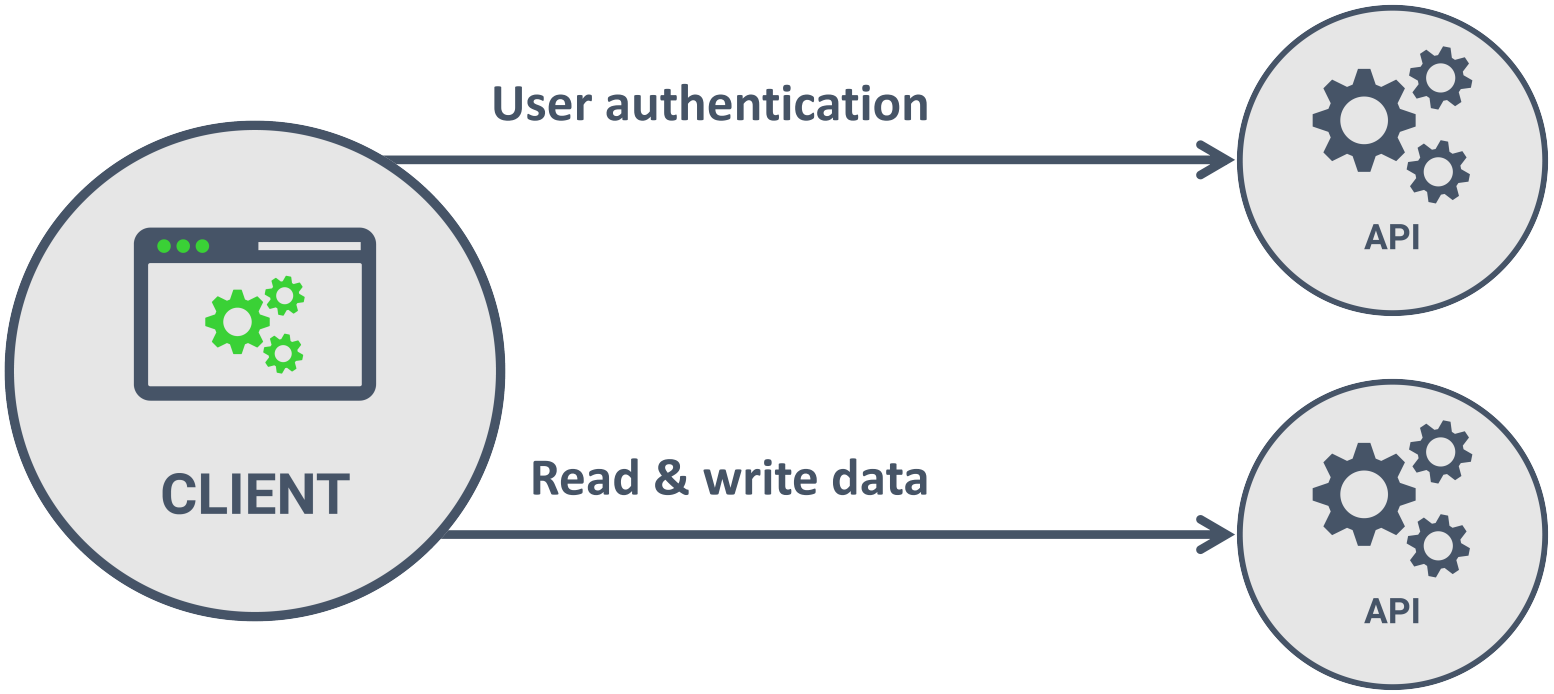
Consulting services on security, Oauth 2.0, OpenID Connect, …

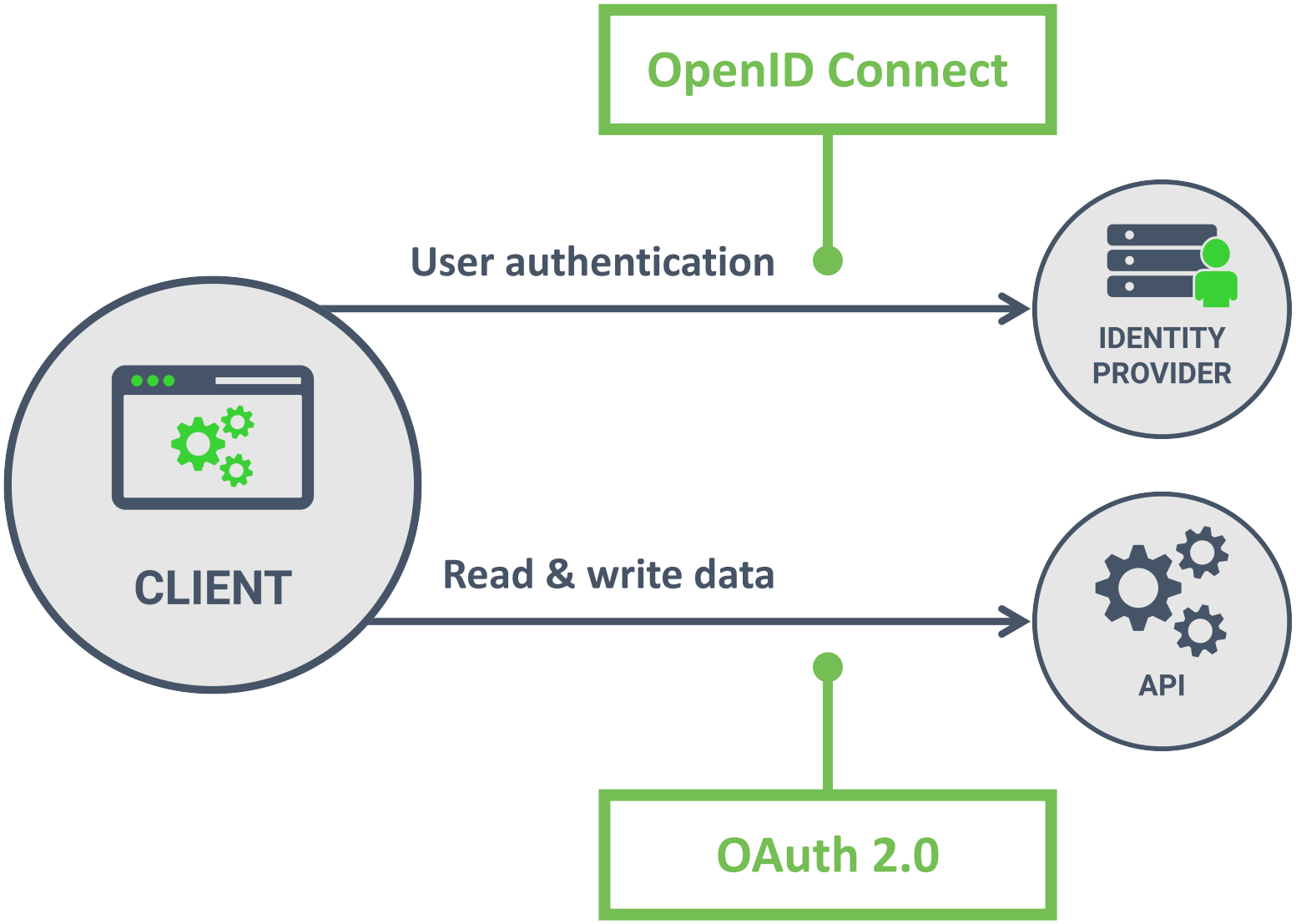
@PhilippeDeRyck

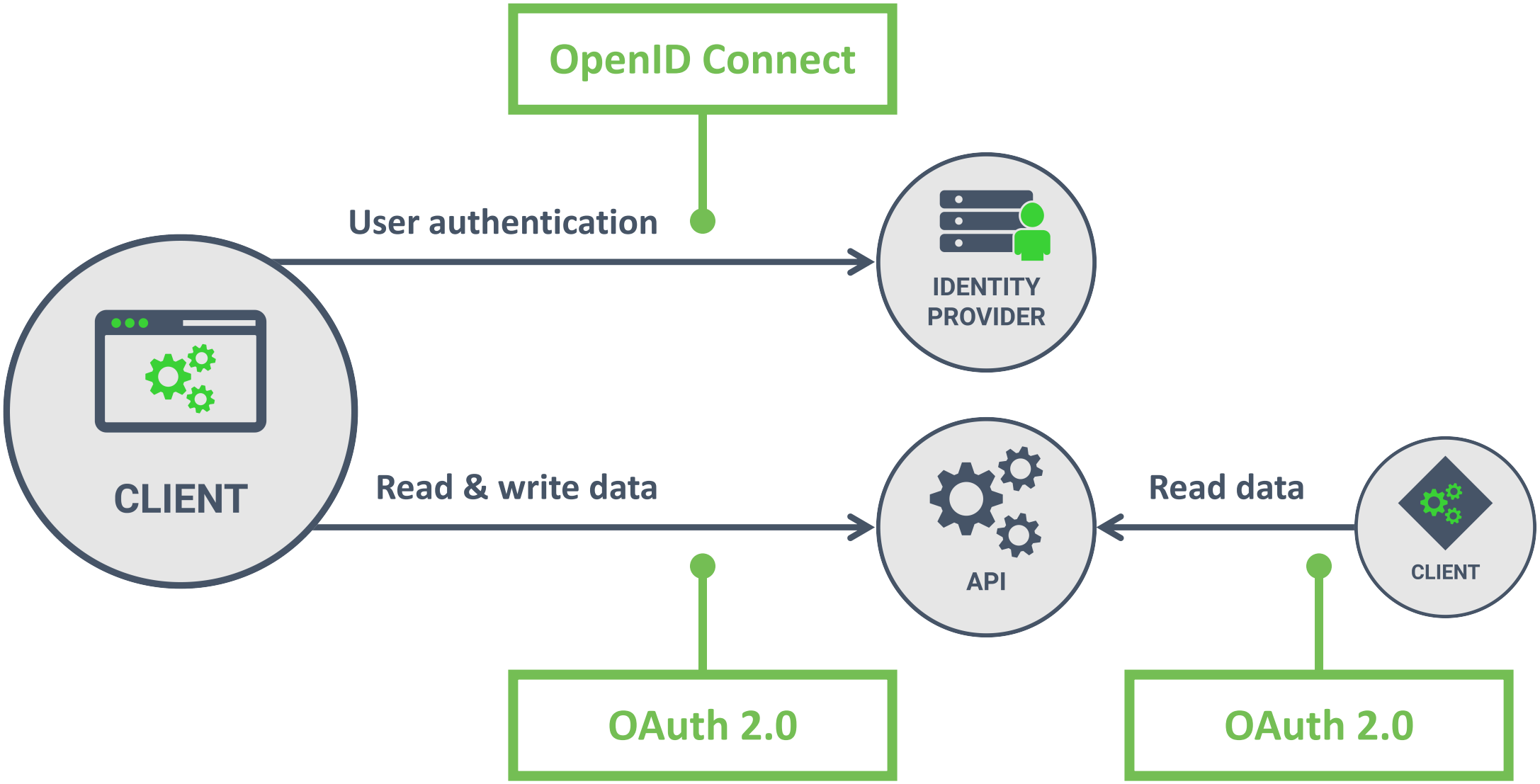
https://PragmaticWebSecurity.com

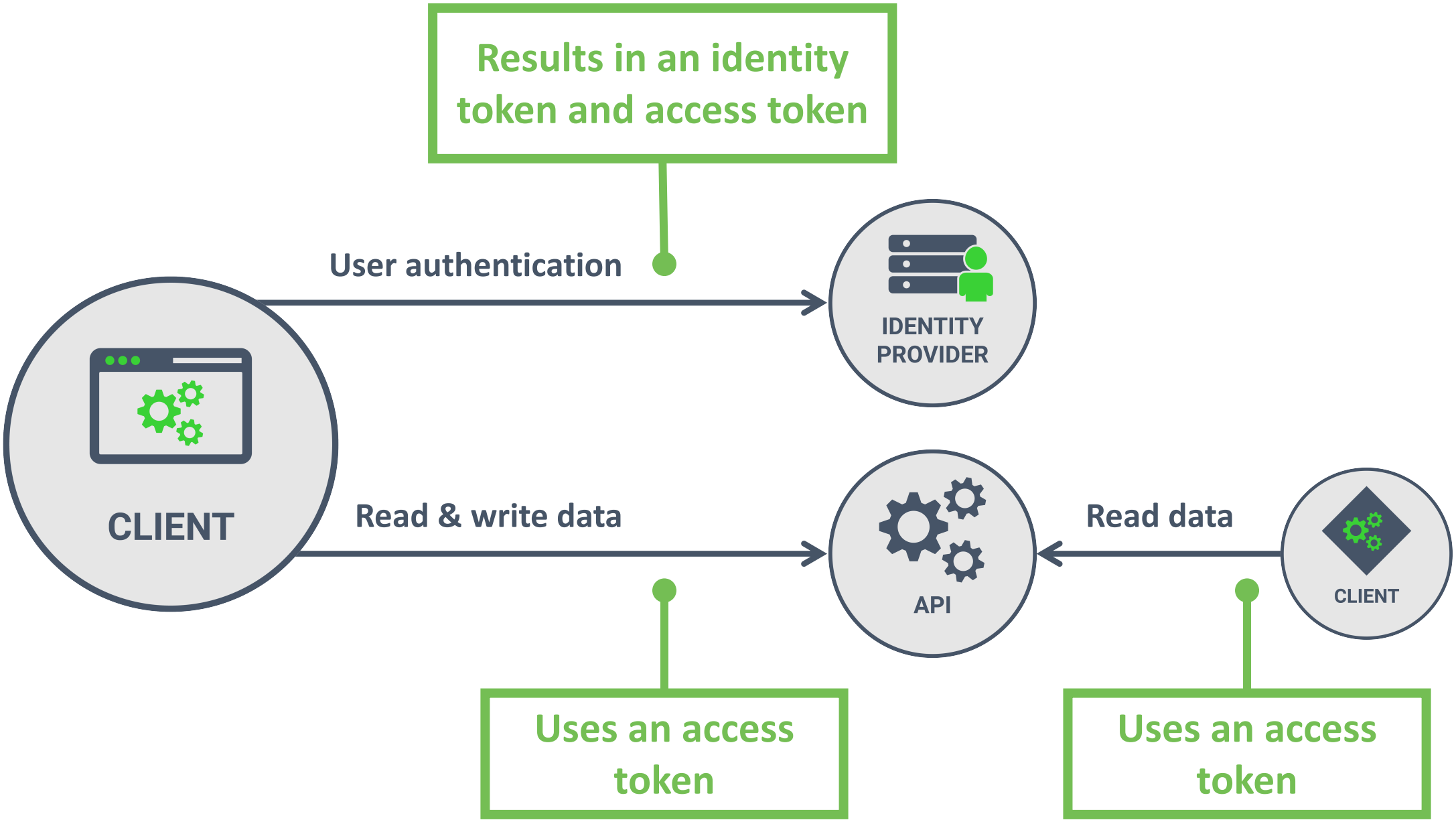

User authentication
Read & write data
BROWSER
BACKEND


User authentication
Read & write data
CLIENT
API


User authentication
CLIENT
API
Read & write data
API


OpenID Connect
User authentication
IDENTITY PROVIDER
CLIENT
Read & write data
API
OAuth 2.0


OpenID Connect
User authentication
IDENTITY PROVIDER
CLIENT
Read & write data
API
Read data
CLIENT
OAuth 2.0
OAuth 2.0


Results in an identity token and access token
User authentication
IDENTITY PROVIDER
CLIENT
Read & write data
API
Read data
CLIENT
Uses an access token
Uses an access token

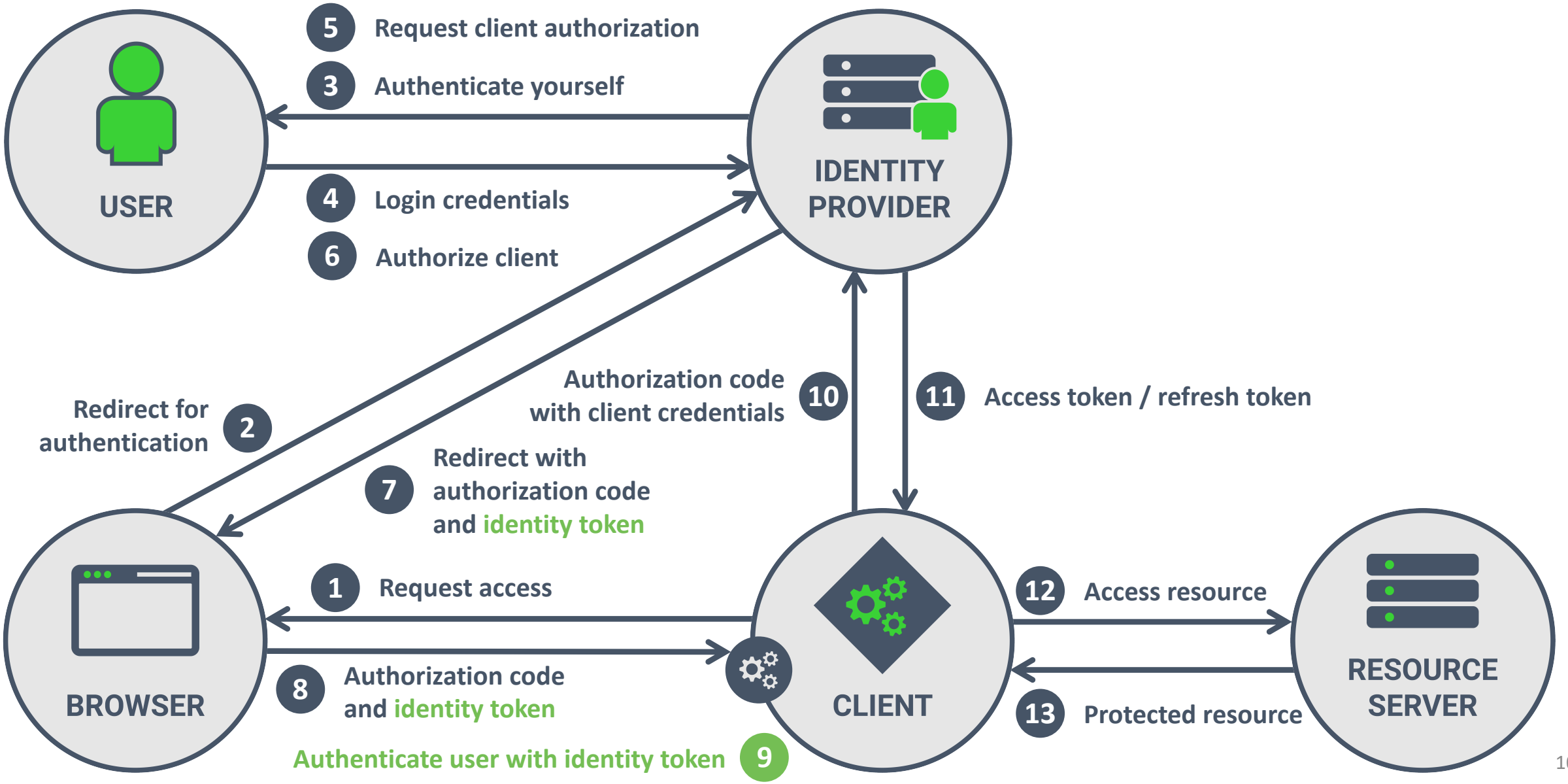
OAuth 2.0 and OpenID Connect

OpenID Connect provides user authentication

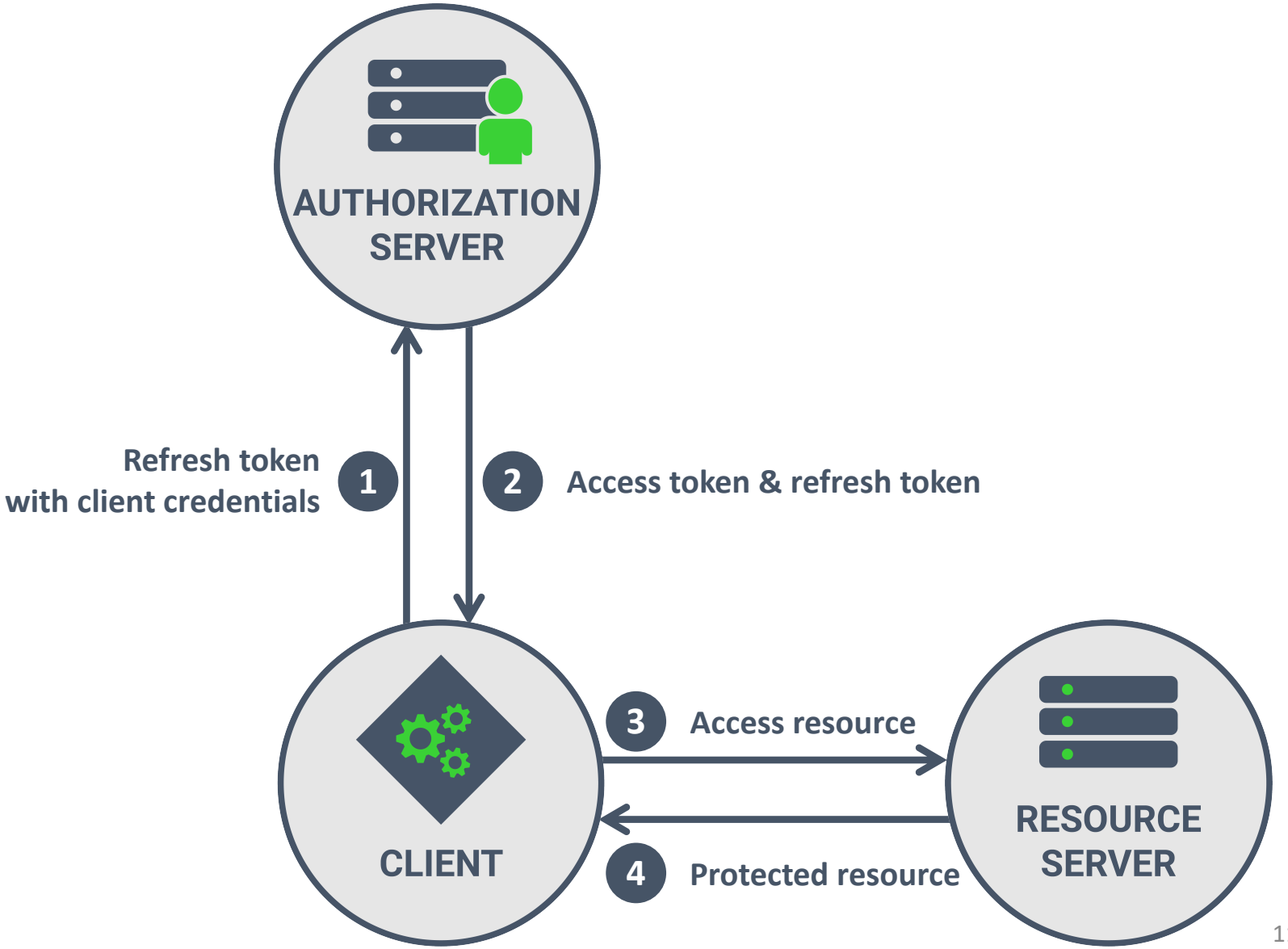
OAuth 2.0 allows a client to access resources on behalf of the user

Modern applications use a combination of both protocols

THE OIDC HYBRID FLOW



The Refresh Token flow



The OIDC Hybrid flow

- Clients are backend applications running in a "secure" environment

- The hybrid flow returns an identity token, access token and refresh token
  - Identity tokens are issued through the frontchannel, along with an authorization code
  - The authorization code can be exchanged for an access token and refresh token
  - Using the authorization code requires client authentication

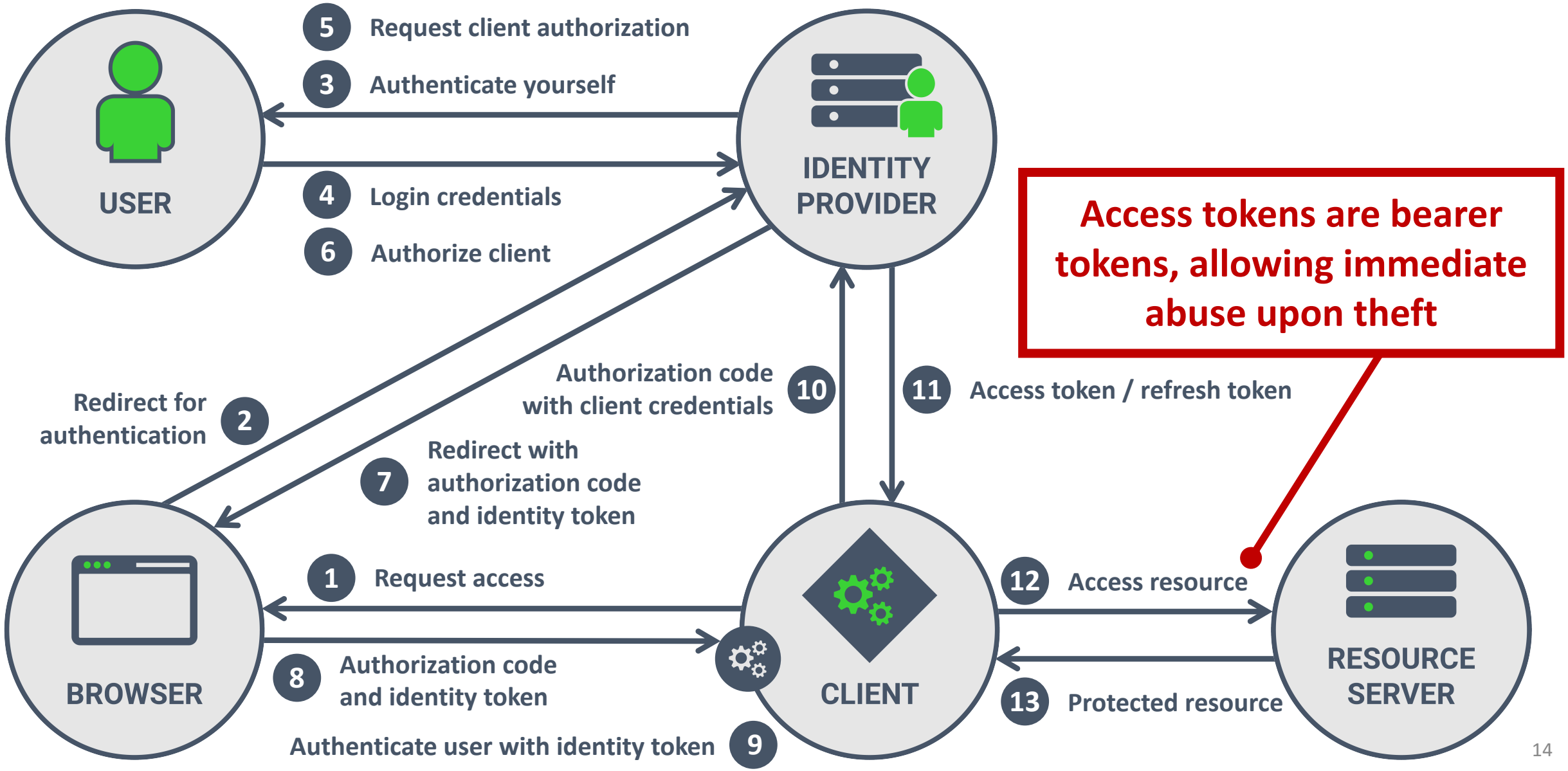
- Refresh tokens allow the client to obtain a new access token
  - Using a refresh token requires client authentication

Buffer security breach has been resolved – here is what you need to know

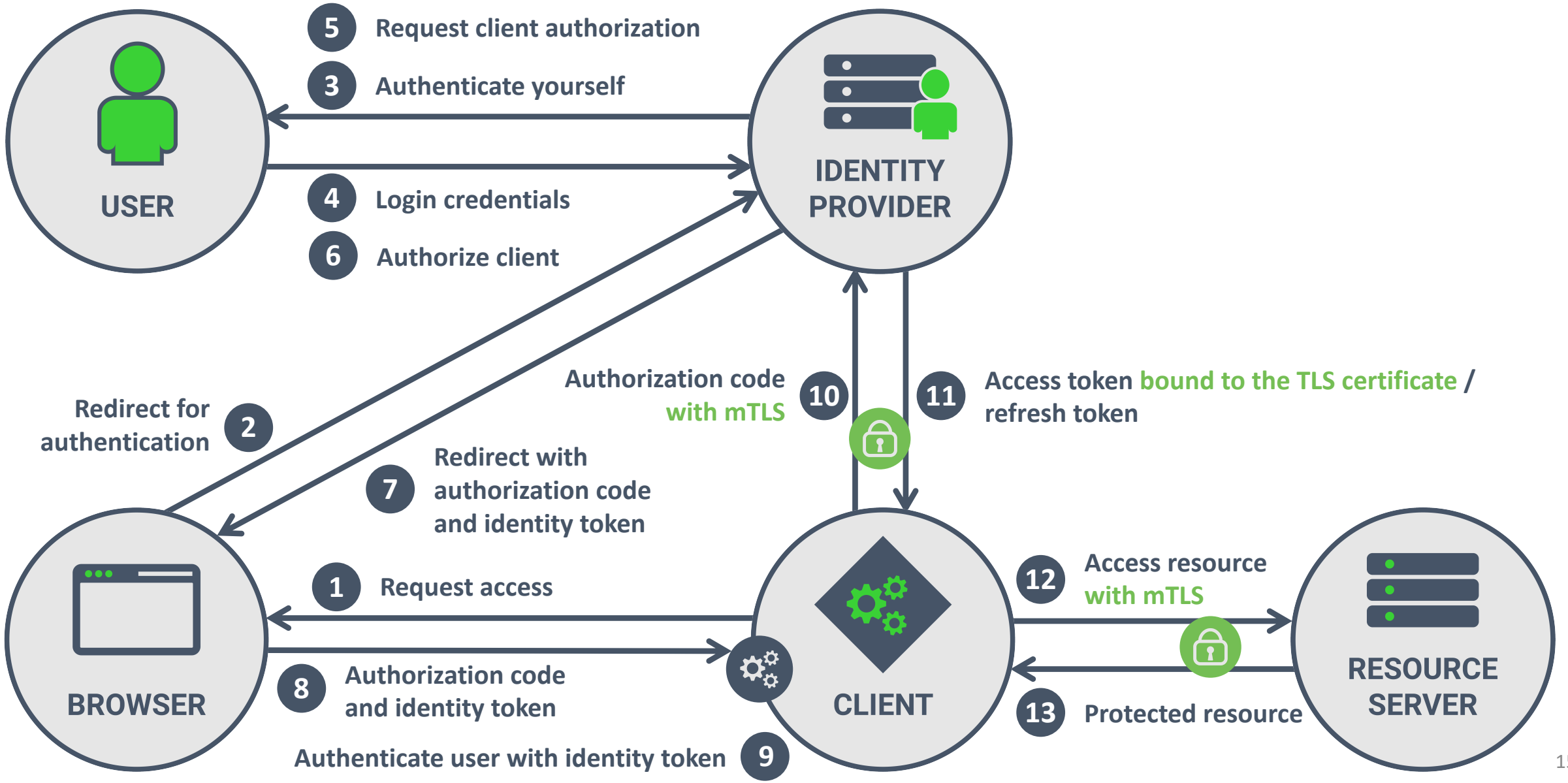
by Joel Gascoigne

> The hackers were able to steal some of our Facebook and Twitter access tokens from our users.

THE DANGER OF BEARER TOKENS



BINDING TOKENS TO TLS CERTIFICATES



```
{

   "sub": "jdoe@example.com",
   "aud": "https://api.example.com",
   "azp": "RandomClientID",
   "iss": "https://authorizationserver.example.com/",
   "exp": 1419356238,
   "iat": 1419350238,
   "scope": "read write",
   "jti": "405b4d4e-8501-4e1a-a138-ed8455cd1d47",
   "cnf":{
     "x5t#S256": "bwcK0esc3ACC3DB2Y5_lESsXE8o9ltc05O89jdN-dg2"
   }
}
```

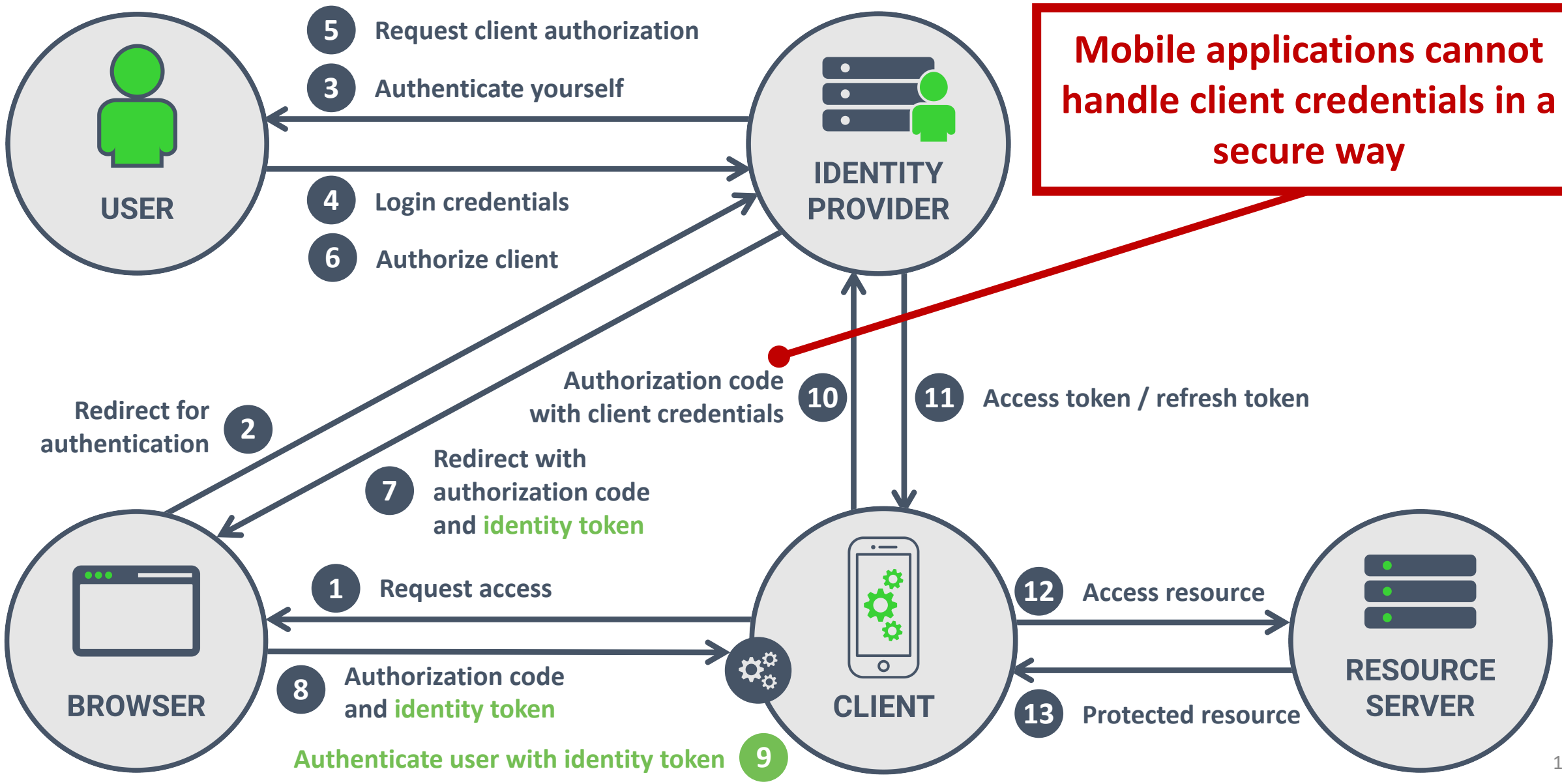
Proof-of-possession for access tokens

Many confidential clients still rely on bearer access tokens

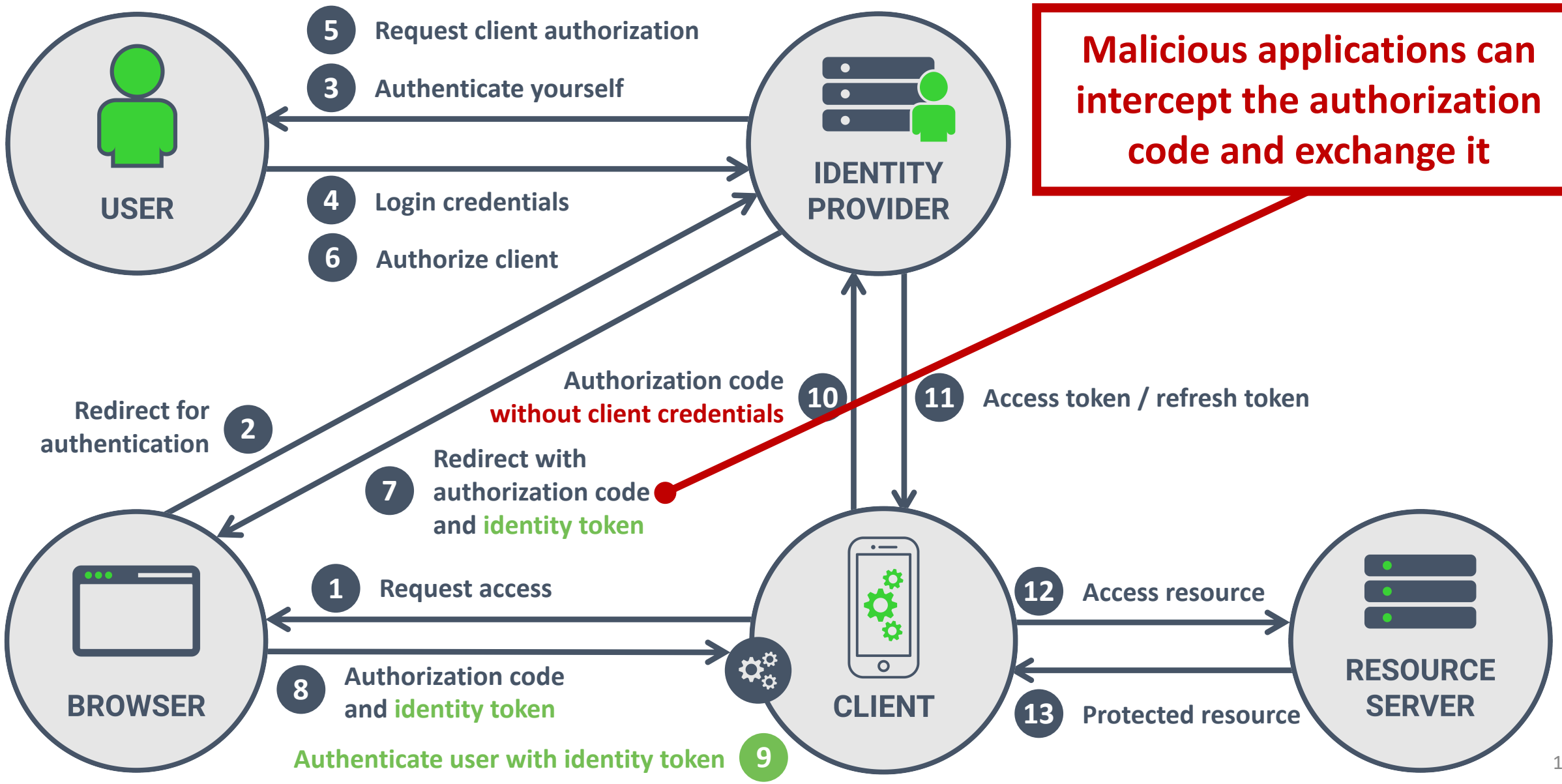
The confidential client can authenticate with a TLS certificate

The TLS certificate can be used to enable token binding

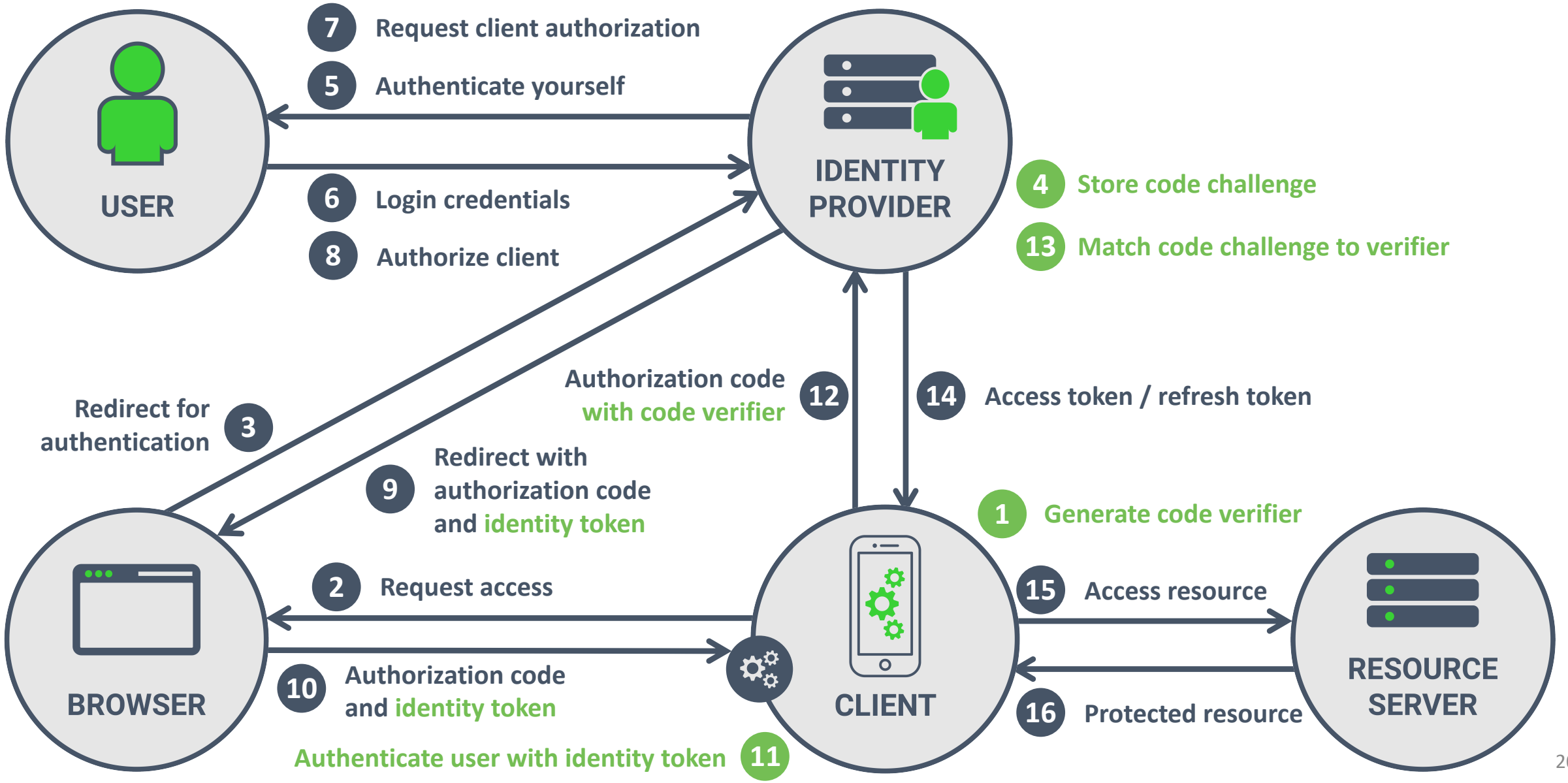
THE OIDC HYBRID FLOW



THE OIDC HYBRID FLOW



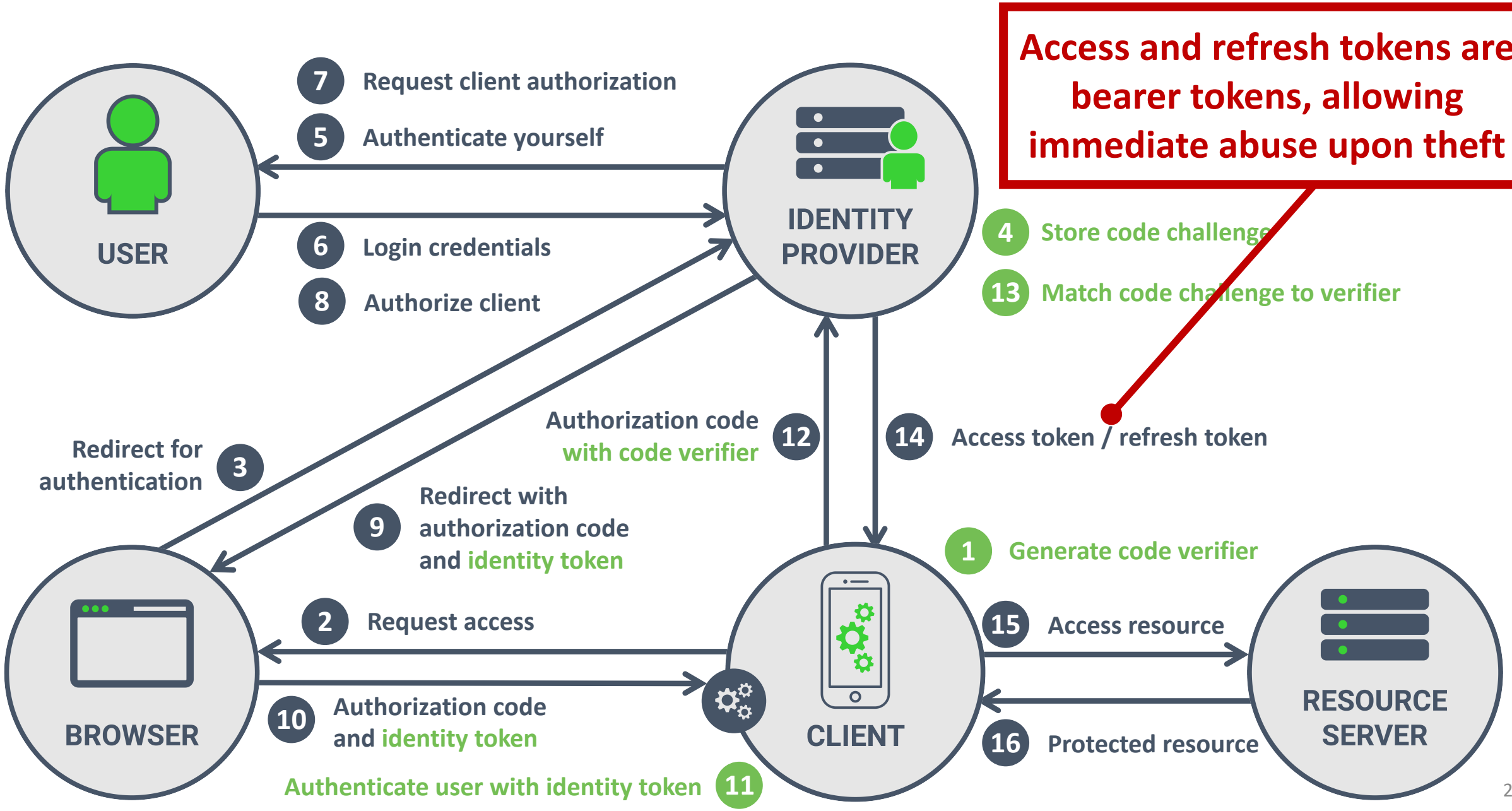
THE OIDC HYBRID FLOW WITH PKCE



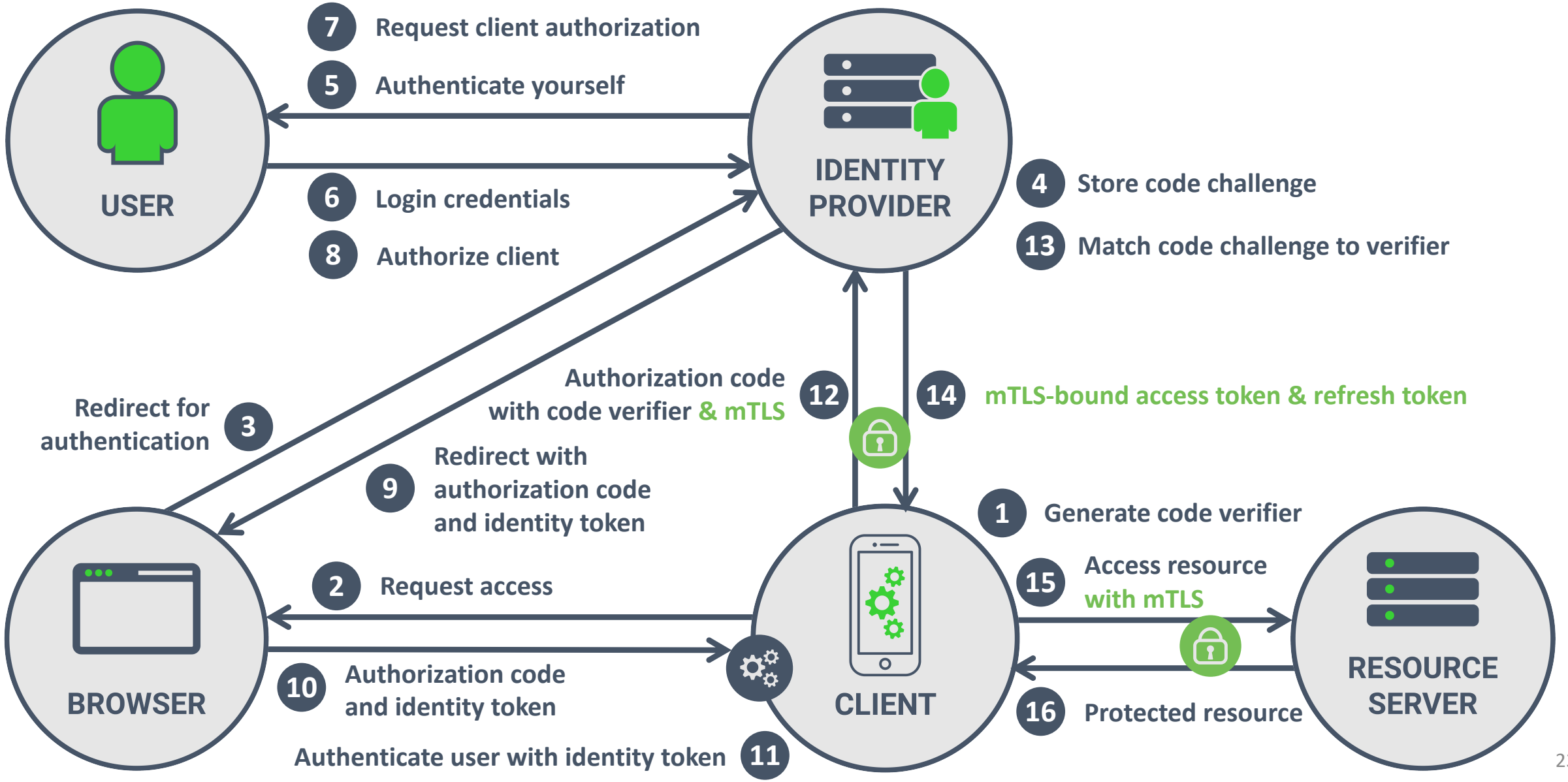
The OIDC Hybrid flow with PKCE

- Mobile applications are public clients
  - The lack of client authentication exposes the authorization code to attacks
- The Proof-Key-for-Code-Exchange addition keeps the authorization code secure
  - PKCE essentially acts as a one-time password for each individual client
  - Prevents the abuse of a stolen authorization code
- Mobile applications can use refresh tokens if they store them securely
  - Refresh tokens do not require authentication, so are bearer tokens
  - Only good place to store is in the OS's secure application storage

THE DANGER OF BEARER TOKENS



BINDING TOKENS TO TLS CERTIFICATES ON PUBLIC CLIENTS



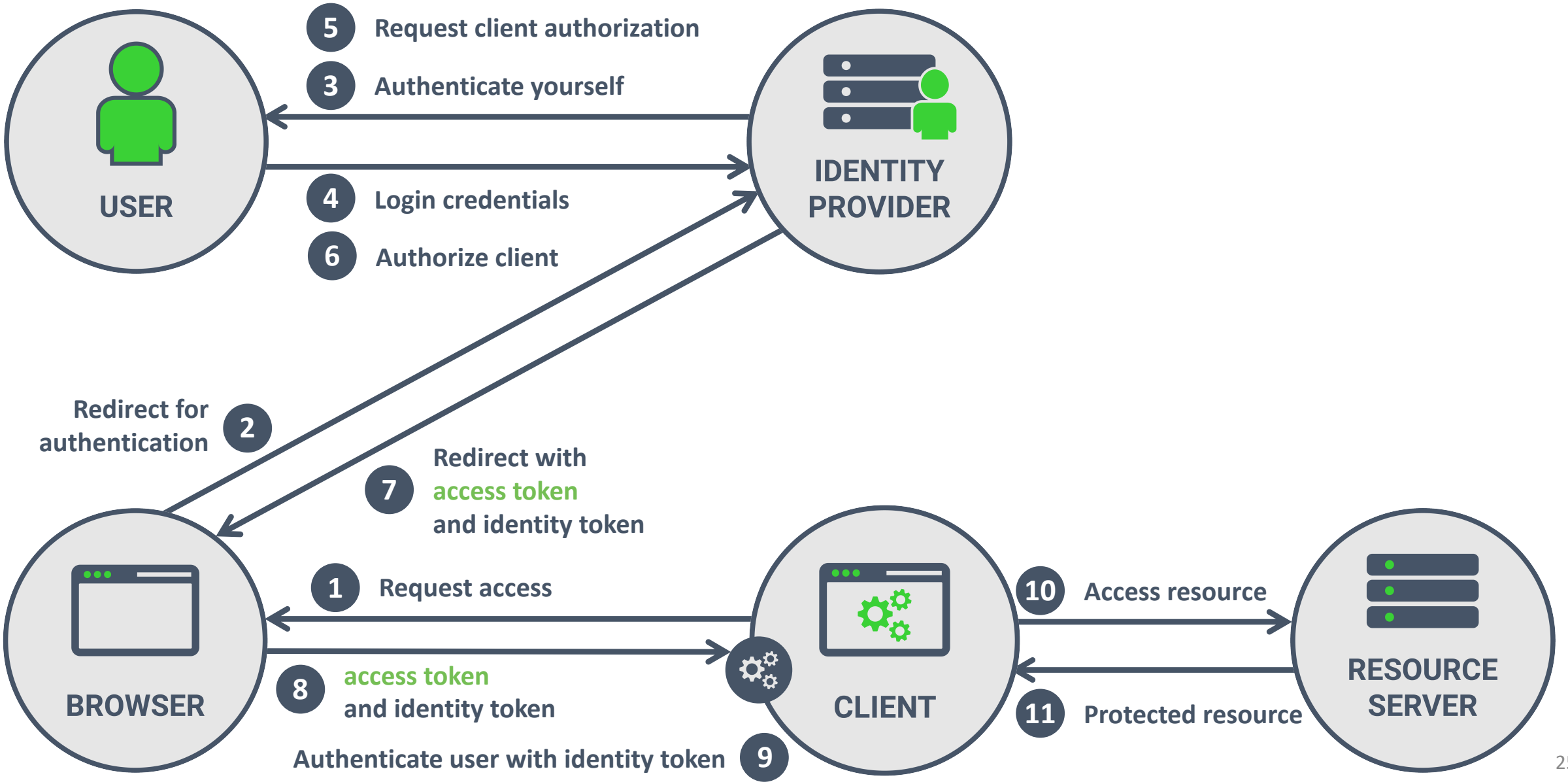
Proof-of-possession in mobile clients

Each client instance generates its own certificate

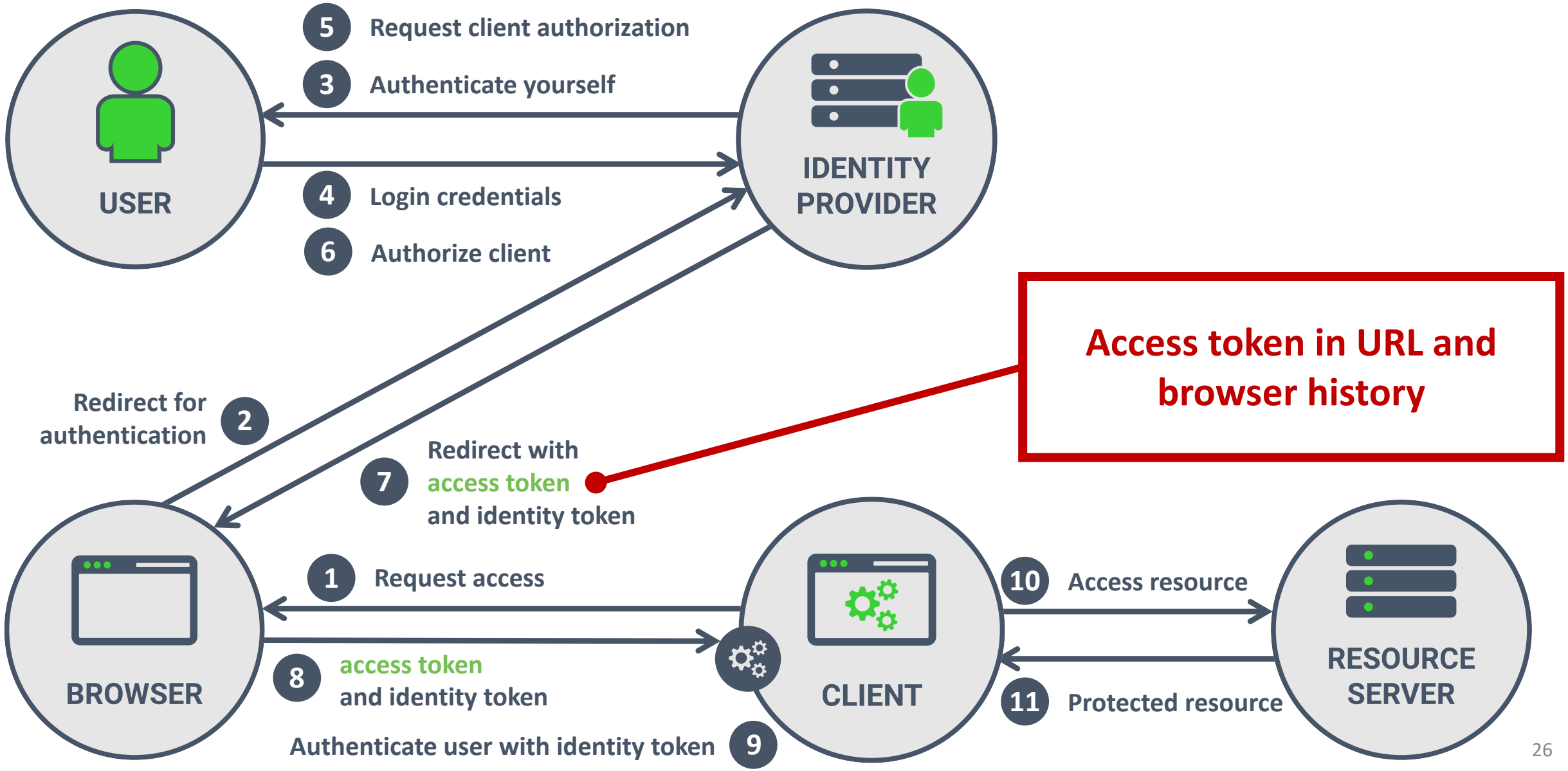
The client uses the self-signed certificate during TLS connections

The authorization server ties the tokens to the client certificate

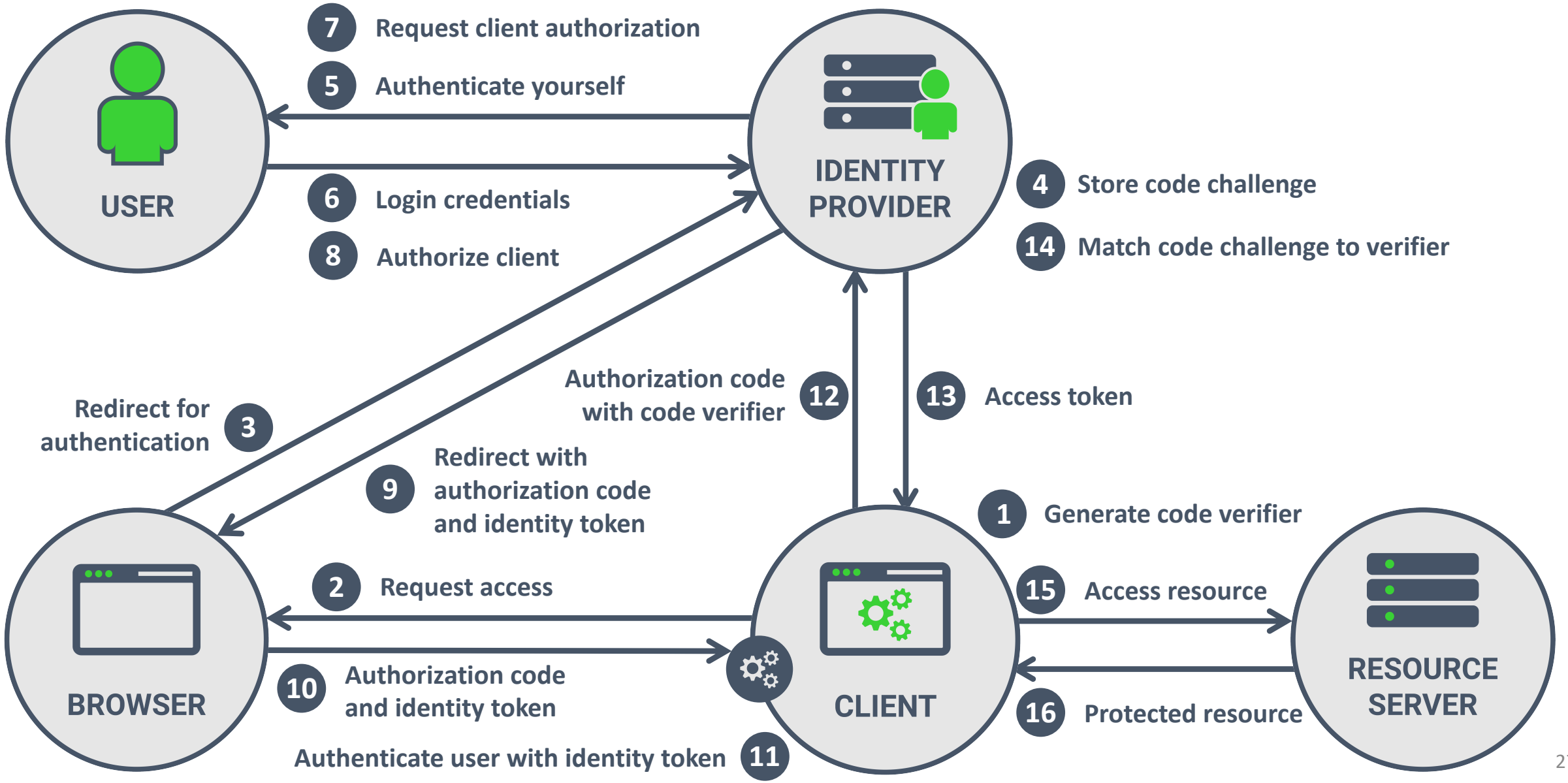
THE OIDC IMPLICIT FLOW



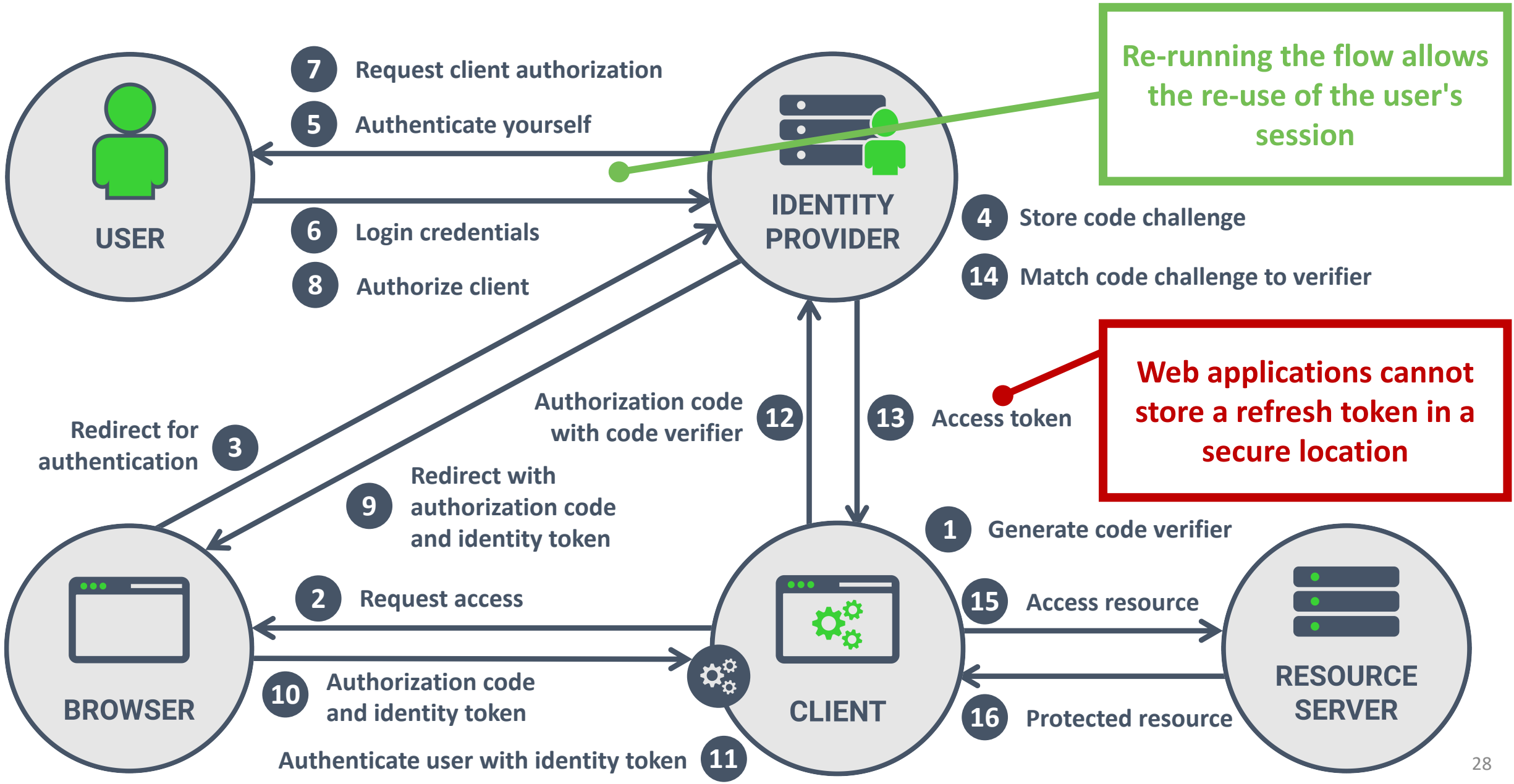
The OIDC Implicit flow



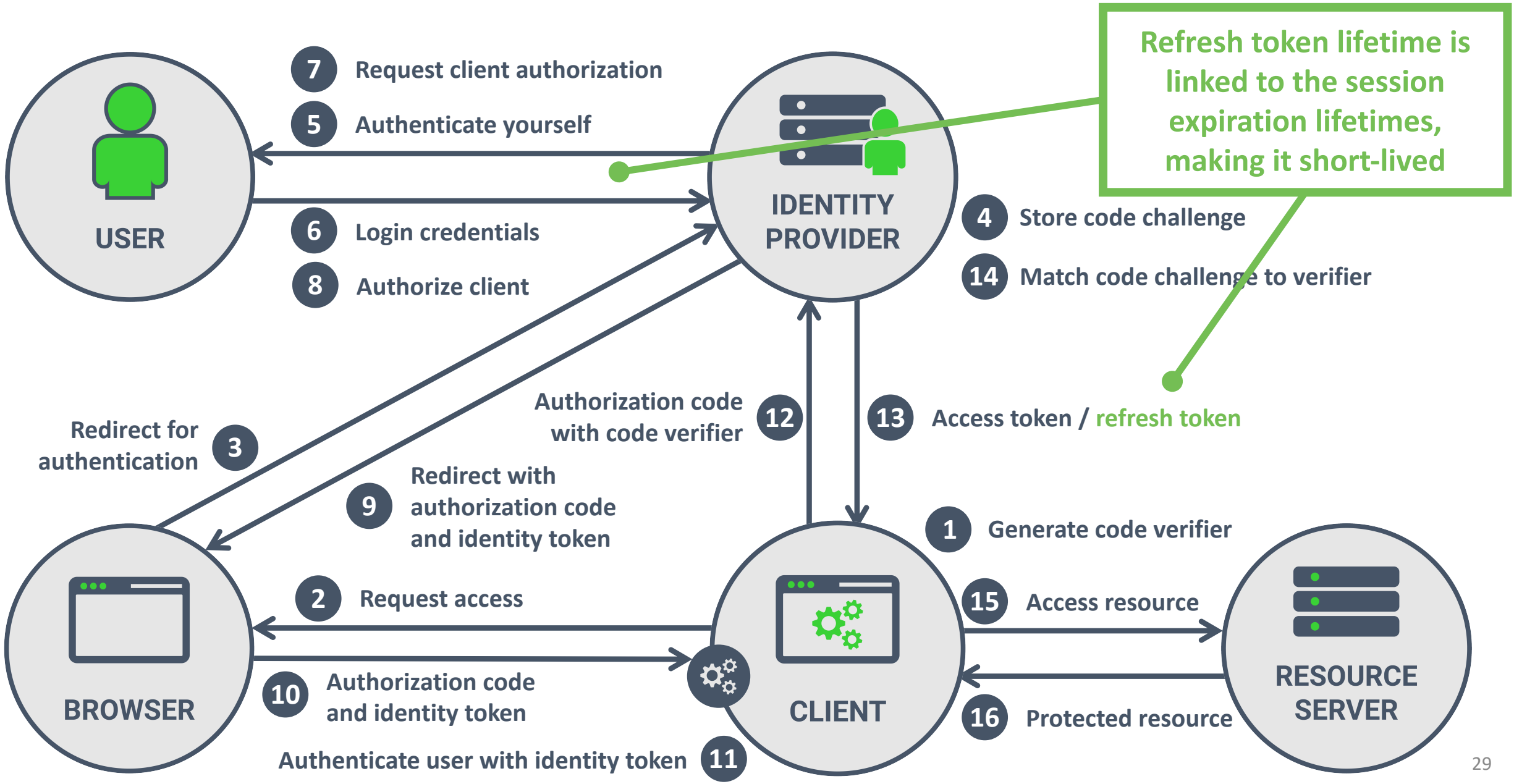
THE OIDC HYBRID FLOW WITH PKCE



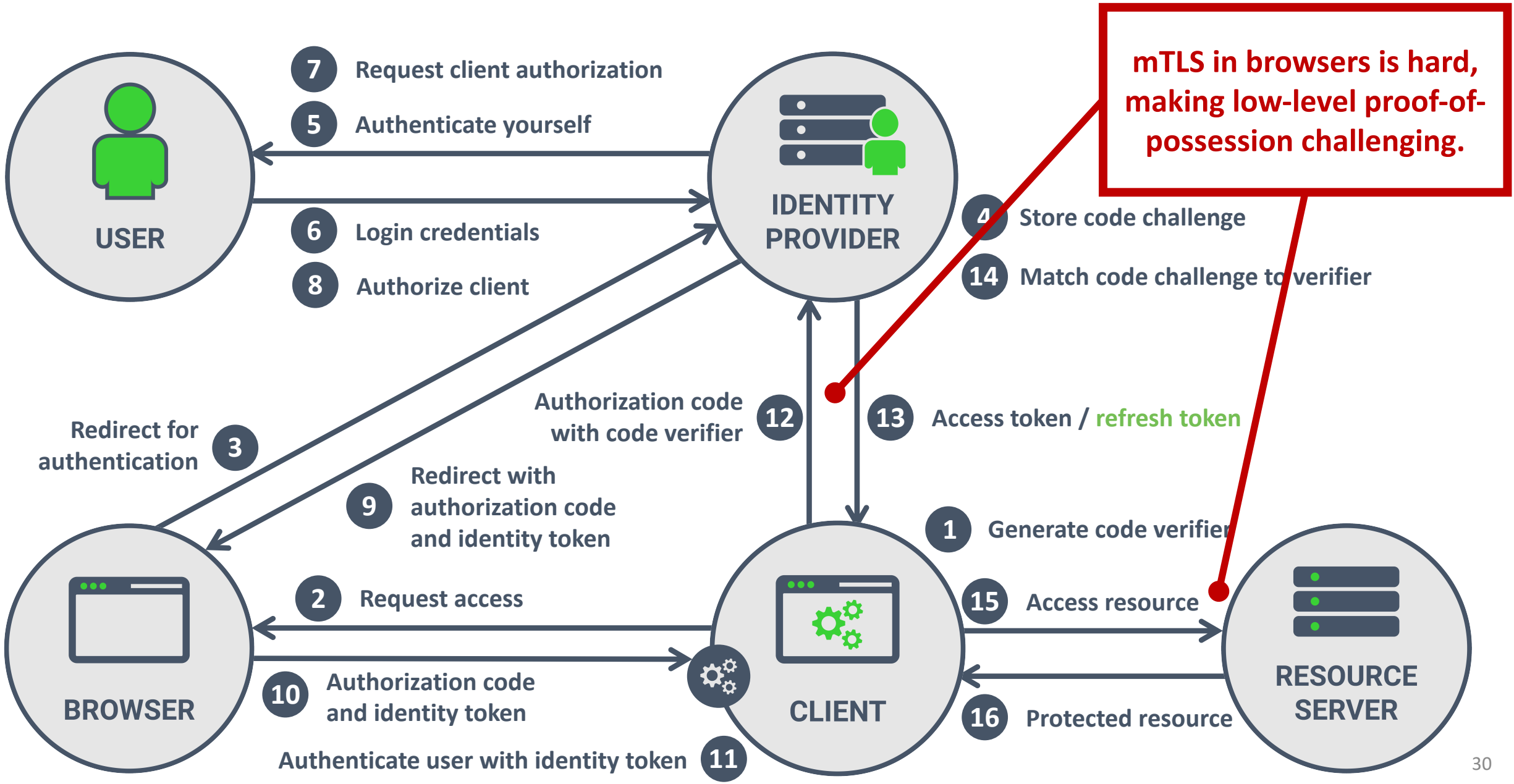
The OIDC Hybrid flow with PKCE



THE OIDC HYBRID FLOW WITH PKCE



The OIDC Hybrid flow with PKCE



WEB SECURITY IS HARD

The Hybrid flow with PKCE is recommended (Implicit flow is still OK)

Refresh tokens cannot be used, unless they are short-lived

PoP tokens for web applications require application-level code

References

Proof Key for Code Exchange by OAuth Public Clients

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

OAuth 2.0 Security Best Current Practice

https://tools.ietf.org/html/draft-ietf-oauth-security-topics-13

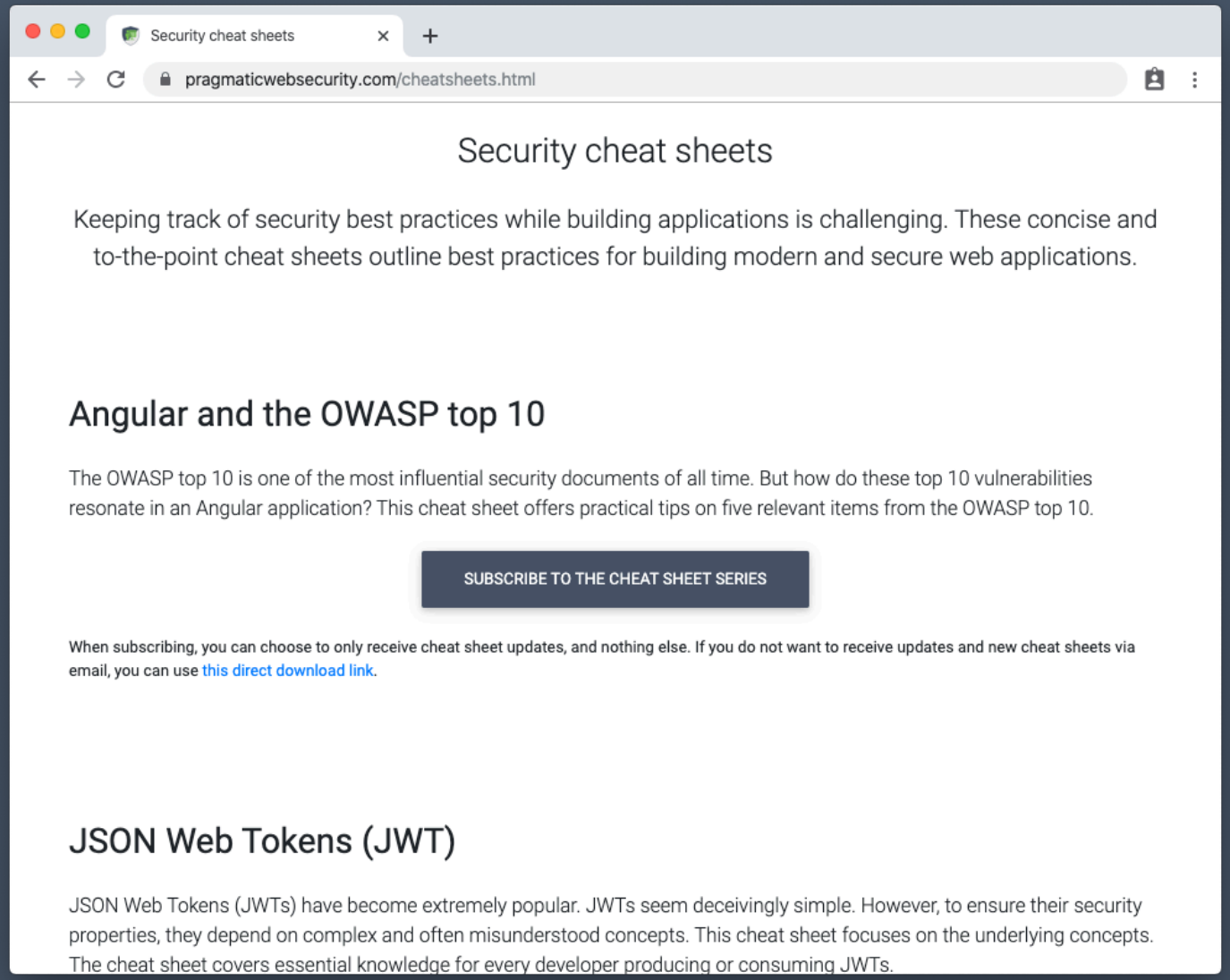
OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens

https://tools.ietf.org/html/draft-ietf-oauth-mtls-17

OAuth 2.0 Demonstration of Proof-of-Possession at the Application Layer

https://tools.ietf.org/html/draft-fett-oauth-dpop-00

FREE SECURITY CHEAT SHEETS FOR MODERN APPLICATIONS



@PhilippeDeRyck

https://cheatsheets.pragmaticwebsecurity.com/

SecAppDev

March 9th – 13th, 2020
Leuven, Belgium

A week-long course on Secure Application Development

Taught by experts from around the world

38 in-depth lectures and 3 one-day workshops

https://secappdev.org

A yearly initiative from the SecAppDev.org non-profit, since 2005

THANK YOU!

Follow me on Twitter to stay up to date on web security best practices

@PhilippeDeRyck