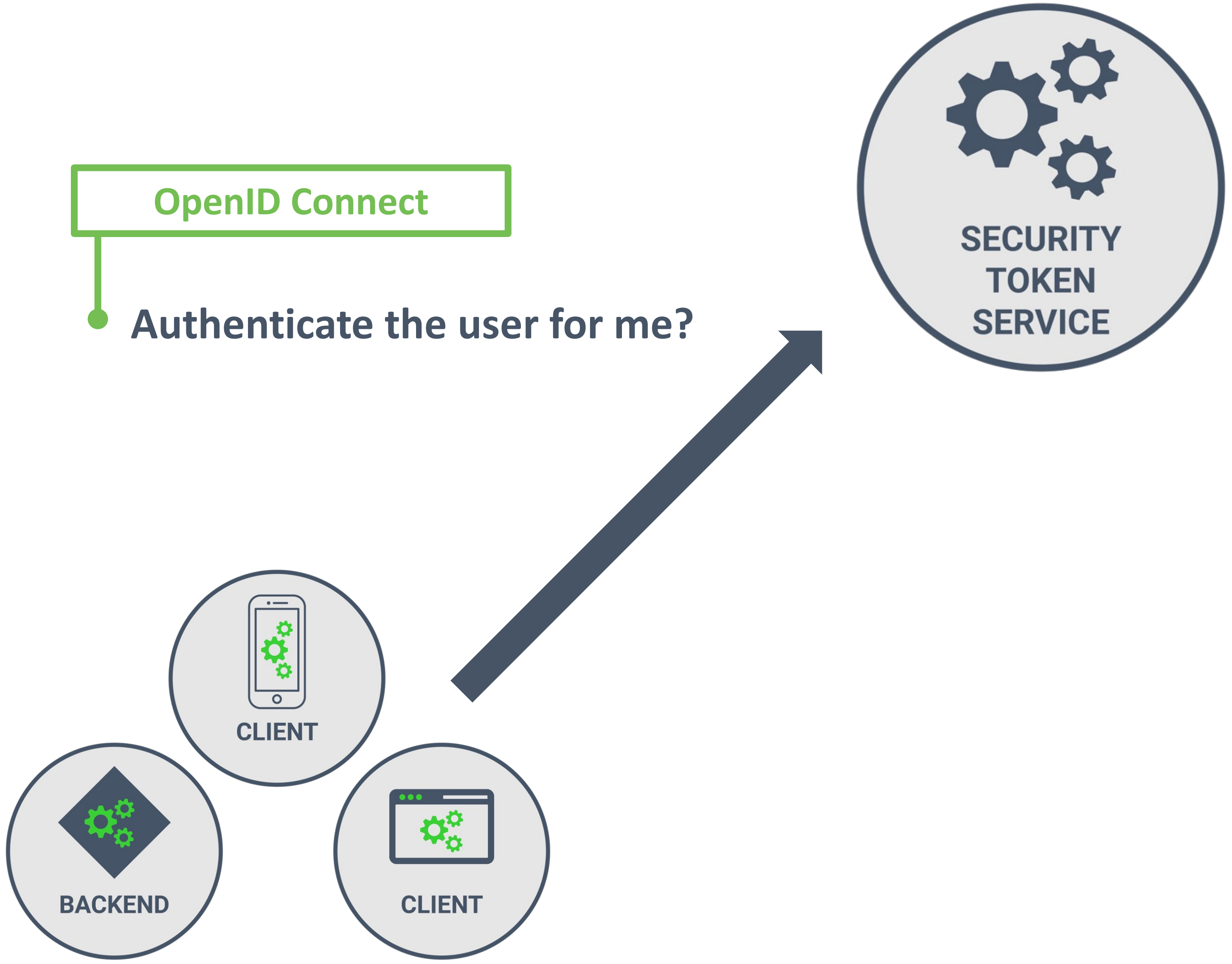
OAuth 2.0 and OpenID Connect architectures

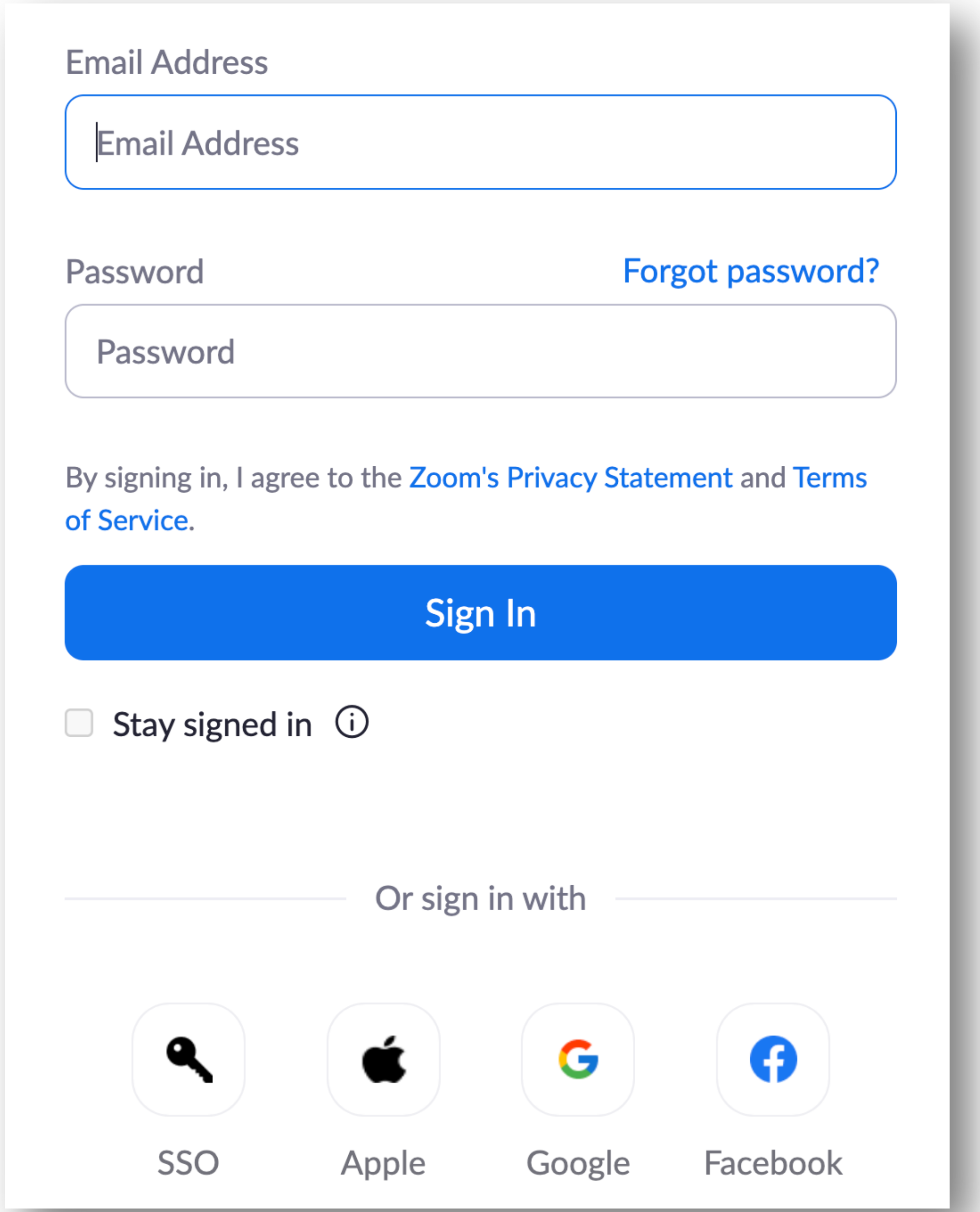
Dr. Philippe De Ryck

https://PragmaticWebSecurity.com

OpenID Connect

Authenticate the user for me?





Email Address

Email Address

Password

Forgot password?

Password

By signing in, I agree to the Zoom's Privacy Statement and Terms of Service.

Sign In

Stay signed in

Or sign in with

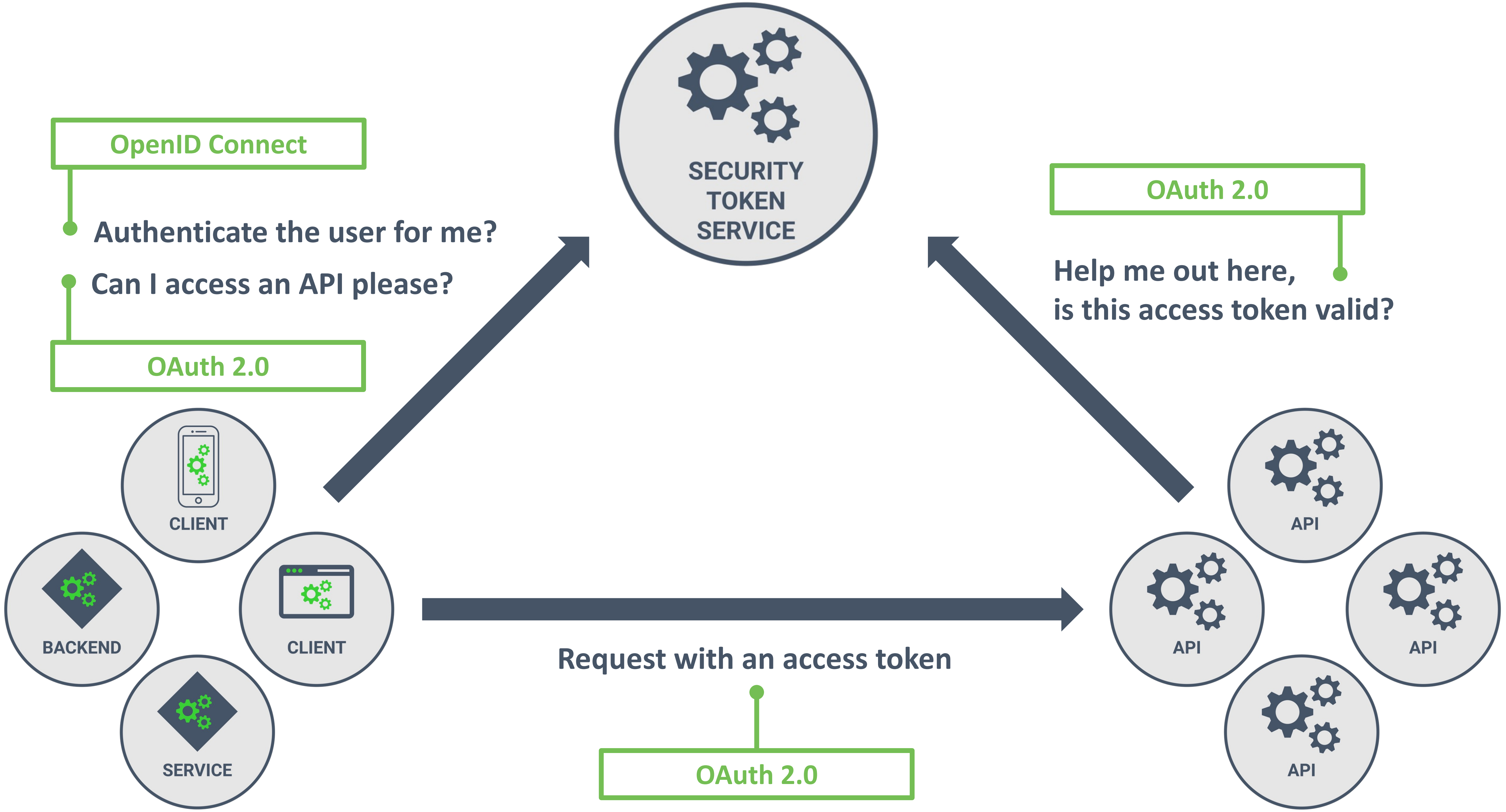
SSO

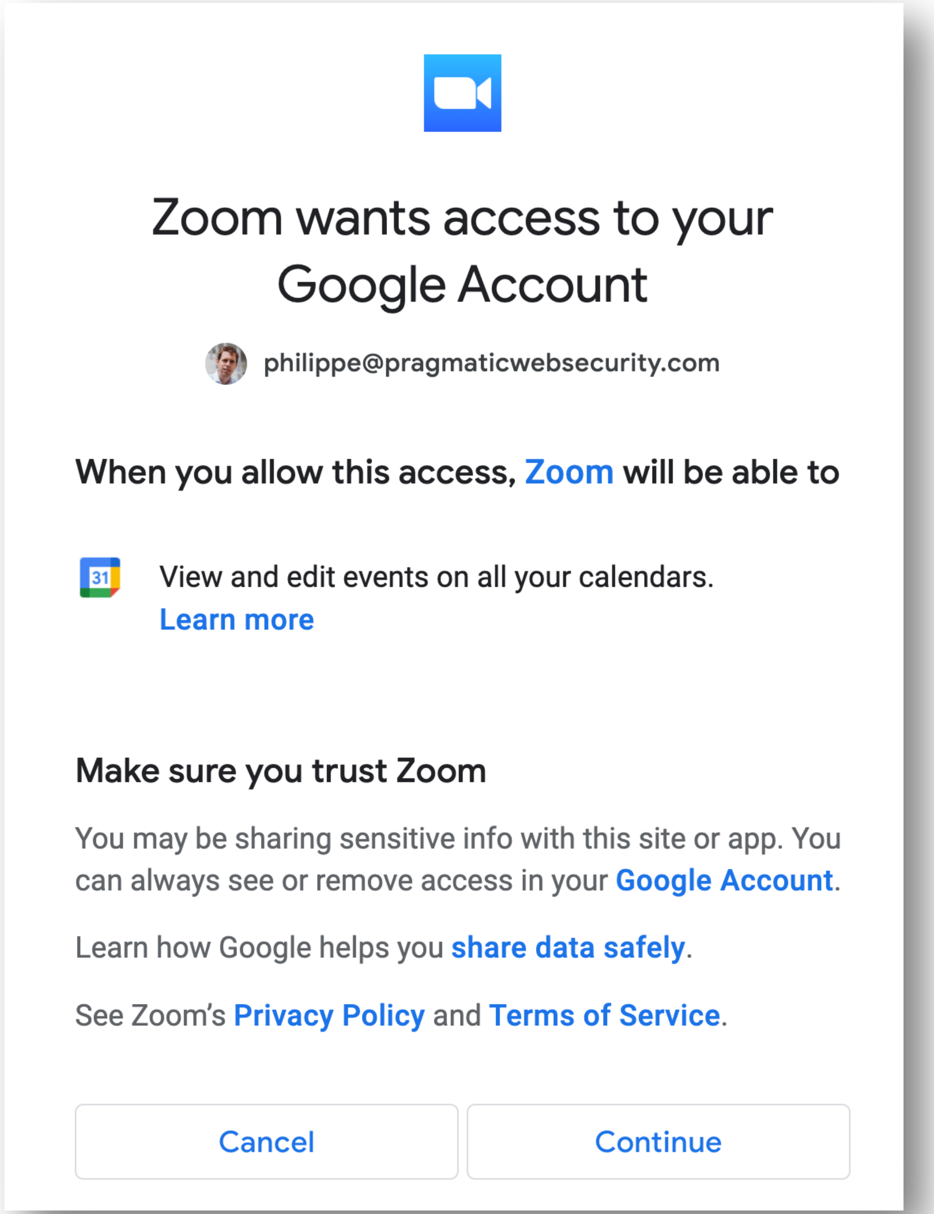
Apple

Google

Facebook

OpenID Connect is an authentication protocol, supporting SSO and federation


SECURITY TOKEN SERVICE
OpenID Connect
Authenticate the user for me?
Can I access an API please?
OAuth 2.0
OAuth 2.0
Help me out here,
is this access token valid?
CLIENT
BACKEND
CLIENT
SERVICE
Request with an access token
OAuth 2.0
API
API
API
API



Zoom wants access to your Google Account

philippe@pragmaticwebsecurity.com

When you allow this access, Zoom will be able to

View and edit events on all your calendars.
Learn more

Make sure you trust Zoom

You may be sharing sensitive info with this site or app. You can always see or remove access in your Google Account.

Learn how Google helps you share data safely.

See Zoom's Privacy Policy and Terms of Service.

OAuth 2.0 offers an authorization framework to support complex applications

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

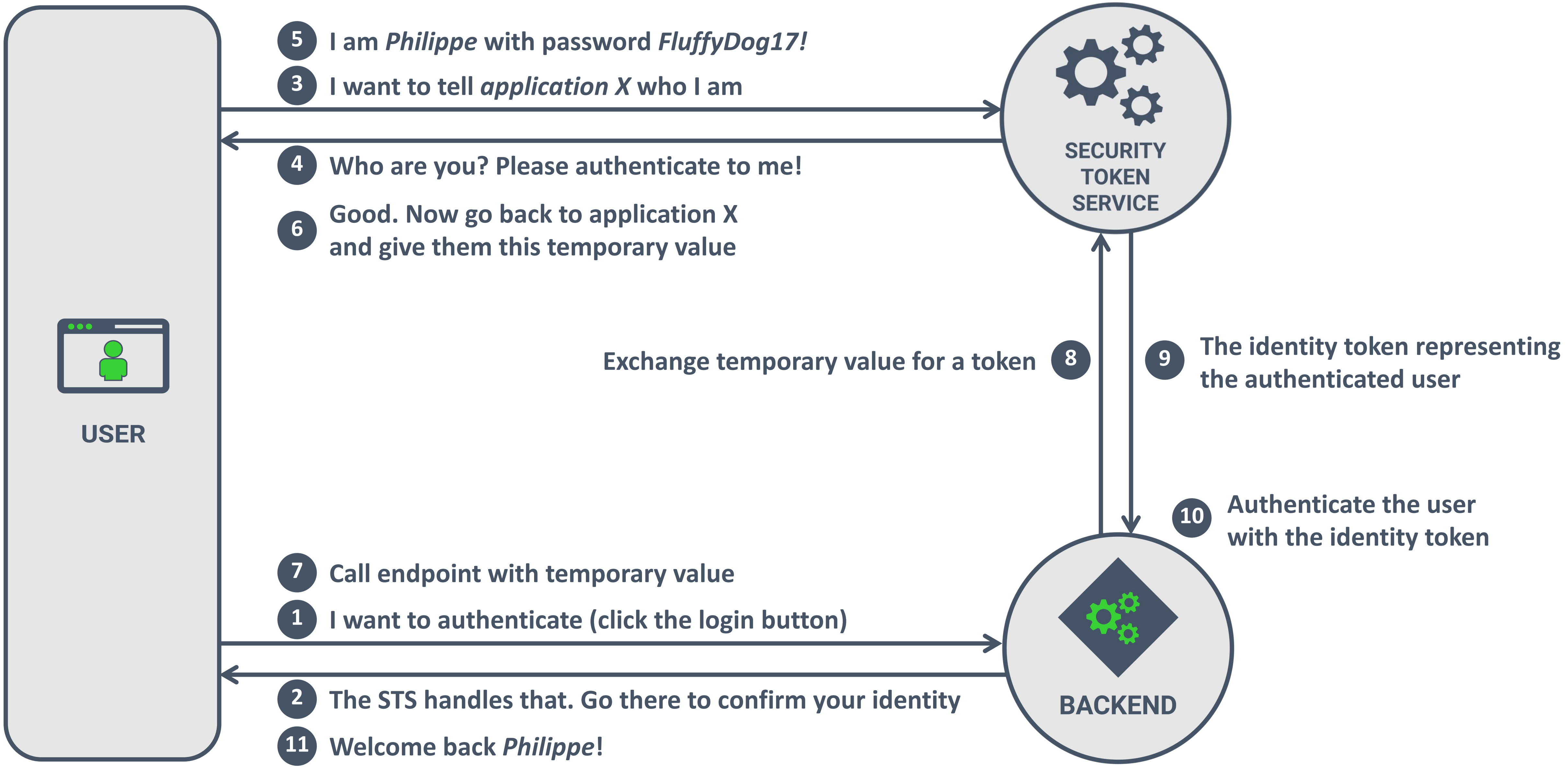
I help developers with security

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

https://pdr.online

Using OpenID Connect for authentication

The concept of OpenID Connect



5 I am *Philippe* with password *FluffyDog17!*
3 I want to tell *application X* who I am
4 Who are you? Please authenticate to me!
6 Good. Now go back to application X and give them this temporary value
SECURITY TOKEN SERVICE
USER
Exchange temporary value for a token 8
9 The identity token representing the authenticated user
10 Authenticate the user with the identity token
7 Call endpoint with temporary value
1 I want to authenticate (click the login button)
2 The STS handles that. Go there to confirm your identity
11 Welcome back *Philippe*!
BACKEND

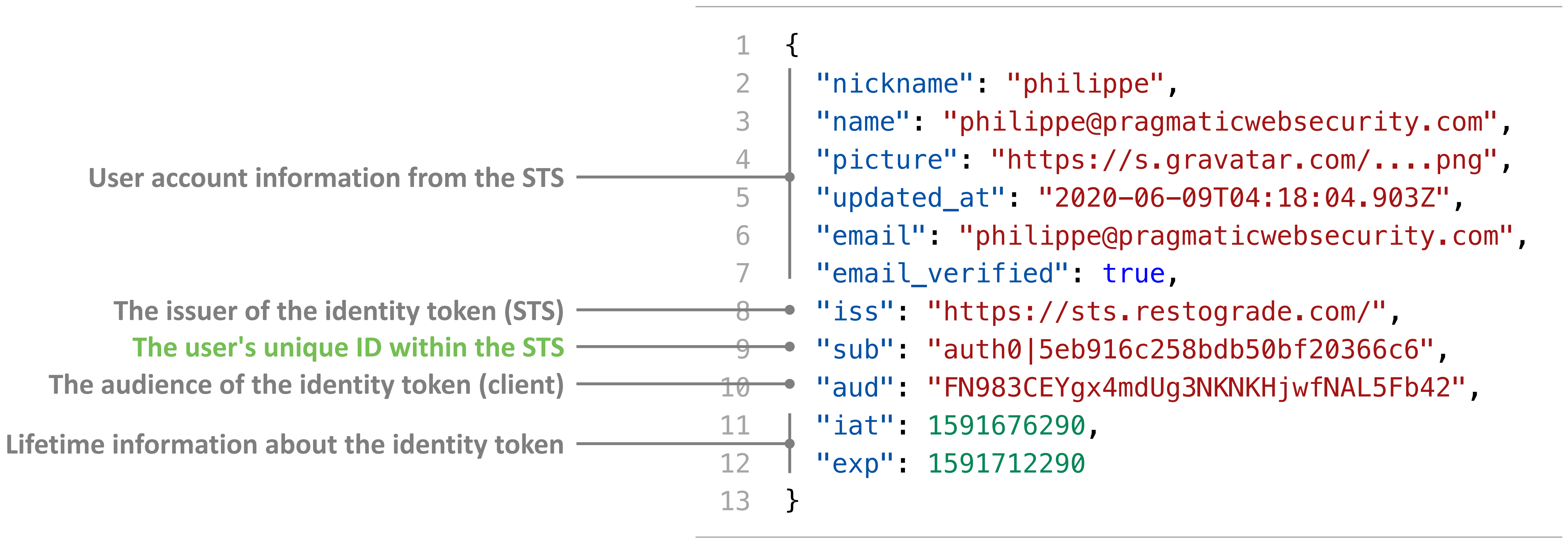
The encoded identity token

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6Ik5U
VkJPVFUzTXpCQk9FVXdOemhCUTBWR01rUTBRVVU1UVRZeFFV
VXlPVU5FUVVVeE5qRXlNdyJ9.eyJuaWNrbmFtZSI6InBoaWx
pcHBlIiwibmFtZSI6InBoaWxpcHBlQHByYWdtYXRpY3dlYnN
lY3VyaXR5LmNvbSIsInBpY3R1cmUiOiJodHRwczovL3MuZ3J
hdmF0YXIuY29tL2F2YXRhci9mNDBkNjRhNGIxNjc4OTUwODA
2MmU2NjRiZTZhZTU3NT9zPTQ4MCZyPXBnJmQ9aHR0cHMlM0E
lMkYlMkZjZG4uYXV0aDAuY29tJTJGYXZhdGFycyUyRnBoLnB
uZyIsInVwZGF0ZWRfYXQiOiIyMDIwLTA2LTA5VDA0OjE4OjA
0LjkwM1oiLCJlbWFpbCI6InBoaWxpcHBlQHByYWdtYXRpY3d
lYnNlY3VyaXR5LmNvbSIsImVtYWlsX3ZlcmlmaWVkIjp0cnV
lLCJpc3MiOiJodHRwczovL3N0cy5yZXN0b2dyYWRlLmNvbS8
iLCJzdWIiOiJhdXRoMHw1ZWI5MTZjMjU4YmRiNTBiZjIwMzY
2YzYiLCJhdWQiOiJGTjk4M0NFWWd4NG1kVWczTktOS0hqd2Z
OQUw1RmI0MiIsImlhdCI6MTU5MTY3NjI5MCwiZXhwIjoxNTk
xNzEyMjkwfQ.m60Br25jY8MOwIpCAjv3tRYF7IMR11ydzaP1
m6qJwsX74Sr5WUh49IK3iwaK72U6r2KXAp3_Oys9aabdoSc6
EkiYo7sho2W_fbLrUz8ocHFcTdHemuM0zoDQ6lVgobDNiwtl
eht8iNnIf9ghlRa-
TBtuL0TIRxkSHsCuJHKlWEG7zVHwll1q34XcLtkq4mnjWKlM
P5dNZoqIB_0Gek-EG05nUuoYwK7IqaZIGFLgc4EaK0fel-
MIqqDAwiD3etAkILSu7Phejk6zHwuEQlt3YzlbP5ZHNPK5hn
Sph80BPL7VMdDUWhjMdl1eW21cRq5CQNIKAJDbVLDdWqemO9
Kp_A
```

The decoded JWT payload

```json
{
  "nickname": "philippe",
  "name": "philippe@pragmaticwebsecurity.com",
  "picture": "https://s.gravatar.com/....png",
  "updated_at": "2020-06-09T04:18:04.903Z",
  "email": "philippe@pragmaticwebsecurity.com",
  "email_verified": true,
  "iss": "https://sts.restograde.com/",
  "sub": "auth0|5eb916c258bdb50bf20366c6",
  "aud": "FN983CEYgx4mdUg3NKNKHjwfNAL5Fb42",
  "iat": 1591676290,
  "exp": 1591712290
}
```

The decoded JWT payload


User account information from the STS
The issuer of the identity token (STS)
The user's unique ID within the STS
The audience of the identity token (client)
Lifetime information about the identity token

```
{
  "nickname": "philippe",
  "name": "philippe@pragmaticwebsecurity.com",
  "picture": "https://s.gravatar.com/....png",
  "updated_at": "2020-06-09T04:18:04.903Z",
  "email": "philippe@pragmaticwebsecurity.com",
  "email_verified": true,
  "iss": "https://sts.restograde.com/",
  "sub": "auth0|5eb916c258bdb50bf20366c6",
  "aud": "FN983CEYgx4mdUg3NKNKHjwfNAL5Fb42",
  "iat": 1591676290,
  "exp": 1591712290
}
```

The backend's internal user database

ID	Name	Sub
1	alice	auth0|8c34361ea1c8bff697e3a81e
2	philippe	auth0|5eb916c258bdb50bf20366c6

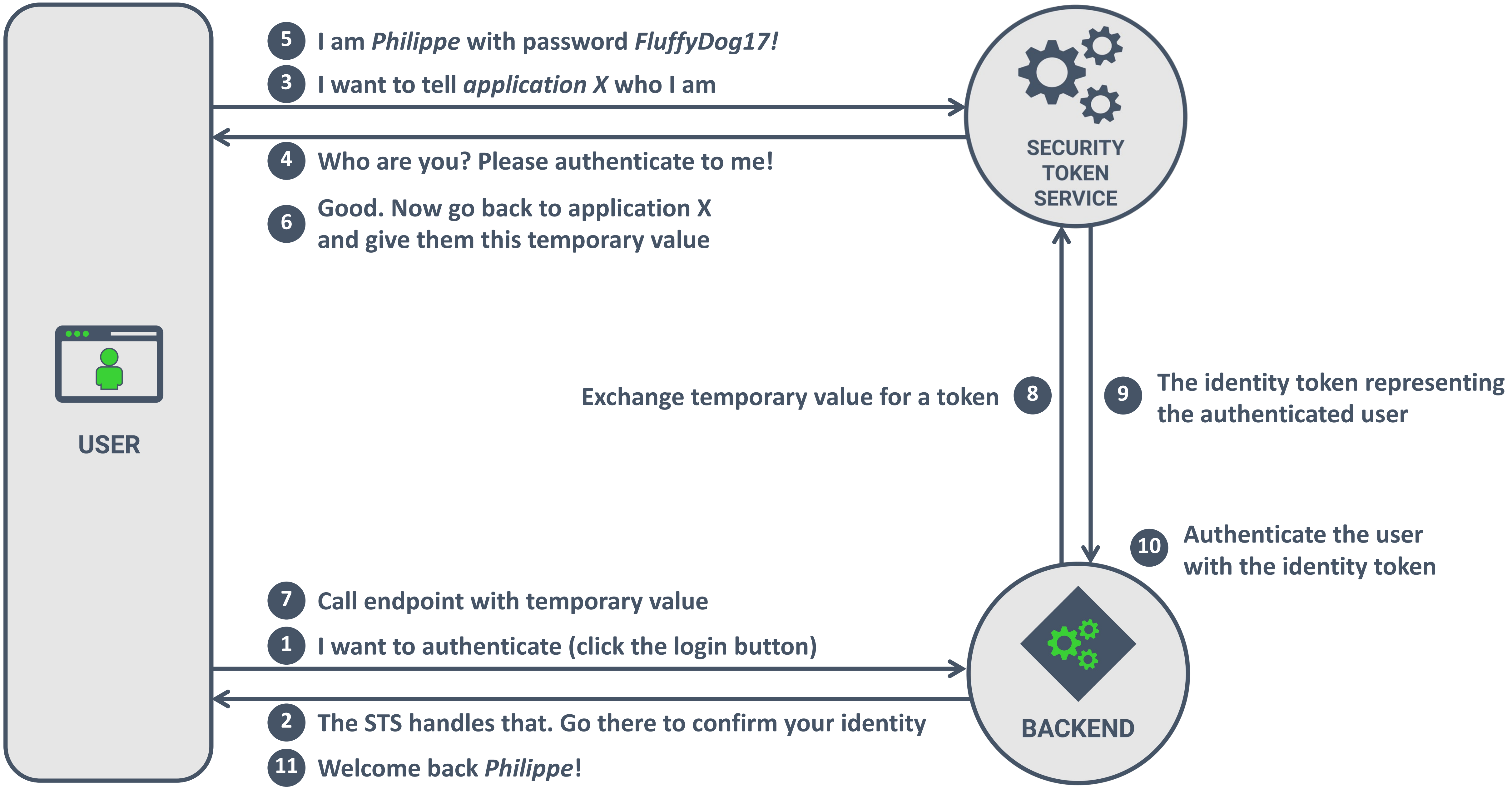
The session is populated with the information about the authenticated user

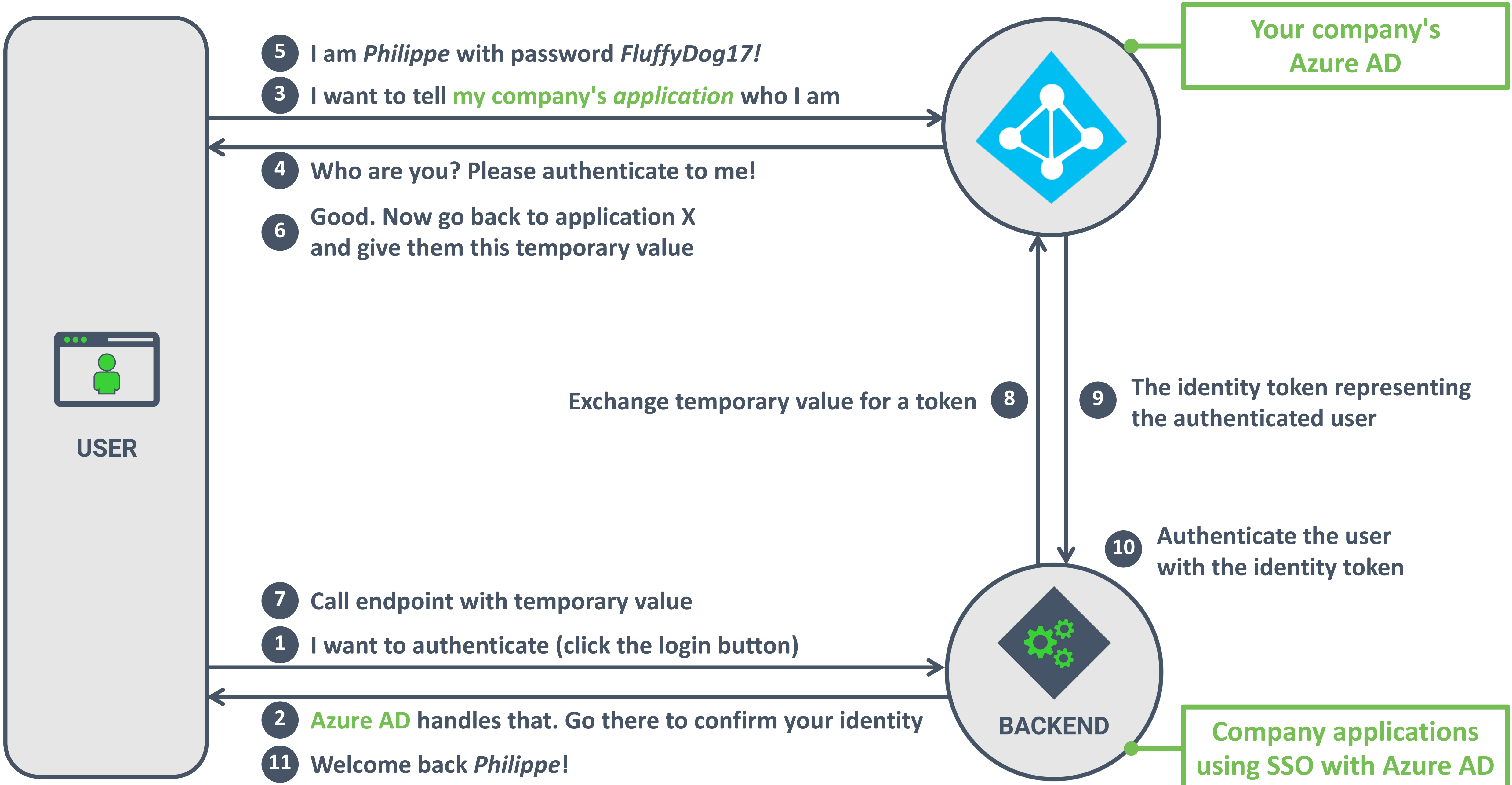
The *sub* value is used to find the authenticated user in the Virtual Foodie database

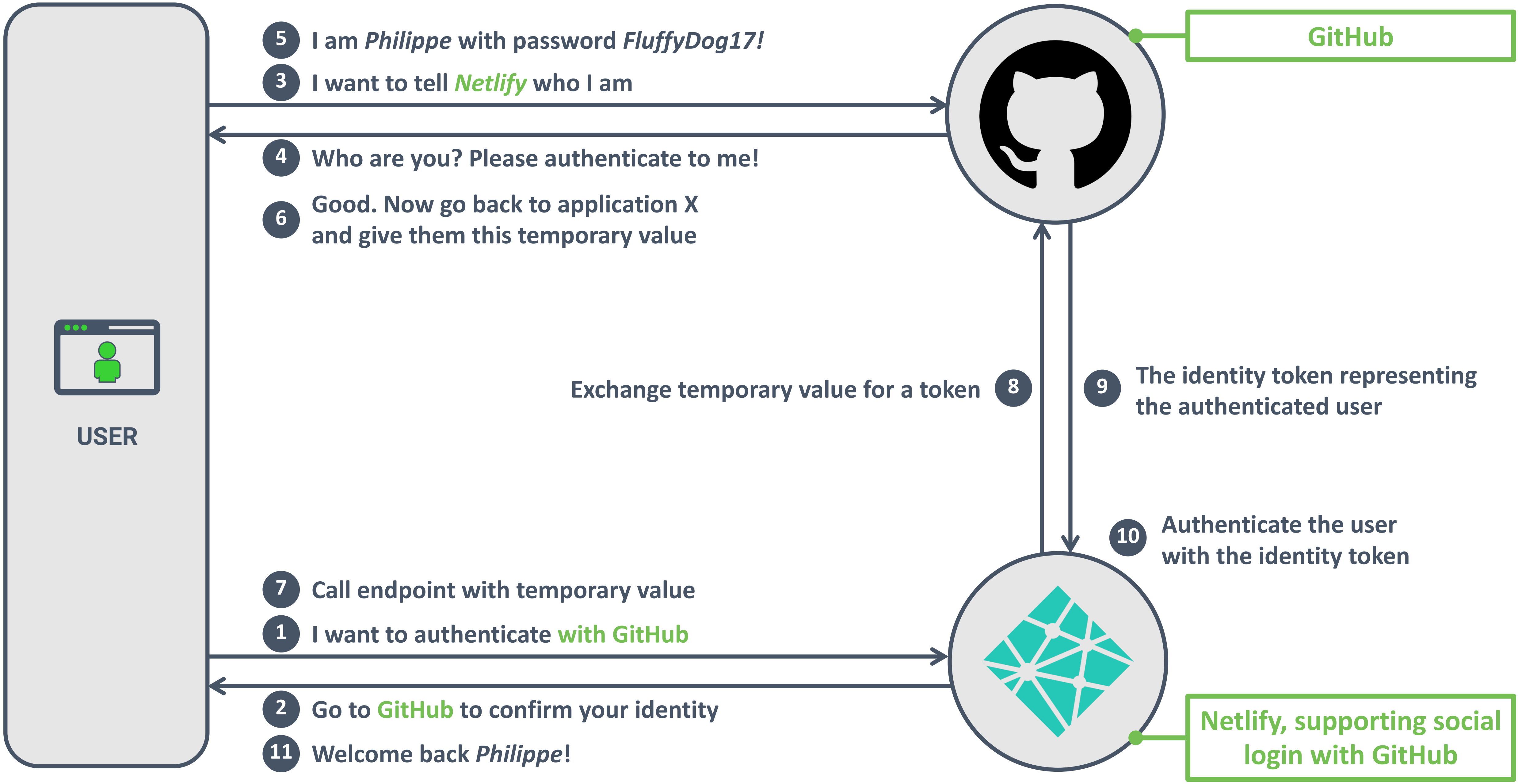
The identity token payload

```
{
  "nickname": "philippe",
  "name": "philippe@pragmaticwebsecurity.com",
  "picture": "https://s.gravatar.com/....png",
  "updated_at": "2020-06-09T04:18:04.903Z",
  "email": "philippe@pragmaticwebsecurity.com",
  "email_verified": true,
  "iss": "https://sts.restograde.com/",
  "sub": "auth0|5eb916c258bdb50bf20366c6",
  "aud": "FN983CEYgx4mdUg3NKNKHjwfNAL5Fb42",
  "iat": 1591676410,
  "exp": 1591712410
}
```

The *sub* claim is guaranteed to be unique and immutable

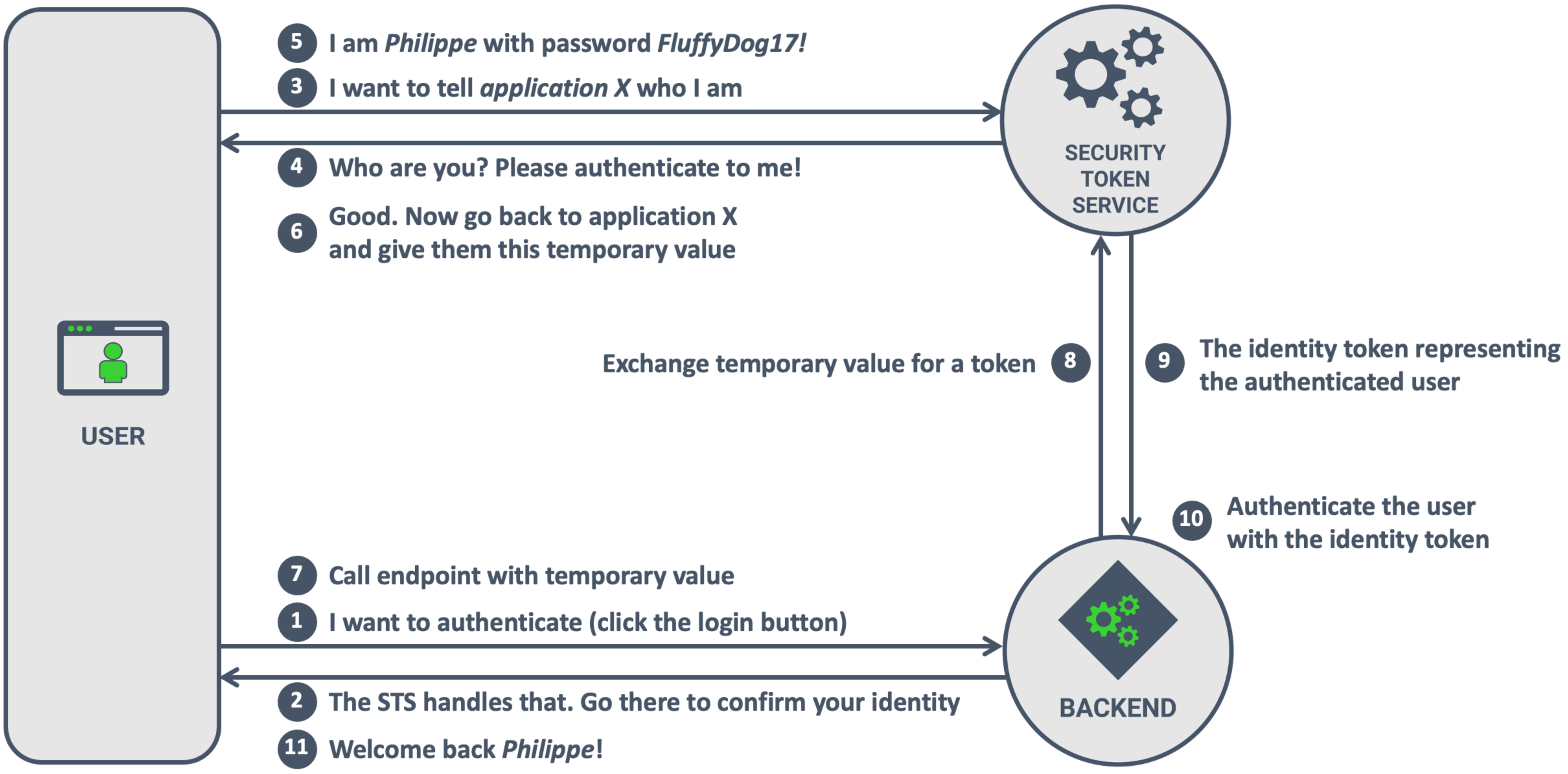

USER
SECURITY TOKEN SERVICE
BACKEND
1 I want to authenticate (click the login button)
2 The STS handles that. Go there to confirm your identity
3 I want to tell *application X* who I am
4 Who are you? Please authenticate to me!
5 I am *Philippe* with password *FluffyDog17!*
6 Good. Now go back to application X and give them this temporary value
7 Call endpoint with temporary value
8 Exchange temporary value for a token
9 The identity token representing the authenticated user
10 Authenticate the user with the identity token
11 Welcome back *Philippe*!


5 I am *Philippe* with password *FluffyDog17!*
3 I want to tell my company's *application* who I am
4 Who are you? Please authenticate to me!
6 Good. Now go back to application X and give them this temporary value
Your company's Azure AD
USER
Exchange temporary value for a token 8
9 The identity token representing the authenticated user
10 Authenticate the user with the identity token
7 Call endpoint with temporary value
1 I want to authenticate (click the login button)
2 Azure AD handles that. Go there to confirm your identity
11 Welcome back *Philippe*!
BACKEND
Company applications using SSO with Azure AD


5 I am *Philippe* with password *FluffyDog17!*
3 I want to tell *Netlify* who I am
4 Who are you? Please authenticate to me!
6 Good. Now go back to application X and give them this temporary value
GitHub
USER
Exchange temporary value for a token 8
9 The identity token representing the authenticated user
10 Authenticate the user with the identity token
7 Call endpoint with temporary value
1 I want to authenticate with GitHub
2 Go to GitHub to confirm your identity
11 Welcome back *Philippe*!
Netlify, supporting social login with GitHub

OpenID Connect has nothing to do with API access or authorization

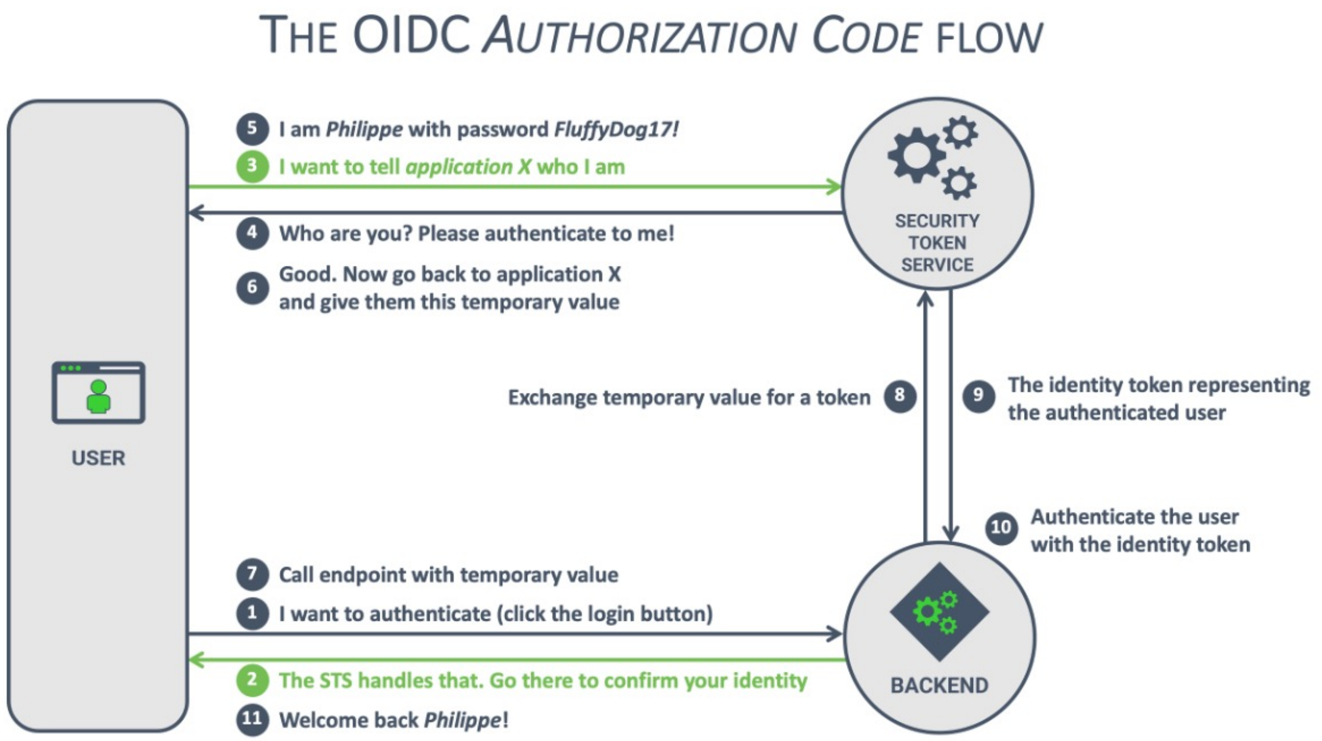
THE OIDC *AUTHORIZATION CODE* FLOW


5 I am *Philippe* with password *FluffyDog17!*
3 I want to tell *application X* who I am
4 Who are you? Please authenticate to me!
6 Good. Now go back to application X and give them this temporary value
SECURITY TOKEN SERVICE
USER
Exchange temporary value for a token 8
9 The identity token representing the authenticated user
10 Authenticate the user with the identity token
7 Call endpoint with temporary value
1 I want to authenticate (click the login button)
2 The STS handles that. Go there to confirm your identity
11 Welcome back *Philippe*!
BACKEND

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
  &nonce=66QK3qqWhxQD_L0ZAuqritZi5Sy6
```

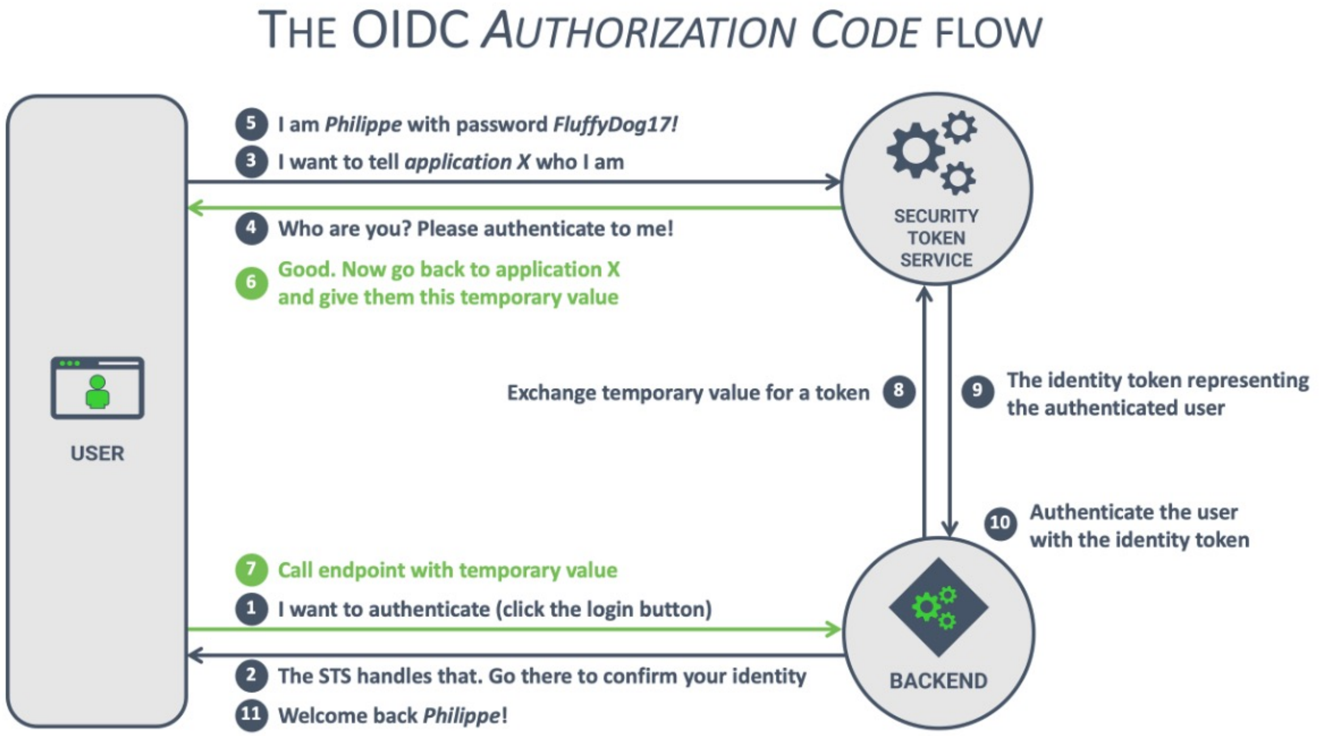
- `?response_type=code` — **Indicates the *authorization code flow***
- `&scope=openid profile email` — **We want an ID token with email/profile info**
- `&client_id=FN983CEYgx4mdUg3NKNKHjwfNAL5Fb42` — **The client requesting authentication**
- `&redirect_uri=https://restograde.com/callback` — **Where the STS should send the code**
- `&nonce=66QK3qqWhxQD_L0ZAuqritZi5Sy6` — **Security measure to preserve flow integrity**


THE OIDC *AUTHORIZATION CODE* FLOW
USER
SECURITY TOKEN SERVICE
BACKEND
5 I am *Philippe* with password *FluffyDog17!*
3 I want to tell *application X* who I am
4 Who are you? Please authenticate to me!
6 Good. Now go back to application X and give them this temporary value
Exchange temporary value for a token 8
9 The identity token representing the authenticated user
10 Authenticate the user with the identity token
7 Call endpoint with temporary value
1 I want to authenticate (click the login button)
2 The STS handles that. Go there to confirm your identity
11 Welcome back *Philippe*!

6 7 *The redirect back to the backend application*

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

The temporary authorization code


THE OIDC AUTHORIZATION CODE FLOW
USER
SECURITY TOKEN SERVICE
BACKEND
5 I am Philippe with password FluffyDog17!
3 I want to tell application X who I am
4 Who are you? Please authenticate to me!
6 Good. Now go back to application X and give them this temporary value
Exchange temporary value for a token 8
9 The identity token representing the authenticated user
10 Authenticate the user with the identity token
7 Call endpoint with temporary value
1 I want to authenticate (click the login button)
2 The STS handles that. Go there to confirm your identity
11 Welcome back Philippe!

8 *The request to exchange the authorization code*

```
POST /oauth/token
Host: sts.restograde.com

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

Indicates the code exchange request

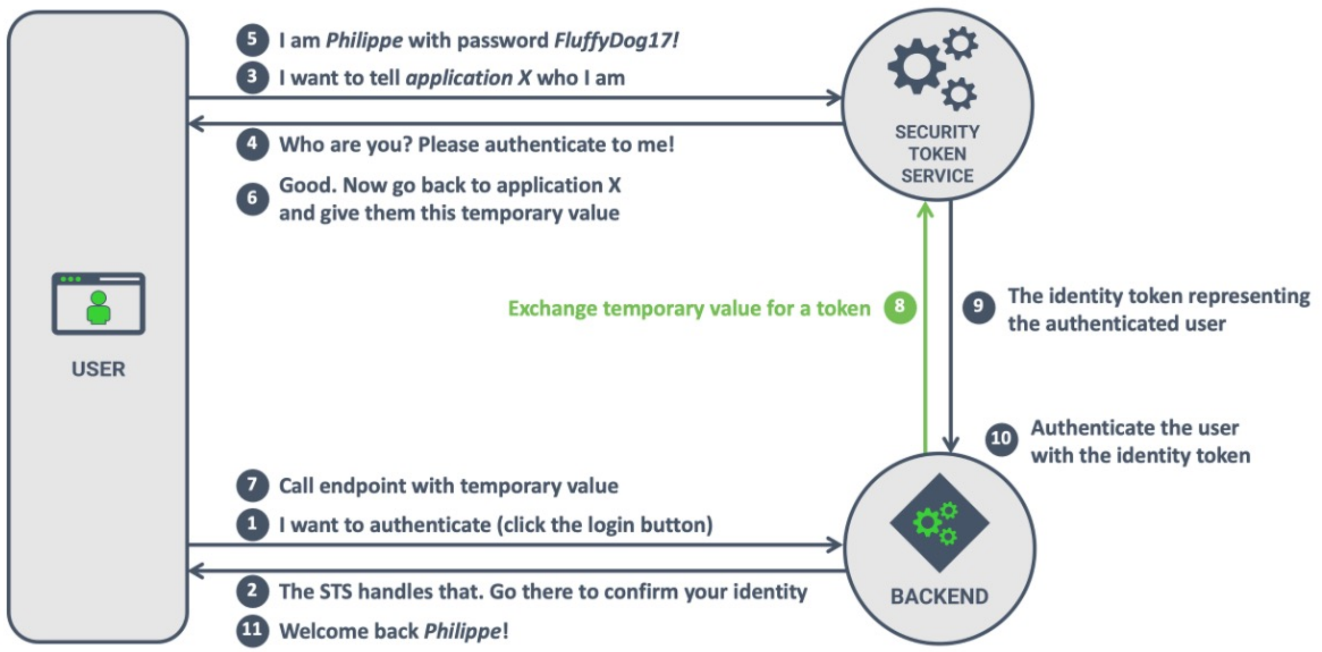
The client exchanging the code

The client needs to authenticate to the STS

The redirect URI used before

The code received in step 9

THE OIDC *AUTHORIZATION CODE* FLOW

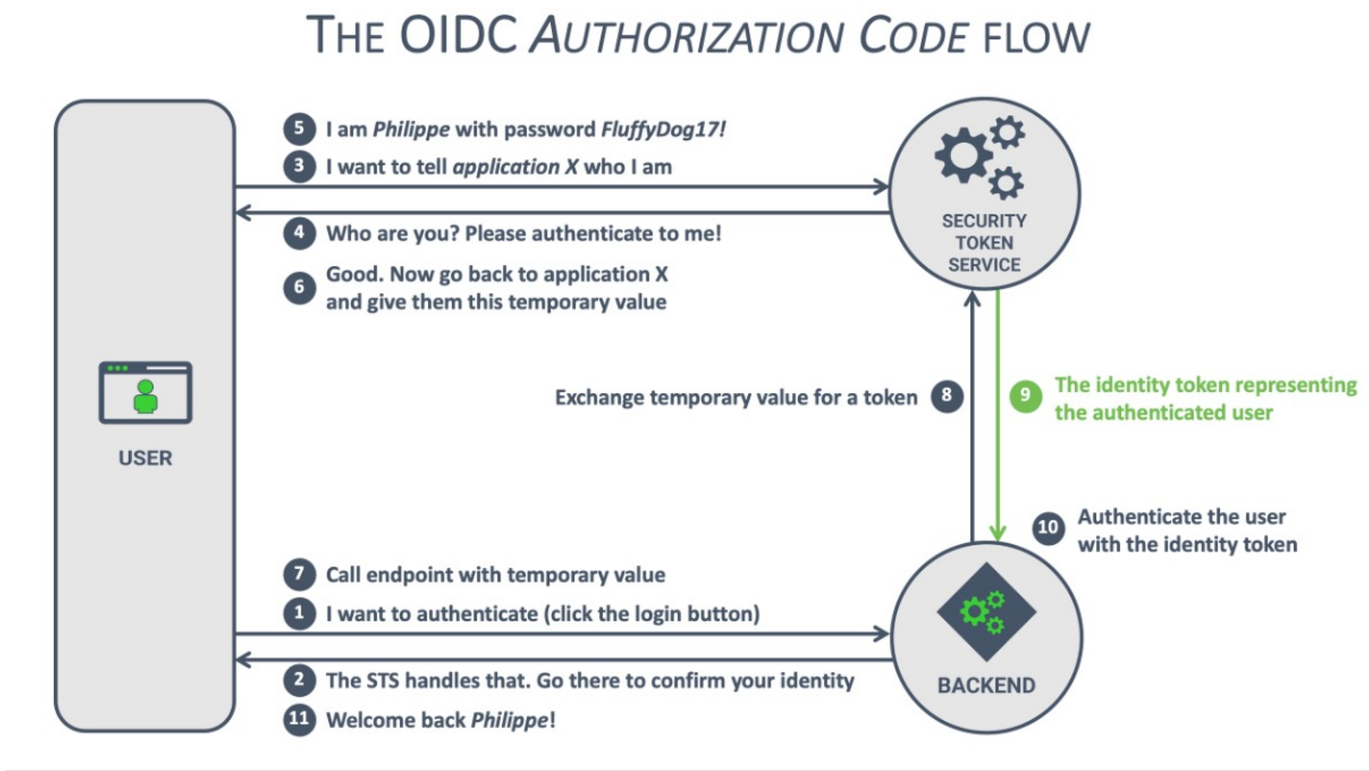


5 I am *Philippe* with password *FluffyDog17!*
3 I want to tell *application X* who I am
4 Who are you? Please authenticate to me!
6 Good. Now go back to application X and give them this temporary value
SECURITY TOKEN SERVICE
USER
Exchange temporary value for a token 8
9 The identity token representing the authenticated user
10 Authenticate the user with the identity token
7 Call endpoint with temporary value
1 I want to authenticate (click the login button)
2 The STS handles that. Go there to confirm your identity
11 Welcome back *Philippe*!
BACKEND

9 The response from the Security Token Service

```
{
  "id_token": "eyJhbGciO...du6TY9w",
}
```

The identity token representing the authenticated user


THE OIDC AUTHORIZATION CODE FLOW
USER
SECURITY TOKEN SERVICE
BACKEND
5 I am Philippe with password FluffyDog17!
3 I want to tell application X who I am
4 Who are you? Please authenticate to me!
6 Good. Now go back to application X and give them this temporary value
Exchange temporary value for a token 8
9 The identity token representing the authenticated user
10 Authenticate the user with the identity token
7 Call endpoint with temporary value
1 I want to authenticate (click the login button)
2 The STS handles that. Go there to confirm your identity
11 Welcome back Philippe!

User authentication with OpenID Connect

OIDC AND THE IDENTITY TOKEN

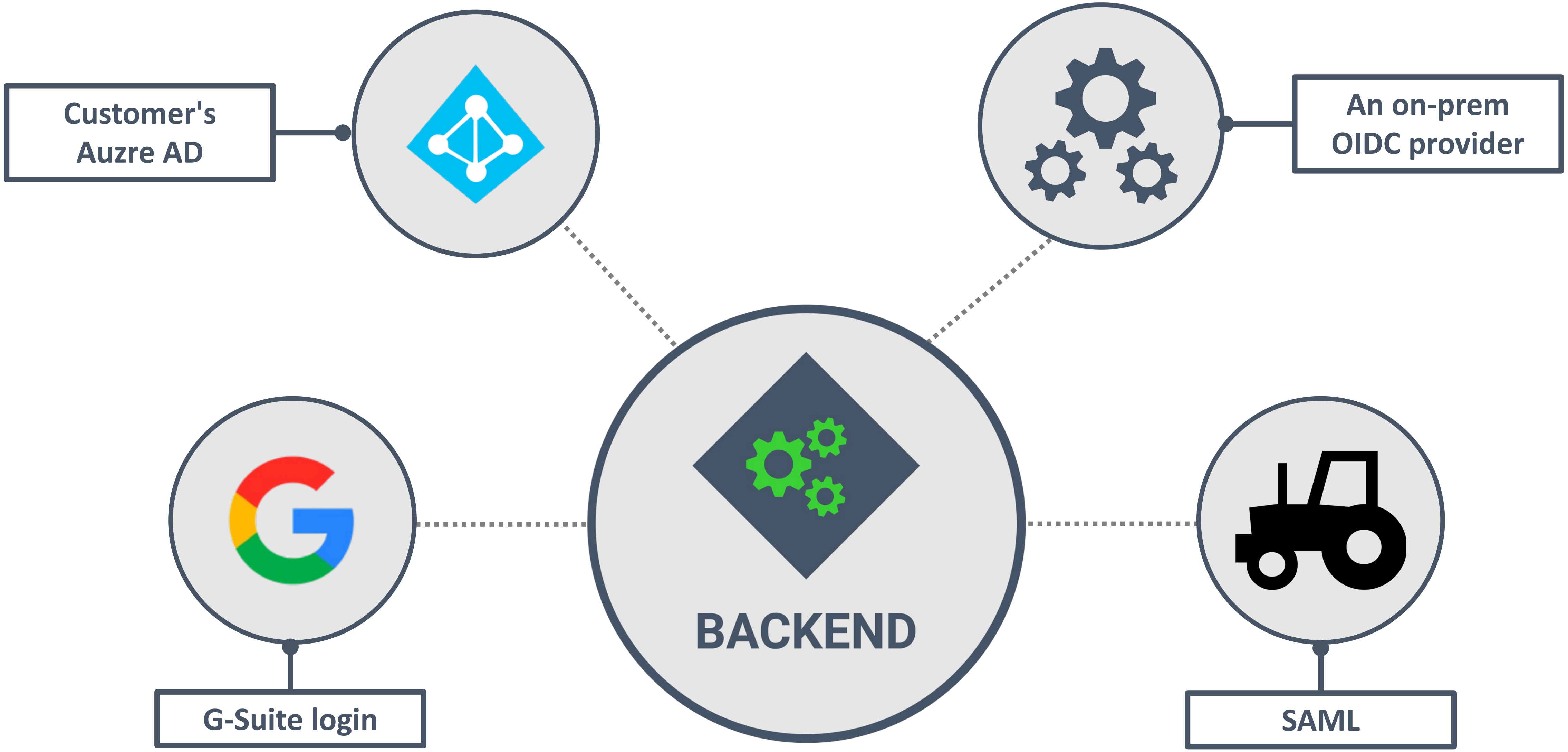
- OIDC allows the client (i.e., the backend app) to delegate authentication
  - OIDC relies on OAuth 2.0 to run a flow with the Security Token Service
  - The de facto standard to implement Single Sign-On in modern applications

- The client runs an OIDC flow to obtain an identity token
  - The client uses scopes to indicate the required information (*openid, profile, email, …)*
  - The identity token contains information about the user's authentication
  - The *iss* claim identifies the STS and the *sub* claim identifies the user

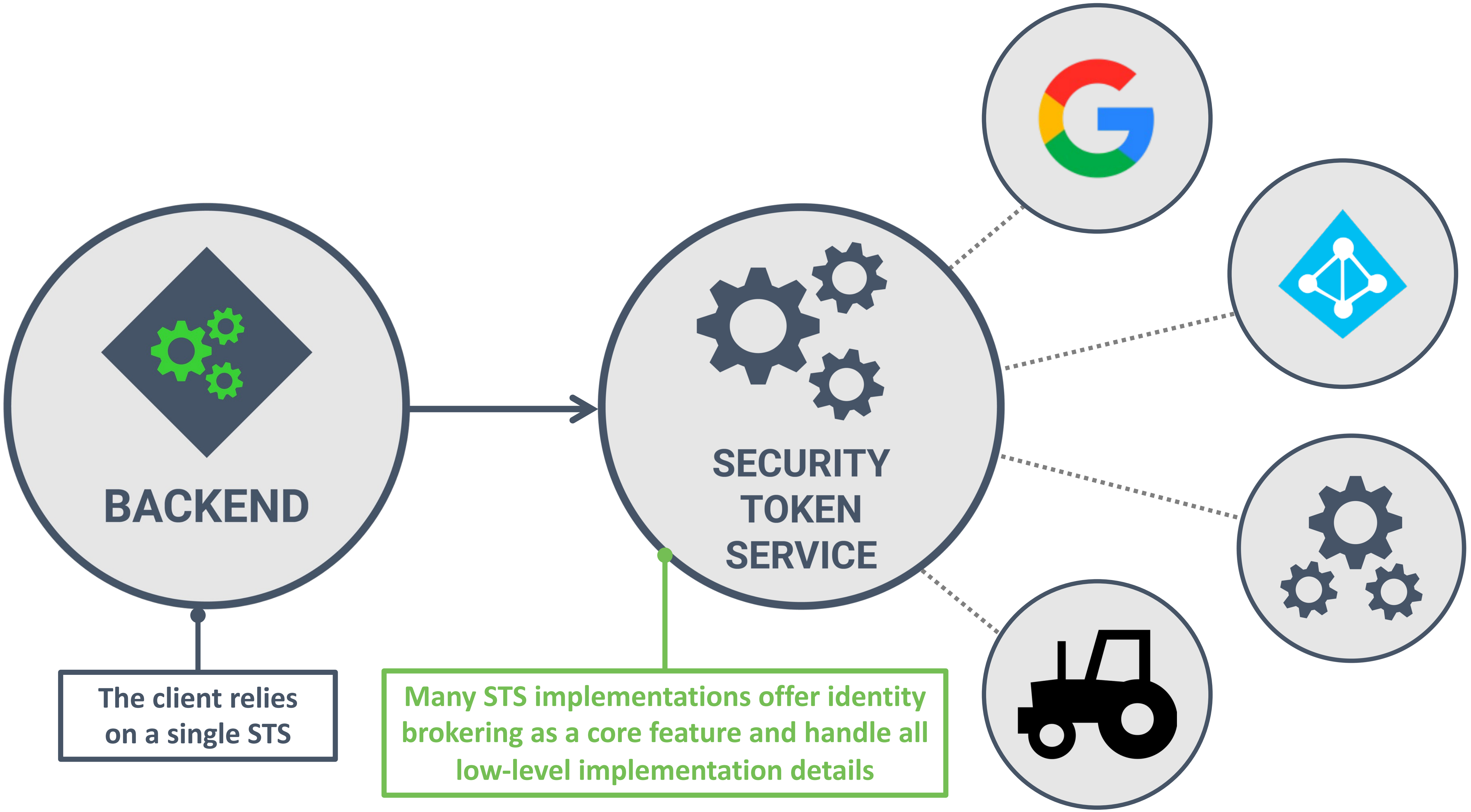
- Once the user is authenticated, the client maintains an authenticated session
  - The client is responsible for keeping track of the authenticated user
  - OIDC is only intended to support authenticating users, not

SaaS applications are often asked to support a customer's STS with OIDC or SAML

How do you support more than one STS?



Identity brokering with OIDC

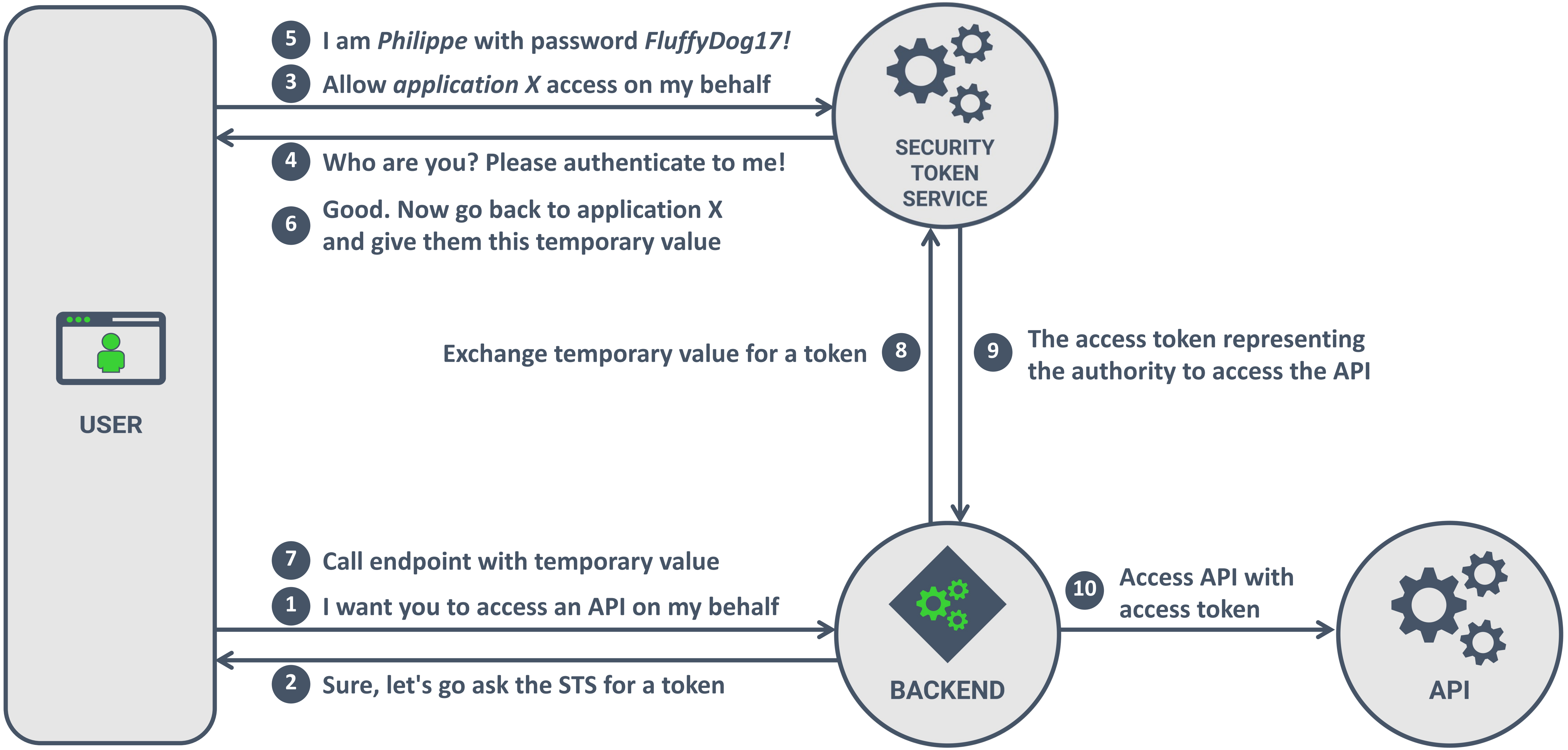


The client relies on a single STS

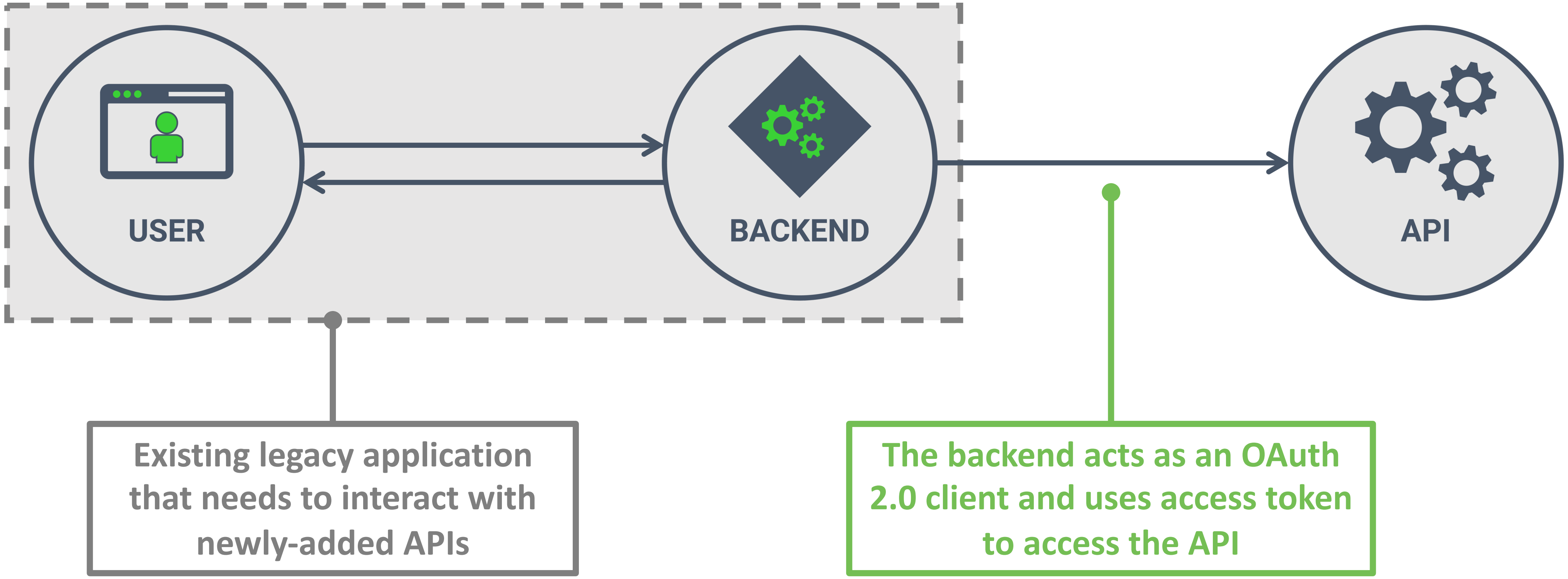
Many STS implementations offer identity brokering as a core feature and handle all low-level implementation details

Using OAuth 2.0 for API access

THE CONCEPT OF OAUTH 2.0


5 I am *Philippe* with password *FluffyDog17!*
3 Allow *application X* access on my behalf
4 Who are you? Please authenticate to me!
6 Good. Now go back to application X and give them this temporary value
SECURITY TOKEN SERVICE
USER
Exchange temporary value for a token 8
9 The access token representing the authority to access the API
7 Call endpoint with temporary value
1 I want you to access an API on my behalf
2 Sure, let's go ask the STS for a token
BACKEND
10 Access API with access token
API

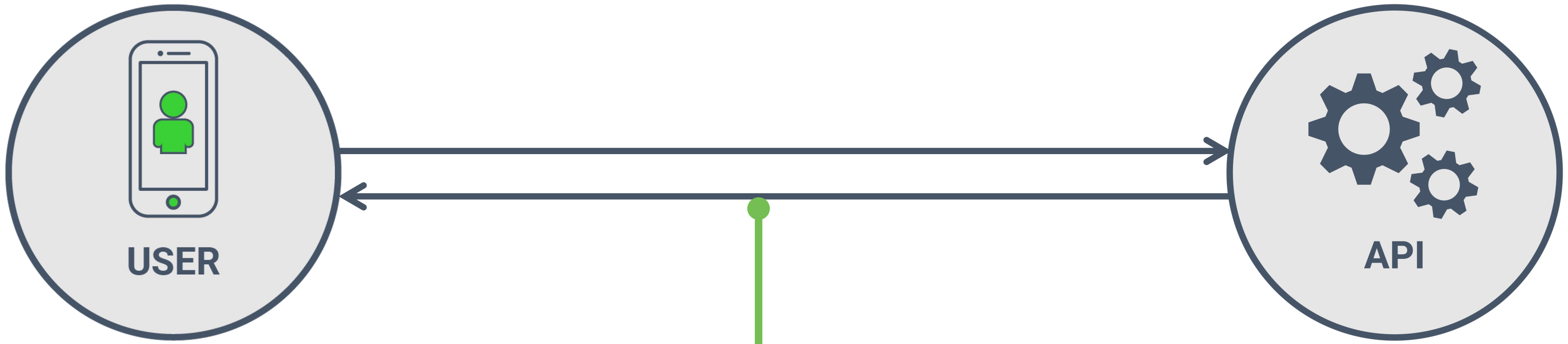
INTEGRATING OAUTH 2.0 IN EXISTING APPLICATIONS



Existing legacy application that needs to interact with newly-added APIs

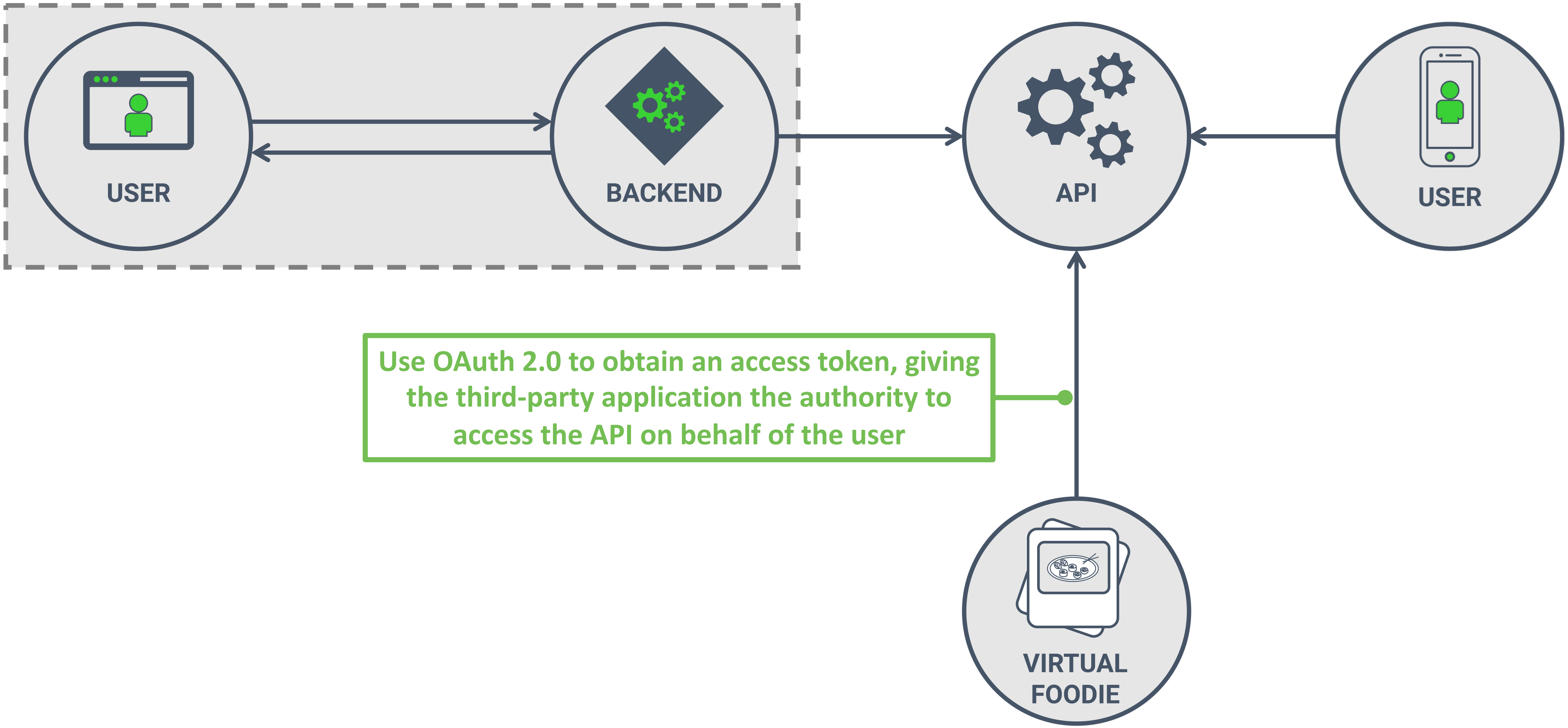
The backend acts as an OAuth 2.0 client and uses access token to access the API

Using OAuth 2.0 with mobile apps / web frontends



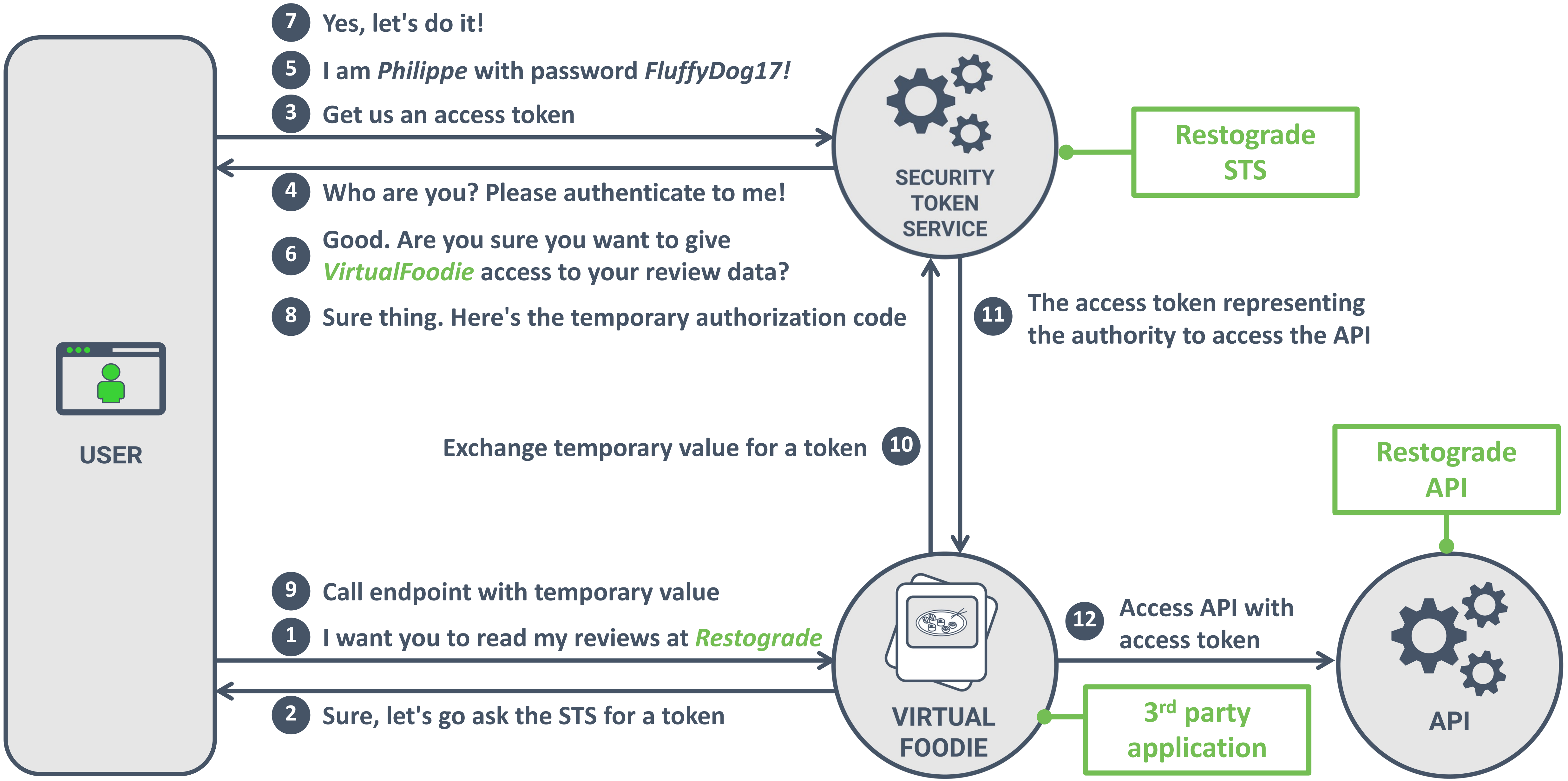
Mobile applications and web frontends can use OAuth 2.0 to contact the API directly

Allowing third-party access with OAuth 2.0



Use OAuth 2.0 to obtain an access token, giving the third-party application the authority to access the API on behalf of the user

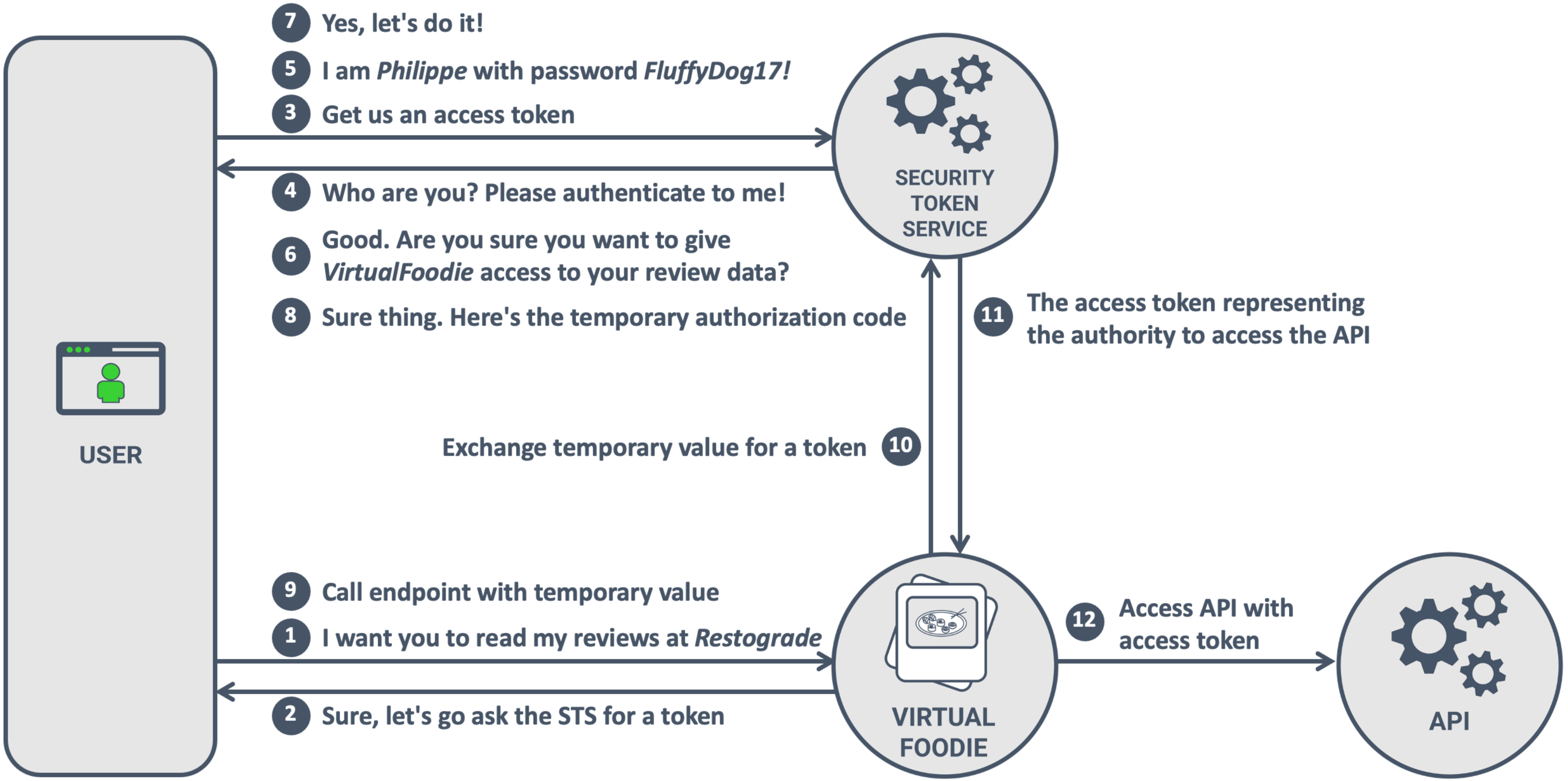
Allowing third-party access with OAuth 2.0


7 Yes, let's do it!
5 I am *Philippe* with password *FluffyDog17!*
3 Get us an access token
4 Who are you? Please authenticate to me!
6 Good. Are you sure you want to give *VirtualFoodie* access to your review data?
8 Sure thing. Here's the temporary authorization code
SECURITY TOKEN SERVICE
Restograde STS
11 The access token representing the authority to access the API
USER
Exchange temporary value for a token 10
Restograde API
9 Call endpoint with temporary value
1 I want you to read my reviews at *Restograde*
2 Sure, let's go ask the STS for a token
12 Access API with access token
VIRTUAL FOODIE
3rd party application
API

ACCESS TOKENS

- Access tokens represent the authority of a client to access specific resources
  - Typically, the access token represents the authority to act on behalf of the user
  - The user has the ability to delegate partial permissions to a client
- An access token associated with the user will contain user-specific details
  - The ***sub*** claim will hold the user's identifier, supporting authorization decisions by the API
  - Additional claims can contain further information about the user
- Access tokens should only be used for their specific purpose
  - They are issued by the STS and used by the client
  - They are consumed by the API

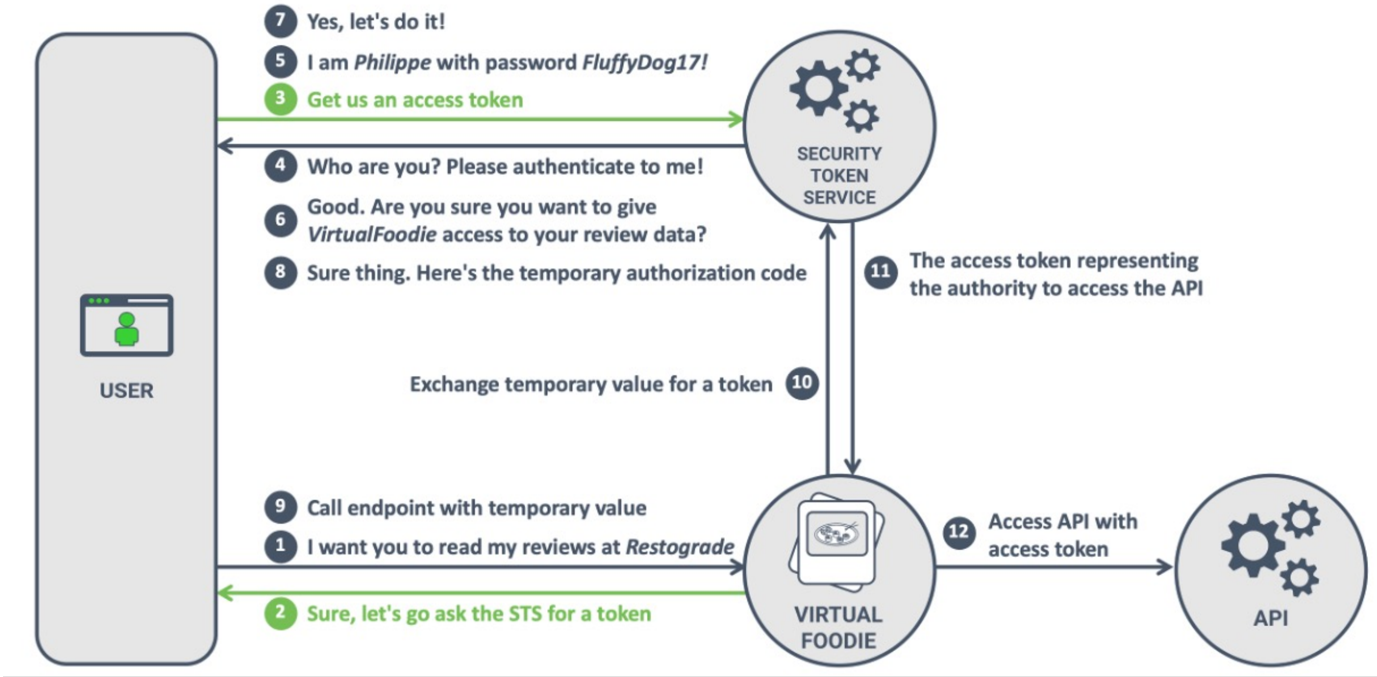
THE OAUTH 2.0 *AUTHORIZATION CODE* FLOW


7 Yes, let's do it!
5 I am *Philippe* with password *FluffyDog17!*
3 Get us an access token
4 Who are you? Please authenticate to me!
6 Good. Are you sure you want to give *VirtualFoodie* access to your review data?
8 Sure thing. Here's the temporary authorization code
SECURITY TOKEN SERVICE
11 The access token representing the authority to access the API
USER
Exchange temporary value for a token 10
9 Call endpoint with temporary value
1 I want you to read my reviews at *Restograde*
2 Sure, let's go ask the STS for a token
VIRTUAL FOODIE
12 Access API with access token
API

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

```
https://sts.restograde.com/authorize
  ?response_type=code                         Indicates the authorization code flow
  &scope=reviews:read                         We want read permissions
  &client_id=FN983CEYgx4mdUg3NKNKHjwfNAL5Fb42 The client requesting authentication
  &redirect_uri=https://restograde.com/callback     Where the STS should send the code
  &… PKCE parameters omitted for brevity …    Security measure to preserve flow integrity
```

THE OAUTH 2.0 *AUTHORIZATION CODE* FLOW

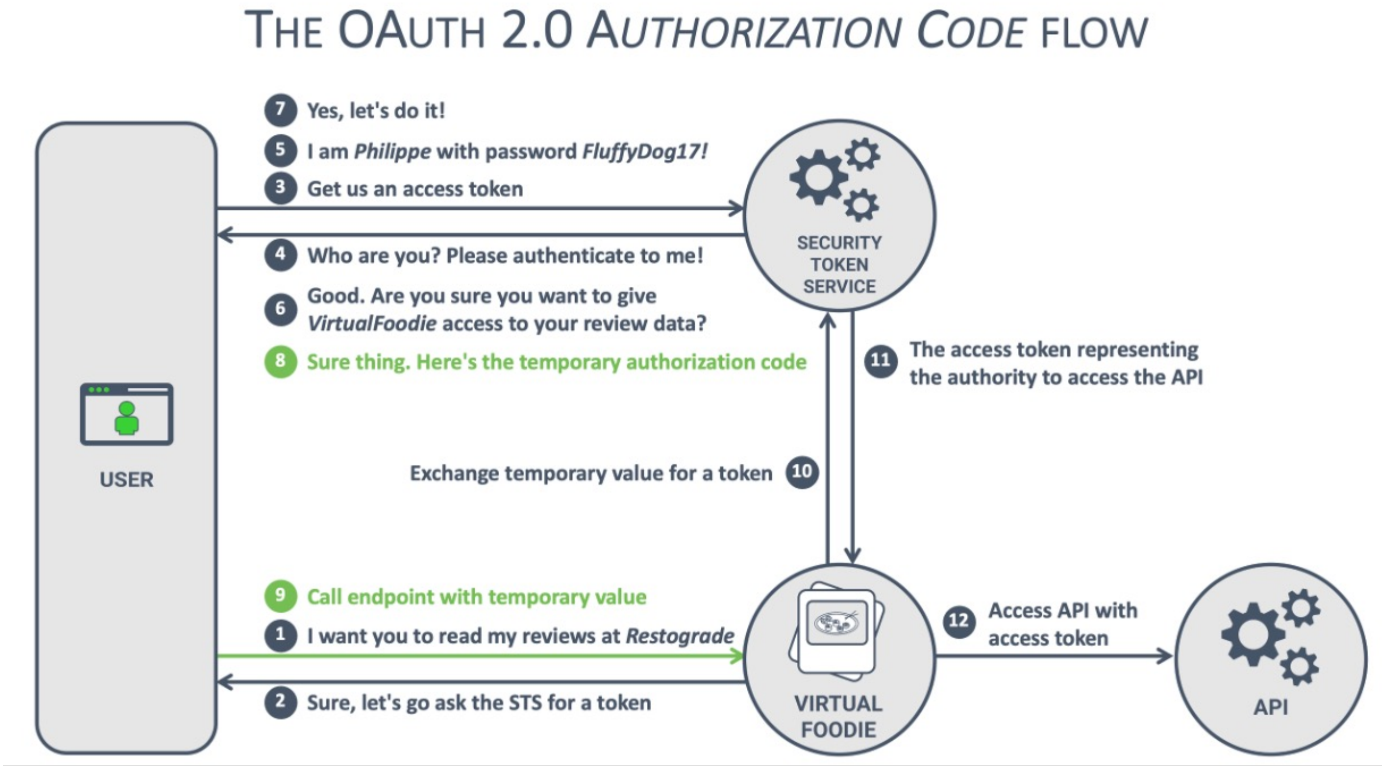


7 Yes, let's do it!
5 I am *Philippe* with password *FluffyDog17!*
3 Get us an access token
4 Who are you? Please authenticate to me!
6 Good. Are you sure you want to give *VirtualFoodie* access to your review data?
8 Sure thing. Here's the temporary authorization code
SECURITY TOKEN SERVICE
11 The access token representing the authority to access the API
USER
Exchange temporary value for a token 10
9 Call endpoint with temporary value
1 I want you to read my reviews at *Restograde*
2 Sure, let's go ask the STS for a token
VIRTUAL FOODIE
12 Access API with access token
API

6 7 *The redirect back to the backend application*

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

The temporary authorization code


THE OAUTH 2.0 AUTHORIZATION CODE FLOW
7 Yes, let's do it!
5 I am Philippe with password FluffyDog17!
3 Get us an access token
4 Who are you? Please authenticate to me!
6 Good. Are you sure you want to give VirtualFoodie access to your review data?
8 Sure thing. Here's the temporary authorization code
SECURITY TOKEN SERVICE
11 The access token representing the authority to access the API
Exchange temporary value for a token 10
USER
9 Call endpoint with temporary value
1 I want you to read my reviews at Restograde
2 Sure, let's go ask the STS for a token
VIRTUAL FOODIE
12 Access API with access token
API

8 *The request to exchange the authorization code*

```
POST /oauth/token
Host: sts.restograde.com

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

Indicates the code exchange request

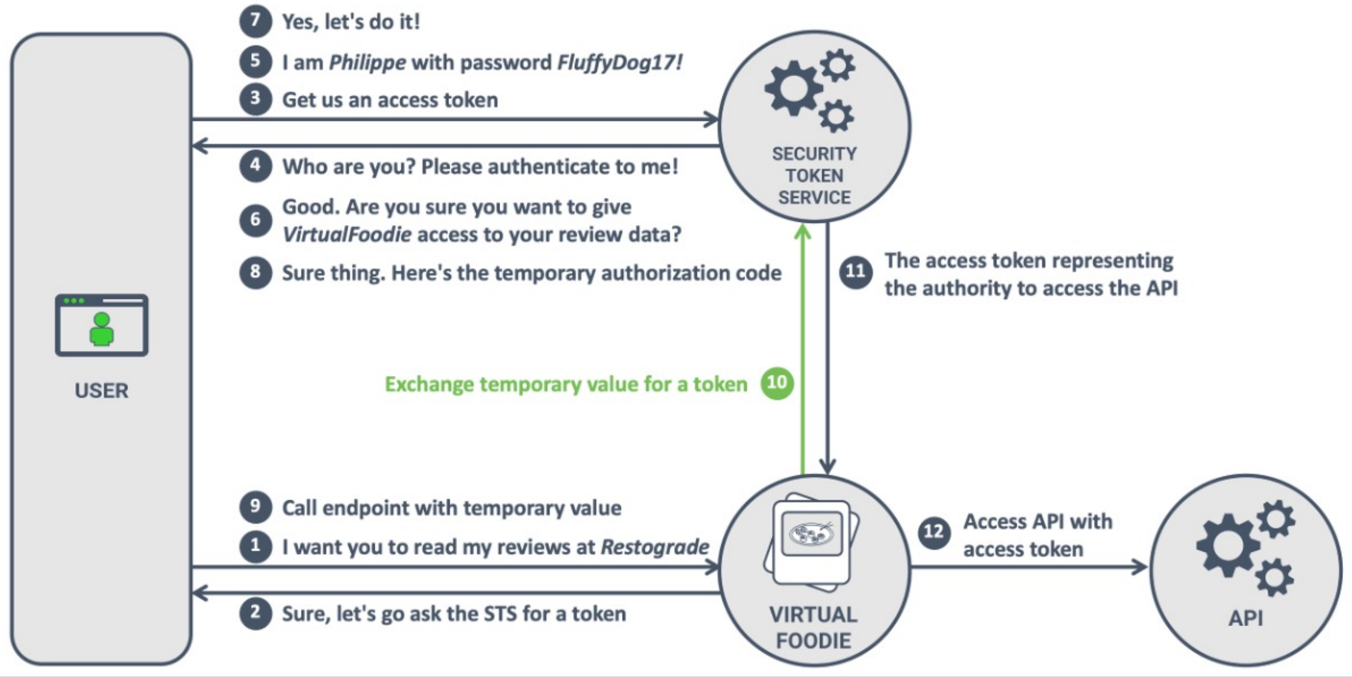
The client exchanging the code

The client needs to authenticate to the STS

The redirect URI used before

The code received in step 9

THE OAUTH 2.0 *AUTHORIZATION CODE* FLOW


7 Yes, let's do it!
5 I am *Philippe* with password *FluffyDog17!*
3 Get us an access token
4 Who are you? Please authenticate to me!
6 Good. Are you sure you want to give *VirtualFoodie* access to your review data?
8 Sure thing. Here's the temporary authorization code
SECURITY TOKEN SERVICE
11 The access token representing the authority to access the API
USER
Exchange temporary value for a token 10
9 Call endpoint with temporary value
1 I want you to read my reviews at *Restograde*
2 Sure, let's go ask the STS for a token
VIRTUAL FOODIE
12 Access API with access token
API

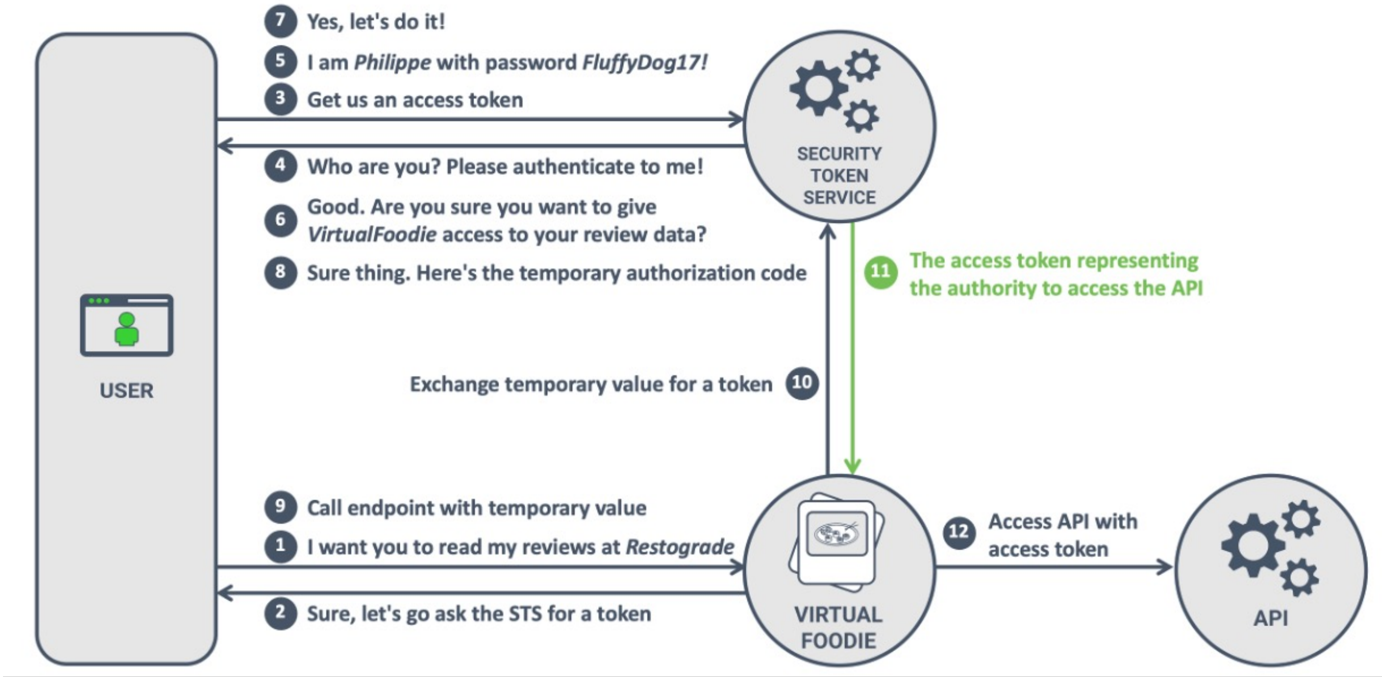
9 *The response from the Security Token Service*

```
{
  "access_token": "eyJhbGciO...du6TY9w",
  "token_type": "Bearer",
  "expires_in": 3600,
}
```

The access token with the authority to call the API

The expiration time of the access token

THE OAUTH 2.0 *AUTHORIZATION CODE* FLOW

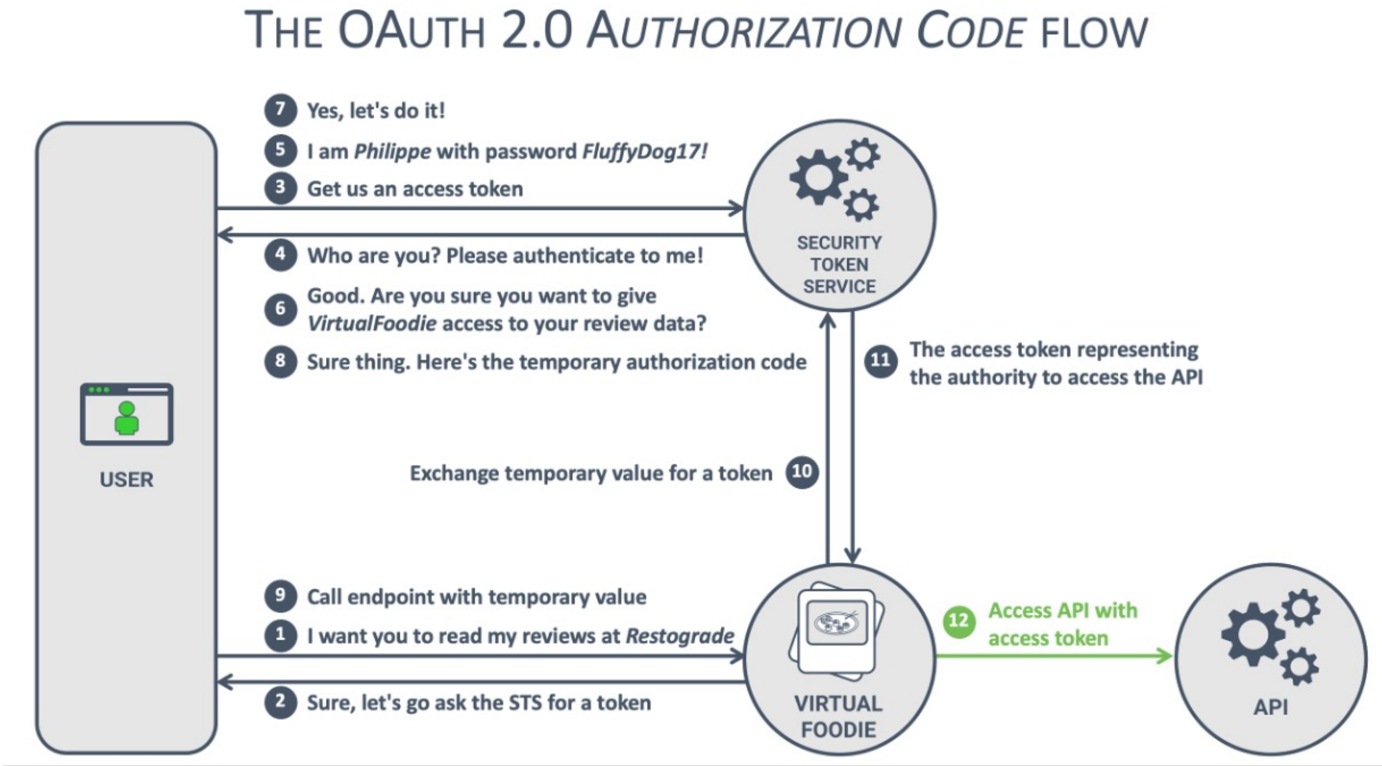


7 Yes, let's do it!
5 I am *Philippe* with password *FluffyDog17!*
3 Get us an access token
SECURITY TOKEN SERVICE
4 Who are you? Please authenticate to me!
6 Good. Are you sure you want to give *VirtualFoodie* access to your review data?
8 Sure thing. Here's the temporary authorization code
11 The access token representing the authority to access the API
USER
Exchange temporary value for a token 10
9 Call endpoint with temporary value
1 I want you to read my reviews at *Restograde*
2 Sure, let's go ask the STS for a token
VIRTUAL FOODIE
12 Access API with access token
API

12 *The request to the API using the access token*

```
GET /reviews
Host: api.restograde.com
Authorization: Bearer eyJhbGciO...du6TY9w
```

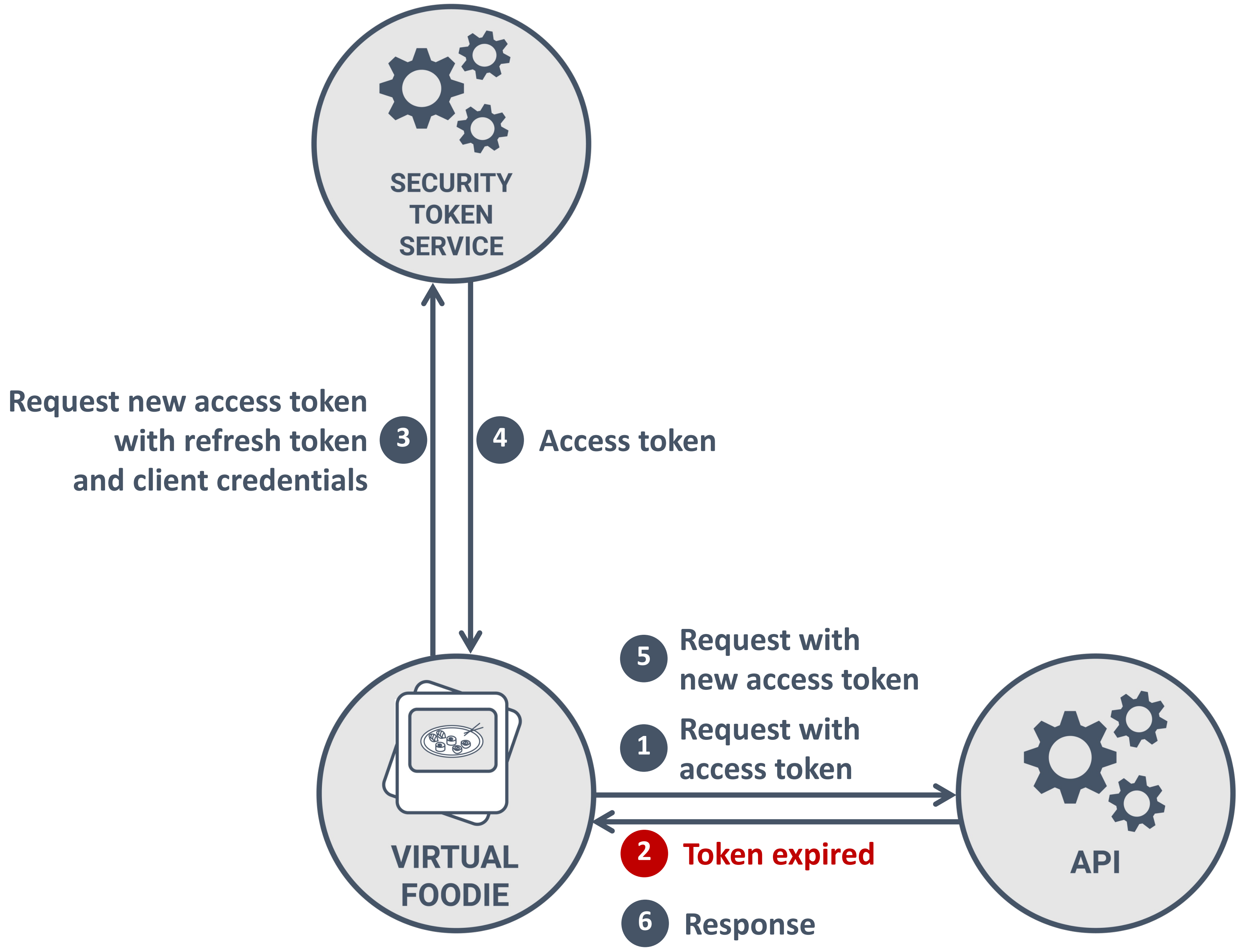
The access token from the OAuth 2.0 flow


THE OAUTH 2.0 AUTHORIZATION CODE FLOW
7 Yes, let's do it!
5 I am Philippe with password FluffyDog17!
3 Get us an access token
4 Who are you? Please authenticate to me!
6 Good. Are you sure you want to give VirtualFoodie access to your review data?
8 Sure thing. Here's the temporary authorization code
SECURITY TOKEN SERVICE
11 The access token representing the authority to access the API
USER
Exchange temporary value for a token 10
9 Call endpoint with temporary value
1 I want you to read my reviews at Restograde
2 Sure, let's go ask the STS for a token
VIRTUAL FOODIE
12 Access API with access token
API

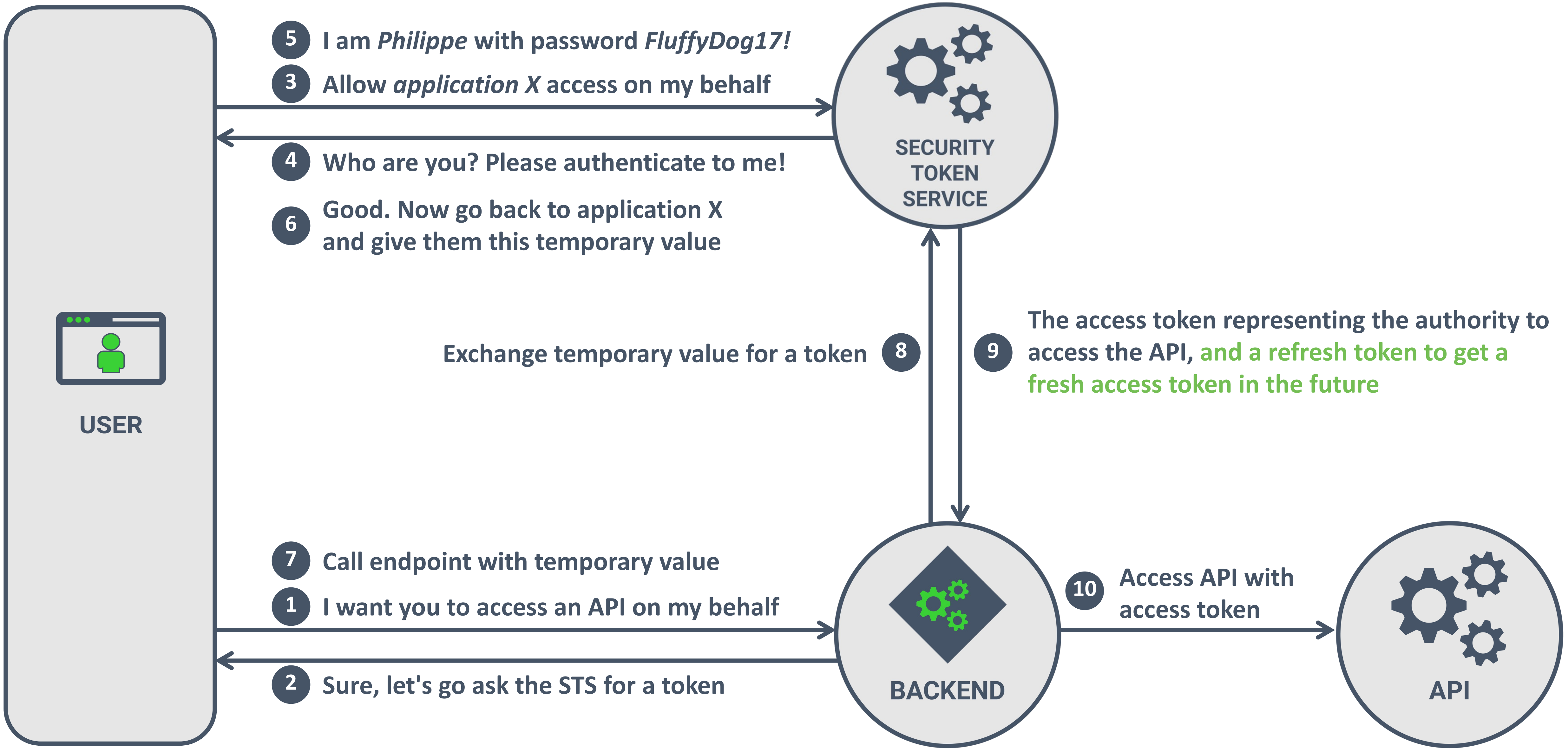
Getting OAuth 2.0 access tokens

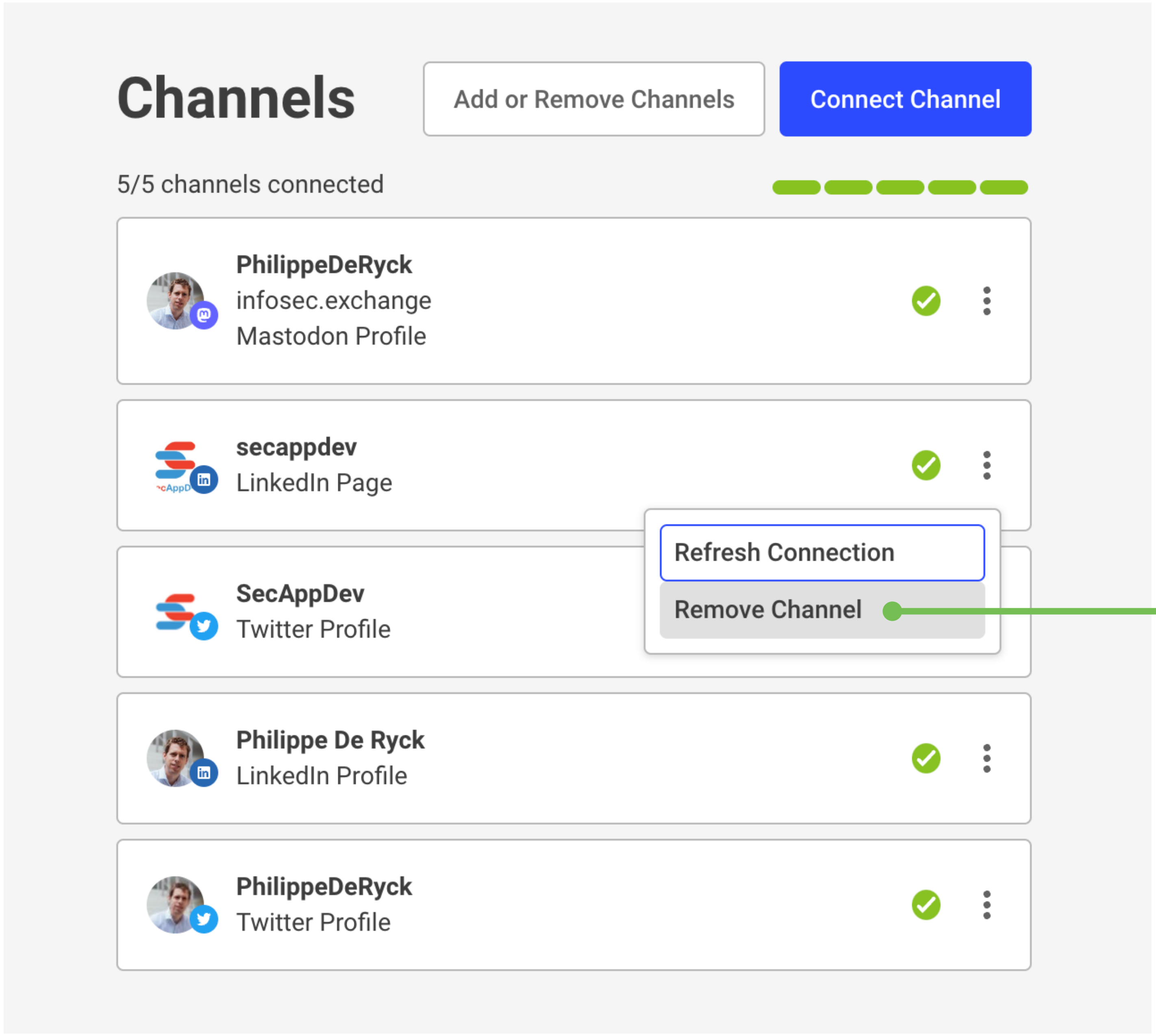
What happens when the access token expires?

THE *REFRESH TOKEN* FLOW


SECURITY TOKEN SERVICE
Request new access token with refresh token and client credentials
3
4
Access token
5
Request with new access token
1
Request with access token
2
Token expired
6
Response
VIRTUAL FOODIE
API

OAUTH 2.0 REFRESH TOKENS


5 I am *Philippe* with password *FluffyDog17!*
3 Allow *application X* access on my behalf
4 Who are you? Please authenticate to me!
6 Good. Now go back to application X and give them this temporary value
SECURITY TOKEN SERVICE
USER
Exchange temporary value for a token 8
9 The access token representing the authority to access the API, and a refresh token to get a fresh access token in the future
7 Call endpoint with temporary value
1 I want you to access an API on my behalf
2 Sure, let's go ask the STS for a token
BACKEND
10 Access API with access token
API



Channels

Add or Remove Channels

Connect Channel

5/5 channels connected

PhilippeDeRyck
infosec.exchange
Mastodon Profile

secappdev
LinkedIn Page

Refresh Connection

Remove Channel

SecAppDev
Twitter Profile

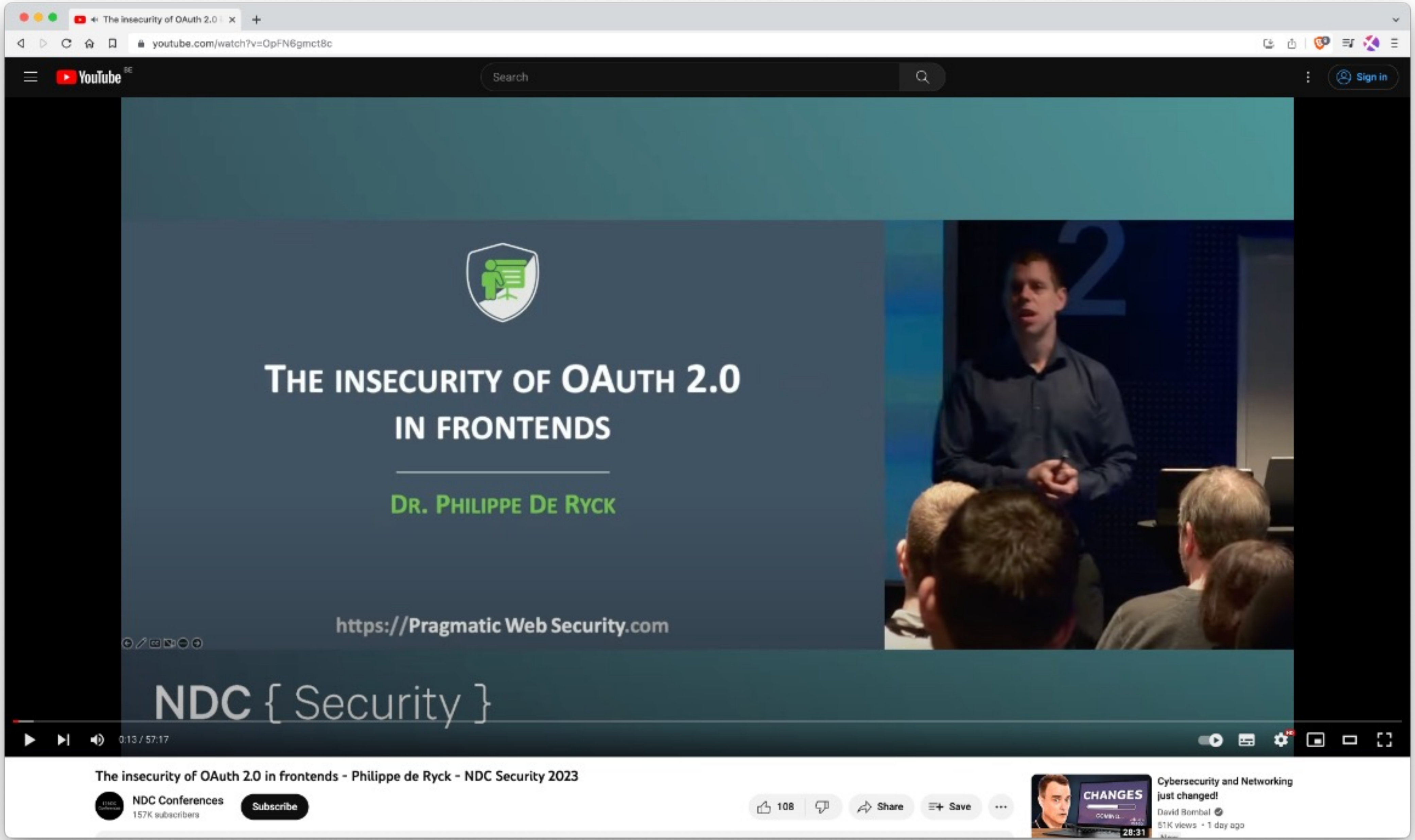
Philippe De Ryck
LinkedIn Profile

PhilippeDeRyck
Twitter Profile

Refresh tokens are often explicitly revocable by the user.

Summarizing access tokens and refresh tokens

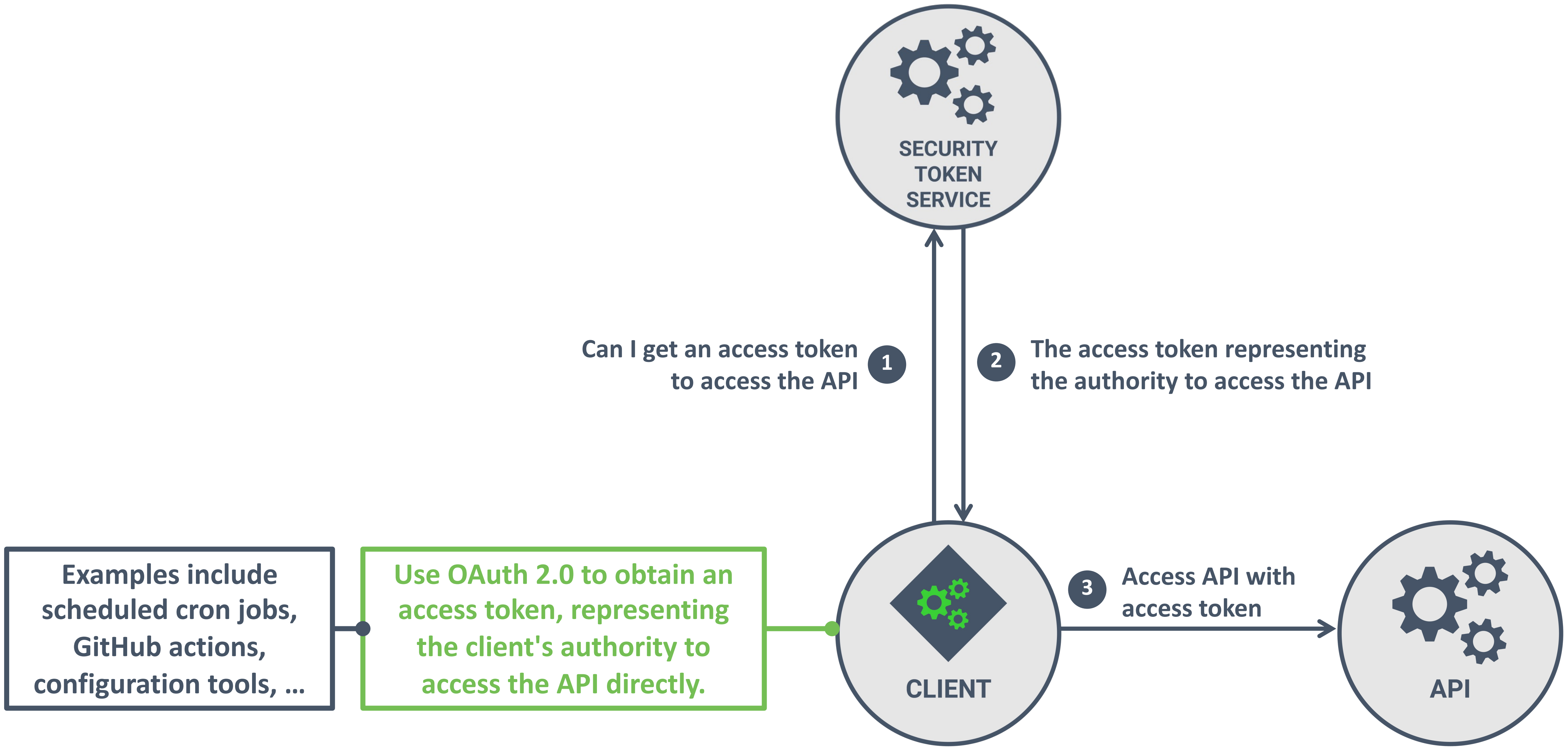
- Access tokens are more exposed than refresh tokens
  - The guideline for access tokens is to keep them short-lived
  - When an access token expires, the refresh token can be used to get a fresh token
- Refresh tokens are consumed by the STS
  - The STS issues them to the client and the client uses them with the STS
  - Refresh tokens are as sensitive as credentials, so they should be handled securely
- The lifetime of refresh tokens is at the discretion of the STS
  - For backend clients, refresh tokens can be valid for months, or even eternally
  - For mobile clients, refresh tokens are stored securely and often long-lived
  - For web clients, refresh tokens should have a lifetime of a few hours


The insecurity of OAuth 2.0
youtube.com/watch?v=OpFN6gmct8c
YouTube
Search
Sign in
THE INSECURITY OF OAUTH 2.0 IN FRONTENDS
DR. PHILIPPE DE RYCK
https://Pragmatic Web Security.com
NDC { Security }
0:13 / 57:17
The insecurity of OAuth 2.0 in frontends - Philippe de Ryck - NDC Security 2023
NDC Conferences
157K subscribers
Subscribe
108
Share
Save
CHANGES
Cybersecurity and Networking just changed!
David Bombal
51K views • 1 day ago
28:31

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

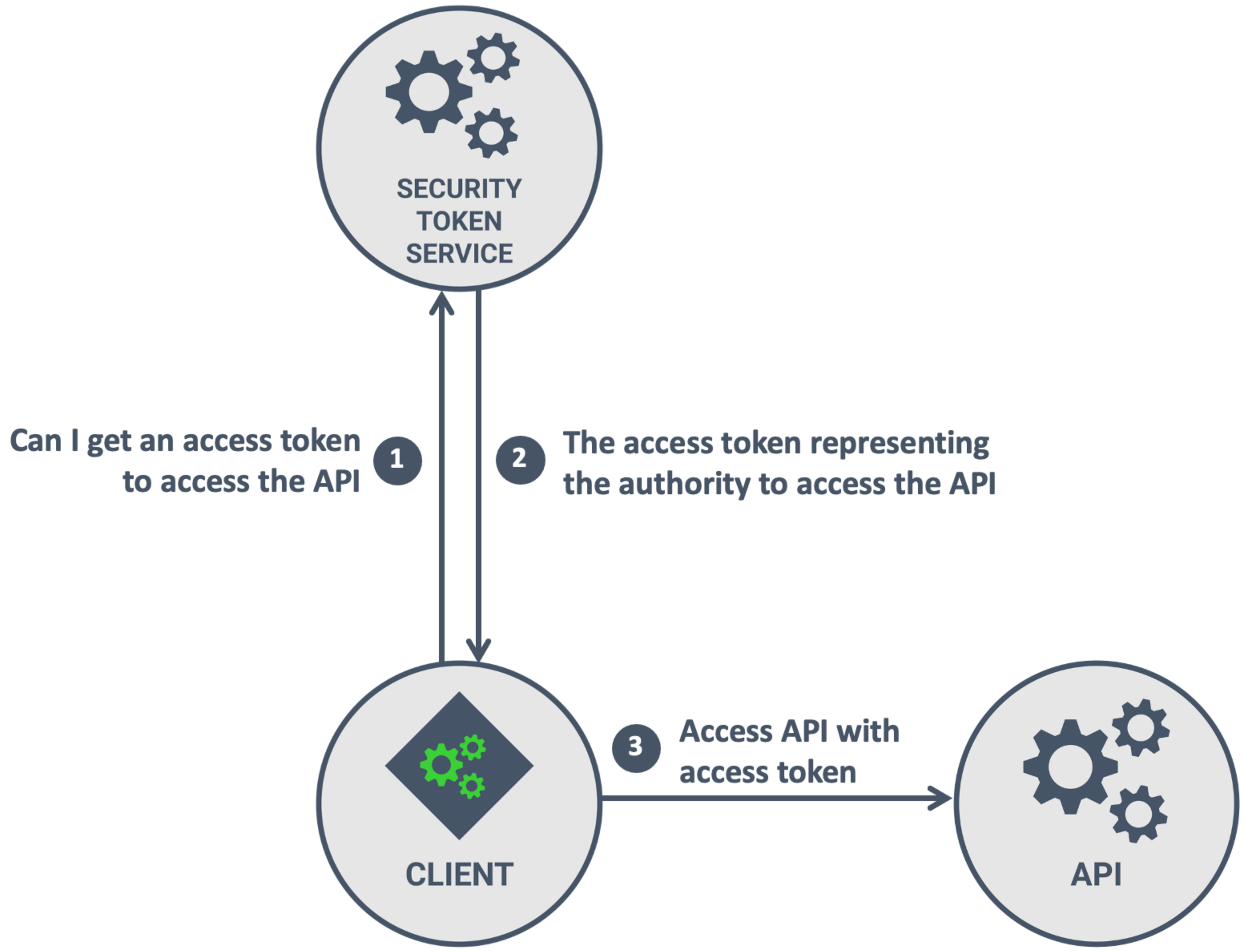
Using OAuth 2.0 without users

Using OAuth 2.0 for machine-to-machine access


SECURITY TOKEN SERVICE
Can I get an access token to access the API
1
2
The access token representing the authority to access the API
Examples include scheduled cron jobs, GitHub actions, configuration tools, ...
Use OAuth 2.0 to obtain an access token, representing the client's authority to access the API directly.
CLIENT
3
Access API with access token
API

Machine-to-machine access in action

The OAuth 2.0 *Client Credentials* flow


SECURITY TOKEN SERVICE
1
Can I get an access token to access the API
2
The access token representing the authority to access the API
CLIENT
3
Access API with access token
API

1 The request to obtain an access token

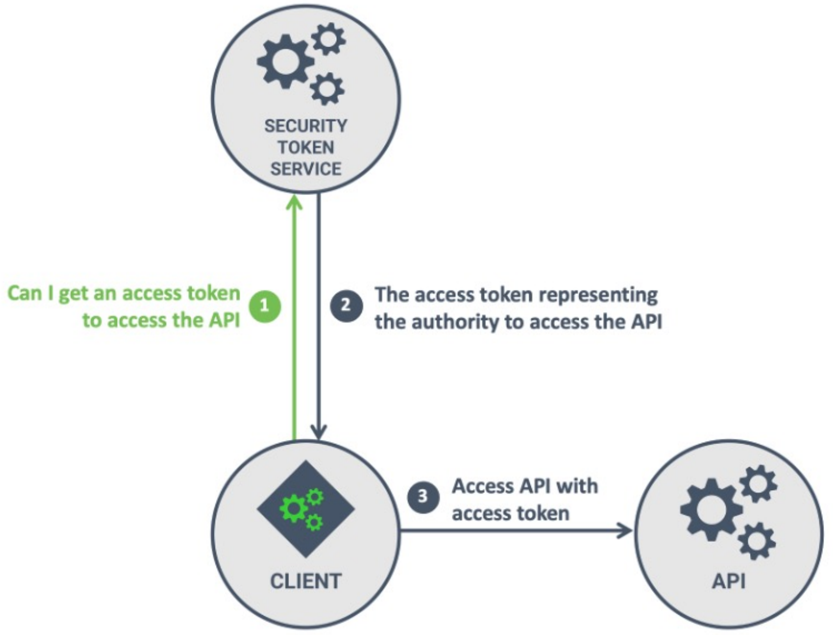
```
POST /oauth/token
Host: sts.restograde.com

 grant_type=client_credentials
&client_id=2JqcsqEpZfYNHxDazVMMkPT6oU6C7ZZS
&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


SECURITY TOKEN SERVICE
1 Can I get an access token to access the API
2 The access token representing the authority to access the API
CLIENT
3 Access API with access token
API

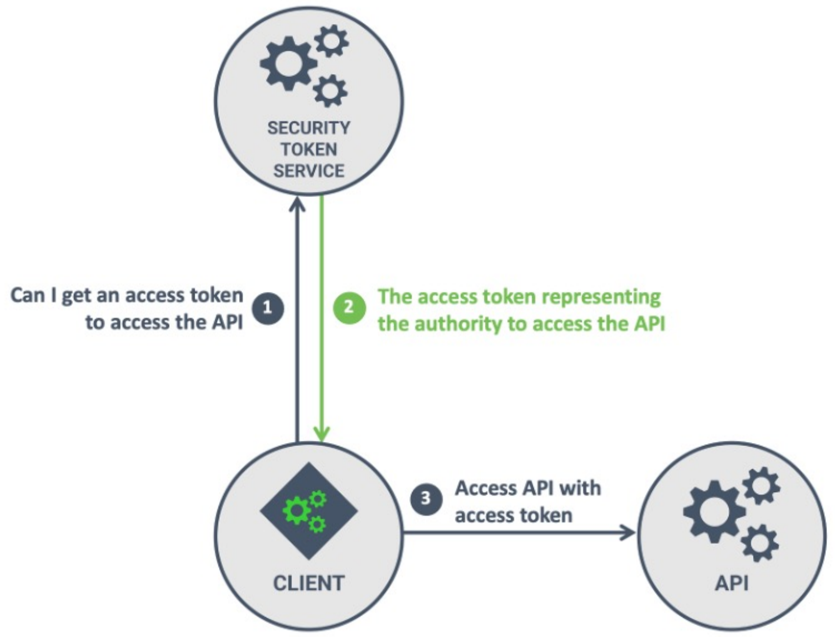
2 The response from the Security Token Service

```
{
  "access_token": "eyJhbGciO…encDDLQ",
  "token_type": "Bearer",
  "expires_in": 3600,
}
```

The access token to access APIs

The expiration time of the access token

THE OAUTH 2.0 *CLIENT CREDENTIALS* FLOW


SECURITY TOKEN SERVICE
Can I get an access token to access the API 1
2 The access token representing the authority to access the API
CLIENT
3 Access API with access token
API

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)

Introduction to Oauth 2.0 and OIDC

Access token types

A self-contained access token

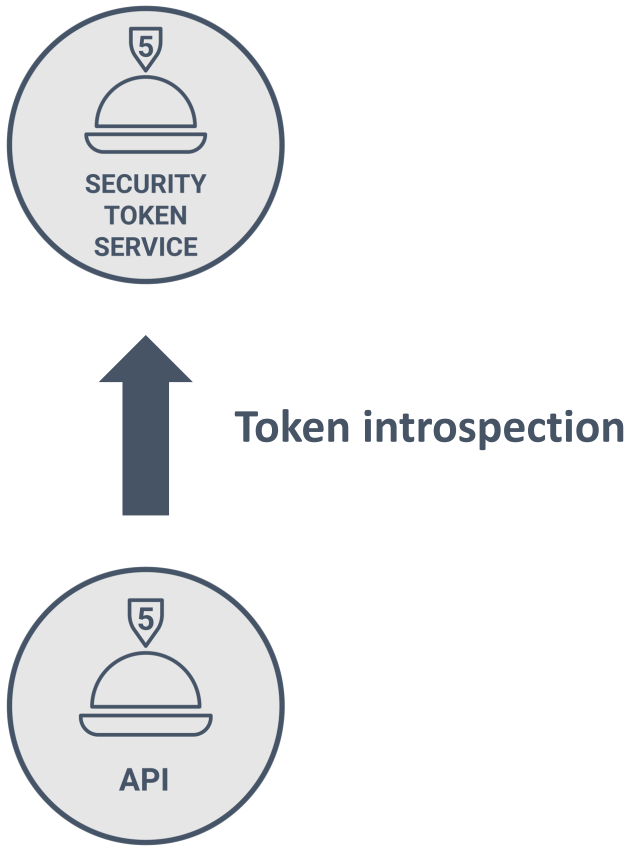
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZC
I6Ik5UVkJPVFUzTXpCQk9FVXdOemhCUTBWR01rUTBR
VVU1UVRZeFFVVXlPVU5FUVVVeE5qRXlNdyJ9.eyJpc
3MiOiJodHRwczovL3N0cy5yZXN0b2dyYWRlLmNvbS8
iLCJzdWIiOiJhdXRoMHw1ZWI5MTZjMjU4YmRiNTBiZ
jIwMzY2YzYiLCJhdWQiOlsiaHR0cHM6Ly9hcGkucmV
zdG9ncmFkZS5jb20iLCJodHRwczovL3Jlc3RvZ3JhZ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

A reference token

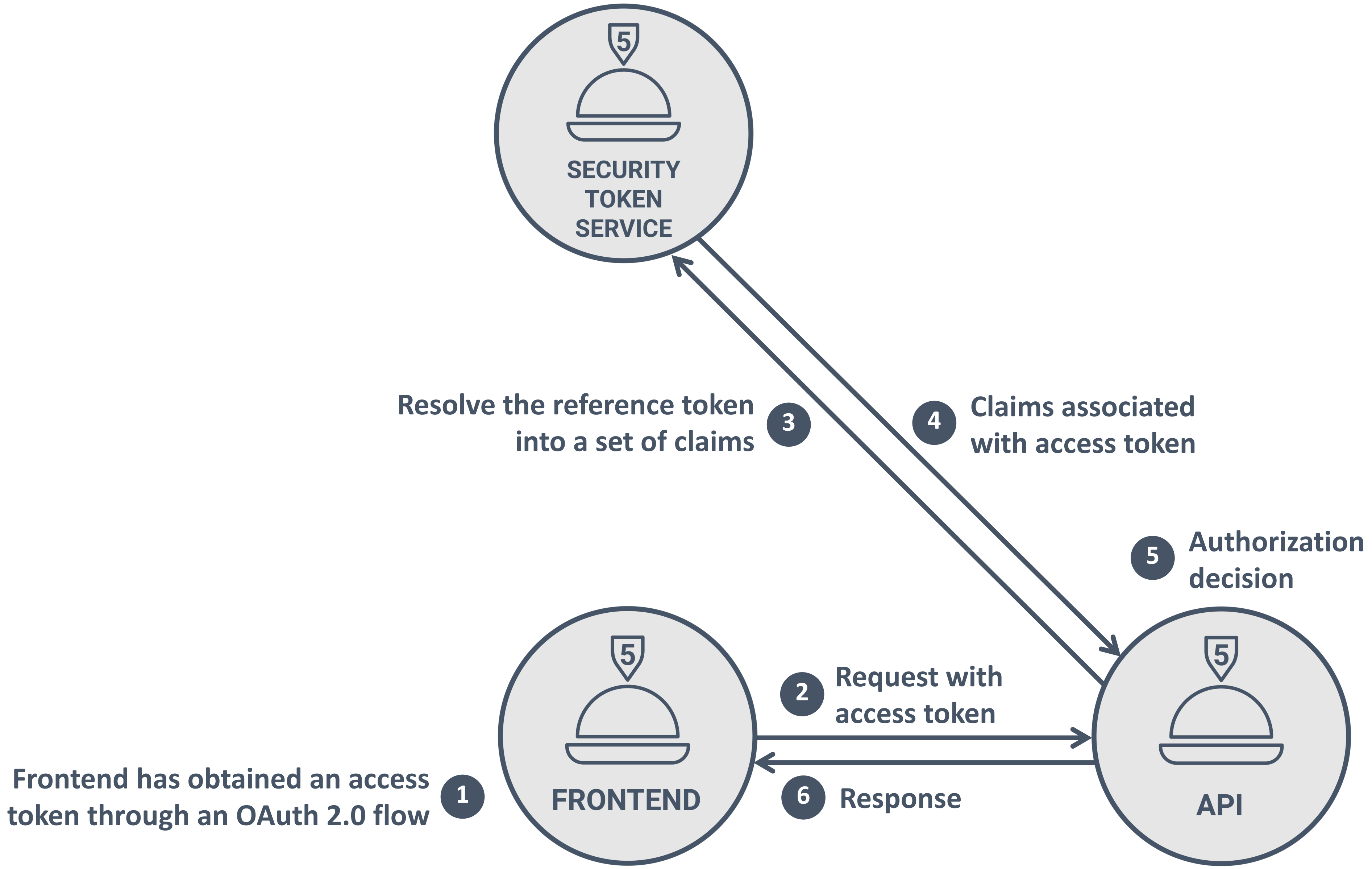
vSvhNDeQLqrzRbvA2eeYE2PthB1cBimS

A reference token

```
vSvhNDeQLqrzRbvA2eeYE2PthB1cBimS
```



Token introspection for reference tokens


SECURITY TOKEN SERVICE
3 Resolve the reference token into a set of claims
4 Claims associated with access token
5 Authorization decision
2 Request with access token
6 Response
1 Frontend has obtained an access token through an OAuth 2.0 flow
FRONTEND
API

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
  - The API can typically rely on a few specific values to be present
    - These include ***iss***, ***client_id***, and ***sub*** if a user is involved
- Ultimately, the STS is in control over what is returned during introspection
  - The returned information can include custom fields
  - Depending on who's asking, more or less information may be included
- The spec also allows token introspection for self-contained tokens (RFC 7662)
  - Introspecting JWTs can be used to detect revocation before the token expires

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
```

A reference token

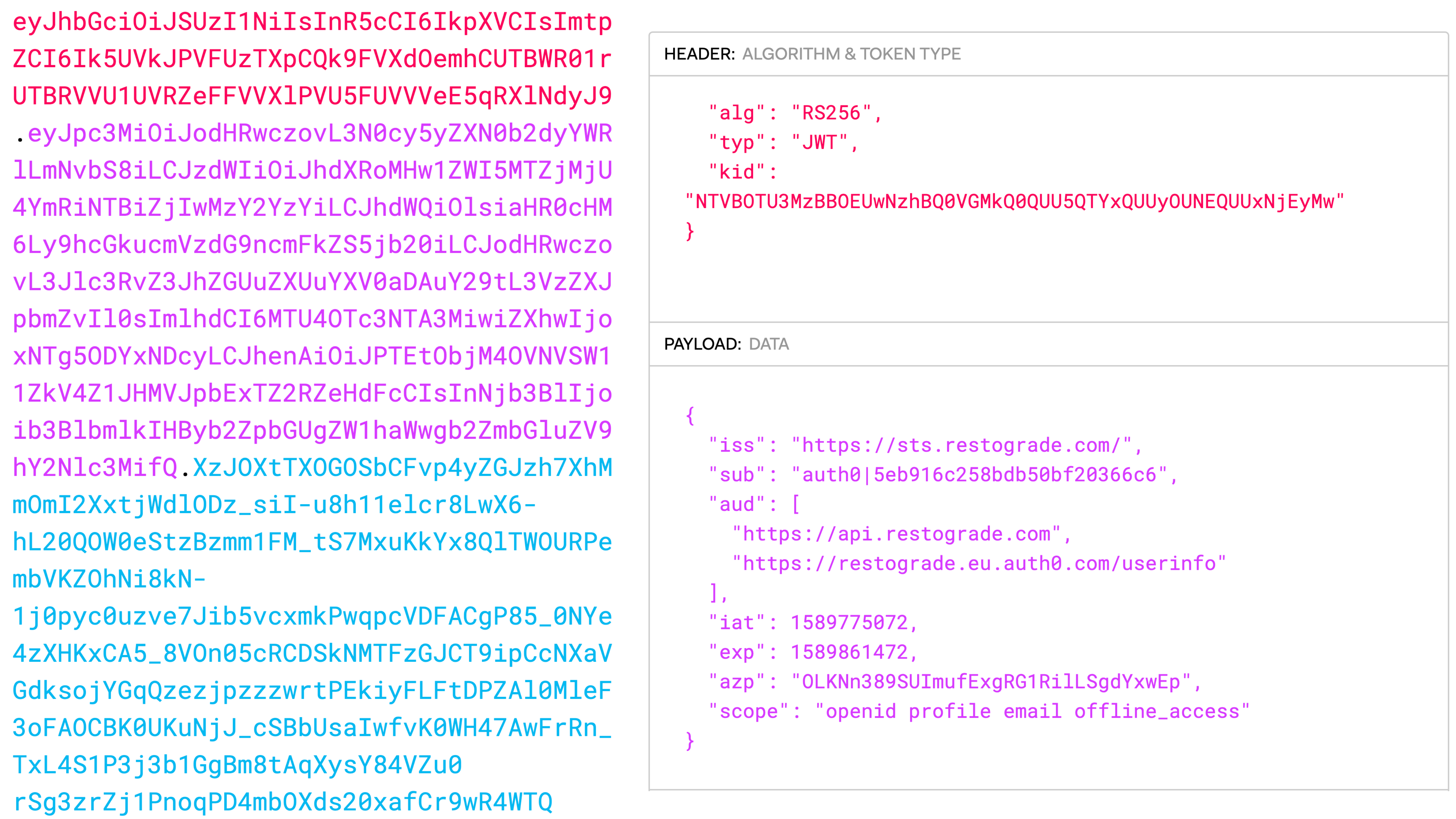
```
vSvhNDeQLqrzRbvA2eeYE2PthB1cBimS
```

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtp
ZCI6Ik5UVkJPVFUzTXpCQk9FVXdOemhCUTBWR01r
UTBRVVU1UVRZeFFVVXlPVU5FUVVVeE5qRXlNdyJ9
.eyJpc3MiOiJodHRwczovL3N0cy5yZXN0b2dyYWR
lLmNvbS8iLCJzdWIiOiJhdXRoMHw1ZWI5MTZjMjU
4YmRiNTBiZjIwMzY2YzYiLCJhdWQiOlsiaHR0cHM
6Ly9hcGkucmVzdG9ncmFkZS5jb20iLCJodHRwczo
vL3Jlc3RvZ3JhZGUuZXUuYXV0aDAuY29tL3VzZXJ
pbmZvIl0sImlhdCI6MTU4OTc3NTA3MiwiZXhwIjo
xNTg5ODYxNDcyLCJhenAiOiJPTEtObjM4OVNVSW1
1ZkV4Z1JHMVJpbExTZ2RZeHdFcCIsInNjb3BlIjo
ib3BlbmlkIHByb2ZpbGUgZW1haWwgb2ZmbGluZV9
hY2Nlc3MifQ.XzJOXtTXOGOSbCFvp4yZGJzh7XhM
mOmI2XxtjWdlODz_siI-u8h11elcr8LwX6-
hL20QOW0eStzBzmm1FM_tS7MxuKkYx8QlTWOURPe
mbVKZOhNi8kN-
1j0pyc0uzve7Jib5vcxmkPwqpcVDFACgP85_0NYe
4zXHKxCA5_8VOn05cRCDSkNMTFzGJCT9ipCcNXaV
GdksojYGqQzezjpzzzwrtPEkiyFLFtDPZAl0MleF
3oFAOCBK0UKuNjJ_cSBbUsaIwfvK0WH47AwFrRn_
TxL4S1P3j3b1GgBm8tAqXysY84VZu0
rSg3zrZj1PnoqPD4mbOXds20xafCr9wR4WTQ
```

Header with token metadata

Payload with a set of claims

Signature protecting the header and payload



```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtp
ZCI6Ik5UVkJPVFUzTXpCQk9FVXdOemhCUTBWR01r
UTBRVVU1UVRZeFFVVXlPVU5FUVVVeE5qRXlNdyJ9
.eyJpc3MiOiJodHRwczovL3N0cy5yZXN0b2dyYWR
lLmNvbS8iLCJzdWIiOiJhdXRoMHw1ZWI5MTZjMjU
4YmRiNTBiZjIwMzY2YzYiLCJhdWQiOlsiaHR0cHM
6Ly9hcGkucmVzdG9ncmFkZS5jb20iLCJodHRwczo
vL3Jlc3RvZ3JhZGUuZXUuYXV0aDAuY29tL3VzZXJ
pbmZvIl0sImlhdCI6MTU4OTc3NTA3MiwiZXhwIjo
xNTg5ODYxNDcyLCJhenAiOiJPTEtObjM4OVNVSW1
1ZkV4Z1JHMVJpbExTZ2RZeHdFcCIsInNjb3BlIjo
ib3BlbmlkIHByb2ZpbGUgZW1haWwgb2ZmbGluZV9
hY2Nlc3MifQ.XzJOXtTXOGOSbCFvp4yZGJzh7XhM
mOmI2XxtjWdlODz_siI-u8h11elcr8LwX6-
hL20QOW0eStzBzmm1FM_tS7MxuKkYx8QlTWOURPe
mbVKZOhNi8kN-
1j0pyc0uzve7Jib5vcxmkPwqpcVDFACgP85_0NYe
4zXHKxCA5_8VOn05cRCDSkNMTFzGJCT9ipCcNXaV
GdksojYGqQzezjpzzzwrtPEkiyFLFtDPZAl0MleF
3oFAOCBK0UKuNjJ_cSBbUsaIwfvK0WH47AwFrRn_
TxL4S1P3j3b1GgBm8tAqXysY84VZu0
rSg3zrZj1PnoqPD4mbOXds20xafCr9wR4WTQ
```

HEADER: ALGORITHM & TOKEN TYPE

```
   "alg": "RS256",
   "typ": "JWT",
   "kid":
 "NTVBOTU3MzBBOEUwNzhBQ0VGMkQ0QUU5QTYxQUUyOUNEQUUxNjEyMw"
 }
```

PAYLOAD: DATA

```
 {
   "iss": "https://sts.restograde.com/",
   "sub": "auth0|5eb916c258bdb50bf20366c6",
   "aud": [
     "https://api.restograde.com",
     "https://restograde.eu.auth0.com/userinfo"
   ],
   "iat": 1589775072,
   "exp": 1589861472,
   "azp": "OLKNn389SUImufExgRG1RilLSgdYxwEp",
   "scope": "openid profile email offline_access"
 }
```

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
- Self-contained tokens are signed by the STS, ensuring their integrity
  - The API ***must*** verify the integrity of a self-contained access token before using the data
  - Verification is typically done by checking the signature with a public key of the STS
- All of these details are 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

Which token type do you prefer?

A Reference tokens

B Self-contained tokens

Which token type has better performance properties?

A Reference tokens

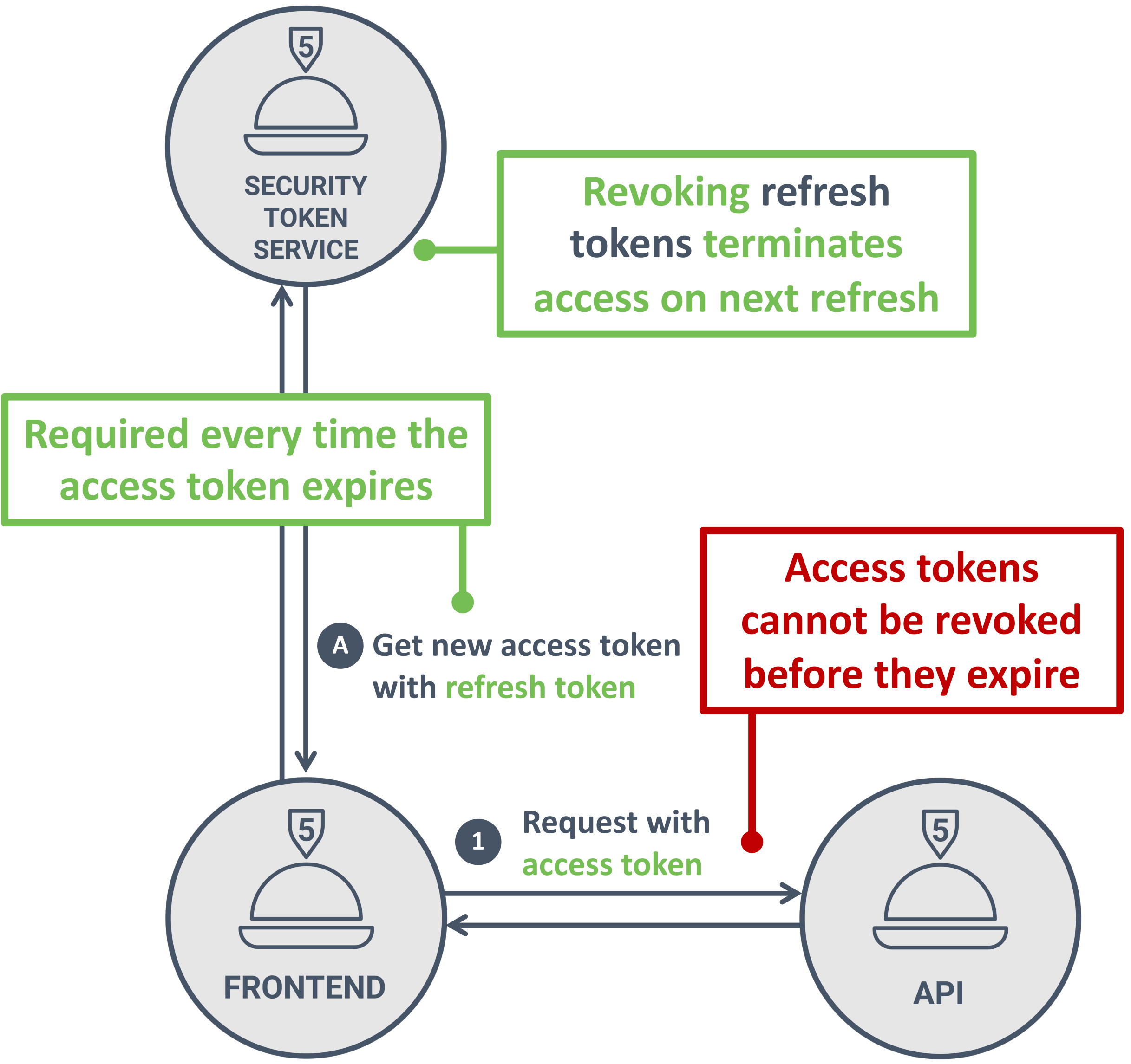
B Self-contained tokens

Which token type has better revocation properties?

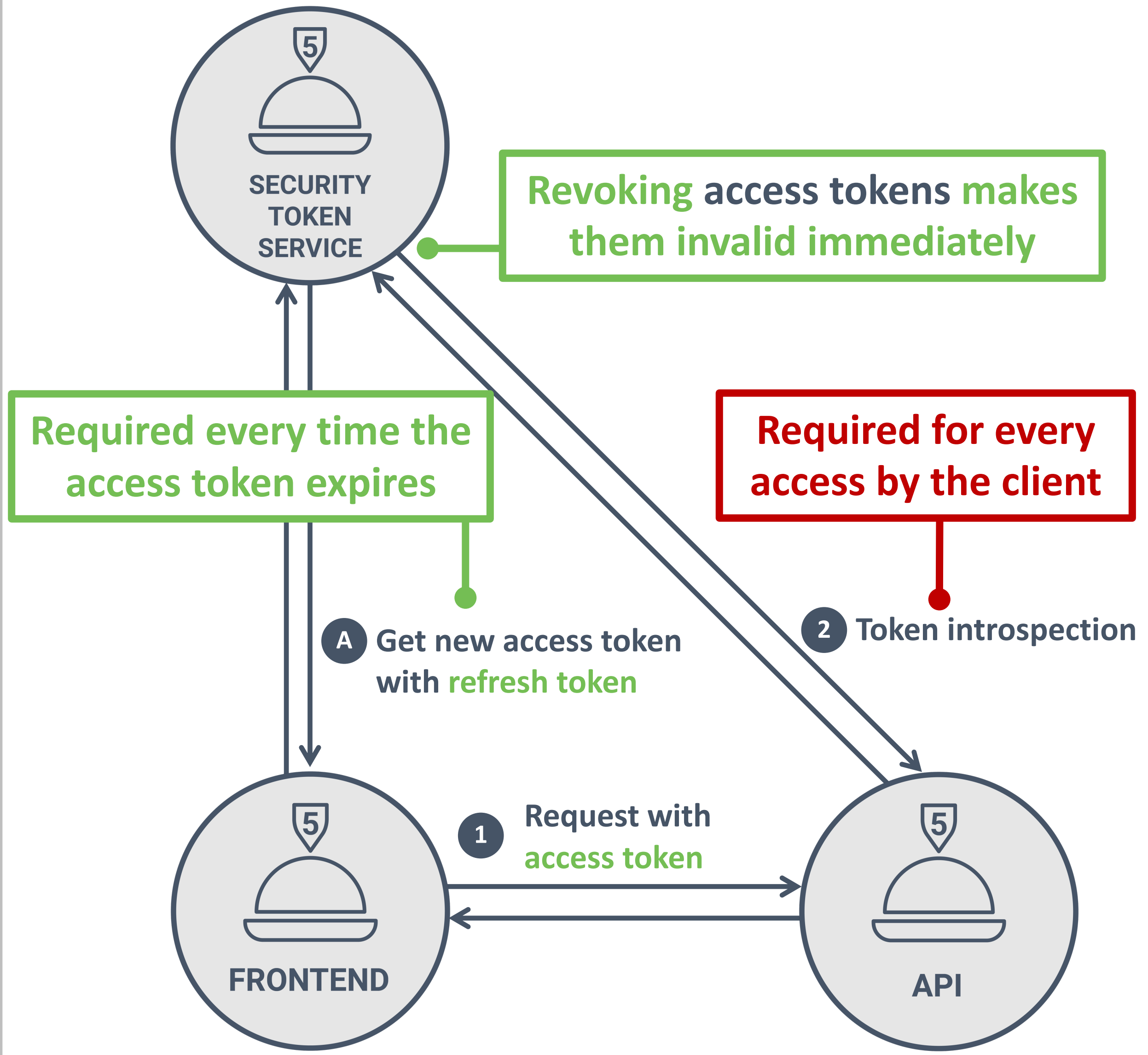
A Reference tokens

B Self-contained tokens

SELF-CONTAINED TOKENS


SECURITY TOKEN SERVICE
Revoking refresh tokens terminates access on next refresh
Required every time the access token expires
A Get new access token with refresh token
Access tokens cannot be revoked before they expire
1 Request with access token
FRONTEND
API

REFERENCE TOKENS


SECURITY TOKEN SERVICE
Revoking access tokens makes them invalid immediately
Required every time the access token expires
Required for every access by the client
A Get new access token with refresh token
2 Token introspection
1 Request with access token
FRONTEND
API

Trade-offs between access token types

- Self-contained tokens can be independently verified by the API
  - The STS is not involved until the access token expires and the client gets a new token
  - Access tokens typically become invalid when they expire
  - Access can be terminated by revoking the client's refresh token
- Reference tokens require token introspection between the API and the STS
  - The STS is always involved, both for token introspection and renewing access tokens
  - Access tokens can be revoked by the STS, making them invalid immediately
  - Caching introspection responses by the API contradicts the security properties
    - Caching is acceptable for handling bursts of requests, but only for 10 – 20 seconds
- The most important trade-off is about security vs performance
  - Reference tokens have better security properties, but they come at a cost

Practical guidelines on access token types

- How short can you make your access token's lifetime?
  - Short lifetimes reduce the window of abuse and force the client to contact the STS
  - Frontend applications are more sensitive, so should have shorter token lifetimes
    - 5 - 10 minutes is quite common
- How important is revocation for your application?
  - If a small potential window of abuse is acceptible, short token lifetimes are a good option
  - If no abuse is acceptible, reference tokens offer the most control
- Revocation sounds great on paper, but can you implement it?
  - *Manual* revocation processes will be ineffective with token lifetimes of 5 – 10 minutes
  - *Automatic* revocation with anomaly-detection systems would be effective

Access token types

- The STS decides on the security properties of access tokens
  - Clients only send access tokens, so they are agnostic of the token type and its properties
  - The API will need to understand how to process different token types
- In practice, self-contained JWT tokens are common for distributed scenarios
  - Running token introspection between different parties is often difficult
  - Keep token lifetimes as short as possible
- Reference tokens are often used for internal systems
  - *On-premise* token introspection is easier to implement
  - Can also be implemented with an API gateway that translates tokens

APIs are responsible for enforcing authorization

- OAuth 2.0 offers a way to transport user / client information to the API
    - The API relies on this information to make authorization decisions
    - Complex systems should avoid overloading access tokens and use a policy service instead
- APIs are responsible for verifying the validity of incoming tokens
    - Verify the validity of the incoming access token (signature or introspection)
    - Enforce restrictions on the sender of the token if applicable
    - Verify the properties of the access token (issuer, audience, …)
- Libraries / middleware can handle most of these responsibilities
    - Make sure your library / middleware / framework handles tokens correctly

Enforcing authorization with access tokens

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

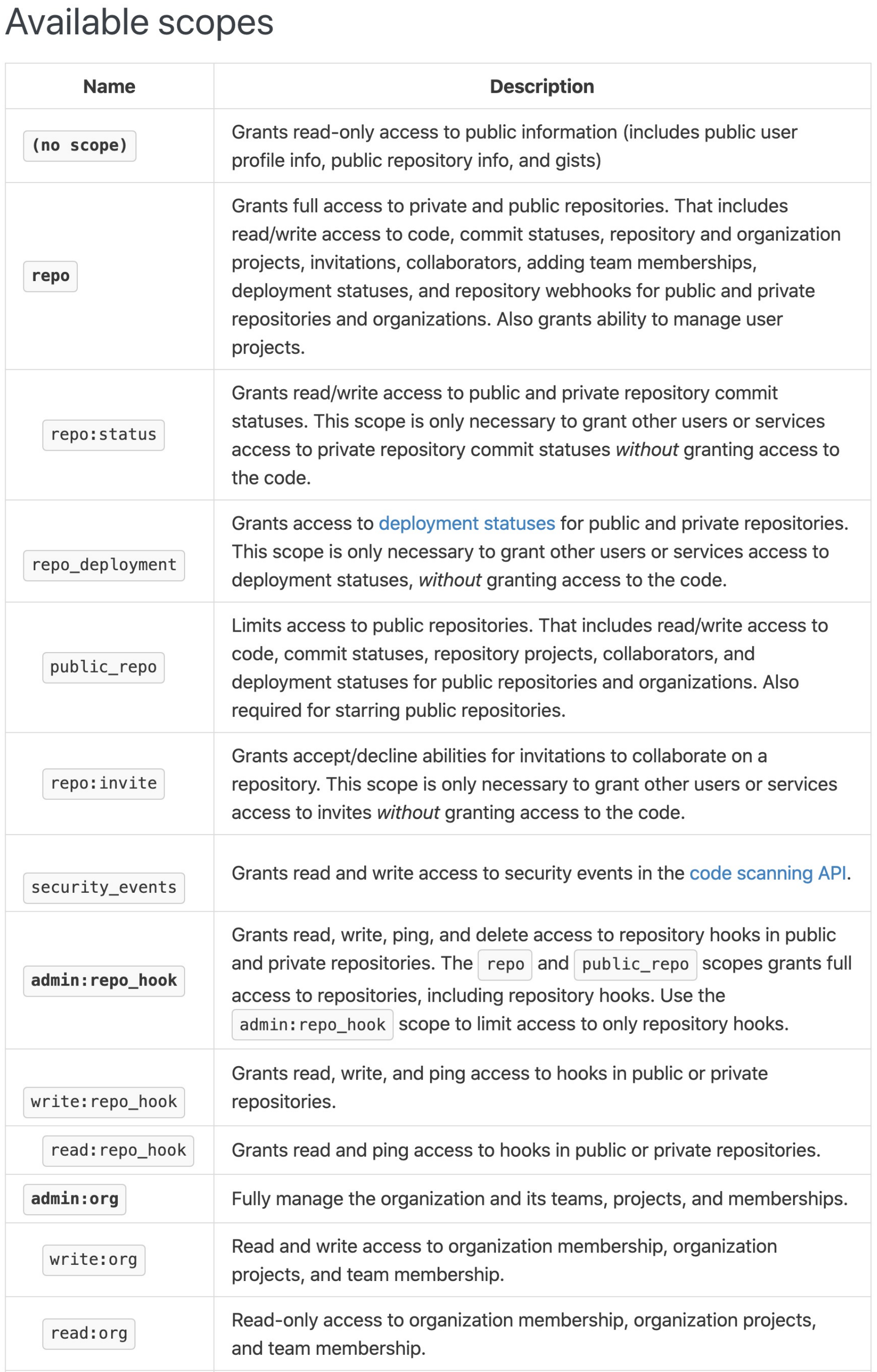
Scopes	
profile	View your basic profile info
email	View your email address
openid	Authenticate using OpenID Connect

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

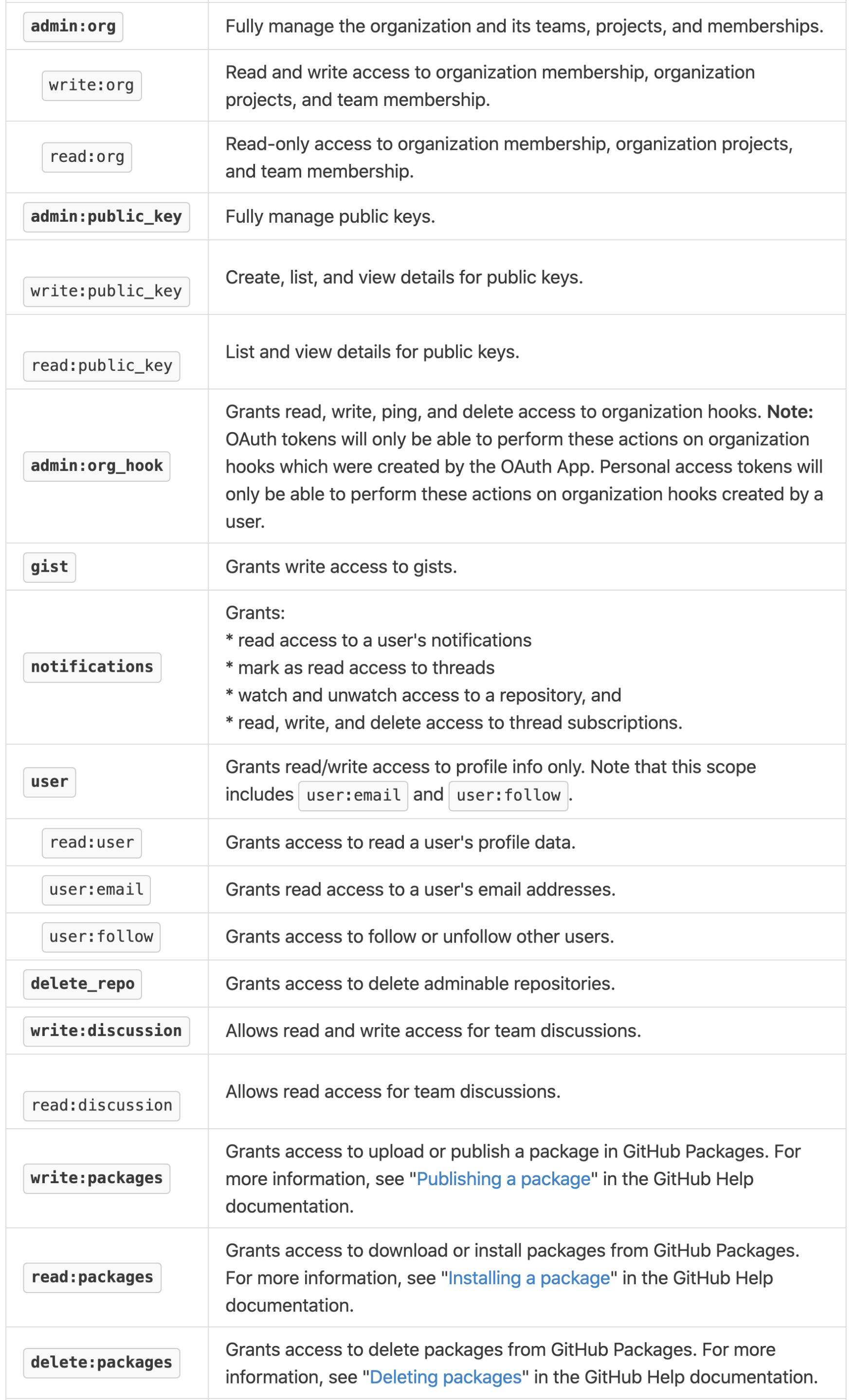
Google Slides API, v1

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



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.

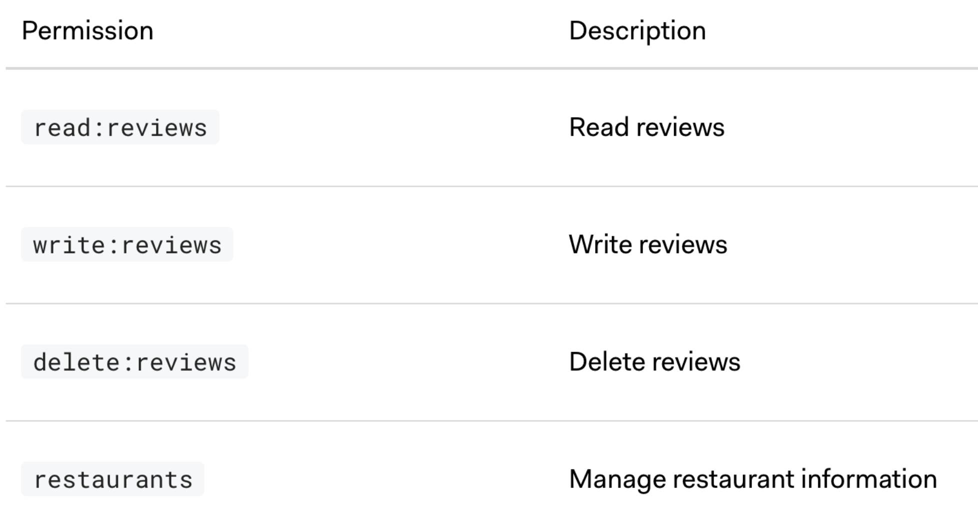


`admin:org`	Fully manage the organization and its teams, projects, and memberships.
`write:org`	Read and write access to organization membership, organization projects, and team membership.
`read:org`	Read-only access to organization membership, organization projects, and team membership.
`admin:public_key`	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.

PRACTICAL GUIDELINES FOR DEFINING SCOPES

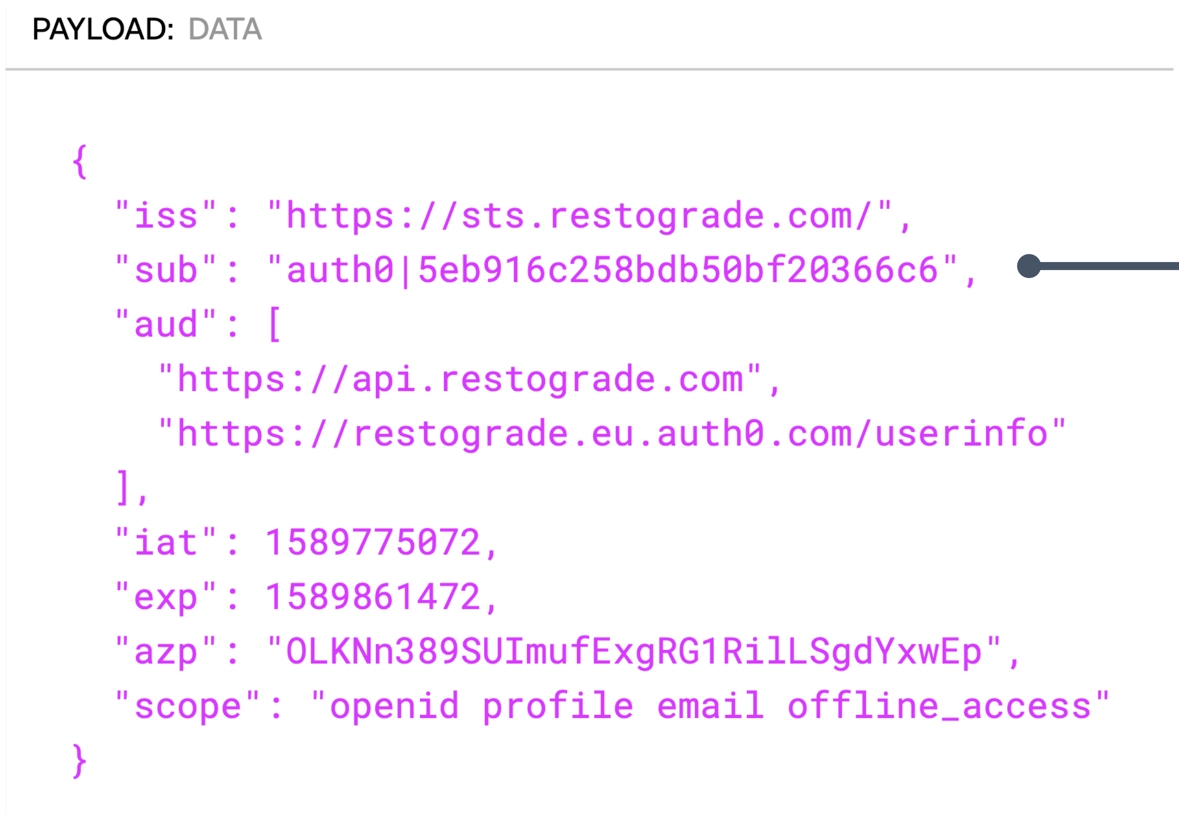
- Unless you are Google, you probably do not need hundreds of scopes
  - People sometimes run into length limits for the scope parameter, which is a bad smell
  - If clients need access to every API in the system, then you don't need scopes
    - Scopes enforce compartmentalization, but do not replace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

MAKING SPECIFIC AUTHORIZATION DECISIONS



PAYLOAD: DATA

```
{
  "iss": "https://sts.restograde.com/",
  "sub": "auth0|5eb916c258bdb50bf20366c6",
  "aud": [
    "https://api.restograde.com",
    "https://restograde.eu.auth0.com/userinfo"
  ],
  "iat": 1589775072,
  "exp": 1589861472,
  "azp": "OLKNn389SUImufExgRG1RilLSgdYxwEp",
  "scope": "openid profile email offline_access"
}
```

The *sub* points to the subject, which is typically the user on whose behalf the request is being made

Making specific authorization decisions

- User-related access tokens carry a ***sub*** claim
  - The ***sub*** is a unique identifier for a particular user within the issuer
  - With the user's identifier, the API can make user-specific authorization decisions
    - E.g., checking object-level permissions

- The value of the ***sub*** is guaranteed to be unique and immutable for an issuer
  - Typically, the ***sub*** value is a randomly generated identifier
  - The issuer will also ensure that the ***sub*** value cannot be reused by other accounts

- The ***sub*** only applies to a specific issuer, so no uniqueness across issuers
  - For most APIs, this does not represent a problem since only one issuer is trusted
  - For APIs serving multiple issuers, the issuer and the ***sub*** value need to be combined

Adding authorization information to access tokens

- Access tokens represent an authorization given to a client
    - They are intended to replace other constructs (e.g., username / password)
    - Access tokens granting authority on behalf of a user carry information about the user
- Access tokens are not supposed to carry API-specific authorization information
    - The OAuth 2.0 spec does not explicitly state this and custom claims can be added
    - Practical implementations often start adding custom claims to support authorization
- Adding authorization information to access tokens raises some issues
    - How many permissions will be added and what about access token size?
    - What is the token lifetime and what about stale permissions?
    - Will you ever be able to change your permission system?

COMMON SCENARIOS USING CUSTOM ACCESS TOKEN CLAIMS

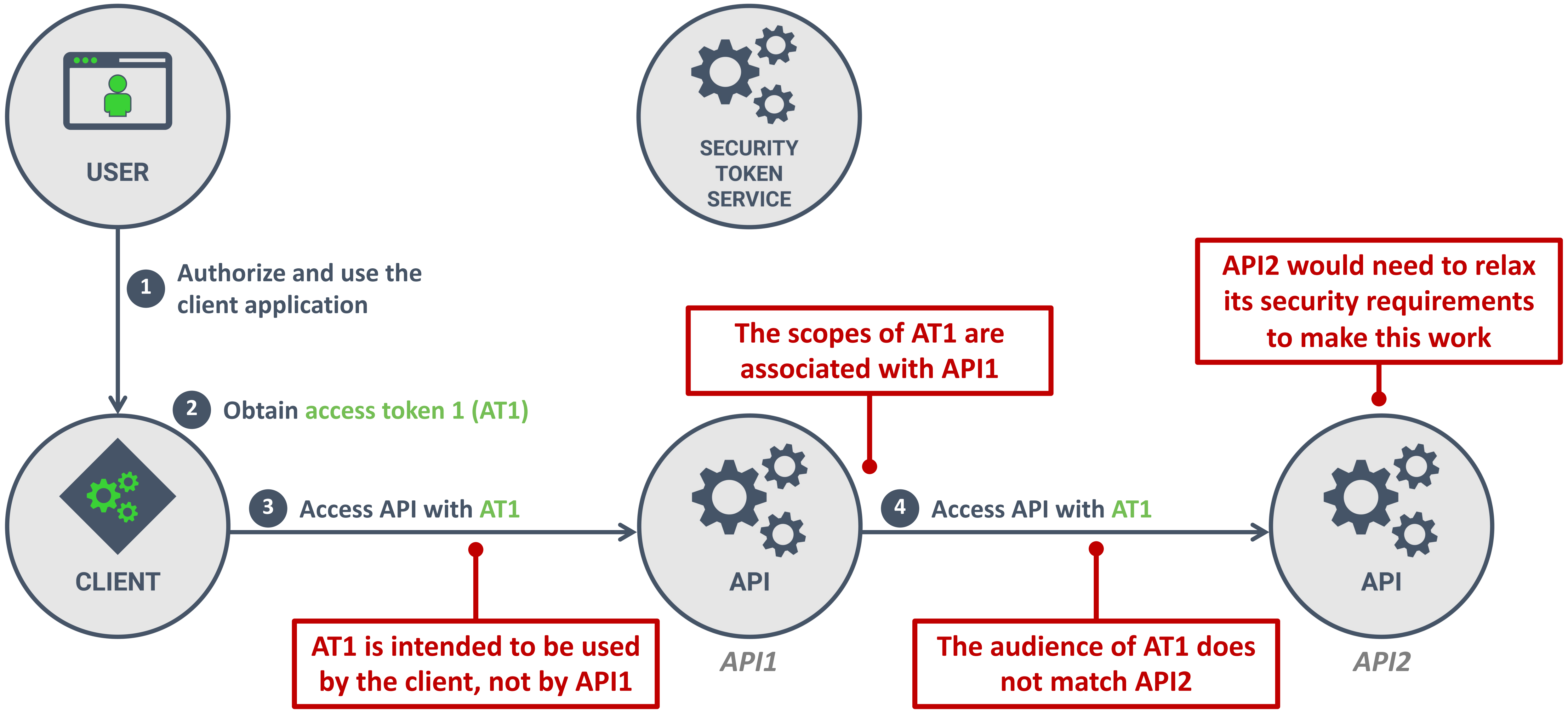
- Adding additional user-specific claims to support authorization decisions
  - E.g., ***customerID*** or ***tenantID*** are common in multi-tenant scenarios
  - Unlikely to change in the future and fully within the spirit of access tokens

- Adding user-specific permissions in a separate ***permissions*** claim
  - Requires the STS to be aware of every API's permissions
  - Less in the spirit of access tokens, since permissions are not about the user's identity

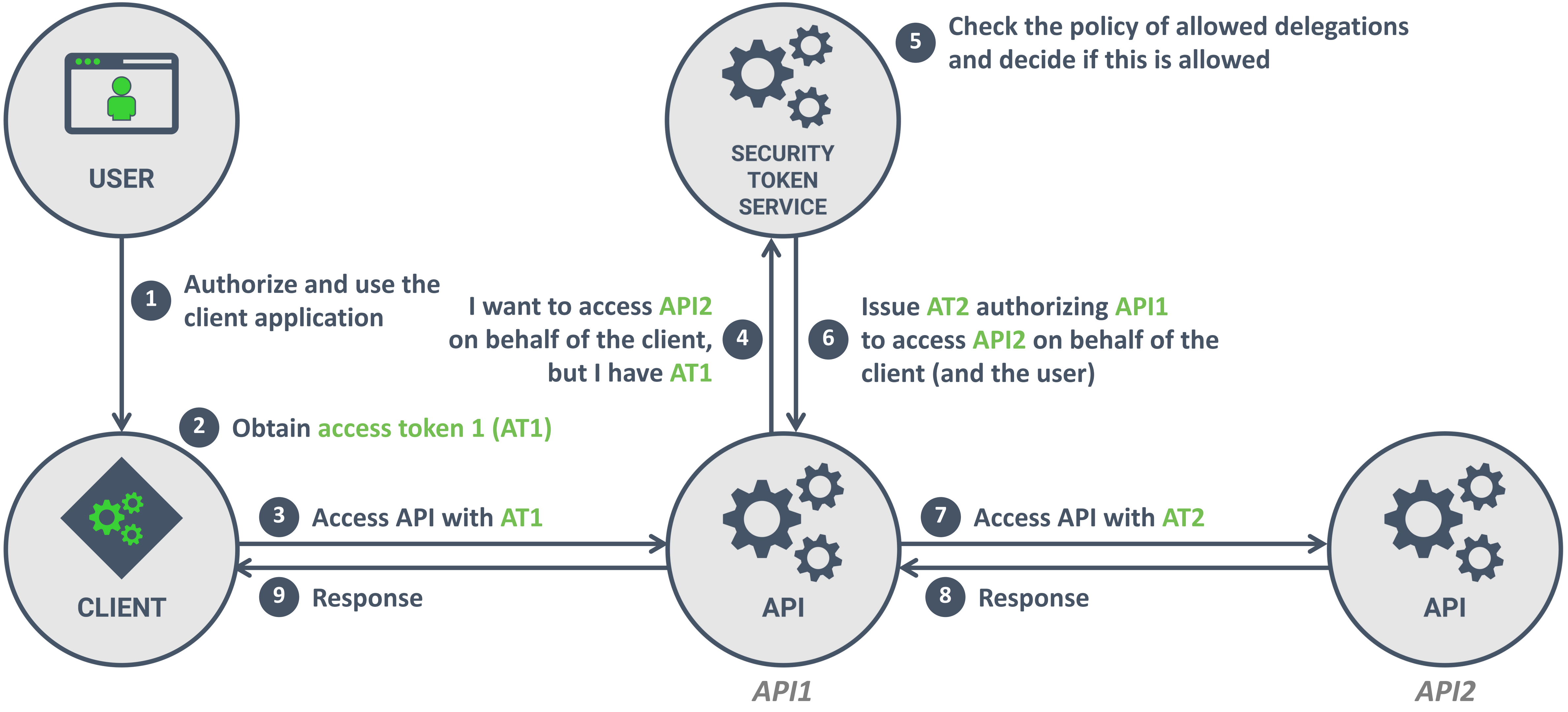
- Adding user roles to access tokens in a separate ***roles*** claim
  - Very common due to existing RBAC systems
  - Unlikely to cause major issues, since roles are not API-specific and belong to a user

Delegation in OAuth 2.0

A NAÏVE APPROACH TO DELEGATION


USER
SECURITY TOKEN SERVICE
1 Authorize and use the client application
2 Obtain access token 1 (AT1)
CLIENT
3 Access API with AT1
API
API1
4 Access API with AT1
API
API2
The scopes of AT1 are associated with API1
API2 would need to relax its security requirements to make this work
AT1 is intended to be used by the client, not by API1
The audience of AT1 does not match API2

The concept of proper delegation


USER
SECURITY TOKEN SERVICE
5 Check the policy of allowed delegations and decide if this is allowed
1 Authorize and use the client application
4 I want to access API2 on behalf of the client, but I have AT1
6 Issue AT2 authorizing API1 to access API2 on behalf of the client (and the user)
2 Obtain access token 1 (AT1)
CLIENT
3 Access API with AT1
9 Response
API
API1
7 Access API with AT2
8 Response
API
API2

Two common approaches

- Impersonation hides the delegation aspect, but relies on *correct* tokens
  - Instead of forwarding tokens with the wrong properties, API1 obtains a new token
  - The new token makes API1 the *client*, thus providing correct information to API2
  - API2 does not know that the request is on behalf of a client that called API1

- Delegation propagates the relevant information, preserving proper semantics
  - The newly issued token will inform API2 that the call is from API1 on behalf of the *client*
  - This token allows API2 to make a fully informed authorization decision

- The STS is responsible for deciding which delegation is allowed
  - Policies involve the different actors, the granted and requested scopes, …

Delegation in OAuth 2.0

- RFC 8693 defines the mechanisms of a *Token Exchange* mechanism
  - The document focuses on the interactions, not the semantics of a token exchange
  - The semantics and the implementation details are custom for each STS

- Use cases that can be implemented with a token exchange mechanism
  - Calling additional APIs on behalf of the original client with the proper semantics
  - Obtaining a user impersonation token as an admin user
  - *Translating* external identity tokens into internal tokens

- Examples of systems that currently support these concepts
  - Keycloak supports a token exchange based on RFC 8693 for these use cases
  - Microsoft supports "On Behalf Of" flows for API delegation, but not RFC8693

All these delegation concepts require a massive amount of work to get working …

Build a solid service architecture first

- Advanced delegation concepts require a solid foundation
  - Implementing delegation requires each API to authenticate as a client
  - Doing all of this at once is very unlikely to succeed

- Start by implementing restrictions between services
  - mTLS is the preferred mechanism to enforce access policies between services
    - Authorization decisions here are made based on API identities, ***not user request properties***
    - Supported by numerous frameworks and libraries, including Istio's service mesh
  - Successfully implementing this gives you a first understanding of interaction patterns

- Once available, mTLS can be re-used as a client authentication mechanism
  - Implement delegation step-by-step, learning more about the practicalities along the way

Takeaways

References

The RFC discussing OAuth 2.0 security best current practices (essential reading!)

https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics

An article discussing patterns that translates between token types in a reverse proxy setup

https://thenewstack.io/securely-scaling-the-myriad-apis-in-real-world-backend-platforms/

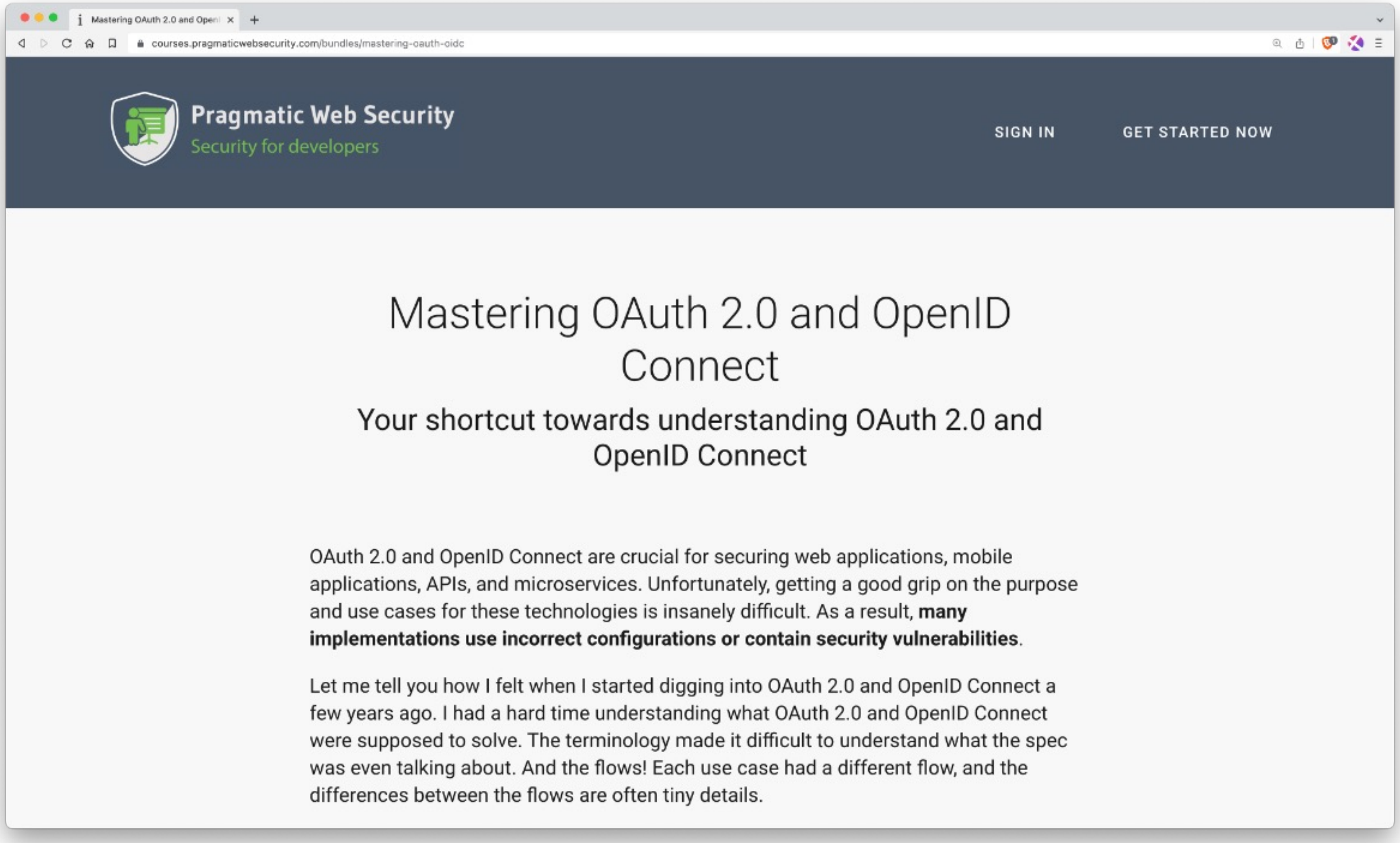
A series of articles on various OAuth 2.0 topics on my website

https://pragmaticwebsecurity.com/articles/tags/oauth.html

Offensive exercises on OAuth 2.0 flows

https://portswigger.net/web-security/all-labs#oauth-authentication

CHECK OUT MY ONLINE COURSE ON OAUTH 2.0 AND OIDC



Mastering OAuth 2.0 and Open
courses.pragmaticwebsecurity.com/bundles/mastering-oauth-oidc
Pragmatic Web Security
Security for developers
SIGN IN
GET STARTED NOW

Mastering OAuth 2.0 and OpenID Connect

Your shortcut towards understanding OAuth 2.0 and OpenID Connect

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.

https://courses.pragmaticwebsecurity.com/bundles/mastering-oauth-oidc

OAuth 2.0 and OpenID Connect

- OAuth 2.0 allows a user to delegate access to a client application
  - Avoids the need for sharing credentials with the client application
  - Defines an authorization framework to allow APIs to make authorization decisions
  - OAuth 2.0 is the de facto standard for implementing distributed authorization scenarios
- OpenID Connect allows a client to delegate authentication to a central provider
  - OIDC is the de facto standard for building modern Single Sign-On systems
  - OIDC uses OAuth 2.0 flows with specific configuration settings
  - OAuth 2.0 and OIDC are typically used together, but can be used separately as well
- How the user authenticates to the central provider is not specified
  - OAuth 2.0 and OIDC define the interactions between the different components