OAuth 2.1 best practices

Dr. Philippe De Ryck

https://Pragmatic Web Security.com

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

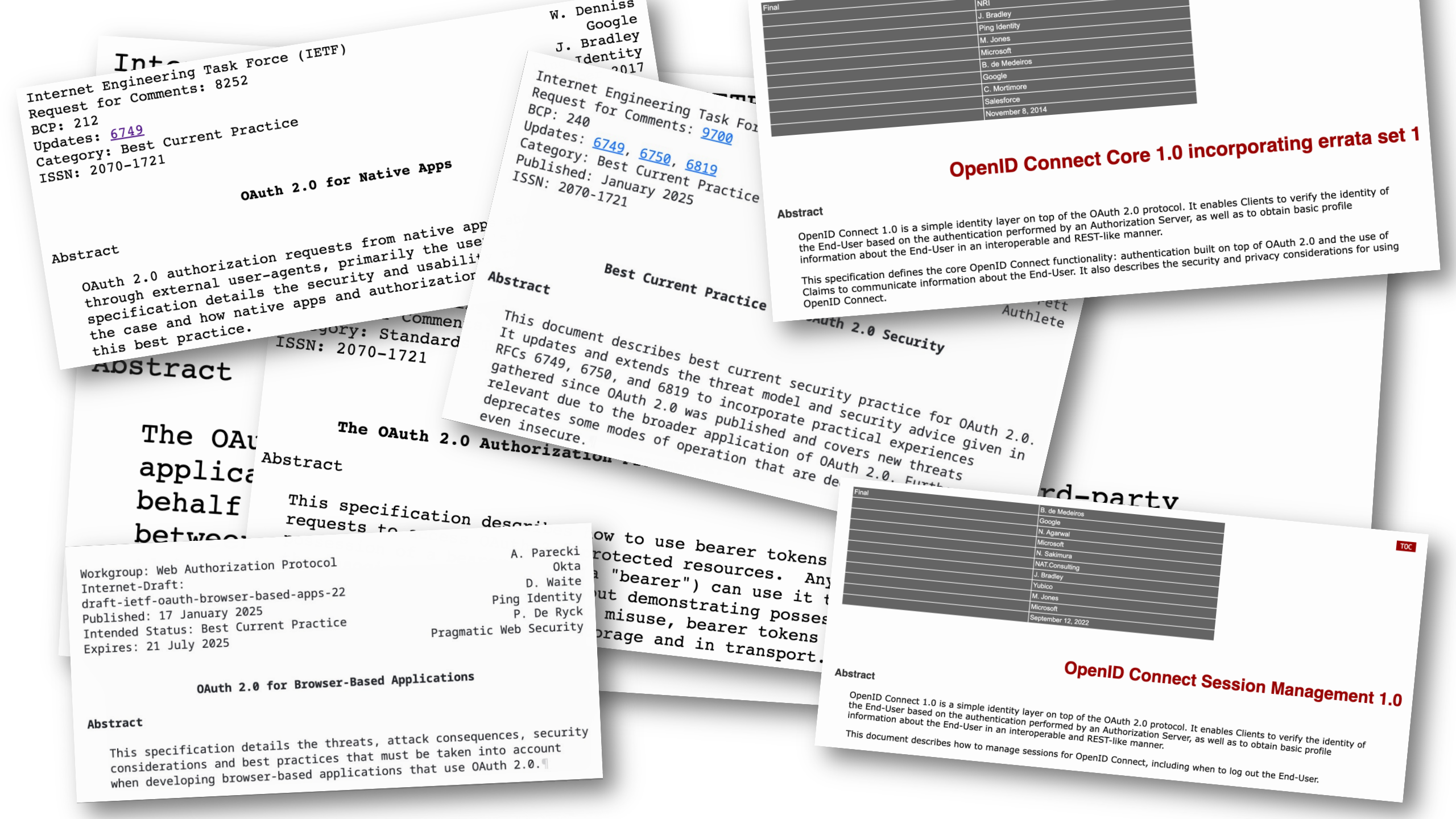
D. Hardt, Ed.
Microsoft

October 2012

The OAuth 2.0 Authorization Framework

Abstract

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



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

OAuth 2.0 for Native Apps

Abstract

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

W. Denniss
Google
J. Bradley
Identity
2017

Internet Engineering Task For
Request for Comments: 9700
BCP: 240
Updates: 6749, 6750, 6819
Category: Best Current Practice
Published: January 2025
ISSN: 2070-1721

Best Current Practice

Abstract

This document describes best current security practice for OAuth 2.0.
It updates and extends the threat model and security advice given in
RFCs 6749, 6750, and 6819 to incorporate practical experiences
gathered since OAuth 2.0 was published and covers new threats
relevant due to the broader application of OAuth 2.0. Furth
deprecates some modes of operation that are de
even insecure.

Authlete

Auth 2.0 Security

Abstract

The OAu
applica
behalf
betwee

This specification descri
requests to

rd-party

ow to use bearer tokens
rotected resources. Any
a "bearer") can use it t
ut demonstrating posses
misuse, bearer tokens
orage and in transport.

Final

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

OpenID Connect Core 1.0 incorporating errata set 1

Abstract

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

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

Workgroup: Web Authorization Protocol
Internet-Draft:
draft-ietf-oauth-browser-based-apps-22
Published: 17 January 2025
Intended Status: Best Current Practice
Expires: 21 July 2025

A. Parecki
Okta
D. Waite
Ping Identity
P. De Ryck
Pragmatic Web Security

OAuth 2.0 for Browser-Based Applications

Abstract

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

Final

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

TOC

OpenID Connect Session Management 1.0

Abstract

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

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

Photo by Andrew Umansky on Unsplash

Workgroup: OAuth Working Group
Internet-Draft: draft-ietf-oauth-v2-1-15
Published: 2 March 2026
Intended Status: Standards Track
Expires: 3 September 2026

D. Hardt
Hellō
A. Parecki
Okta
T. Lodderstedt
SPRIND

The OAuth 2.1 Authorization Framework

Abstract

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

OAuth 2.1 simplifies OAuth 2.0 into the current best practices, but will not introduce new features

OAuth 2.1 can be augmented with advanced security features, such as PAR and DPoP

I am *Dr. Philippe De Ryck*

Pragmatic Web Security
Security for developers

Founder of Pragmatic Web Security

Google Developers Experts

Google Developer Expert

SecAppDev

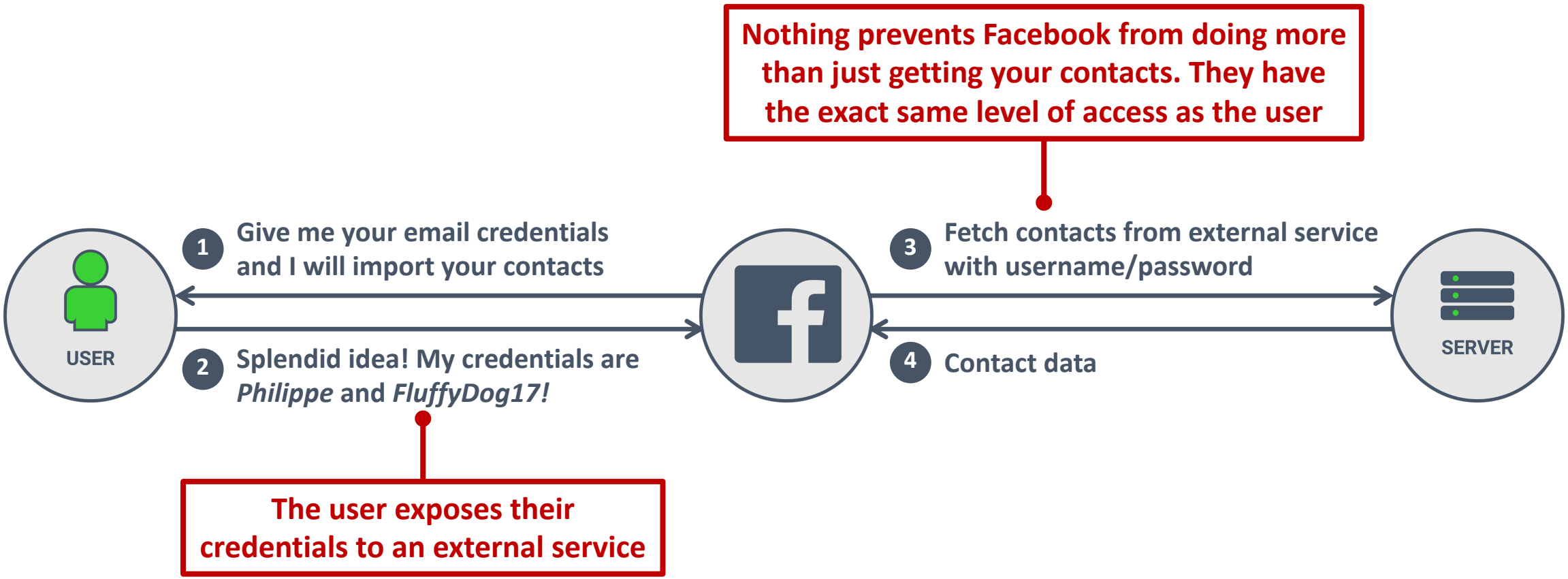
SecAppDev organizer

I help developers with security

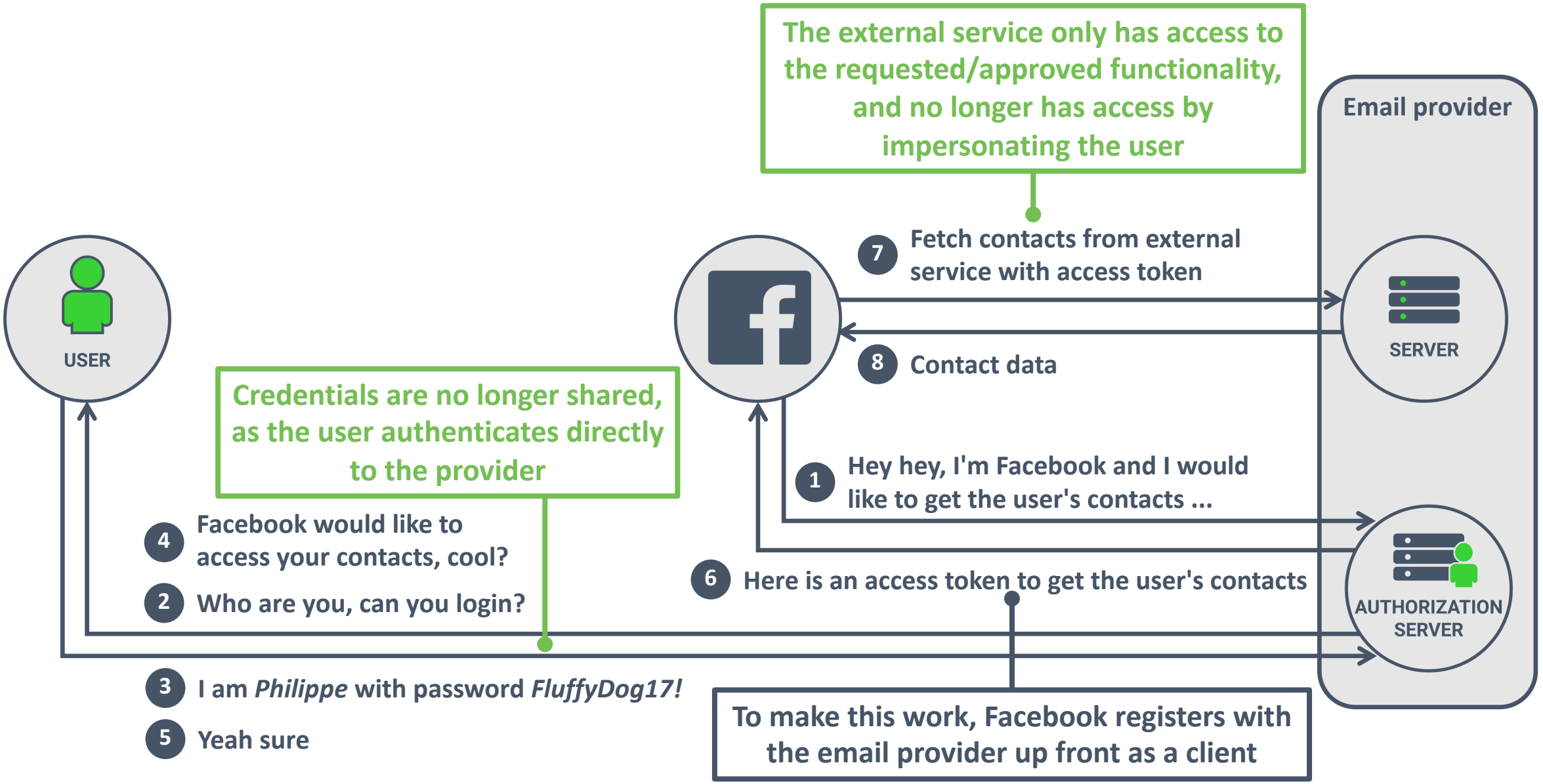
- **Hands-on in-depth security training**
- **Advanced online security courses**
- **Security advisory services**

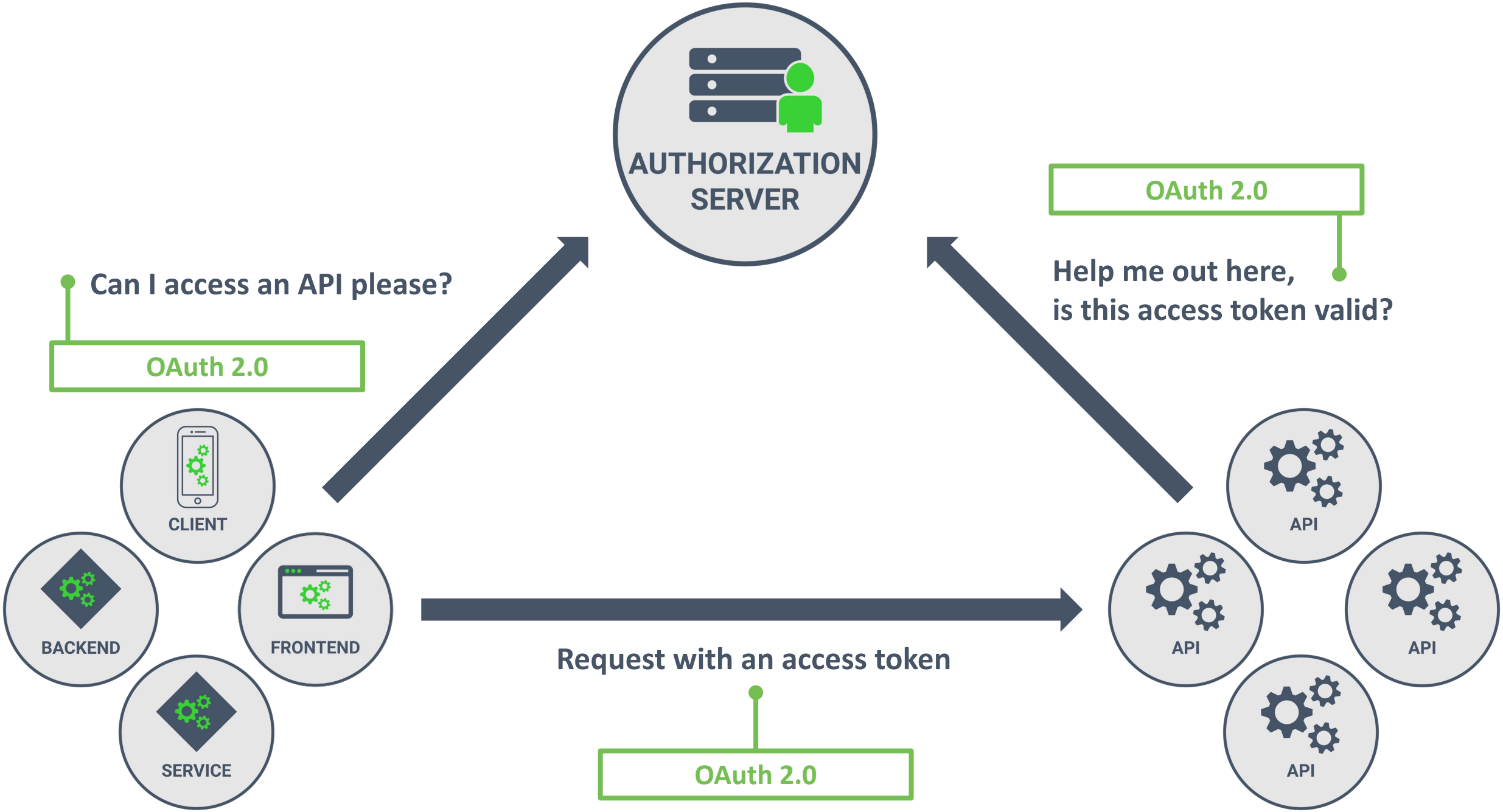
https://pdr.online

So what does OAuth do?

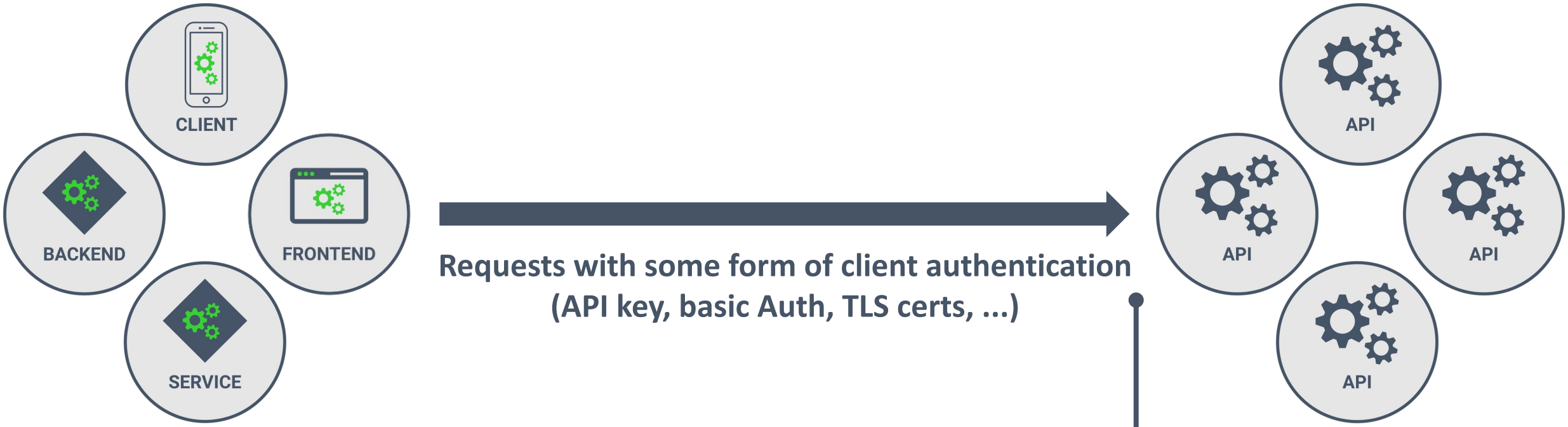

Nothing prevents Facebook from doing more than just getting your contacts. They have the exact same level of access as the user
USER
1
Give me your email credentials and I will import your contacts
2
Splendid idea! My credentials are *Philippe* and *FluffyDog17!*
3
Fetch contacts from external service with username/password
4
Contact data
SERVER
The user exposes their credentials to an external service

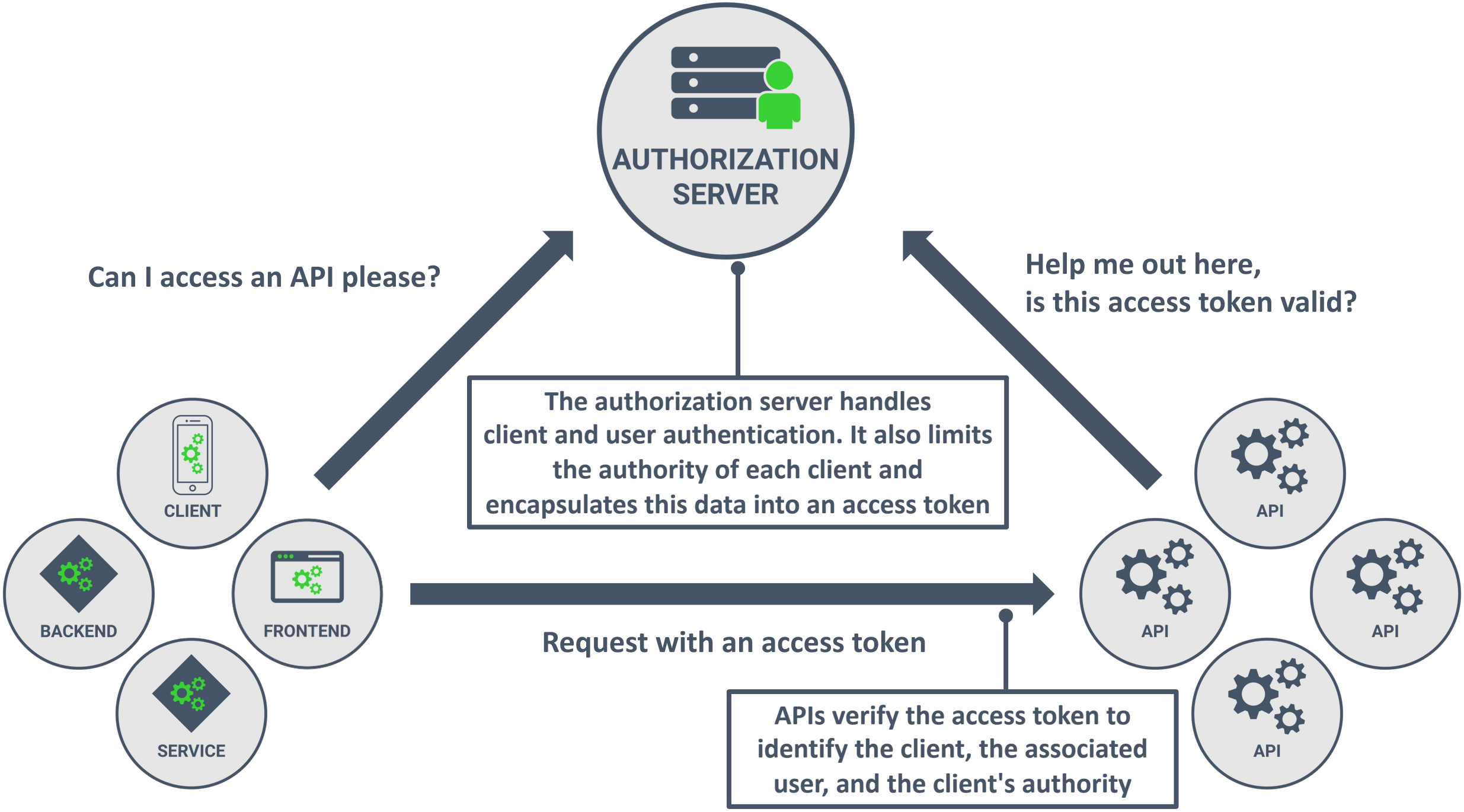
THE CONCEPT BEHIND OAUTH

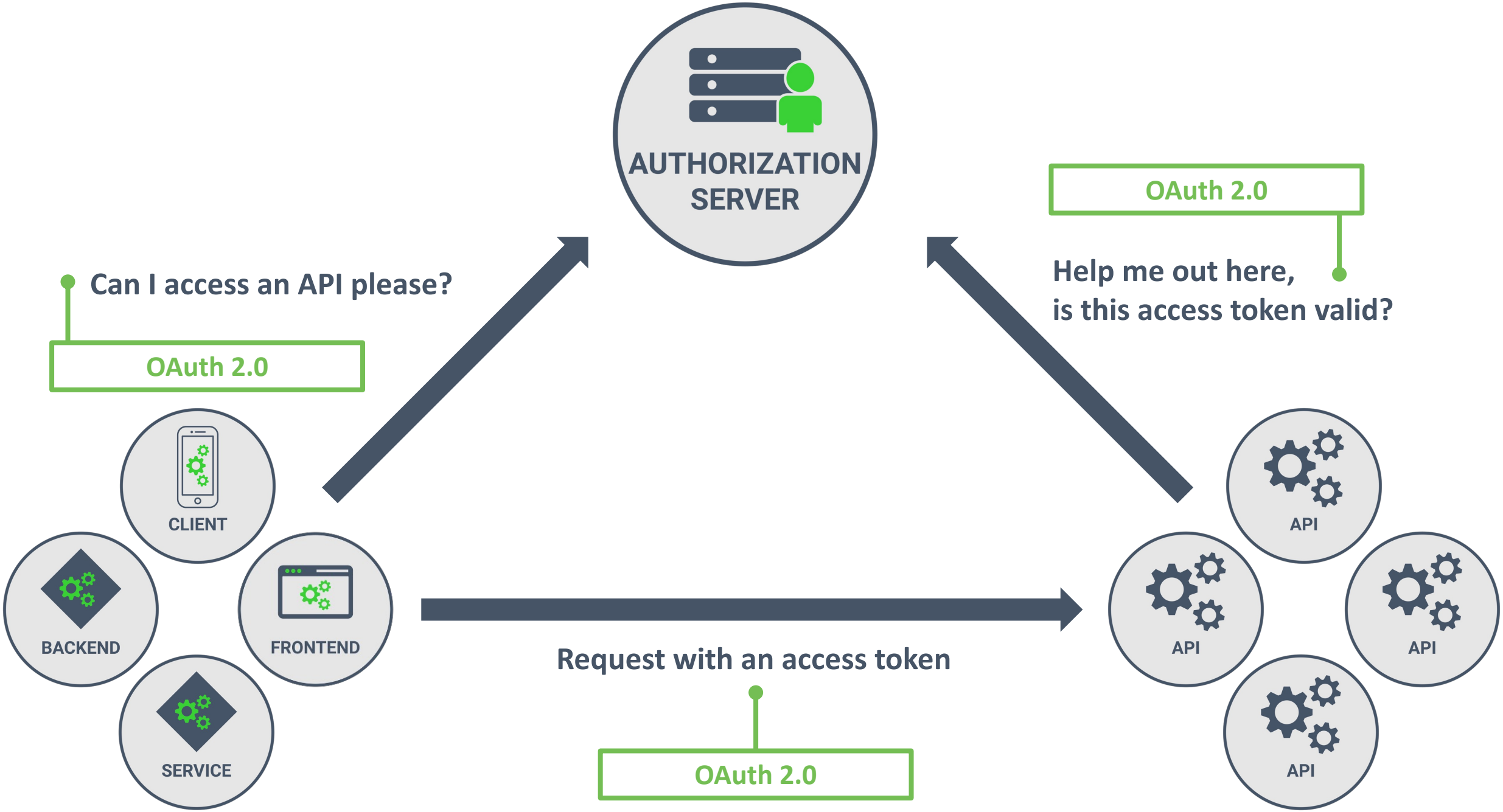



AUTHORIZATION SERVER
OAuth 2.0
Help me out here, is this access token valid?
Can I access an API please?
OAuth 2.0
CLIENT
BACKEND
FRONTEND
SERVICE
API
API
API
API
Request with an access token
OAuth 2.0

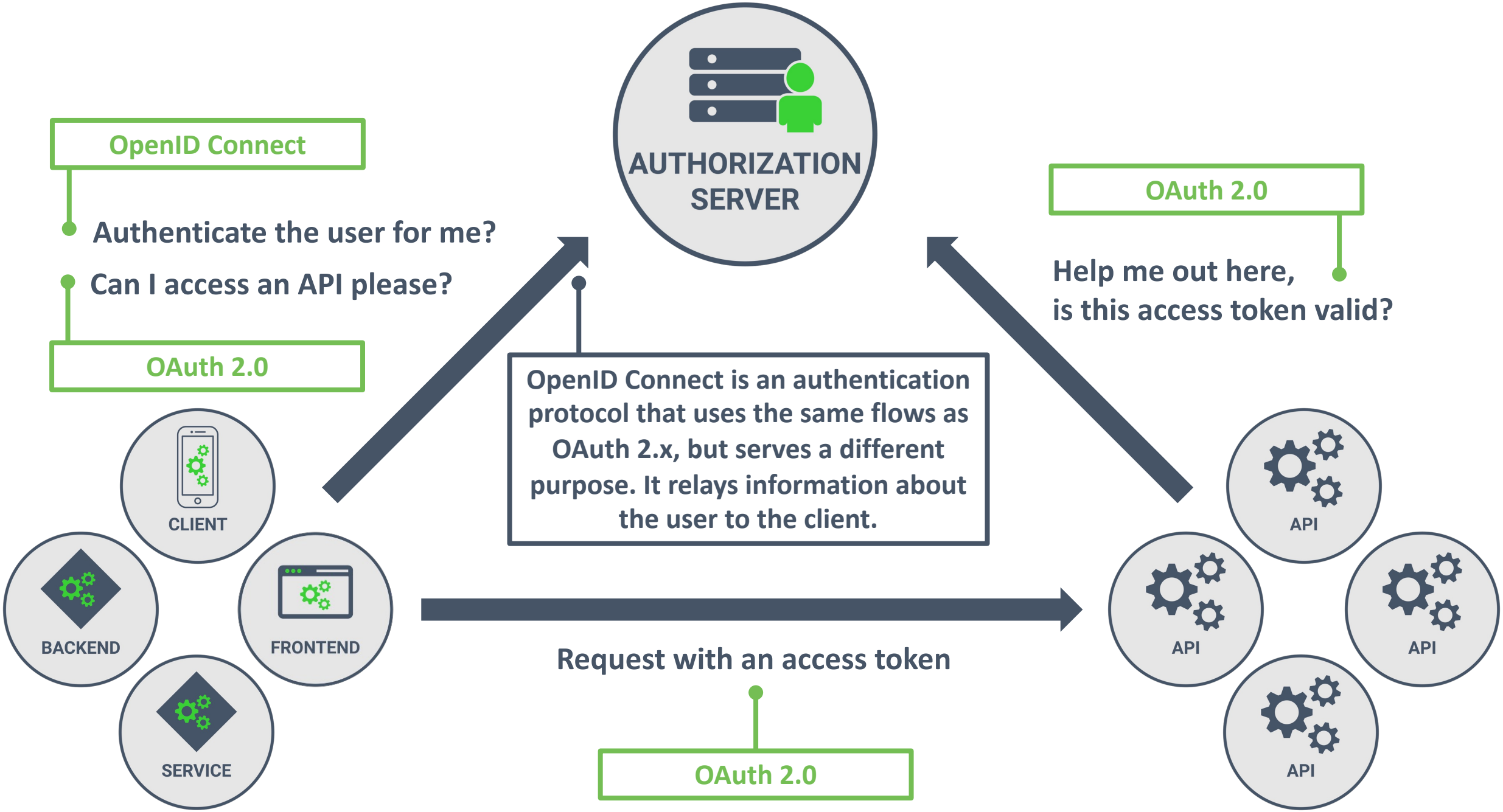
Third party access is cool, but what about enterprise?


CLIENT
BACKEND
FRONTEND
SERVICE
API
API
API
API
Requests with some form of client authentication
(API key, basic Auth, TLS certs, ...)
APIs become responsible for permission mapping, limiting what certain clients can do on behalf of the user
The APIs need to be aware of all clients and the way they authenticate. Additionally, the client needs a way to tell the API on behalf of what user they are acting.


AUTHORIZATION SERVER
Can I access an API please?
Help me out here, is this access token valid?
The authorization server handles client and user authentication. It also limits the authority of each client and encapsulates this data into an access token
CLIENT
BACKEND
FRONTEND
SERVICE
API
API
API
API
Request with an access token
APIs verify the access token to identify the client, the associated user, and the client's authority

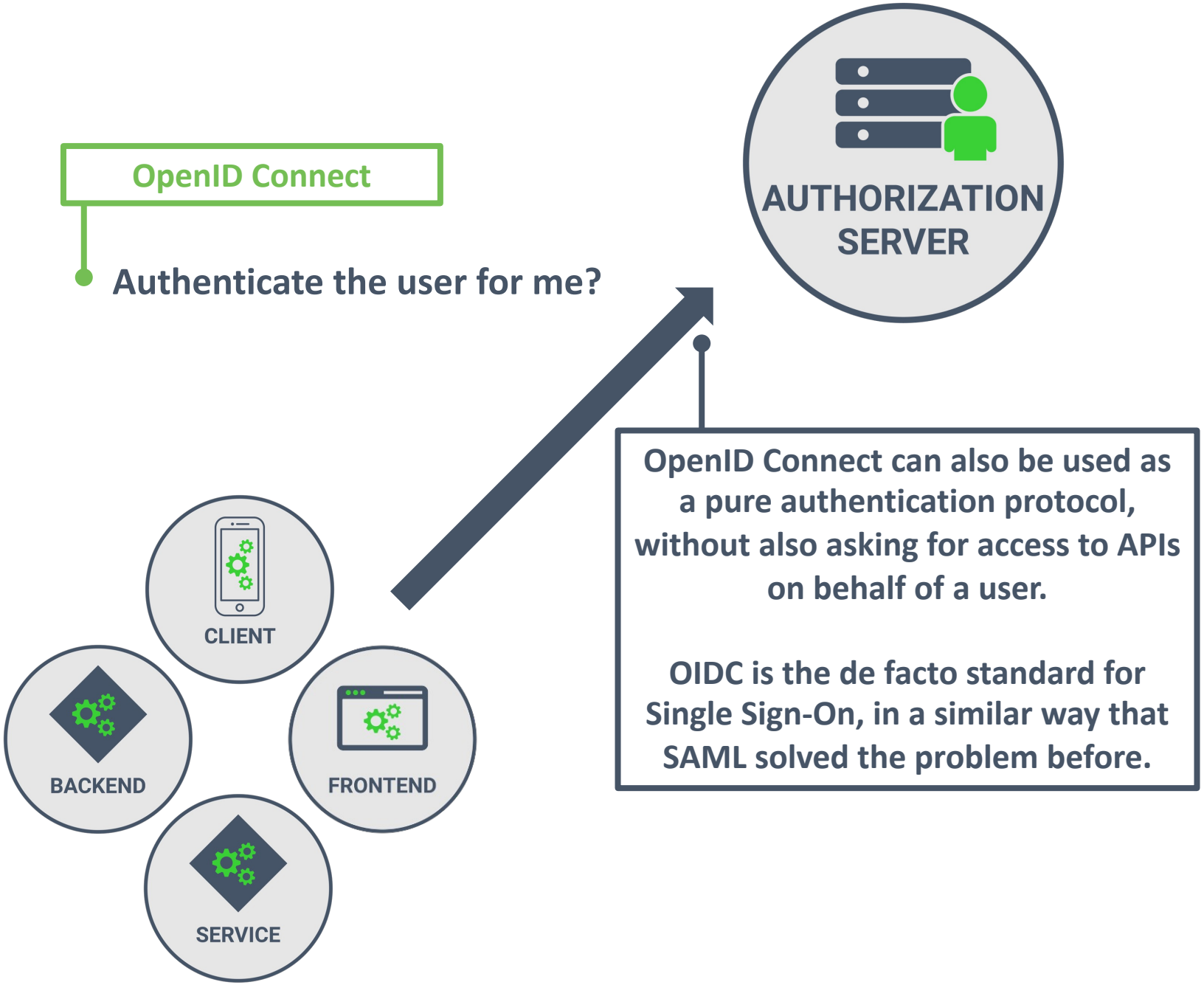

AUTHORIZATION SERVER
OAuth 2.0
Help me out here, is this access token valid?
Can I access an API please?
OAuth 2.0
CLIENT
BACKEND
FRONTEND
SERVICE
API
API
API
API
Request with an access token
OAuth 2.0

What about OpenID Connect?


AUTHORIZATION SERVER
OpenID Connect
Authenticate the user for me?
Can I access an API please?
OAuth 2.0
OpenID Connect is an authentication protocol that uses the same flows as OAuth 2.x, but serves a different purpose. It relays information about the user to the client.
OAuth 2.0
Help me out here, is this access token valid?
CLIENT
BACKEND
FRONTEND
SERVICE
API
API
API
API
Request with an access token
OAuth 2.0

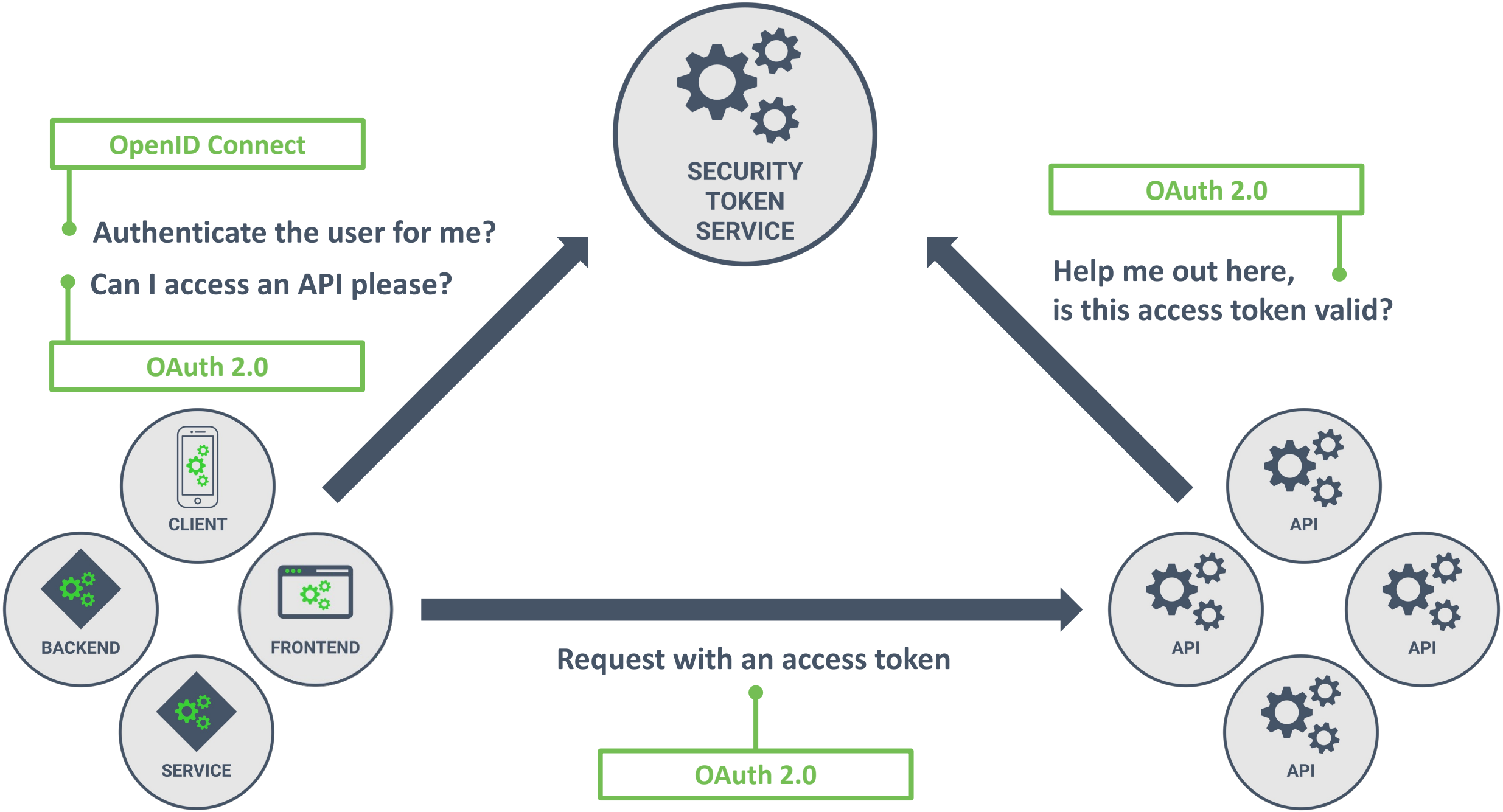
OpenID Connect

Authenticate the user for me?

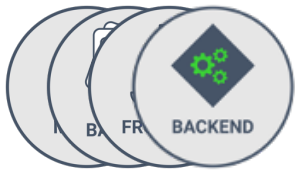


OpenID Connect can also be used as a pure authentication protocol, without also asking for access to APIs on behalf of a user.

OIDC is the de facto standard for Single Sign-On, in a similar way that SAML solved the problem before.

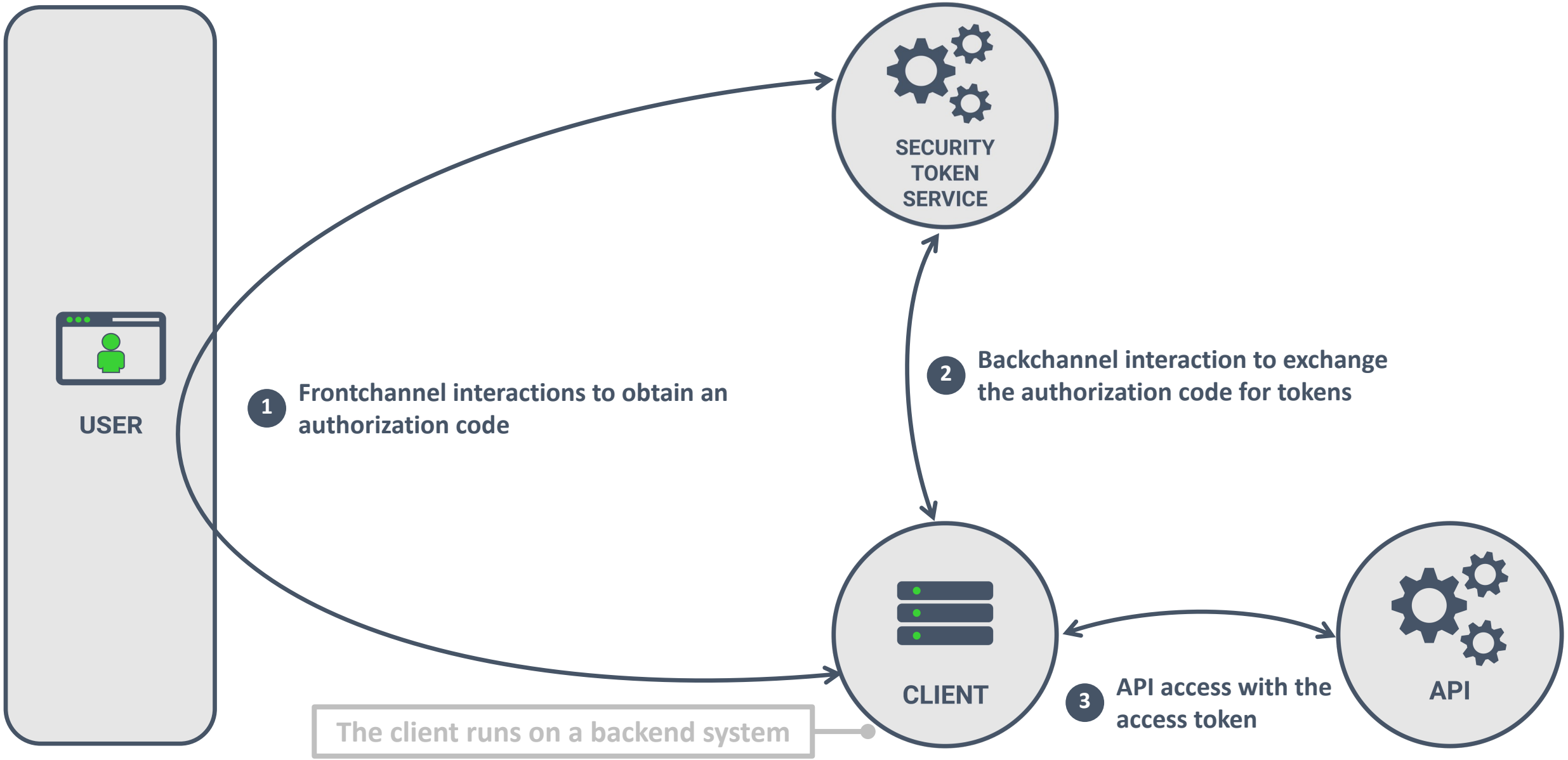

OpenID Connect
Authenticate the user for me?
Can I access an API please?
OAuth 2.0
SECURITY TOKEN SERVICE
OAuth 2.0
Help me out here, is this access token valid?
CLIENT
BACKEND
FRONTEND
SERVICE
Request with an access token
OAuth 2.0
API
API
API
API

TERMINOLOGY

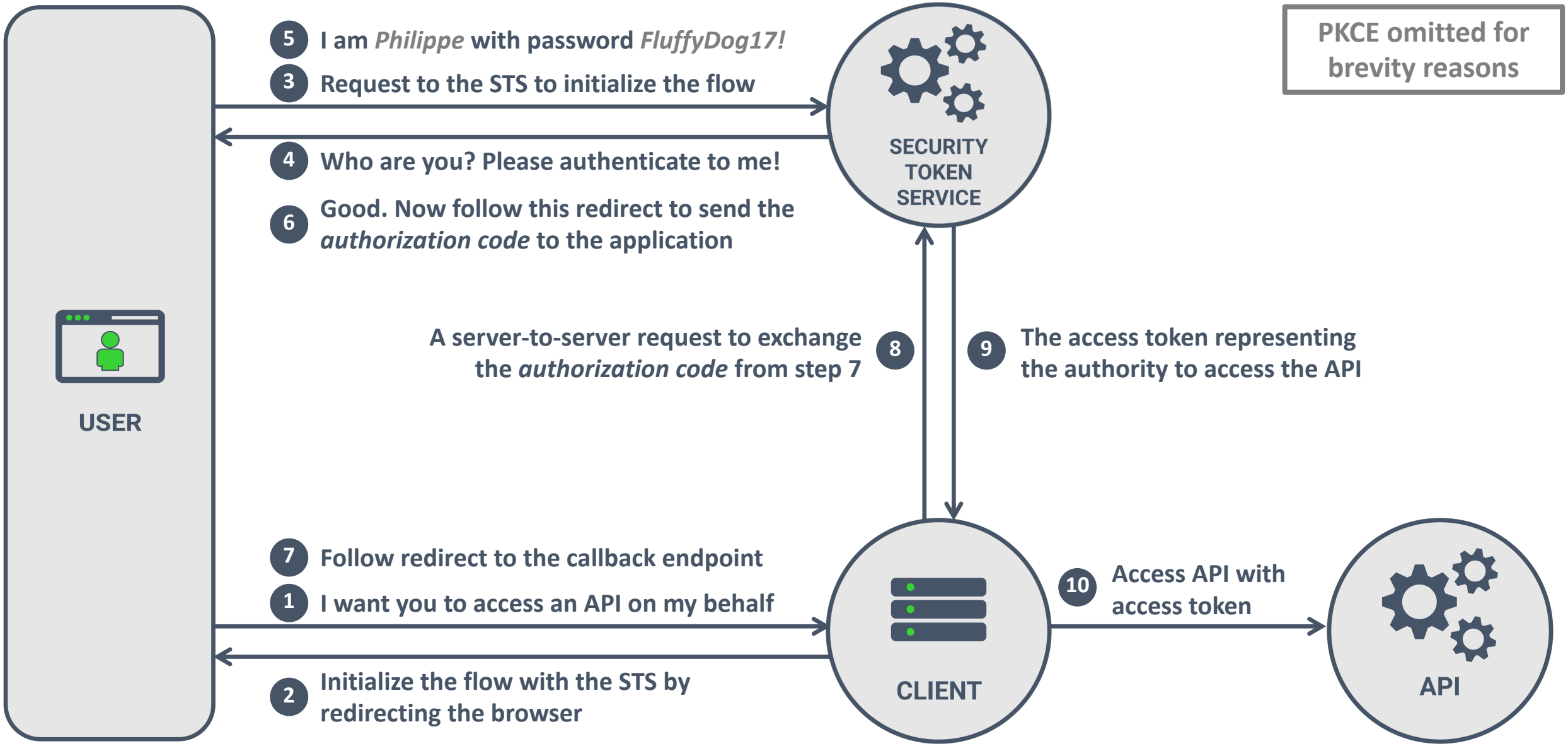
	This course	OAuth 2.0	OpenID Connect
	**User**	Resource Owner	End-User
	**API**	Resource Server	
	**Security Token Service (STS)**	Authorization Server	OpenID Provider
	**Client**	Client	Relying Party

The OAuth 2.1 flow for backend clients

HIGH-LEVEL OVERVIEW OF THE *AUTHORIZATION CODE* FLOW



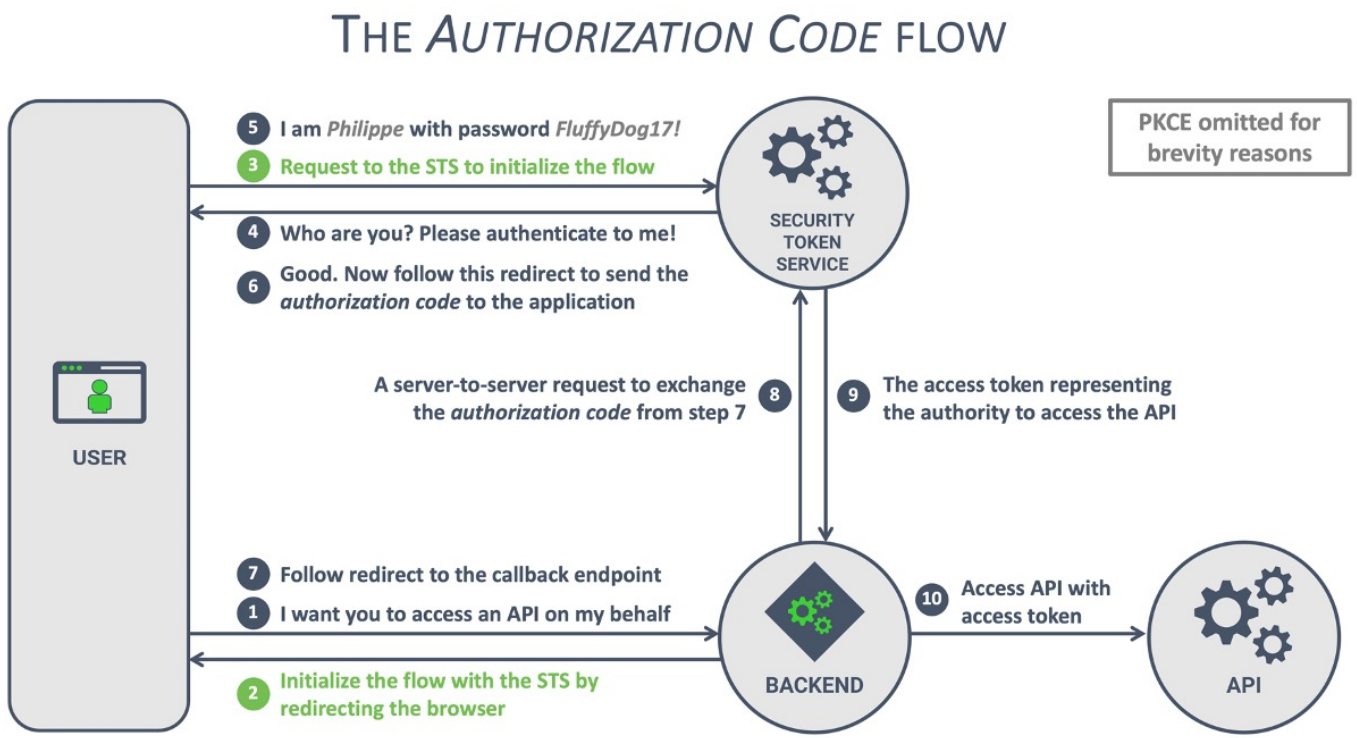
THE *AUTHORIZATION CODE* FLOW FOR BACKEND CLIENTS



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

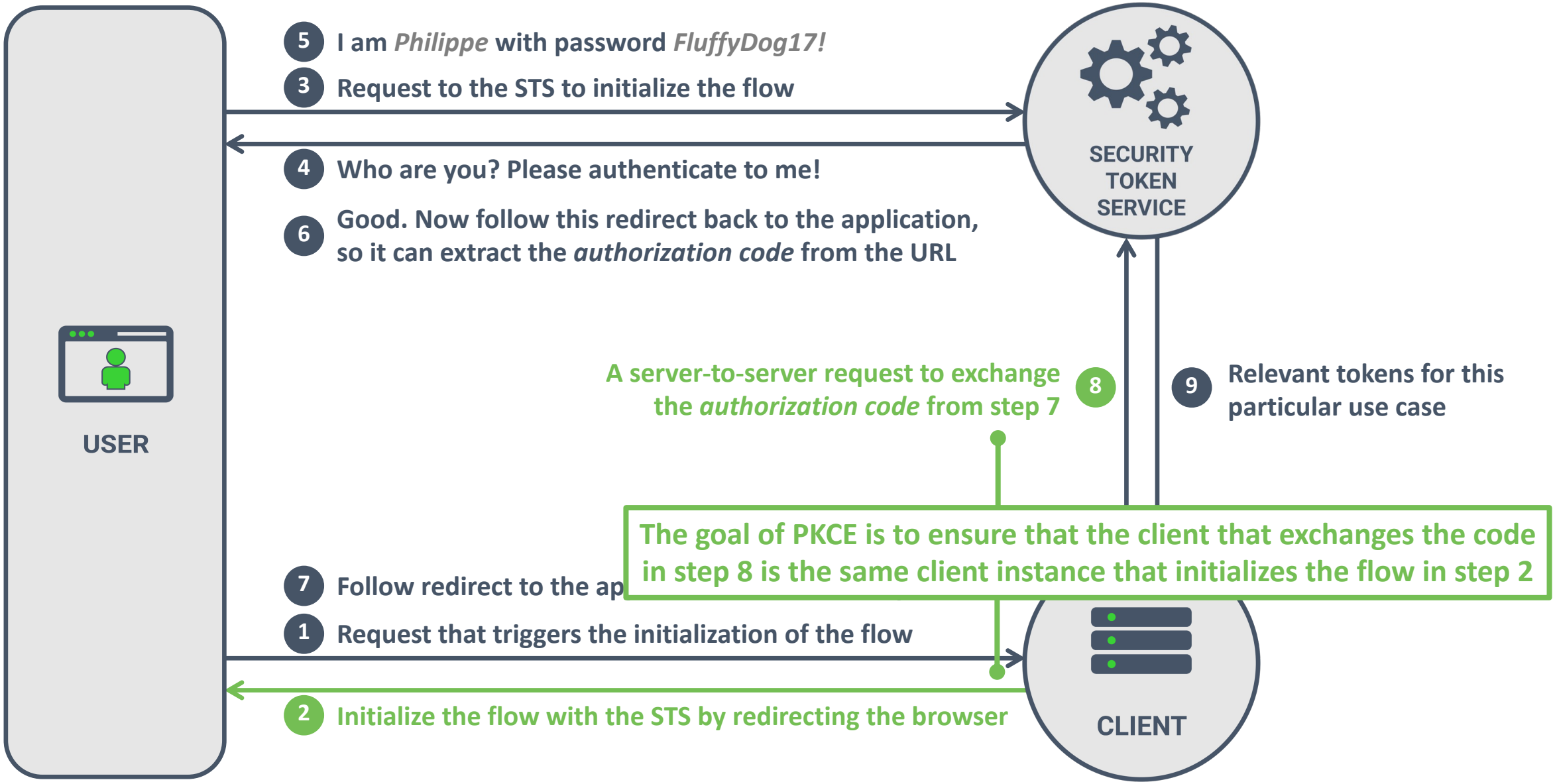
```
1  https://sts.restograde.com/authorize
2    ?response_type=code
3    &scope=reviews
4    &client_id=AB983CEYgx4mdUg3NKNKHjwfNAL5Fb42
5    &redirect_uri=https://virtualfoodie.com/callback
6    &code_challenge=29K8tipblinCeP … HZ1PqLVxd9s
7    &code_challenge_method=S256
```

- Indicates the *authorization code flow*
- We want a token with *reviews* access
- The client requesting the token
- Where the STS should send the code
- Flow security feature (PKCE)

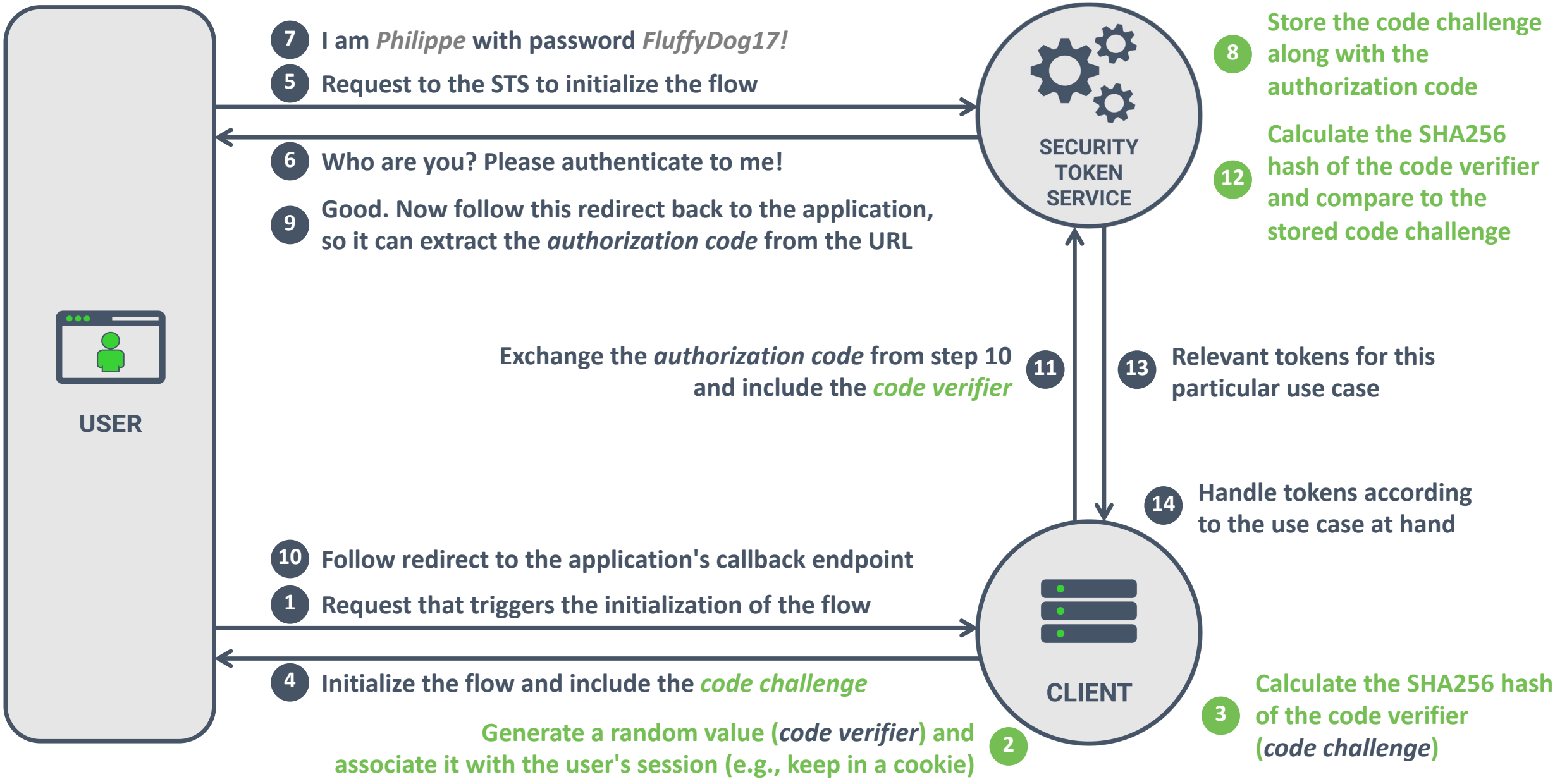


Proof Key for Code Exchange (PKCE) helps protect the integrity of the *Authorization Code* flow

THE CONCEPT OF PKCE



THE *AUTHORIZATION CODE* FLOW WITH PKCE

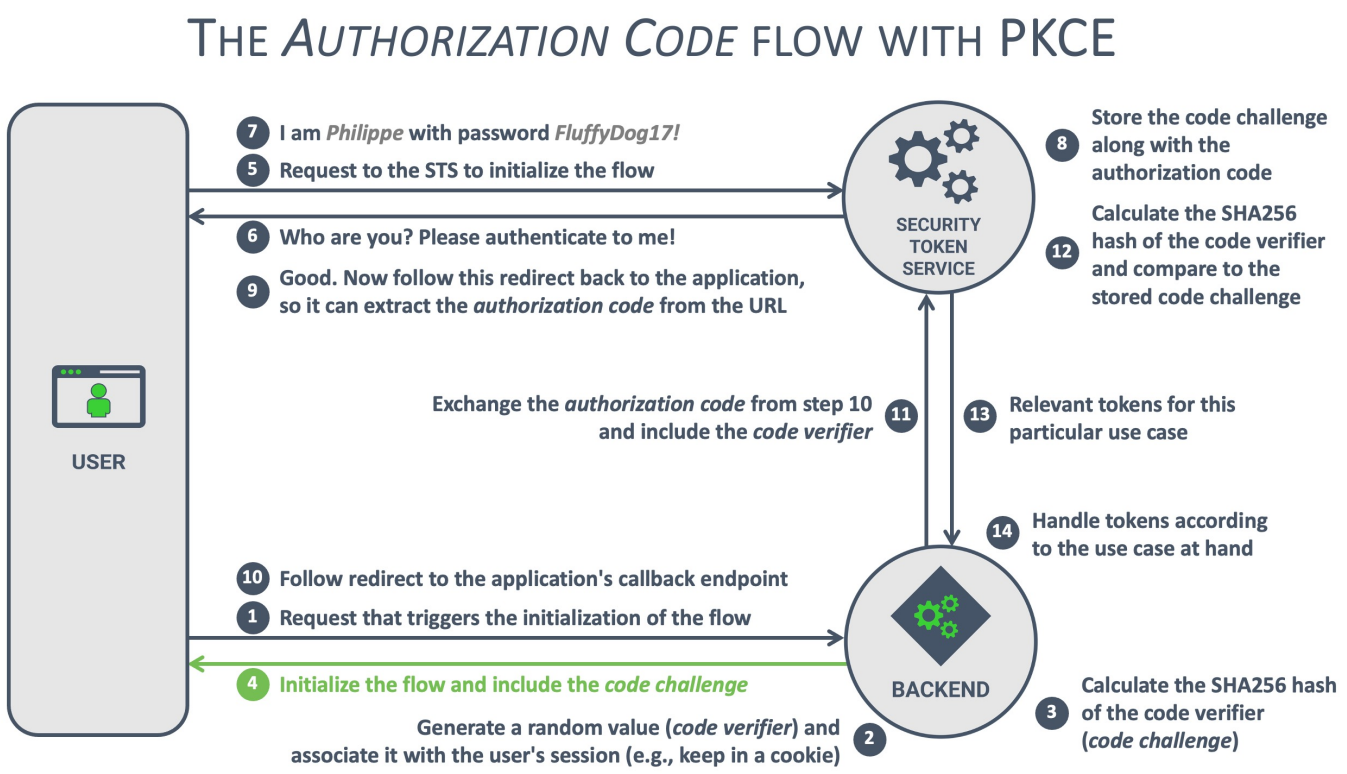


4 5 *The authorization request (a redirect to the STS)*

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

The code challenge (hash of code verifier)

The hash function used (for upgradeability)

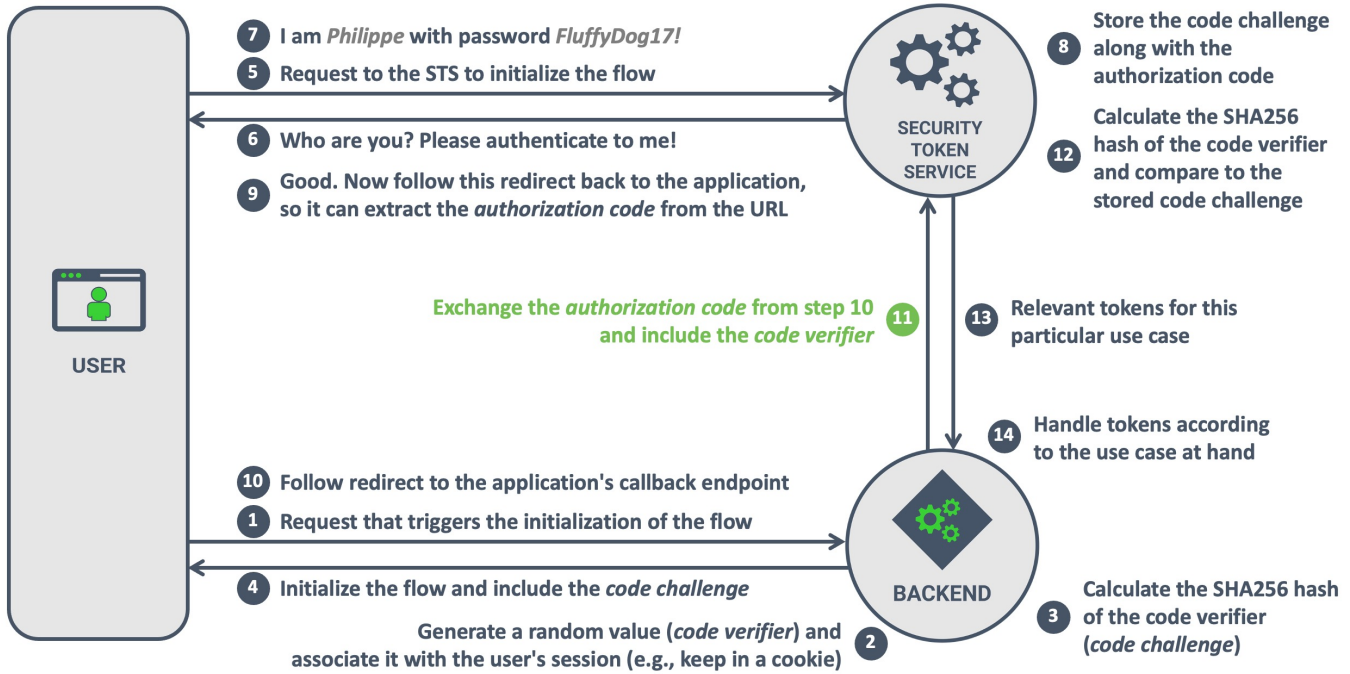


11 *The request to exchange the authorization code*

```
1  POST /oauth/token
2
3   grant_type=authorization_code
4  &client_id=FN983CEYgx4mdUg3NKNKHjwfNAL5Fb42
5  &client_secret=6ODRv0g…OVOSWI
6  &redirect_uri=https://restograde.com/callback
7  &code=ySVyktqNkEKJyyIjOKCVwCurNlGoRDcaLYEbW2j5WxZY
8  &code_verifier=DOHpp1yiK0iElVij … K8HBZBqr75fKPps
```

The code verifier from step 2

THE *AUTHORIZATION CODE* FLOW WITH PKCE



Proof Key for Code Exchange (PKCE)

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

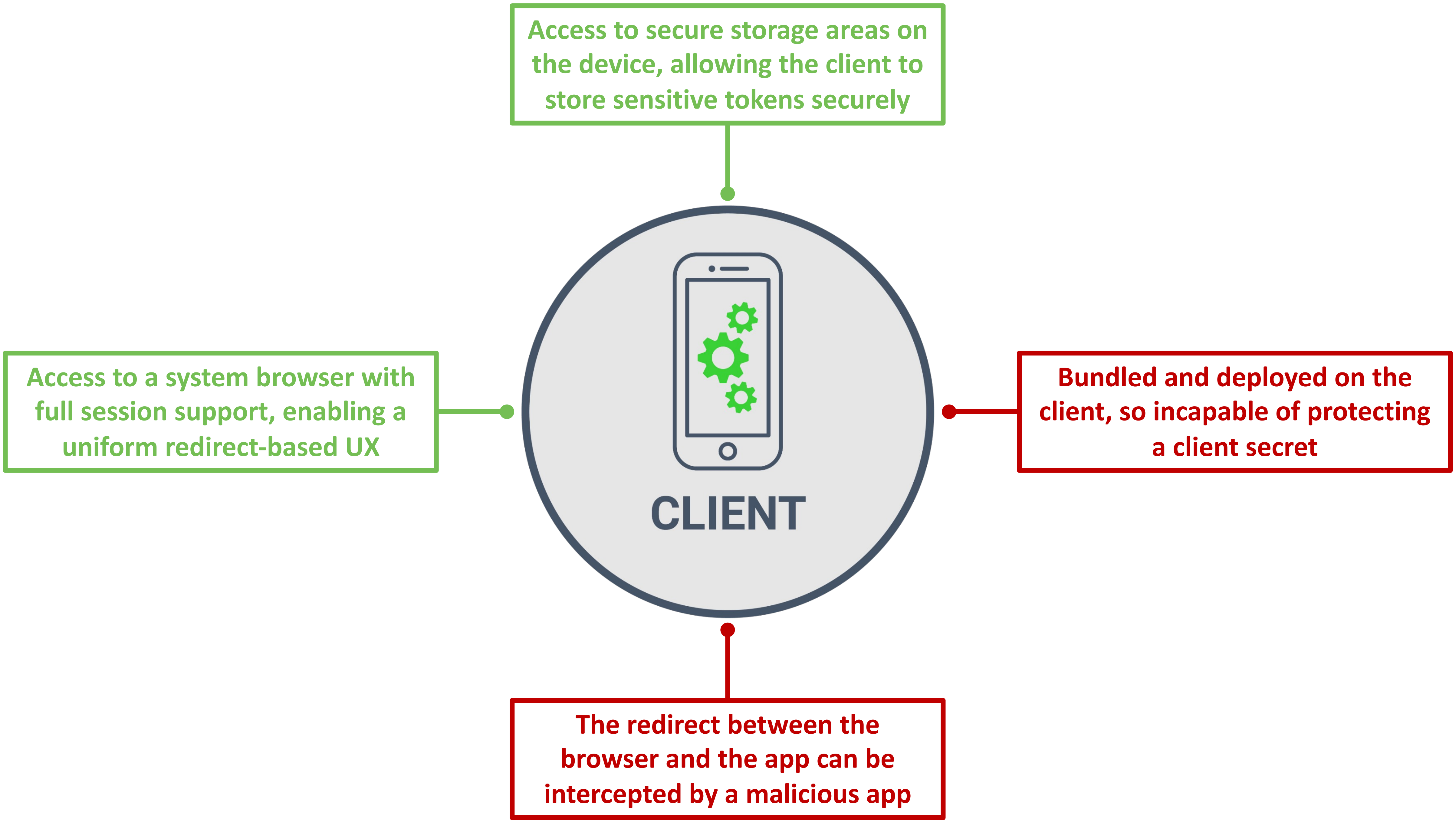
- PKCE ensures that the same client intializes and finalizes the flow
  - PKCE was originally intended to secure flows of public clients (no client authentication)
  - Today, PKCE is a recommended best practice to guarantee flow integrity

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

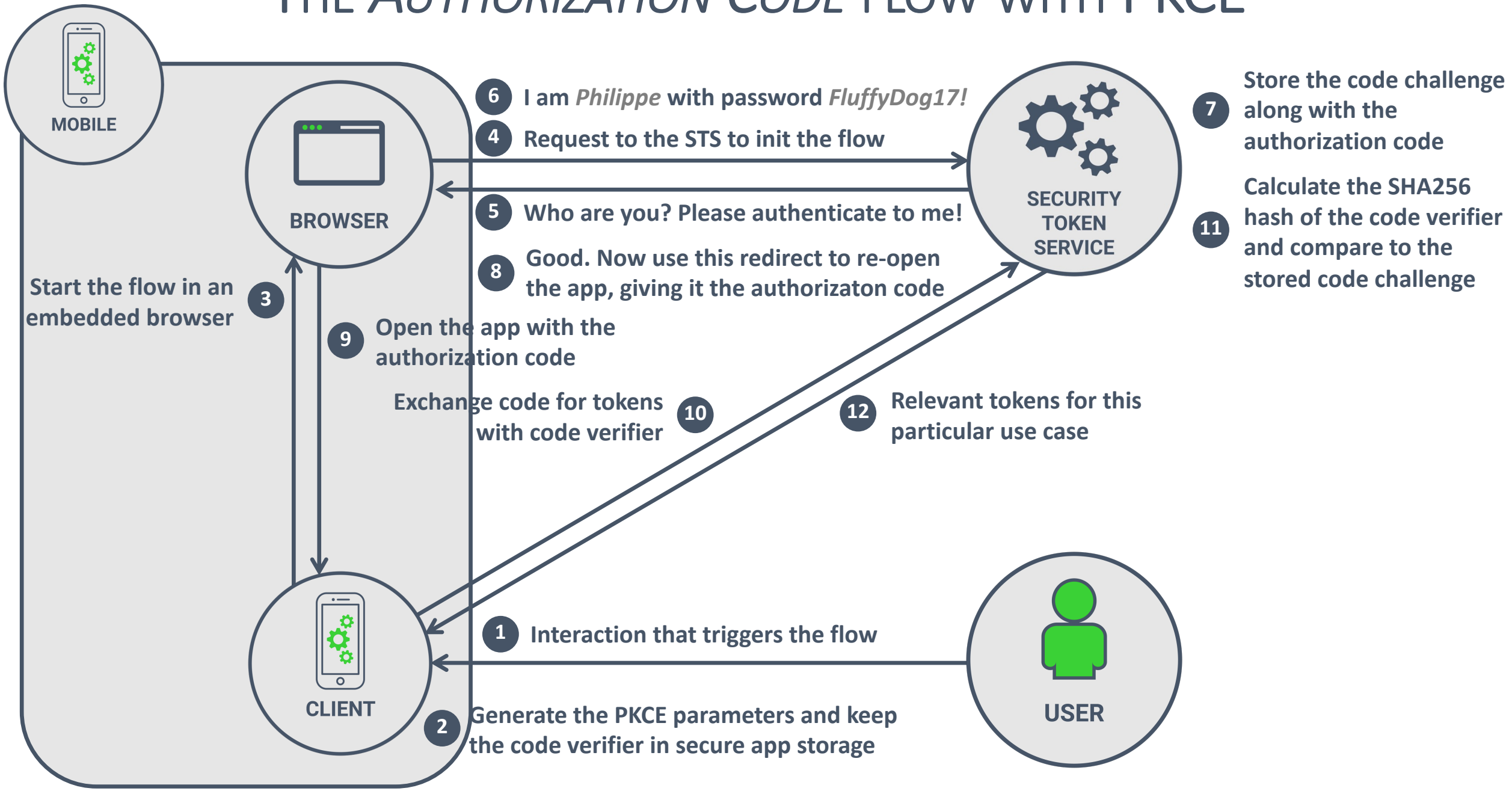
PKCE is a mandatory security best practice for all *Authorization Code* flows in OAuth 2.1

PKCE in action

OAuth 2.1 flows for user-side clients

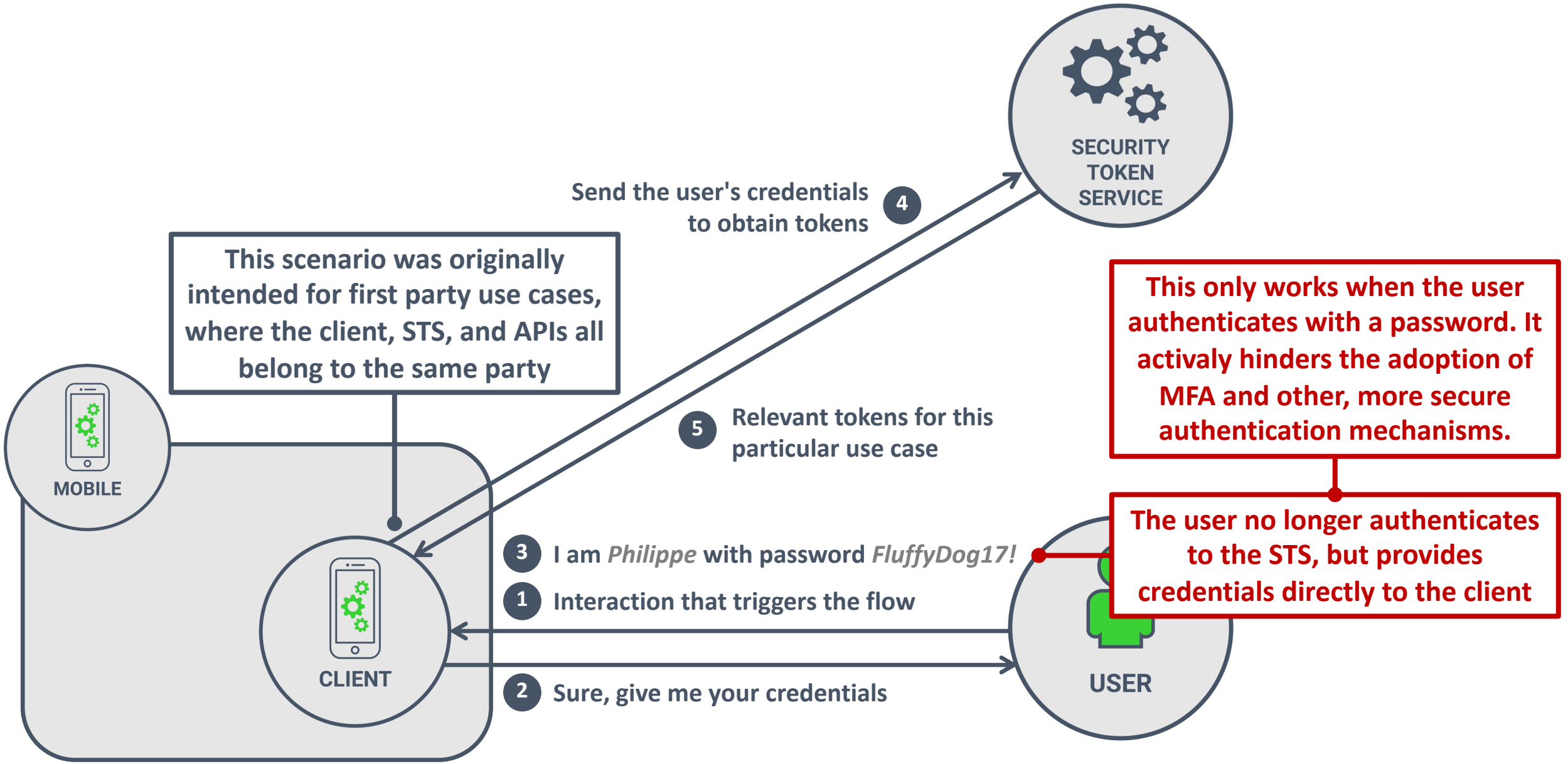

Access to secure storage areas on the device, allowing the client to store sensitive tokens securely
Access to a system browser with full session support, enabling a uniform redirect-based UX
CLIENT
Bundled and deployed on the client, so incapable of protecting a client secret
The redirect between the browser and the app can be intercepted by a malicious app

THE *AUTHORIZATION CODE* FLOW WITH PKCE



The current best practice for mobile apps is to run an Authorization Code flow in an embedded system browser

THE DEPRECATED *RESOURCE OWNER PASSWORD CREDENTIALS* FLOW



Native user authentication within the app is not a recommended best practice (for now)

Workgroup: Web Authorization Protocol
Internet-Draft:
draft-ietf-oauth-first-party-apps-03
Published: 28 February 2026
Intended Status: Standards Track
Expires: 1 September 2026

A. Parecki
Okta
G. Fletcher
Practical Identity LLC
P. Kasselman
Defakto Security

OAuth 2.0 for First-Party Applications

Abstract

This document defines the Authorization Challenge Endpoint, which supports clients that want to control the process of obtaining authorization from the user using a native experience.

In many cases, this can provide an entirely browserless OAuth 2.0 experience suited for native applications, only delegating to the browser in unexpected, high risk, or error conditions.

https://datatracker.ietf.org/doc/draft-ietf-oauth-first-party-apps/

Workgroup: Web Authorization Protocol
Internet-Draft:
draft-ietf-oauth-attestation-based-client-auth-09
Published: 26 May 2026
Intended Status: Standards Track
Expires: 27 November 2026

T. Looker
MATTR
P. Bastian
Bundesdruckerei
C. Bormann
SPRIND

OAuth 2.0 Attestation-Based Client Authentication

Abstract

This specification defines an extension to the OAuth 2.0 protocol [RFC6749] that enables a client instance to include a key-bound attestation when interacting with an Authorization Server or Resource Server. This mechanism allows a client instance to prove its authenticity verified by a client attester without revealing its target audience to that attester. It may also serve as a mechanism for client authentication as per OAuth 2.0.

https://datatracker.ietf.org/doc/html/draft-ietf-oauth-attestation-based-client-auth

What about web frontends as a client?

Workgroup: Web Authorization Protocol
Internet-Draft:
draft-ietf-oauth-browser-based-apps-26
Published: 4 December 2025
Intended Status: Best Current Practice
Expires: 7 June 2026

A. Parecki
Okta
P. De Ryck
Pragmatic Web Security
D. Waite
Ping Identity

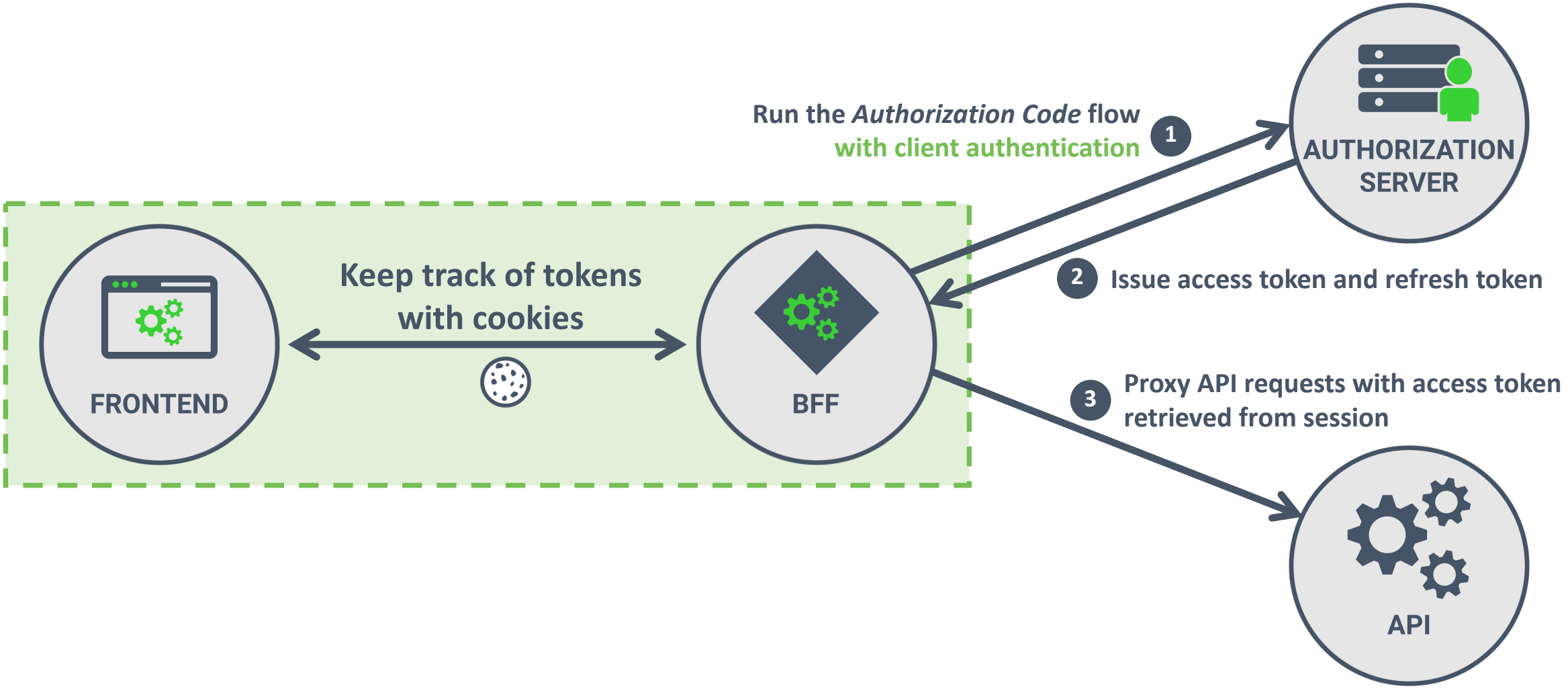
OAuth 2.0 for Browser-Based Applications

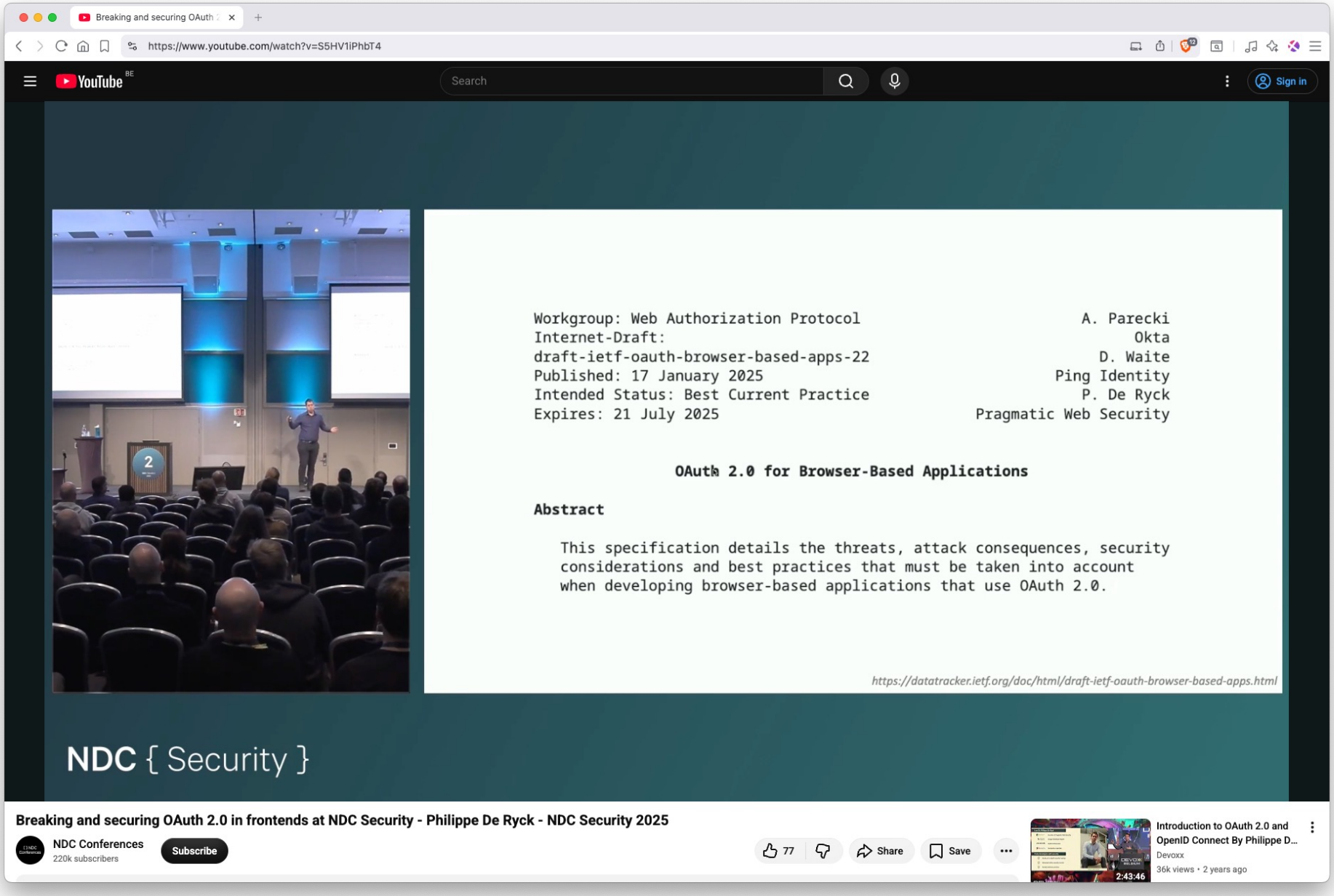
Abstract

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

https://datatracker.ietf.org/doc/html/draft-ietf-oauth-browser-based-apps

THE CONCEPT OF A BACKEND-FOR-FRONTEND (BFF)





https://www.youtube.com/watch?v=S5HV1iPhbT4

Web frontends should use a BFF, moving OAuth responsibilities into a backend client

Augmenting the flow security of OAuth 2.1

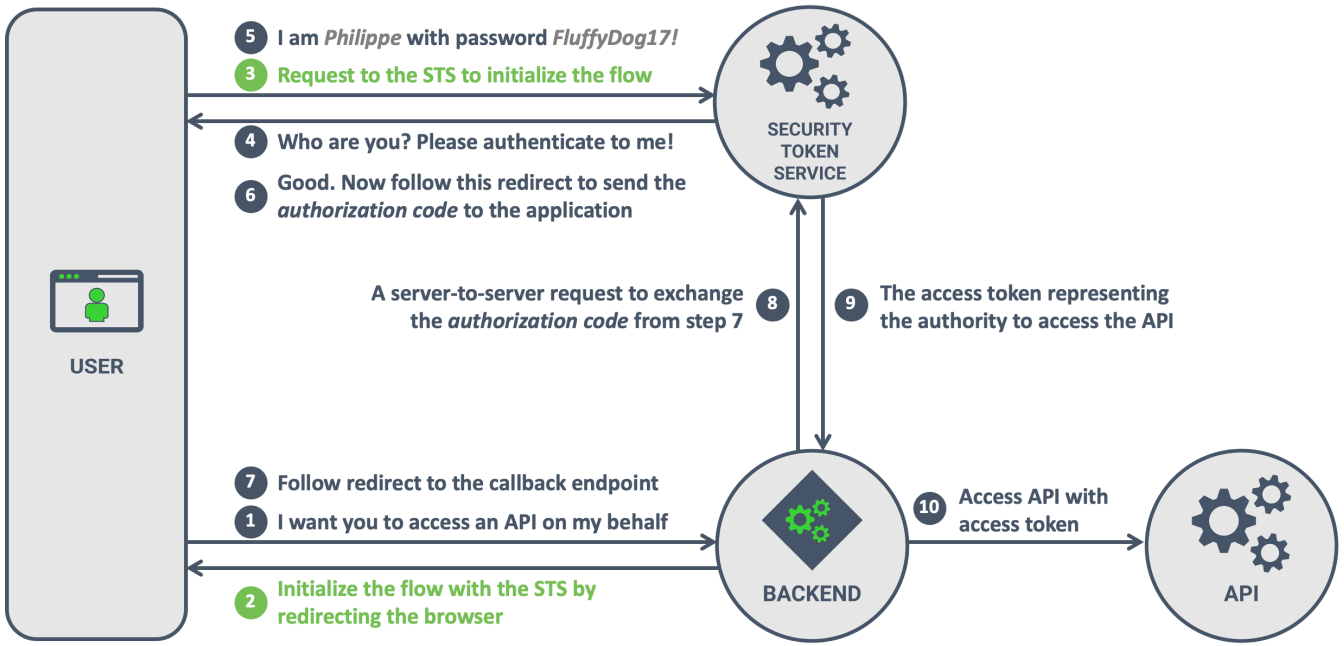
2 3 *The authorization request (a redirect to the STS)*

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

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

This URL cannot ensure the integrity of the parameters, nor does it authenticate the client that initiated the flow

THE *AUTHORIZATION CODE* FLOW FOR OAUTH



Internet Engineering Task Force (IETF)
Request for Comments: 9700
BCP: 240
Updates: 6749, 6750, 6819
Category: Best Current Practice
Published: January 2025
ISSN: 2070-1721

T. Lodderstedt
SPRIND
J. Bradley
Yubico
A. Labunets
Independent Researcher
D. Fett
Authlete

Best Current Practice for OAuth 2.0 Security

Abstract

This document describes best current security practice for OAuth 2.0. It updates and extends the threat model and security advice given in RFCs 6749, 6750, and 6819 to incorporate practical experiences gathered since OAuth 2.0 was published and covers new threats relevant due to the broader application of OAuth 2.0. Further, it deprecates some modes of operation that are deemed less secure or even insecure.¶

4.1. Insufficient Redirect URI Validation

Some authorization servers allow clients to register redirect URI patterns instead of complete redirect URIs. The authorization servers then match the redirect URI parameter value at the authorization endpoint against the registered patterns at runtime. This approach allows clients to encode transaction state into additional redirect URI parameters or to register a single pattern for multiple redirect URIs.¶

This approach turned out to be more complex to implement and more error-prone to manage than exact redirect URI matching. Several successful attacks exploiting flaws in the pattern-matching implementation or concrete config
wild (see, e.g., [research.rub2])
redirect URI effectively breaks c
authentication (depending on grar
attacker to obtain an authorizati

T. Lodderstedt
SPRIND
J. Bradley
Yubico
A. Labunets
Independent Researcher
D. Fett
Authlete

This document desc
It updates and ext
RFCs 6749, 6750, a
gathered since OAu
relevant due to th
deprecates some mo
even insecure.¶

4.8. PKCE Downgrade Attack

An authorization server that supports PKCE but does not make its use mandatory for all flows can be susceptible to a PKCE downgrade attack.¶

The first prerequisite for this attack is that there is an attacker-controllable flag in the authorization request that enables or disables PKCE for the particular flow. The presence or absence of the code_challenge parameter lends itself for this purpose, i.e., the authorization server enables and enforces PKCE if this parameter is present in the authorization request, but does not enforce PKCE if the parameter is missing.¶

Writeup: Keycloak open redirect (CVE-2023-6927)

11 January 2024

This post covers the technical details of CVE-2023-6927 which allows an attacker to create malicious Keycloak authorization request URLs that bypass the redirect URI validation. This can be exploited to steal a victim's authorization code or access token, depending on the client configuration.

The vulnerability affects *all* OAuth 2.0 clients configured with a redirect URI ending with a `*` in Keycloak < 23.0.4.

For additional information, see GitHub security advisory GHSA-9vm7-v8wj-3fqw.

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

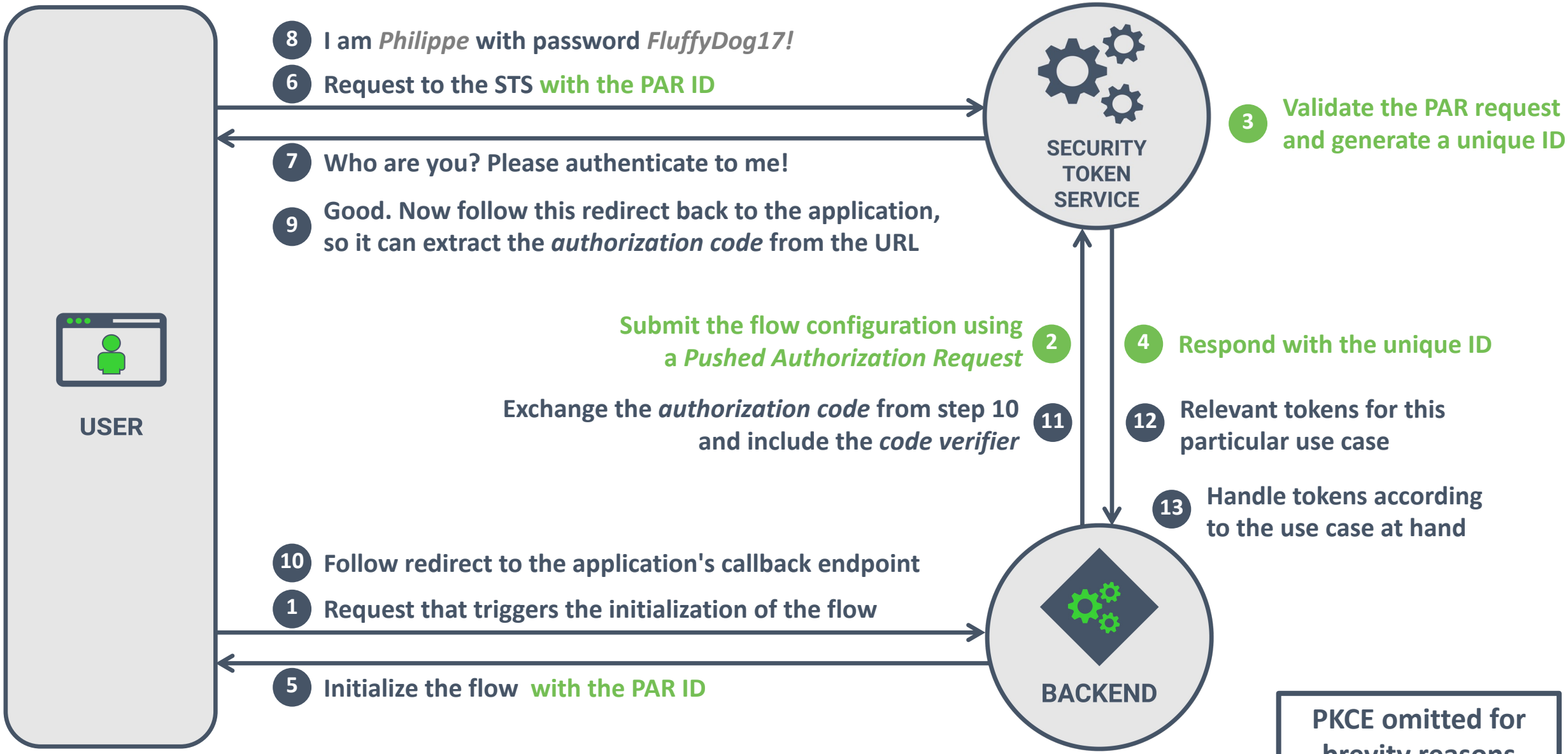
T. Lodderstedt
yes.com
B. Campbell
Ping Identity
N. Sakimura
NAT.Consulting
D. Tonge
Moneyhub Financial
Technology
F. Skokan
Auth0

OAuth 2.0 Pushed Authorization Requests

Abstract

This document defines the pushed authorization request (PAR) endpoint, which allows clients to push the payload of an OAuth 2.0 authorization request to the authorization server via a direct request and provides them with a request URI that is used as reference to the data in a subsequent call to the authorization endpoint.¶

THE *AUTHORIZATION CODE* FLOW WITH PAR



2 *The request to push the data of the authorization request*

```
1  POST /oauth/par
2
3   response_type=code
4  &scope=openid profile email
5  &client_id=FN983CEYgx4mdUg3NKNKHjwfNAL5Fb42
6  &redirect_uri=https://restograde.com/callback
7  &code_challenge=29K8tipblinCeP … HZ1PqLVxd9s
8  &code_challenge_method=S256
9  &client_secret=6ODRv0g…OVOSWI
```

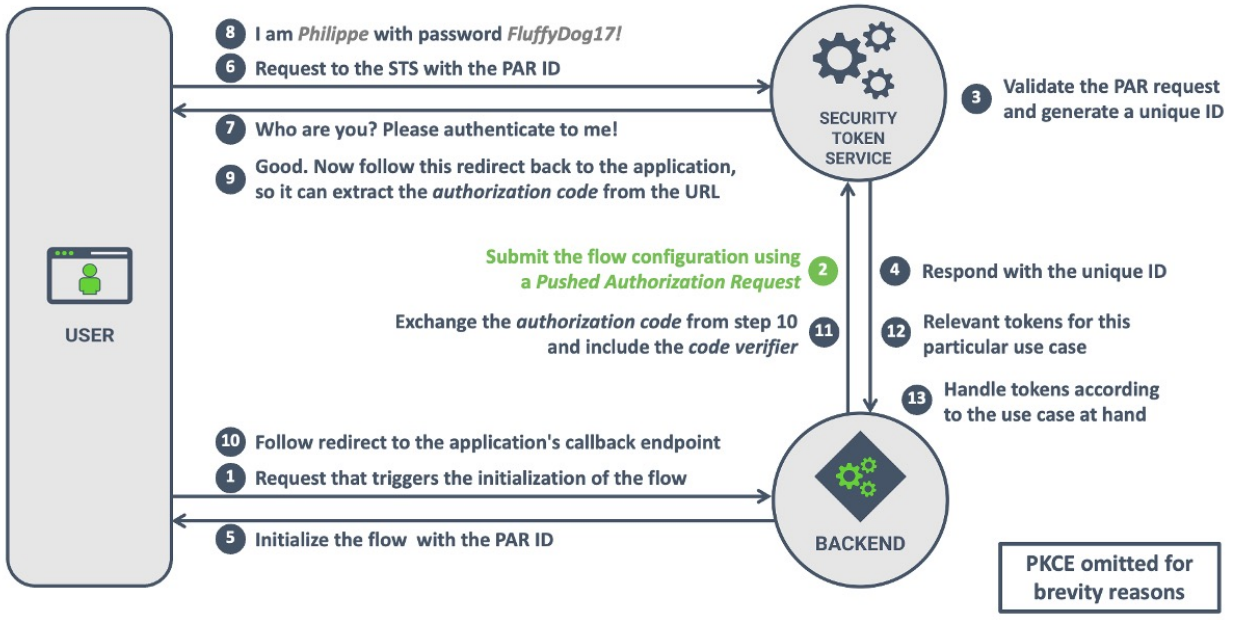
The STS supports a new PAR endpoint

The configuration of the flow

The client authenticates when using PAR

The STS can now validate the flow parameters before the actual flow has even started.

THE *AUTHORIZATION CODE* FLOW WITH PAR

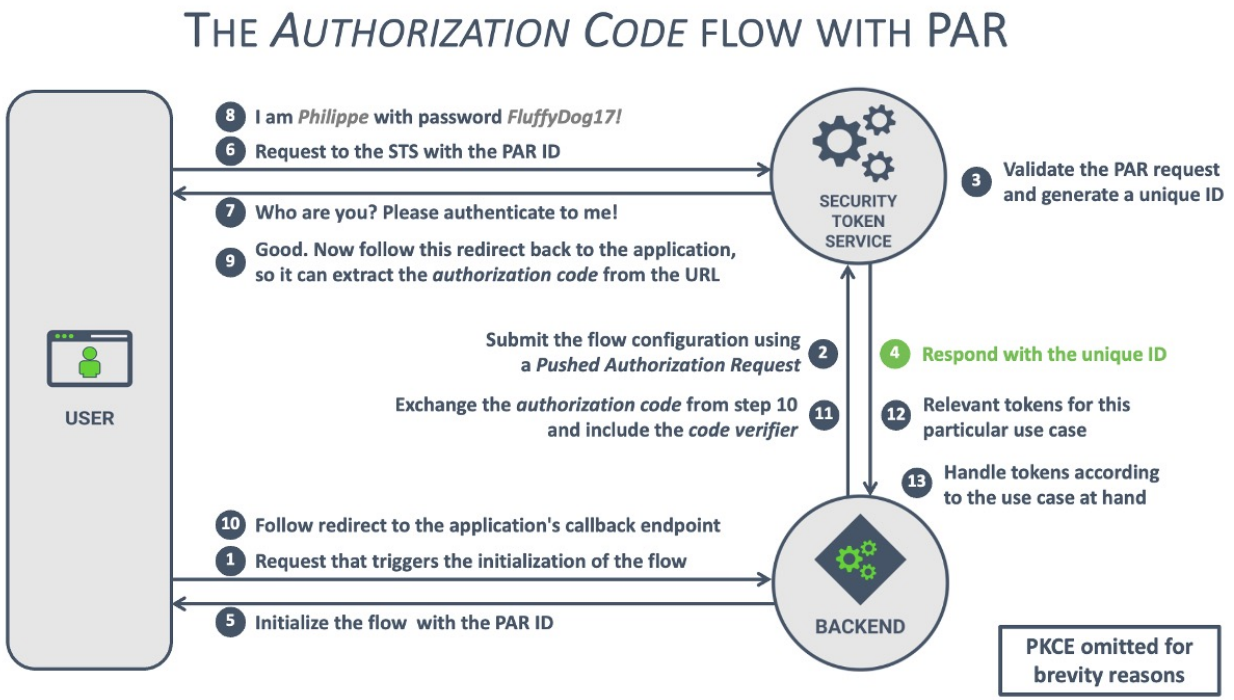


4 *The response to the PAR request*

```
1  {
2    "request_uri": "urn:ietf:params:oauth:request_uri:
3                    6esc_11ACC5bwc014ltc14eY22c",
4    "expires_in": 60
5  }
```

The ID for this PAR configuration

The lifetime of this PAR config



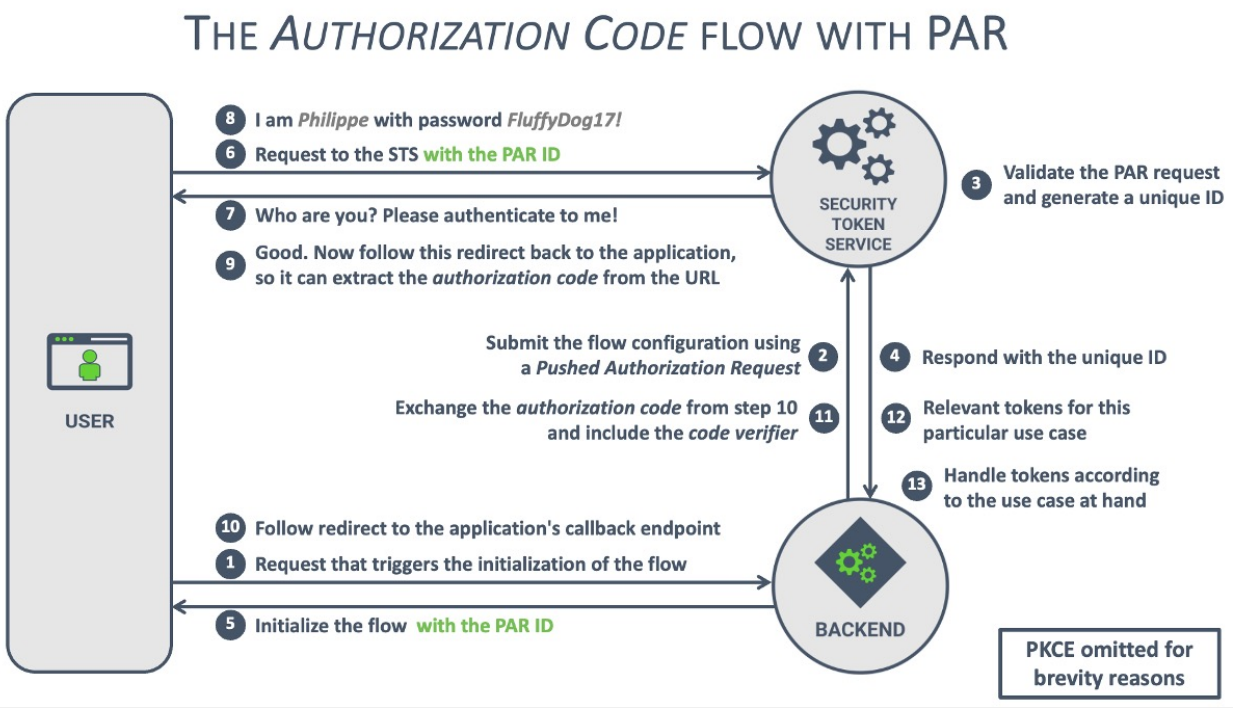
5 6 *The authorization request with the PAR ID (a redirect to the STS)*

```
1  https://sts.restograde.com/authorize
2    ?client_id=lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv
3    &request_uri=urn:ietf:params:oauth:request_uri:6e11ACC5
4                bwc014ltc14eY22c
```

Indicates the client making the request

The ID provided by the STS in step 4

The PAR identifier is formatted as a URI and refers to a configuration that was pushed to the STS by the client before initializing the flow

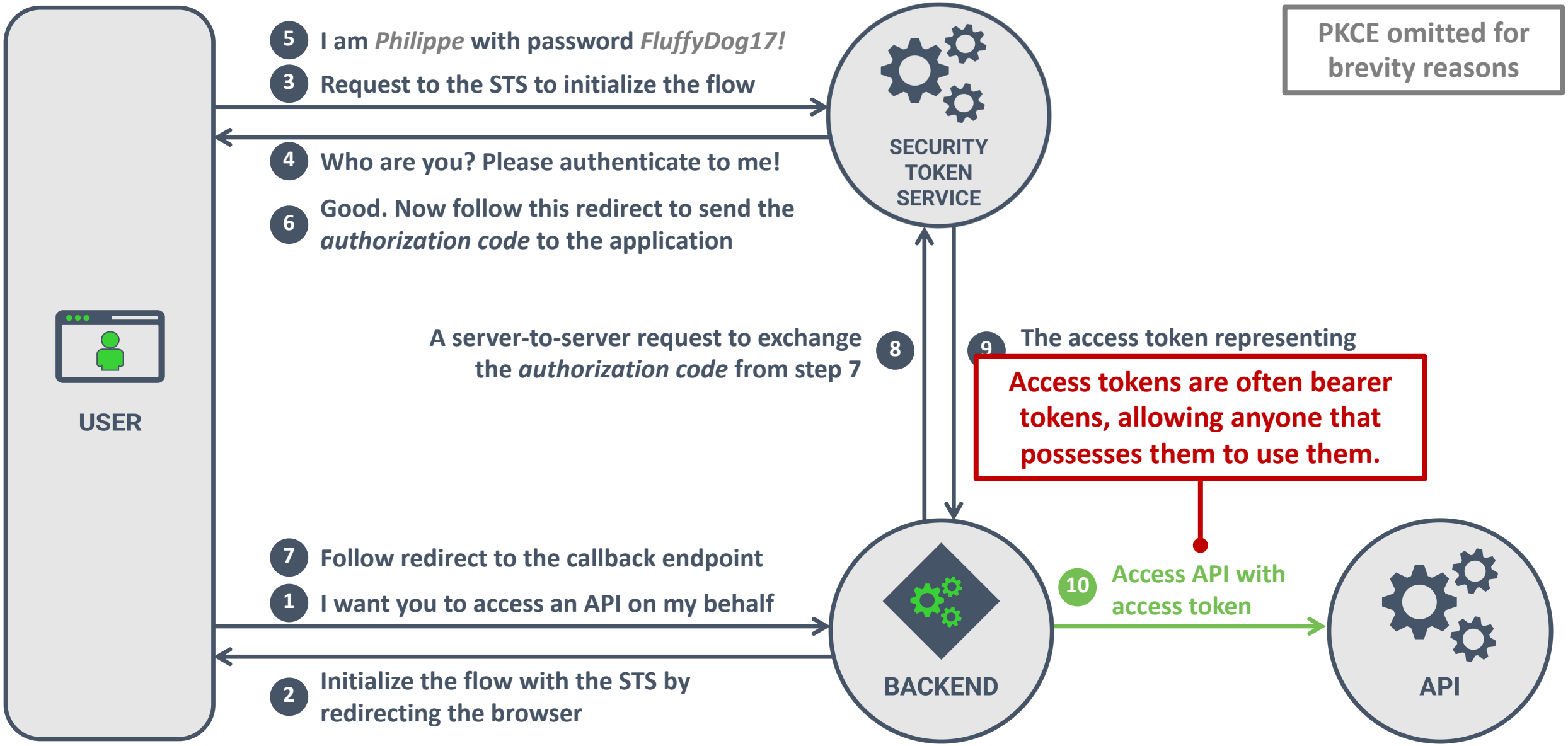


PAR in action

When using PAR, make sure clients are no longer allowed to use a regular Authorization Code flow

PAR is an addition to OAuth 2.1 and is quickly becoming the default mode for backend clients

Augmenting the token security of OAuth 2.1


PKCE omitted for brevity reasons
USER
5 I am *Philippe* with password *FluffyDog17!*
3 Request to the STS to initialize the flow
4 Who are you? Please authenticate to me!
6 Good. Now follow this redirect to send the *authorization code* to the application
SECURITY TOKEN SERVICE
A server-to-server request to exchange the *authorization code* from step 7
8
9 The access token representing
Access tokens are often bearer tokens, allowing anyone that possesses them to use them.
7 Follow redirect to the callback endpoint
1 I want you to access an API on my behalf
2 Initialize the flow with the STS by redirecting the browser
BACKEND
10 Access API with access token
API

Proof-of-possession mechanisms transform bearer tokens into sender-constrained tokens

Internet Engineering Task Force (IETF)
Request for Comments: 8705
Category: Standards Track
Published: February 2020
ISSN: 2070-1721

B. Campbell
Ping Identity
J. Bradley
Yubico
N. Sakimura
Nomura Research Institute
T. Lodderstedt
YES.com AG

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

Abstract

This document describes OAuth client authentication and certificate-bound access and refresh tokens using mutual Transport Layer Security (TLS) authentication with X.509 certificates. OAuth clients are provided a mechanism for authentication to the authorization server using mutual TLS, based on either self-signed certificates or public key infrastructure (PKI). OAuth authorization servers are provided a mechanism for binding access tokens to a client's mutual-TLS certificate, and OAuth protected resources are provided a method for ensuring that such an access token presented to it was issued to the client presenting the token.

mTLS is not always practical to use, especially in restricted application environments or complex infrastructures

DPoP offers an application-level alternative to enable sender-constrained tokens

Internet Engineering Task Force (IETF)
Request for Comments: 9449
Category: Standards Track
Published: September 2023
ISSN: 2070-1721

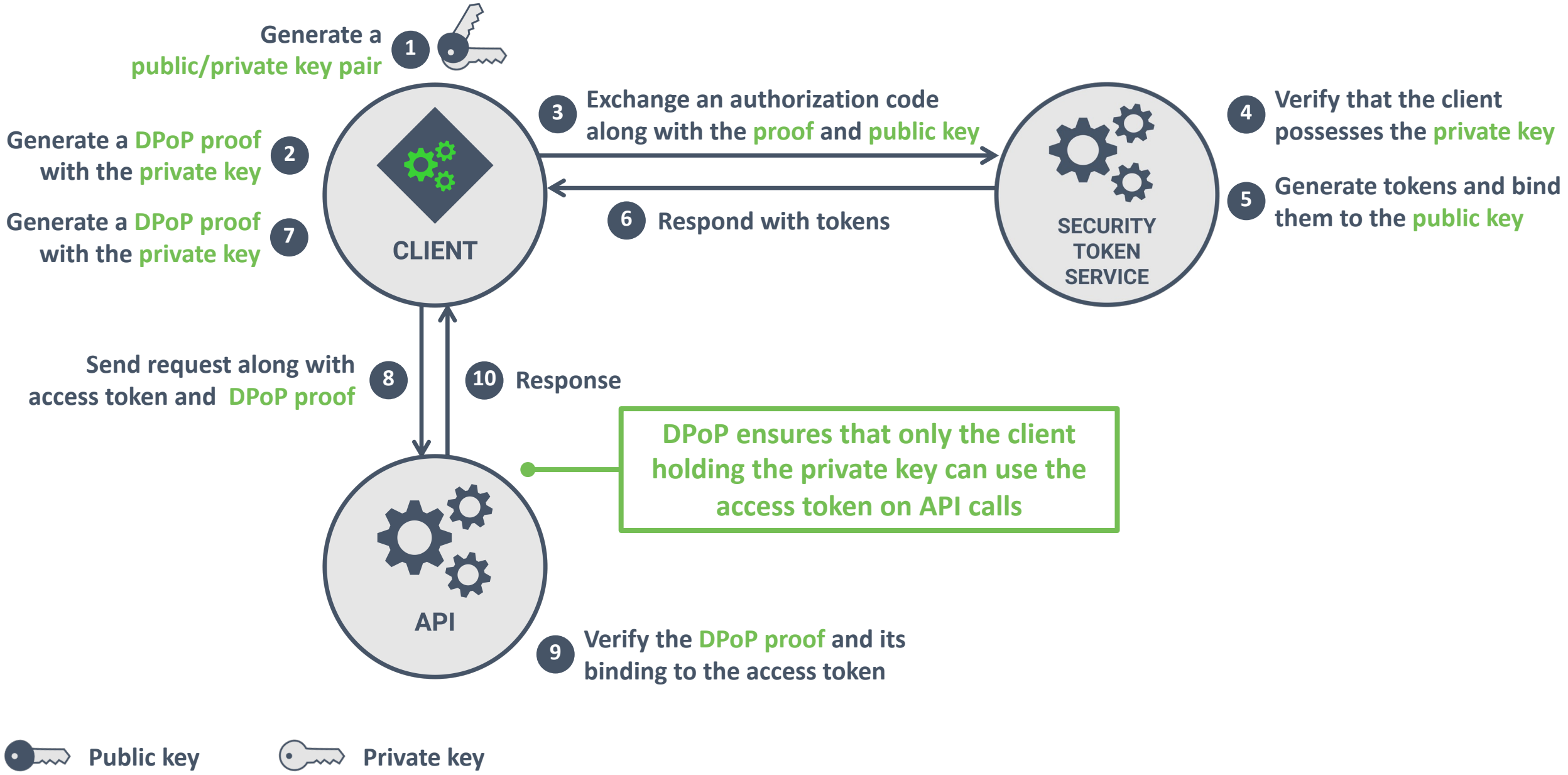
D. Fett
Authlete
B. Campbell
Ping Identity
J. Bradley
Yubico
T. Lodderstedt
Tuconic
M. Jones
Self-Issued Consulting
D. Waite
Ping Identity

OAuth 2.0 Demonstrating Proof of Possession (DPoP)

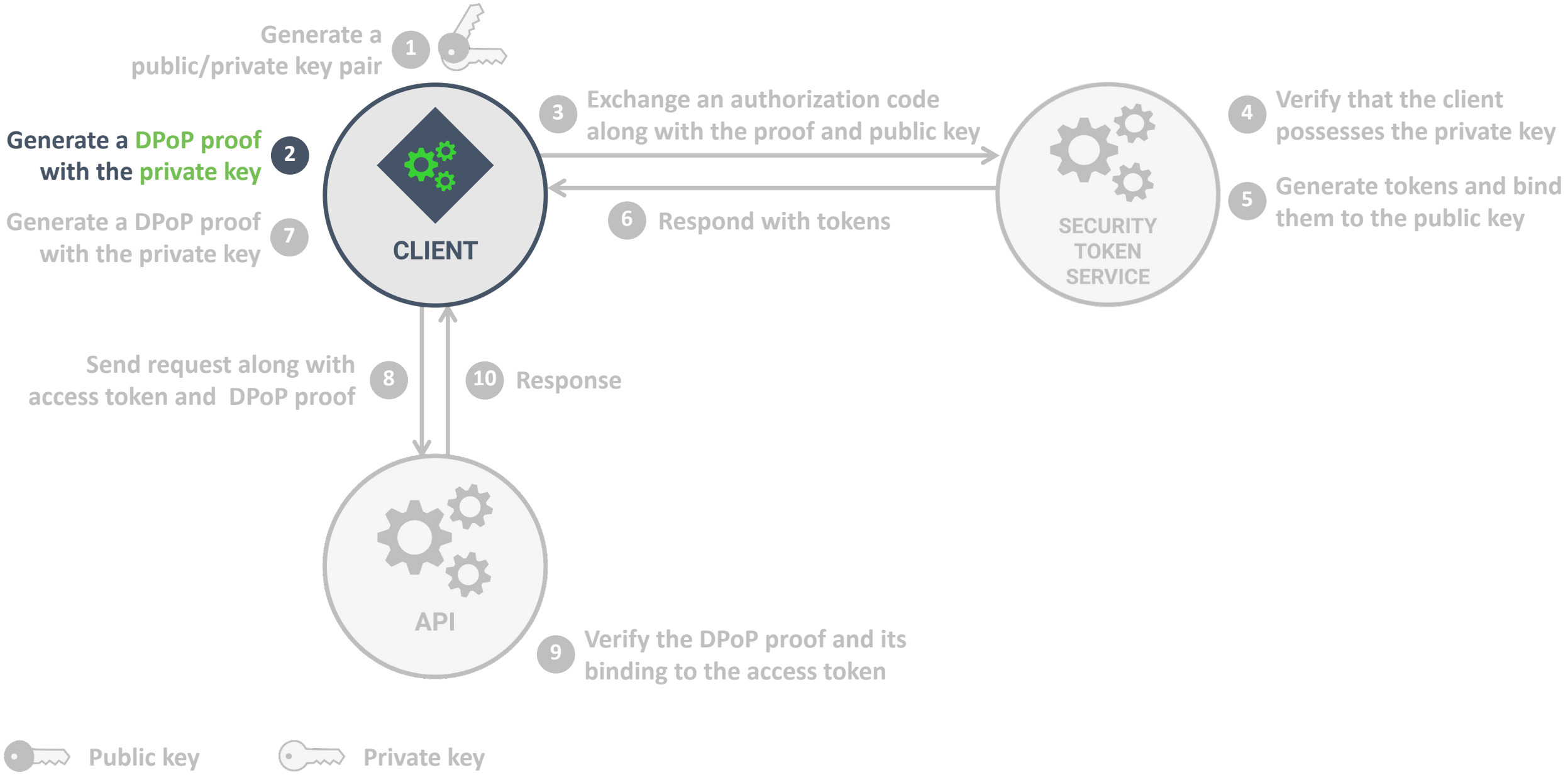
Abstract

This document describes a mechanism for sender-constraining OAuth 2.0 tokens via a proof-of-possession mechanism on the application level. This mechanism allows for the detection of replay attacks with access and refresh tokens.

The concept of DPoP



THE CONCEPT OF DPOP



2 *The header and payload of the DPoP proof JWT*

The JWT is signed by the client's private key

```
// Header
{
  "typ": "dpop+jwt",
  "alg": "ES256",
  "jwk": { ... public key ... }
}

//Payload
{
  "jti": "-BwC3ESc6acc2lTc",
  "htm": "POST",
  "htu": "https://sts.restograde.com/token",
  "iat": 1562262616
}
```

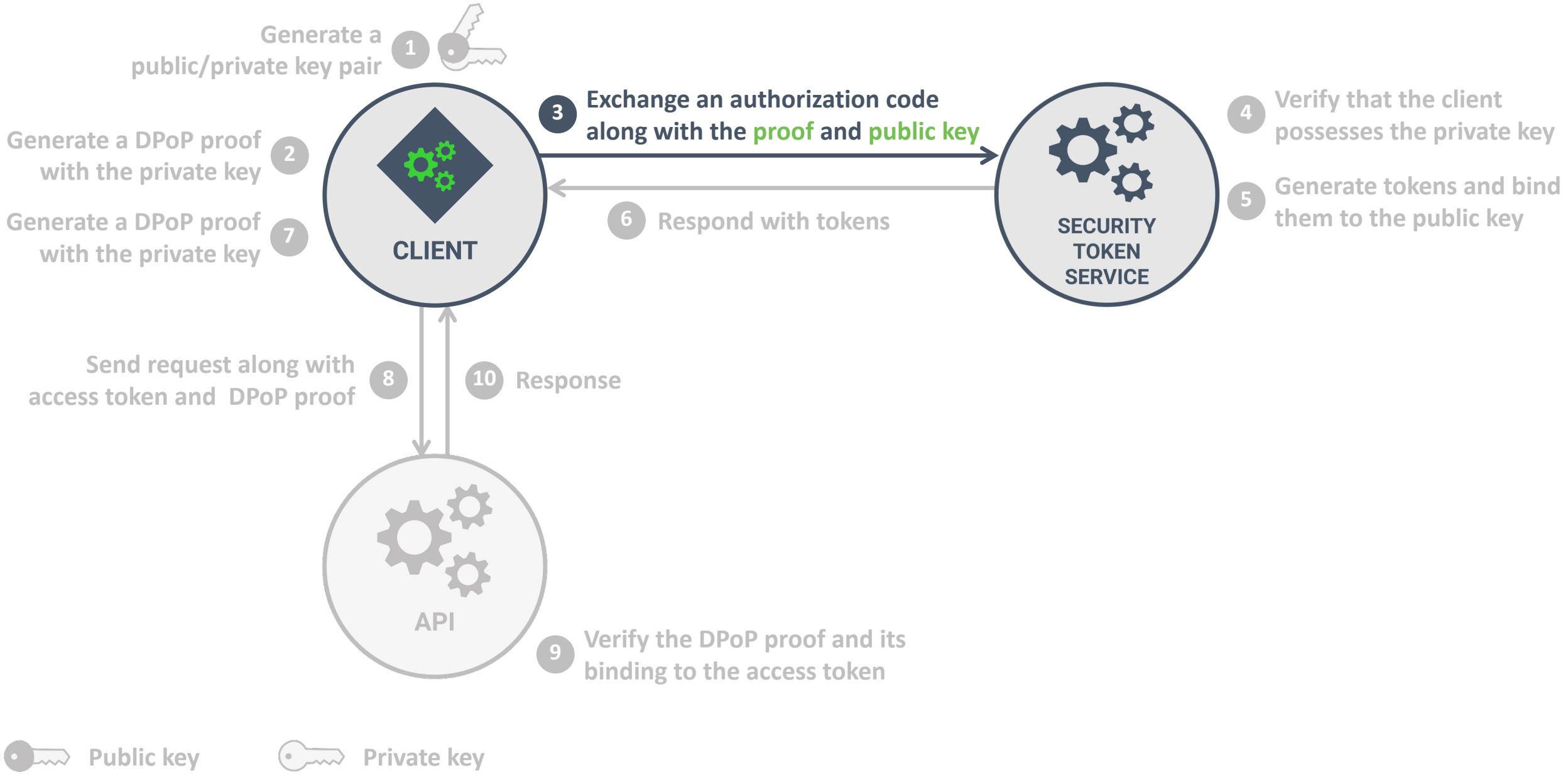
The token type indicates a JWT DPoP proof

The client's public key is part of the header

A unique identifier generated by the client

This DPoP proof is for a token request to the STS

THE CONCEPT OF DPOP



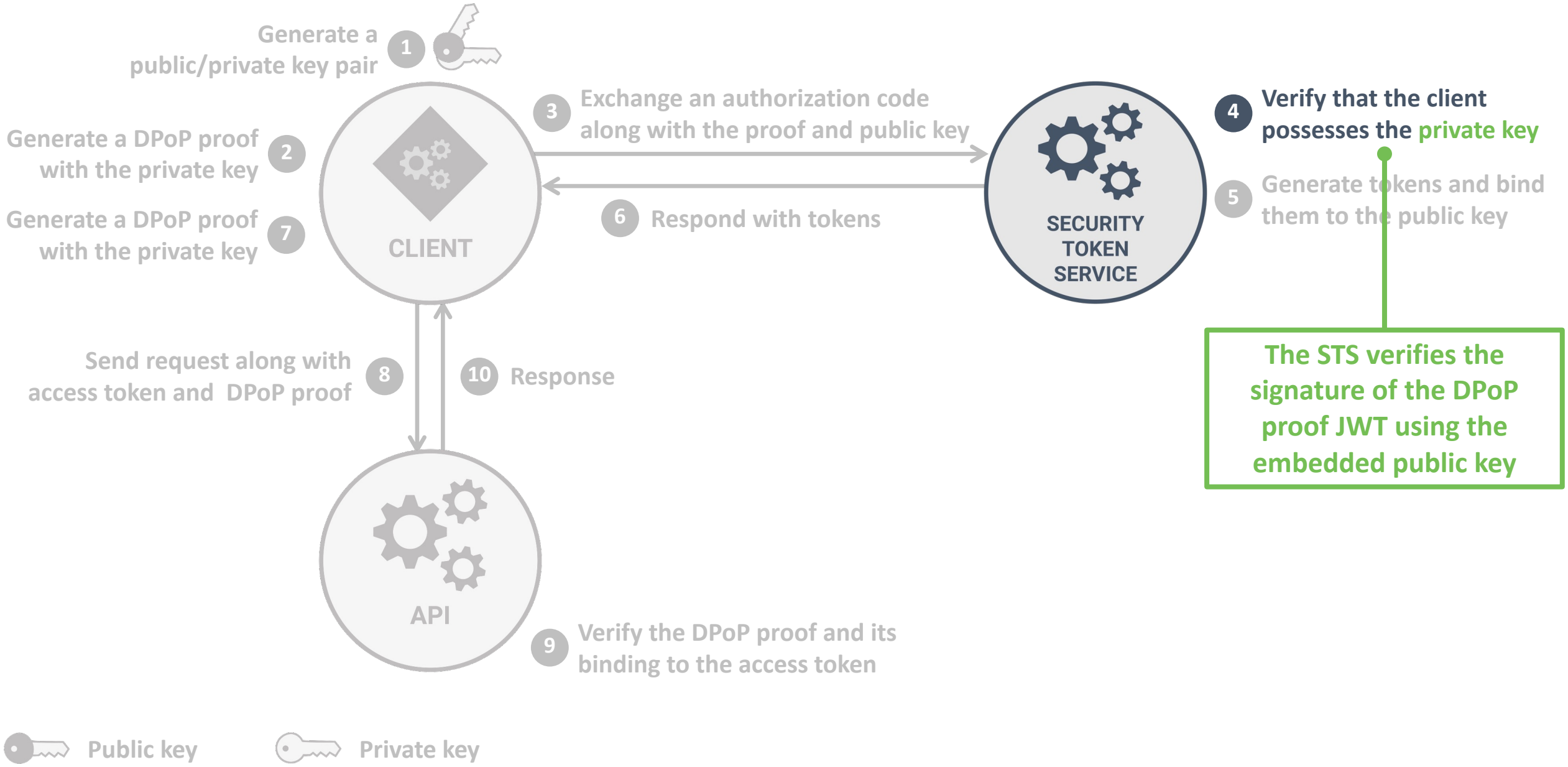
3 *The token request with the authorization code and DPoP proof*

A traditional request to the token endpoint

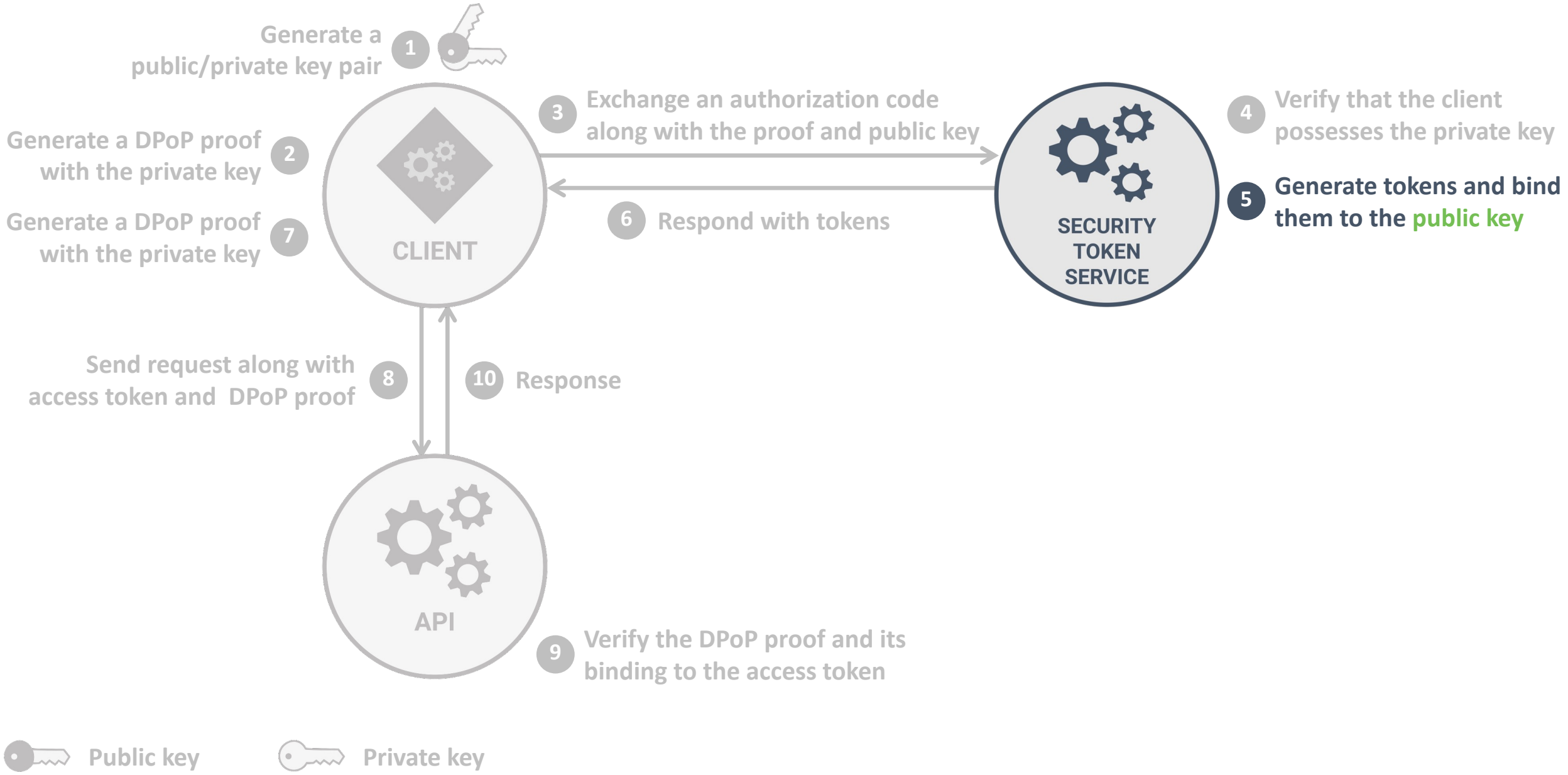
```
POST /token
Host: sts.restograde.com
DPoP: eyJ0eXAiOiJkcG9 … fbV37xRZT3Lg
 
 grant_type=authorization_code
&client_id=lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv
&redirect_uri=https://app.restograde.com/callback
&code=SplxlOBeZQQYbYS6WxSbIA
&code_verifier=lT5q6nbPQRtdj…~IUdkErVDFG.fF4z7CzCxo
```

The DPoP proof generated in step 2

The concept of DPoP



THE CONCEPT OF DPOP



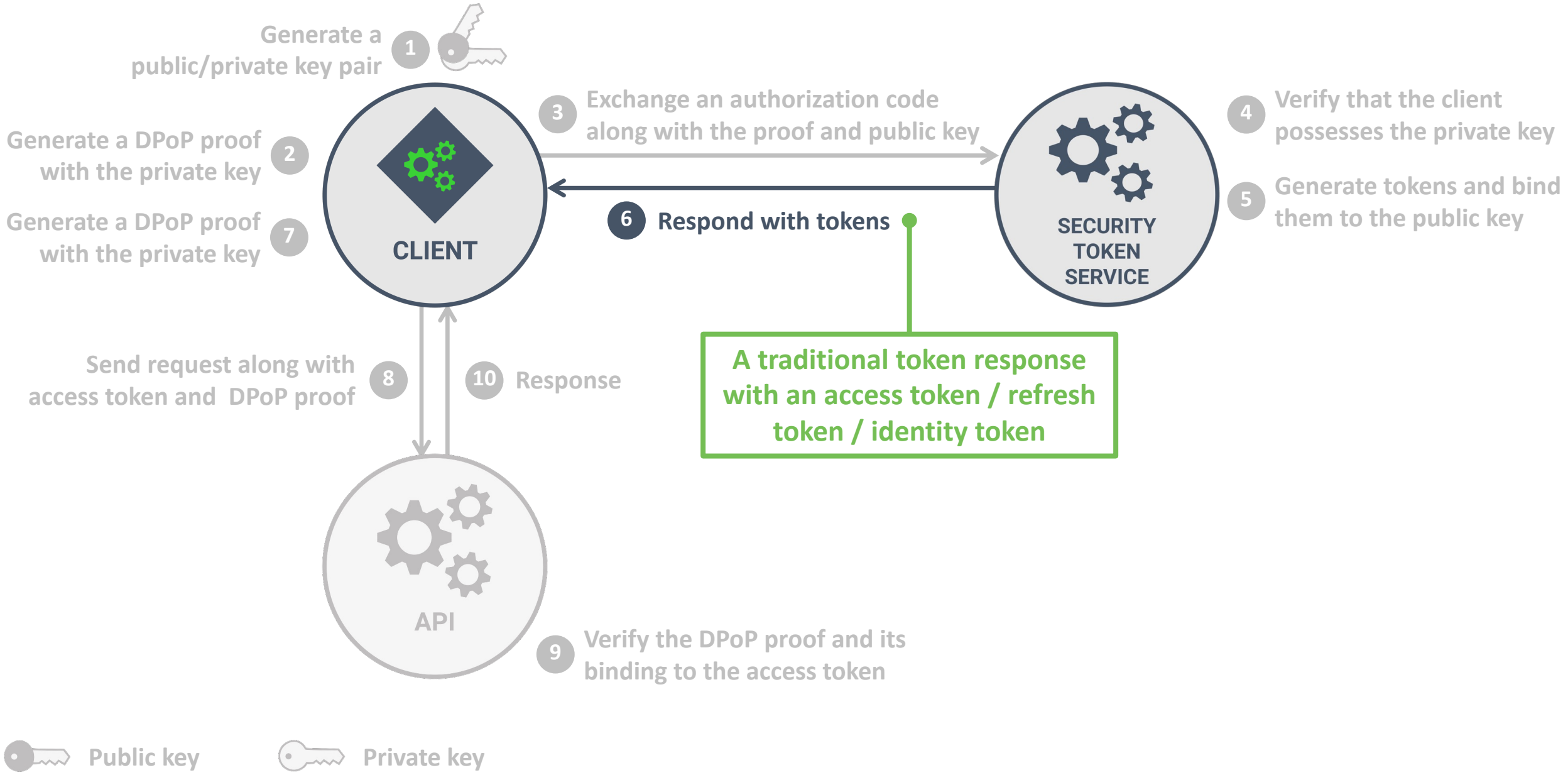
5 *The generated access token bound to the client's public key*

```
{
  "iss": "https://sts.restograde.com",
  "aud": "https://api.restograde.com",
  "client_id": "lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv",
  "sub": "2262430d-c9cb-484f-9770-805893ff9518",
  …
  "cnf": {
    "jkt": "bwcK0esc3ACC3DB2Y … 8o9ltc05O89jdN-dg2"
  }
}
```

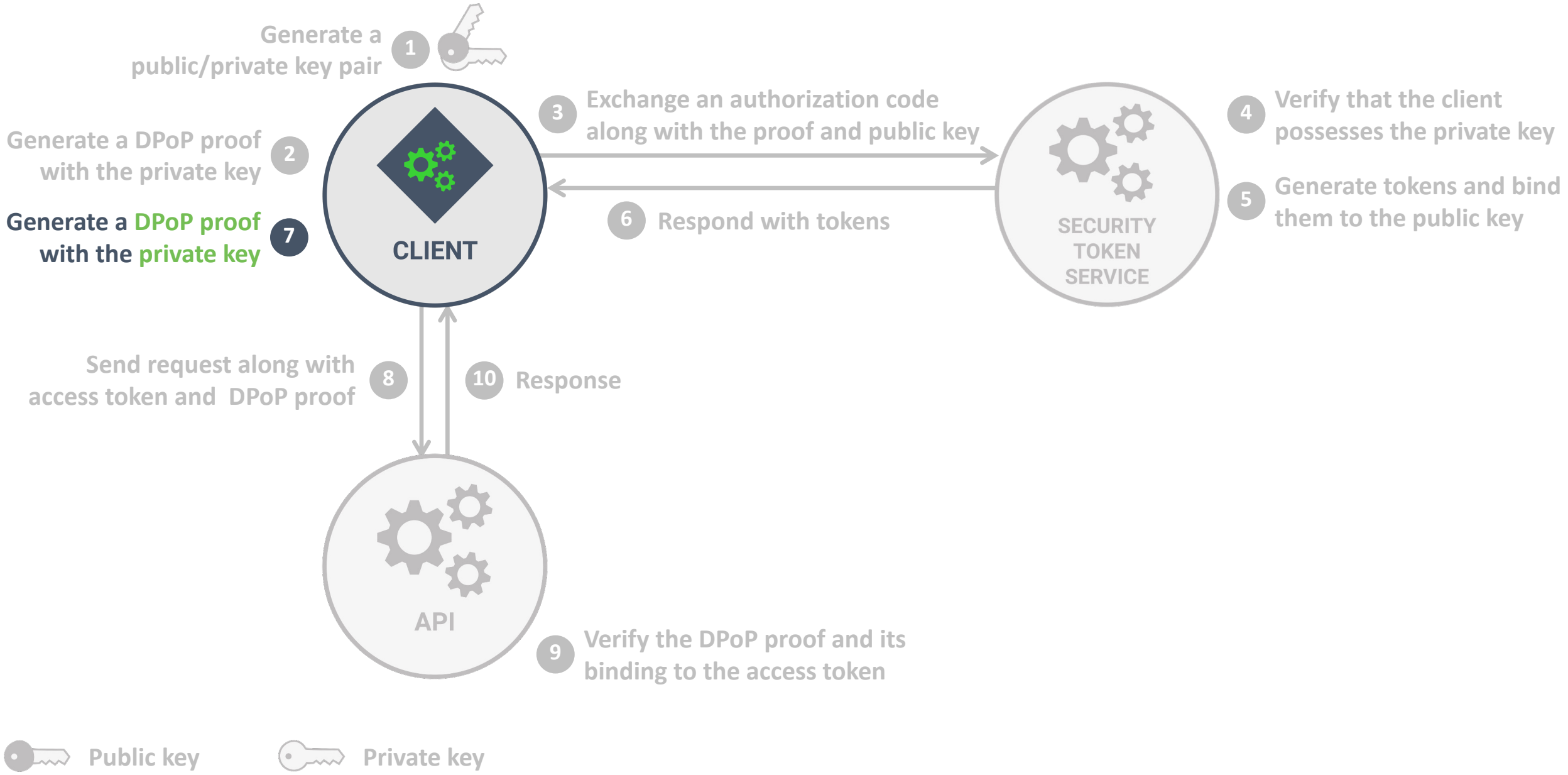
A traditional self-contained access token

The fingerprint of the client's public key

THE CONCEPT OF DPOP



The concept of DPoP



7 *The header and payload of the DPoP proof JWT*

The JWT is signed by the client's private key

```
// Header
{
  "typ": "dpop+jwt",
  "alg": "ES256",
  "jwk": { ... public key ... }
}

//Payload
{
  "jti": "e1j3V_bKic8-LAEB",
  "htm": "GET",
  "htu": "https://api.restograde.com/reviews",
  "iat": 1562262618
  "ath": "fUHyO2r2Z3DZ…53EsNrWBb0xWXoaN",
}
```

The token type indicates a JWT DPoP proof

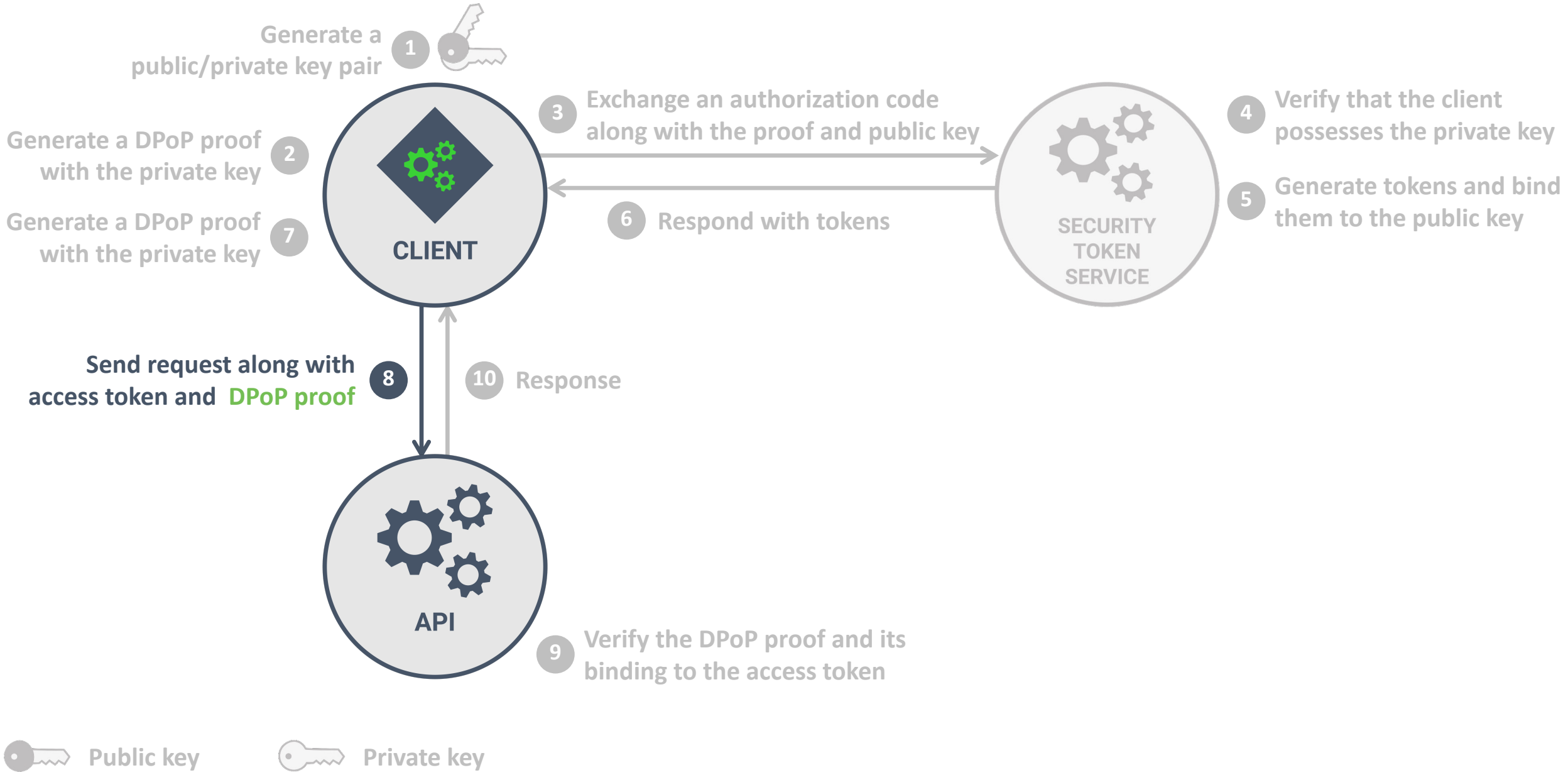
The client's public key is part of the header

A unique identifier generated by the client

This DPoP proof is for a GET request to an API endpoint

The hash of the access token associated with this proof

The concept of DPoP



8 *The request to the API with the access token and DPoP proof JWT*

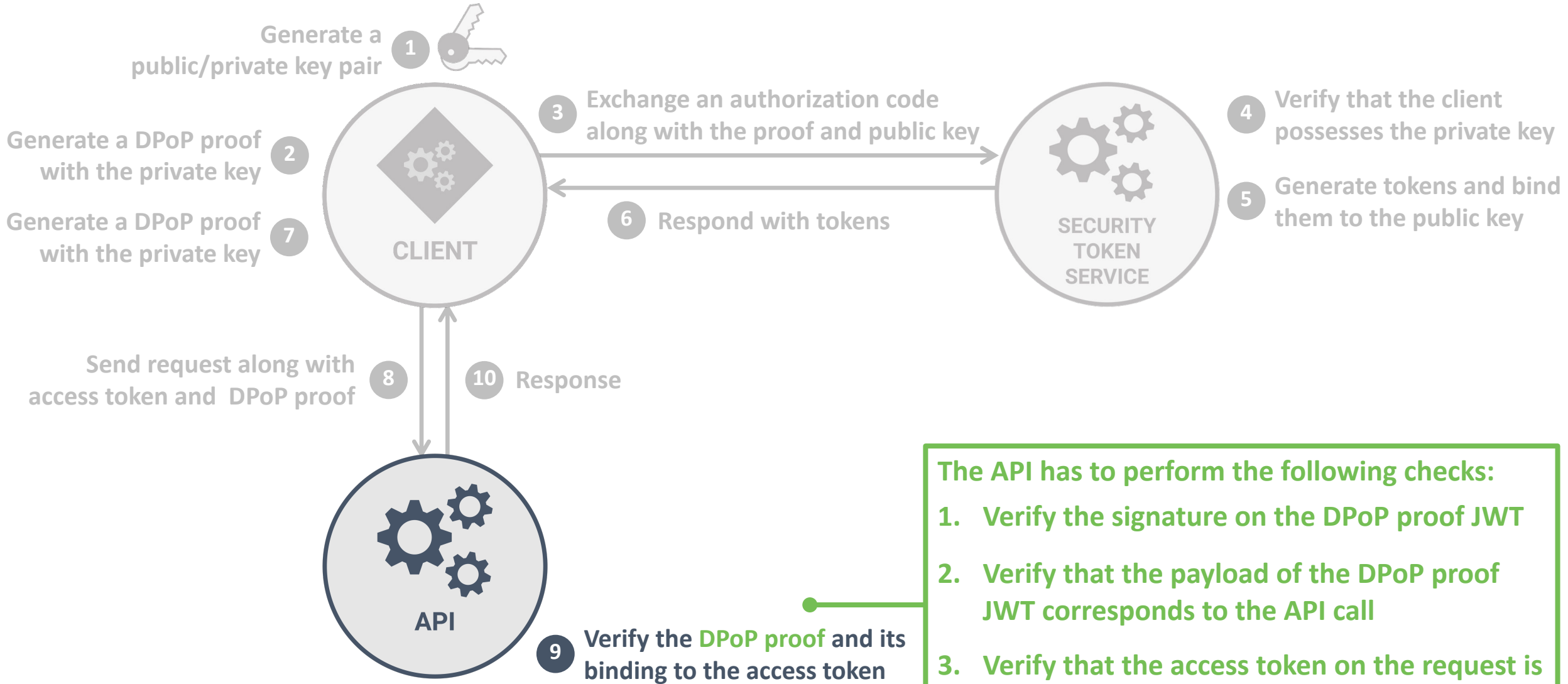
```
1  GET /reviews
2  Host: api.restograde.com
3  Authorization: DPoP eyJ0f37x3LgRZTbV … eRAiOiJkcG9
4  DPoP: eyJ0eXAiOiJkcG9 … fbV37xRZT3Lg
```

The access token issued by the STS

The DPoP proof JWT generated in step 7

The ***Authorization*** header no longer carries a **bearer token**, but a **DPoP token**

THE CONCEPT OF DPOP



The API has to perform the following checks:

1. **Verify the signature on the DPoP proof JWT**
2. **Verify that the payload of the DPoP proof JWT corresponds to the API call**
3. **Verify that the access token on the request is linked to the public key used to sign the DPoP proof JWT**

The API must actively check the absence of a *cnf* claim in bearer tokens!

Key takeaways

1. OAuth 2.1 is like OAuth 2.0, but cleaned up and pretty
2. User-facing applications rely on the authorization code flow
3. PAR and DPoP are essential mechanisms to increase security

Thank you!

Need training or security guidance?
Reach out to discuss how I can help

https://pragmaticwebsecurity.com