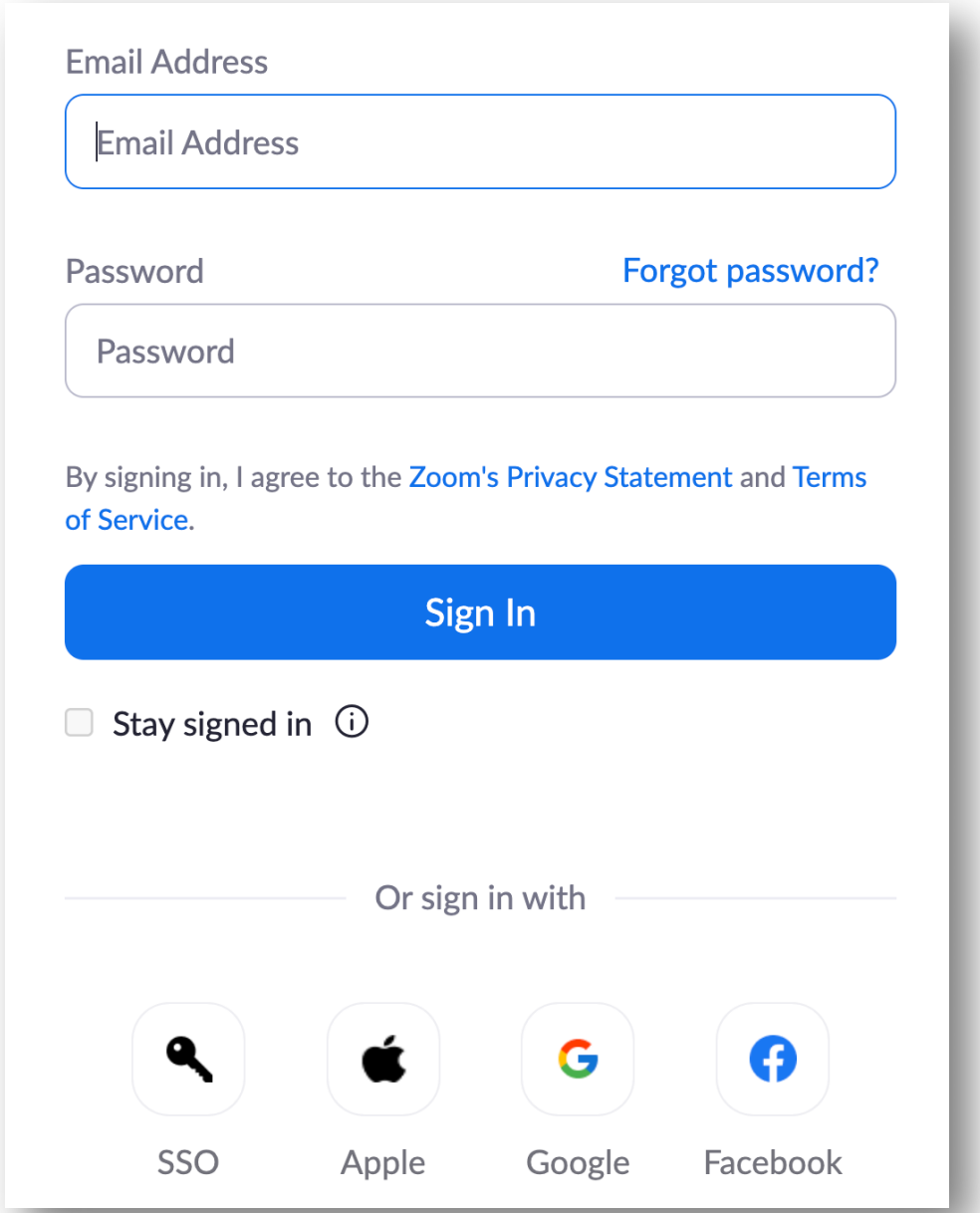
Introduction to OAuth 2.0 and OpenID Connect

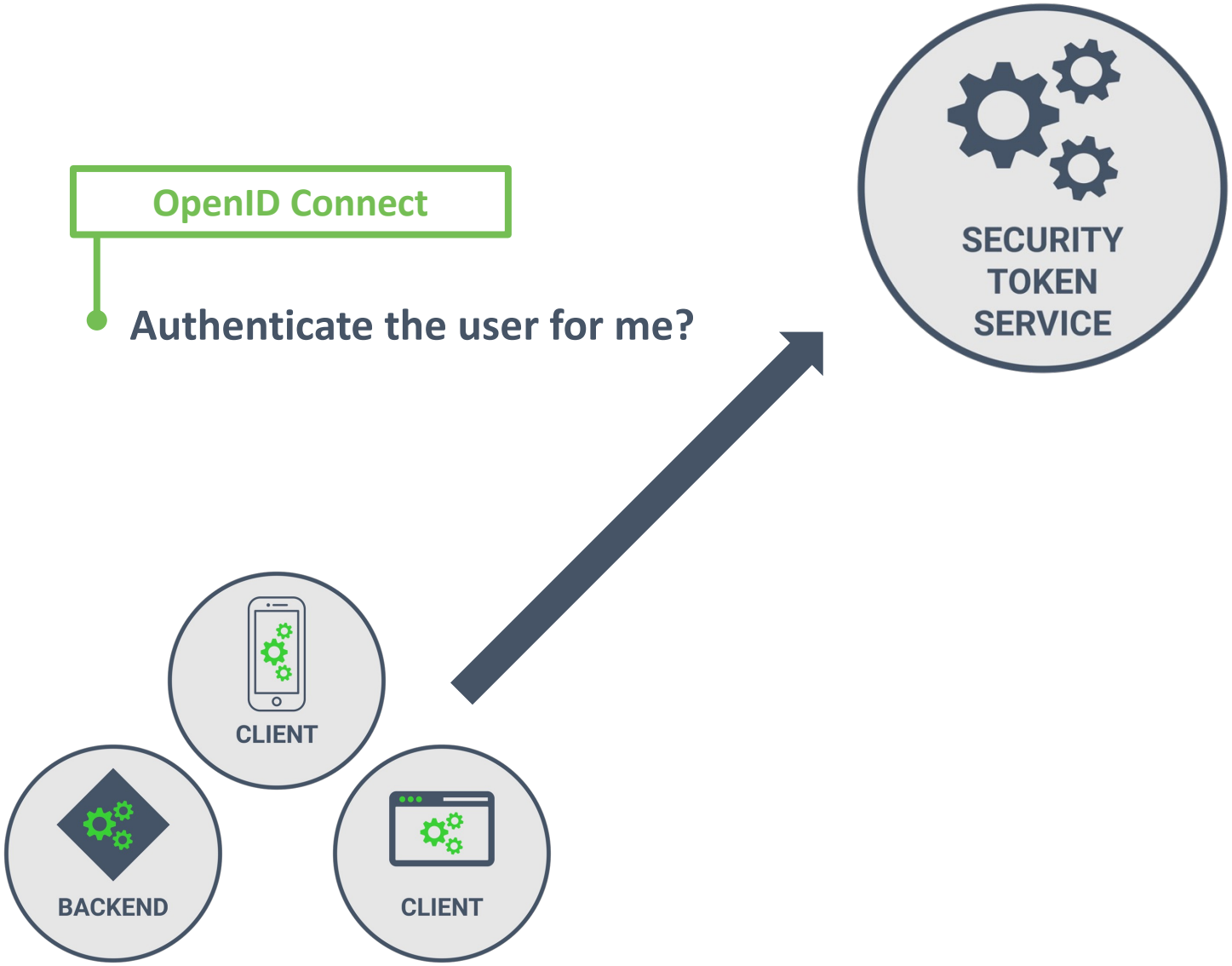
Dr. Philippe De Ryck

https://Pragmatic Web Security.com


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

Authenticate the user for me?



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

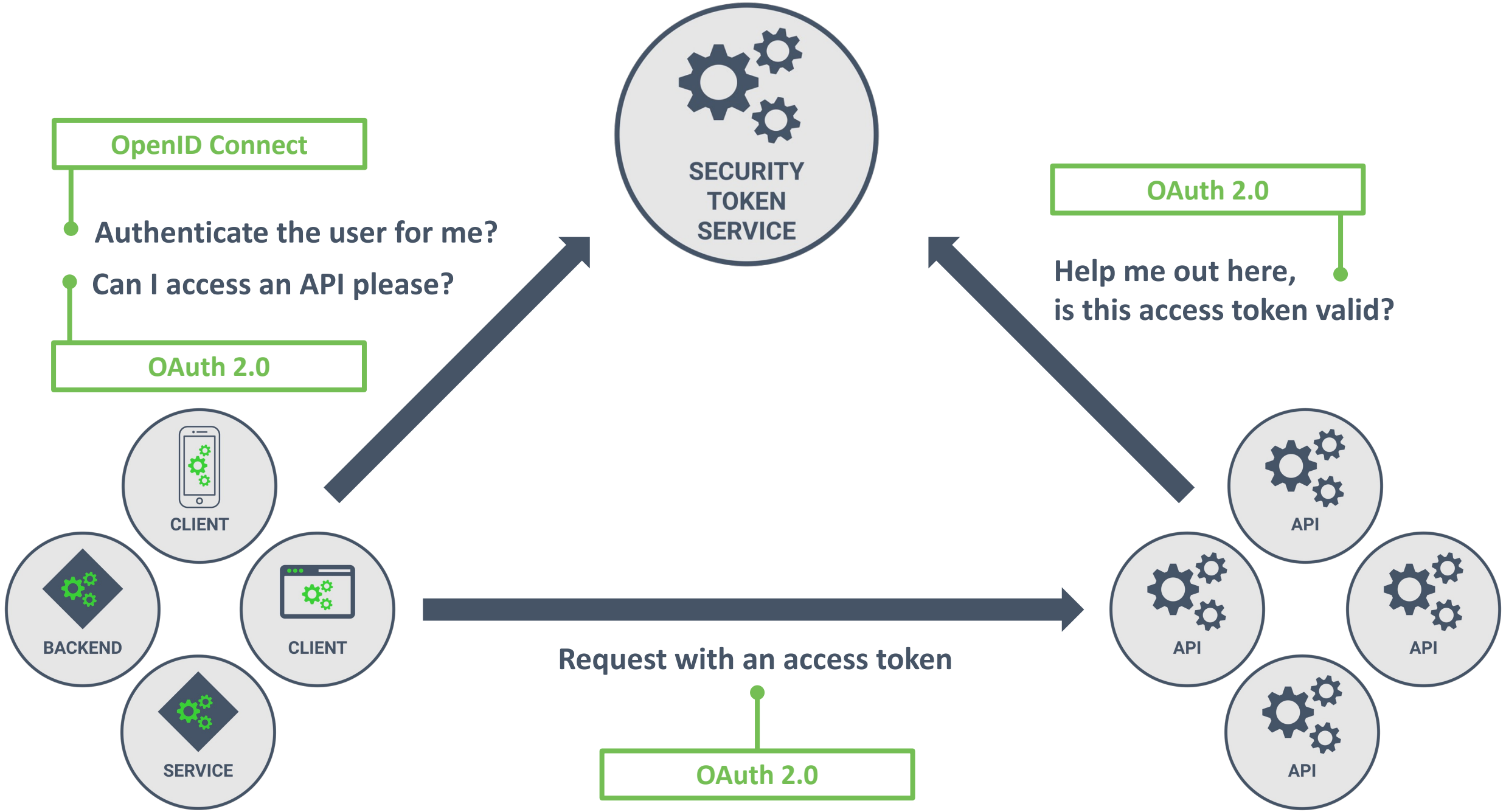
Make sure you trust Zoom

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**.

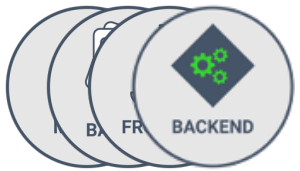
See Zoom's **Privacy Policy** and **Terms of Service**.

Cancel | Continue



@PhilippeDeRyck

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

Secure Application Development

SecAppDev organizer

I help developers with security

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

https://pragmaticwebsecurity.com

The client's perspective


SECURITY
TOKEN
SERVICE

KEYCLOAK

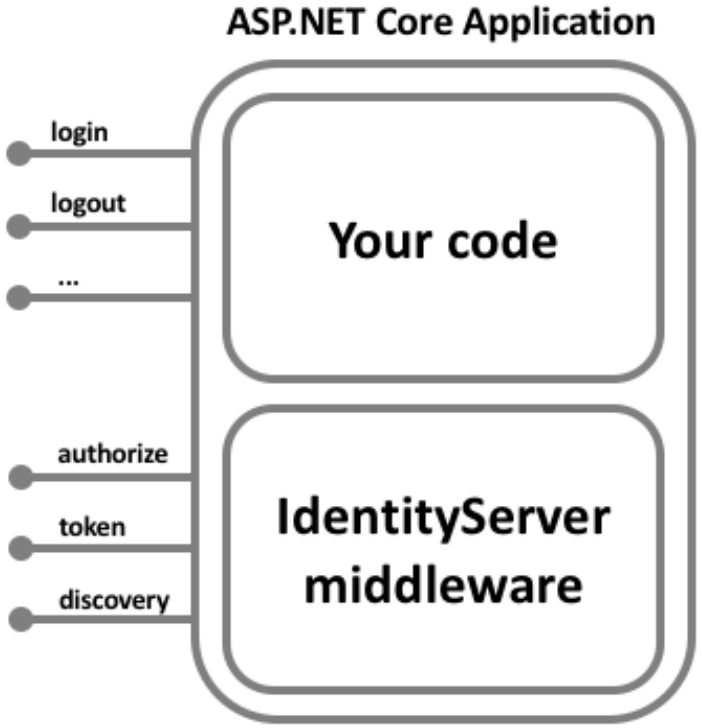
A standalone service handling OAuth 2.0, OIDC, and user management

Auth0

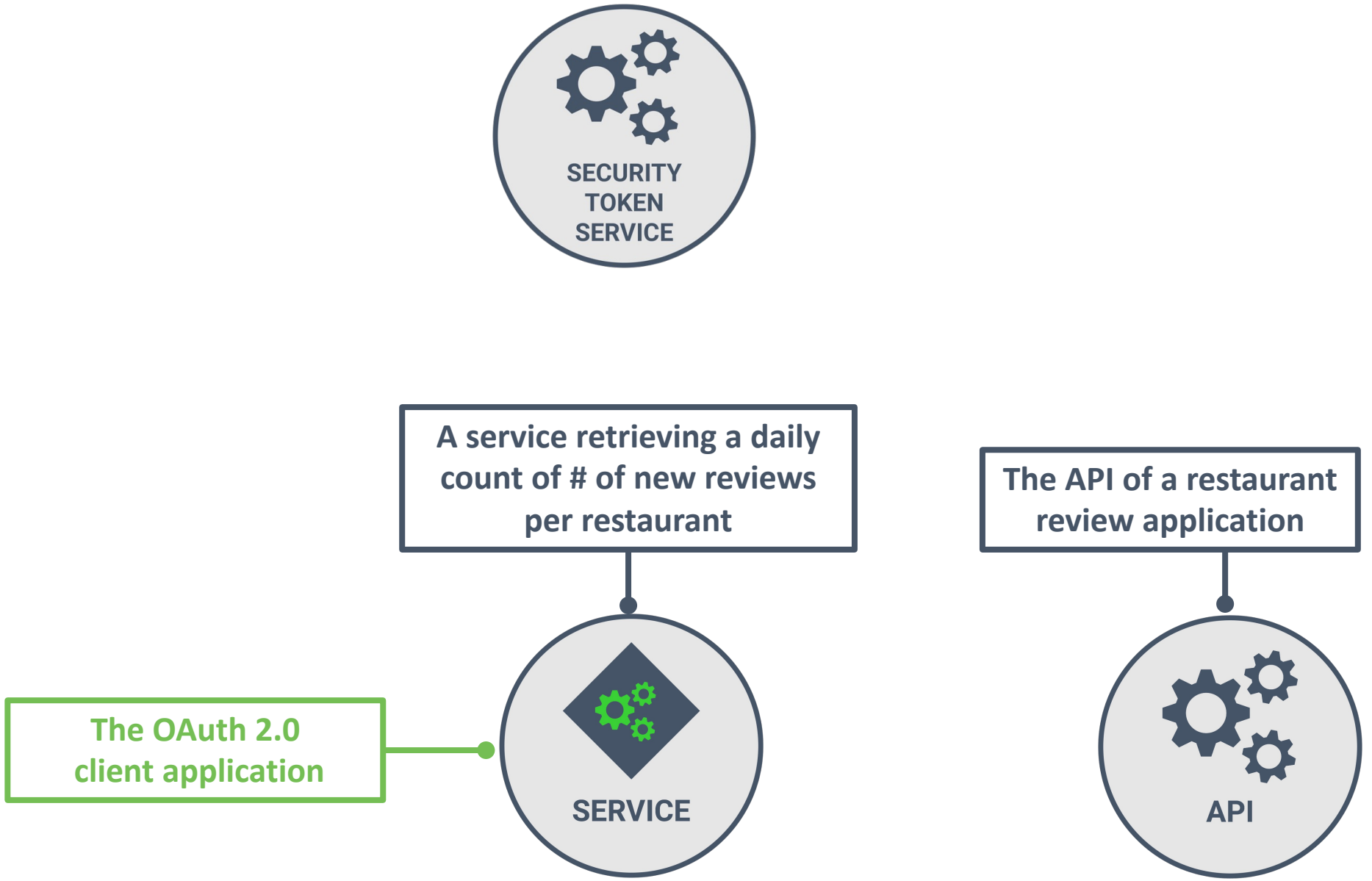
okta

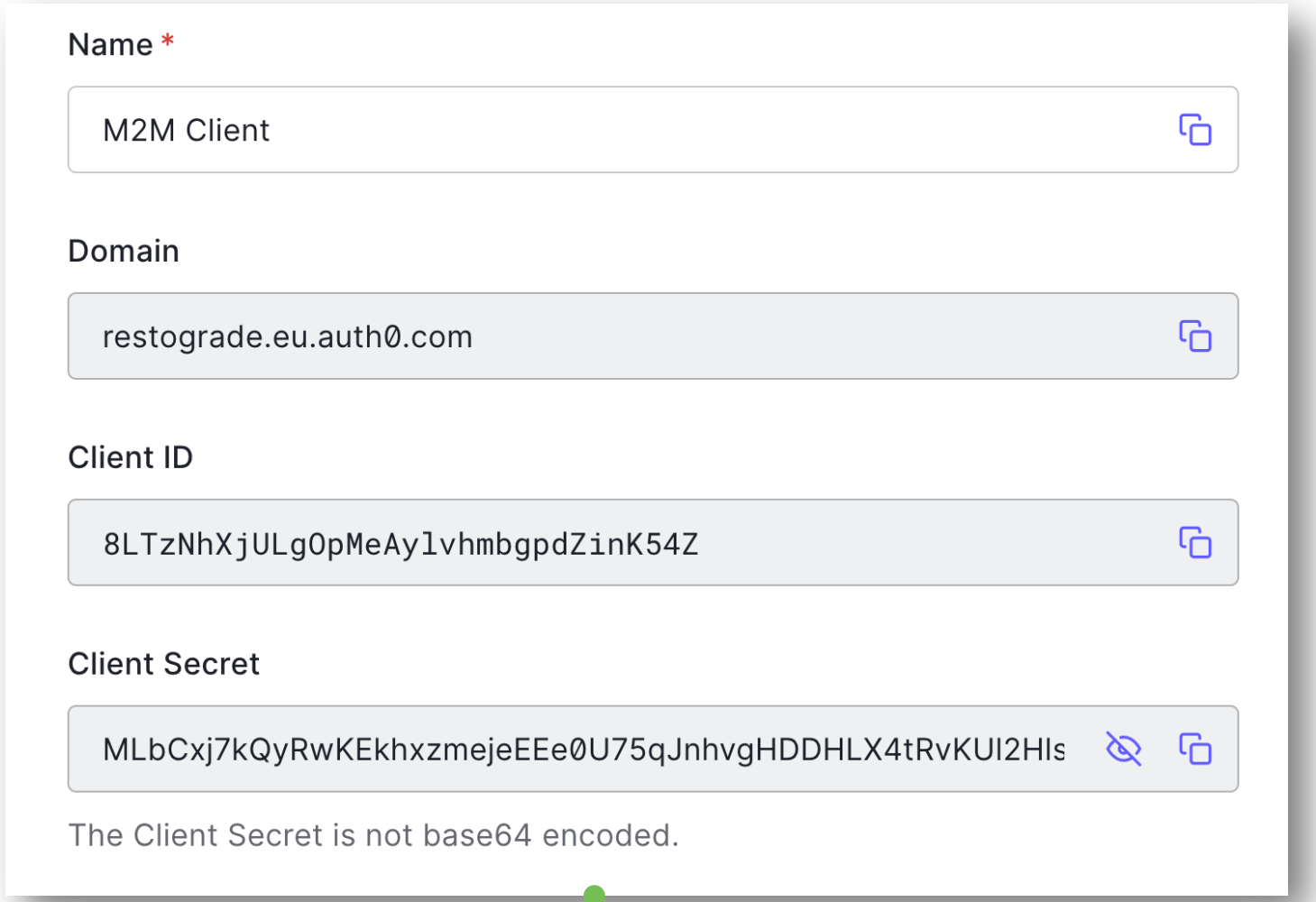
Azure Active Directory

Offload all responsibilities to "Identity as a Service"



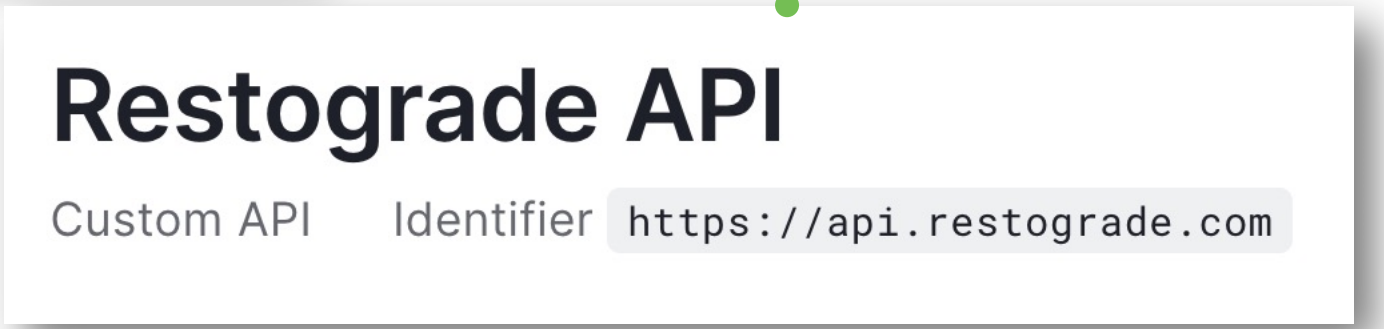
Augment your user management service with OAuth 2.0 & OIDC


SECURITY
TOKEN
SERVICE
A service retrieving a daily count of # of new reviews per restaurant
The API of a restaurant review application
The OAuth 2.0 client application
SERVICE
API

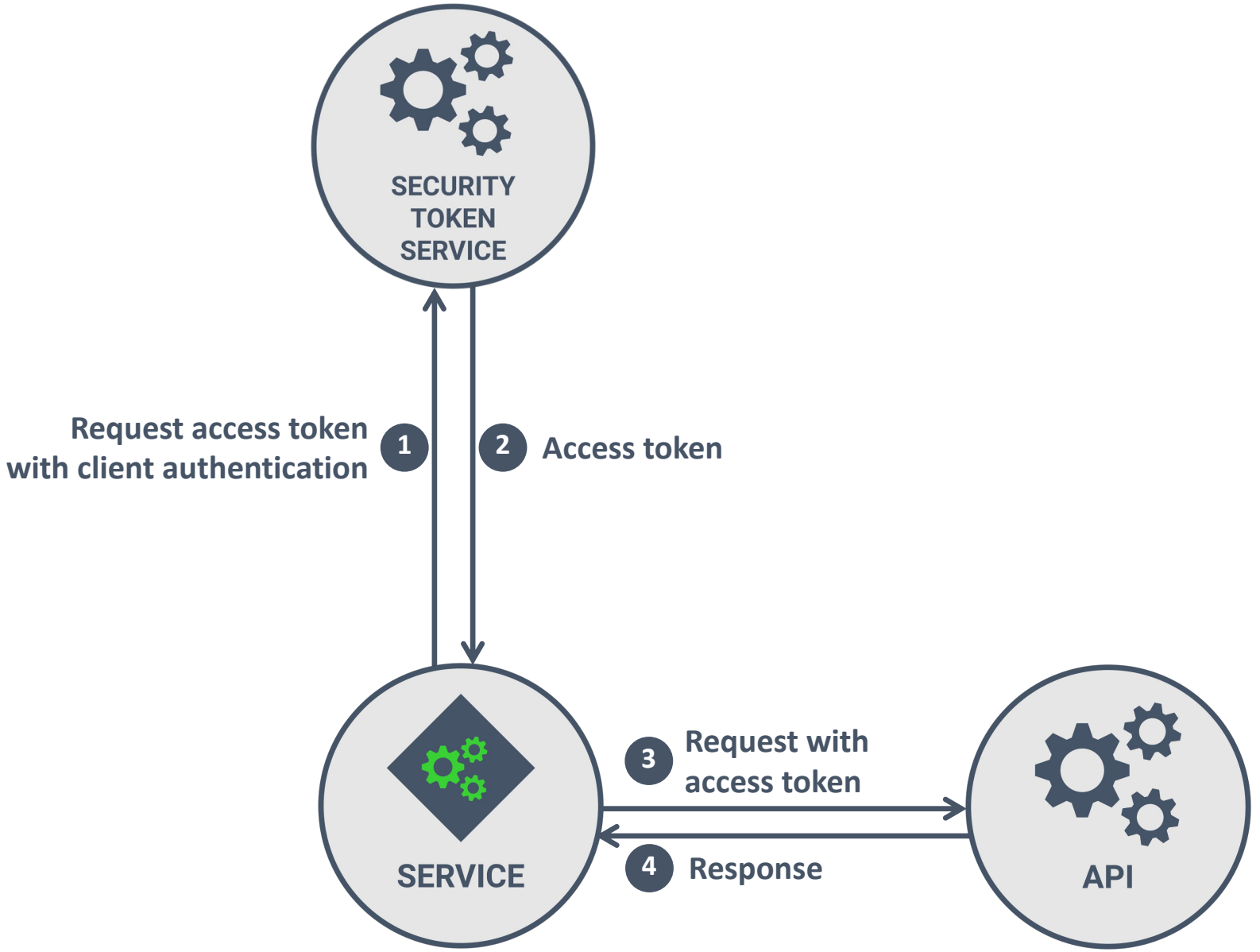


Clients are registered with the STS with an ID and a secret

APIs have a meaningful identifier and are known by the STS



Scenarios that do not involve user-based access rely on the *Client Credentials* grant



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=8LTzNhXjULgOpMeAylvhmbgpdZinK54Z
7  &client_secret=xEJRXoe…Vd_BjB
8  &audience=https://api.restograde.com
```

Indicates the *client credentials* flow

The client exchanging the code

The client needs to authenticate to the STS

Optional indication of the target API

Running the Client Credentials flow

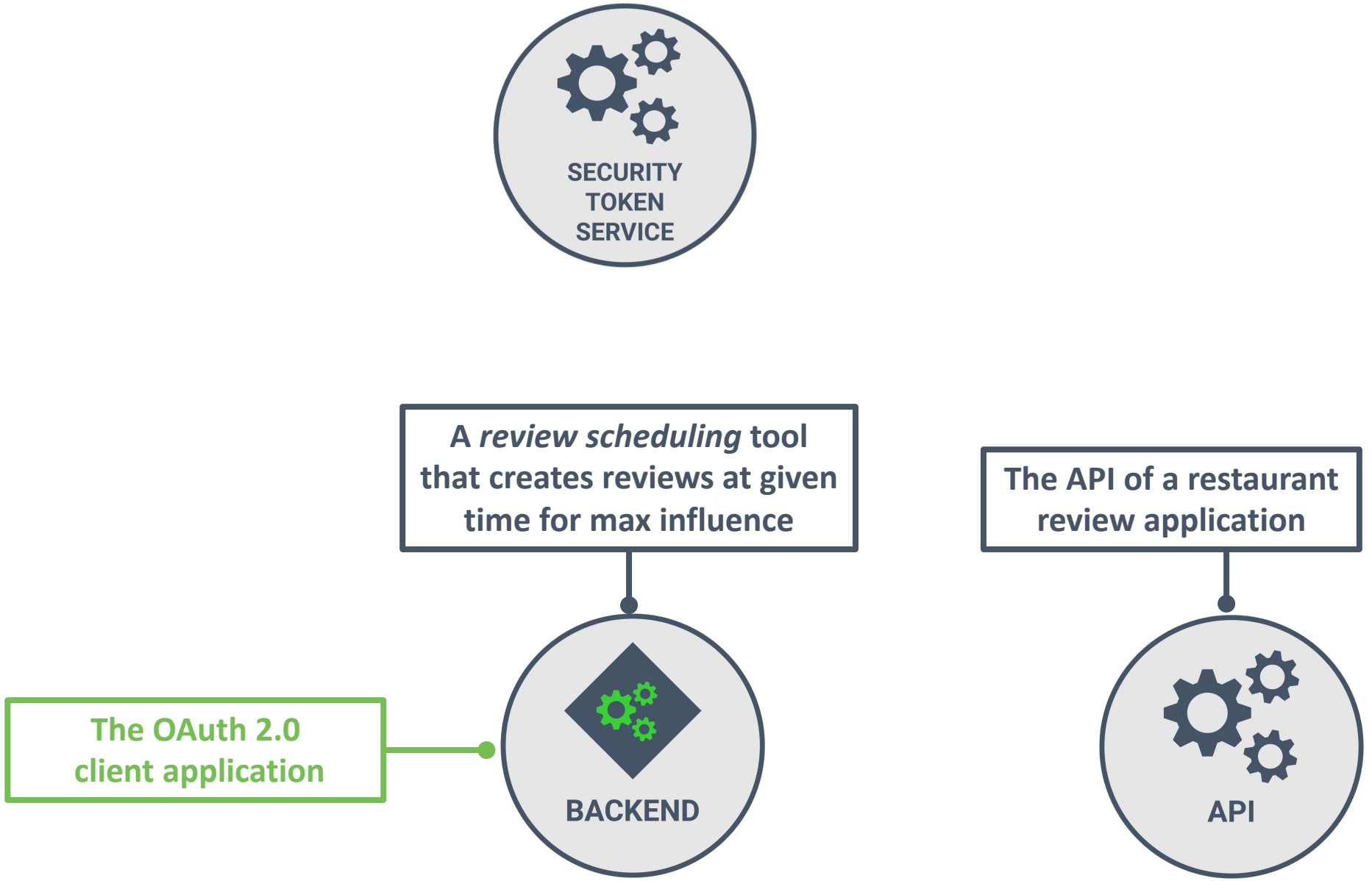
Full disclosure: I am a happy Auth0 user. I am also working closely with the Auth0 developer advocates as an Ambassador. I am not paid by Auth0, nor do I benefit from recommending Auth0 to others.

Running the Client Credentials flow

The client credentials grant enables M2M access

The client credentials grant supports direct machine-to-machine access.

The grant relies on client credentials which have to be kept in a secure location (i.e., not on an untrusted device)

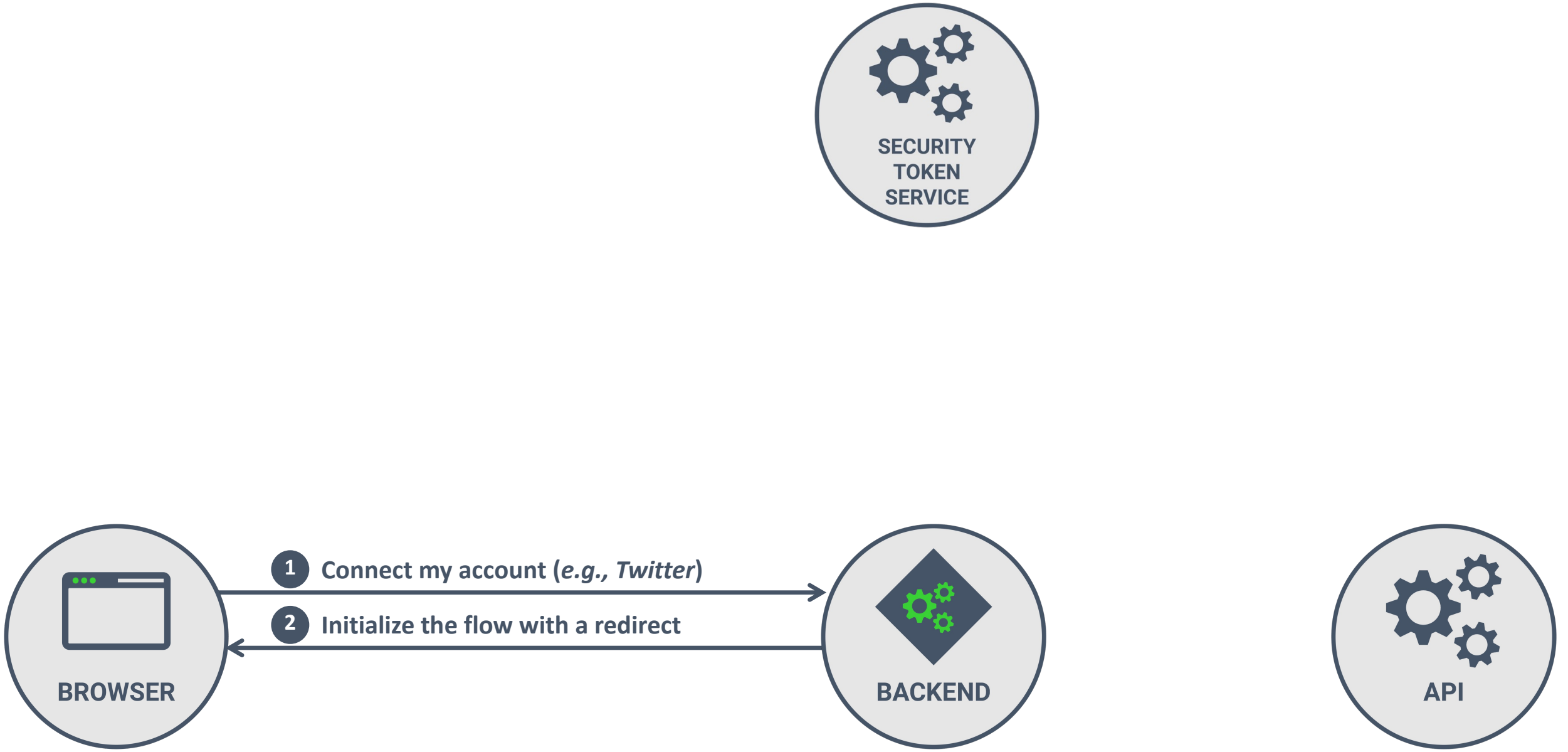

SECURITY TOKEN SERVICE
A *review scheduling* tool that creates reviews at given time for max influence
The API of a restaurant review application
The OAuth 2.0 client application
BACKEND
API

Allowed Callback URLs

https://schedule.restograde.com/callback

After the user authenticates we will only call back to any of these URLs. You can specify multiple valid URLs by comma-separating them (typically to handle different environments like QA or testing). Make sure to specify the protocol (`https://`) otherwise the callback may fail in some cases. With the exception of custom URI schemes for native clients, all callbacks should use protocol `https://`. You can use Organization URL parameters in these URLs.

The redirect URI restricts how the STS can send data through the browser to the client, preventing an attacker from hijacking valuable resources


SECURITY
TOKEN
SERVICE
1
Connect my account (e.g., Twitter)
2
Initialize the flow with a redirect
BROWSER
BACKEND
API

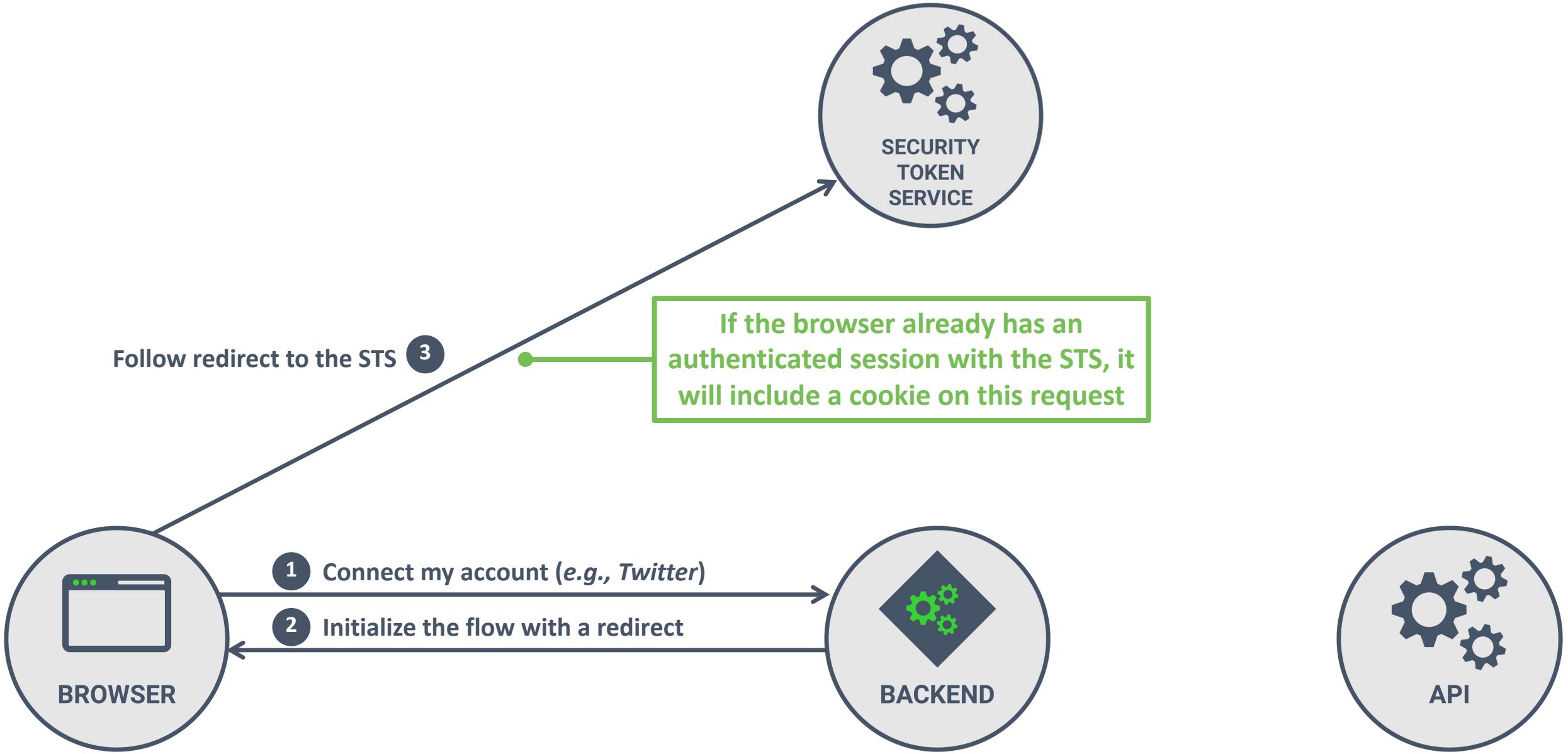
2 3 *The initialization request*

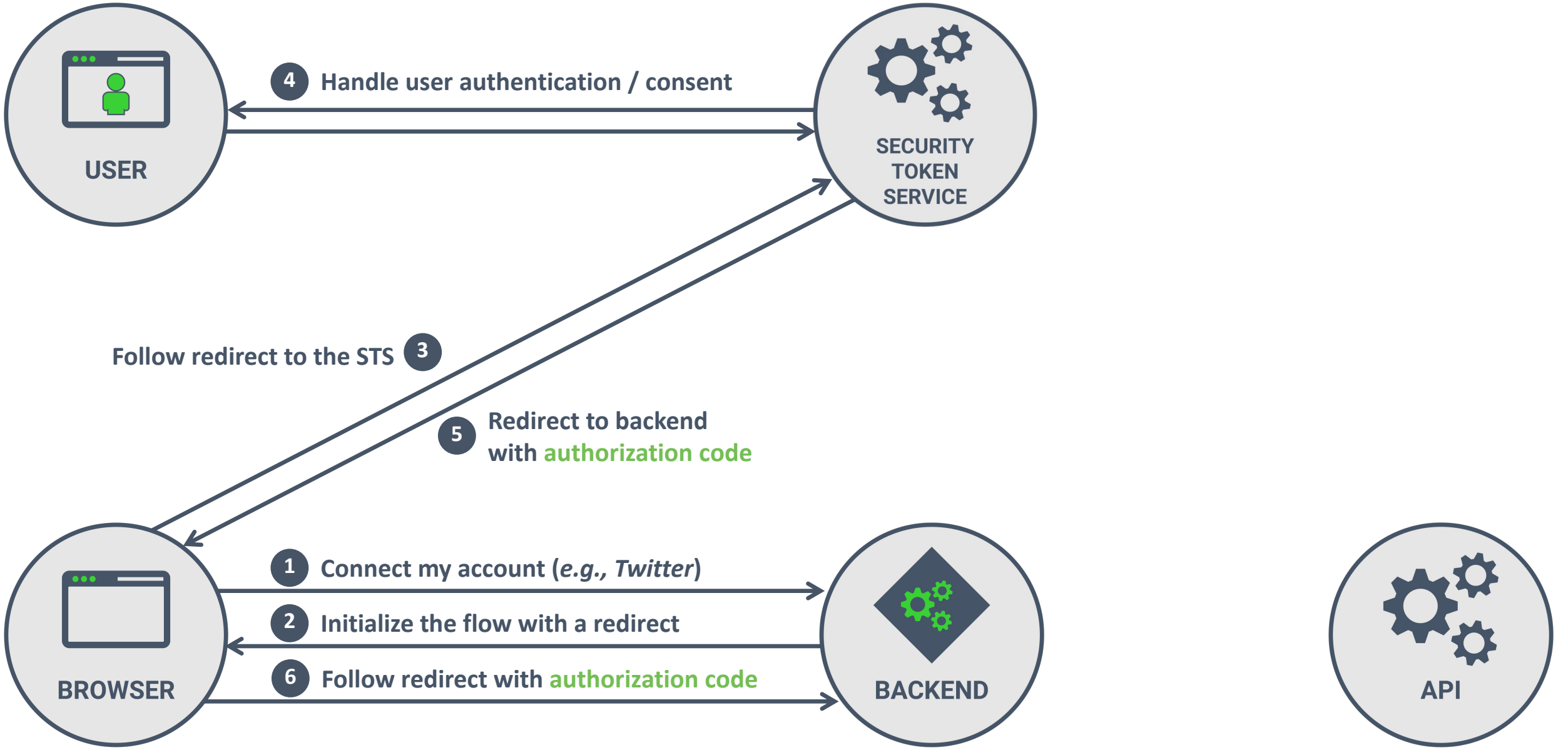
```
1  https://sts.restograde.com/authorize
2    ?response_type=code
3    &client_id=lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv
4    &scope=read
5    &redirect_uri=https://schedule.restograde.com/callback
6    &code_challenge=JhEN0Amnj7B…Wh5PxWitZYK1woWh5PxWitZY
7    &code_challenge_method=S256
```

Indicates the *authorization code flow*

The client requesting access

Where the STS should send the code


SECURITY
TOKEN
SERVICE
If the browser already has an authenticated session with the STS, it will include a cookie on this request
Follow redirect to the STS 3
1 Connect my account (e.g., Twitter)
2 Initialize the flow with a redirect
BROWSER
BACKEND
API

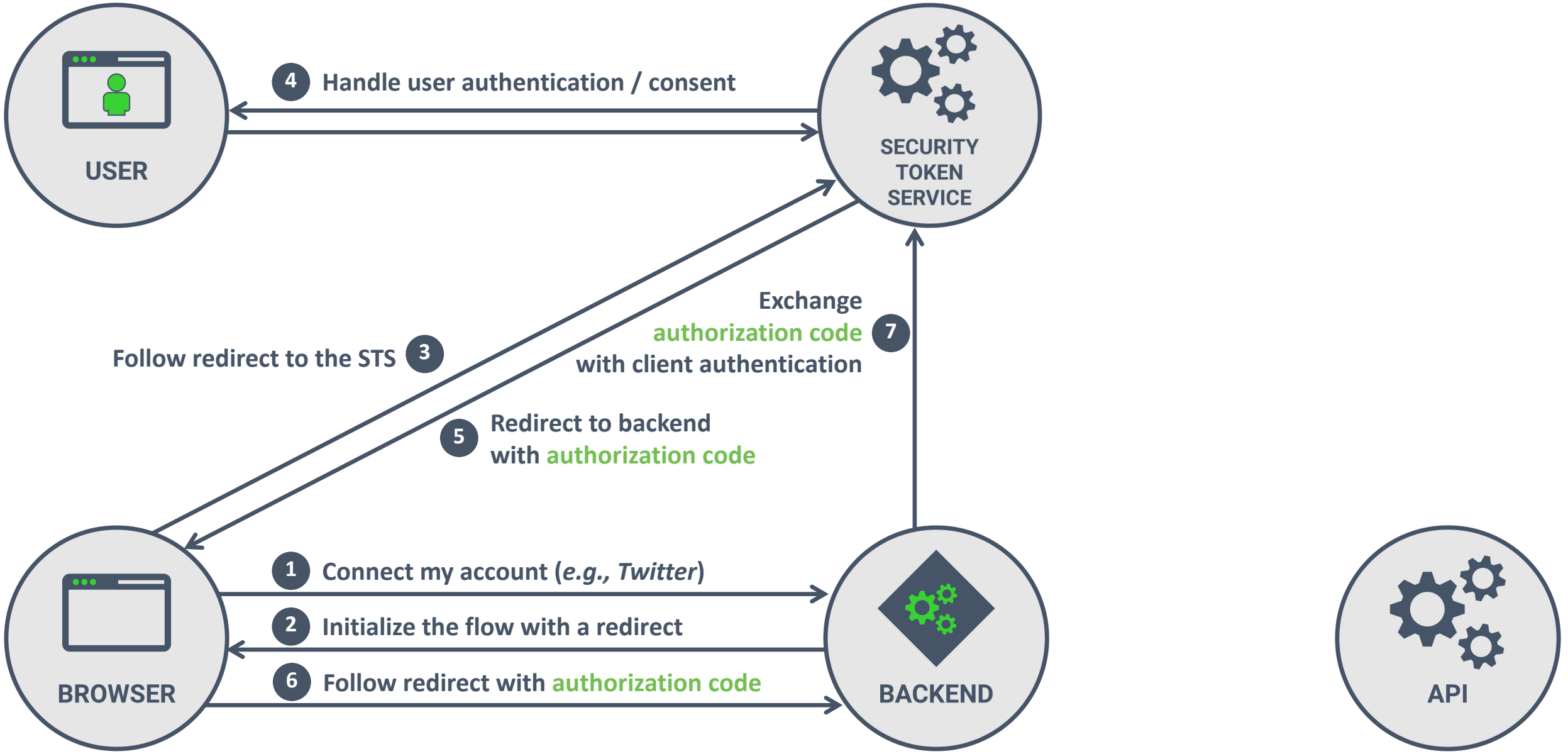

USER
4 Handle user authentication / consent
SECURITY TOKEN SERVICE
Follow redirect to the STS 3
5 Redirect to backend with authorization code
BROWSER
1 Connect my account (e.g., Twitter)
2 Initialize the flow with a redirect
6 Follow redirect with authorization code
BACKEND
API

5 6 *The callback URI*

```
1 https://schedule.restograde.com/callback
2   ?code=SplxlOBeZQQYbYS6WxSbIA
```

The callback URI from before

The authorization code


USER
4 Handle user authentication / consent
SECURITY TOKEN SERVICE
Follow redirect to the STS 3
Exchange authorization code 7 with client authentication
5 Redirect to backend with authorization code
1 Connect my account (e.g., Twitter)
2 Initialize the flow with a redirect
6 Follow redirect with authorization code
BROWSER
BACKEND
API

7 *The request to exchange the authorization code*

```
1  POST /oauth/token
2  Host: sts.restograde.com
3
4   grant_type=authorization_code
5  &client_id=lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv
7  &redirect_uri=https://schedule.restograde.com/callback
8  &code=SplxlOBeZQQYbYS6WxSbIA
9  &code_verifier=lT5q6nbPQRtdj…~IUdkErVDFG.fF4z7CzCxo
```

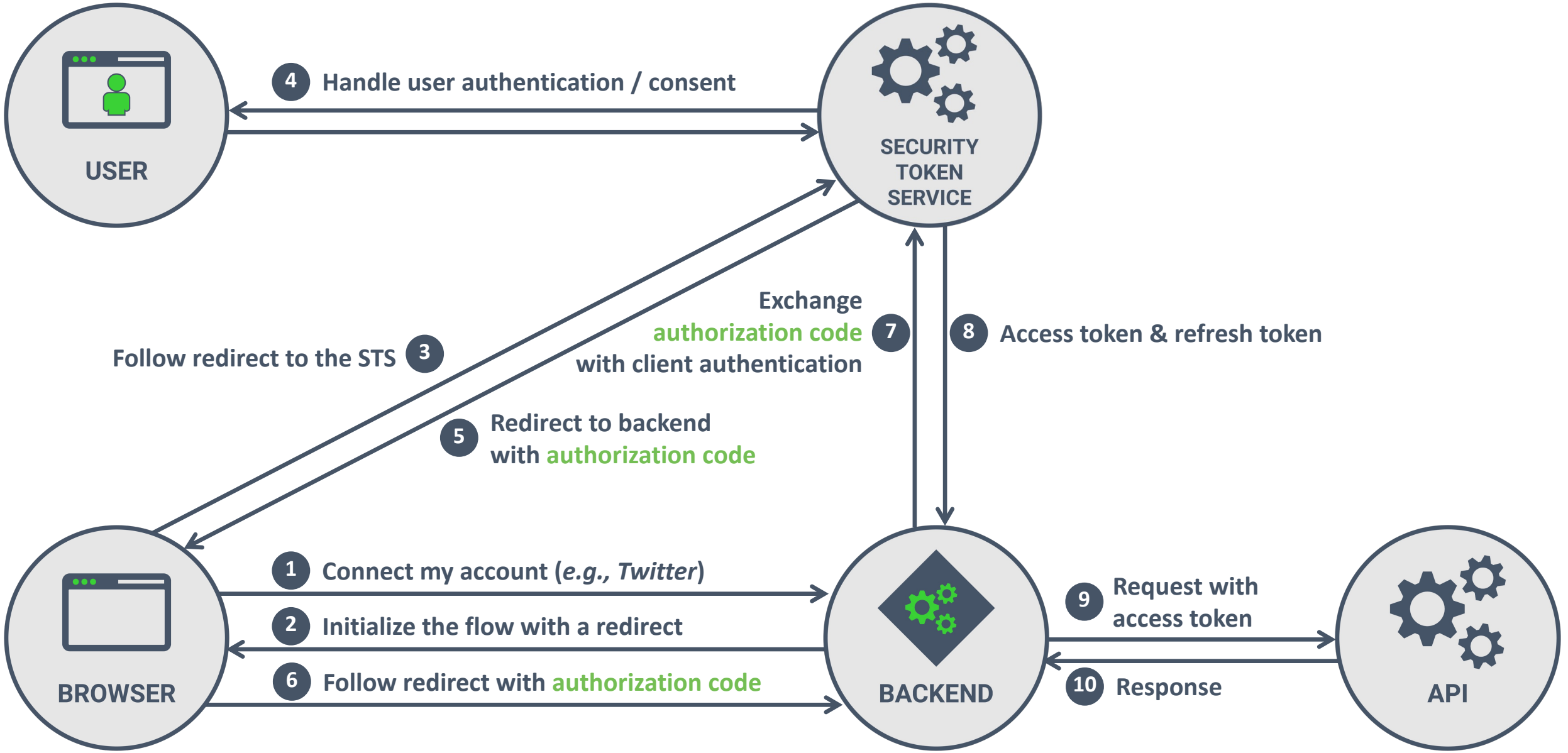
Indicates the code exchange request

The client exchanging the code

The redirect URI used before

The code received in step 6

The code verifier from step 1



@PhilippeDeRyck

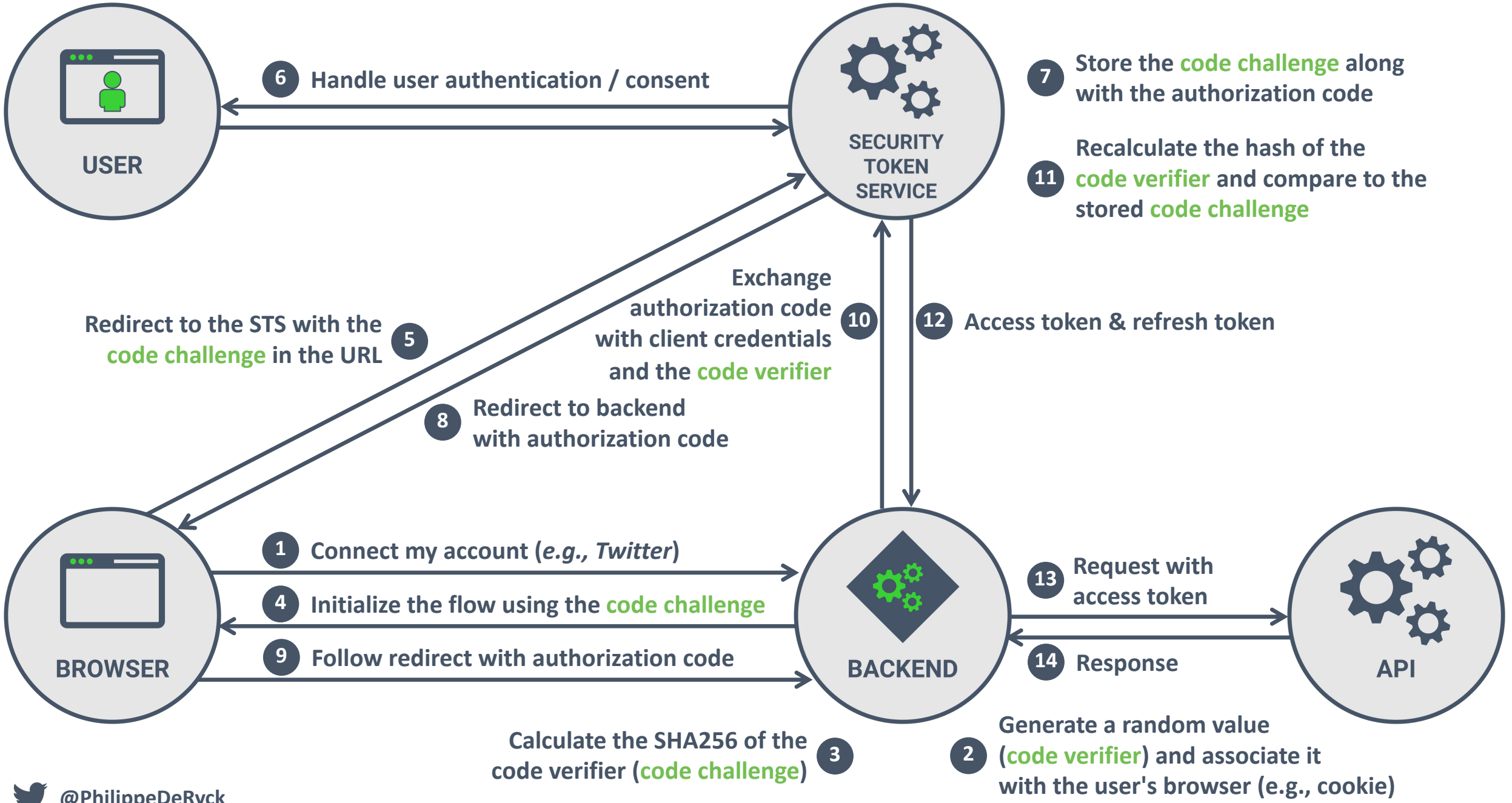
The OAuth 2.0 *Authorization Code* flow

- The client is a backend web application, running in a *secure* environment
  - The user authorizes the client to access APIs on their behalf
  - The *Authorization Code* flow gives the client an access token and refresh token
    - The access token is used by the client to access APIs on behalf of the user
    - The refresh token allows the client to obtain new access tokens, without user involvement

- The *Authorization Code* flow has several security measures built-in
  - The client needs to be registered with the proper redirect URIs
  - The client needs to authenticate on backchannel requests to the STS
    - Concretely, client authentication is needed to exhange an authorization code or refresh token
  - Authorization codes can only be used once in a very limited time window

Modern best practices require the use of PKCE

WTF is PKCE?



@PhilippeDeRyck

4 5 *The initialization request*

```
1  https://sts.restograde.com/authorize
2    ?response_type=code
3    &client_id=lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv
4    &scope=read
5    &redirect_uri=https://schedule.restograde.com/callback
6    &code_challenge=JhEN0Amnj7B…Wh5PxWitZYK1woWh5PxWitZY
7    &code_challenge_method=S256
```

- `response_type=code` — Indicates the *authorization code flow*
- `client_id=lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv` — The client requesting access
- `redirect_uri=https://schedule.restograde.com/callback` — Where the STS should send the code
- `code_challenge=JhEN0Amnj7B…Wh5PxWitZYK1woWh5PxWitZY` — The PKCE code challenge
- `code_challenge_method=S256` — The PKCE hash function

10 *The request to exchange the authorization code*

```
1  POST /oauth/token
2  Host: sts.restograde.com
3
4   grant_type=authorization_code
5  &client_id=lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv
7  &redirect_uri=https://schedule.restograde.com/callback
8  &code=SplxlOBeZQQYbYS6WxSbIA
9  &code_verifier=lT5q6nbPQRtdj…~IUdkErVDFG.fF4z7CzCxo
```

- `grant_type=authorization_code` — Indicates the code exchange request
- `&client_id=lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv` — The client exchanging the code
- `&redirect_uri=https://schedule.restograde.com/callback` — The redirect URI used before
- `&code=SplxlOBeZQQYbYS6WxSbIA` — The code received in step 9
- `&code_verifier=lT5q6nbPQRtdj…~IUdkErVDFG.fF4z7CzCxo` — The code verifier from step 1

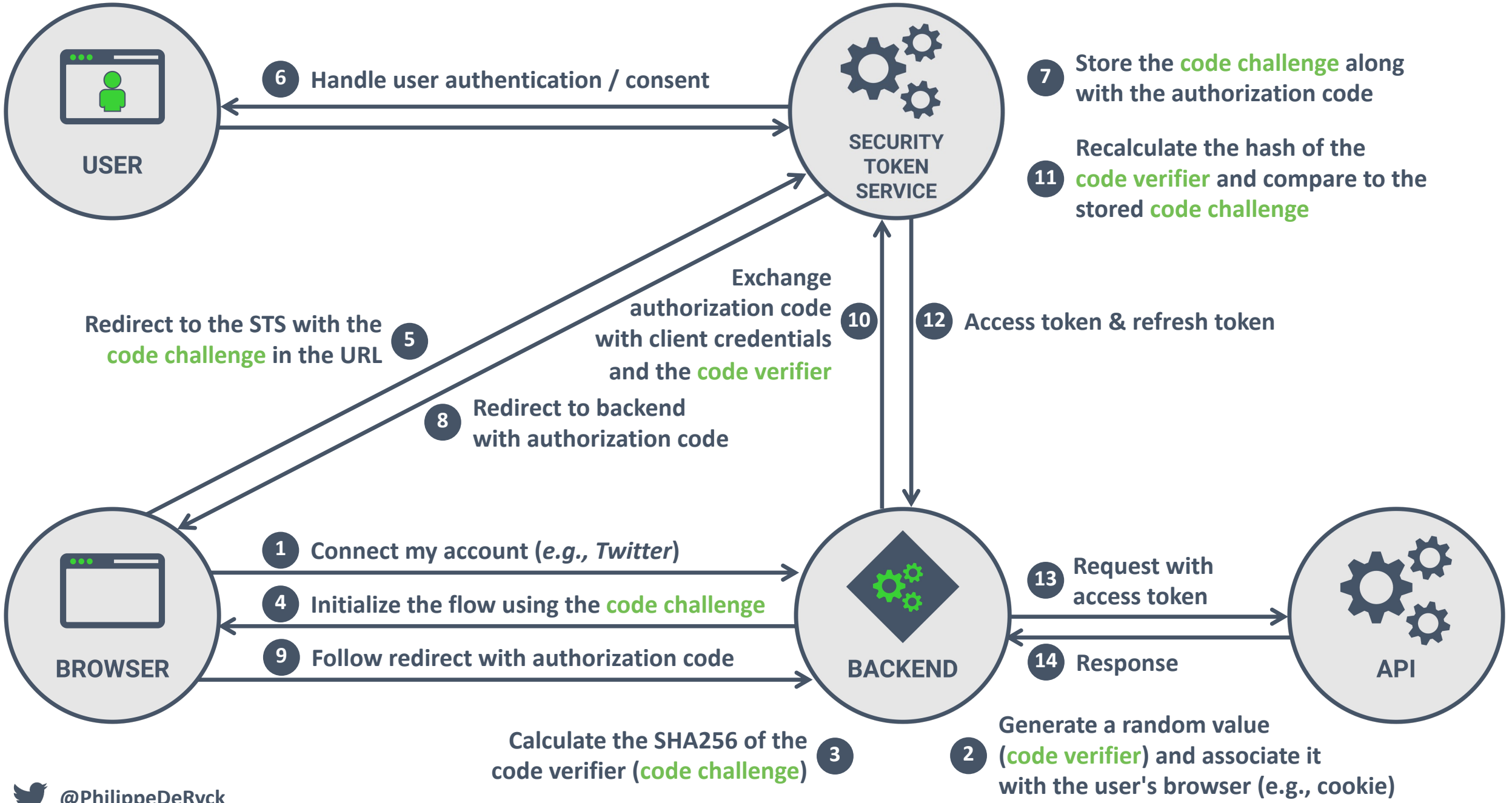
Proof Key for Code Exchange (PKCE)

- PKCE ensures that the same client intializes and finalizes the flow
  - The main use case for PKCE is to prevent authorization code theft in public clients
  - PKCE acts as a one-time password for a particular client instance

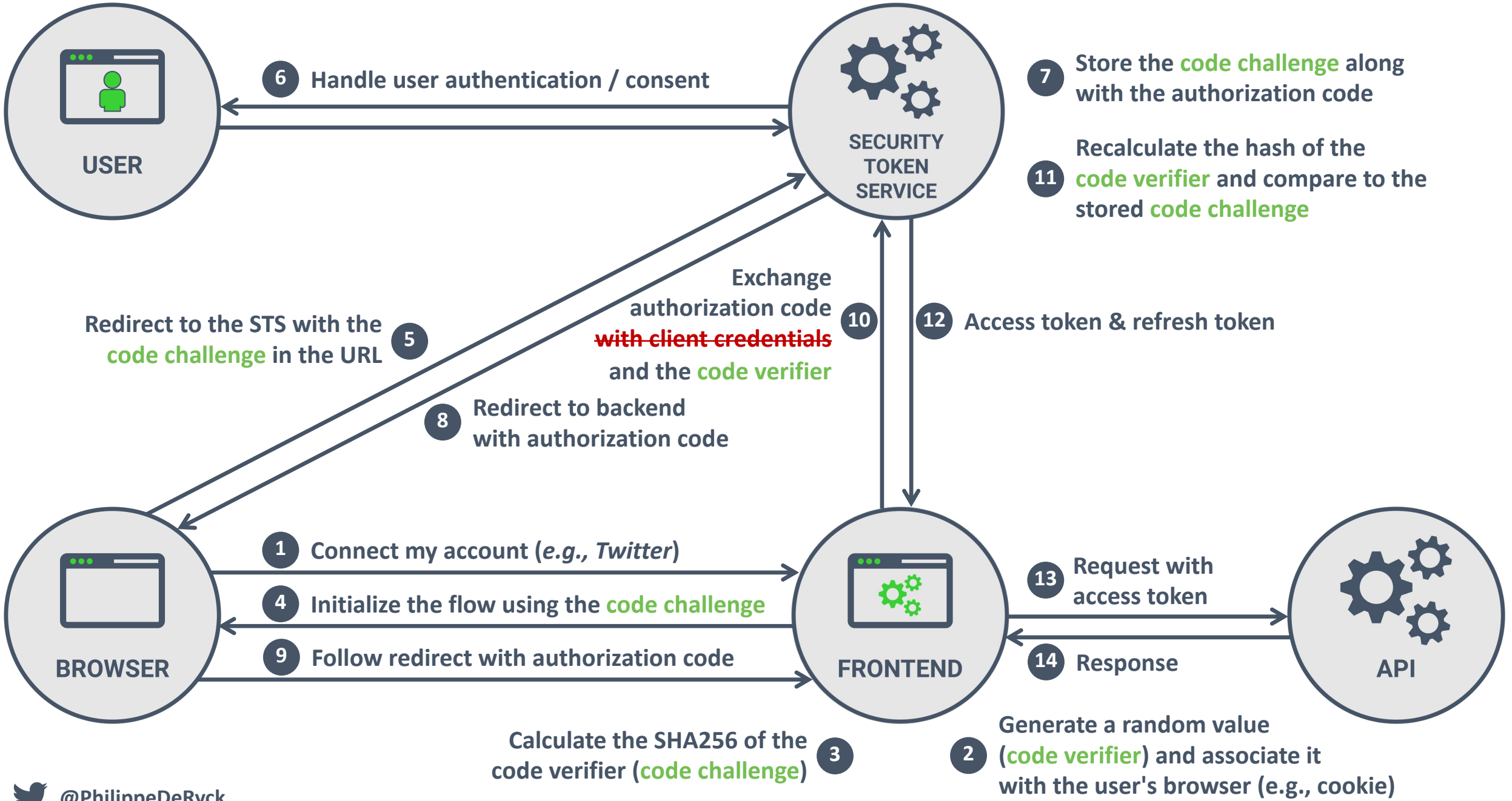
- PKCE consists of a code verifier and a code challenge
  - The code verifier is a cryptographically secure random string
    - Between 43 and 128 characters of this character set: ***[A-Z] [a-z] [0-9] - . _ ~***
  - The code challenge is a ***base64 urlencoded SHA256*** hash of the code verifier
  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

- ***PKCE is a current best practice for all types of clients***

What about frontend applications?



@PhilippeDeRyck



@PhilippeDeRyck

Running the Authorization Code flow with PKCE

THE *AUTHORIZATION CODE* FLOW WITH PKCE FOR FRONTENDS

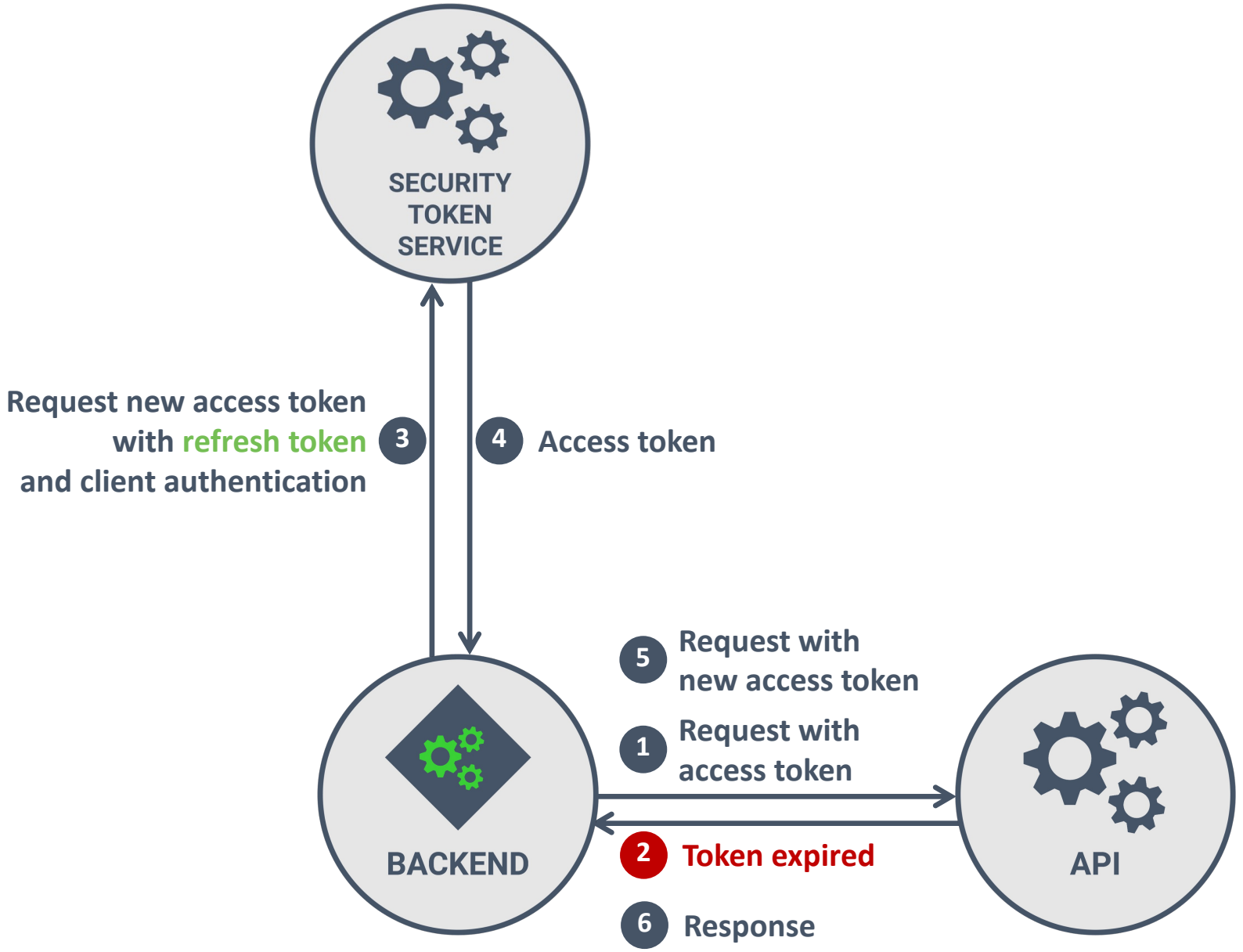
- The client is a public frontend web application, running in the user's browser
  - OAuth 2.0 enables the user to authorize the client to access APIs on their behalf
  - OIDC allows the client to obtain authentication information about the user
  - Both modern Single Page Applications as legacy JS pages can use this new flow

- Web frontends cannot rely on client authentication
  - The authorization code is exchanged without client credentials
  - The security of the flow relies on PKCE and pre-registered redirect URIs
  - Redirect URIs should use exact string matching, not dynamic matching with regexes

- Additional security requirements are needed for frontend OAuth 2.0 clients
  - Attackers succeeding in executing JS code (e.g., XSS) can fully compromise the client

The authorization code grant enables access on behalf of a user

The authorization code grant with PKCE allows the user to delegate authority to an application to access APIs on their behalf


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

Refresh tokens allow long-term access

Refresh tokens allow a client to obtain fresh access tokens without user interactions.

Refresh tokens are crucial to keep access tokens short-lived to reduce the window of abuse for stolen tokens.

The API's perspective

A reference access token

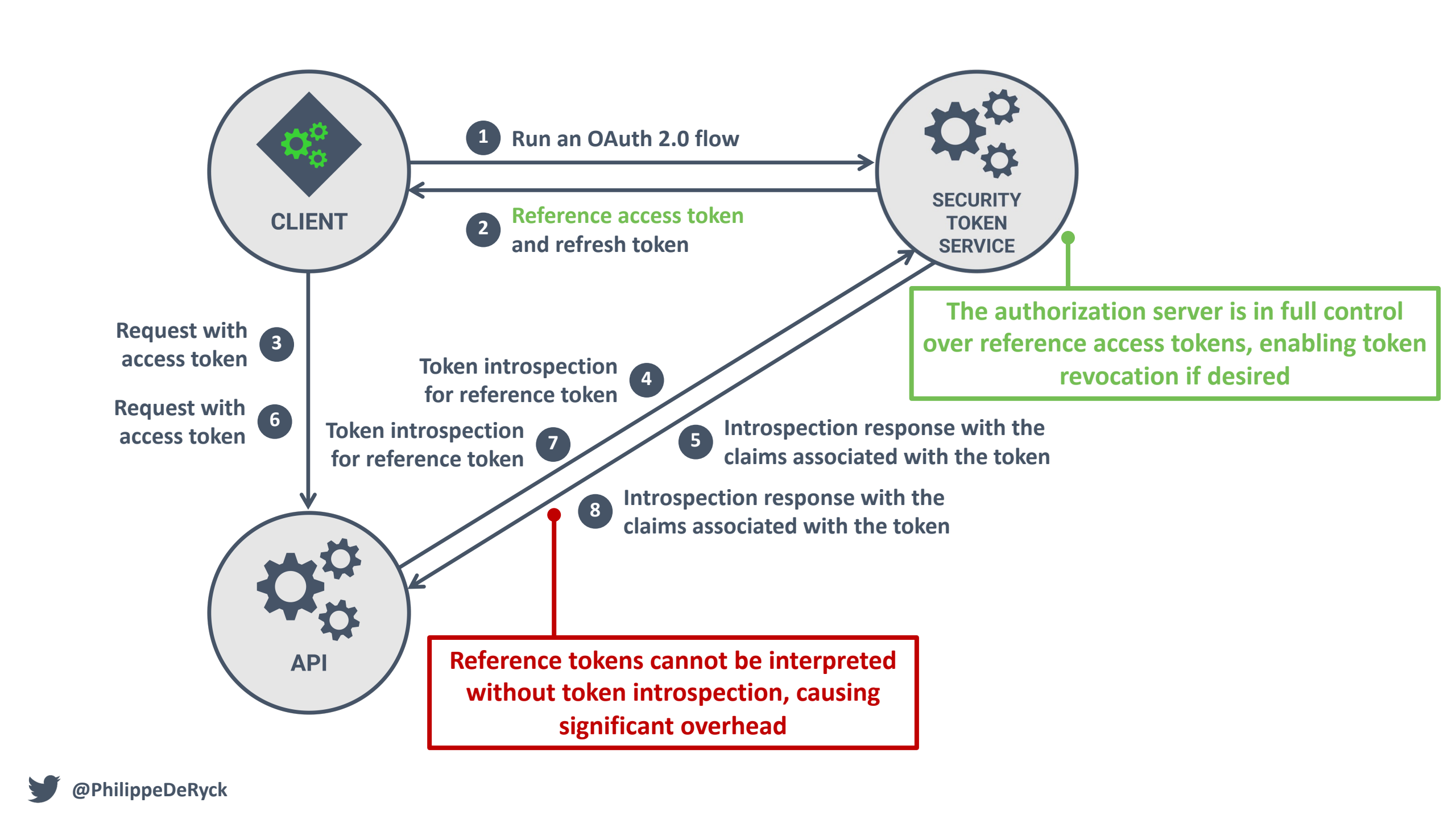
```
vSvhNDeQLqrzRbvA2eeYE2PthB1cBimS
```

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
```

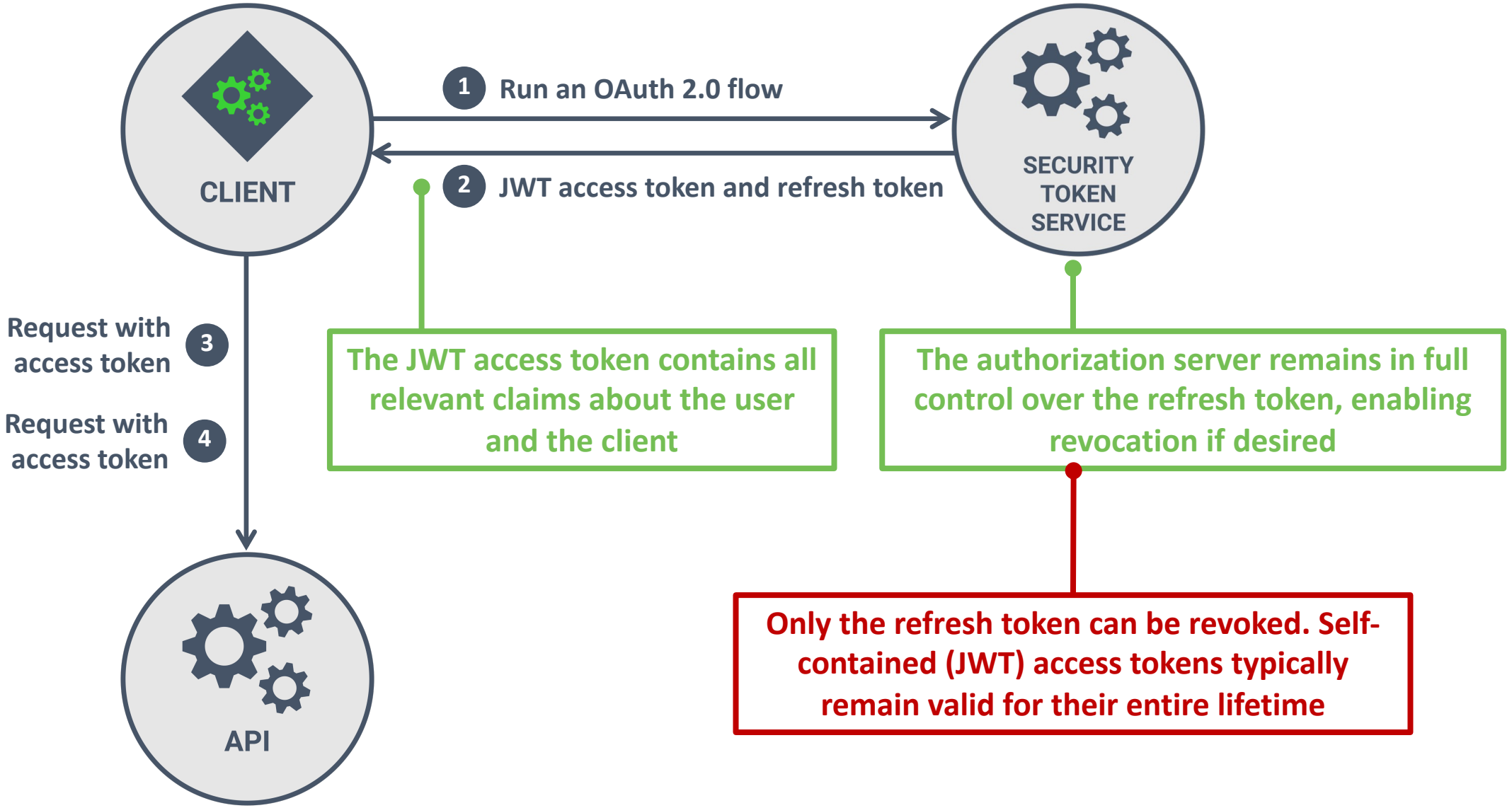
A reference access token

```
vSvhNDeQLqrzRbvA2eeYE2PthB1cBimS
```


CLIENT
SECURITY TOKEN SERVICE
API
1 Run an OAuth 2.0 flow
2 Reference access token and refresh token
3 Request with access token
4 Token introspection for reference token
5 Introspection response with the claims associated with the token
6 Request with access token
7 Token introspection for reference token
8 Introspection response with the claims associated with the token
The authorization server is in full control over reference access tokens, enabling token revocation if desired
Reference tokens cannot be interpreted without token introspection, causing significant overhead

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
```


CLIENT
1 Run an OAuth 2.0 flow
2 JWT access token and refresh token
SECURITY TOKEN SERVICE
Request with access token 3
Request with access token 4
API
The JWT access token contains all relevant claims about the user and the client
The authorization server remains in full control over the refresh token, enabling revocation if desired
Only the refresh token can be revoked. Self-contained (JWT) access tokens typically remain valid for their entire lifetime

Revocation vs performance

Token security is often a trade-off between performance and security. Short-lived self-contained access tokens typically offer a good balance

The claims obtained via token introspection

```
{
  "active": true,
  "iss": "https://sts.restograde.com",
  "aud": "https://api.restograde.com",
  "sub": "5eb916c258bdb50bf20366c6",
  "client_id": "OLKNn389SU…ilLSgdYxwEp",
  "scope": "reviews:read reviews:write"
}
```

The payload of a JWT-based access token

```
{
  "iss": "https://sts.restograde.com",
  "aud": "https://api.restograde.com",
  "sub": "5eb916c258bdb50bf20366c6",
  "exp": 1589861472,
  "azp": "OLKNn389SUufExgRG1RilLSgdYxwEp",
  "scope": "reviews:read reviews:write"
}
```

The claims obtained via token introspection

```
{
  "active": true,
  "iss": "https://sts.restograde.com",
  "aud": "https://api.restograde.com",
  "sub": "5eb916c258bdb50bf20366c6",
  "client_id": "OLKNn389SU…ilLSgdYxwEp",
  "scope": "reviews:read reviews:write"
}
```

Token introspection responses contain an *active* claim

The payload of a JWT-based access token

```
{
  "iss": "https://sts.restograde.com",
  "aud": "https://api.restograde.com",
  "sub": "5eb916c258bdb50bf20366c6",
  "exp": 1589861472,
  "azp": "OLKNn389SUufExgRG1RilLSgdYxwEp",
  "scope": "reviews:read reviews:write"
}
```

JWT tokens include an *expiration* timestamp (and optionally *issued at (iat)* / *not before (nbf)*)

The claims obtained via token introspection

```
{
  "active": true,
  "iss": "https://sts.restograde.com",
  "aud": "https://api.restograde.com",
  "sub": "5eb916c258bdb50bf20366c6",
  "client_id": "OLKNn389SU…ilLSgdYxwEp",
  "scope": "reviews:read reviews:write"
}
```

The payload of a JWT-based access token

```
{
  "iss": "https://sts.restograde.com",
  "aud": "https://api.restograde.com",
  "sub": "5eb916c258bdb50bf20366c6",
  "exp": 1589861472,
  "azp": "OLKNn389SUufExgRG1RilLSgdYxwEp",
  "scope": "reviews:read reviews:write"
}
```

The *iss* claim indicates which service issued the token.

The *aud* claim indicates which API is supposed to consume the token.

The claims obtained via token introspection

```
{
  "active": true,
  "iss": "https://sts.restograde.com",
  "aud": "https://api.restograde.com",
  "sub": "5eb916c258bdb50bf20366c6",
  "client_id": "OLKNn389SU…ilLSgdYxwEp",
  "scope": "reviews:read reviews:write"
}
```

The payload of a JWT-based access token

```
{
  "iss": "https://sts.restograde.com",
  "aud": "https://api.restograde.com",
  "sub": "5eb916c258bdb50bf20366c6",
  "exp": 1589861472,
  "azp": "OLKNn389SUufExgRG1RilLSgdYxwEp",
  "scope": "reviews:read reviews:write"
}
```

The client_id/*azp* claim indicates which client is authorized to use the token

Make sure the access token is acceptable

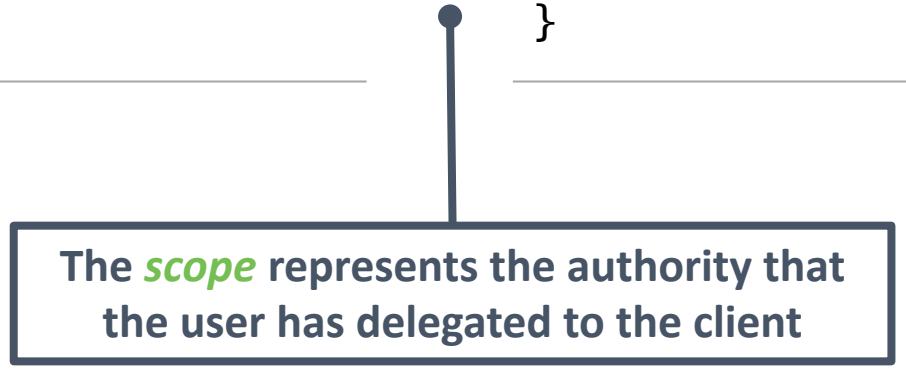
Access tokens have various claims that describe the token metadata (e.g., issuer, audience). Make sure these values make sense to your API.

The claims obtained via token introspection

```
{
  "active": true,
  "iss": "https://sts.restograde.com",
  "aud": "https://api.restograde.com",
  "sub": "5eb916c258bdb50bf20366c6",
  "azp": "OLKNn389SUufExgRG1RilLSgdYxwEp",
  "scope": "reviews:read reviews:write"
}
```

The payload of a JWT-based access token

```
{
  "iss": "https://sts.restograde.com",
  "aud": "https://api.restograde.com",
  "sub": "5eb916c258bdb50bf20366c6",
  "exp": 1589861472,
  "azp": "OLKNn389SUufExgRG1RilLSgdYxwEp",
  "scope": "reviews:read reviews:write"
}
```



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**.

Cancel | Continue

@PhilippeDeRyck

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

@PhilippeDeRyck

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: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.

@PhilippeDeRyck

USE SCOPES FOR FUNCTION-LEVEL ACCESS CONTROL

Scopes define the authority to perform certain operations.

The API can rely on the presence of a certain scope in the access token claims for authorization purposes.

The use of custom permission claims

```
{
  "iss": "https://sts.restograde.com",
  "aud": "https://api.restograde.com",
  "sub": "5eb916c258bdb50bf20366c6",
  "azp": "OLKNn389SUufExgRG1RilLSgdYxwEp",
  "permissions": ["reviews:fullaccess"]
}
```

Permissions are typically used in a first-party scenario, where the authorization server enforces a specific authorization policy

OAuth 2.0 supports the use of custom claims in access tokens. A *permissions* claim is quite common to include concrete user or client permissions in a token

Custom permission claims offer more flexibility

When the entire ecosystem is tightly controlled, the access token often includes client/user-specific permissions instead of coarse-grained scopes.

A user-specific access token

```
{
  "iss": "https://sts.restograde.com",
  "aud": "https://api.restograde.com",
  "sub": "5eb916c258bdb50bf20366c6",
  "azp": "OLKNn389SUufExgRG1RilLSgdYxwEp",
  "permissions": ["reviews:read"]
}
```

The sub claim contains the unique identifier of the user associated with the access token

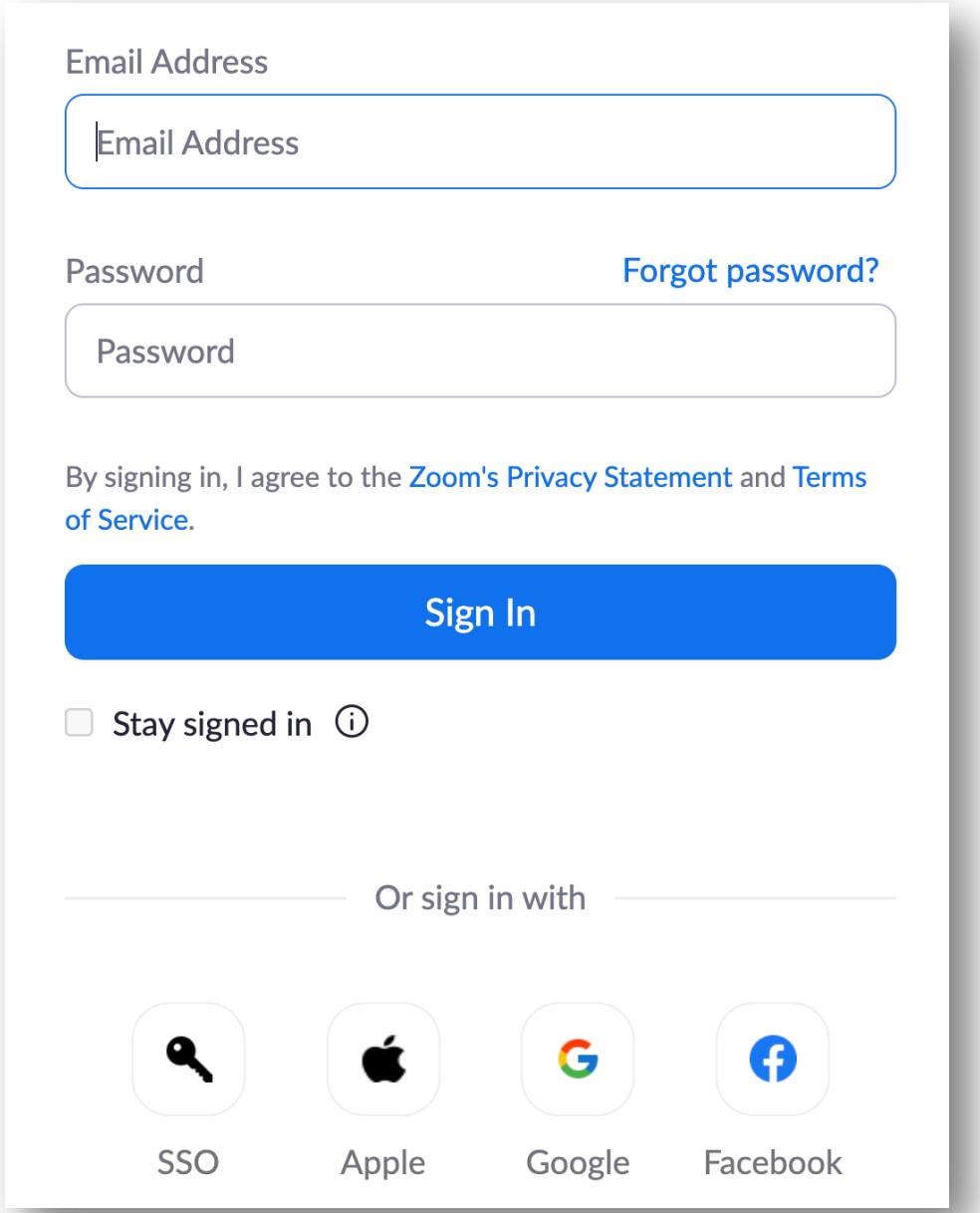
Use the user's identifier to make access control decisions (e.g., author of the review) or to collect additional information for more advanced authorization decisions

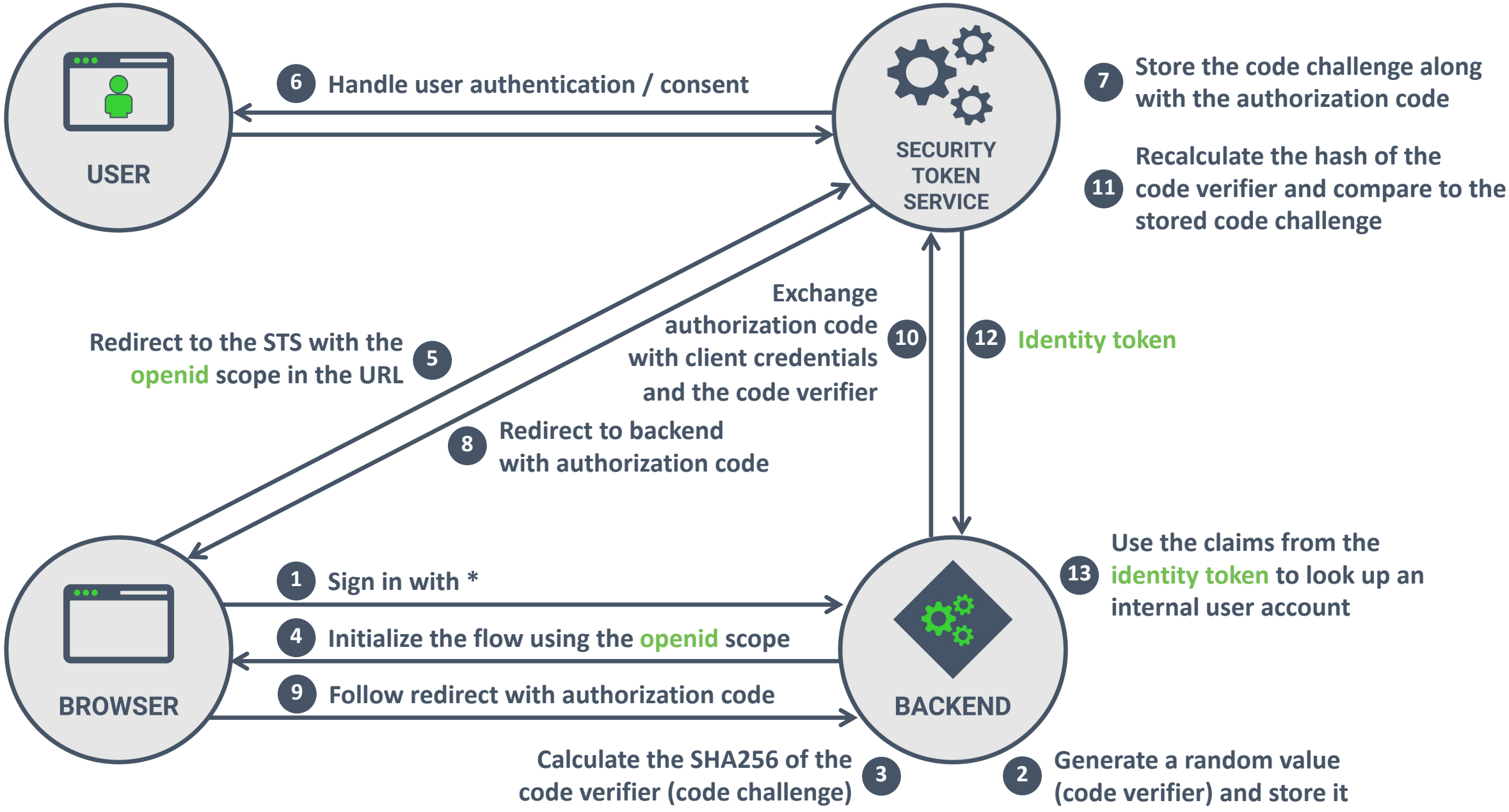
The user has permission to read reviews, but the permission does not specify ***which*** reviews

Perform object-level authorization checks

Broken Object-Level Authorization is the #1 API security failure. Use the* sub *claim to establish the user's identity and make sure the user is allowed to perform the requested operation on the specified object.

User authentication with OIDC


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



@PhilippeDeRyck

The application'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 application database

The identity token payload

```
1  {
2    "iss": "https://sts.restograde.com/",
3    "sub": "auth0|5eb916c258bdb50bf20366c6",
4    "aud": "FN983CEYgx4mdUg3NKNKHjwfNAL5Fb42",
5    "iat": 1591676290,
6    "exp": 1591712290
7  }
```

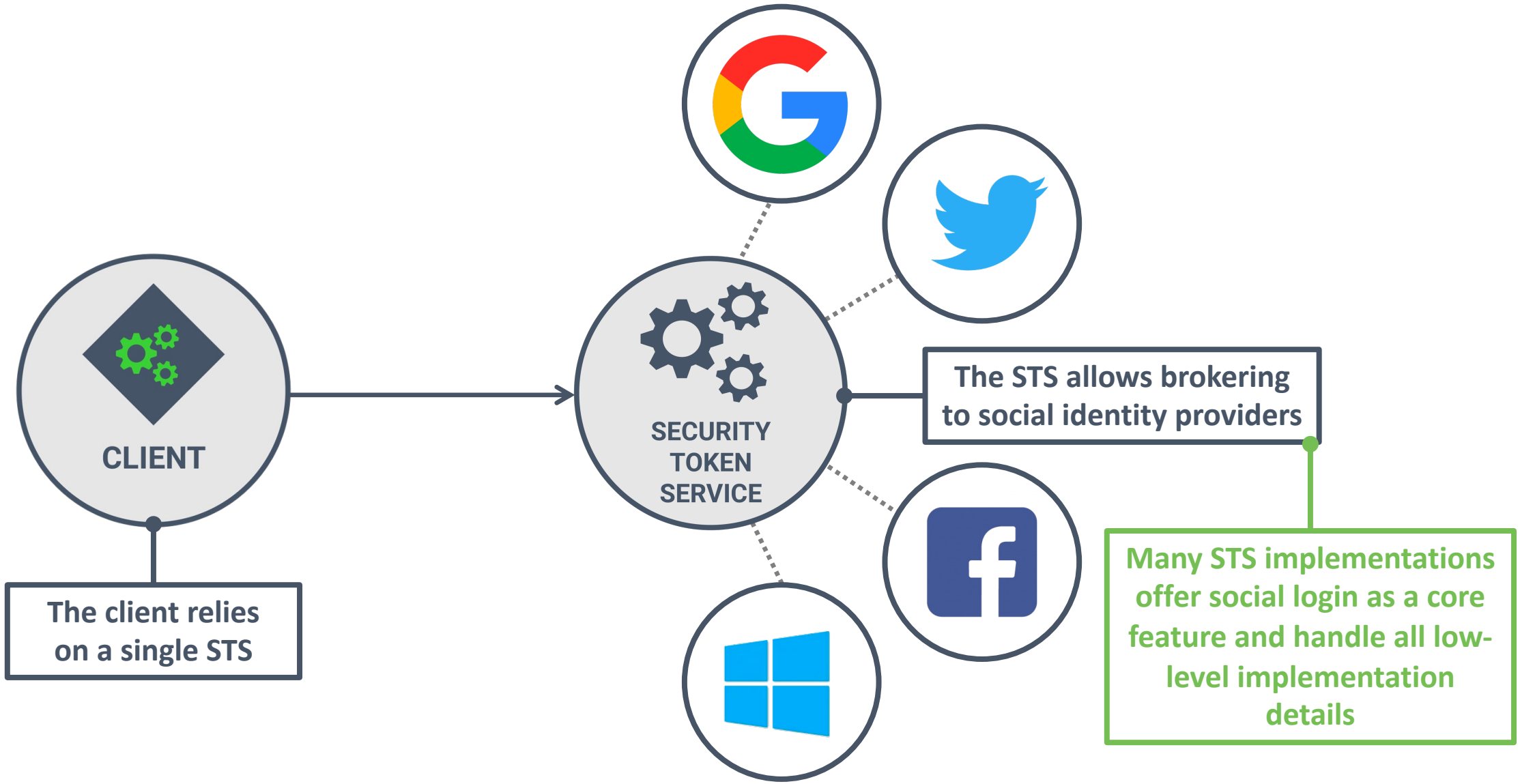
The *sub* claim is guaranteed to be unique and immutable

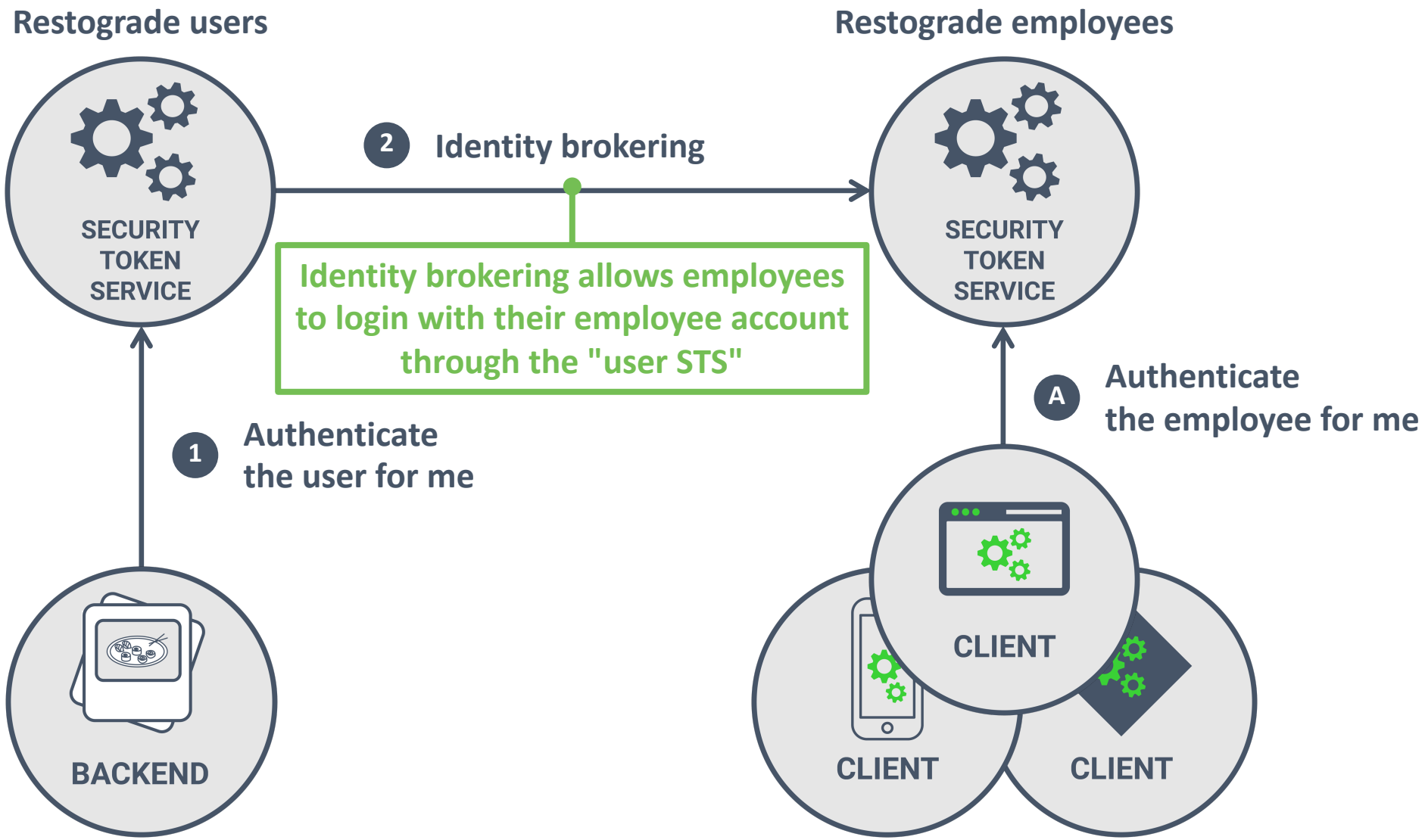
OpenID Connect in action

OIDC SUPPORTS OFFLOADING USER AUTHENTICATION

Standalone applications can offload user authentication to a central provider.

The authenticated user is matched against an internal user in the system maintained in a "session".


CLIENT
SECURITY TOKEN SERVICE
The client relies on a single STS
The STS allows brokering to social identity providers
Many STS implementations offer social login as a core feature and handle all low-level implementation details


Restograde users
Restograde employees
SECURITY TOKEN SERVICE
SECURITY TOKEN SERVICE
2
Identity brokering
Identity brokering allows employees to login with their employee account through the "user STS"
1
Authenticate the user for me
A
Authenticate the employee for me
BACKEND
CLIENT
CLIENT
CLIENT

Identity brokering is a powerful feature

Identity brokering allows one STS to offload the authentication to another STS, and so on.

It offers a lot of flexibility while hiding the complexity from the client applications.

Introduction to OAuth 2.0 and OIDC

RECAP

- OAuth 2.0 is not about authentication or authorization, but delegation
  - Access tokens represent authority given to a client by the STS
  - The API relies on the access token to make authorization decisions

- OpenID Connect is about delegating authentication to a third-party
  - OIDC flows result in an identity token containing properties about the authentication
  - OIDC combines with OAuth 2.0, as the same flow can also issue access tokens

- Authorization is the responsibility of the resource server
  - The authorization server has to properly verify incoming access tokens
  - The claims in the access token support function-level and object-level authorization

BEST PRACTICES

- OAuth 2.0 best practices only support three flows
  - The ***Client Credentials*** grant supports direct machine-to-machine access
  - The ***Authorization Code flow with PKCE*** supports user-based access
  - The ***Device flow*** supports OAuth 2.0 on input-constrained devices

- OpenID Connect is the current standard for implementing authentication
  - OIDC relies on the same flows as OAuth 2.0, with the same best practices

- Minimize the attack surface following from using OAuth 2.0
  - Use least-privilege scopes to limit the power of an access token
  - Use strict ***redirect URIs*** to prevent token stealing attacks
  - Use ***PKCE*** in all occurrences of the Authorization Code flow to preserve flow integrity

Thank you!

Connect on social media for more in-depth security content

@PhilippeDeRyck

/in/PhilippeDeRyck