SUPERCHARGING OAUTH 2.0 SECURITY

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: 8707
Category: Standards Track
Published: February 2020
ISSN: 2070-1721

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

Resource Indicators for OAuth 2.0

Abstract

This document specifies an extension to the ...ization Framework ...

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. Skoka
Auth

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. 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.

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

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

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

Abstract

The authorization request in OAuth 2.0 described in RFC 6749 utilizes query parameter serialization, which means that authorization request parameters are encoded in the URI of the request and sent through user agents such as web browsers. While it is easy to implement, it means that a) the communication through the user agents is not integrity protected and thus, the ... b) the ... use of ... been put ... ers in a ... signed ... ryption ... ttained.

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

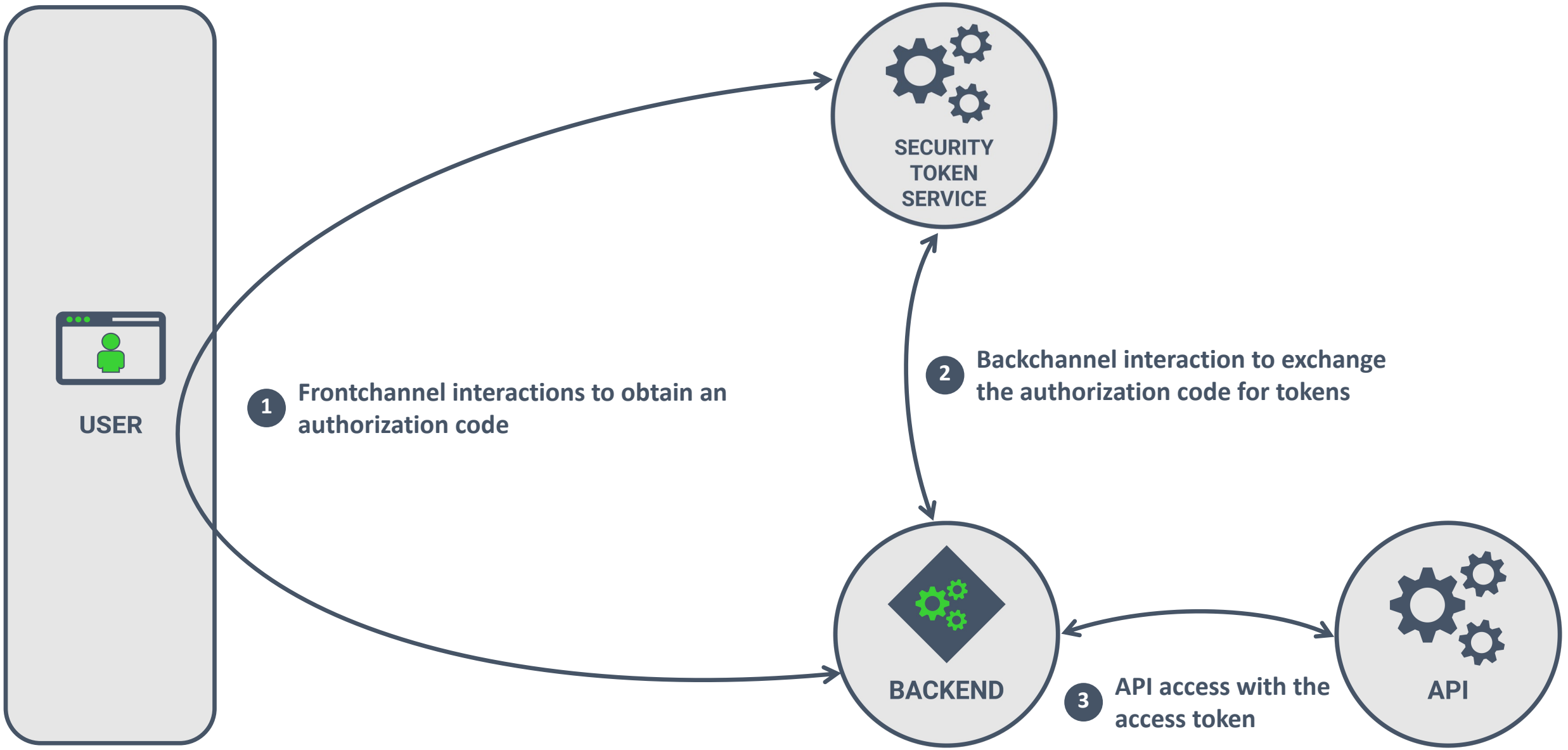
D. Fett
Authlete
B. Campbell
Ping Identity
J. Bradley
Yubico
T. Lodderstedt
Tuconic
M. Jones
Self-Issued Consulting
D. Waite
Ping Identity

OAuth 2.0 Demonstrating Proof of Possession (DPoP)

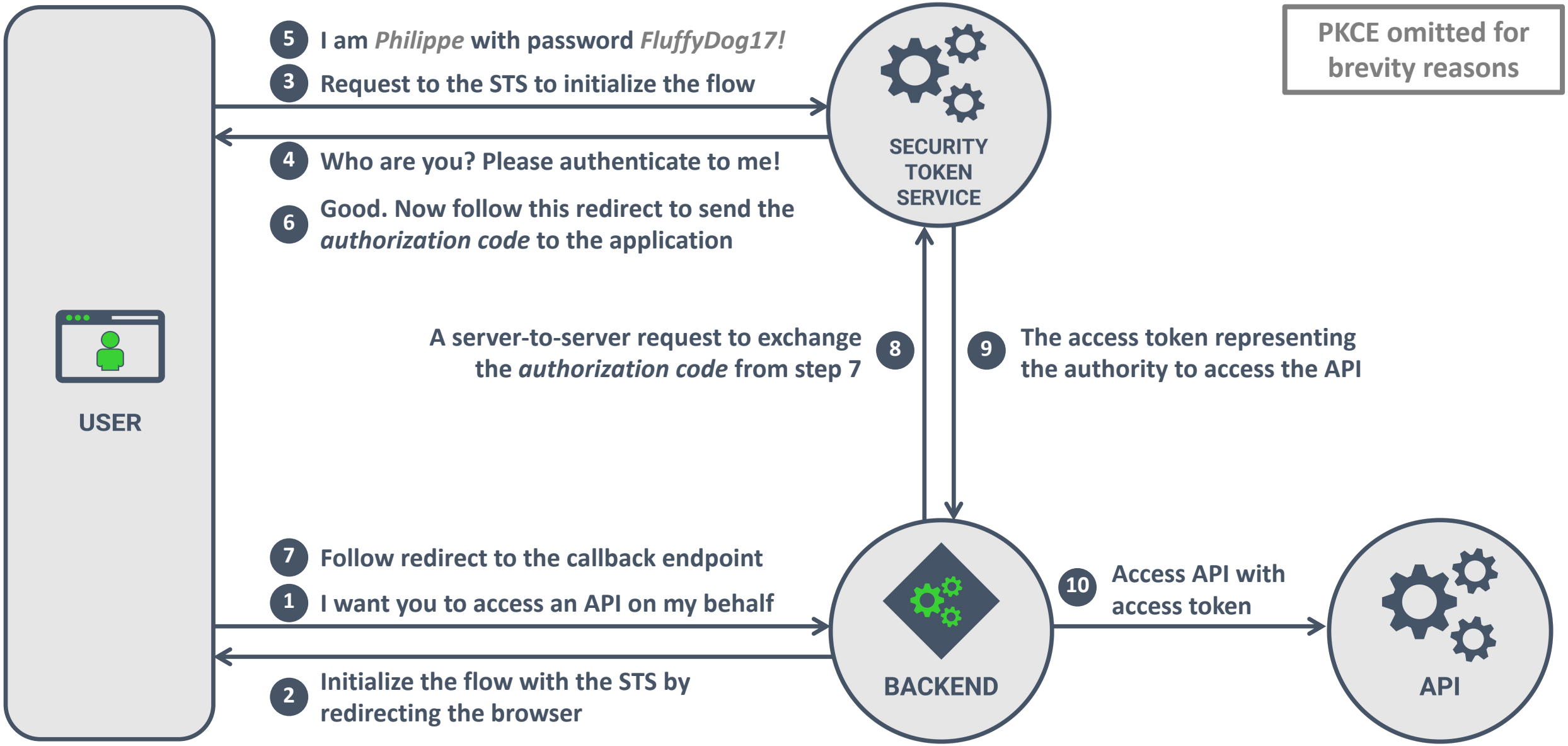
Abstract

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.

High-level overview of the *Authorization Code* flow



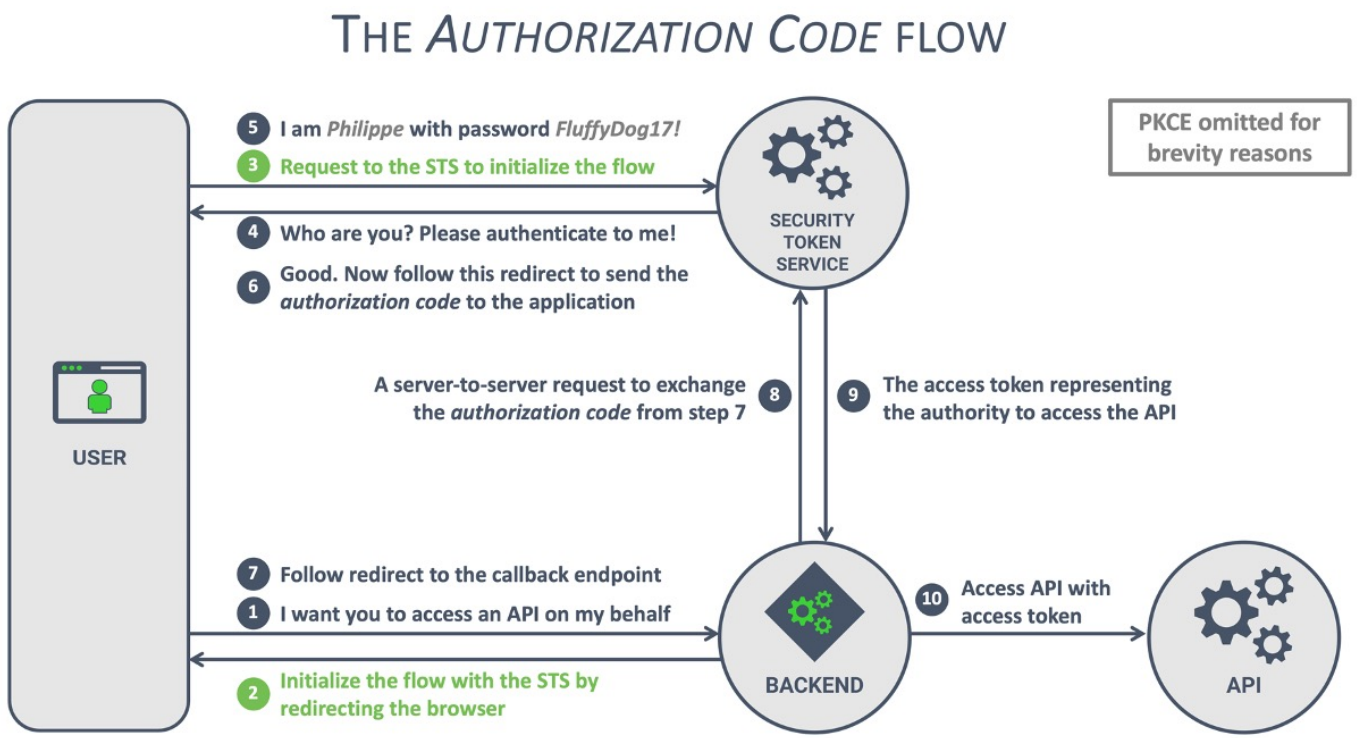
THE *AUTHORIZATION CODE* FLOW



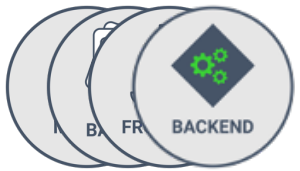
2 3 *The authorization request (a redirect to the STS)*

```
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)



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

SecAppDev

SecAppDev organizer

I help developers with security

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

https://pdr.online

Grab a copy of the slides ...

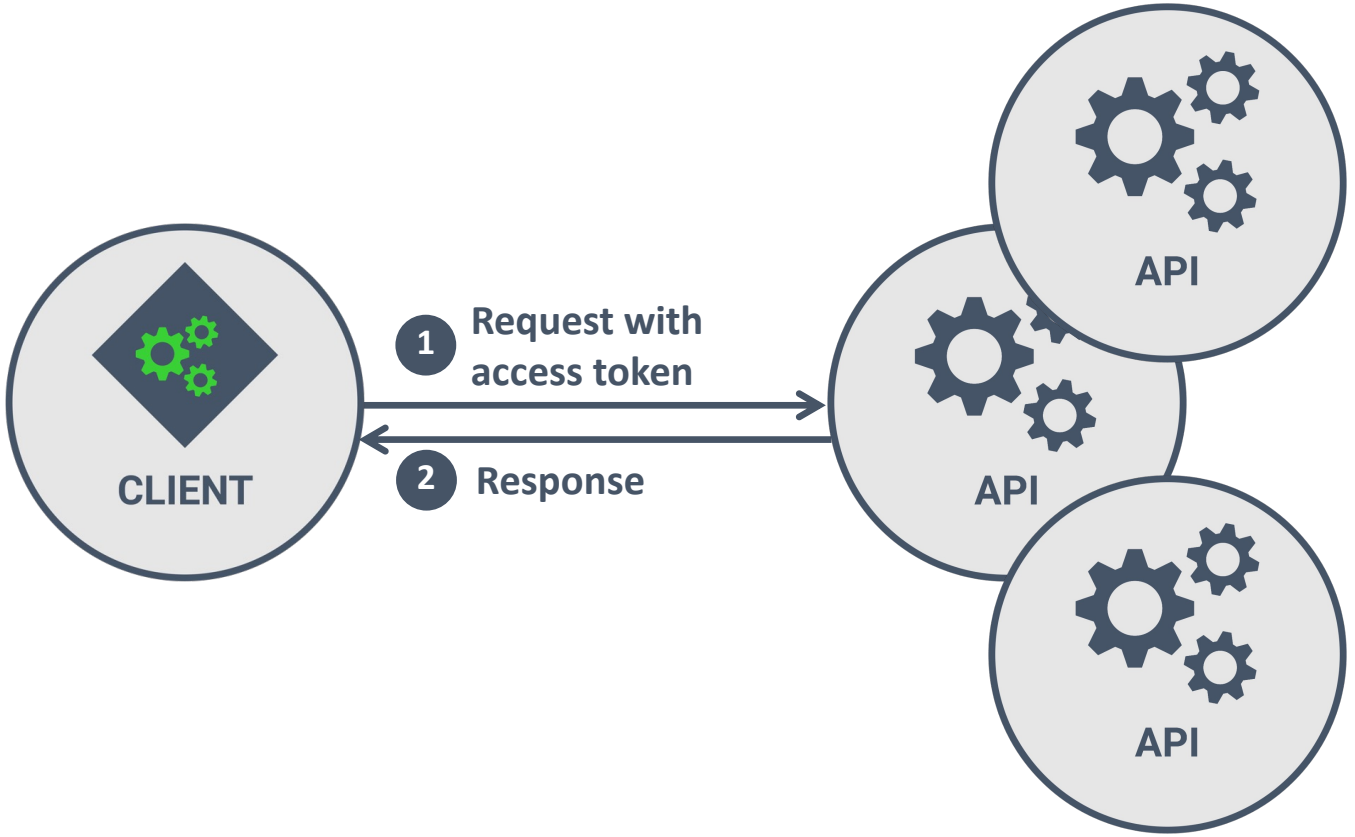
https://pragmaticwebsecurity.com/talks

/in/PhilippeDeRyck

https://infosec.exchange/@PhilippeDeRyck

Website icons created by Uniconlabs - Flaticon

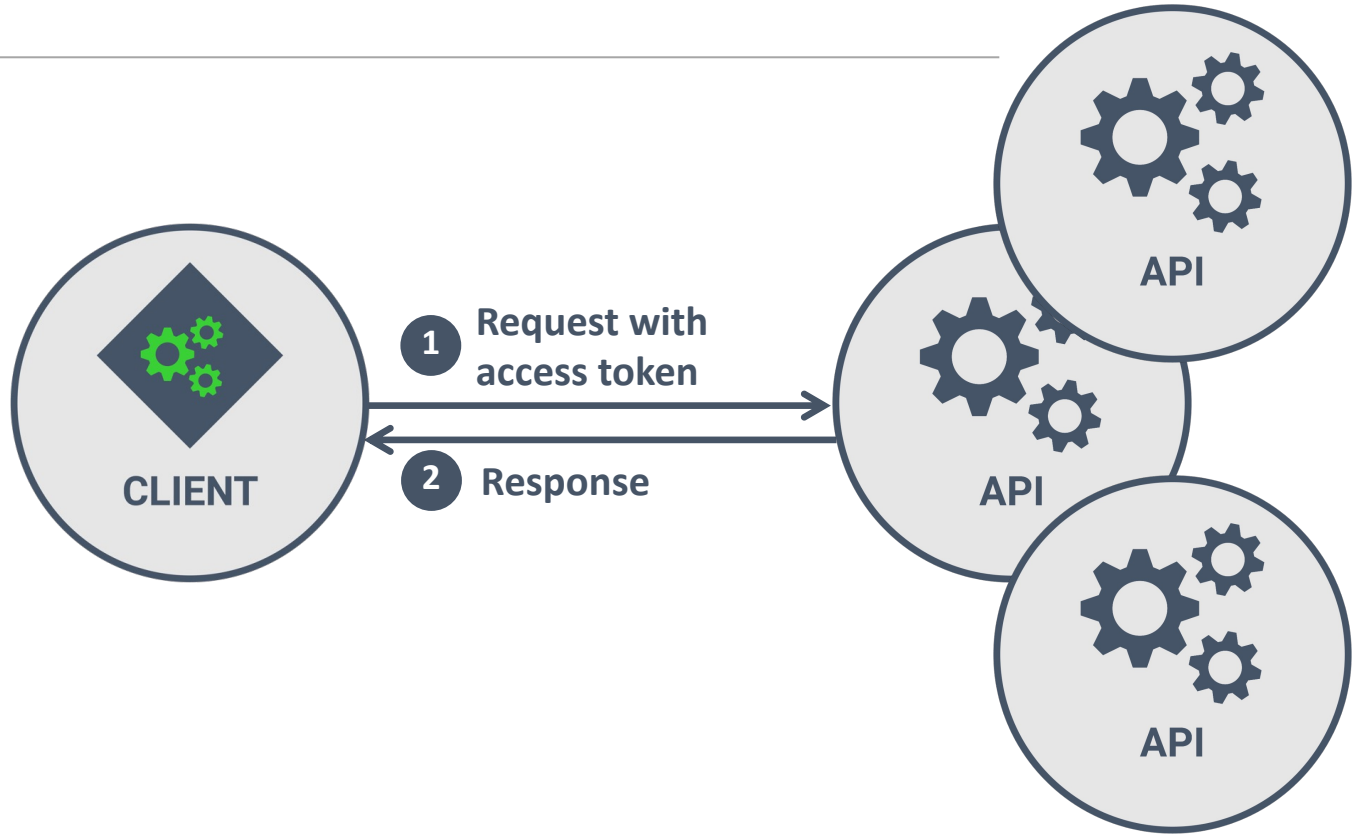
Reducing the power of access tokens


CLIENT
1
Request with
access token
2
Response
API
API
API

An access token to access three separate Restograde APIs

```
{
  "iss": "https://sts.restograde.com",
  "aud": [ "https://api.restograde.com/reviews",
           "https://api.restograde.com/restaurants",
           "https://api.restograde.com/users" ],
  "sub": "2262430d-c9cb-484f-9770-805893ff9518",
  "scope": "restaurants:read reviews:write"
}
```

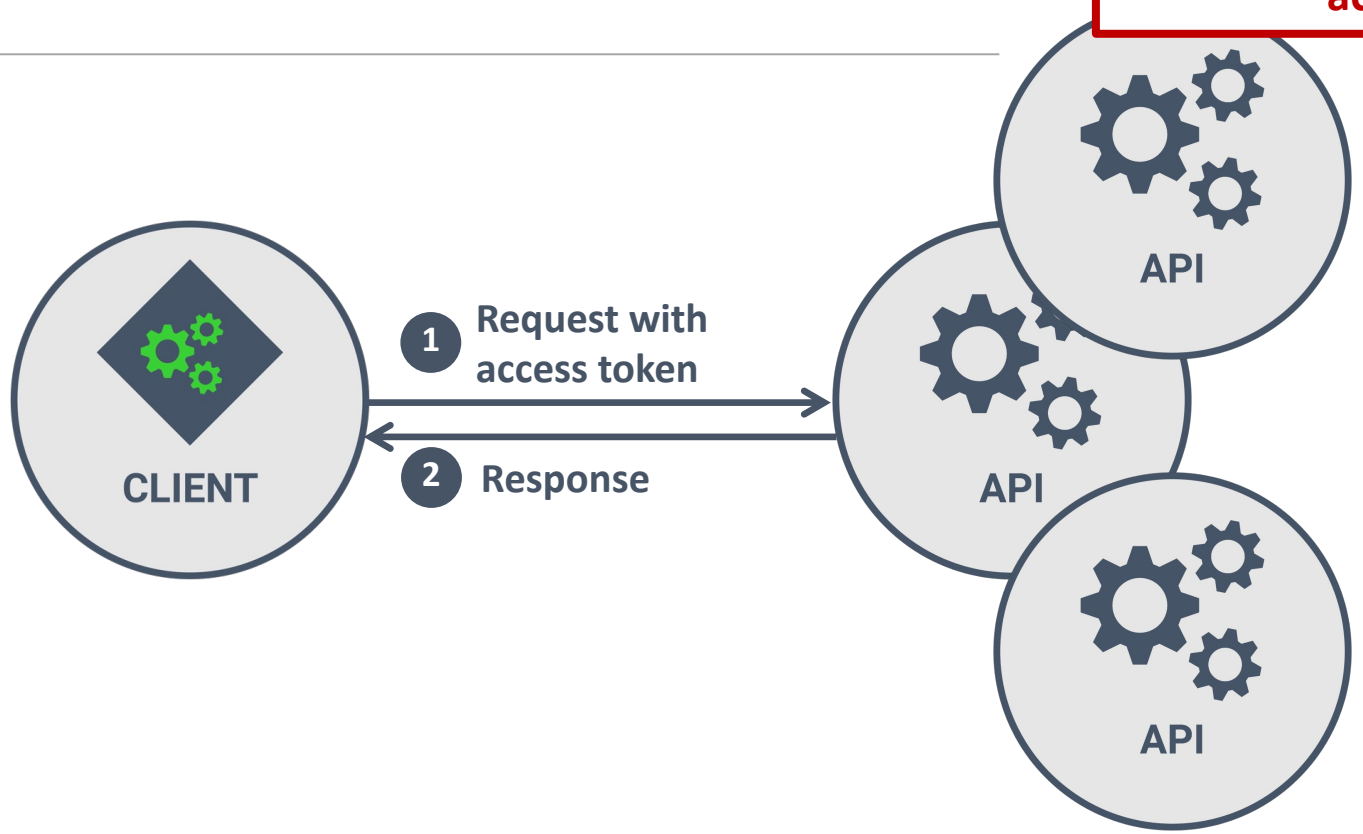
Including a full list of audiences leaks information about valid APIs to any receiver of this token



An access token to access three separate Restograde APIs

```
{
  "iss": "https://sts.restograde.com",
  "aud": [ "https://api.restograde.com/reviews",
           "https://api.restograde.com/restaurants",
           "https://api.restograde.com/users" ],
  "sub": "2262430d-c9cb-484f-9770-805893ff9518",
  "scope": "restaurants:read reviews:write"
}
```

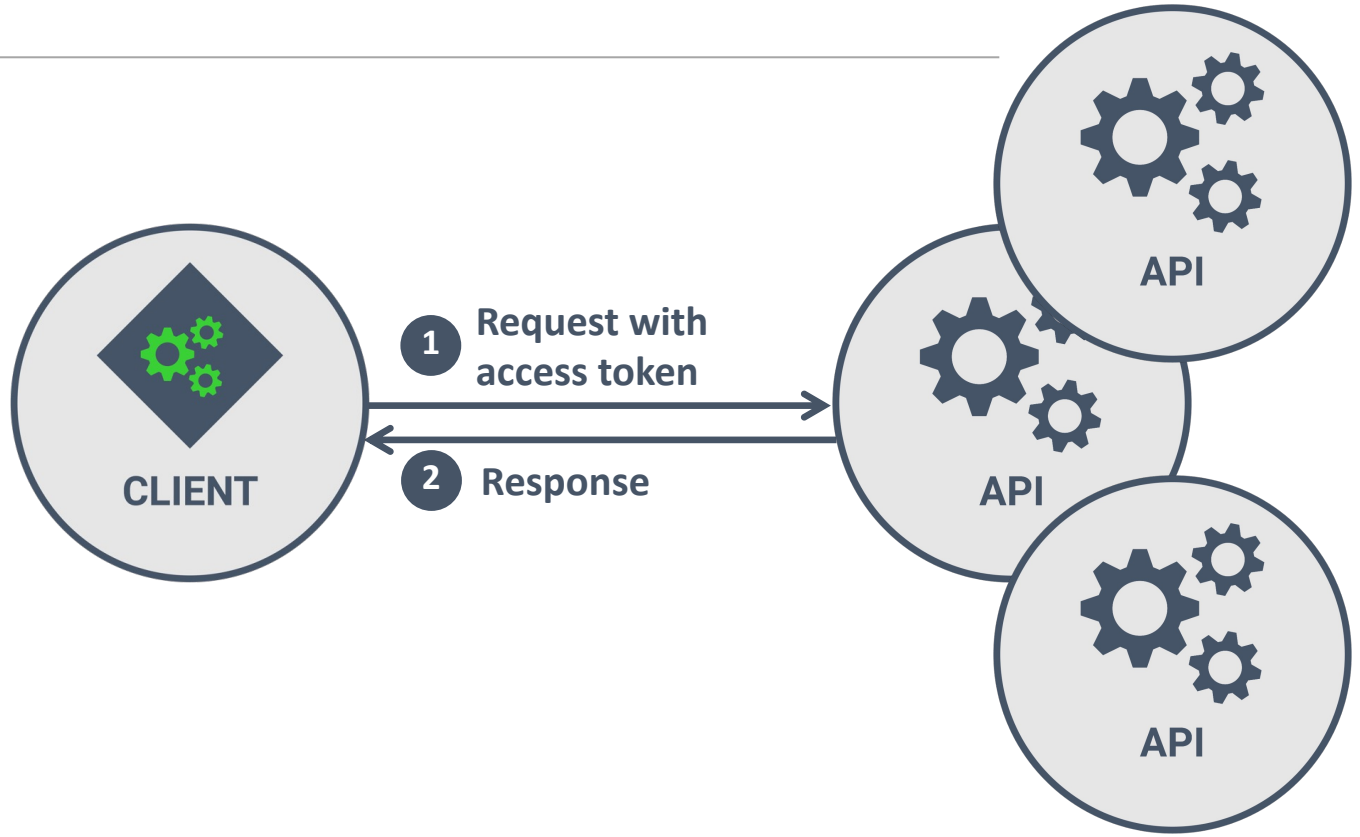
Scopes for specific APIs are also leaked to every receiver of the access token



An access token to access three separate Restograde APIs

```
{
  "iss": "https://sts.restograde.com",
  "aud": [ "https://api.restograde.com/reviews",
           "https://api.restograde.com/restaurants",
           "https://api.restograde.com/users" ],
  "sub": "2262430d-c9cb-484f-9770-805893ff9518",
  "scope": "restaurants:read reviews:write"
}
```

Encrypting the token is difficult when there are multiple receivers
(decryption happens with a private key that should not be shared)



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

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

Resource Indicators for OAuth 2.0

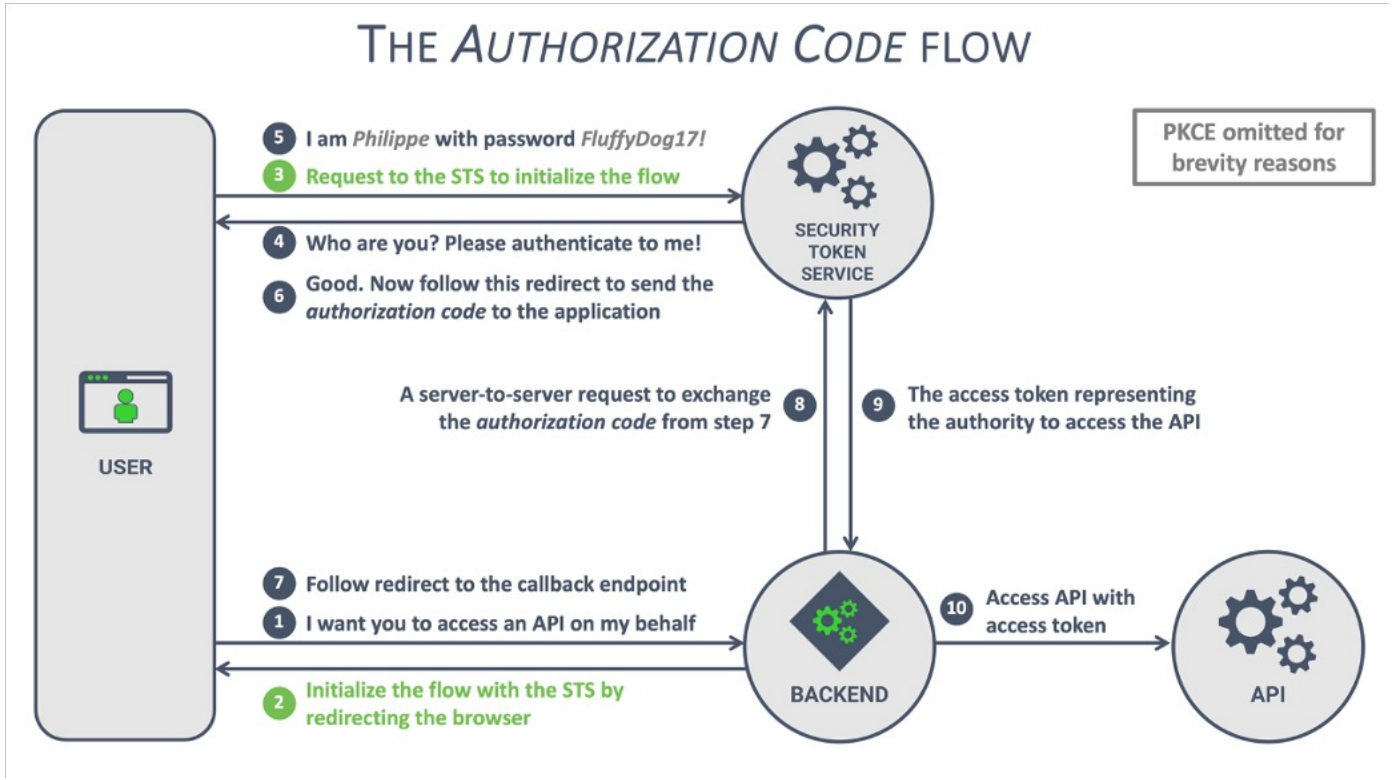
Abstract

This document specifies an extension to the OAuth 2.0 Authorization Framework defining request parameters that enable a client to explicitly signal to an authorization server about the identity of the protected resource(s) to which it is requesting access.¶

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

```
https://sts.restograde.com/authorize
  ?response_type=code
  &client_id=lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv
  &scope=read:restaurants write:reviews
  &resource=https://api.restograde.com/reviews
  &resource=https://api.restograde.com/restaurants
  &redirect_uri=https://app.restograde.com/callback
  & […  state / code_challenge / code_challenge_method …]
```

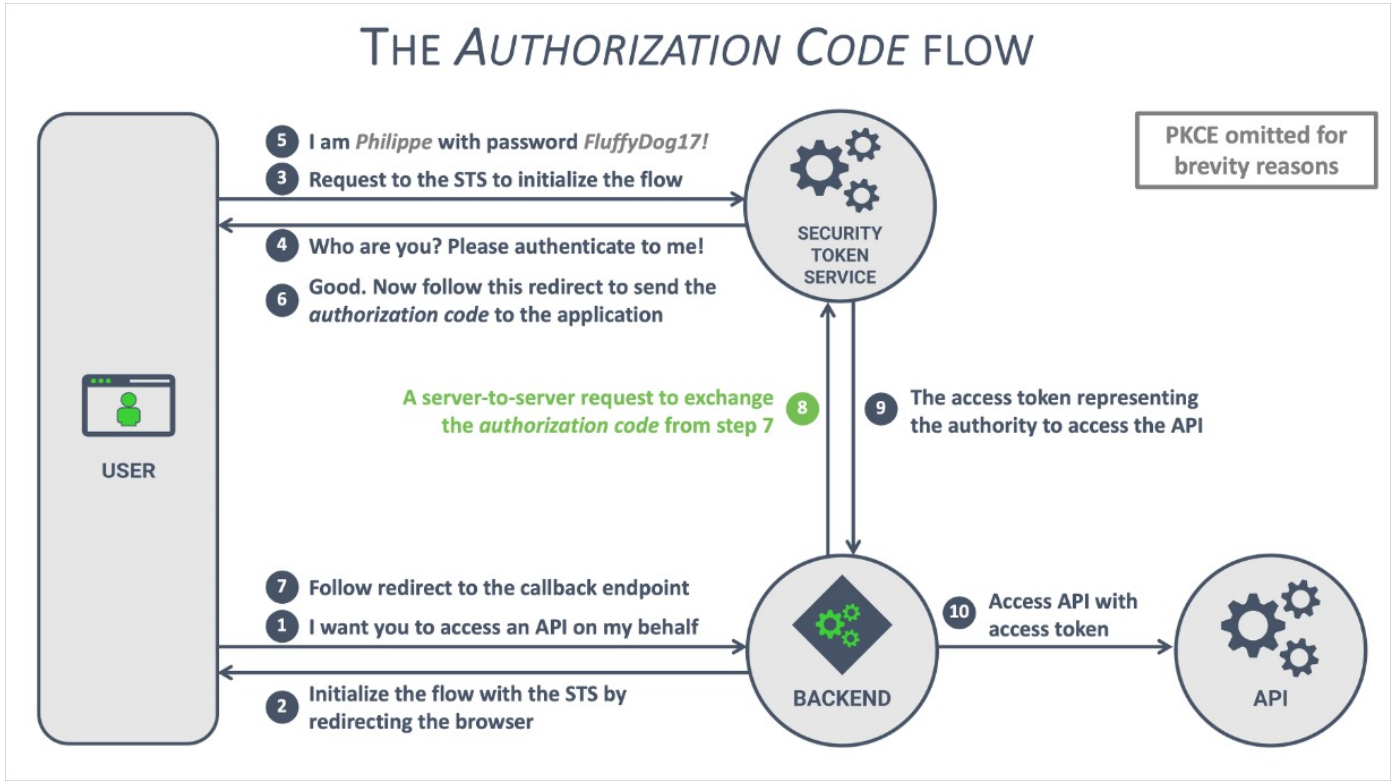
The identifiers of the requested resource servers (APIs)



8 *Requesting access tokens with a specific resource*

```
1  POST /oauth/token
2  Host: sts.restograde.com
3
4   grant_type=authorization_code
5  &client_id=lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv
6  &code=SplxlOBeZQQYbYS6WxSbIA
7  &resource=https://api.restograde.com/reviews
8  & […  redirect_uri / code_verifier …]
```

Requesting an access token for a specific resource server (API)

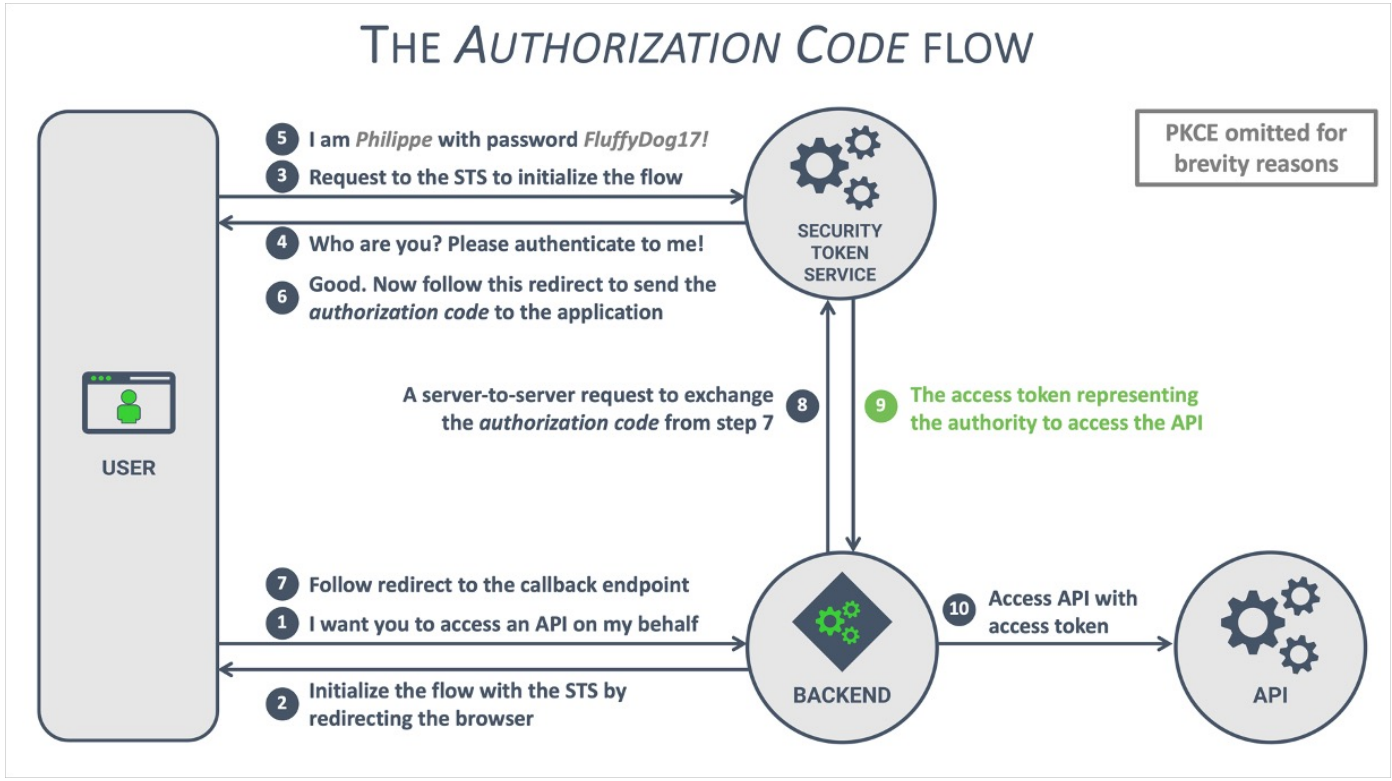


9 *The access token issued by the STS*

```
{
  "iss": "https://sts.restograde.com",
  "aud": "https://api.restograde.com/reviews",
  "sub": "2262430d-c9cb-484f-9770-805893ff9518",
  "scope": "reviews:write"
}
```

A specific target audience

The STS does "downscoping" by only including relevant scopes for the audience



The client can obtain additional access tokens by running a refresh token flow with the *resource* parameter

Using Resource Indicators

The Resource Indicators spec helps to reduce the authority of an access token to a single audience.

Resource Indicators are especially useful in large and complex architectures.

Securing the authorization request

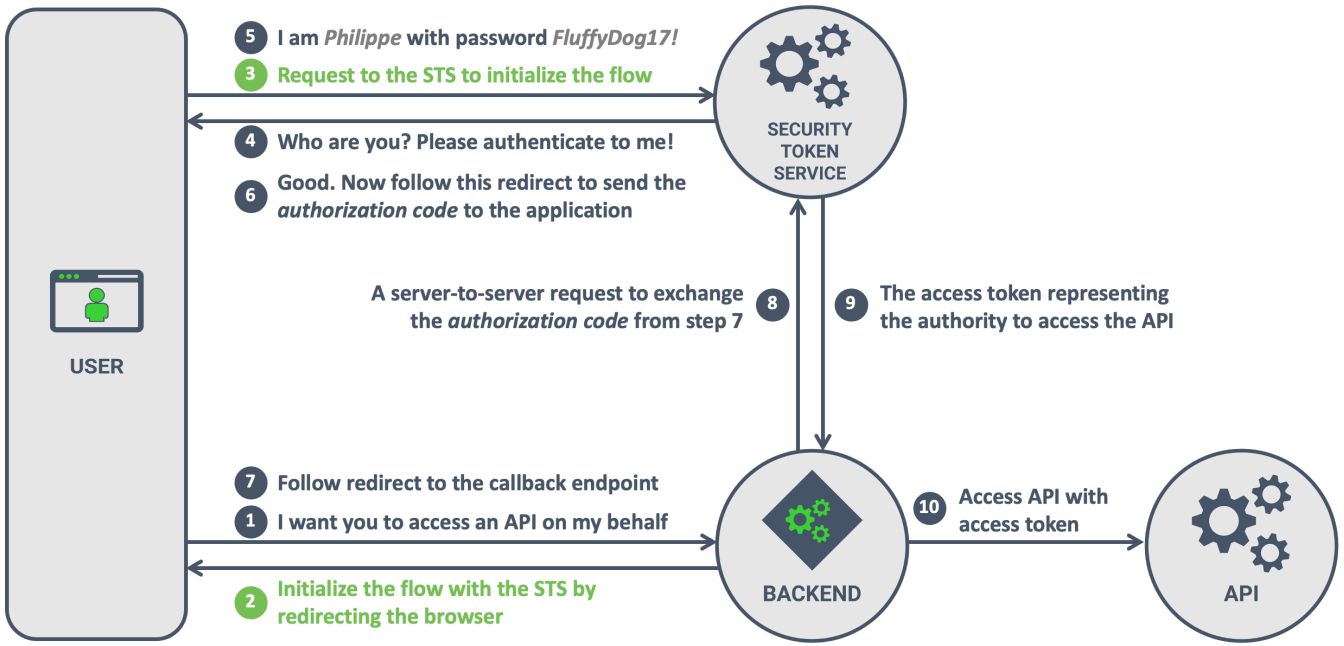
2 3 *The authorization request (a redirect to the STS)*

```
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



Workgroup: Web Authorization Protocol
Internet-Draft:
draft-ietf-oauth-security-topics-27
Updates: 6749, 6750, 6819 (if approved)
Published: 7 May 2024
Intended Status: Best Current Practice
Expires: 8 November 2024

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

OAuth 2.0 Security 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 RFC 6749, RFC 6750, and RFC 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.

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

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 gran
attacker to obtain an authorizati

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: 9101
Category: Standards Track
Published: August 2021
ISSN: 2070-1721

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

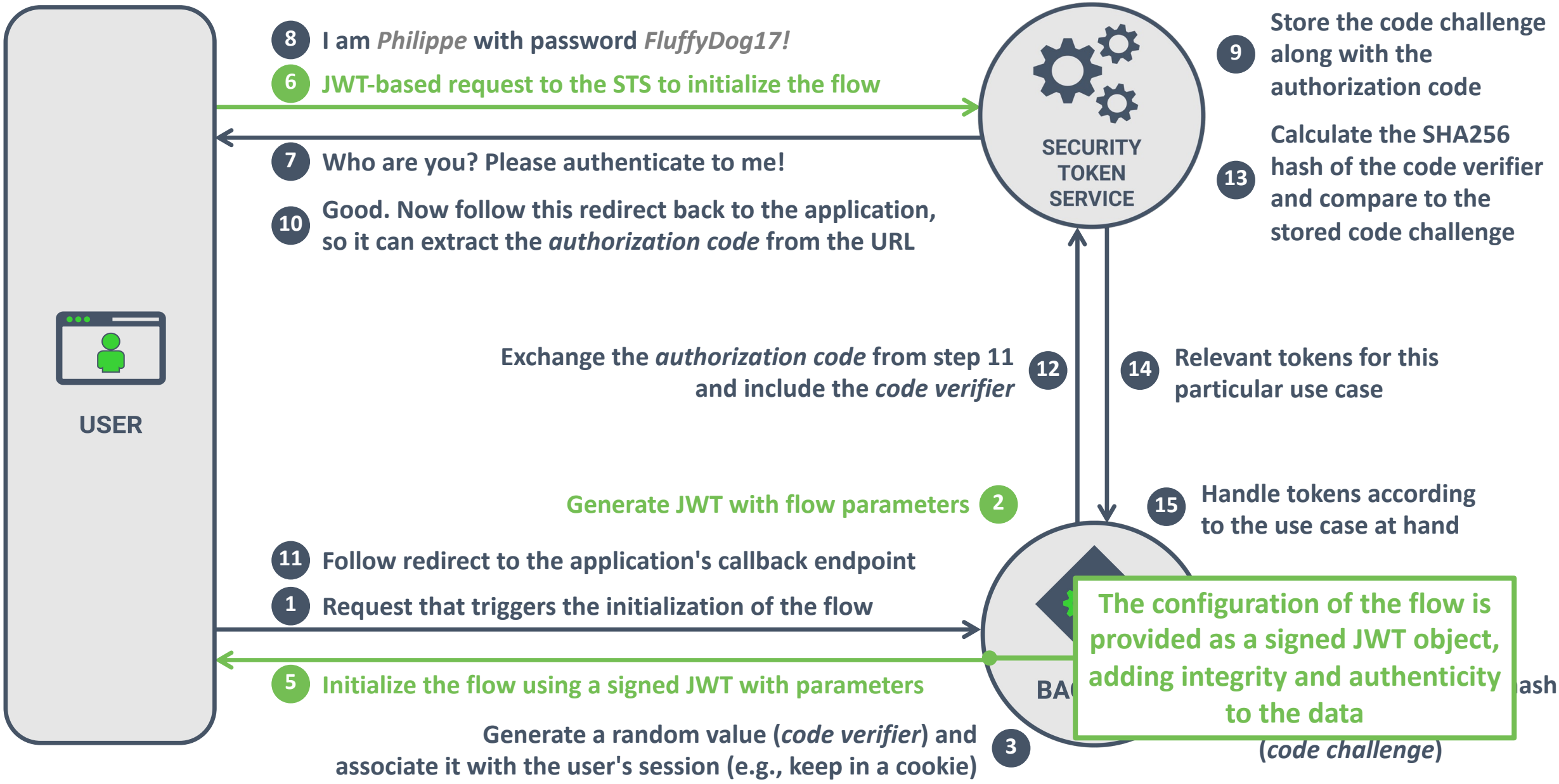
The OAuth 2.0 Authorization Framework: JWT-Secured Authorization Request (JAR)

Abstract

The authorization request in OAuth 2.0 described in RFC 6749 utilizes query parameter serialization, which means that authorization request parameters are encoded in the URI of the request and sent through user agents such as web browsers. While it is easy to implement, it means that a) the communication through the user agents is not integrity protected and thus, the parameters can be tainted, b) the source of the communication is not authenticated, and c) the communication through the user agents can be monitored. Because of these weaknesses, several attacks to the protocol have now been put forward.

This document introduces the ability to send request parameters in a JSON Web Token (JWT) instead, which allows the request to be signed with JSON Web Signature (JWS) and encrypted with JSON Web Encryption (JWE) so that the integrity, source authentication, and confidentiality properties of the authorization request are attained. The request can be sent by value or by reference.

THE *AUTHORIZATION CODE* FLOW WITH JAR



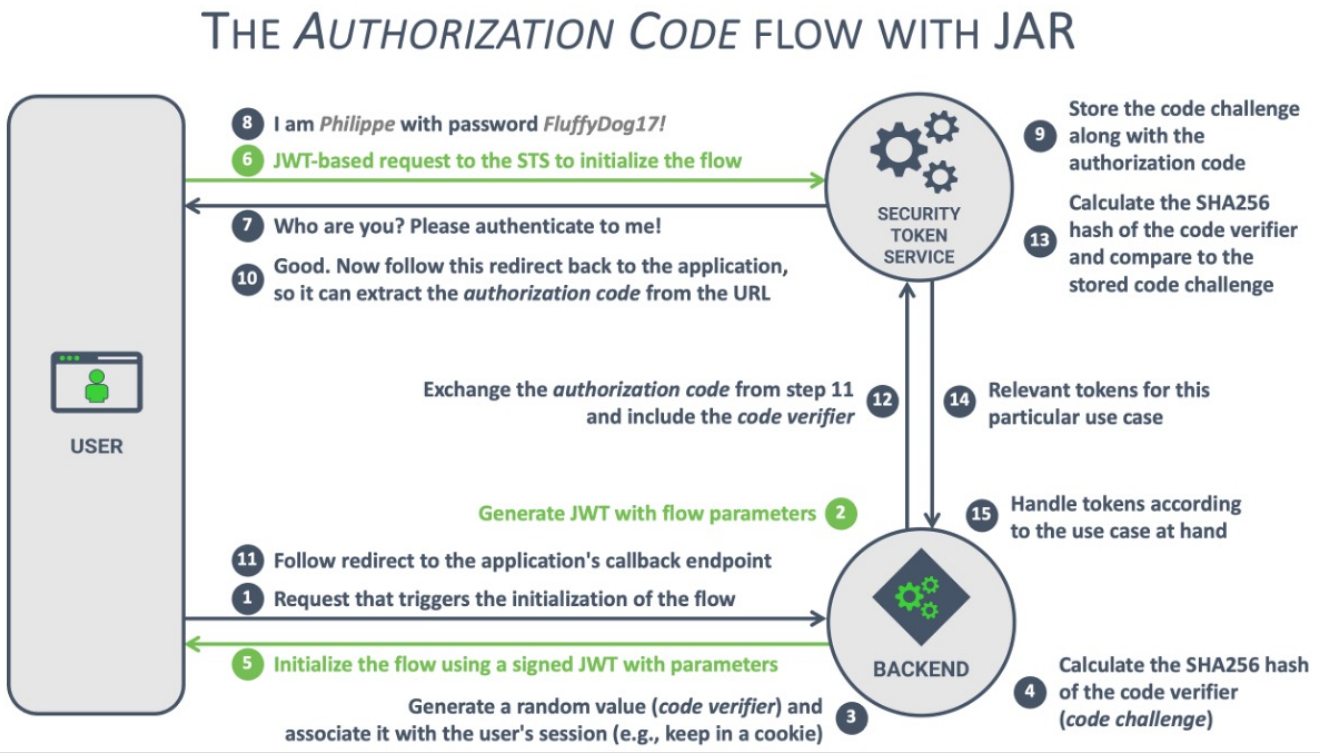
5 6 *The redirect URI*

```
https://sts.restograde.com/authorize
  ?client_id=lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv
  &request=eyJhbGciOiJQUzI1NiIsInR5cCI6Im9hdXRoLWF1dGh6LX
  JlcStqd3QifQ.eyJpc3MiOiJsWTVnMEJLQjdNb3c0eURsYjZyZEdQc0
  8yaTFnN09zdiIsImF1ZCI6Imh0dHBzOi8vc3RzLnJlc3RvZ3JhZGUuY
  …
  a8JSiQtbP4IKzGXvHoJvPh-T4OxgA9QZj9erIT2wEVBcieAO0340zl2
  Y5Z953bgpSb4O4NbFKXa_lD4GTJ2LGF48IGjRQ
```

Indicates the client making the request

The configuration of the flow

The JWT is signed by the private key of the client and contains all the traditional flow configuration parameters



The encoded JWT request


```
eyJhbGciOiJQUzI1NiIsInR5cCI6Im9hdXRoLWF
1dGh6LXJlcStqd3QiLCJraWQiOiJoaGJHeGxibW
RsSWpvaVNtaEZUIn0.eyJpc3MiOiJsWTVnMEJLQ
jdNb3c0eURsYjZyZEdQc08yaTFnN09zdiIsImF1
ZCI6Imh0dHBzOi8vc3RzLnJlc3RvZ3JhZGUuY29
tIiwicmVzcG9uc2VfdHlwZSI6ImNvZGUiLCJjbG
llbnRfaWQiOiJsWTVnMEJLQjdNb3c0eURsYjZyZ
EdQc08yaTFnN09zdiIsInJlZGlyZWN0X3VyaSI6
Imh0dHBzOi8vYXBwLnJlc3RvZ3JhZGUuY29tL2N
hbGxiYWNrIiwic2NvcGUiOiJyZWFkIiwic3RhdG
UiOiJzMHd6b2ptMnc4YzIzeHpwcmtrNiIsImNvZ
GVfY2hhbGxlbmdlIjoiSmhFTjBBbW5qN0LigKZX
aDVQeFdpdFpZSzF3b1doNVB4V2l0WlkiLCJjb2R
lX2NoYWxsZW5nZV9tZXRob2QiOiJTMjU2In0.LJ
pskbj0rYhwxt4Bwiiw1Ku-
nmhGuOFUvqBrv7xLFu6Tkkes6p9c7xvyulp017Q
ptCZlN5i7wQyXp5VY32fZ0dF9akGEhQymPSvyBe
wzZgDrEOM8ZD_-
LbQhlg2OwE3ekq4mwIsYVZVRA4RQJMmN9JuoQHU
cuBRDke_bdR1K6XosHQuy-
wEz7j8yix8vcqGgSe6MvPN3nZjShMAcTd9QJpZX
qin5NqXlByFj9iRecBygOK6snJwz7S2s79R6987
1Tz8Ap_vCcVtJRLinBCzyjS0JHEBMvrvuOxzxCH
4comCM96fyi47D5yRZFsUJmfIDJr1D4yOIVbQIu
2GKA_bULw
```


The header of the decoded JWT object


```
{
  "alg": "PS256",
  "typ": "oauth-authz-req+jwt",
  "kid": "hhbGxlbmdlIjoiSmhFT"
}
```


The payload of the decoded JWT object


```
{
  "iss": "lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv",
  "aud": "https://sts.restograde.com",
  "response_type": "code",
  "client_id": "lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv",
  "redirect_uri": "https://app.restograde.com/callback",
  "scope": "read",
  "state": "s0wzojm2w8c23xzprkk6",
  "code_challenge": "JhEN0Amnj … xWitZYK1woWh5PxWitZY",
  "code_challenge_method": "S256"
}
```

The header of the decoded JWT object

The header defines the JWT type

The key identifier helps the STS to select the right key from the client

```
{
  "alg": "PS256",
  "typ": "oauth-authz-req+jwt",
  "kid": "hhbGxlbmdlIjoiSmhFT"
}
```

The payload of the decoded JWT object

The issuer of the JWT is the client, and the audience is the STS

The client ID must match the client ID provided in the URL

The JWT request contains the parameters that used to be present in the URL

```
{
  "iss": "lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv",
  "aud": "https://sts.restograde.com",
  "response_type": "code",
  "client_id": "lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv",
  "redirect_uri": "https://app.restograde.com/callback",
  "scope": "read",
  "state": "s0wzojm2w8c23xzprkk6",
  "code_challenge": "JhEN0Amnj … xWitZYK1woWh5PxWitZY",
  "code_challenge_method": "S256"
}
```

JAR in action

USING JAR

JWT-Secured Authorization Requests enable integrity protection for the parameters in the authorization request.

JAR eliminates increasingly common attacks against the authorization request being sent over the insecure frontchannel.

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

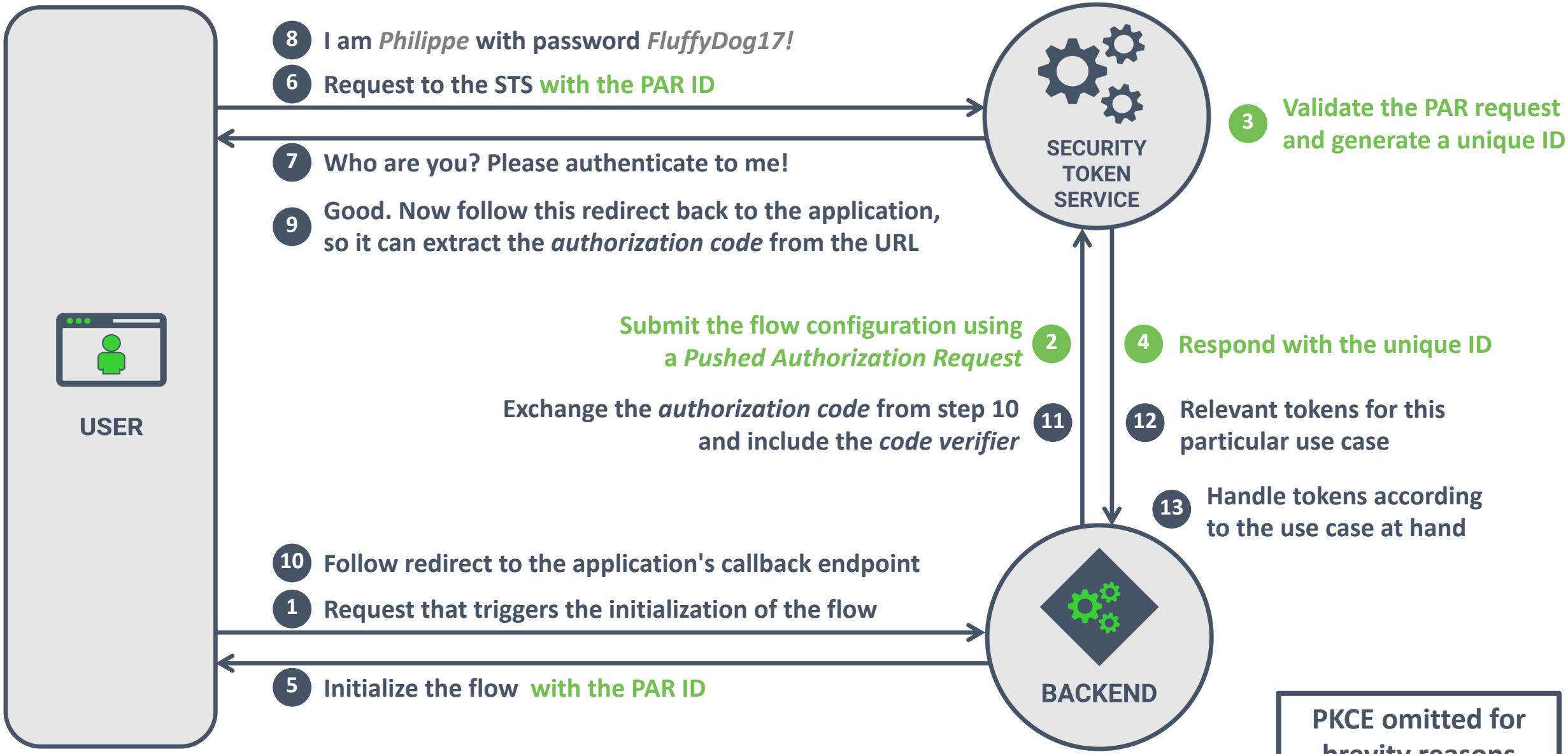
T. Lodderstedt
yes.com
B. Campbell
Ping Identity
N. Sakimura
NAT.Consulting
D. Tonge
Moneyhub Financial
Technology
F. Skokan
Auth0

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
```

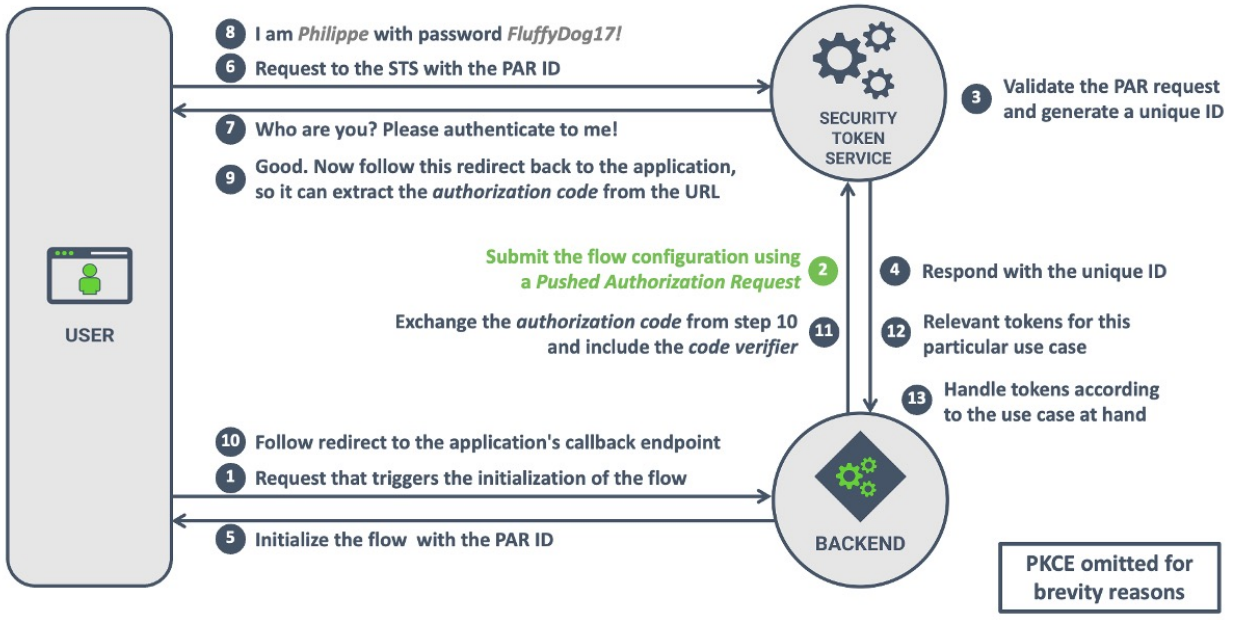
The STS supports a new PAR endpoint

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

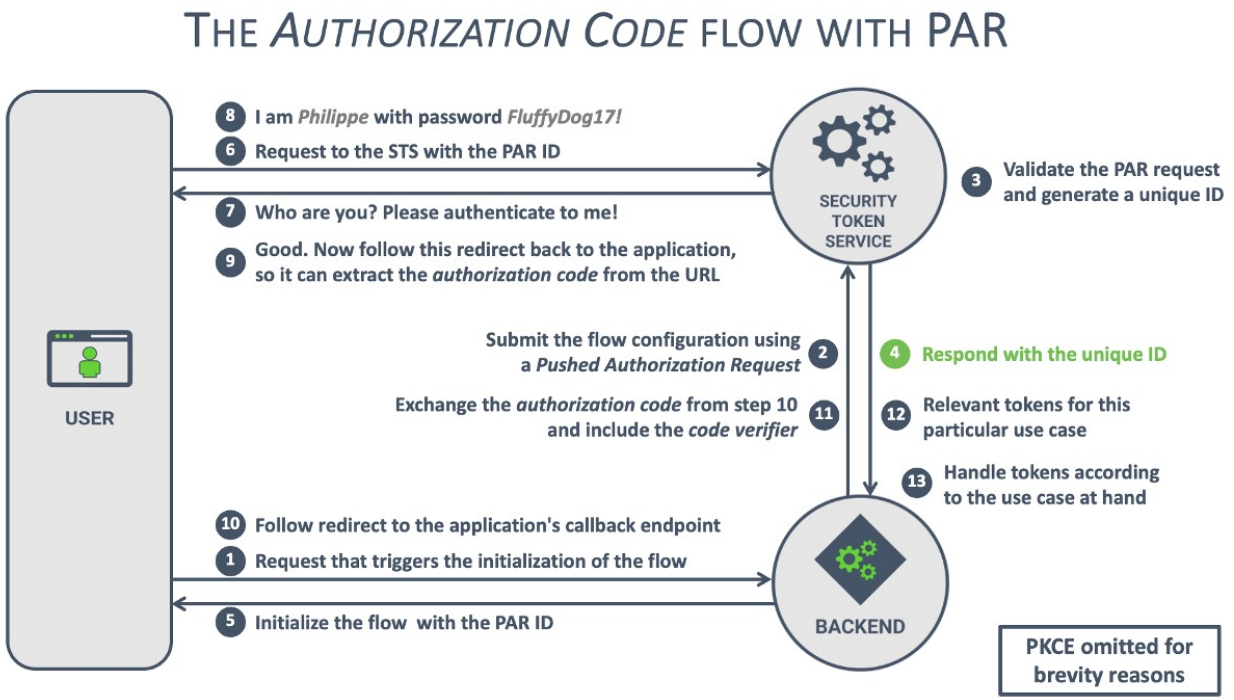


4 *The response to the PAR request*

```
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 redirect URI with the PAR ID*

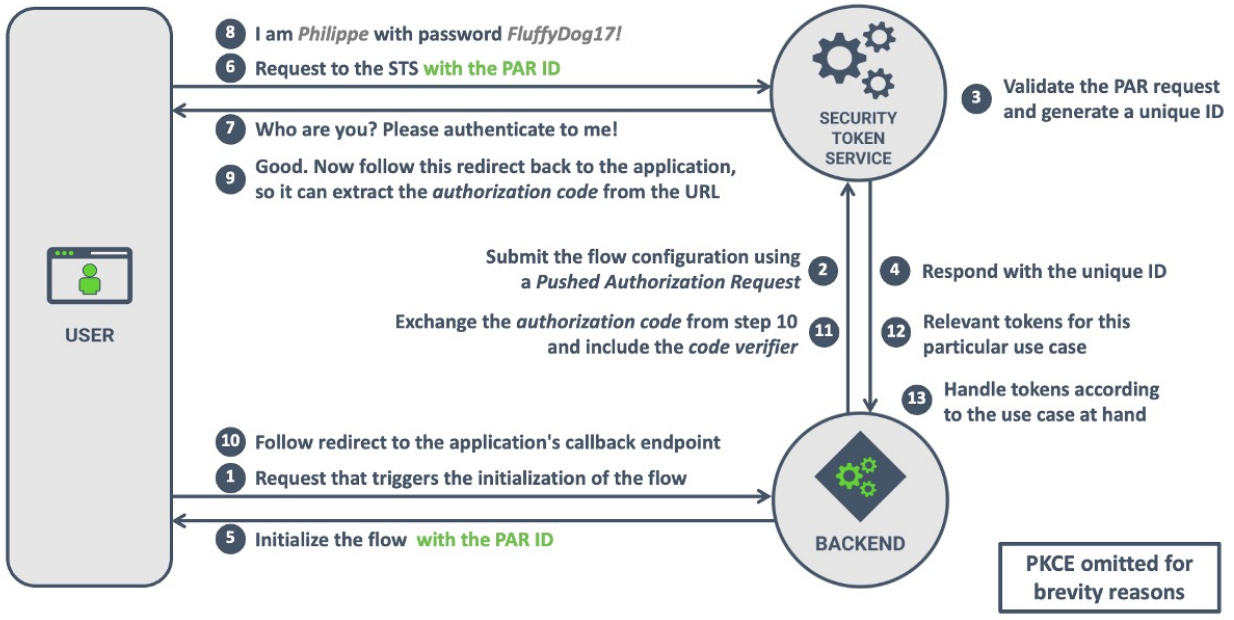
```
1  https://sts.restograde.com/authorize
2    ?client_id=lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv
3    &request_uri=urn:ietf:params:oauth:request_uri:6e11ACC5
4                 bwc014ltc14eY22c
```

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

THE *AUTHORIZATION CODE* FLOW WITH PAR



PAR in action

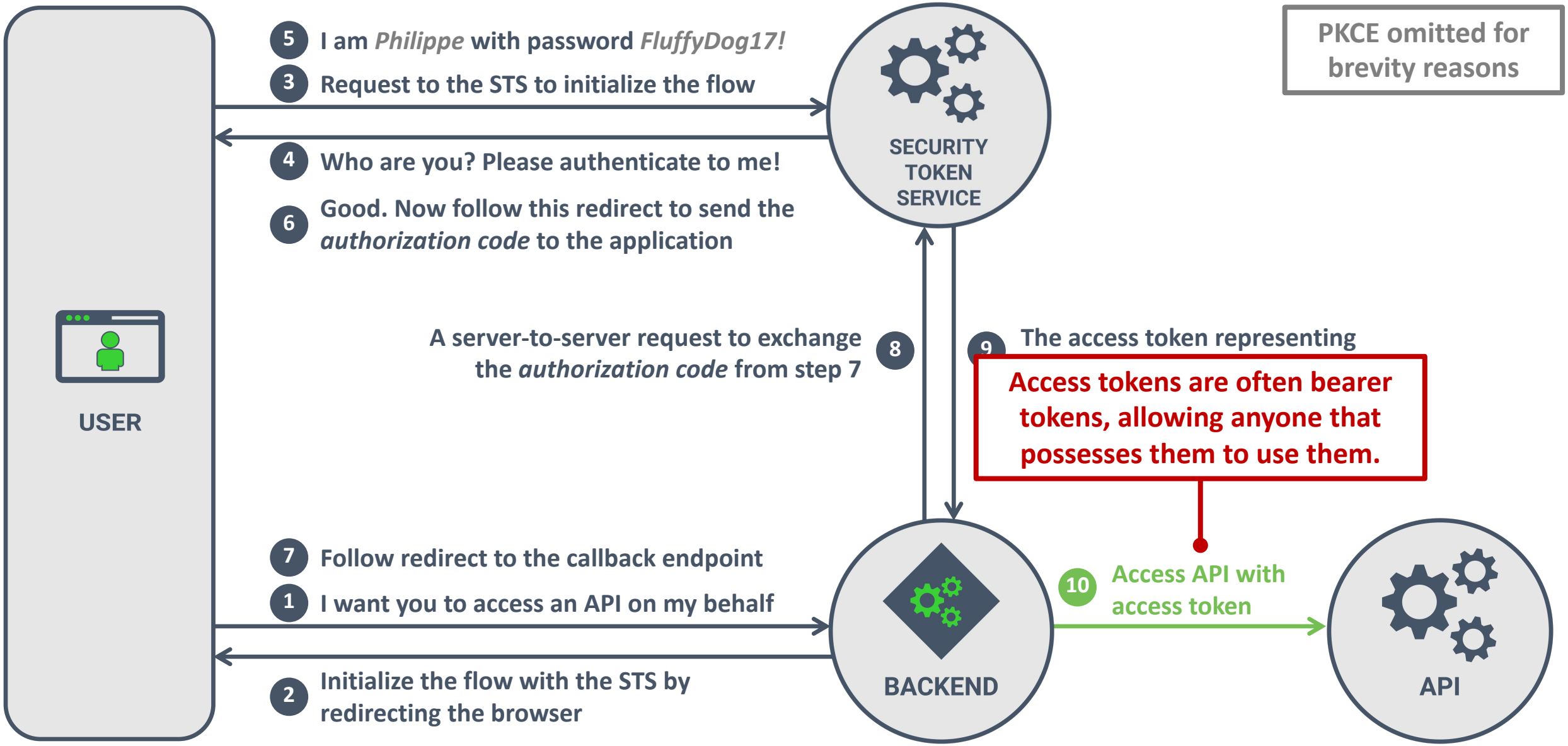
When using PAR, make sure clients are no longer allowed to use a regular Authorization Code flow

USING PAR

Pushed Authorization Requests eliminate the need to send flow configuration parameters over the frontchannel and is quickly becoming a recommended best practice.

If desired, the PAR request can contain a JWT-secured Authorization Request, even though the combination of PAR and JAR is mostly overkill.

Sender-constrained Tokens


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

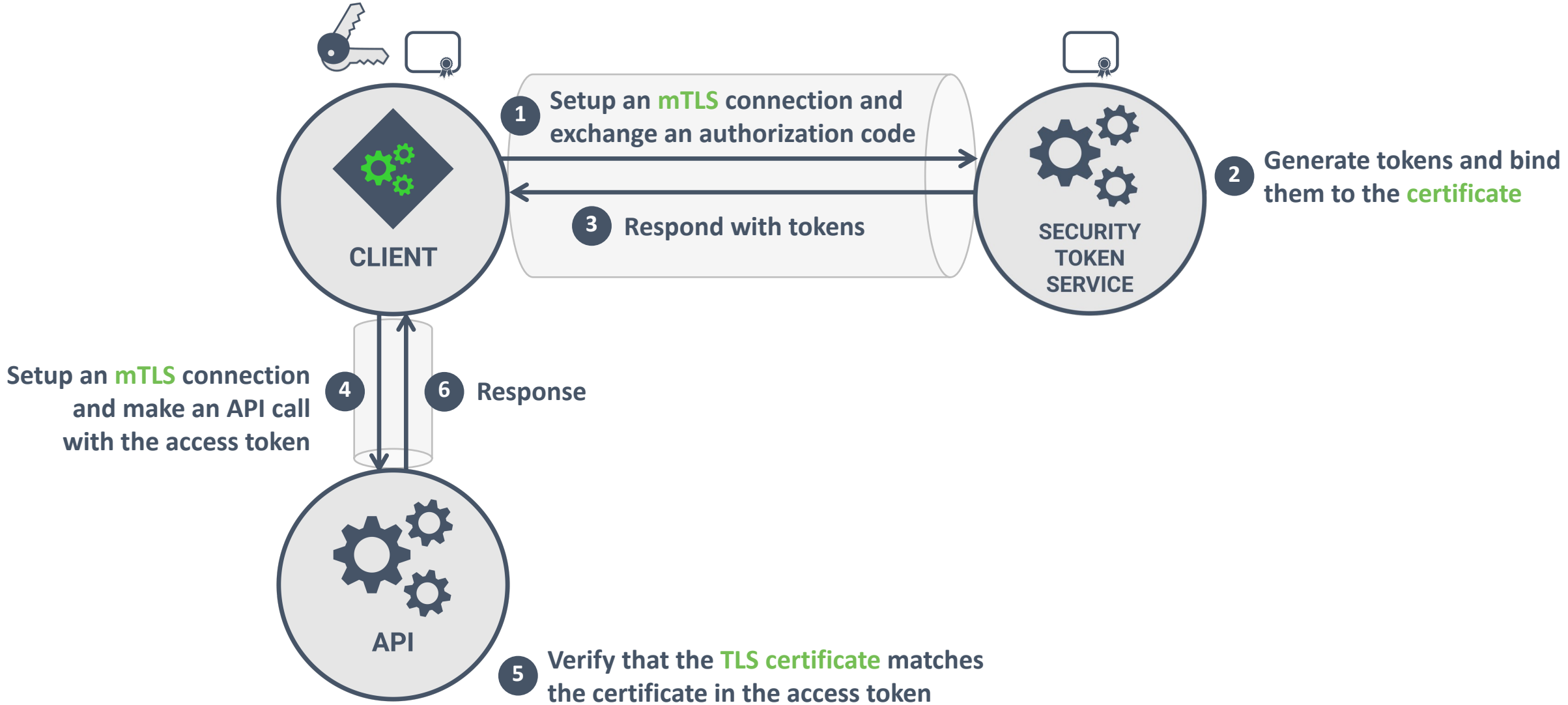
B. Campbell
Ping Identity
J. Bradley
Yubico
N. Sakimura
Nomura Research Institute
T. Lodderstedt
YES.com AG

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.

PROOF-OF-POSSESSION THROUGH TLS CERTIFICATES



SENDER-CONSTRAINED TOKENS WITH MTLS

A JWT access token with an embedded certificate fingerprint

```
1  {
2    "sub": "b6rdGPsO2iBKB7sO2i",
3    "aud": "https://api.example.com",
4    "azp": "lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv",
5    "iss": "https://sts.restograde.com/",
6    "exp": 1419356238,
7    "iat": 1419350238,
8    "scope": "read write",
9    "cnf": {
10     "x5t#S256": "bwcK0esc3ACC3DB2Y5_lESsXE8o9ltc05O89jdN-dg2"
11   }
12 }
```

The fingerprint of the cert

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

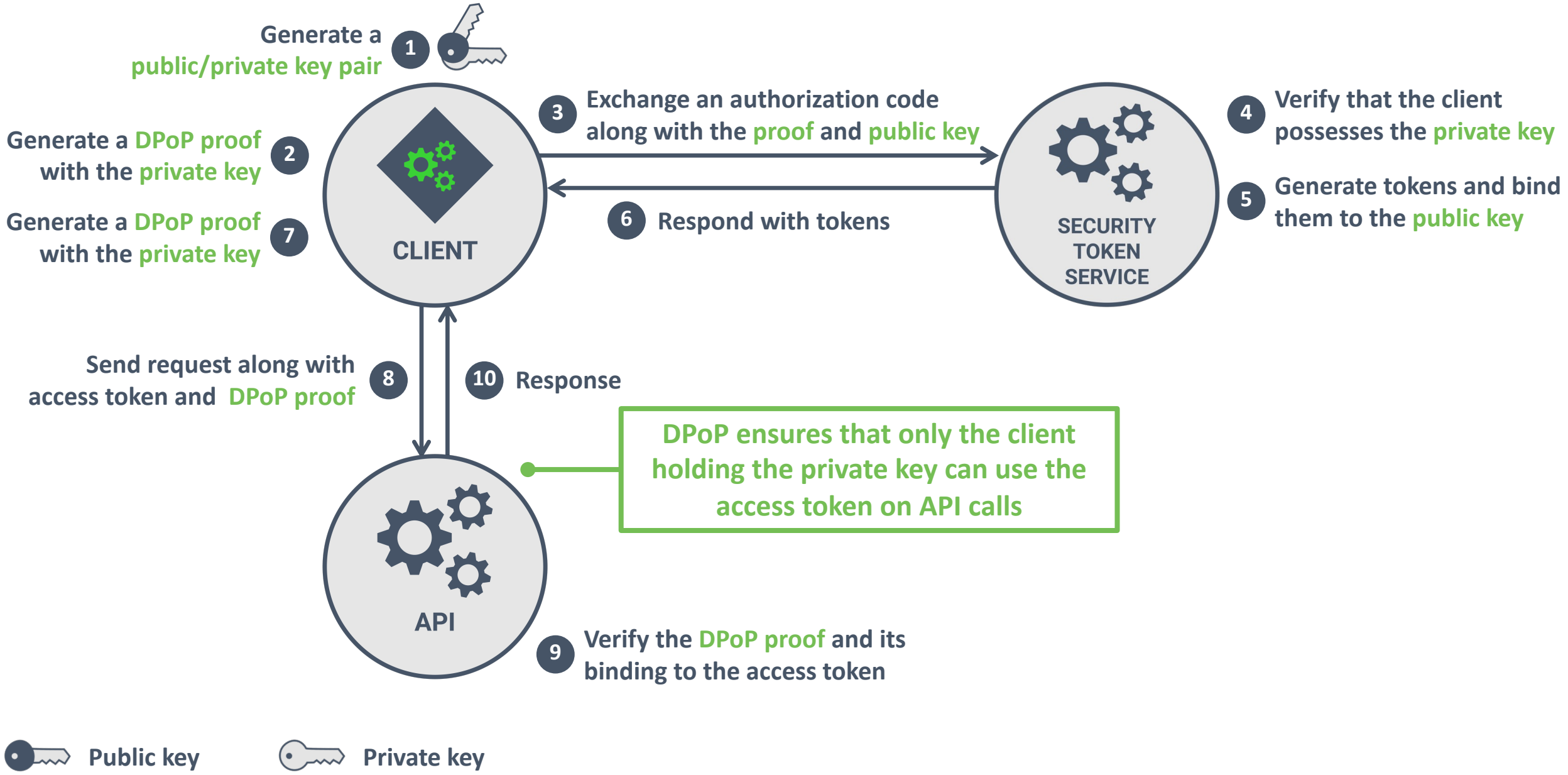
D. Fett
Authlete
B. Campbell
Ping Identity
J. Bradley
Yubico
T. Lodderstedt
Tuconic
M. Jones
Self-Issued Consulting
D. Waite
Ping Identity

OAuth 2.0 Demonstrating Proof of Possession (DPoP)

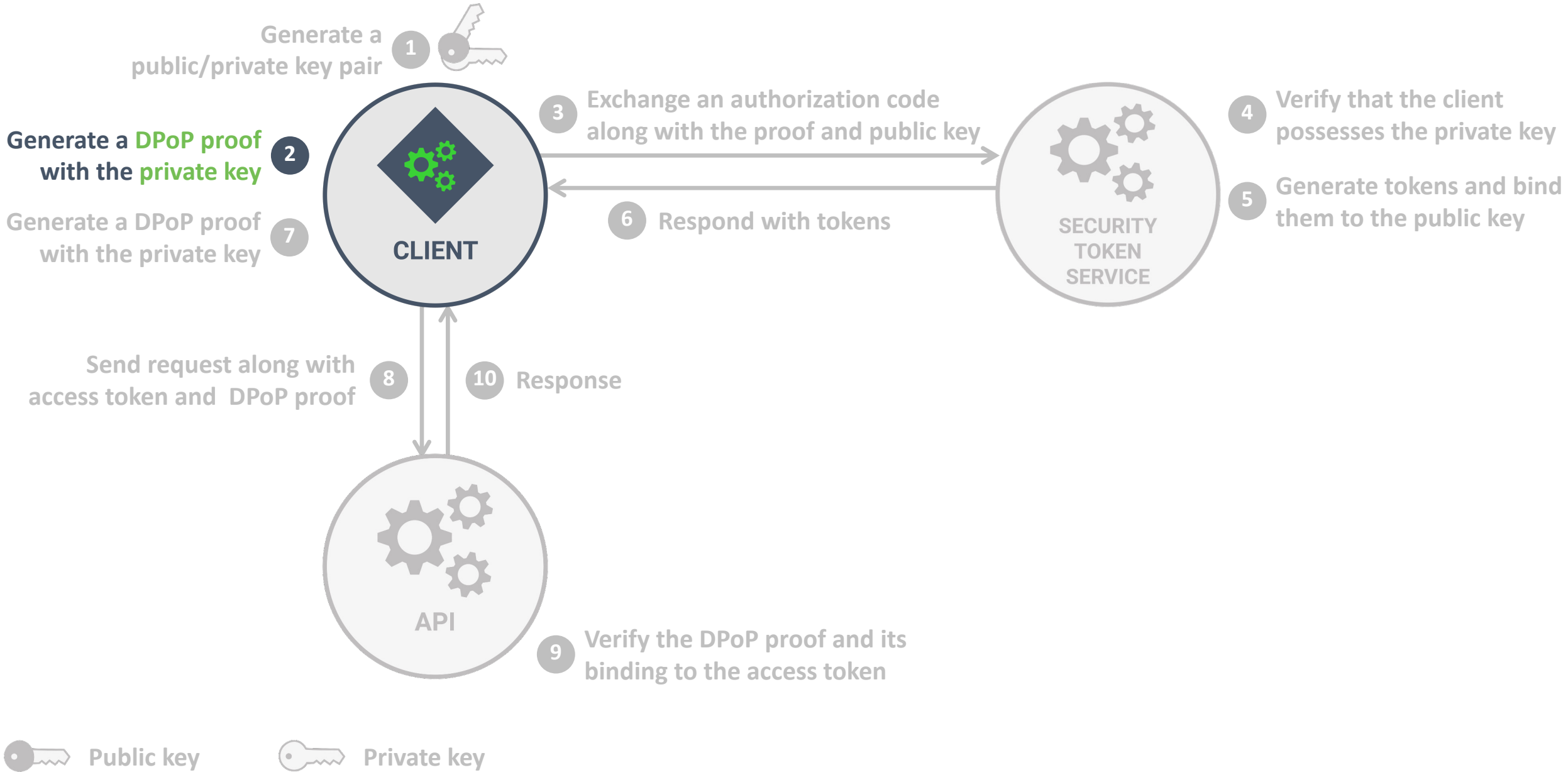
Abstract

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
}
```

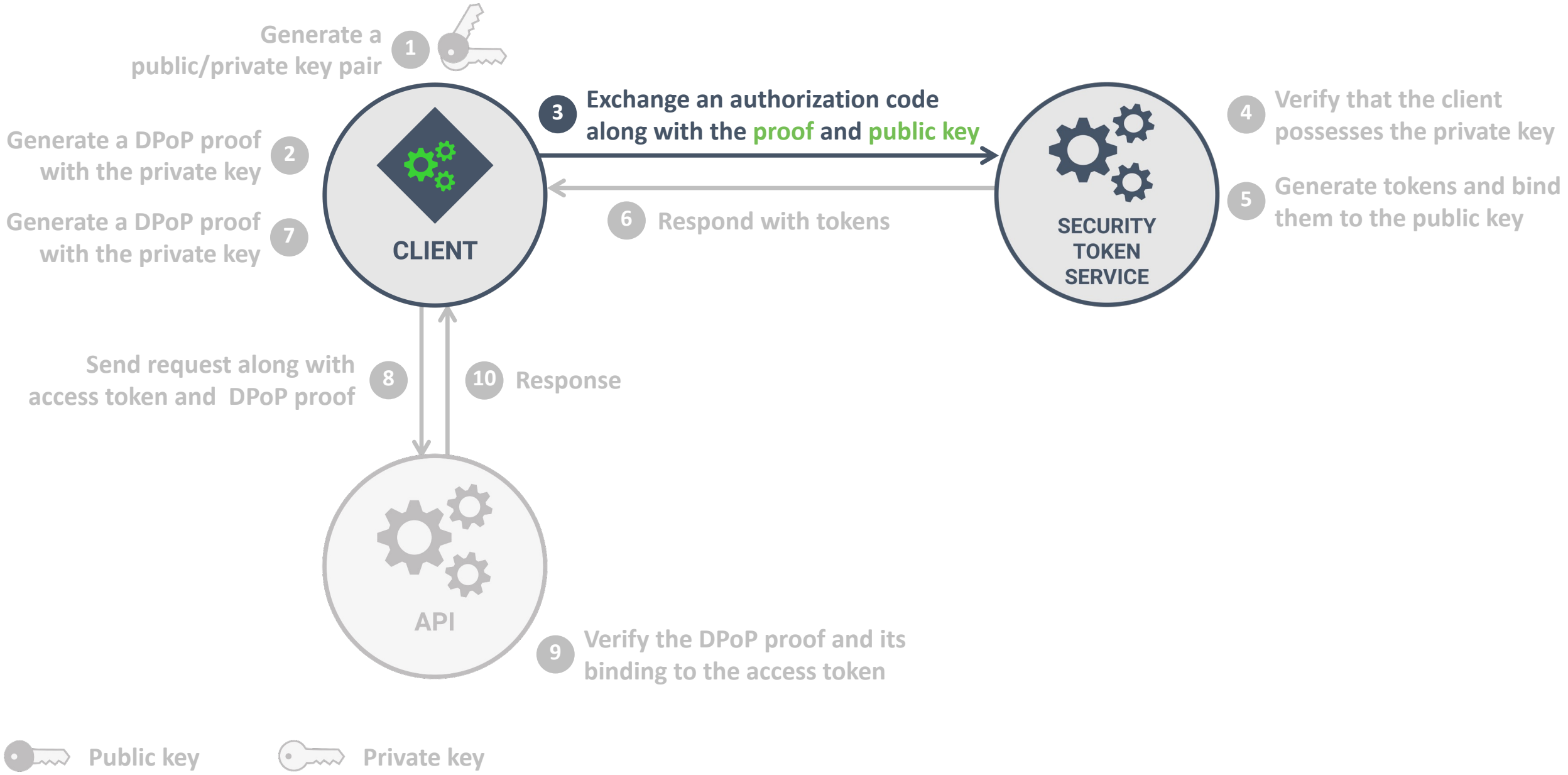
The token type indicates a JWT DPoP proof

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



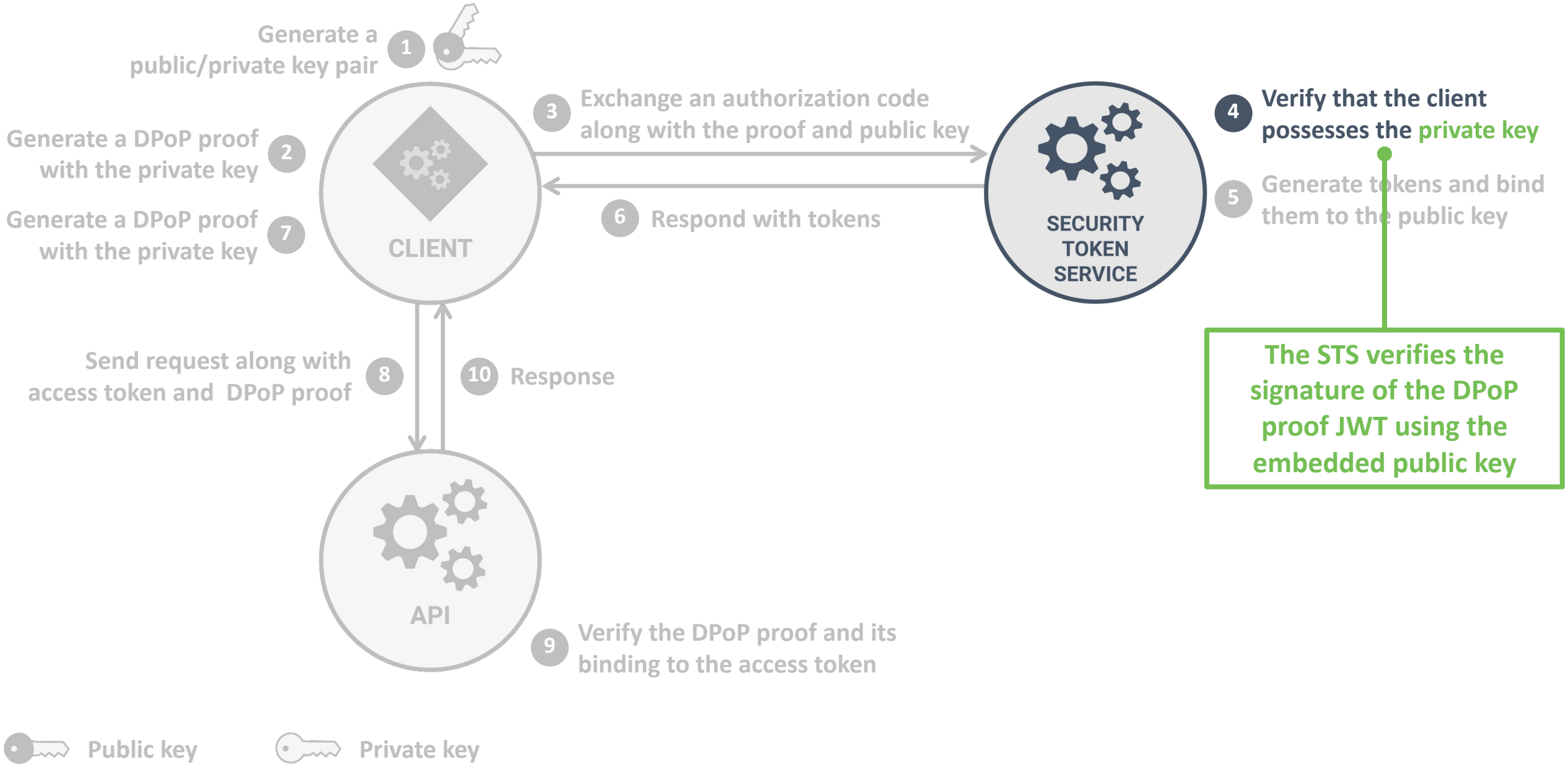
3 *The token request with the authorization code and DPoP proof*

A traditional request to the token endpoint

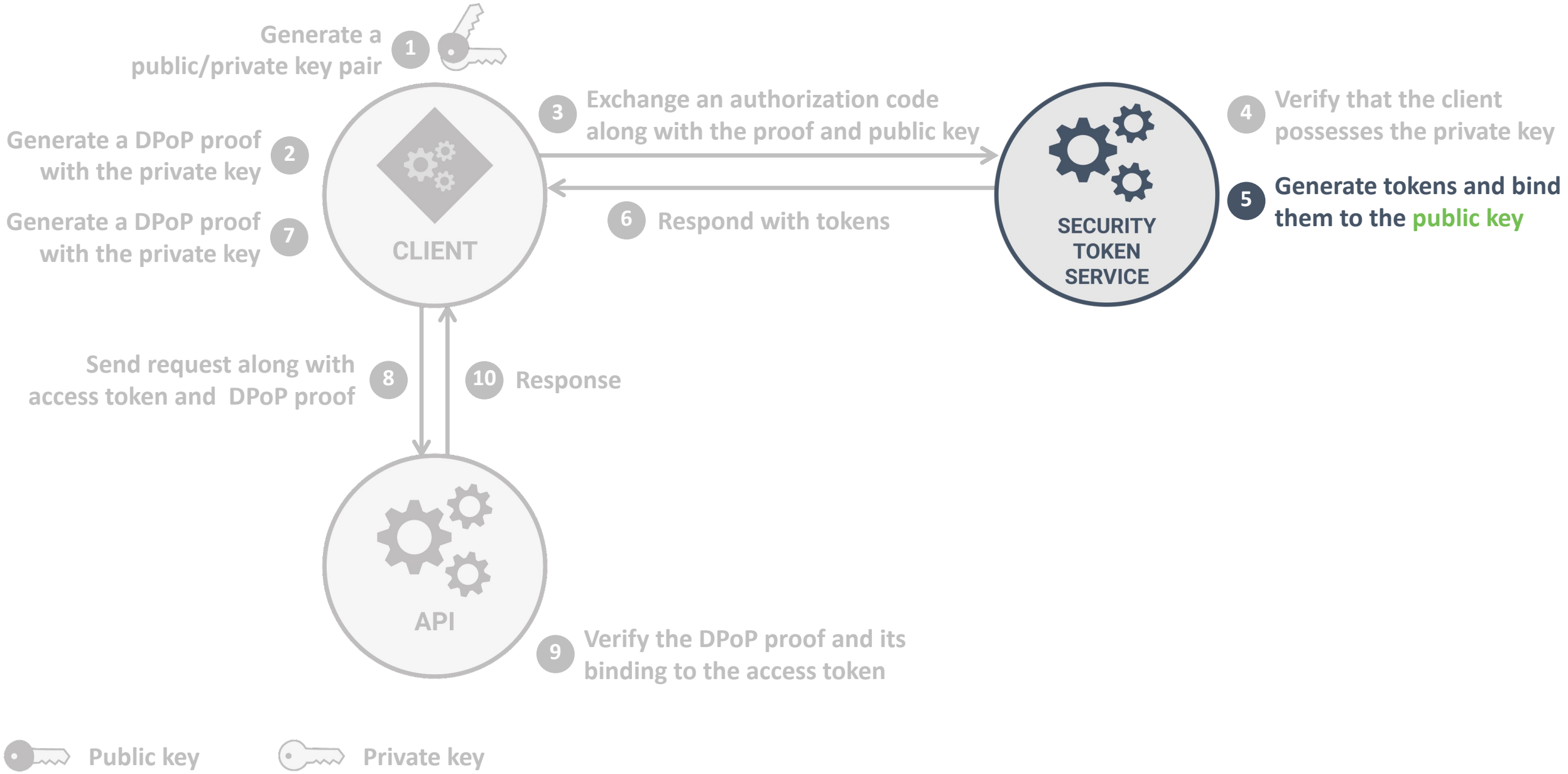
```
POST /token
Host: sts.restograde.com
DPoP: eyJ0eXAiOiJkcG9 … fbV37xRZT3Lg
 
 grant_type=authorization_code
&client_id=lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv
&redirect_uri=https://app.restograde.com/callback
&code=SplxlOBeZQQYbYS6WxSbIA
&code_verifier=lT5q6nbPQRtdj…~IUdkErVDFG.fF4z7CzCxo
```

The DPoP proof generated in step 2

The concept of DPoP



THE CONCEPT OF DPOP



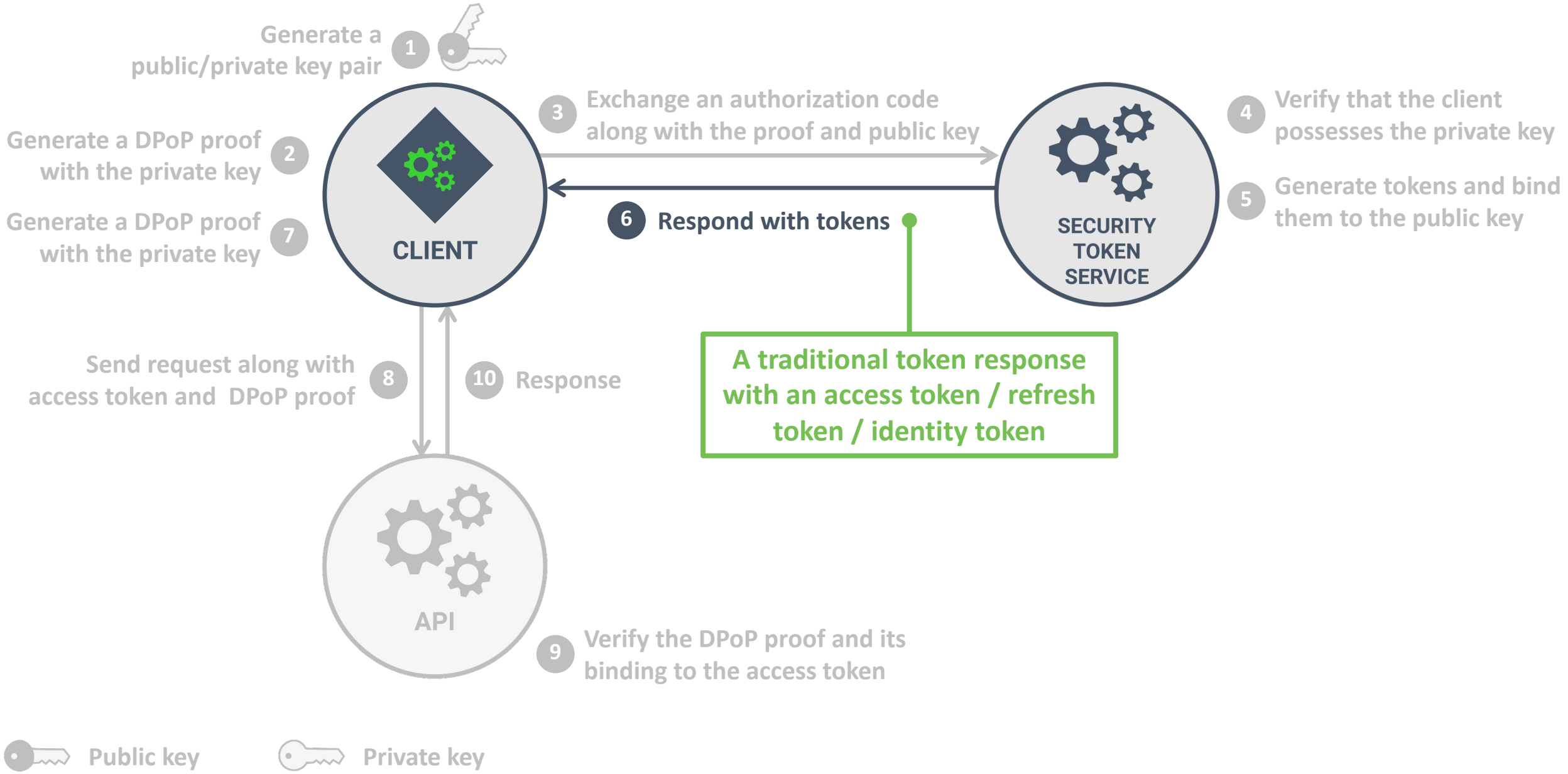
5 *The generated access token bound to the client's public key*

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

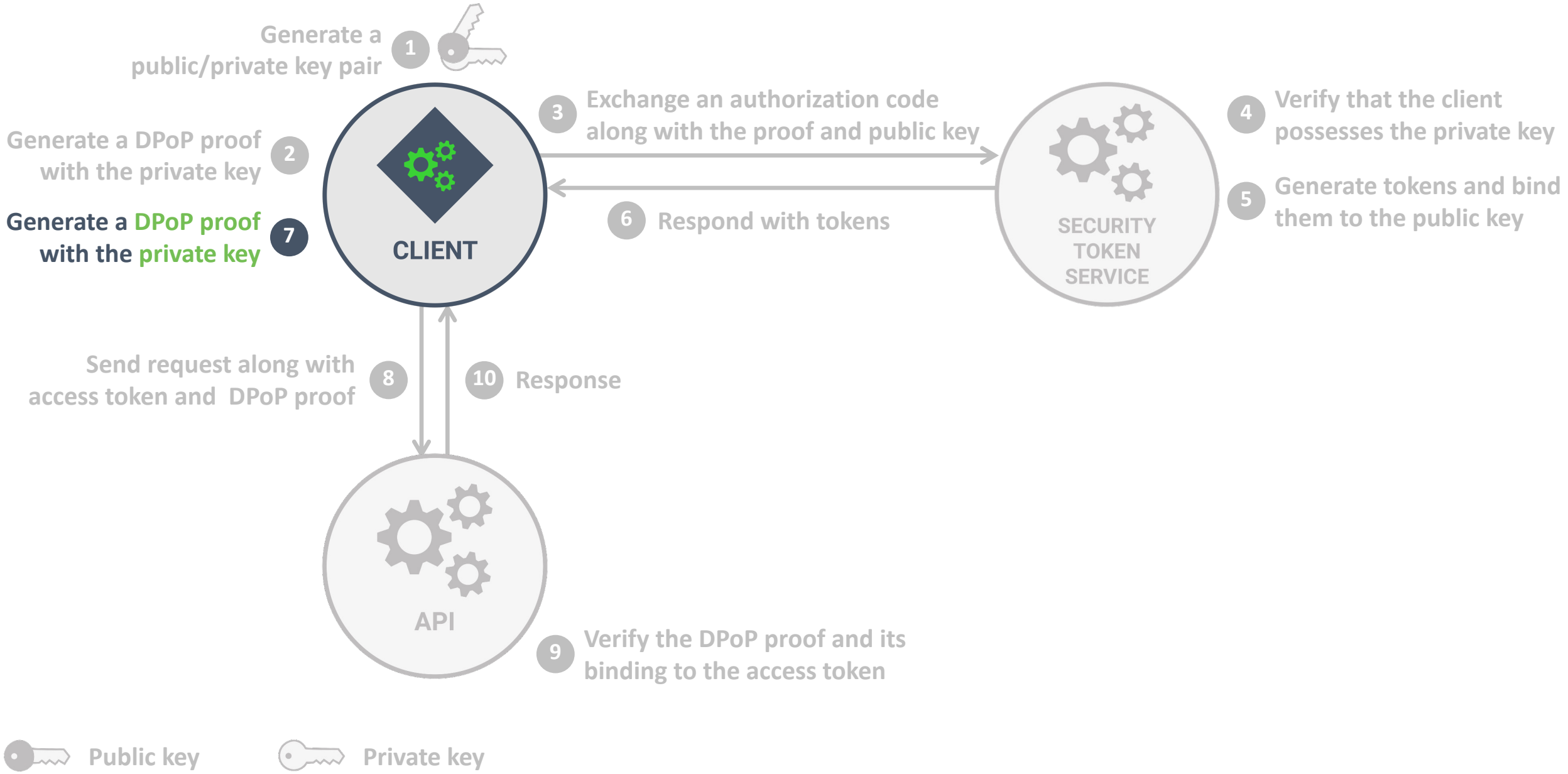
A traditional self-contained access token

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

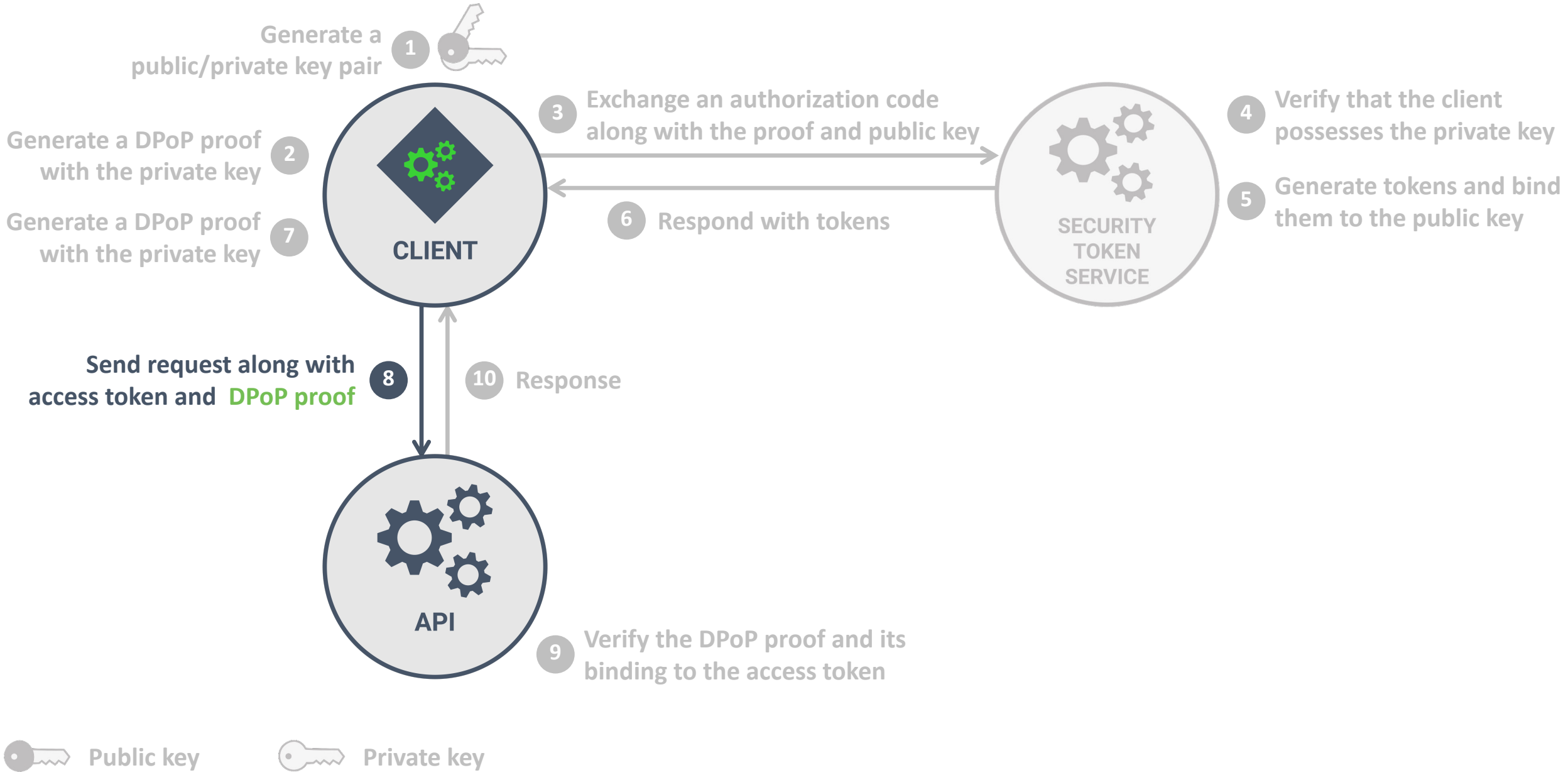
The client's public key is part of the header

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*

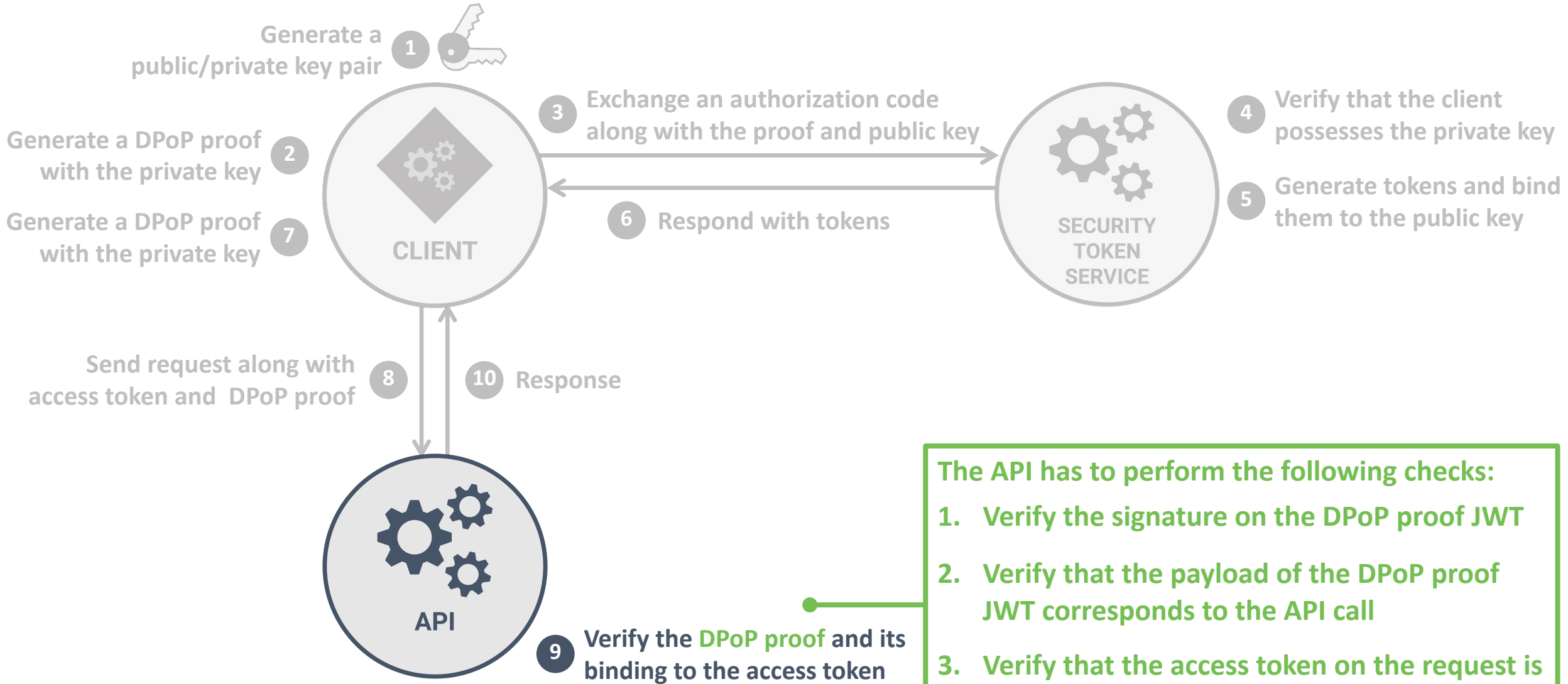
```
1  GET /reviews
2  Host: api.restograde.com
3  Authorization: DPoP eyJ0f37x3LgRZTbV … eRAiOiJkcG9
4  DPoP: eyJ0eXAiOiJkcG9 … fbV37xRZT3Lg
```

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**

SENDER-CONSTRAINING TOKENS

Security-sensitive applications should consider adopting sender-constrained tokens over bearer tokens.

mTLS handles most of the heavy lifting for using sender-constrained tokens. DPoP operates on the application layer and requires more effort, but offers more flexibility.

Key takeaways

1. Use Resource Indicators to reduce the authority of access tokens
2. Make PAR the default for your Authorization Code flows
3. Sender-constrained tokens should be preferred over bearer tokens

Thank you!

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

https://pragmaticwebsecurity.com