Introduction to OAuth 2.0 and OpenID Connect

Dr. Philippe De Ryck

https://Pragmatic Web Security.com

Internet Engineering Task Force (IETF)
Request for Comments: 6749
Obsoletes: 5849
Category: Standards Track
ISSN: 2070-1721

D. Hardt, Ed.
Microsoft
October 2012

The OAuth 2.0 Authorization Framework

Abstract

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. This specification replaces and obsoletes the OAuth 1.0 protocol described in RFC 5849.

Internet Engineering Task Force (IETF)
Request for Comments: 8252
BCP: 212
Updates: 6749
Category: Best Current Practice
ISSN: 2070-1721

OAuth 2.0 for Native Apps

Abstract

OAuth 2.0 authorization requests from native
through external user-agents, primarily the
specification details the security and usa
the case and how native apps and authoriz
this best practice.

W. Denniss
Google
J. Bradley
Identity
2017

Web Authorization Protocol
Internet-Draft
Intended status: Best Current Practi
Expires: 29 January 2023

OAuth 2.0 Security Best C
draft-ietf-oauth-securit

Abstract

This document describes best current security practice for OAuth 2.0.
It updates and extends the OAuth 2.0 Security Threat Model to
incorporate practical experiences gathered since OAuth 2.0 was
published and covers new threats relevant due to the broader
application of OAuth 2.0.

Final

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

OpenID Connect Core 1.0 incorporating errata set 1

Abstract

OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 protocol. It enables Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner.

This specification defines the core OpenID Connect functionality: authentication built on top of OAuth 2.0 and the use of Claims to communicate information about the End-User. It also describes the security and privacy considerations for using OpenID Connect.

Abstract

The OAu
applica
behalf
betwee

ISSN: 2070-1721

The OAuth 2.0 Authorization Framework: Bearer Toke

Abstract

This specification descri
requests to

use bearer tokens
ted resources. Any
arer") can use it t
emonstrating posses
se, bearer tokens
and in transport.

Network Working Group
Internet-Draft
Intended status: Best Current Practice
Expires: 8 September 2022

A. Parecki
Okta
D. Waite
Ping Identity
7 March 2022

OAuth 2.0 for Browser-Based Apps
draft-ietf-oauth-browser-based-apps-09

Abstract

This specification details the security considerations and best practices that must be taken into account when developing browser-based applications that use OAuth 2.0.

Final

	B. de Medeiros
	Google
	N. Agarwal
	Microsoft
	N. Sakimura
	NAT.Consulting
	J. Bradley
	Yubico
	M. Jones
	Microsoft
	September 12, 2022

TOC

OpenID Connect Session Management 1.0

Abstract

OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 protocol. It enables Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner.

This document describes how to manage sessions for OpenID Connect, including when to log out the End-User.

Photo by Andrew Umansky on Unsplash

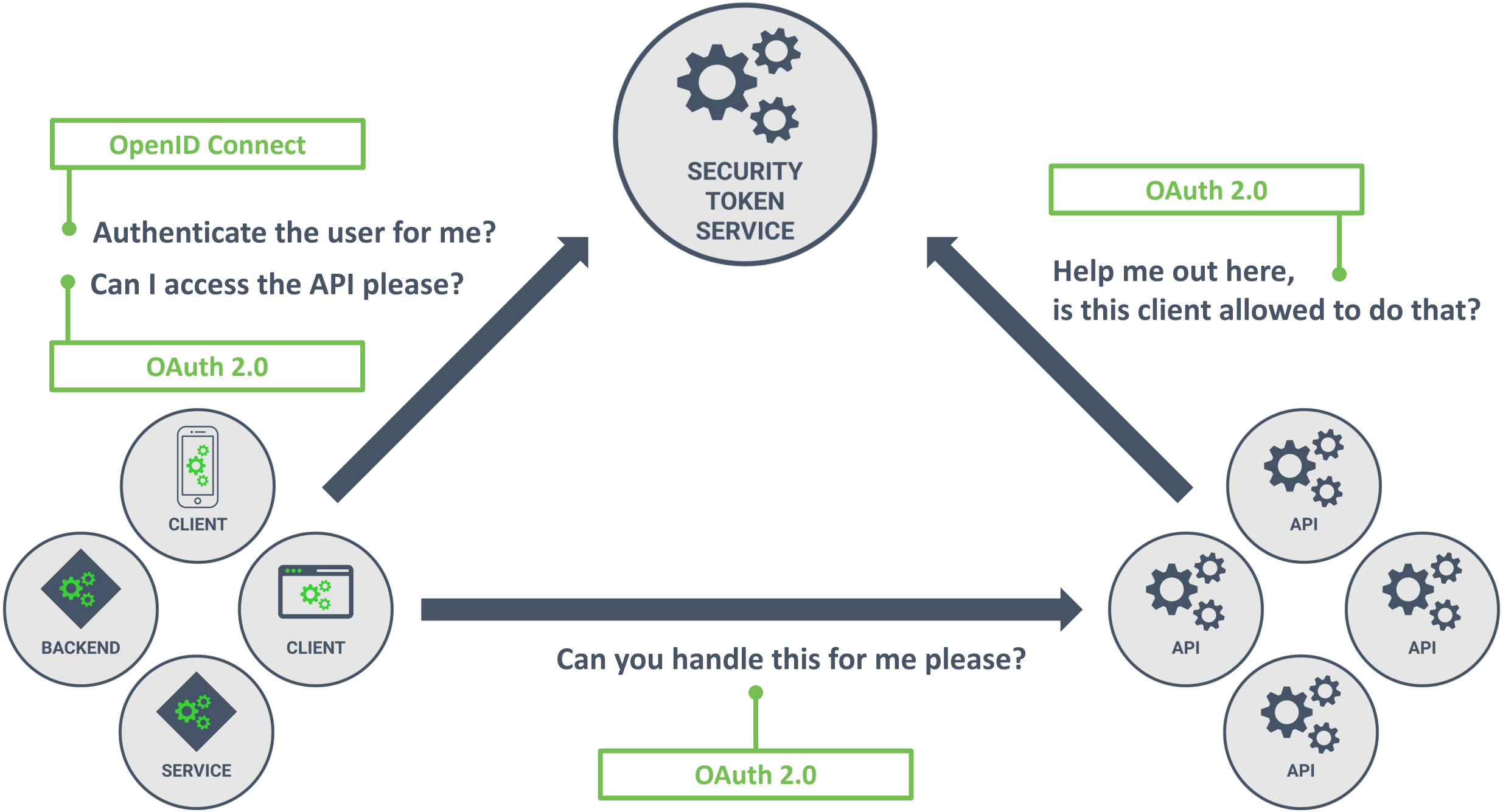
Workgroup: OAuth Working Group
Internet-Draft: draft-ietf-oauth-v2-1-09
Published: 10 July 2023
Intended Status: Standards Track
Expires: 11 January 2024

D. Hardt
Hellō
A. Parecki
Okta
T. Lodderstedt
yes.com

The OAuth 2.1 Authorization Framework

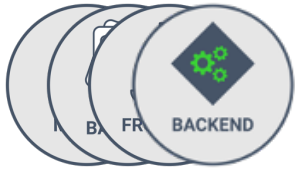
Abstract

The OAuth 2.1 authorization framework enables an application to obtain limited access to a protected resource, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and an authorization service, or by allowing the application to obtain access on its own behalf. This specification replaces and obsoletes the OAuth 2.0 Authorization Framework described in RFC 6749 and the Bearer Token Usage in RFC 6750.


OpenID Connect
Authenticate the user for me?
Can I access the API please?
OAuth 2.0
SECURITY TOKEN SERVICE
OAuth 2.0
Help me out here,
is this client allowed to do that?
CLIENT
BACKEND
CLIENT
SERVICE
Can you handle this for me please?
OAuth 2.0
API
API
API
API

pdr.online

Terminology

	This session	OAuth 2.0	OpenID Connect
	User	Resource Owner	End-User
	API	Resource Server	
	Security Token Service (STS)	Authorization Server	OpenID Provider
	Client	Client	Relying Party

I am *Dr. Philippe De Ryck*

Pragmatic Web Security
Security for developers

Founder of Pragmatic Web Security

Google Developers Experts

Google Developer Expert

AMBASSADOR PROGRAM

Auth0 Ambassador

SecAppDev

SecAppDev organizer

I help developers with security

- **Hands-on in-depth security training**
- **Advanced online security courses**
- **Security advisory services**

https://pdr.online

Grab a copy of the slides …

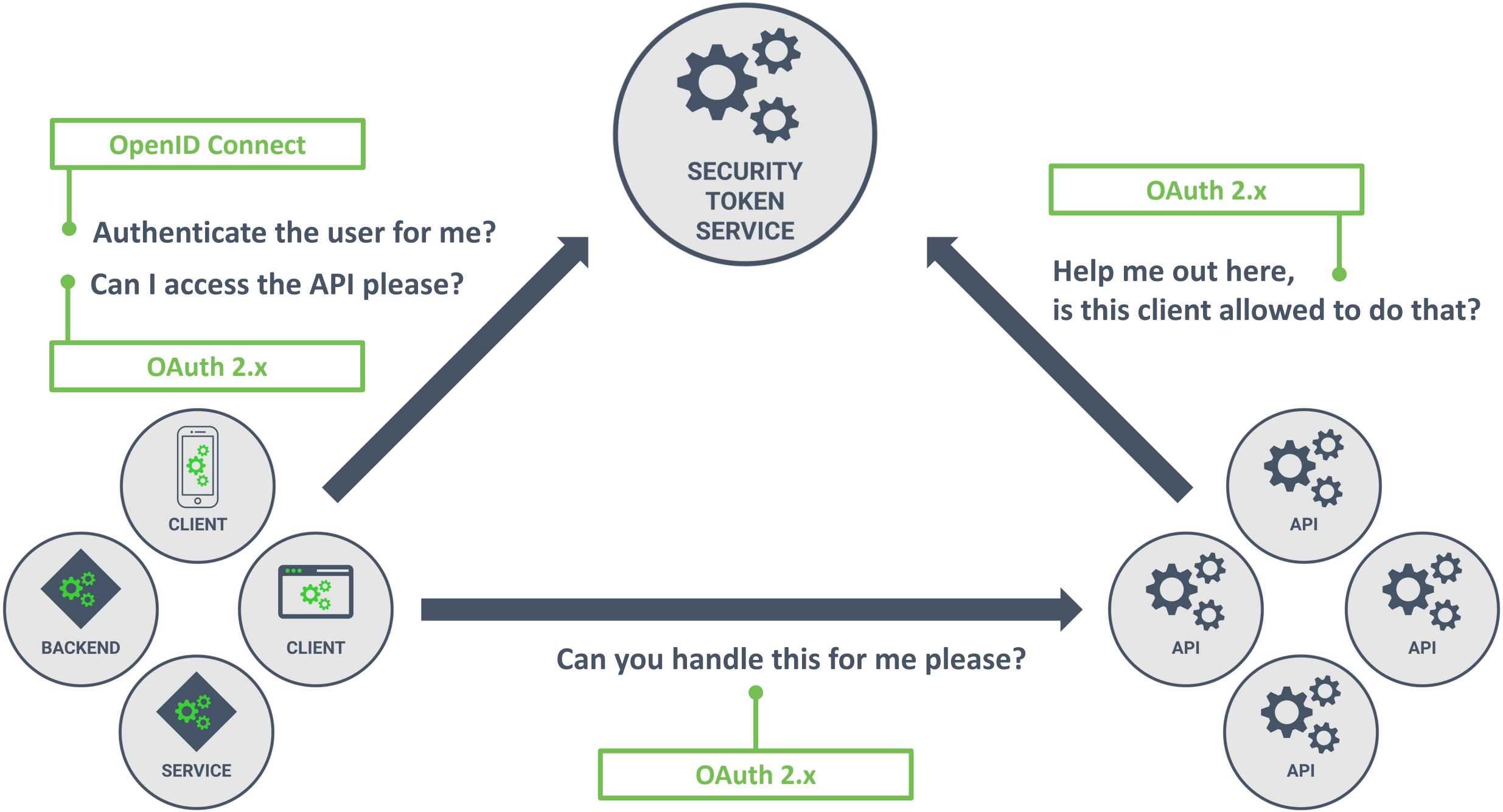
https://pragmaticwebsecurity.com/talks

/in/PhilippeDeRyck

https://infosec.exchange/@PhilippeDeRyck

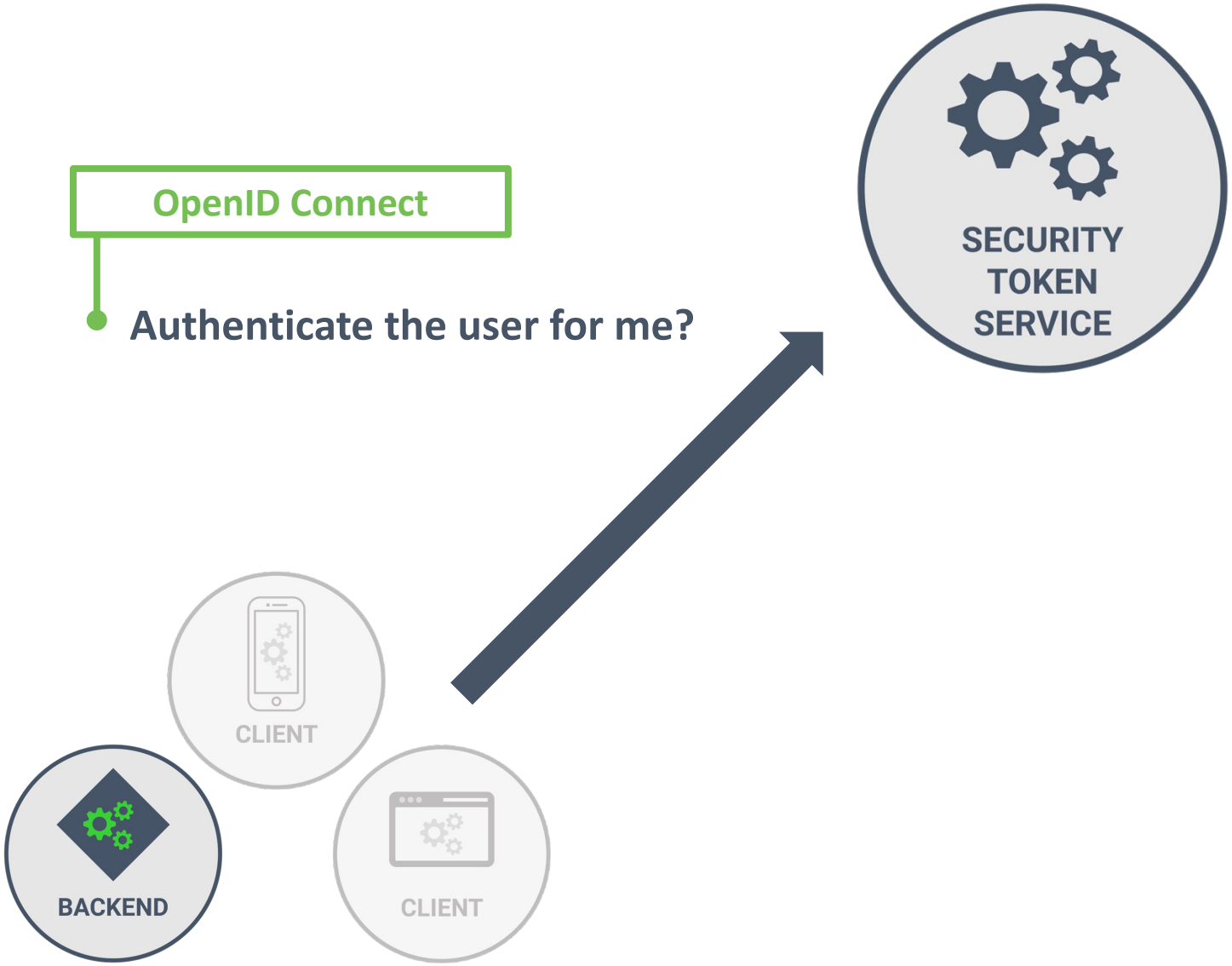
Website icons created by Uniconlabs - Flaticon

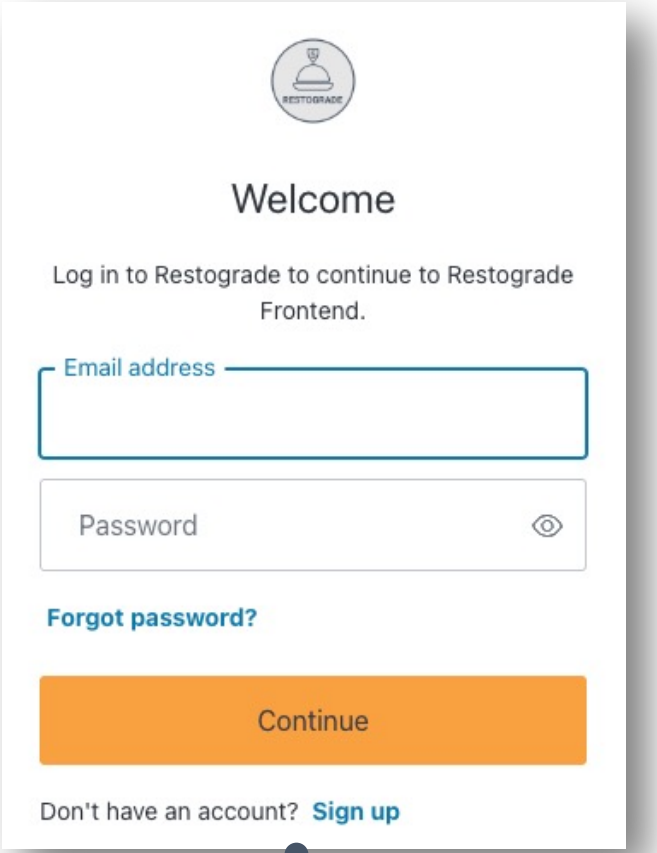
Use cases and flows


OpenID Connect
Authenticate the user for me?
Can I access the API please?
OAuth 2.x
SECURITY TOKEN SERVICE
OAuth 2.x
Help me out here,
is this client allowed to do that?
CLIENT
BACKEND
CLIENT
SERVICE
Can you handle this for me please?
OAuth 2.x
API
API
API
API

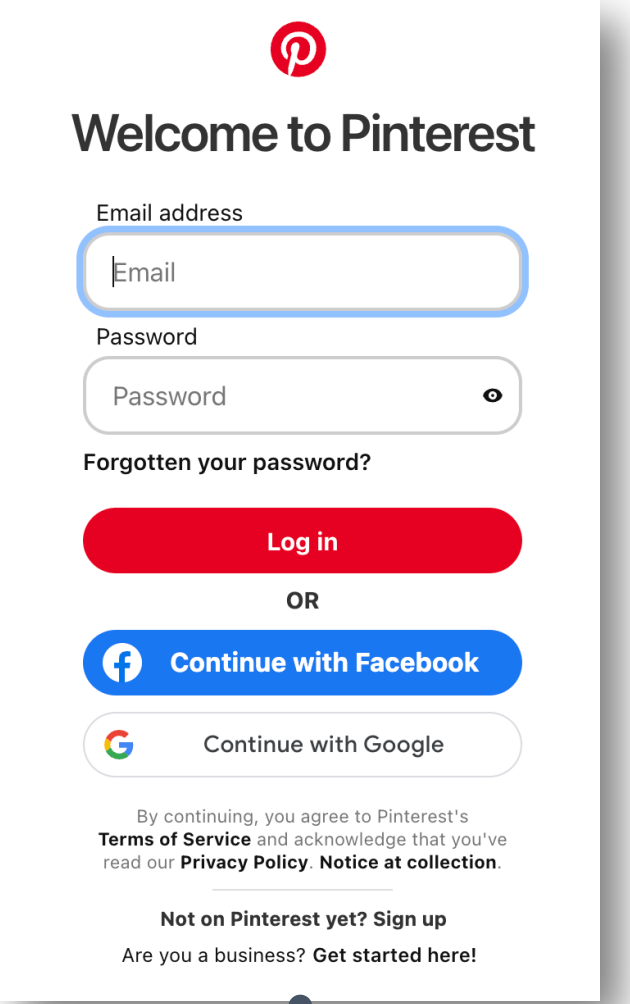
OpenID Connect

Authenticate the user for me?

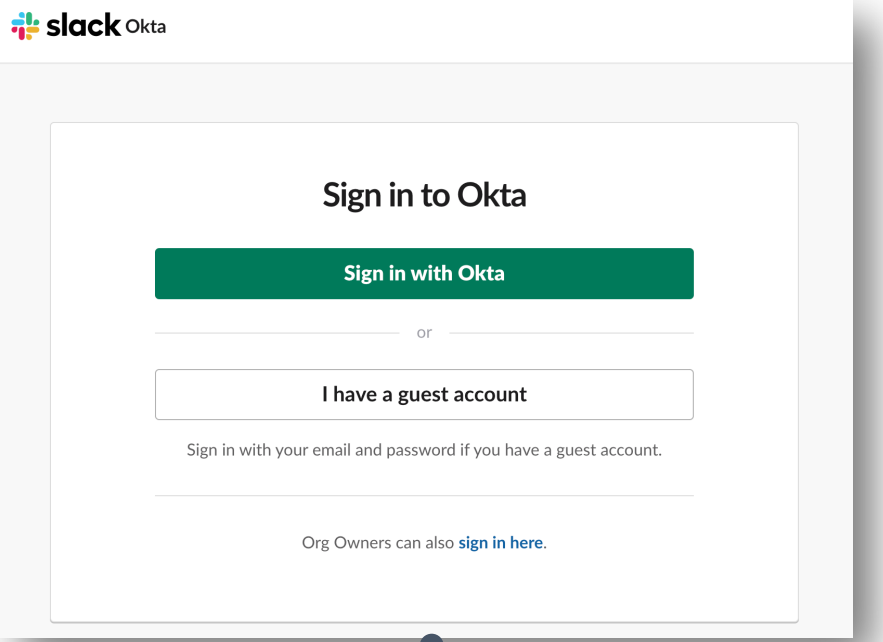




Offloading authentication to a dedicated identity provider



Implementing social login (e.g., Google, Facebook)



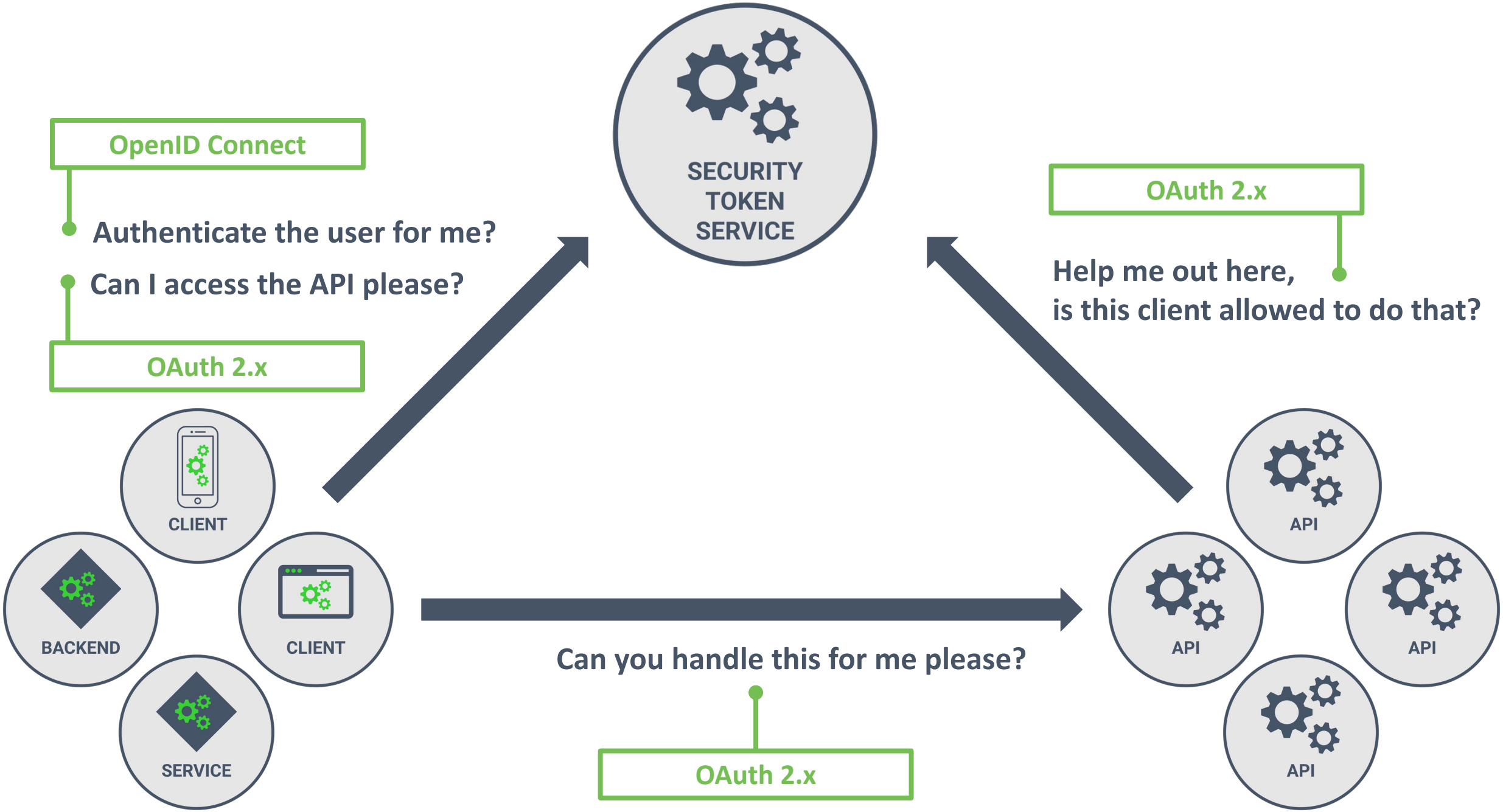
Integrating enterprise SSO in SaaS applications

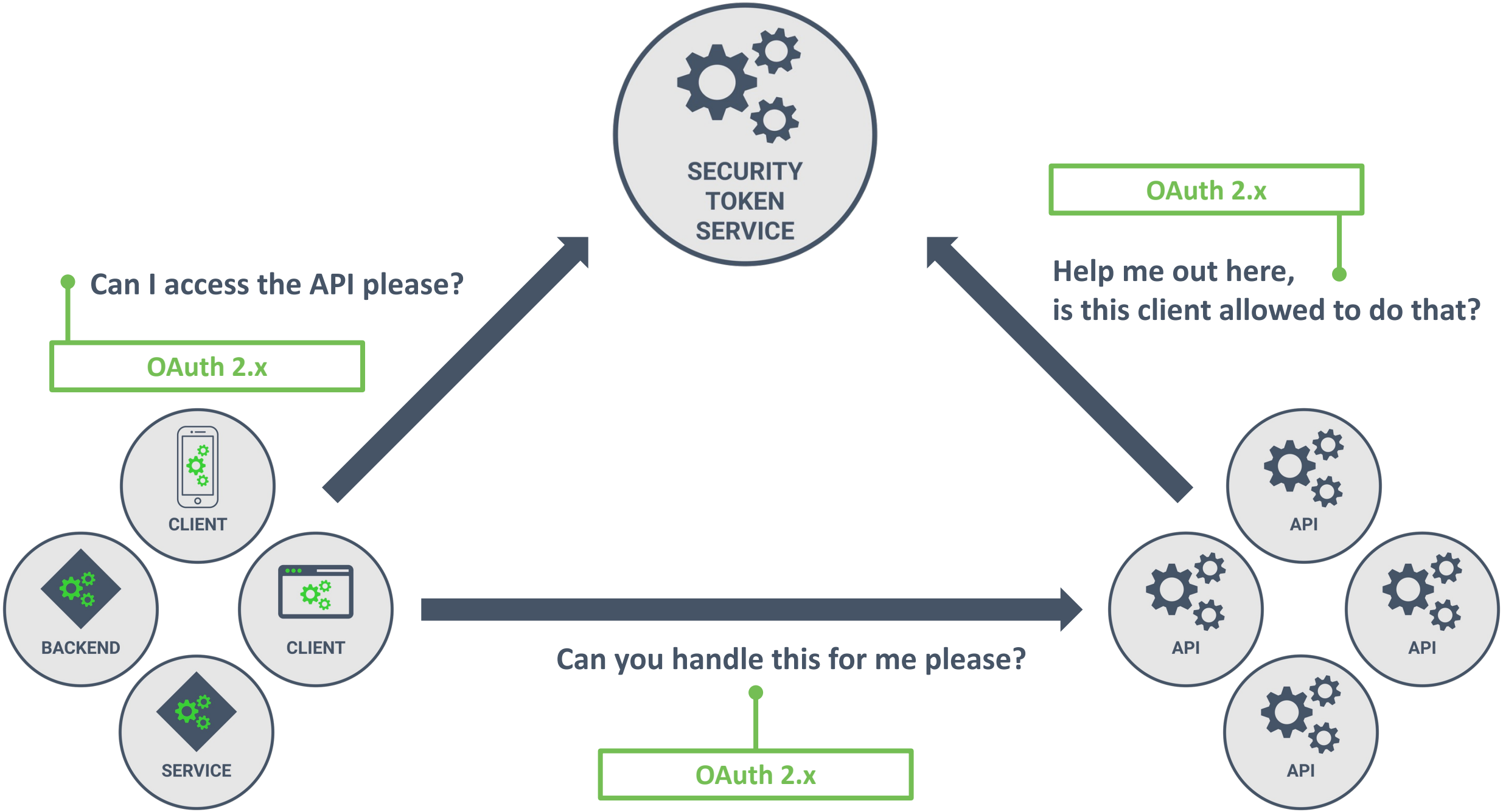
OpenID Connect in action

Overview of use cases for OAuth and OIDC

- An application wants to authenticate users using an external identity provider
  - E.g., Delegating login to an identity provider, social login (e.g., Google), or enterprise SSO
  - The client that wants to authenticate the user needs an identity token
  - ***This scenario only uses OpenID Connect***

- An application wants to use an API on behalf of the user
  - E.g., Accessing the Restograde API to read or create reviews for the user
  - The client needs an access token to make requests to the Restograde API
  - ***This scenario only uses OAuth***


OpenID Connect
Authenticate the user for me?
Can I access the API please?
OAuth 2.x
SECURITY TOKEN SERVICE
OAuth 2.x
Help me out here,
is this client allowed to do that?
CLIENT
BACKEND
CLIENT
SERVICE
Can you handle this for me please?
OAuth 2.x
API
API
API
API


SECURITY TOKEN SERVICE
OAuth 2.x
Help me out here,
is this client allowed to do that?
Can I access the API please?
OAuth 2.x
CLIENT
BACKEND
CLIENT
SERVICE
API
API
API
API
Can you handle this for me please?
OAuth 2.x

pdr.online

OAuth in action

Overview of use cases for OAuth and OIDC

- An application wants to authenticate users using an external identity provider
  - E.g., Delegating login to an identity provider, social login (e.g., Google), or enterprise SSO
  - The client that wants to authenticate the user needs an identity token
  - ***This scenario only uses OpenID Connect***

- An application wants to use an API on behalf of the user
  - E.g., Accessing the Restograde API to read or create reviews for the user
  - The client needs an access token to make requests to the Restograde API
  - ***This scenario only uses OAuth***

- An application wants to authenticate users and access APIs on their behalf
  - E.g., the Restograde mobile app authenticates the user and then accesses the API on their behalf
  - The client needs an identity token and an access token
  - ***This scenario combines OpenID Connect and OAuth***

Clients obtain tokens by running an OAuth or OIDC flow (aka *grants*)

OAUTH AND OIDC FLOWS

Implicit flow

Resource Owner Password Credentials flow

Authorization Code flow

Hybrid flow

Client Credentials flow

Device flow

Client-Initiated Backchannel Authentication flow (CIBA)

OAuth and OIDC flows

Implicit flow	Deprecated
Resource Owner Password Credentials flow	Deprecated
Authorization Code flow	
Hybrid flow	No longer used
Client Credentials flow	
Device flow	
Client-Initiated Backchannel Authentication flow (CIBA)	

OAuth and OIDC flows

Implicit flow

Resource Owner Password Credentials flow

Authorization Code flow — Commonly used

Hybrid flow

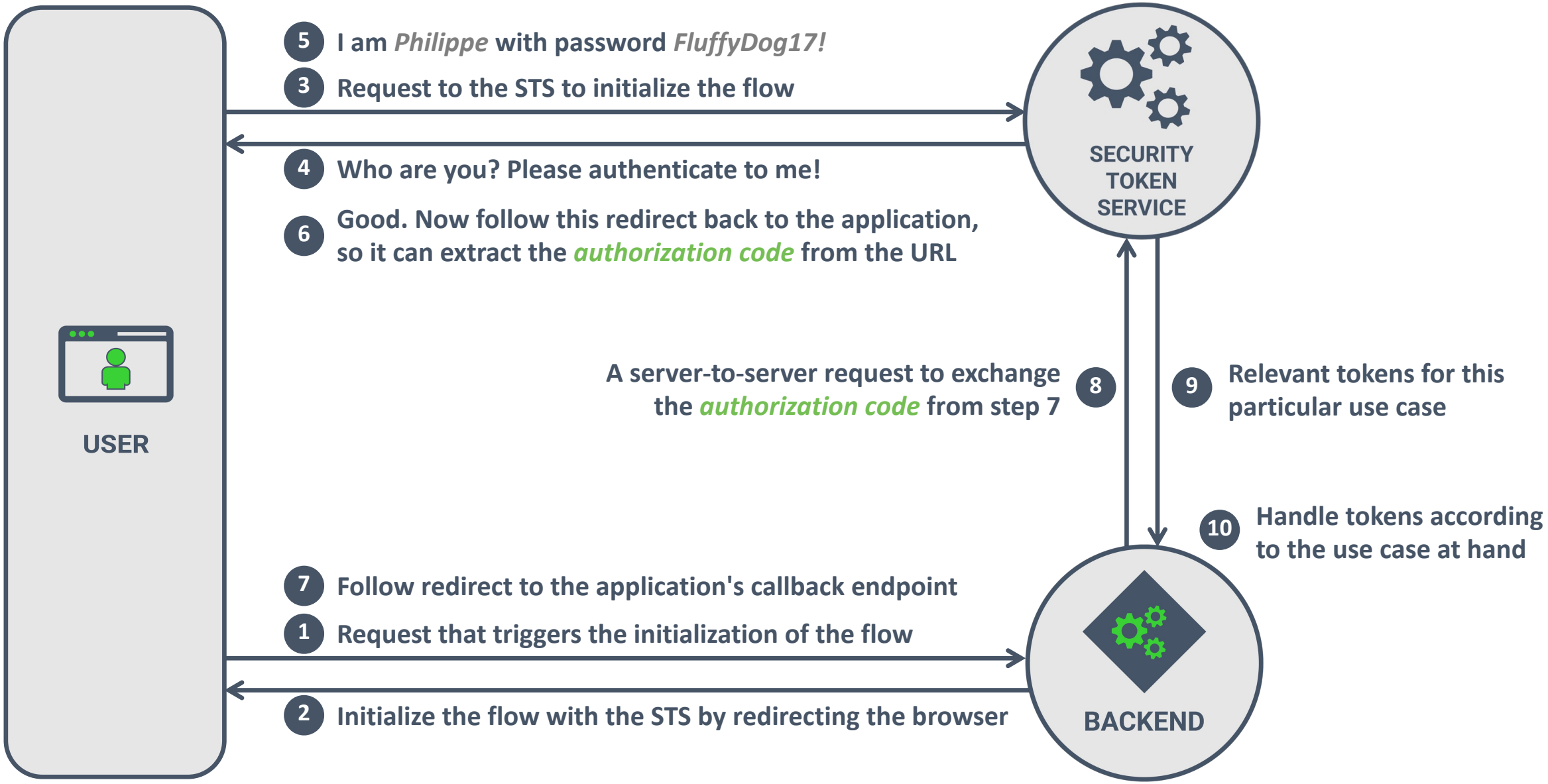
Client Credentials flow — Commonly used

Device flow

Client-Initiated Backchannel Authentication flow (CIBA)

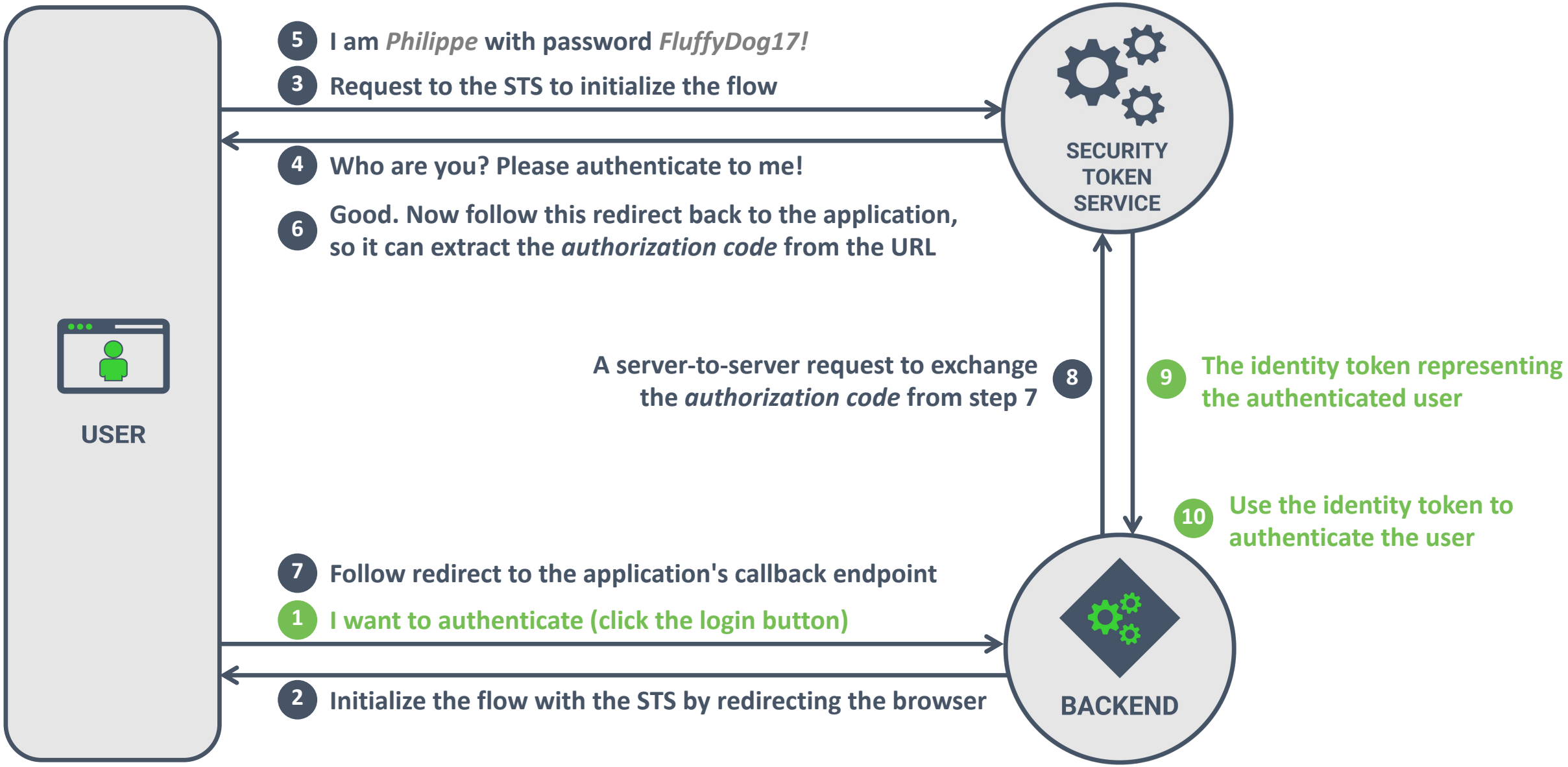
The *Authorization Code* flow

THE *AUTHORIZATION CODE* FLOW

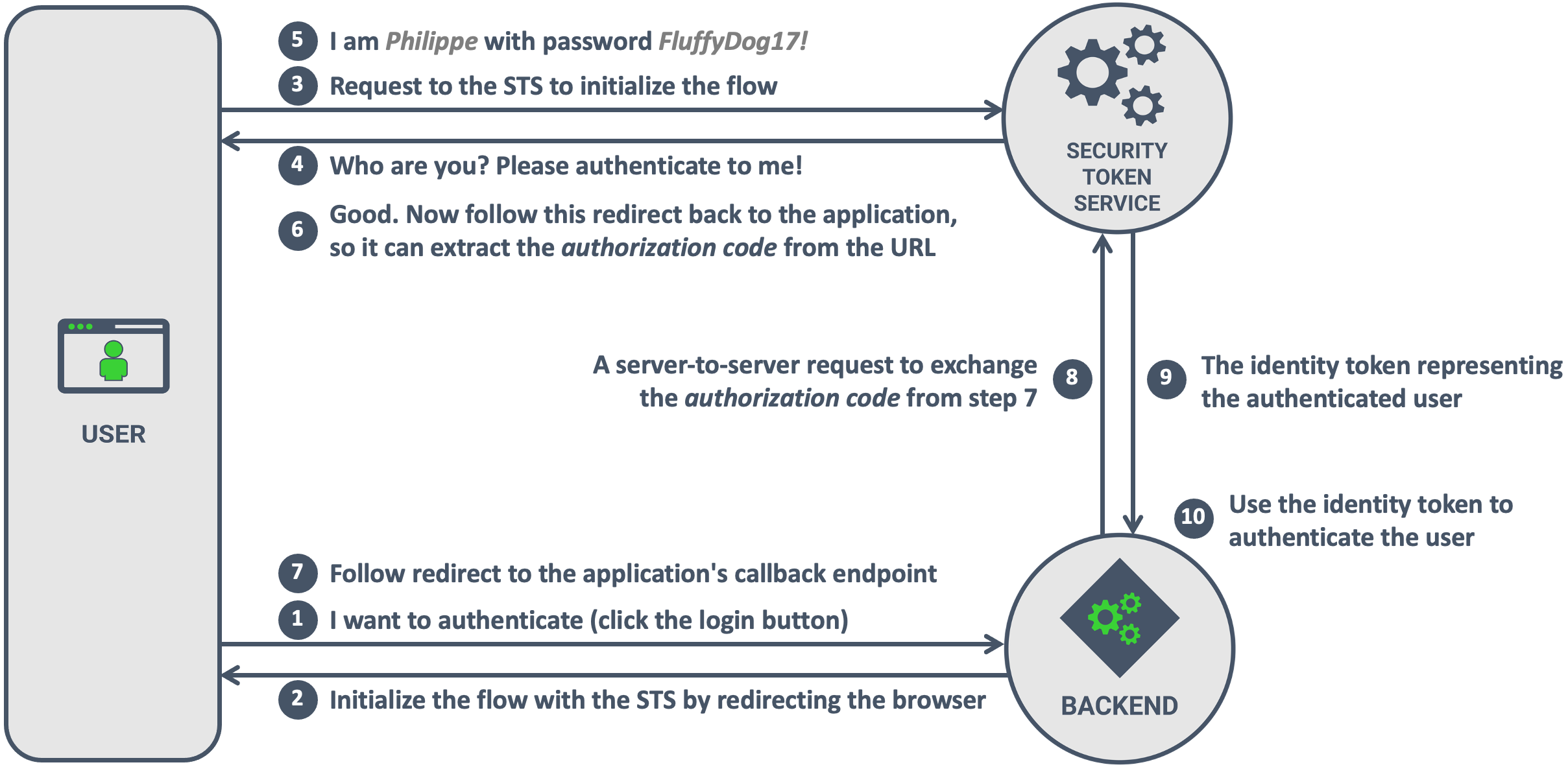


The *Authorization Code* flow supports both OAuth and OIDC scenarios

THE *AUTHORIZATION CODE* FLOW FOR OIDC



THE *AUTHORIZATION CODE* FLOW FOR OIDC

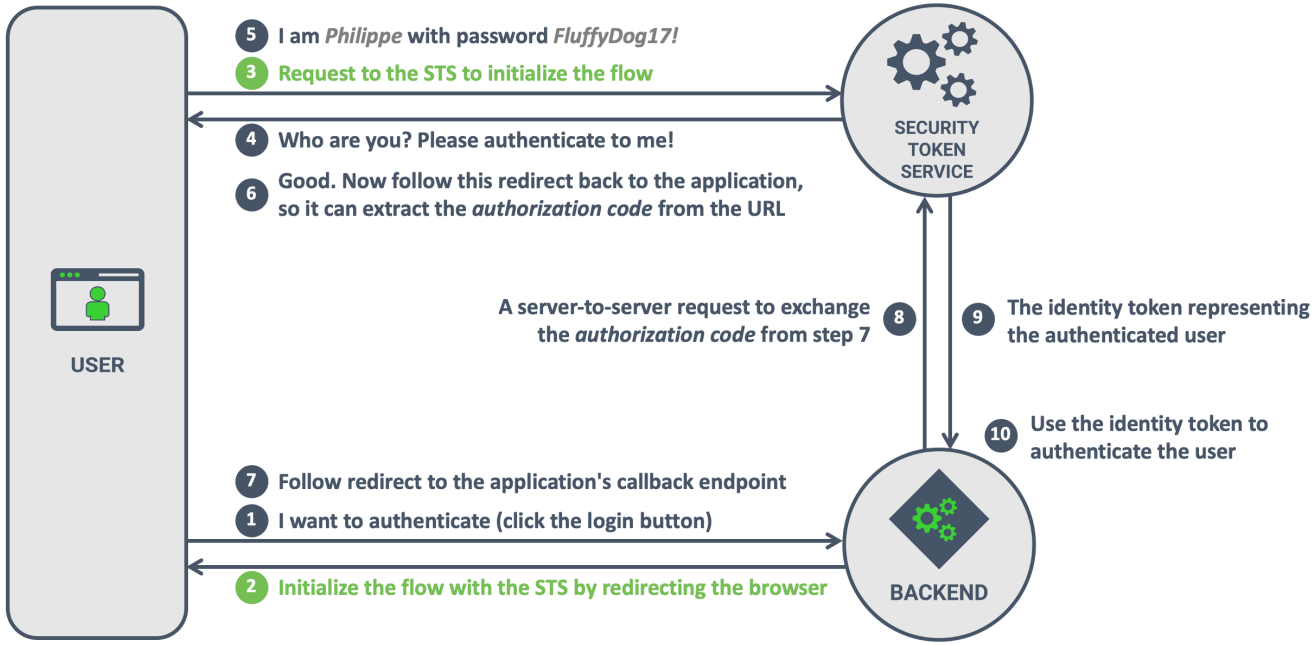


2 3 *The authorization request (a redirect to the STS)*

```
https://sts.restograde.com/authorize
  ?response_type=code
  &scope=openid profile email
  &client_id=FN983CEYgx4mdUg3NKNKHjwfNAL5Fb42
  &redirect_uri=https://restograde.com/callback
  &code_challenge=29K8tipblinCeP … HZ1PqLVxd9s
  &code_challenge_method=S256
```

- Indicates the *authorization code flow*
- We want an ID token with email/profile info
- The client requesting authentication
- Where the STS should send the code
- Flow security feature (PKCE)

THE *AUTHORIZATION CODE* FLOW FOR OIDC

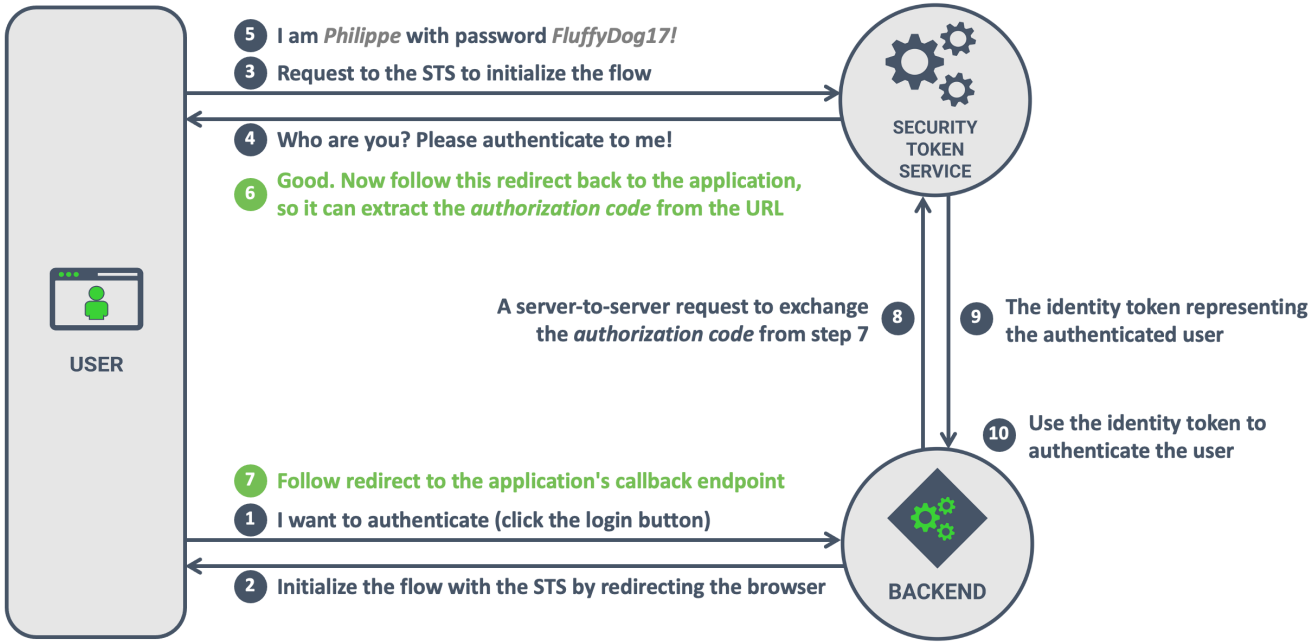


6 7 *The redirect back to the client application*

```
https://restograde.com/callback
  ?code=ySVyktqNkEKJyyIjOKCVwCurNlGoRDcaLYEbW2j5WxZY
```

The temporary authorization code

THE *AUTHORIZATION CODE* FLOW FOR OIDC

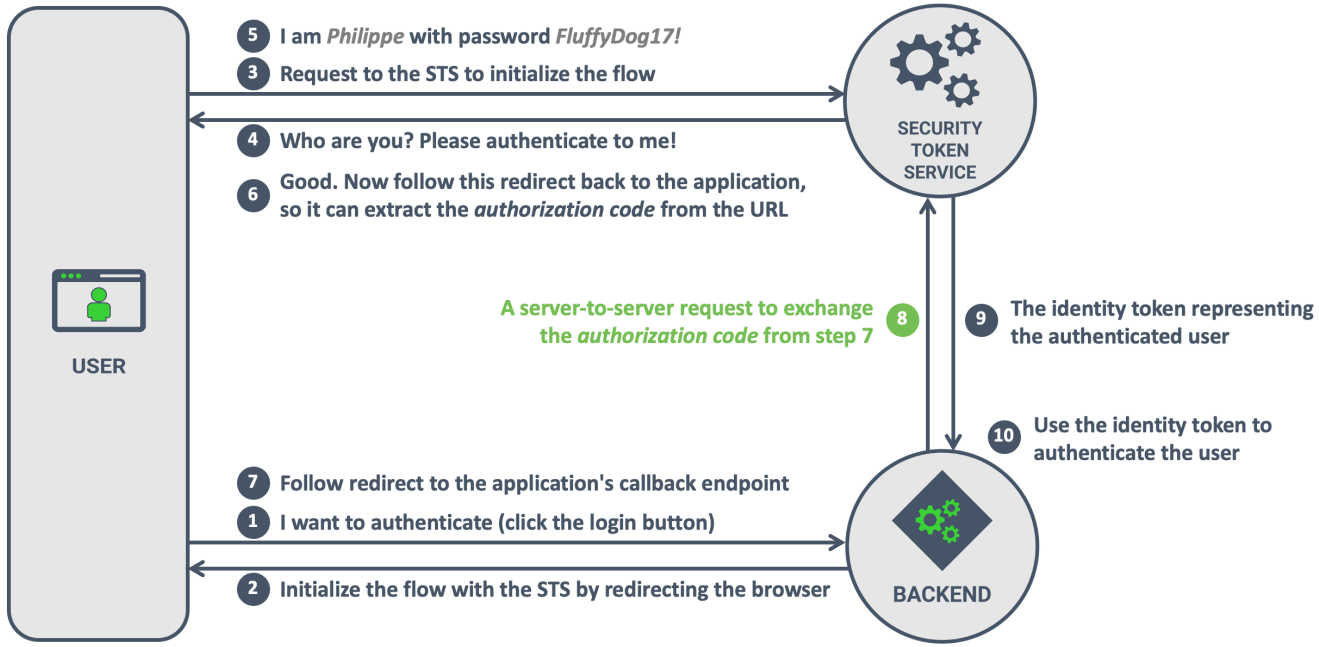


8 *The request to exchange the authorization code*

```
POST /oauth/token

 grant_type=authorization_code
&client_id=FN983CEYgx4mdUg3NKNKHjwfNAL5Fb42
&client_secret=6ODRv0g…OVOSWI
&redirect_uri=https://restograde.com/callback
&code=ySVyktqNkEKJyyIjOKCVwCurNlGoRDcaLYEbW2j5WxZY
&code_verifier=DOHpp1yiK0iElVij … K8HBZBqr75fKPps
```

- `grant_type=authorization_code` — Indicates the code exchange request
- `client_id=FN983CEYgx4mdUg3NKNKHjwfNAL5Fb42` — The client exchanging the code
- `client_secret=6ODRv0g…OVOSWI` — The client needs to authenticate to the STS
- `redirect_uri=https://restograde.com/callback` — The redirect URI used before
- `code=ySVyktqNkEKJyyIjOKCVwCurNlGoRDcaLYEbW2j5WxZY` — The code received in step 7
- `code_verifier=DOHpp1yiK0iElVij … K8HBZBqr75fKPps` — Flow security feature (PKCE)

THE *AUTHORIZATION CODE* FLOW FOR OIDC



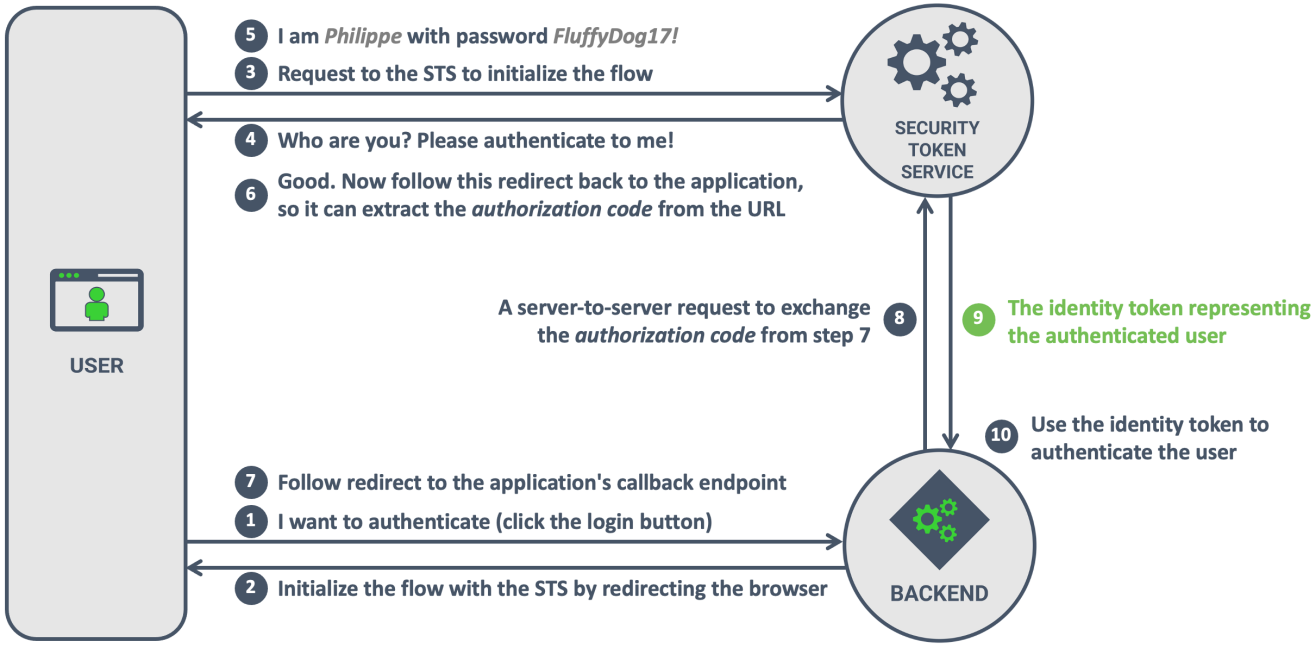
9 The response from the Security Token Service

```
1 {
2   "id_token": "eyJhbGciO...du6TY9w",
3 }
```

The identity token representing the authenticated user

The identity token contains a *sub* claim with the user's unique identifier. The application can use this claim to lookup the user in its database and establish and authenticated session

THE *AUTHORIZATION CODE* FLOW FOR OIDC



The *Authorization Code* flow and OIDC

THE *AUTHORIZATION CODE* FLOW FOR OIDC

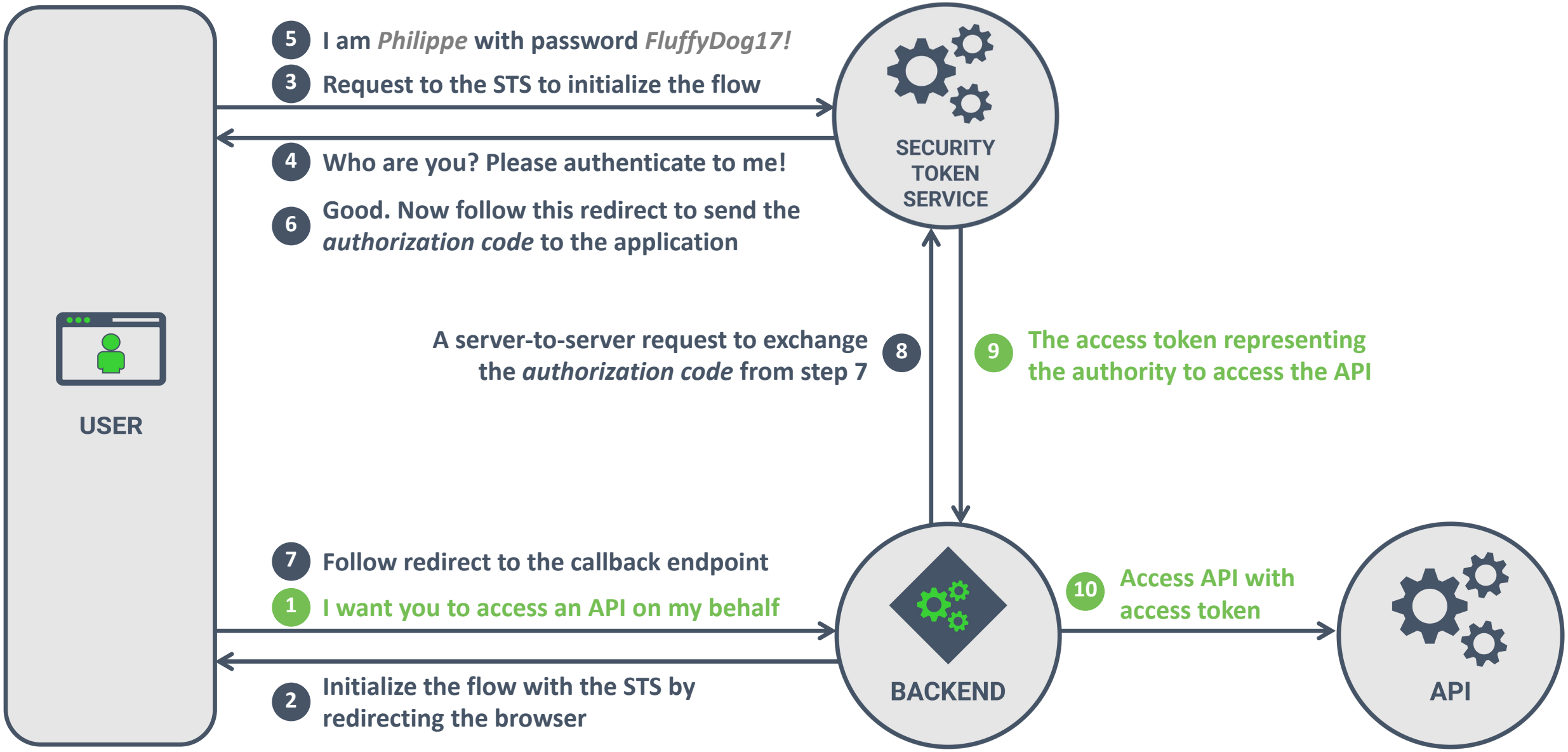
- The ***openid*** scope makes the *Authorization Code* flow an OIDC flow
  - In an OIDC flow, the STS provides the client with an identity token at the end of the flow
  - Additional scopes (e.g., *email*, *profile*) allow the client to request more user data

- The ***identity token*** provides information about the user's authentication
  - The mandatory ***sub*** claim contains the user's unique identifier at the STS
  - User-specific claims provide additional information about the user's identity
  - Additional claims can inform the client of authentication time, method, strength, …

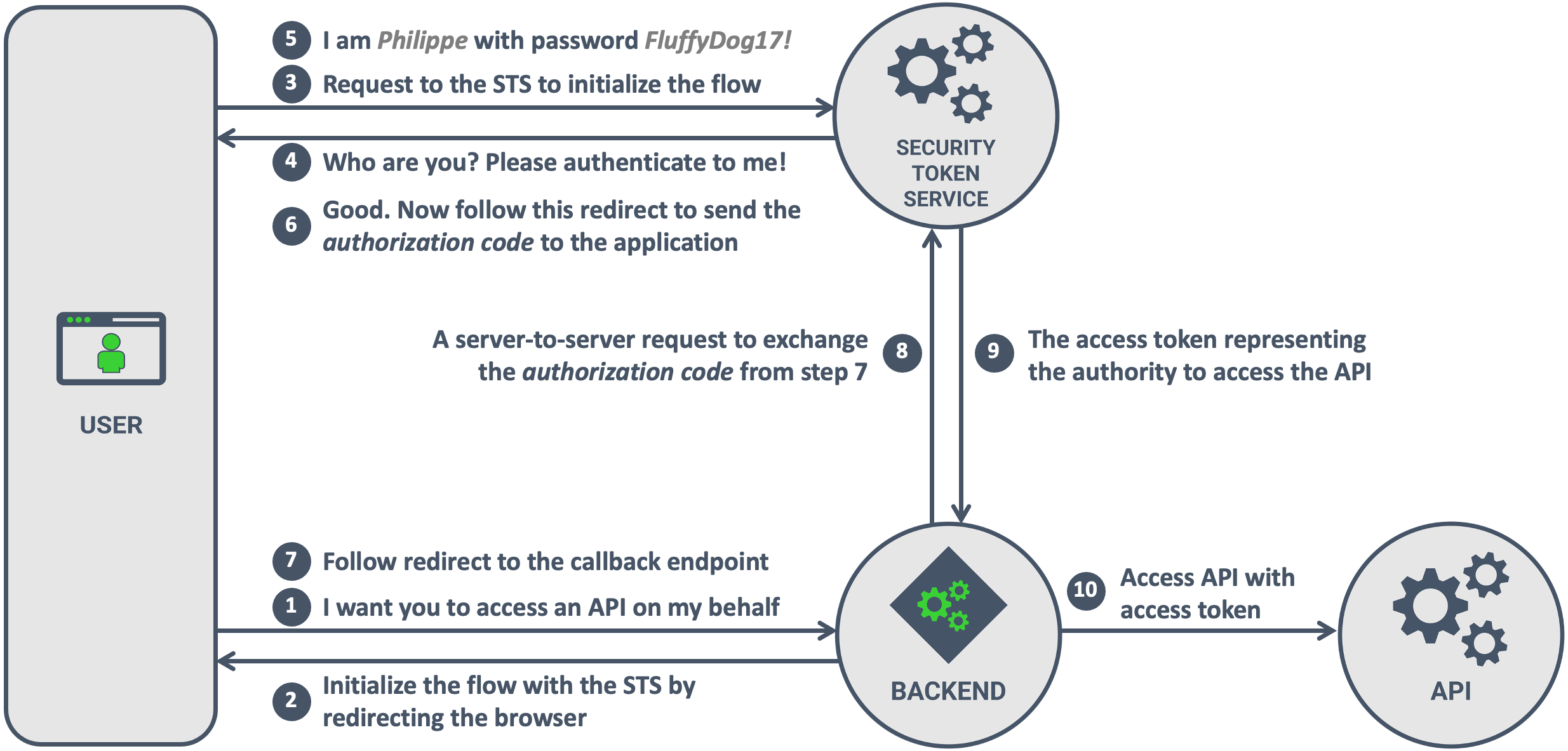
- The core OIDC specification supports two additional flows
  - The ***Implicit*** flow and ***Hybrid*** flow include the identity token directly in the callback
  - These flows avoid the authorization code exchange, but are significantly harder to secure

The *Authorization Code* flow is the current best practice to implement OIDC

THE *AUTHORIZATION CODE* FLOW FOR OAUTH



THE *AUTHORIZATION CODE* FLOW FOR OAUTH

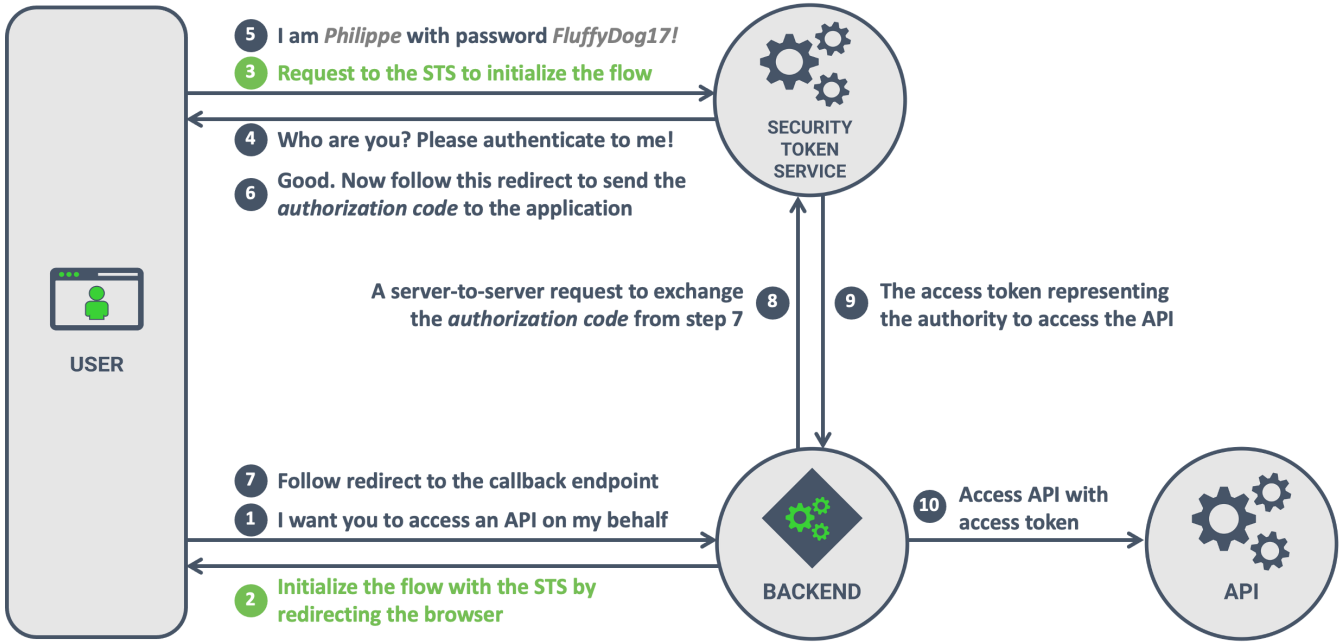


2 3 *The authorization request (a redirect to the STS)*

```
https://sts.restograde.com/authorize
  ?response_type=code
  &scope=reviews
  &client_id=AB983CEYgx4mdUg3NKNKHjwfNAL5Fb42
  &redirect_uri=https://virtualfoodie.com/callback
  &code_challenge=29K8tipblinCeP … HZ1PqLVxd9s
  &code_challenge_method=S256
```

- `?response_type=code` — Indicates the *authorization code flow*
- `&scope=reviews` — We want a token with *reviews* access
- `&client_id=AB983CEYgx4mdUg3NKNKHjwfNAL5Fb42` — The client requesting the token
- `&redirect_uri=https://virtualfoodie.com/callback` — Where the STS should send the code
- `&code_challenge=29K8tipblinCeP … HZ1PqLVxd9s` / `&code_challenge_method=S256` — Flow security feature (PKCE)

THE *AUTHORIZATION CODE* FLOW FOR OAUTH

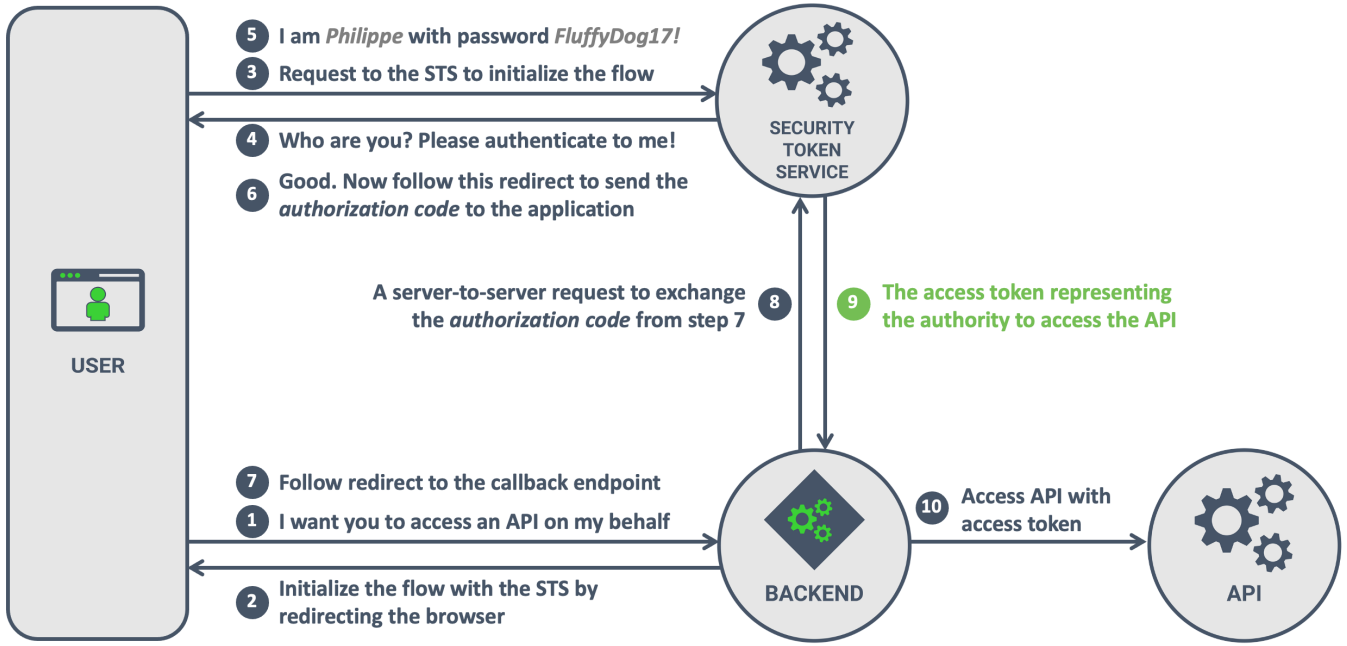


9 The response from the Security Token Service

```
1 {
2   "access_token": "eyJhbGciO...du6TY9w",
3   "expires_in": 3600,
4   "scope": "reviews",
5   "token_type": "Bearer"
6 }
```

- `"access_token"` — The access token intended for accessing APIs
- `"expires_in"` — The lifetime of the access token (seconds)
- `"scope"` — The scopes associated with the token
- `"token_type"` — The type of token

THE *AUTHORIZATION CODE* FLOW FOR OAUTH



THE *AUTHORIZATION CODE* FLOW

- The *Authorization Code* flow supports both OAuth and OIDC scenarios
  - The ***openid*** scope augments the OAuth *Authorization Code* flow with OIDC features

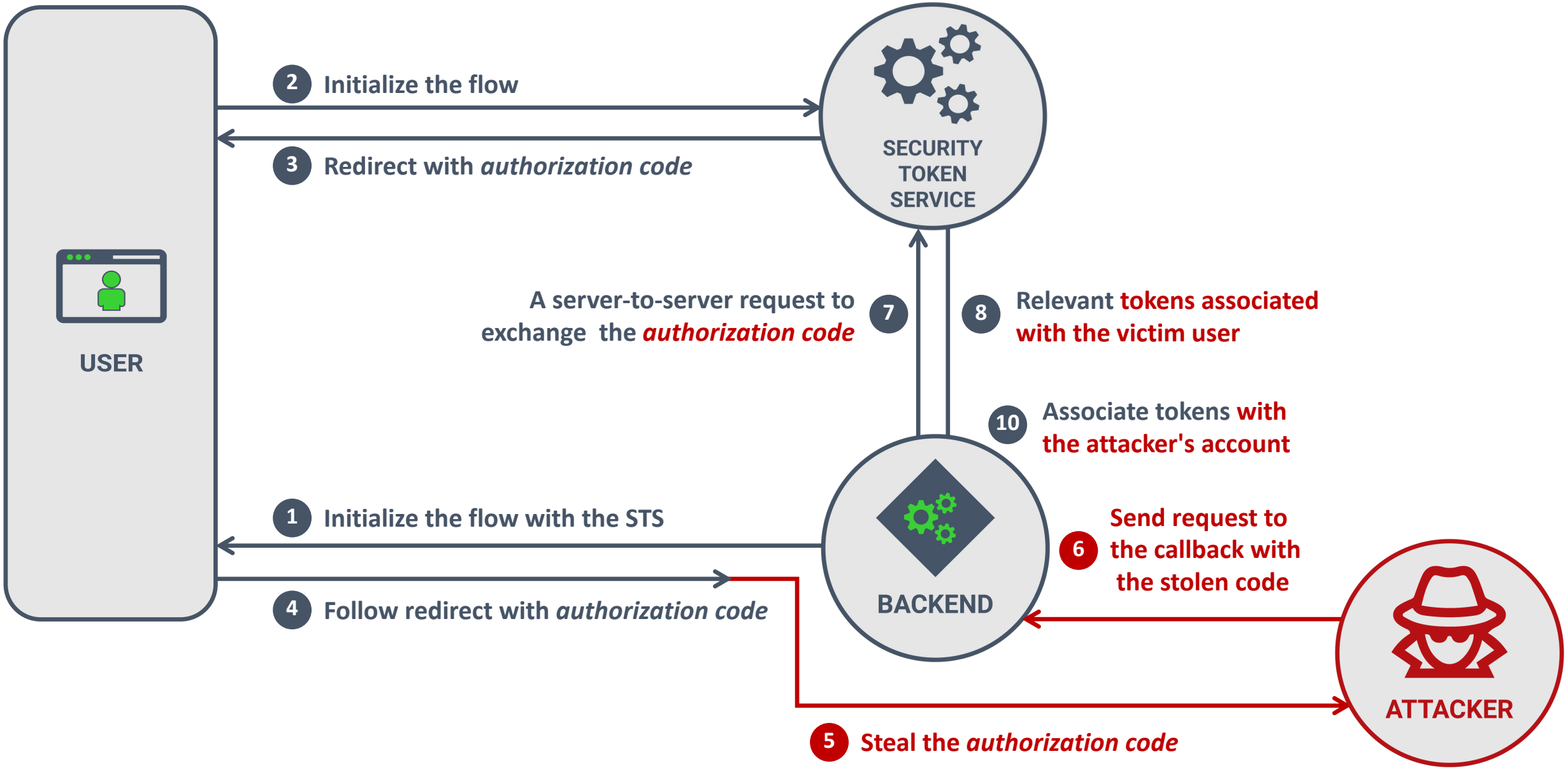
- The client application is known as a ***confidential client***
  - Confidential clients run in a restricted environment (e.g., a server environment)
  - Confidential clients have access to a secret, allowing them to authenticate to the STS

- The authorization code is protected against abuse
  - A confidential client needs to authenticate to exchange an authorization code
  - Authorization codes should be short-lived and should only be valid for one-time use

Securing the flow with PKCE

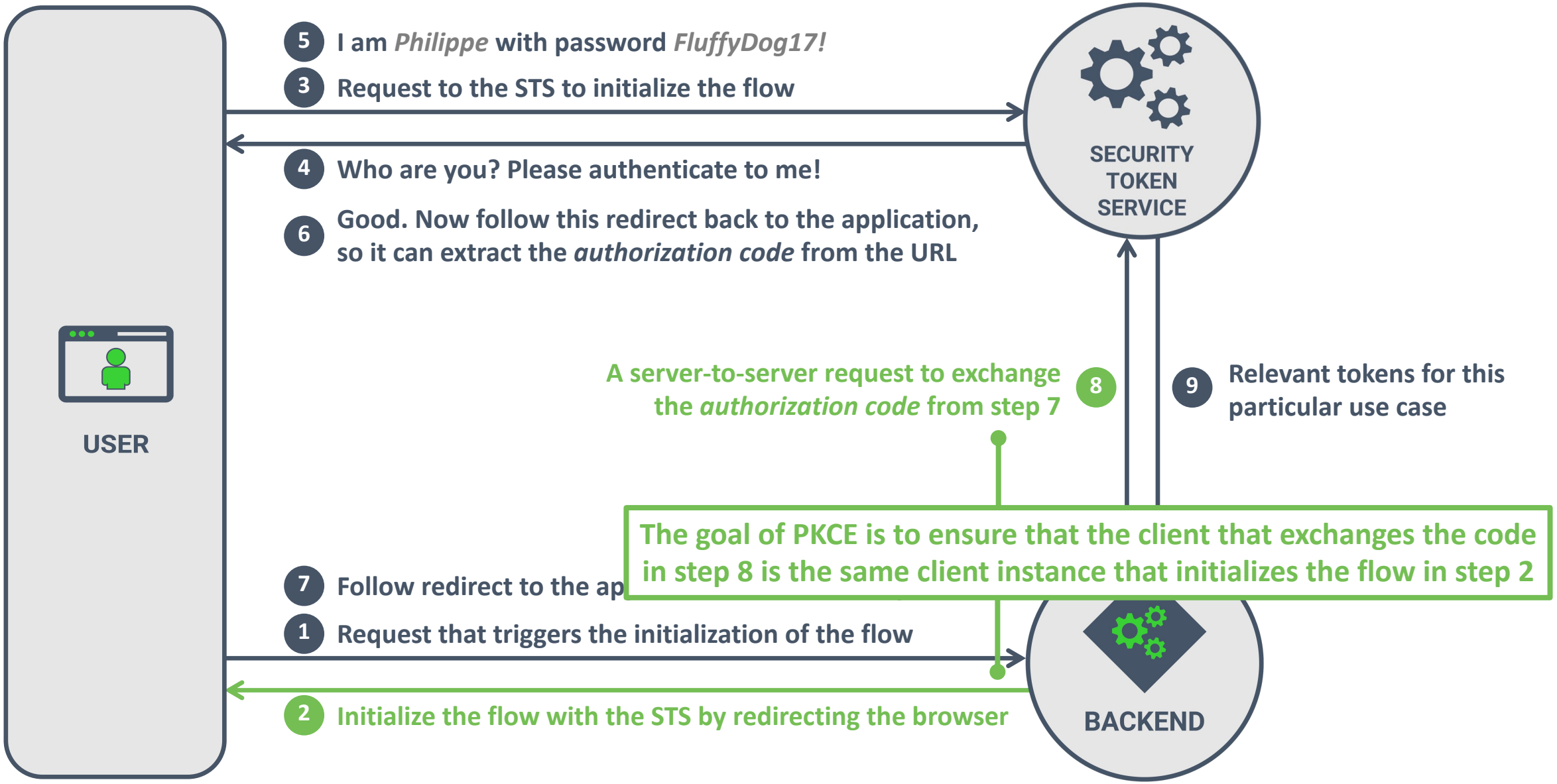
The *Authorization Code* flow relies on the insecure front channel to relay the code

AN *AUTHORIZATION CODE* INJECTION ATTACK

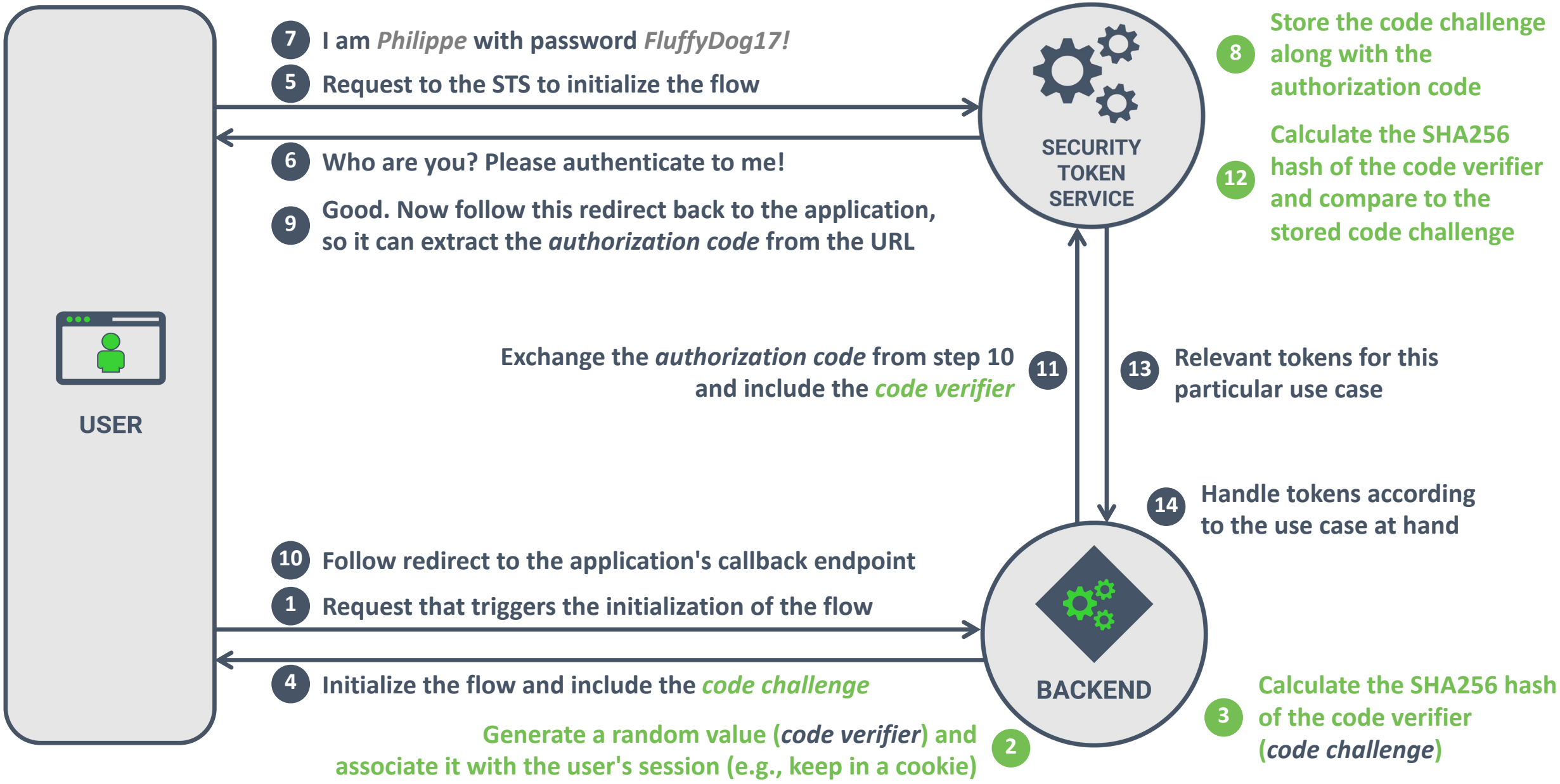


Proof Key for Code Exchange (PKCE) helps protect the integrity of the *Authorization Code* flow

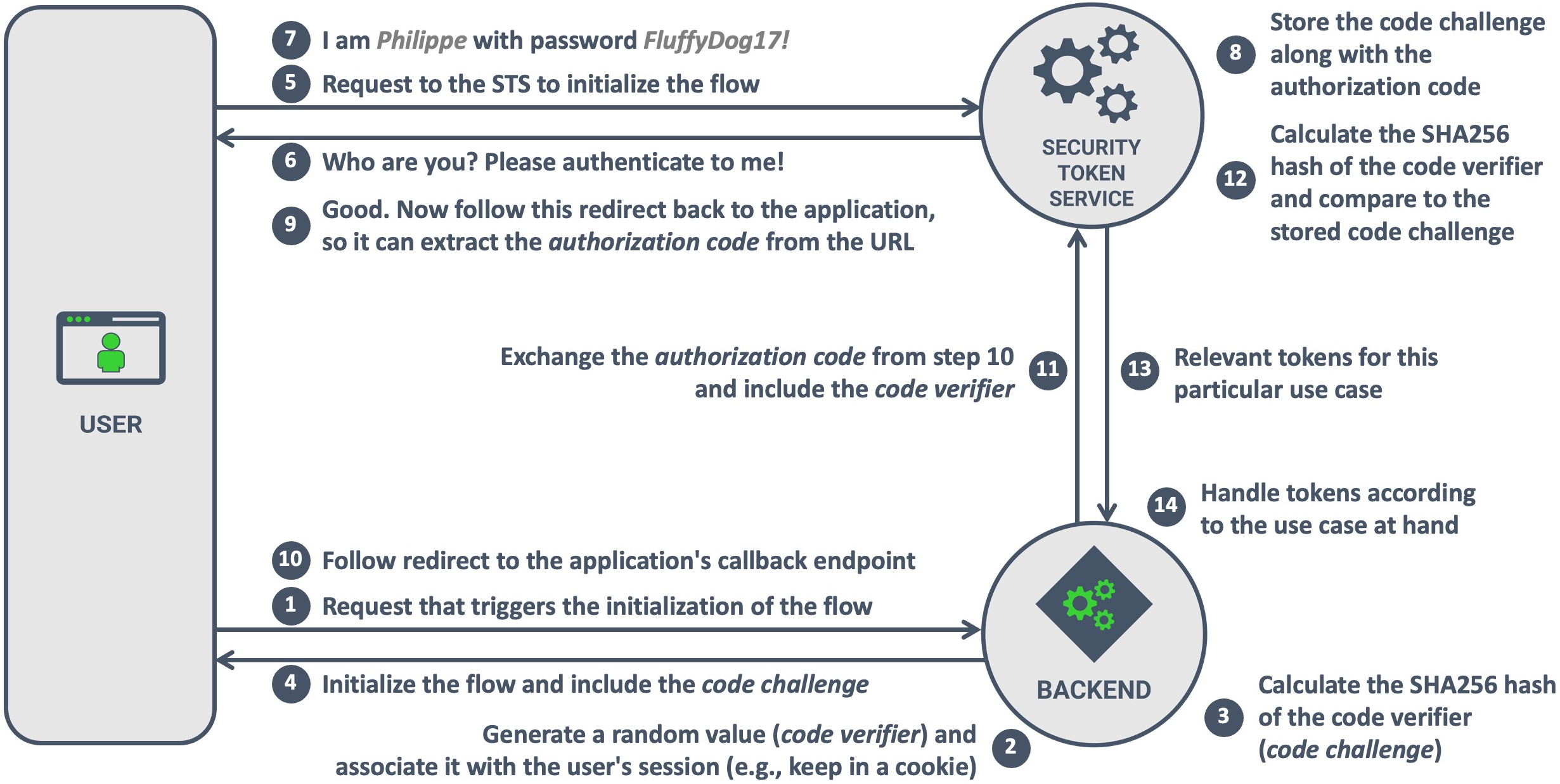
THE CONCEPT OF PKCE



THE *AUTHORIZATION CODE* FLOW WITH PKCE



THE *AUTHORIZATION CODE* FLOW WITH PKCE



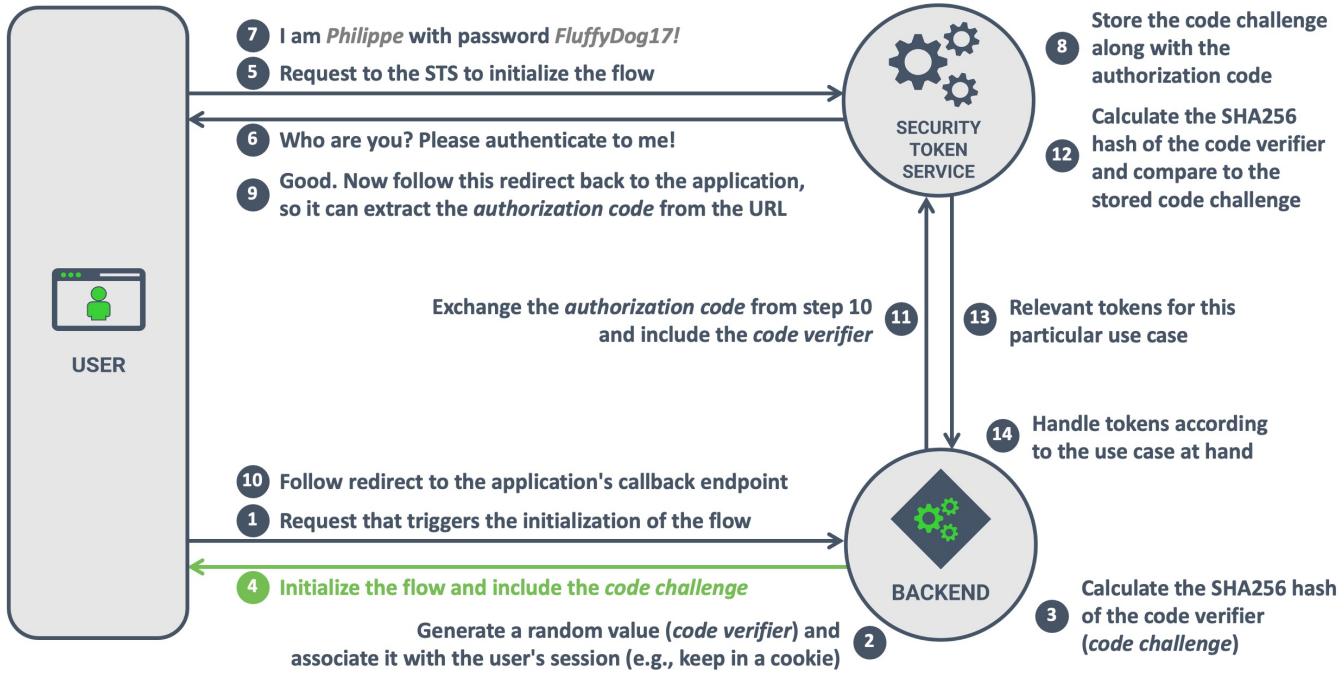
2 3 *The authorization request (a redirect to the STS)*

```
https://sts.restograde.com/authorize
  ?response_type=code
  &scope=openid profile email
  &client_id=FN983CEYgx4mdUg3NKNKHjwfNAL5Fb42
  &redirect_uri=https://restograde.com/callback
  &code_challenge=29K8tipblinCeP … HZ1PqLVxd9s
  &code_challenge_method=S256
```

The code challenge (hash of code verifier)

The hash function used (for upgradeability)

THE *AUTHORIZATION CODE* FLOW WITH PKCE

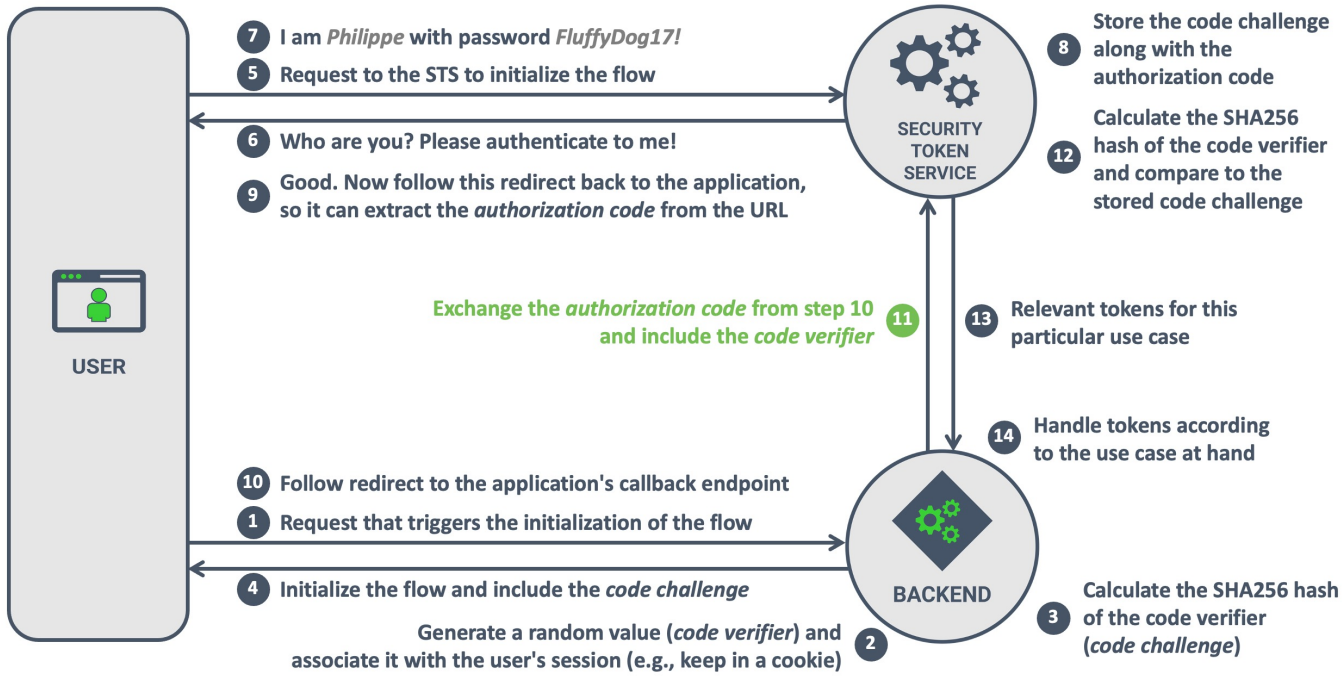


8 *The request to exchange the authorization code*

```
POST /oauth/token

 grant_type=authorization_code
&client_id=FN983CEYgx4mdUg3NKNKHjwfNAL5Fb42
&client_secret=6ODRv0g…OVOSWI
&redirect_uri=https://restograde.com/callback
&code=ySVyktqNkEKJyyIjOKCVwCurNlGoRDcaLYEbW2j5WxZY
&code_verifier=DOHpp1yiK0iElVij … K8HBZBqr75fKPps
```

The code verifier from step 2

THE *AUTHORIZATION CODE* FLOW WITH PKCE



Proof Key for Code Exchange (PKCE)

- PKCE consists of a code verifier and a code challenge
  - The code verifier is a **cryptographically secure random string**
    - Between 43 and 128 characters of this character set: [A-Z] [a-z] [0-9] - . _ ~
  - The code challenge is a ***base64 urlencoded SHA256*** hash of the code verifier
    - The hash function uniquely connects the code challenge to the code verifier
    - The code verifier cannot be derived from the code challenge

- PKCE ensures that the same client intializes and finalizes the flow
  - PKCE was originally intended to secure flows of public clients (no client authentication)
  - Today, PKCE is a recommended best practice to guarantee flow integrity

- **PKCE replaces the OAuth *state* parameter or OIDC *nonce* for security**

PKCE in action

PKCE has become a security best practice for all *Authorization Code* flows

Modern libraries handle all of the heavy lifting

4. Spring Security Support for PKCE

As of Spring Security 5.7, PKCE is fully supported for both servlet and reactive flavored web applications. However, this feature is not enabled by
extension yet. Spring Boot applications must use ver
standard dependency management. This ensures th
along with its transitive dependencies.

Pk
ea

PKCE Support for OAuth 2.0

5, 2019

port has been added to `passport-`
C 7636.

Out of the box PKCE in ASP.NET Core 3

With ASP.NET Core 3, it's a simple case of setting a property on `OpenIdConnectOptions` to true:

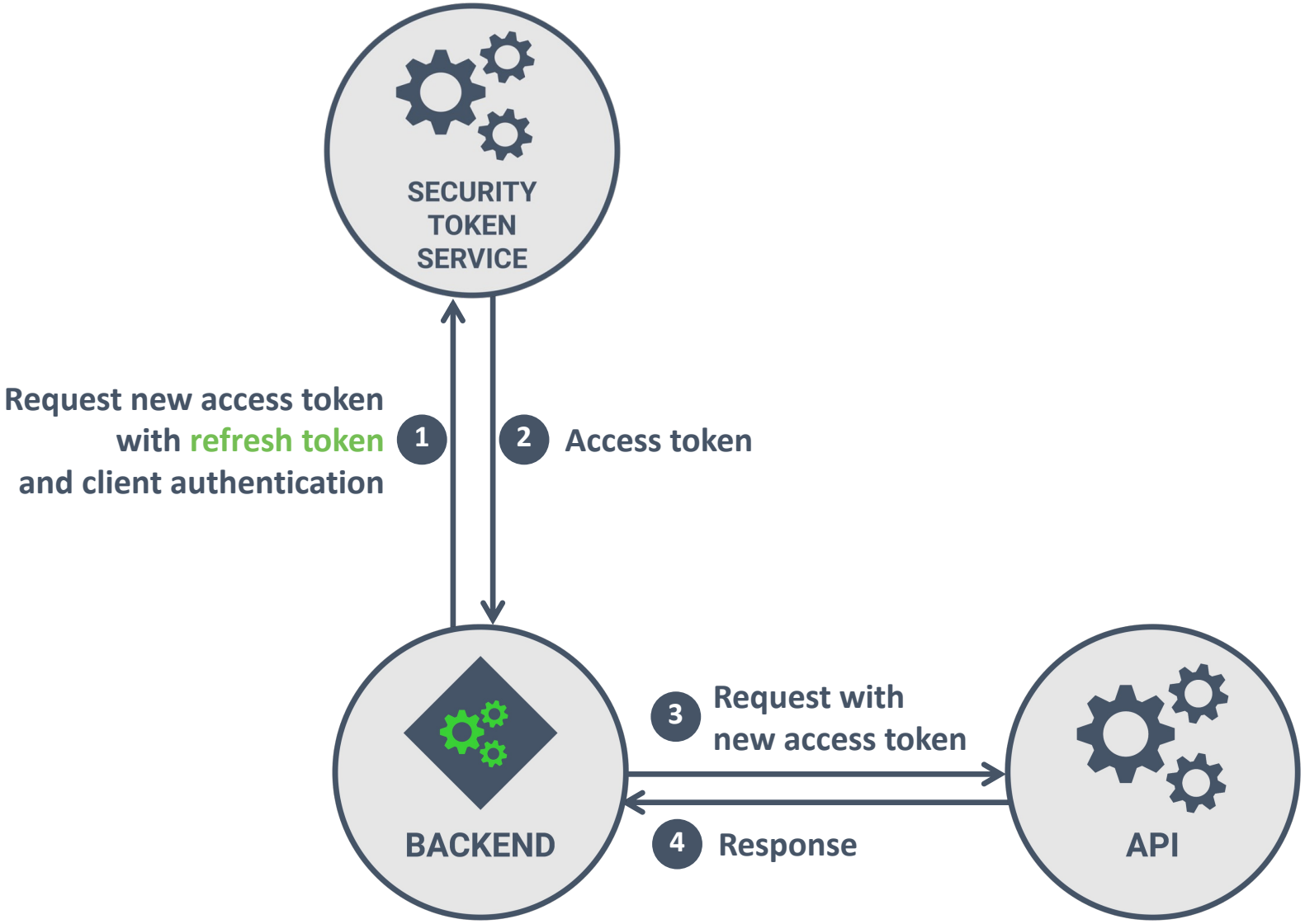
```
services.AddAuthentication()
    // other registrations
    .AddOpenIdConnect("oidc", options => {

        // existing config

        // Enable PKCE (authorization code flow only)
        options.UsePkce = true;
    });
```

https://www.baeldung.com/spring-security-pkce-secret-clients
https://medium.com/passportjs/pkce-support-for-oauth-2-0-e3a77013b278
https://www.scottbrady91.com/openid-connect/aspnet-core-using-proof-key-for-code-exchange-pkce#pkce-3

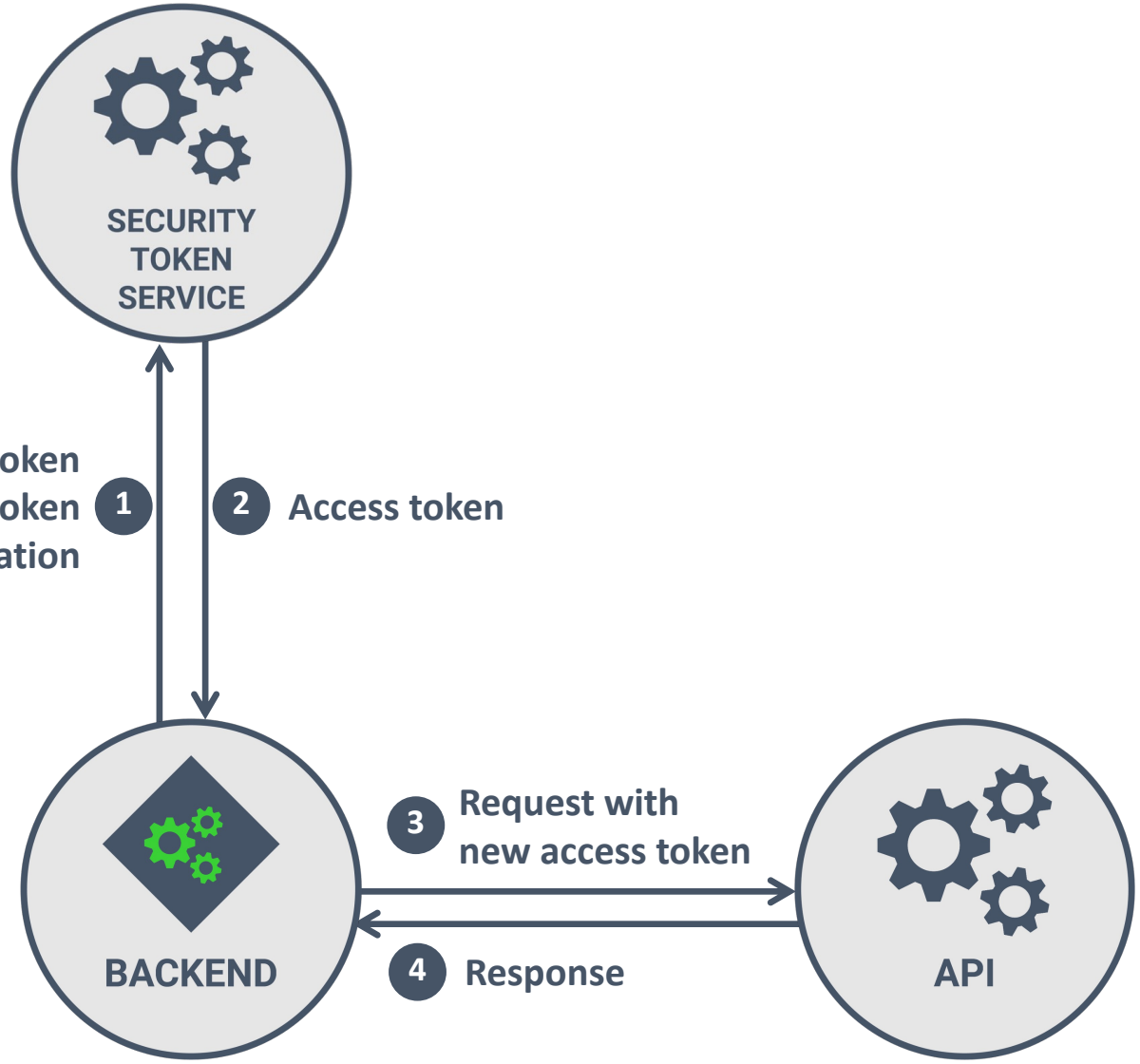
Understanding OAuth and OpenID Connect

Long-term access with refresh tokens

THE REFRESH TOKEN FLOW



THE REFRESH TOKEN FLOW

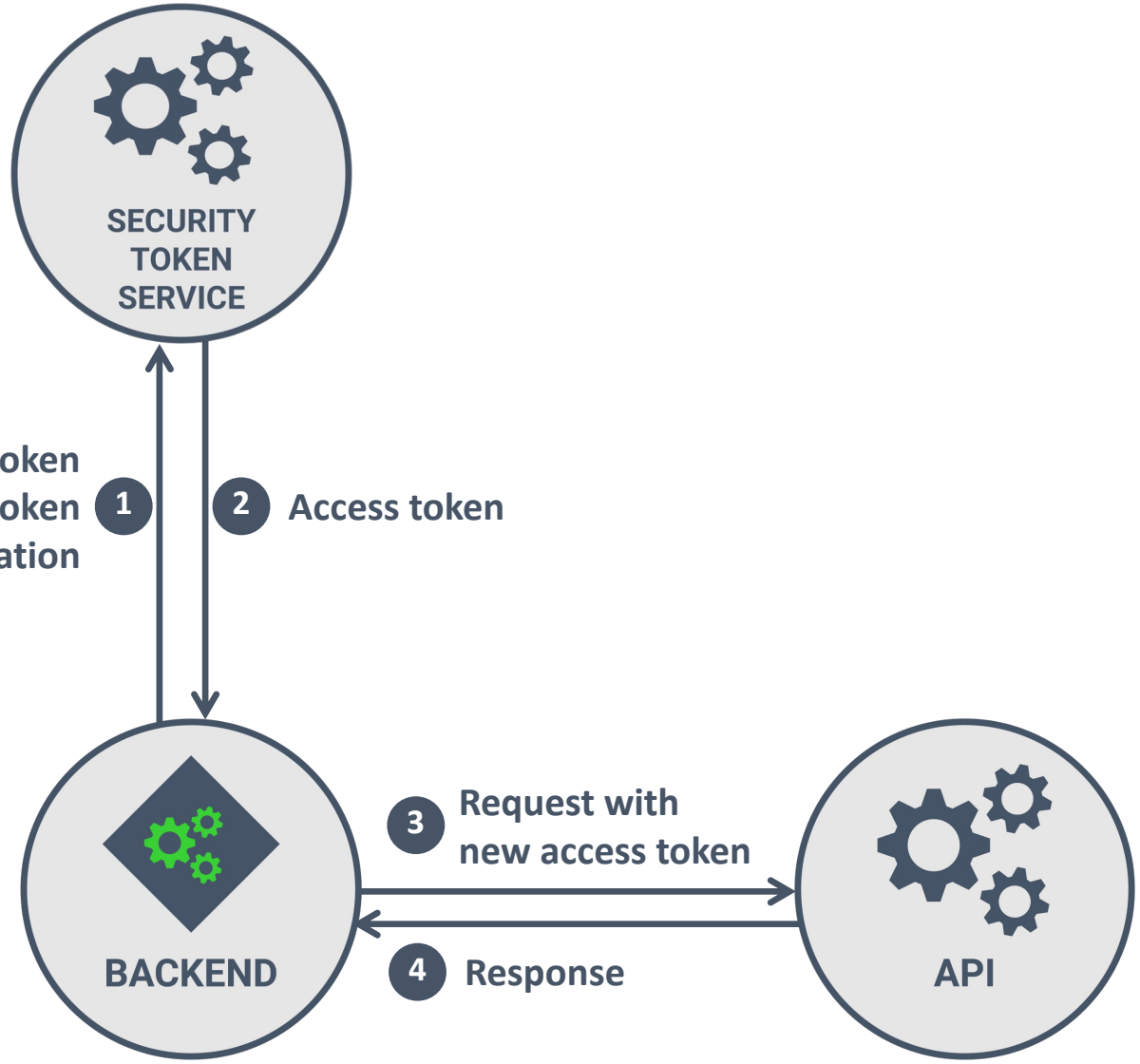


The client application decides when to use a refresh token to get new access tokens

One option is to use the *expires_in* parameter in the token response to figure out when the access token expires.

Another option is to monitor error responses from the API when an expired token is rejected.

THE REFRESH TOKEN FLOW



The client application decides when to use a refresh token to get new access tokens

If the client only needs infrequent one-shot access to the API, it can consider not storing access tokens and running a refresh token flow each time it needs an access token

Using refresh tokens

How does the client store a refresh token?

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

Oct 27, 2013 8 min read

Joel Gascoigne
CEO and co-founder @ Buffer

Update: This article was originally titled "Buffer has been hacked – here is what's going on". The hacking incident happened yesterday (Saturday) and below is a recap of everything that happened. Please ask us any questions you have in the comments below.

If you're reading this, the most important section for you is Update 7.

We've discovered the source of the breach and closed the vulnerability. Keep reading for the full story.

https://buffer.com/resources/buffer-has-been-hacked-here-is-whats-going-on/

Handling refresh tokens at the client

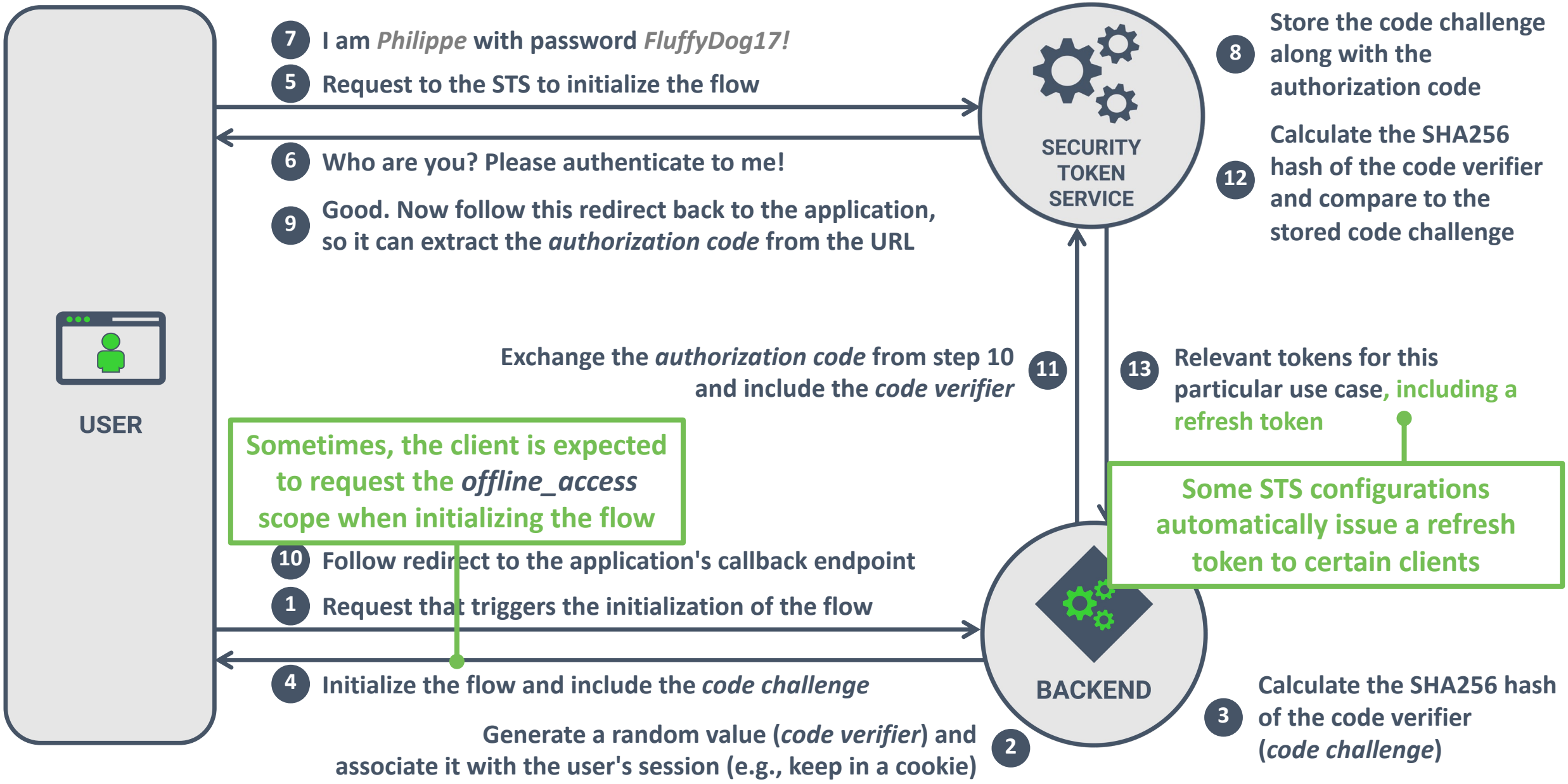
- The refresh token should be considered as sensitive as user credentials
  - This is somewhat nuanced since using the refresh token requires client authentication
  - When an attacker gains access to both, the users are in major trouble

- A minimum security requirement is guaranteeing confidential storage
  - This approach fails if an attacker gains access to the encrypted data and the keys

- Consider moving refresh tokens to an isolated service in your architecture
  - The main application can request a new access token from this service
  - Only the service has access to the encrypted refresh tokens and associated keys
  - This compartmentalization reduces the impact of application-level compromises

How does the client get a refresh token?

Getting a refresh token from the STS



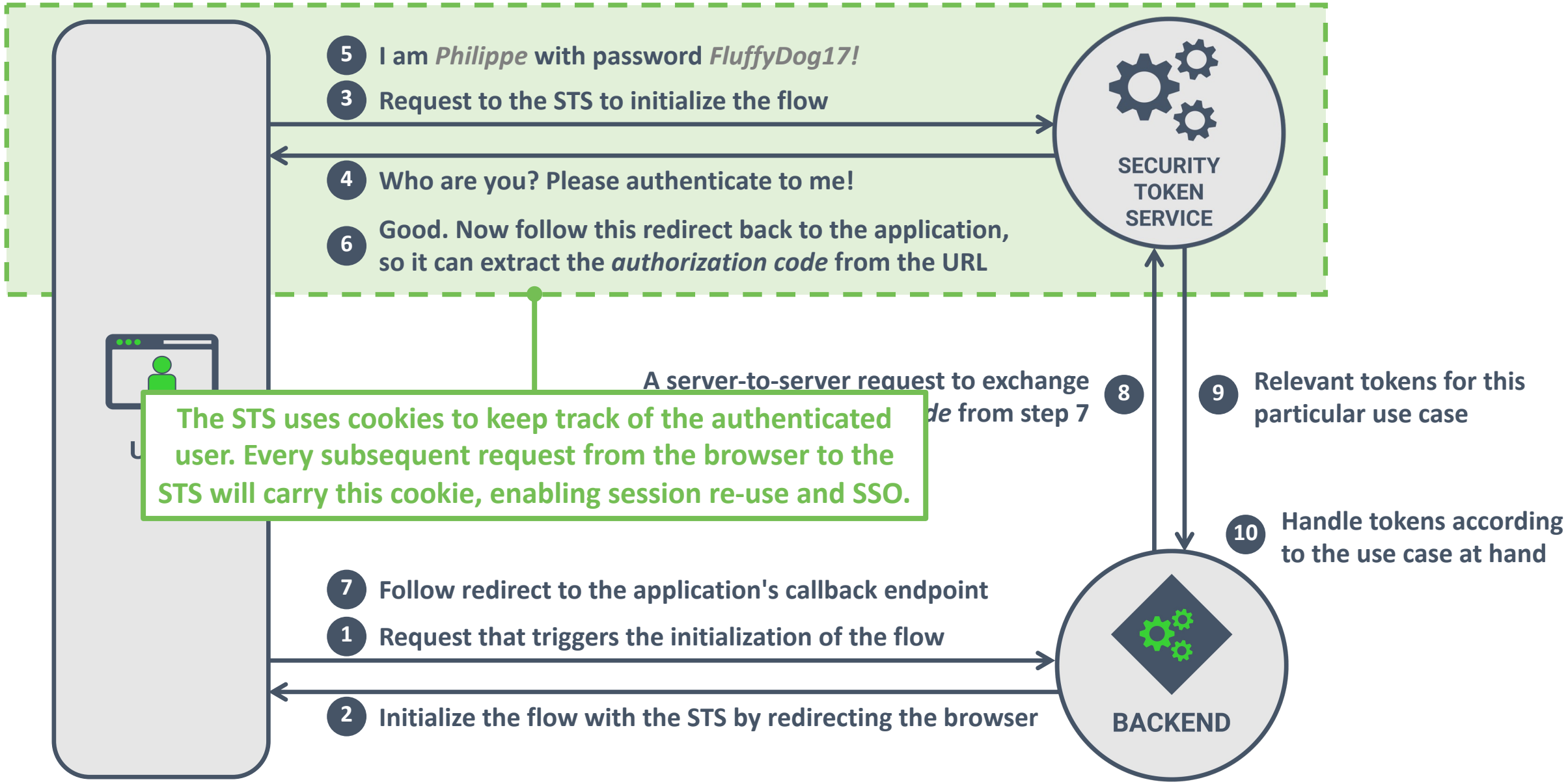
What is the lifetime of the refresh token?

Refresh token lifetimes

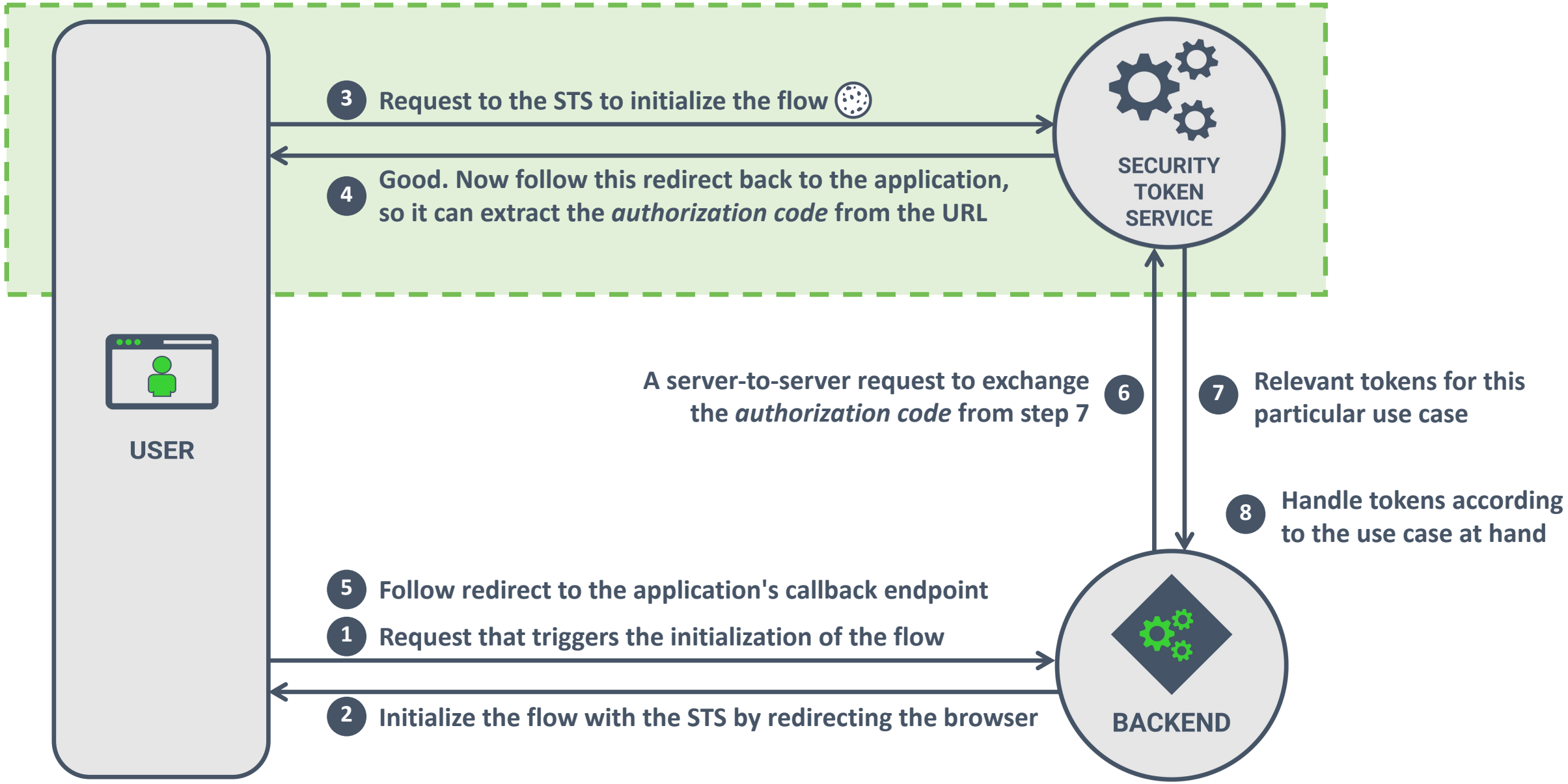
- The exact refresh token lifetime is at the discretion of the STS
  - Refresh token lifetimes in real-world scenarios can be hours, months, or eternity
  - The STS can change its lifetime policy at will, or make it dependent on the type of client
- Refresh tokens can also be revoked at the STS
  - Clients can revoke refresh tokens when they no longer need them
  - Users can often revoke refresh tokens to revoke a client's authority to act on their behalf
- When a refresh token is no longer valid, there is no path to recovery
  - The only way for the client to regain access is by running a new *Authorization Code* flow
  - For backend client applications, this often includes explicitly requesting user involvement

Session re-use and Single Sign-On

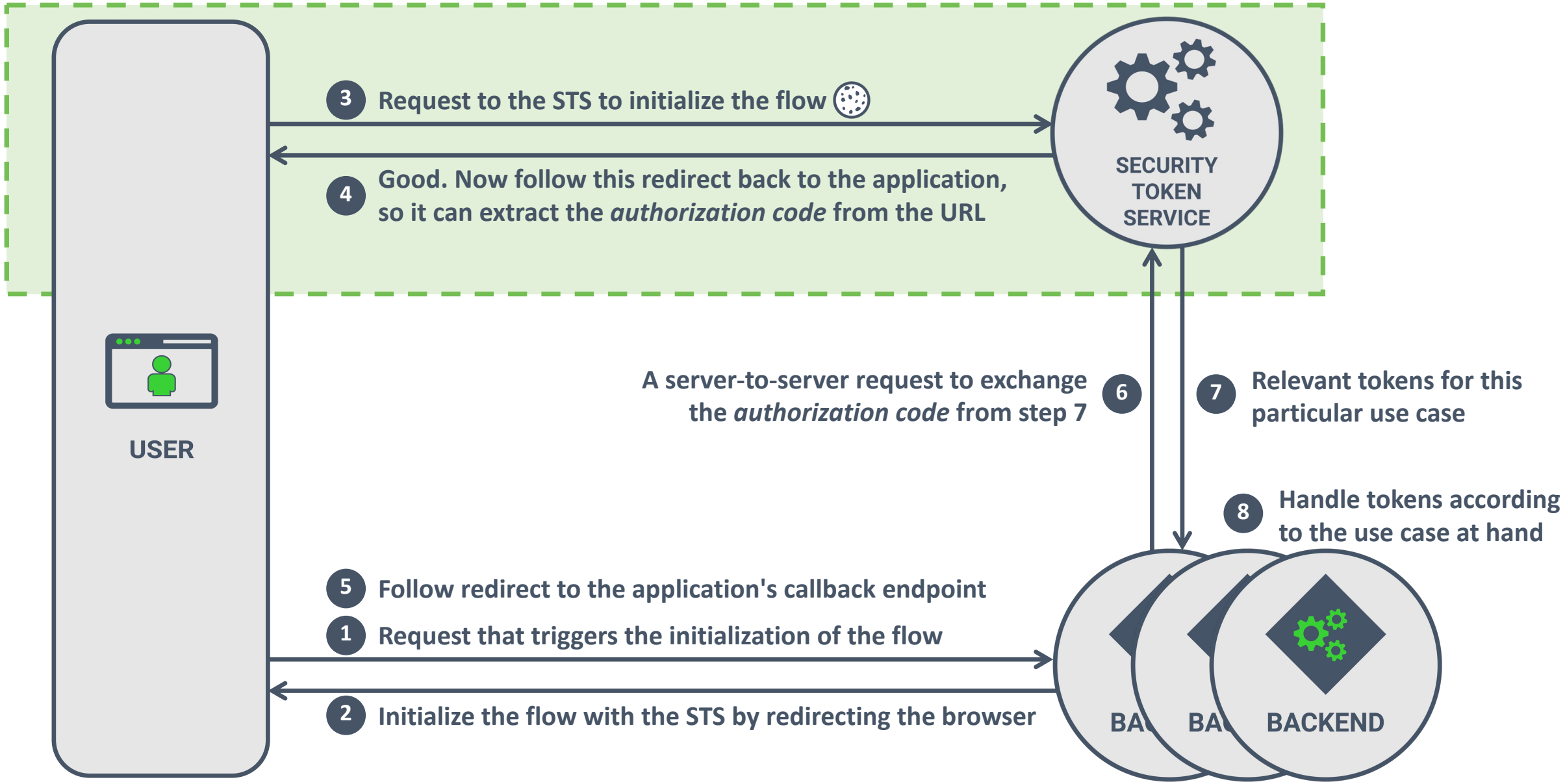
THE *AUTHORIZATION CODE* FLOW



Running a flow with an authenticated session



Re-using an authenticated session for Single Sign-On

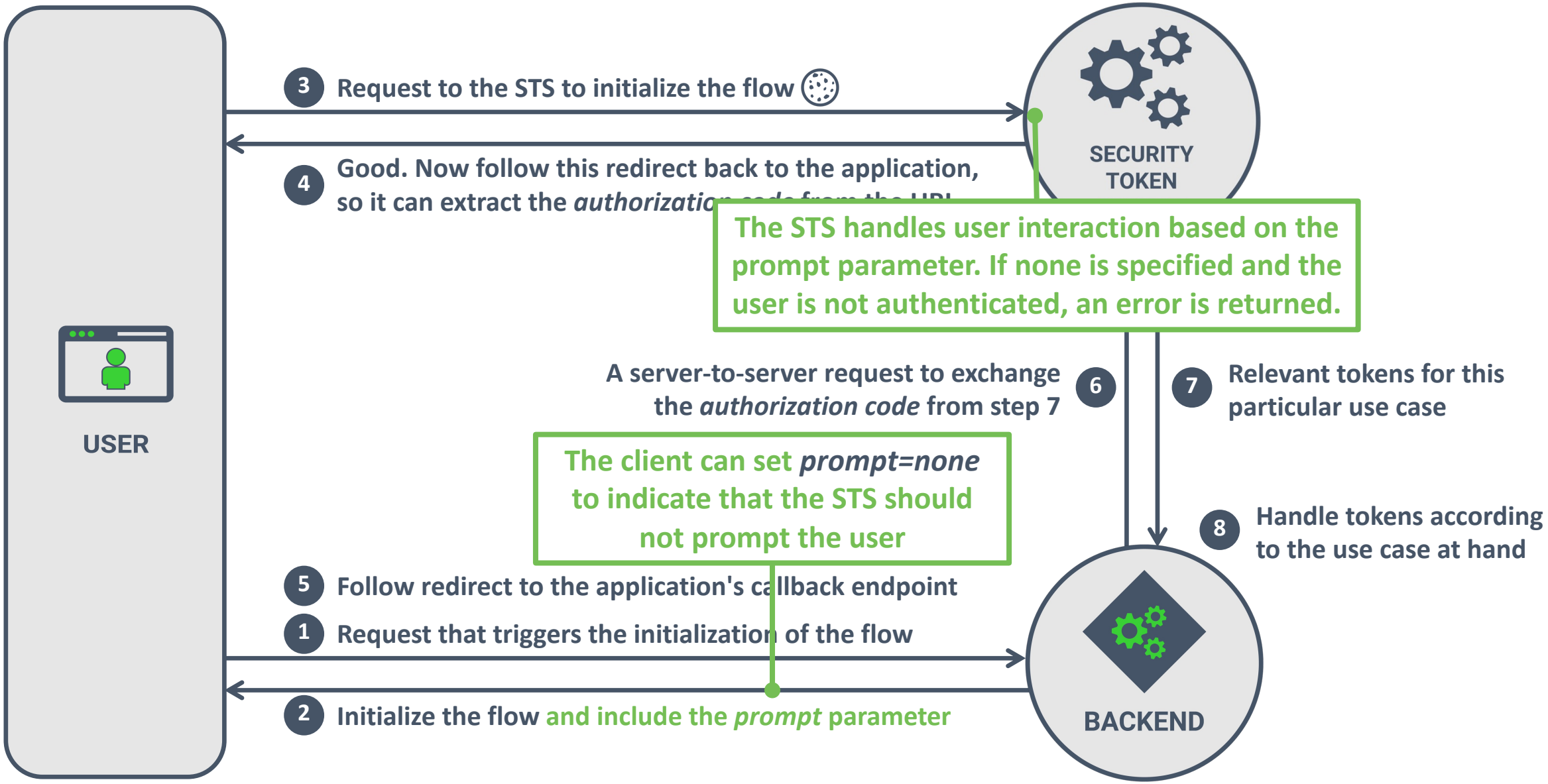


User sessions with the STS

- Managing the session of the user is the responsibility of the STS
  - The STS has full control over how the session is managed and set
  - The STS decides how long a user's session should be valid
  - The STS can use inactivity timeouts to terminate sessions when desired
- As long as the user has an active session with the STS, there is no logout
  - Whenever a client runs an *Authorization Code* flow, it will re-use the existing session
  - Application architectures often have to decide if they want to implement ***Single Logout***
- There is no explicit link between session lifetimes and token lifetimes
  - For backend clients, use cases typically require long-term access using refresh tokens
  - For other types of clients (web, mobile), refresh tokens may resemble sesion lifetimes
  - Some highly-restrictive scenarios actively invalidate refresh tokens upon user logout

The *prompt* parameter allows the client to control user interaction with the STS

USING THE *PROMPT* PARAMETER TO CONTROL USER INTERACTION



The *prompt* parameter in action

THE *PROMPT* PARAMETER

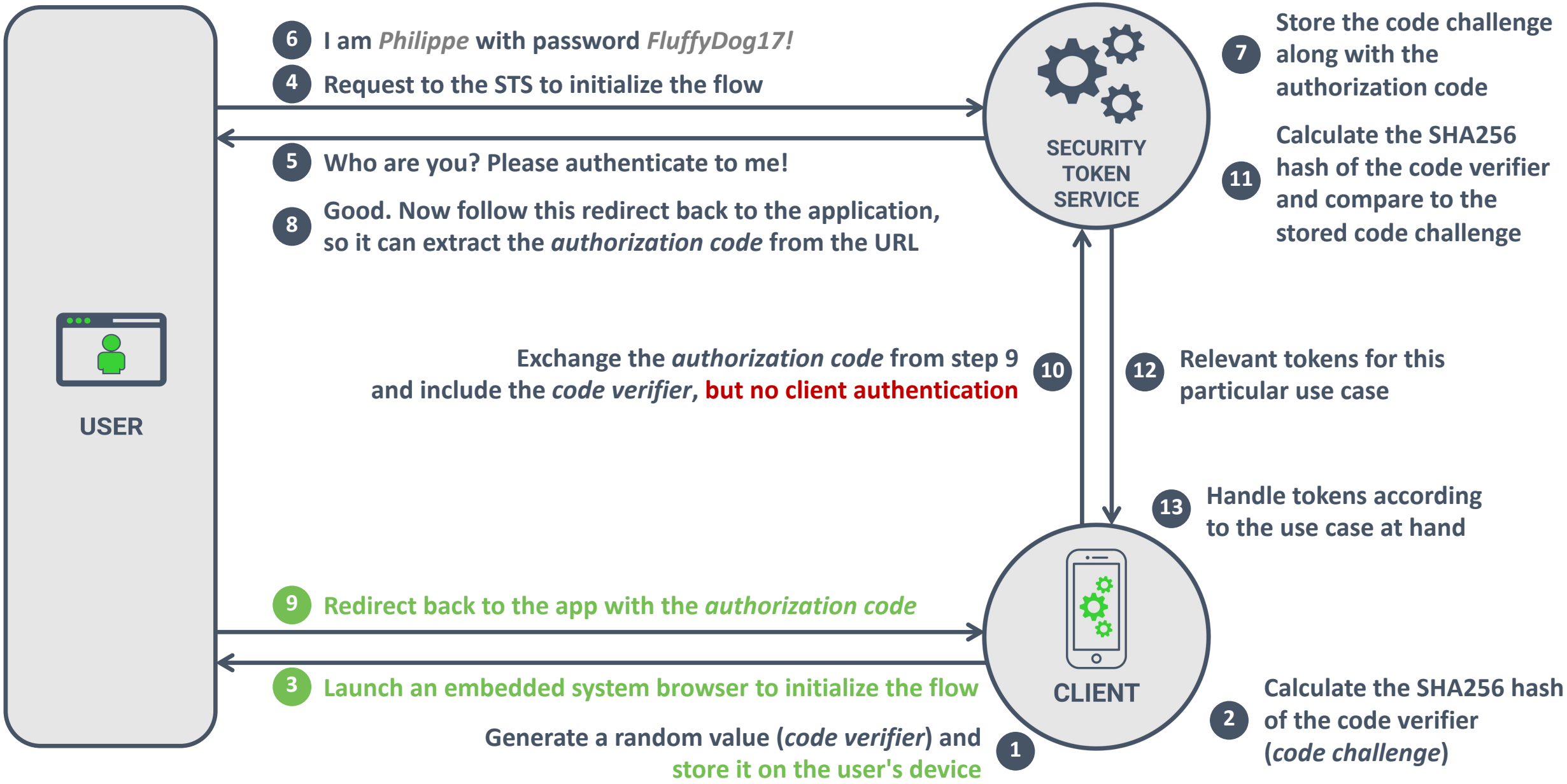
- The ***prompt*** parameter can be used to advise the STS on user interaction
  - Part of the OIDC specification, but supported by most OAuth implementations
  - The value is a space-delimited list with these defined values:
    - The value ***none*** implies that user interaction is not allowed
    - The value ***login*** implies that user authentication is required, even if a session exists
    - The value ***consent*** implies that user consent is required, even if previously given
    - The value ***select_account*** implies that the user has to explicitly select an account

- Running flows without user interaction is useful for background scenarios
  - E.g., running a silent flow during bootstrapping to get tokens if the user is authenticated
  - E.g., running a silent flow to renew access or refresh tokens without prompting the user

- Silent flows only work if the user's browser has an active session with the STS

OAuth 2.0 and OIDC for mobile apps

THE *AUTHORIZATION CODE* FLOW FOR MOBILE APPS



Ongoing work in the OAuth working group is looking into a *native UX* for mobile apps

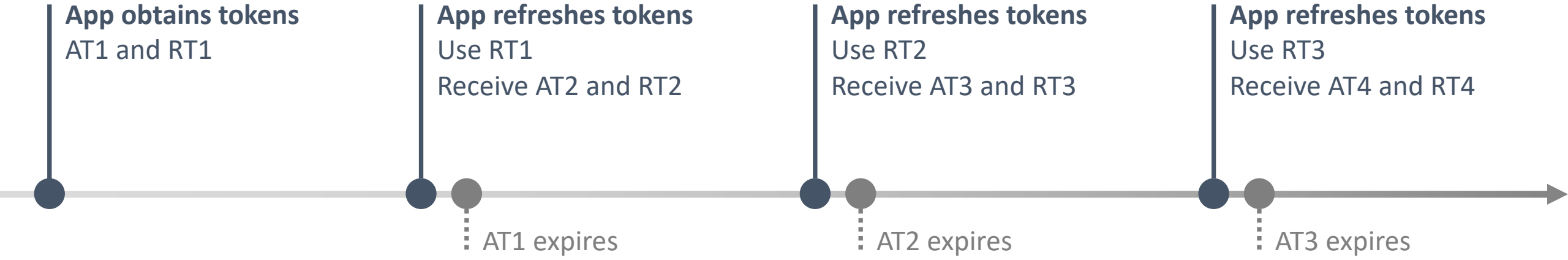
OAUTH AND OIDC FOR MOBILE APPS

- Current best practice for mobile apps is to use the *Authorization Code* flow
  - The mobile app is a public client, without the ability to authenticate to the STS
  - PKCE ensures the security of the flow, since only the right client can exchange the code

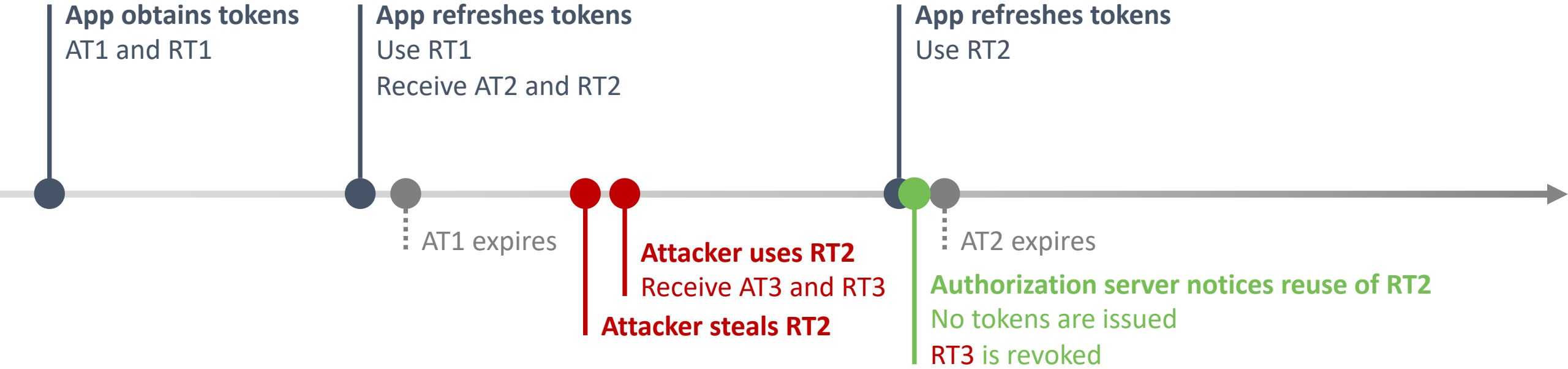
- Mobile apps are supposed to run the flow in an embedded system browser
  - Available as the ***SFSafariViewController*** (iOS) or ***Chrome Custom Tabs*** (Android)
  - This browser is more secure than a webview because the application cannot inspect it
  - The embedded system browser can re-use existing sessions, enabling SSO scenarios

- The mobile app can obtain a refresh token for long-term access
  - Secure token storage options include the OS' keychain, or using OS-protected encryption
  - The use of *refresh token rotation* helps avoid refresh token abuse

Refresh token rotation



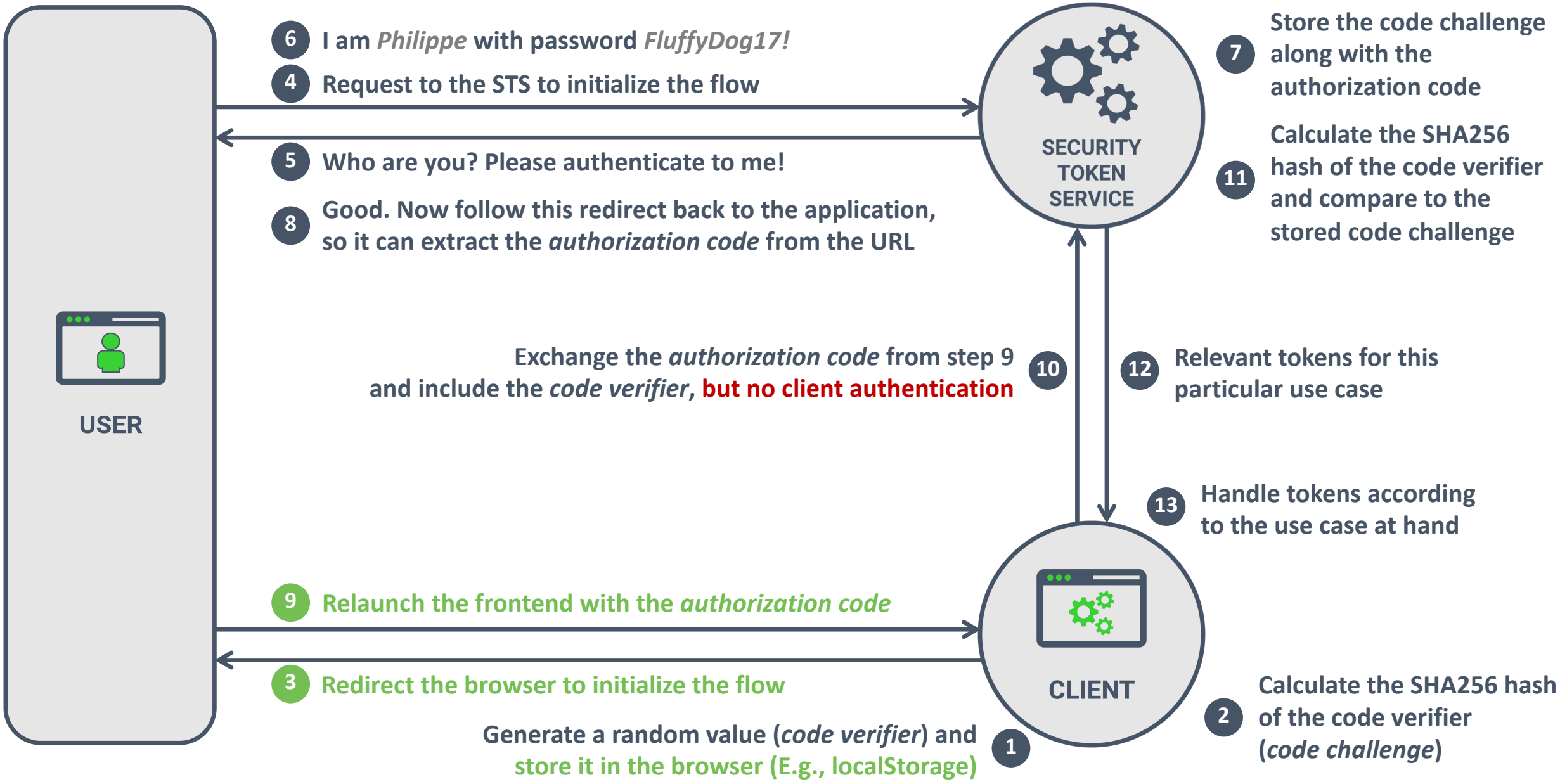
Detecting refresh token abuse



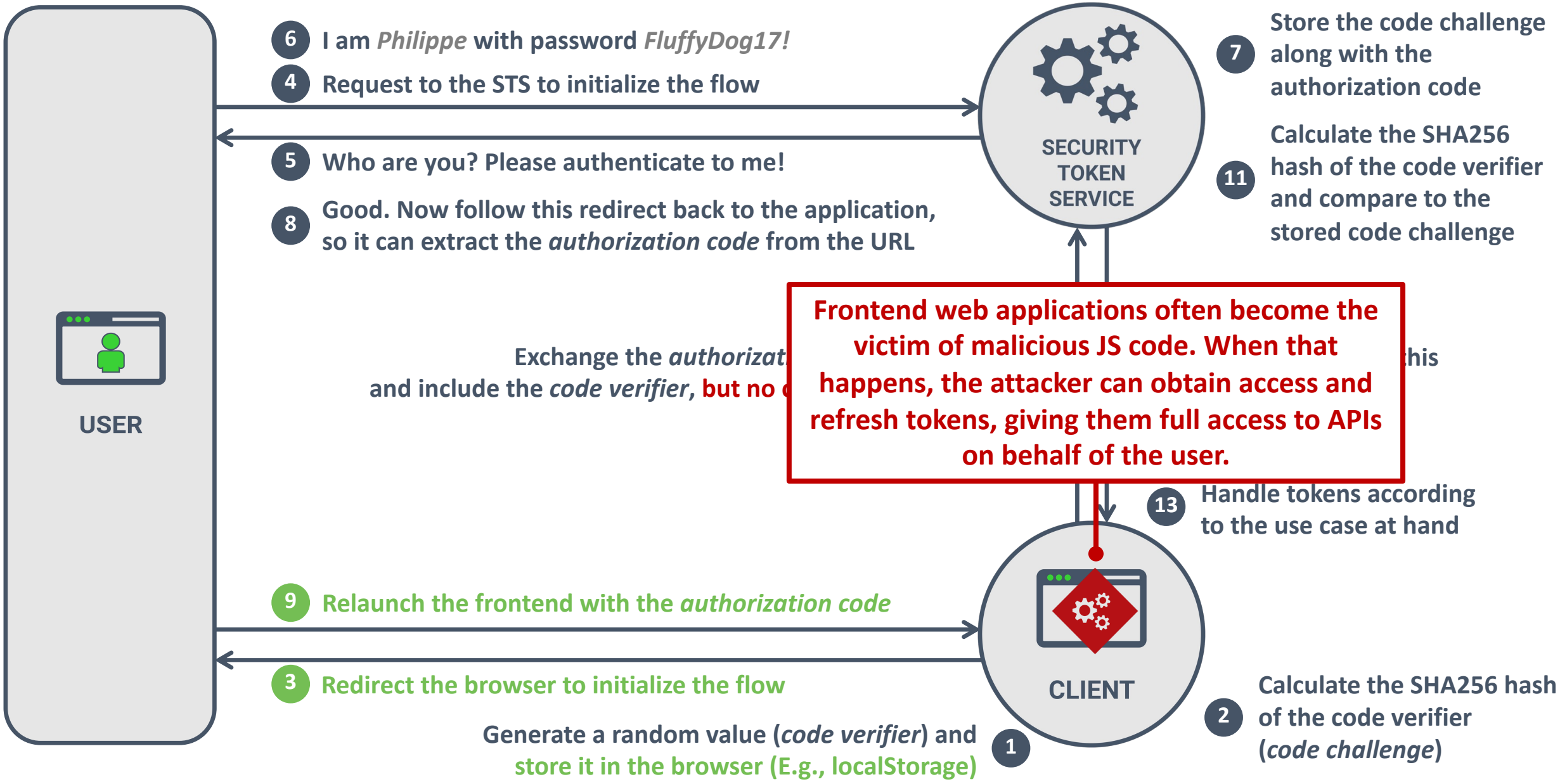
Refresh token rotation in action

OAuth 2.0 and OIDC for web frontends

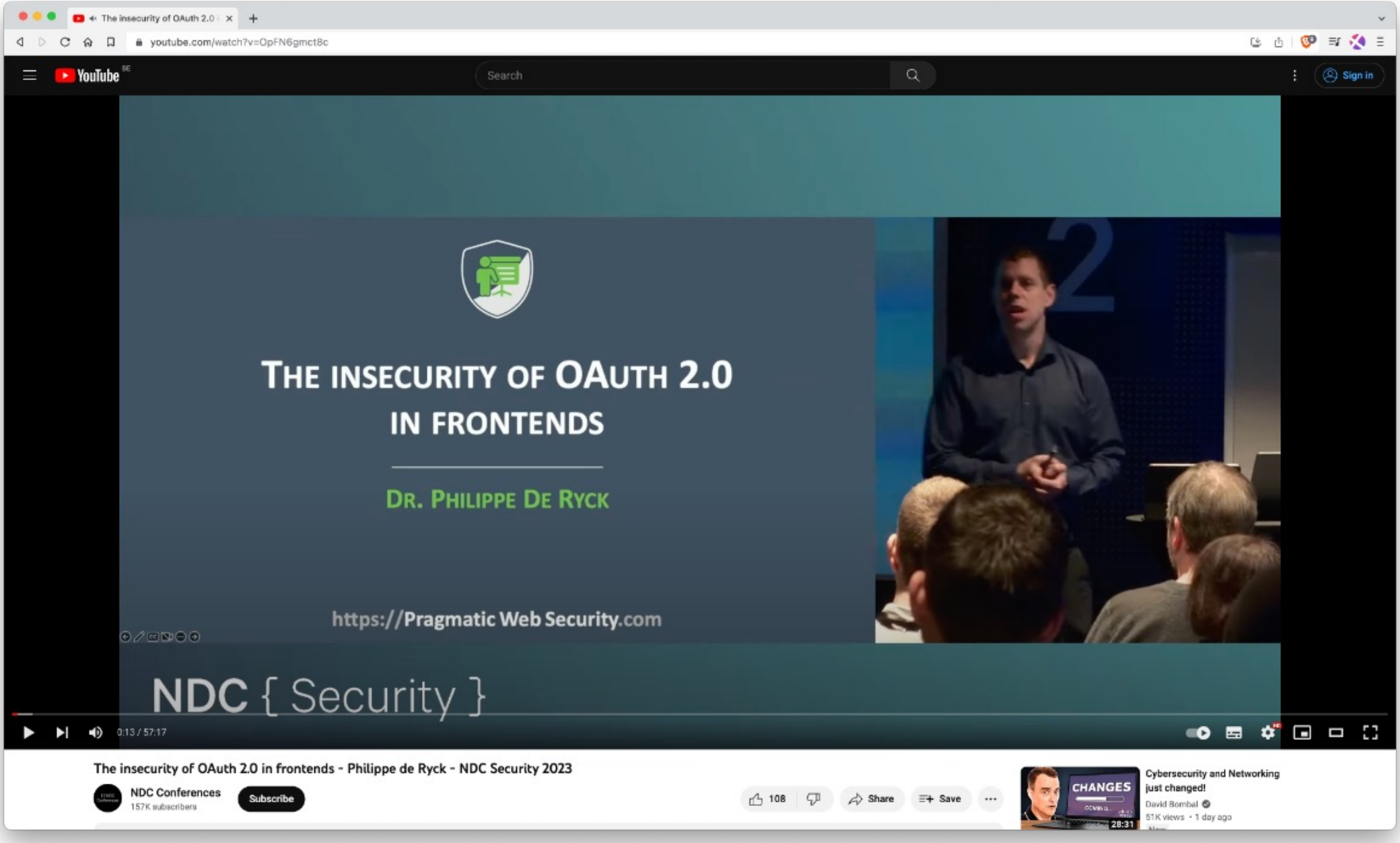
THE *AUTHORIZATION CODE* FLOW FOR FRONTEND WEB APPS



THE *AUTHORIZATION CODE* FLOW FOR FRONTEND WEB APPS



Frontend web applications should use the *Backend-For-Frontend* pattern to secure their OAuth implementations



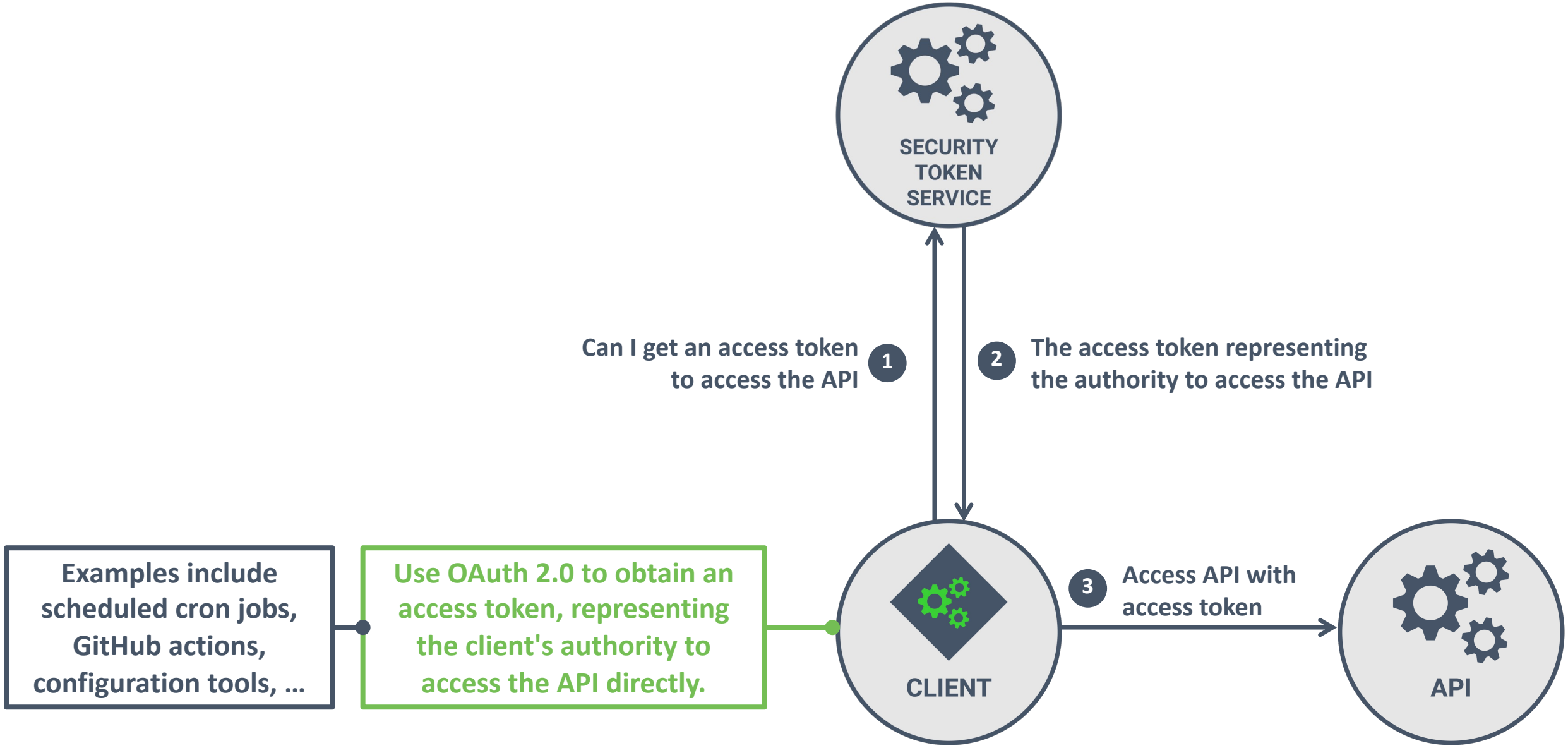
https://www.youtube.com/watch?v=OpFN6gmct8c

15 minute break

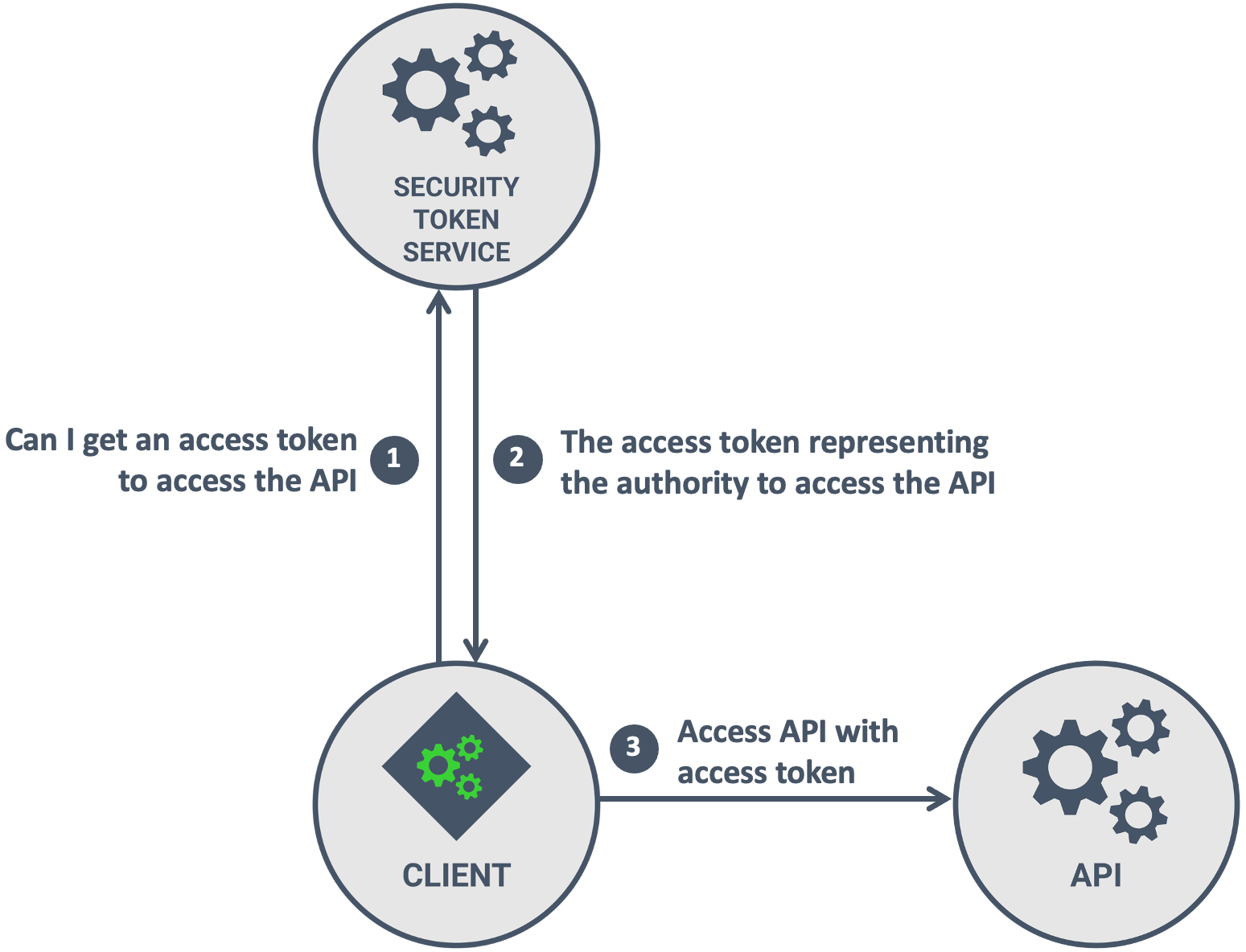
Photo by Pawel Czerwinski on Unsplash

THE *CLIENT CREDENTIALS* FLOW

Using OAuth 2.0 for machine-to-machine access



THE OAUTH 2.0 *CLIENT CREDENTIALS* FLOW

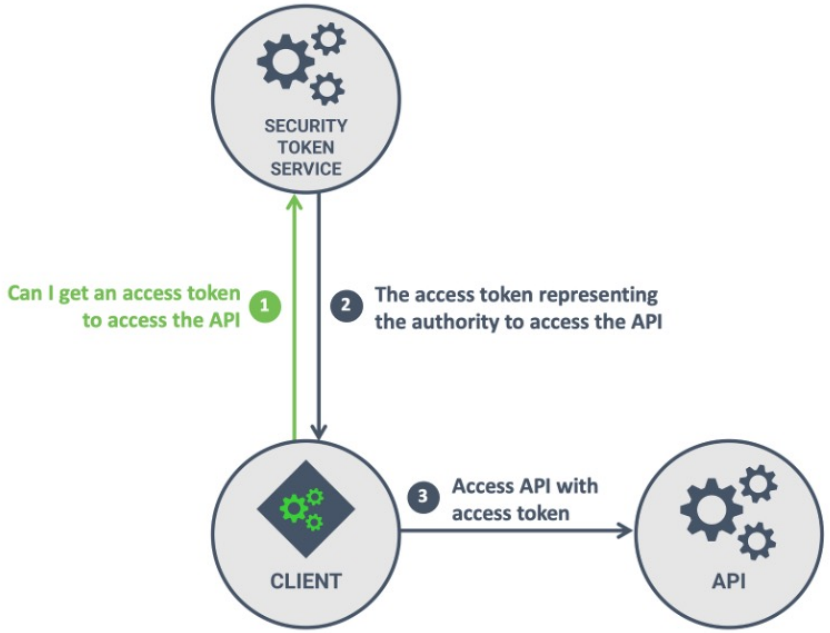


1 *The request to obtain an access token*

```
1  POST /oauth/token
2  Host: sts.restograde.com
3
4   grant_type=client_credentials
5  &client_id=2JqcsqEpZfYNHxDazVMMkPT6oU6C7ZZS
6  &client_secret=xEJRXoe…Vd_BjB
```

- Indicates the *client credentials* flow
- The client exchanging the code
- The client needs to authenticate to the STS

THE OAUTH 2.0 *CLIENT CREDENTIALS* FLOW



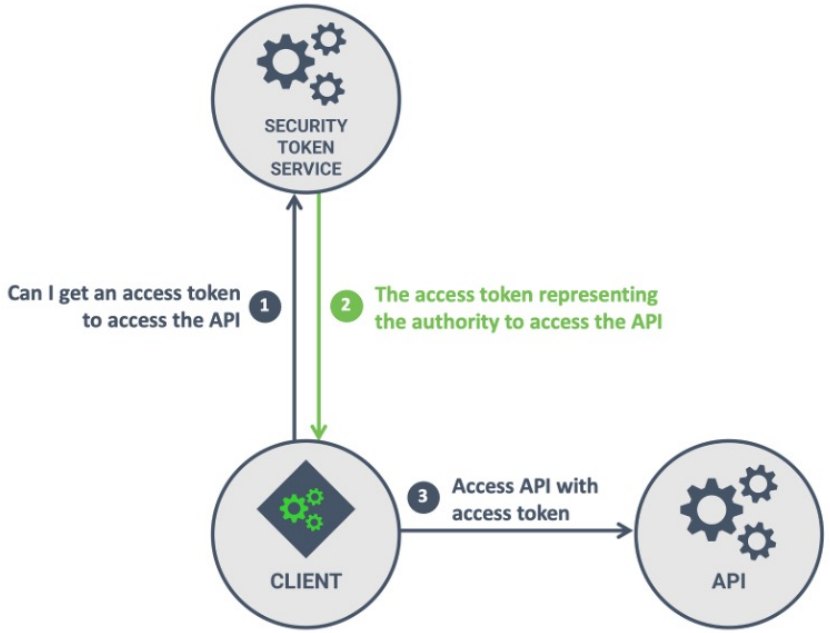
2 The response from the Security Token Service

```
1 {
2   "access_token": "eyJhbGciO…encDDLQ",
3   "token_type": "Bearer",
4   "expires_in": 3600,
5 }
```

The access token to access APIs

The expiration time of the access token

THE OAUTH 2.0 *CLIENT CREDENTIALS* FLOW



THE OAUTH 2.0 *CLIENT CREDENTIALS* FLOW

- The client is another application that needs to access APIs
  - The client is accessing the API directly, on its own behalf
  - There is no user involved in the *Client Credentials* flow
    - This is an OAuth 2.0-only flow, not an OpenID Connect flow, so identity tokens are not used
- The *Client Credentials* flow fits within OAuth 2.0 as an authorization framework
  - The access token issued by the STS represents the client's authority
  - APIs already know how to handle access tokens, so little needs to change
- The *Client Credentials* flow only works with confidential clients
  - Requesting access tokens requires authentication with a secret kept by the client
  - Confidential clients need to run in a secure environment (server-side systems)

The purpose of scopes

The value is a space-delimited string with scope values

Applications can define custom scopes

scope=openid email profile read:reviews

A mechanism provided by OAuth 2.0 to define the scope of an access token

OAuth 2.0 does not define any scope values, but OIDC has a set of reserved scopes

Gmail API, v1

Scopes	
https://mail.google.com/	Read, compose, send, and permanently delete all your email from Gmail
https://www.googleapis.com/auth/gmail.addons.current.action.compose	Manage drafts and send emails when you interact with the add-on
https://www.googleapis.com/auth/gmail.addons.current.message.action	View your email messages when you interact with the add-on
https://www.googleapis.com/auth/gmail.addons.current.message.metadata	View your email message metadata when the add-on is running
https://www.googleapis.com/auth/gmail.addons.current.message.readonly	View your email messages when the add-on is running
https://www.googleapis.com/auth/gmail.compose	Manage drafts and send emails
https://www.googleapis.com/auth/gmail.insert	Insert mail into your mailbox
https://www.googleapis.com/auth/gmail.labels	Manage mailbox labels
https://www.googleapis.com/auth/gmail.metadata	View your email message metadata such as labels and headers, but not the email body
https://www.googleapis.com/auth/gmail.modify	View and modify but not delete your email
https://www.googleapis.com/auth/gmail.readonly	View your email messages and settings
https://www.googleapis.com/auth/gmail.send	Send email on your behalf
https://www.googleapis.com/auth/gmail.settings.basic	Manage your basic mail settings
https://www.googleapis.com/auth/gmail.settings.sharing	Manage your sensitive mail settings, including who can manage your mail

Google Analytics API, v3

Scopes	
https://www.googleapis.com/auth/analytics	View and manage your Google Analytics data
https://www.googleapis.com/auth/analytics.edit	Edit Google Analytics management entities
https://www.googleapis.com/auth/analytics.manage.users	Manage Google Analytics Account users by email address
https://www.googleapis.com/auth/analytics.manage.users.readonly	View Google Analytics user permissions
https://www.googleapis.com/auth/analytics.provision	Create a new Google Analytics account along with its default property and view
https://www.googleapis.com/auth/analytics.readonly	View your Google Analytics data
https://www.googleapis.com/auth/analytics.user.deletion	Manage Google Analytics user deletion requests

Google Sheets API, v4

Scopes	
https://www.googleapis.com/auth/drive	See, edit, create, and delete all of your Google Drive files
https://www.googleapis.com/auth/drive.file	View and manage Google Drive files and folders that you have opened or created with this app
https://www.googleapis.com/auth/drive.readonly	See and download all your Google Drive files
https://www.googleapis.com/auth/spreadsheets	See, edit, create, and delete your spreadsheets in Google Drive
https://www.googleapis.com/auth/spreadsheets.readonly	View your Google Spreadsheets

Google Sign-In

Scopes	
profile	View your basic profile info
email	View your email address
openid	Authenticate using OpenID Connect

Google Site Verification API, v1

Scopes	
https://www.googleapis.com/auth/siteverification	Manage the list of sites and domains you control
https://www.googleapis.com/auth/siteverification.verify_only	Manage your new site verifications with Google

Google Slides API, v1

Scopes	
https://www.googleapis.com/auth/drive	See, edit, create, and delete all of your Google Drive files
https://www.googleapis.com/auth/drive.file	View and manage Google Drive files and folders that you have opened or created with this app
https://www.googleapis.com/auth/drive.readonly	See and download all your Google Drive files
https://www.googleapis.com/auth/presentations	View and manage your Google Slides presentations
https://www.googleapis.com/auth/presentations.readonly	View your Google Slides presentations
https://www.googleapis.com/auth/spreadsheets	See, edit, create, and delete your spreadsheets in Google Drive
https://www.googleapis.com/auth/spreadsheets.readonly	View your Google Spreadsheets

https://developers.google.com/identity/protocols/oauth2/scopes

GitHub

Available scopes

Name	Description
`(no scope)`	Grants read-only access to public information (includes public user profile info, public repository info, and gists)
`repo`	Grants full access to private and public repositories. That includes read/write access to code, commit statuses, repository and organization projects, invitations, collaborators, adding team memberships, deployment statuses, and repository webhooks for public and private repositories and organizations. Also grants ability to manage user projects.
`repo:status`	Grants read/write access to public and private repository commit statuses. This scope is only necessary to grant other users or services access to private repository commit statuses *without* granting access to the code.
`repo_deployment`	Grants access to deployment statuses for public and private repositories. This scope is only necessary to grant other users or services access to deployment statuses, *without* granting access to the code.
`public_repo`	Limits access to public repositories. That includes read/write access to code, commit statuses, repository projects, collaborators, and deployment statuses for public repositories and organizations. Also required for starring public repositories.
`repo:invite`	Grants accept/decline abilities for invitations to collaborate on a repository. This scope is only necessary to grant other users or services access to invites *without* granting access to the code.
`security_events`	Grants read and write access to security events in the code scanning API.
`admin:repo_hook`	Grants read, write, ping, and delete access to repository hooks in public and private repositories. The `repo` and `public_repo` scopes grants full access to repositories, including repository hooks. Use the `admin:repo_hook` scope to limit access to only repository hooks.
`write:repo_hook`	Grants read, write, and ping access to hooks in public or private repositories.
`read:repo_hook`	Grants read and ping access to hooks in public or private repositories.
`admin:org`	Fully manage the organization and its teams, projects, and memberships.
`write:org`	Read and write access to organization membership, organization projects, and team membership.
`read:org`	Read-only access to organization membership, organization projects, and team membership.

`admin:org`	Fully manage the organization and its teams, projects, and memberships.
`write:org`	Read and write access to organization membership, organization projects, and team membership.
`read:org`	Read-only access to organization membership, organization projects, and team membership.
`admin:public_key`	Fully manage public keys.
`write:public_key`	Create, list, and view details for public keys.
`read:public_key`	List and view details for public keys.
`admin:org_hook`	Grants read, write, ping, and delete access to organization hooks. **Note:** OAuth tokens will only be able to perform these actions on organization hooks which were created by the OAuth App. Personal access tokens will only be able to perform these actions on organization hooks created by a user.
`gist`	Grants write access to gists.
`notifications`	Grants: * read access to a user's notifications * mark as read access to threads * watch and unwatch access to a repository, and * read, write, and delete access to thread subscriptions.
`user`	Grants read/write access to profile info only. Note that this scope includes `user:email` and `user:follow`.
`read:user`	Grants access to read a user's profile data.
`user:email`	Grants read access to a user's email addresses.
`user:follow`	Grants access to follow or unfollow other users.
`delete_repo`	Grants access to delete adminable repositories.
`write:discussion`	Allows read and write access for team discussions.
`read:discussion`	Allows read access for team discussions.
`write:packages`	Grants access to upload or publish a package in GitHub Packages. For more information, see "Publishing a package" in the GitHub Help documentation.
`read:packages`	Grants access to download or install packages from GitHub Packages. For more information, see "Installing a package" in the GitHub Help documentation.
`delete:packages`	Grants access to delete packages from GitHub Packages. For more information, see "Deleting packages" in the GitHub Help documentation.

https://docs.github.com/en/apps/oauth-apps/building-oauth-apps/scopes-for-oauth-apps

Practical guidelines for defining scopes

- Unless you are Google, you probably do not need hundreds of scopes
  - People sometimes run into length limits for the scope parameter, which is a bad smell
  - If clients need access to every API in the system, then you don't need scopes
    - Scopes enforce compartmentalization, but do not replace existing authorization systems

- Guidelines to define scopes
  - Start by identifying logical groupings in the APIs
    - E.g., ***reviews*** and ***restaurants***
  - Determine if different access levels are needed
    - E.g., ***restaurants*** is used by a single client
    - E.g., ***read:reviews*** is for third-party clients
  - Isolate extremely sensitive permissions
    - E.g., ***delete:reviews*** is only possible after consent

Permission	Description
`read:reviews`	Read reviews
`write:reviews`	Write reviews
`delete:reviews`	Delete reviews
`restaurants`	Manage restaurant information

Scopes allow the user to delegate a subset of their full authority to a client application

Using scopes

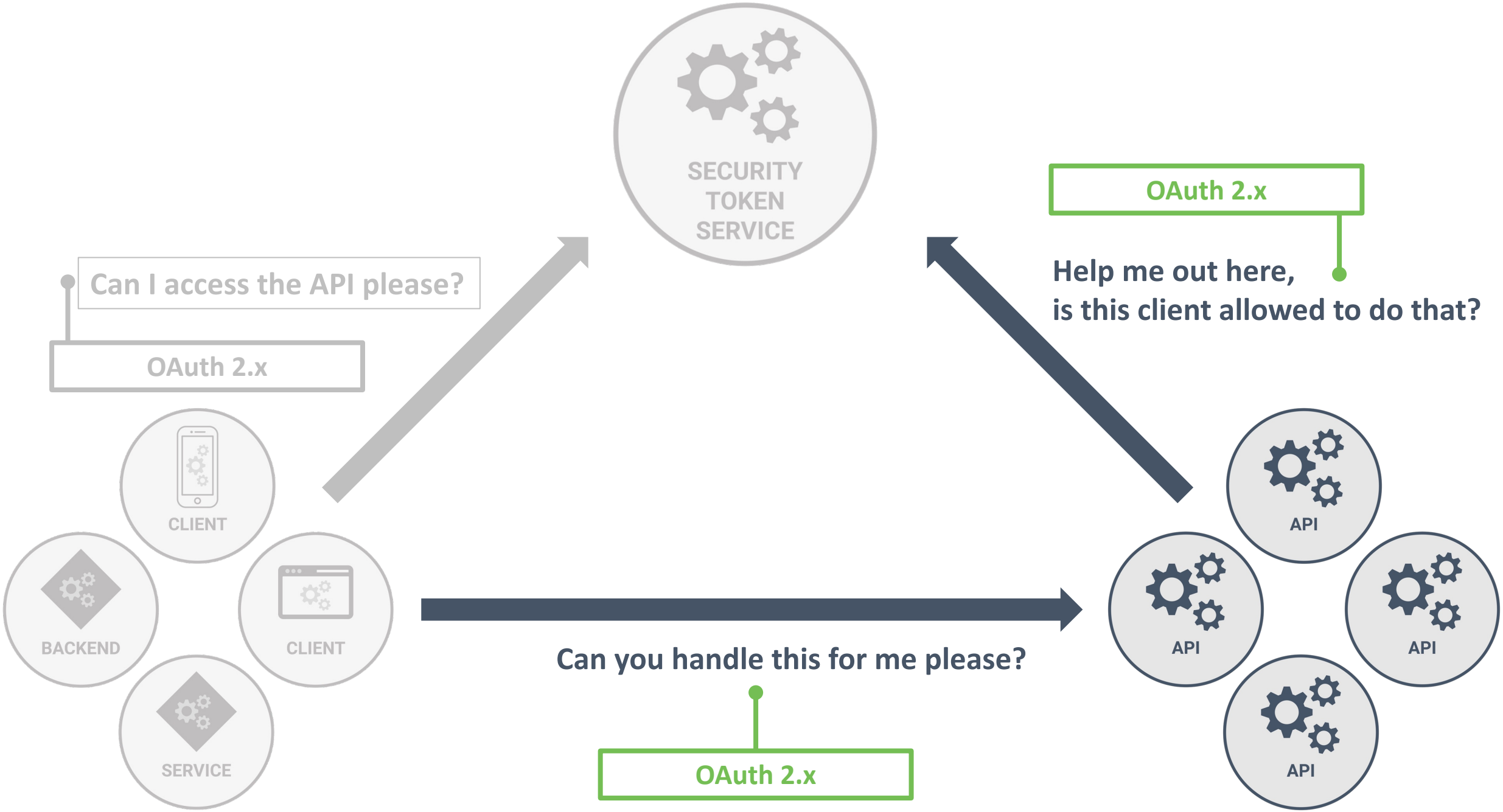
Scopes and their limitations

- Scopes were initially defined to reduce the authority given to a client
  - Scopes are closely linked to user consent, which is relevant in third-party scenarios
  - Statically defined scopes are mainly useful for static delegation scenarios

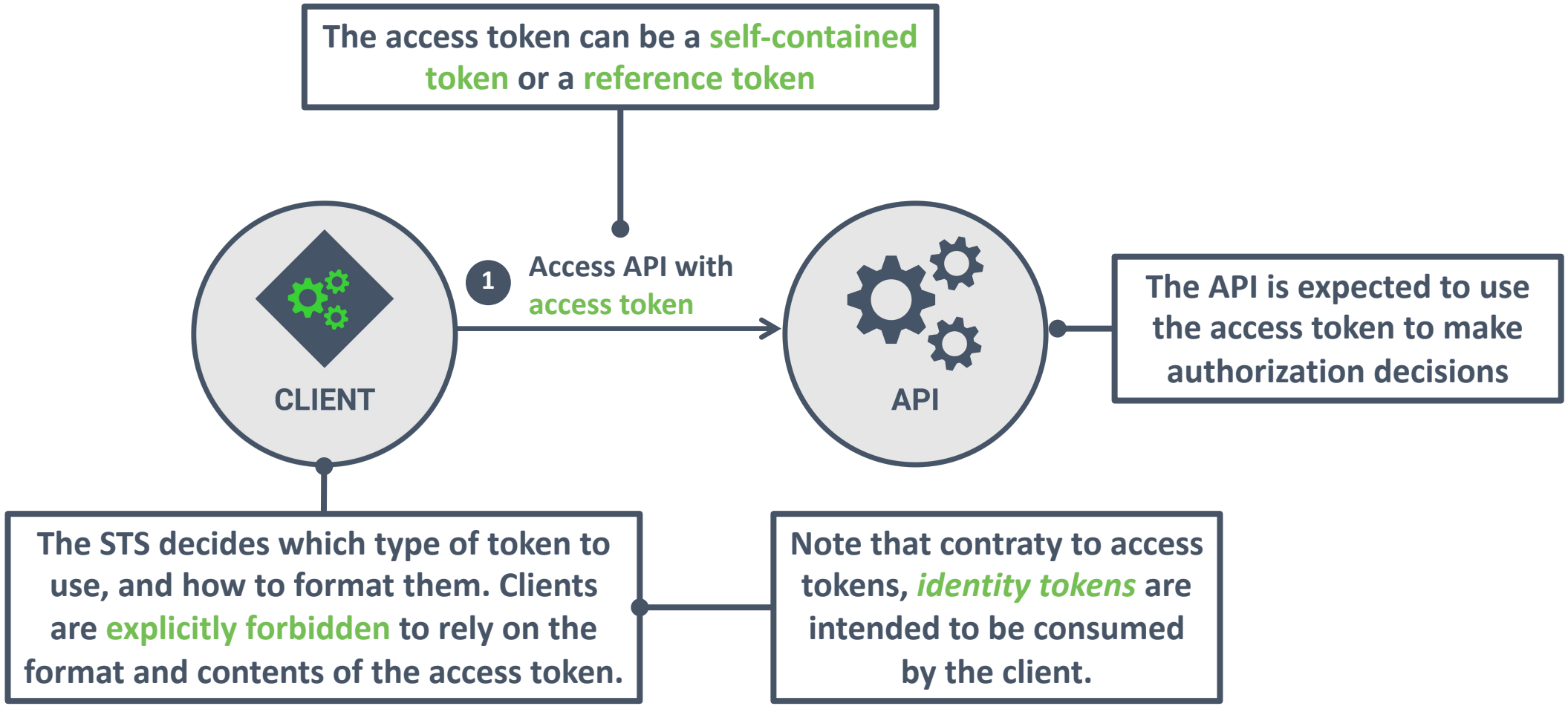
- Advanced use cases often use dynamic scopes that relate to business domains
  - With a dynamic scope, clients can request the authority to access specific objects
  - Dynamic scopes require a close coupling between the STS and authorization logic

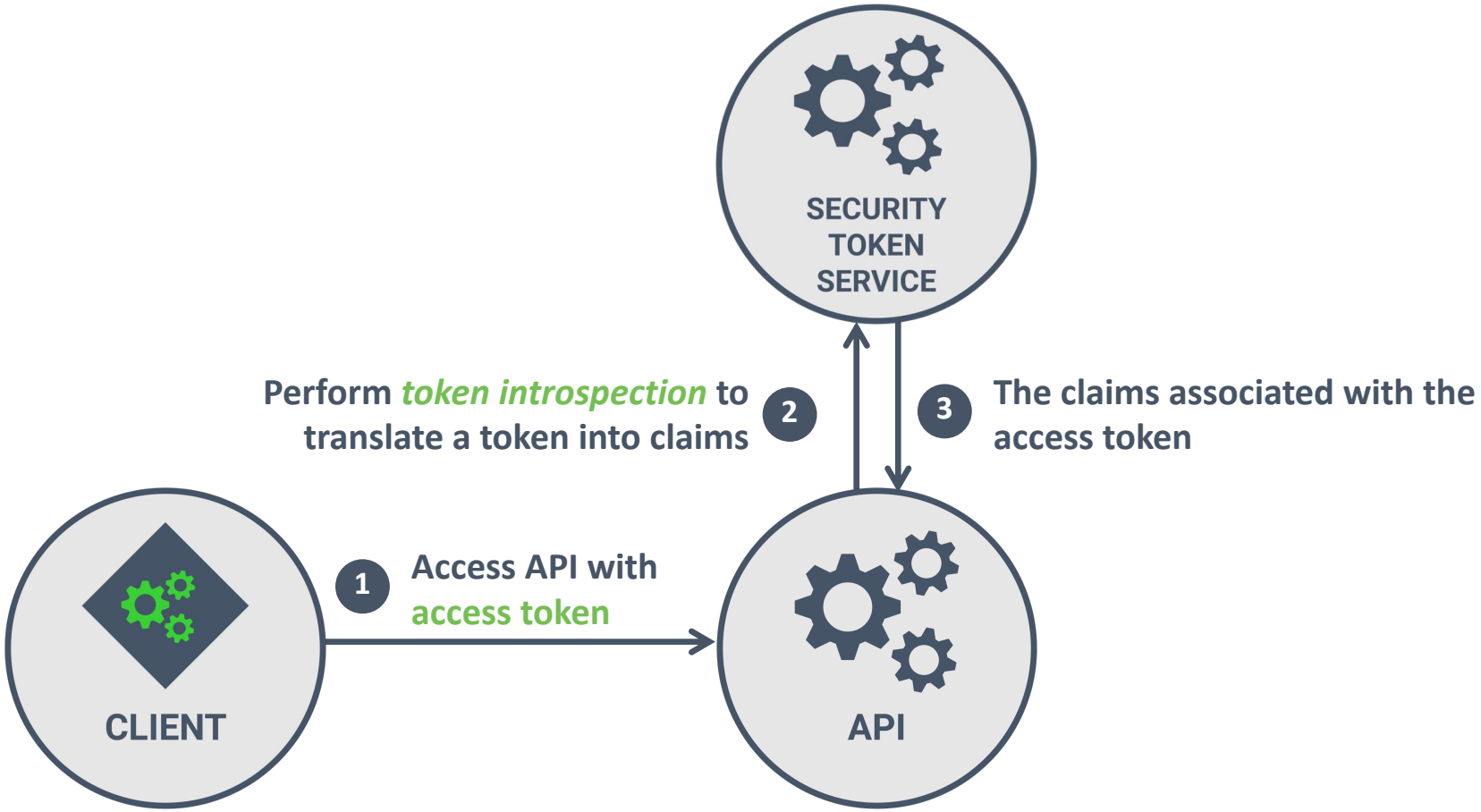
- Rich Authorization Requests (RAR) further enhance the concept of scopes
  - RAR is a recent addition to the OAuth landscape, aimed to support complex scenarios
  - E.g., a client can request the authority to perform a wire transfer for a certain amount

Access tokens and access token types


SECURITY
TOKEN
SERVICE
OAuth 2.x
Help me out here,
is this client allowed to do that?
Can I access the API please?
OAuth 2.x
CLIENT
BACKEND
CLIENT
SERVICE
API
API
API
API
Can you handle this for me please?
OAuth 2.x

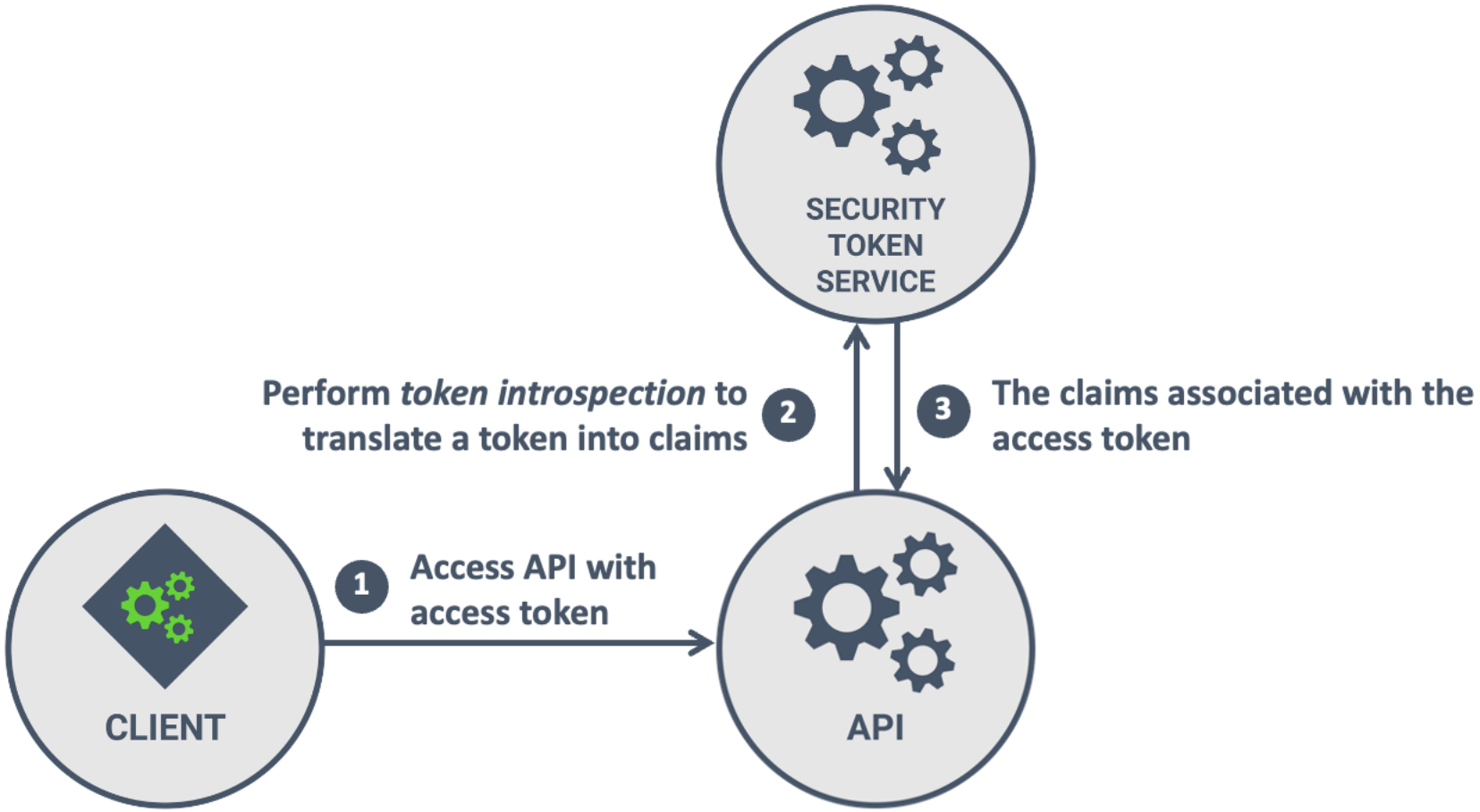
pdr.online


The access token can be a self-contained token or a reference token
CLIENT
1
Access API with access token
API
The API is expected to use the access token to make authorization decisions
The STS decides which type of token to use, and how to format them. Clients are explicitly forbidden to rely on the format and contents of the access token.
Note that contraty to access tokens, identity tokens are intended to be consumed by the client.



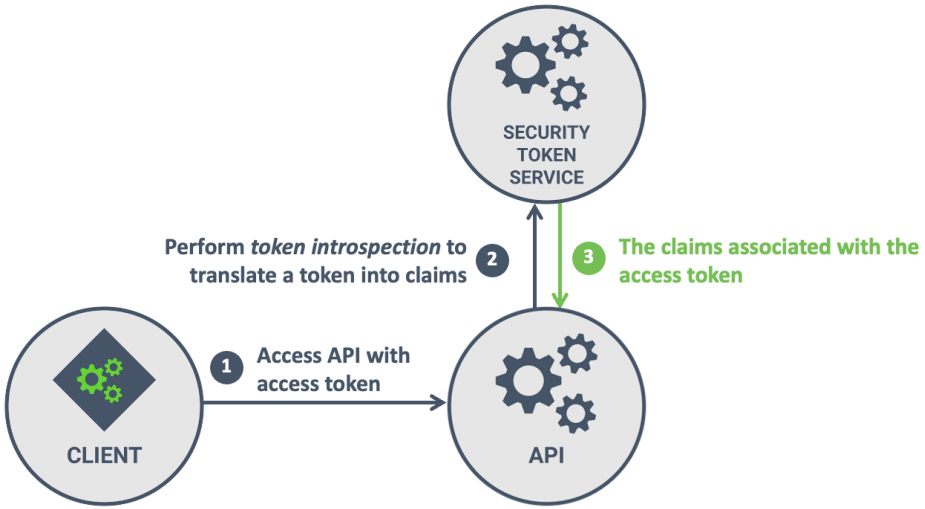
A reference token

```
vSvhNDeQLqrzRbvA2eeYE2PthB1cBimS
```


SECURITY TOKEN SERVICE
Perform *token introspection* to translate a token into claims
2
3
The claims associated with the access token
CLIENT
1
Access API with access token
API

3 The token introspection response

```
{
  "active": true,
  "iss": "https://sts.restograde.com",
  "sub": "2262430d-c9cb-484f-9770-805893ff9518",
  "scope": "reviews:read",
  …
}
```

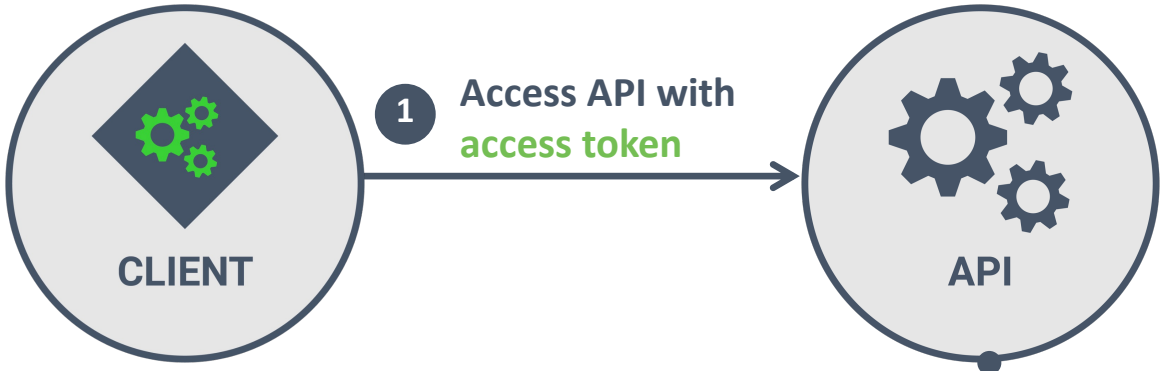


Token introspection

- The fields returned are all marked as optional, except for ***active***
  - The ***active*** field indicates if a token is still valid or not
  - The other fields are only present if a token is valid and provide context information
- Ultimately, the STS is in control over what is returned during introspection
  - The returned information can include custom fields
  - Depending on which API is asking, more or less information may be included
- The main benefit of reference tokens is the high degree of control by the STS
  - Revoked tokens will be invalid the next time they are introspected
  - The downside of reference tokens is the mandatory token introspection step

A self-contained access token


```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZC
I6Ik5UVkJPVFUzTXpCQk9FVXdOemhCUTBWR01rUTBR
VVU1UVRZeFFVVXlPVU5FUVVVeE5qRXlNdyJ9.eyJpc
3MiOiJodHRwczovL3N0cy5yZXN0b2dyYWRlLmNvbS8
iLCJzdWIiOiJhdXRoMHw1ZWI5MTZjMjU4YmRiNTBiZ
jIwMzY2YzYiLCJhdWQiOlsiaHR0cHM6Ly9hcGkucmV
zdG9ncmFkZS5jb20iLCJodHRwczovL3Jlc3RvZ3JhZ
GUuZXUuYXV0aDAuY29tL3VzZXJpbmZvIl0sImlhdCI
6MTU4OTc3NTA3MiwiZXhwIjoxNTg5ODYxNDcyLCJhe
nAiOiJPTEtObjM4OVNVSW11ZkV4Z1JHMVJpbExTZ2R
ZeHdFcCIsInNjb3BlIjoib3BlbmlkIHByb2ZpbGUgZ
W1haWwgb2ZmbGluZV9hY2Nlc3MifQ.XzJOXtTXOGOS
bCFvp4yZGJzh7XhMmOmI2XxtjWdlODz_siI-u8h11e
lcr8LwX6-hL20QOW0eStzBzmm1FM_tS7MxuKkYx8Ql
TWOURPembVKZOhNi8kN-1j0pyc0uzve7Jib5vcxmkP
wqpcVDFACgP85_0NYe4zXHKxCA5_8VOn05cRCDSkNM
TFzGJCT9ipCcNXaVGdksojYGqQzezjpzzzwrtPEkiy
FLFtDPZAl0MleF3oFAOCBK0UKuNjJ_cSBbUsaIwfvK
0WH47AwFrRn_TxL4S1P3j3b1GgBm8tAqXysY84VZu0
rSg3zrZj1PnoqPD4mbOXds20xafCr9wR4WTQ
```

To validate a self-contained token, the API uses the public key of the STS, which can be loaded from a dedicated endpoint on the STS. Once loaded, the public key can be reused for future token validations

Self-contained tokens are typically formatted as JWTs and are signed by the STS

The contents of the self-contained token are similar claims as obtained through token introspection.

Verifying self-contained access tokens

- The API is typically configured with a trusted STS
  - The STS will provide access tokens, which will be used to make authorization decisions
  - With the URL of the STS, the API can bootstrap its token verification mechanism
  - The API ***must*** verify the integrity of a self-contained access token before using the data
- Access token verification is typically implemented in middleware
  - Barebones JWT libraries can handle most of these details
  - Many languages offer *resource server* libraries, which deal with access tokens specifically
- The introspection RFC also allows token introspection for self-contained tokens
  - Introspecting JWTs can be used to detect revocation before the token expires

Which token type is right for you?

The trade-off is between security and performance

Reference tokens vs self-contained tokens

- Due to the perfomance impact, token introspection is often only used locally
  - SaaS-based STS implementations often do not support reference tokens
  - APIs can handle token introspection, but gateways often take this responsibility

- Reference tokens are easy to revoke before they expire
  - Revoking self-contained tokens is possible, but requires propagating this info to all APIs
  - ***Relying on fast revocation is typically handled automatically, not manually***
    - E.g., an anomaly-detection system that revokes tokens from suspicious requests

- Both reference tokens and self-contained access tokens have a limited lifetime
  - When an access token expires, the client uses a refresh token to contact the STS
  - Refresh tokens can also easily be revoked, preventing the issuing of a new access token
  - ***Short access token lifetimes (e.g., 5 – 10 mins) improve revocation properties***

Making authorization decisions with access tokens

The client provides the access token

Verify JWT signature and check *exp*, *nbf*, and *iss* claims

Introspect the token with the STS and check the *active* claim

At the end of this step, the access token is transformed into a uniform set of claims

Check additional/optional API-specific claims (E.g., *aud*)

The audience is used by the *Resource Indicators* spec & some STS implementations

Check generic authorization (E.g., *scope*)

Generic authorization checks often correspond to function-level access control

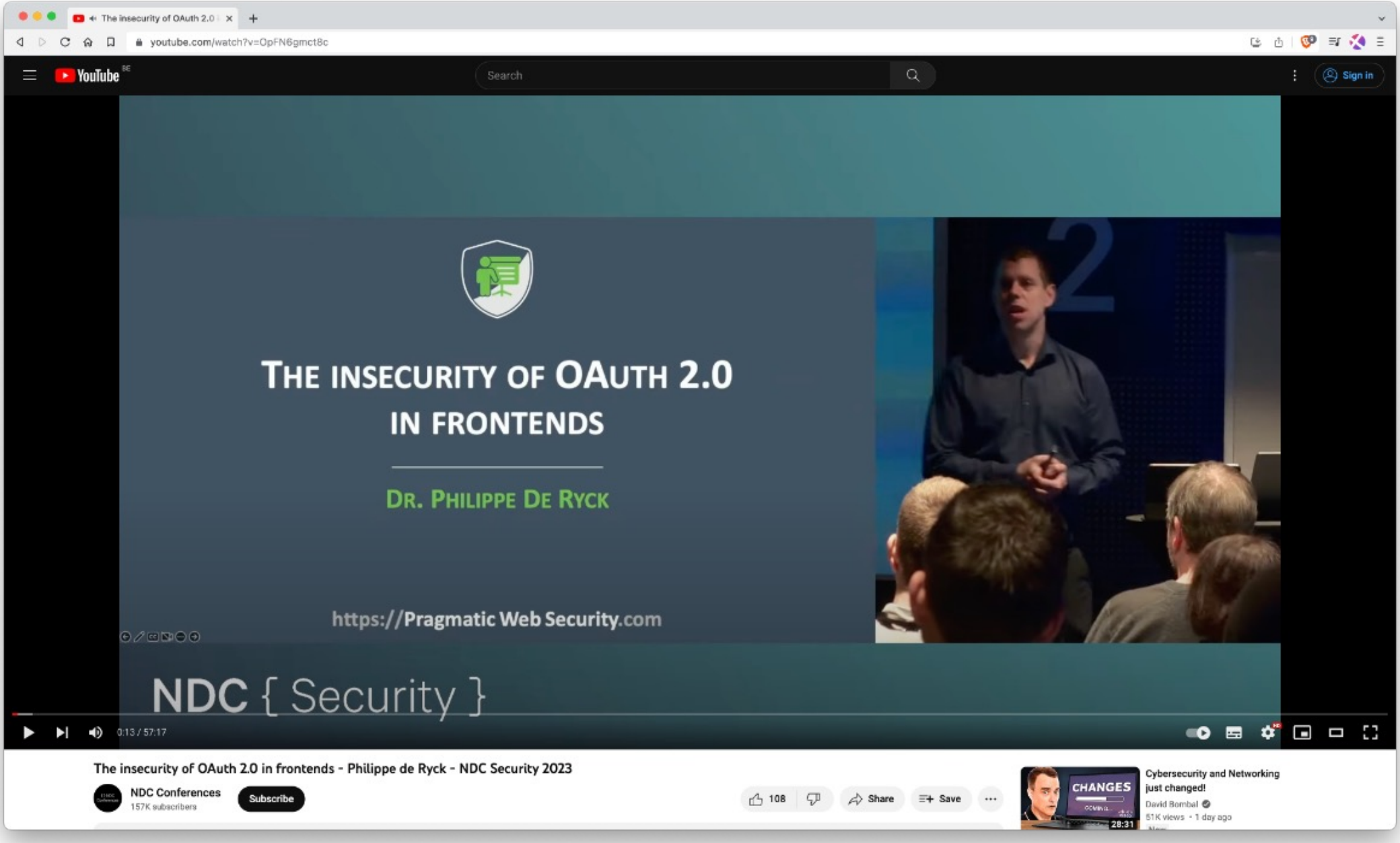
Make specific authorization decisions
(E.g., check the user, the client ID, a customer ID, ...)

The *sub* claim contains the user ID (if relevant), but custom claims can contain all kinds of data

Implementing API authorization

And there's more …

And there's more, so much more …



https://www.youtube.com/watch?v=OpFN6gmct8c

Internet Engineering Task Force (IETF)
Request for Comments: 8707
Category: Standards Track
Published: February 2020
ISSN: 2070-1721

B. Campbell
Ping Identity
J. Bradley
Yubico
H. Tschofenig
Arm Limited

Resource Indicators for OAuth 2.0

Abs

Internet Engineering Task Force (IETF)
Request for Comments: 9126
Category: Standards Track
Published: September 2021
ISSN: 2070-1721

T. Lodderstedt
yes.com
B. Campbell
Ping Identity
N. Sakimura
NAT.Consulting
D. Tonge
Moneyhub Financial
Technology
F. Skokan
Auth0

OAuth 2.0 Pushed Authorization Requests

Abstract

This document c
endpoint, which
authorization r
request and pro
reference to th
endpoint.

Internet Engineering Task Force (IETF)
Request for Comments: 9396
Category: Standards Track
Published: May 2023
ISSN: 2070-1721

Bespol

OAuth 2.0 Rich Authorization Requests

Abstract

This document specifies a new parameter authorization_details that is used to carry fine-grained authorization data in OAuth messages.

Internet Engineering Task Force (IETF)
Request for Comments: 9101
Category: Standards Track
Published: August 2021
ISSN: 2070-1721

N. Sakimura
NAT.Consulting
J. Bradley
Yubico
M. Jones
Microsoft

The OAuth 2.0 Authorization Framework: JWT-Secured Authorization Request (JAR)

Abstract

The authorization request in OAuth 2.0 described in RFC 6749 utilizes query parameter serialization, which means that authorization request parameters are encoded in the URI of the request and sent through user agents such as web browsers. While it is easy to implement, it means that a) the communication through the user agents is not

Workgroup: fapi
Published: 13 September 2023
Authors: D. Fett, *Authlete*; D. Tonge, *Moneyhub Financial Technology*

FAPI 2.0 Security Profile — draft

Foreword

The OpenID Foundation (OIDF) promotes, protects and nurtures the OpenID community and technologies. As a non-profit international standardizing body, it is comprised by over 160 participating entities (workgroup participant). The work of preparing implementer drafts and final international standards is carried out through OIDF workgroups in accordance with the OpenID Process. Participants interested in a subject for which a workgroup has been established have the right to be represented in that workgroup. International organizations, governmental and non-governmental, in liaison with OIDF, also take part in the work. OIDF collaborates closely with other standardizing bodies in the related fields.¶

Final drafts adopted by the Workgroup through consensus are circulated publicly for the public review for 60 days and for the OIDF members for voting. Publication as an OIDF Standard requires approval by at least 50% of the members casting a vote. There is a possibility that some of the elements of this document may be subject to patent rights. OIDF shall not be held responsible for identifying any or all such patent rights.

Mastering OAuth 2.0 and Ope

pragmaticwebsecurity.com/courses/mastering-oauth-oidc

TRAINING SERVICES PUBLIC EVENTS ONLINE COURSES RESOURCES ABOUT CONTACT

Mastering OAuth 2.0 and OpenID Connect

An intense deep-dive on the latest best practices

OAuth 2.0 and OpenID Connect are crucial for securing web applications, mobile applications, APIs, and microservices. Unfortunately, getting a good grip on the purpose and use cases for these technologies is insanely difficult. As a result, **many implementations use incorrect configurations or contain security vulnerabilities**.

Let me tell you how I felt when I started digging into OAuth 2.0 and OpenID Connect a few years ago. I had a hard time understanding what OAuth 2.0 and OpenID Connect were supposed to solve. The terminology made it difficult to understand what the spec was even talking about. And the flows! Each use case had a different flow, and the differences between the flows are often tiny details.

Do you recognize your struggle here? If you are feeling frustrated and overwhelmed with OAuth 2.0 and OpenID Connect, **this course is going to help you**. This course takes you on a step-by-step journey into the world of OAuth 2.0 and OpenID Connect. It explains the purpose of each technology, along with its use cases. In the course, we also dig deep into concrete scenarios, enabling you to design and implement secure applications with OAuth 2.0 and OpenID Connect.

In a matter of hours, you gain the knowledge that usually takes months of researching dozens of specifications. Throughout the course, we build up a set of best practices that allow you to secure modern applications.

I have worked with thousands of developers to help them understand the ins and outs of OAuth 2.0 and OpenID Connect. I have seen them struggle, but I have also seen them succeed. That's why I can promise you that **by the end of this course, you will be confident** in using OAuth 2.0 and OpenID Connect.

This course is your shortcut towards understanding OAuth 2.0 and OpenID Connect.

JOIN THIS LIVE TRAINING

https://bit.ly/oauthcourse

Conclusion

KEY TAKEAWAYS

1. OAuth 2.0 allows a client to access APIs (on behalf of a user)
2. OpenID Connect allows a client to offload authentication
3. User-facing apps use the *Authorization Code* flow with PKCE

Thank you!

Connect on LinkedIn
to stay in touch

PhilippeDeRyck

Want more?

Join me in November for an 8 in-depth sessions on *OAuth 2.x and OpenID Connect*

https://bit.ly/oauthcourse