TRADE-OFFS WITH TOKEN SECURITY

DR. PHILIPPE DE RYCK

https://Pragmatic Web Security.com

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6Ik5UVkJPVFUzTXpCQk9FVXdOemhCUTBWR0
1rUTBRVVU1UVRZeFFVVXlPVU5FUVVVeE5qRXlNdyJ9.eyJpc3MiOiJodHRwczovL3N0cy5yZXN0b2d
yYWRlLmNvbS8iLCJzdWIiOiJhdXRoMHw1ZWI5MTZjMjU4YmRiNTBiZjIwMzY2YzYiLCJhdWQiOlsia
HR0cHM6Ly9hcGkucmVzdG9ncmFkZS5jb20iLCJodHRwczovL3Jlc3RvZ3JhZGUuZXUuYXV0aDAuY29
tL3VzZXJpbmZvIl0sImlhdCI6MTU4OTc3NTA3MiwiZXhwIjoxNTg5ODYxNDcyLCJhenAiOiJPTEtOb
jM4OVNVSW11ZkV4Z1JHMVJpbExTZ2RZeHdFcCIsInNjb3BlIjoib3BlbmlkIHByb2ZpbGUgZW1haWw
gb2ZmbGluZV9hY2Nlc3MifQ.XzJOXtTXOGOSbCFvp4yZGJzh7XhMmOmI2XxtjWdlODz_siI-
u8h11elcr8LwX6-hL20QOW0eStzBzmm1FM_tS7MxuKkYx8QlTWOURPembVKZOhNi8kN-
1j0pyc0uzve7Jib5vcxmkPwqpcVDFACgP85_0NYe4zXHKxCA5_8VOn05cRCDSkNMTFzGJCT9ipCcNX
aVGdksojYGqQzezjpzzzwrtPEkiyFLFtDPZAl0MleF3oFAOCBK0UKuNjJ_cSBbUsaIwfvK0WH47AwF
rRn_TxL4S1P3j3b1GgBm8tAqXysY84VZu0rSg3zrZj1PnoqPD4mbOXds20xafCr9wR4WTQ

vSvhNDeQLqrzRbvA2eeYE2PthB1cBimS
```

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

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

https://pragmaticwebsecurity.com

The technicalities of JSON Web Tokens

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
J1c2VyIjoiZTcyZDFhMjZmNDBlNGU4Nzk5NjciL
CJ0ZW5hbnQiOiJkOGNmM2ZhMzAxYTM0Yzk2ODUw
MmE3MDUxYmZkYzBhOCIsImlhdCI6MTYyMDE5MjY
0NDkxNCwiZXhwIjoxNjIwMTk2MjQ0OTE0fQ.bnd
YFgq1sHD-
vH8h1lARD8M0uZgoALThQu7CURkuSVs
```

The base64-encoded header and payload, along with the signature

The signature is crucial to ensure the integrity of the header and payload

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

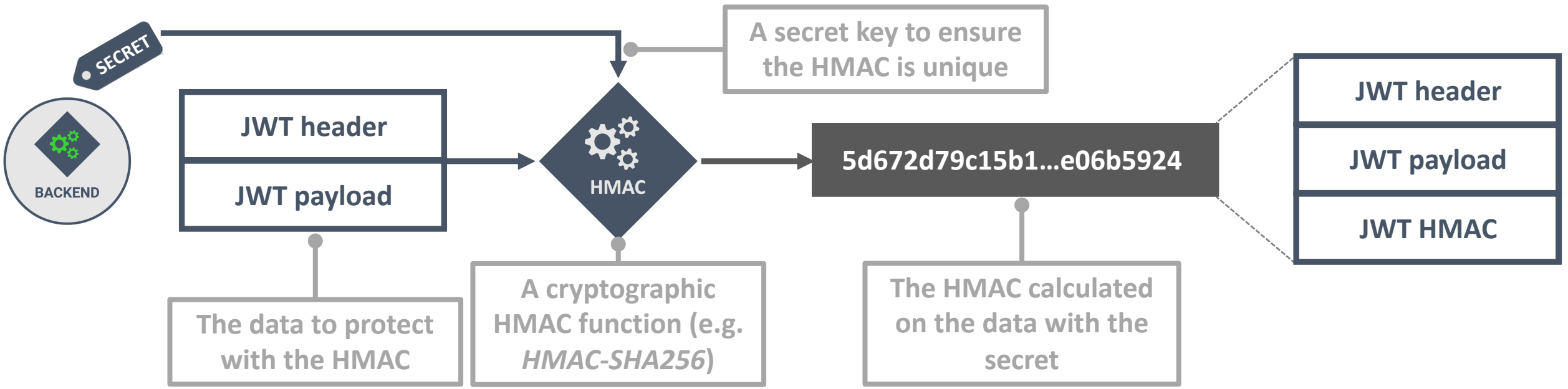
```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

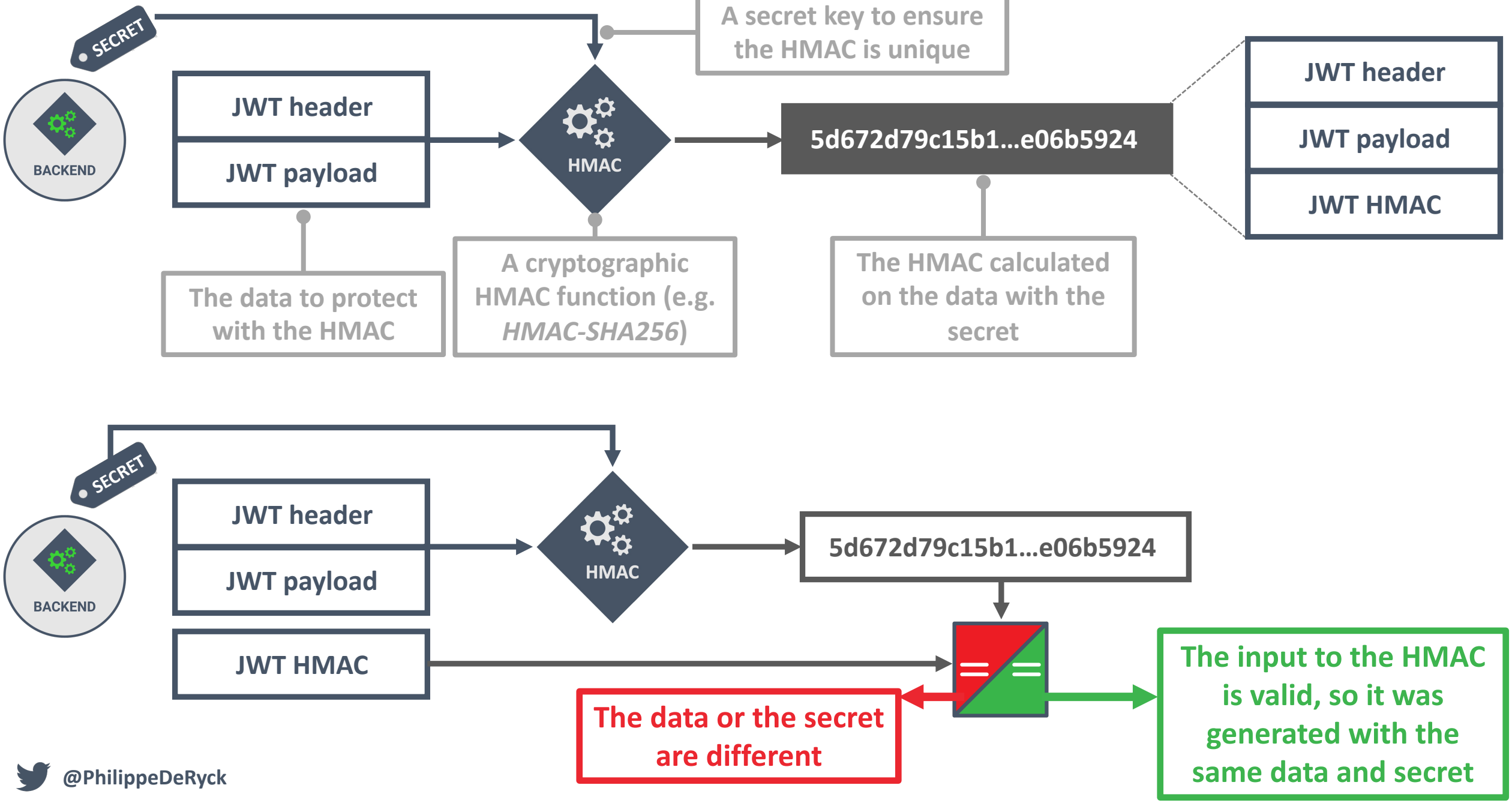
PAYLOAD: DATA

```
{
  "user": "e72d1a26f40e4e879967",
  "tenant": "d8cf3fa301a34c968502a7051bfdc0a8",
  "iat": 1620192644914,
  "exp": 1620196244914
}
```

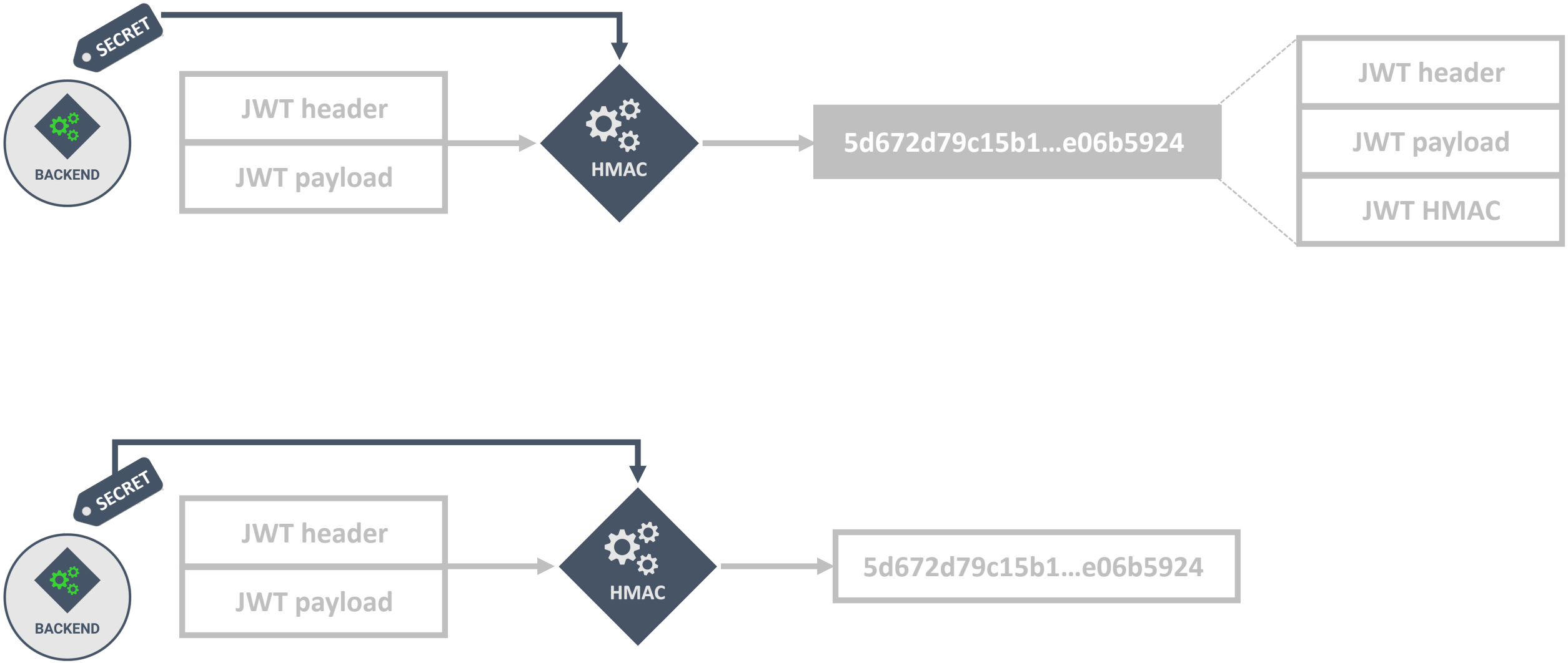
VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  SuperSecretHMACKey
) ☐ secret base64 encoded
```


SECRET
BACKEND
A secret key to ensure the HMAC is unique
JWT header
JWT payload
HMAC
5d672d79c15b1…e06b5924
JWT header
JWT payload
JWT HMAC
The data to protect with the HMAC
A cryptographic HMAC function (e.g. *HMAC-SHA256*)
The HMAC calculated on the data with the secret



@PhilippeDeRyck


SECRET
BACKEND
JWT header
JWT payload
HMAC
5d672d79c15b1…e06b5924
JWT header
JWT payload
JWT HMAC
SECRET
BACKEND
JWT header
JWT payload
HMAC
5d672d79c15b1…e06b5924

HMACs CANNOT BE USED IN DISTRIBUTED SCENARIOS

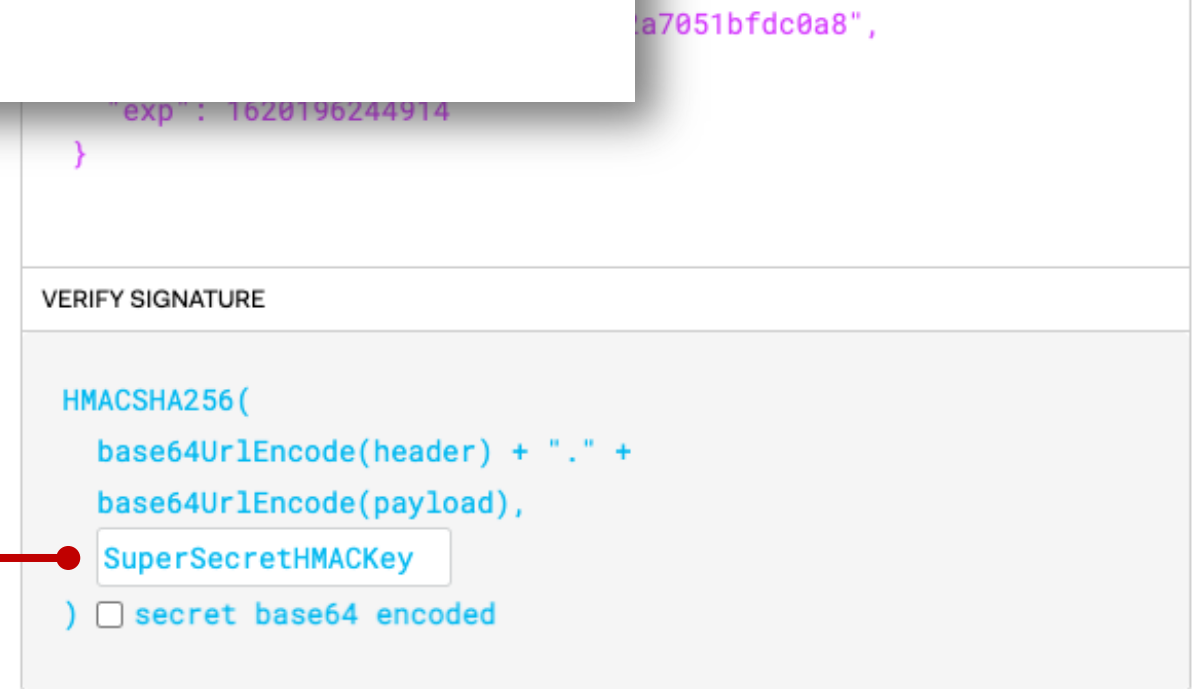
HMAC generation and verification happens with the same secret. Any verifier can also generate arbitrary tokens.

Decoded EDIT THE PAYLOAD AND SECRET

WEB APPLICATION SECURITY

Meet JWT heartbreaker, a Burp extension that finds thousands weak secrets automatically

OCTOBER 1, 2020 - 2 MINS READ



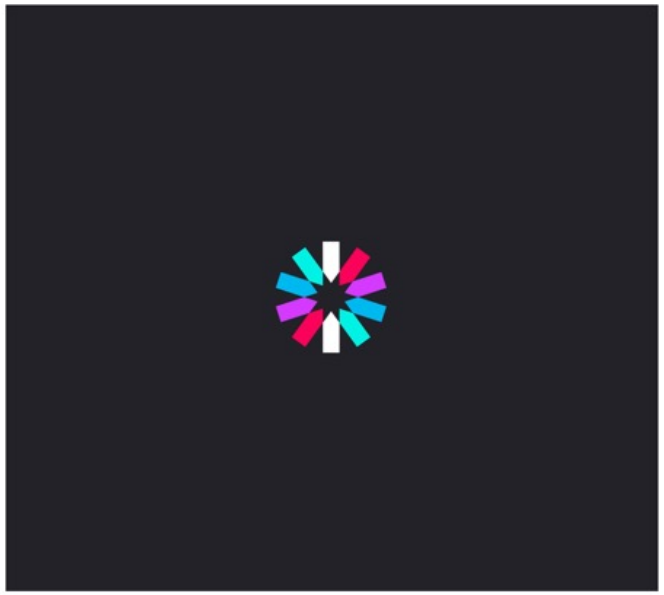
Your secret should be more random, and should not be published on a Powerpoint slide

https://lab.wallarm.com/meet-jwt-heartbreaker-a-burp-extension-that-finds-thousands-weak-secrets-automatically/

VULNERABILITIES

Brute Forcing HS256 is Possible: The Importance of Using Strong Keys in Signing JWTs

Cracking a JWT signed with weak keys is possible via brute force attacks. Learn how Auth0 protects against such attacks and alternative JWT signing methods provided.



Prosper Otemuyiwa
Former Auth0 Employee

March 23, 2017

oded EDIT THE PAYLOAD AND SECRET

R: ALGORITHM & TOKEN TYPE

```
alg": "HS256",
typ": "JWT"
```

AD: DATA

```
user": "e72d1a26f40e4e879967",
tenant": "d8cf3fa301a34c968502a7051bfdc0a8",
iat": 1620192644914,
exp": 1620196244914
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  SuperSecretHMACKey
) secret base64 encoded
```

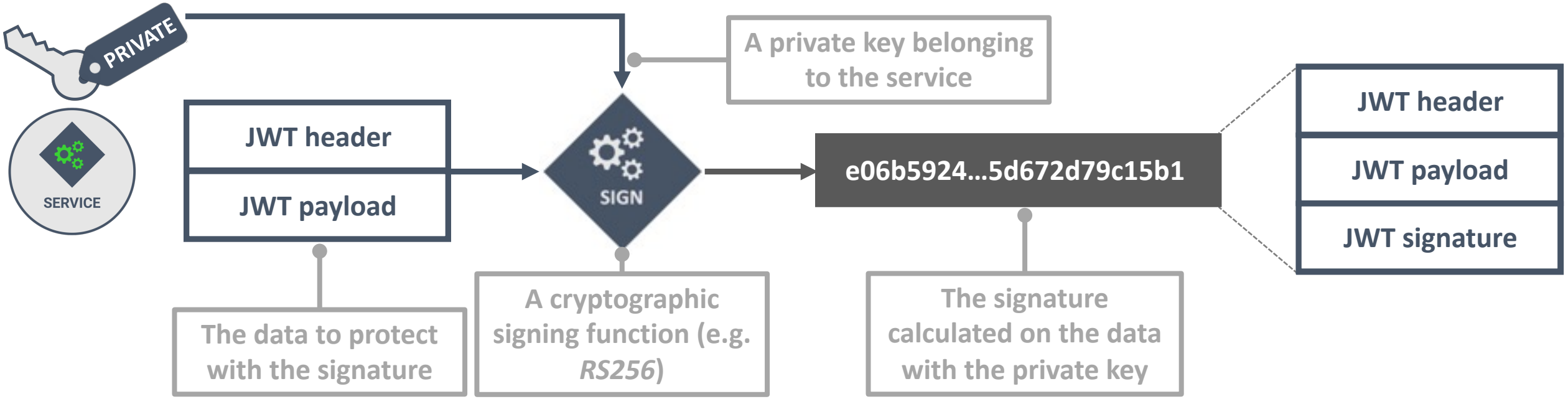
A key of the same size as the hash output (for instance, 256 bits for "HS256") or larger MUST be used with this algorithm.

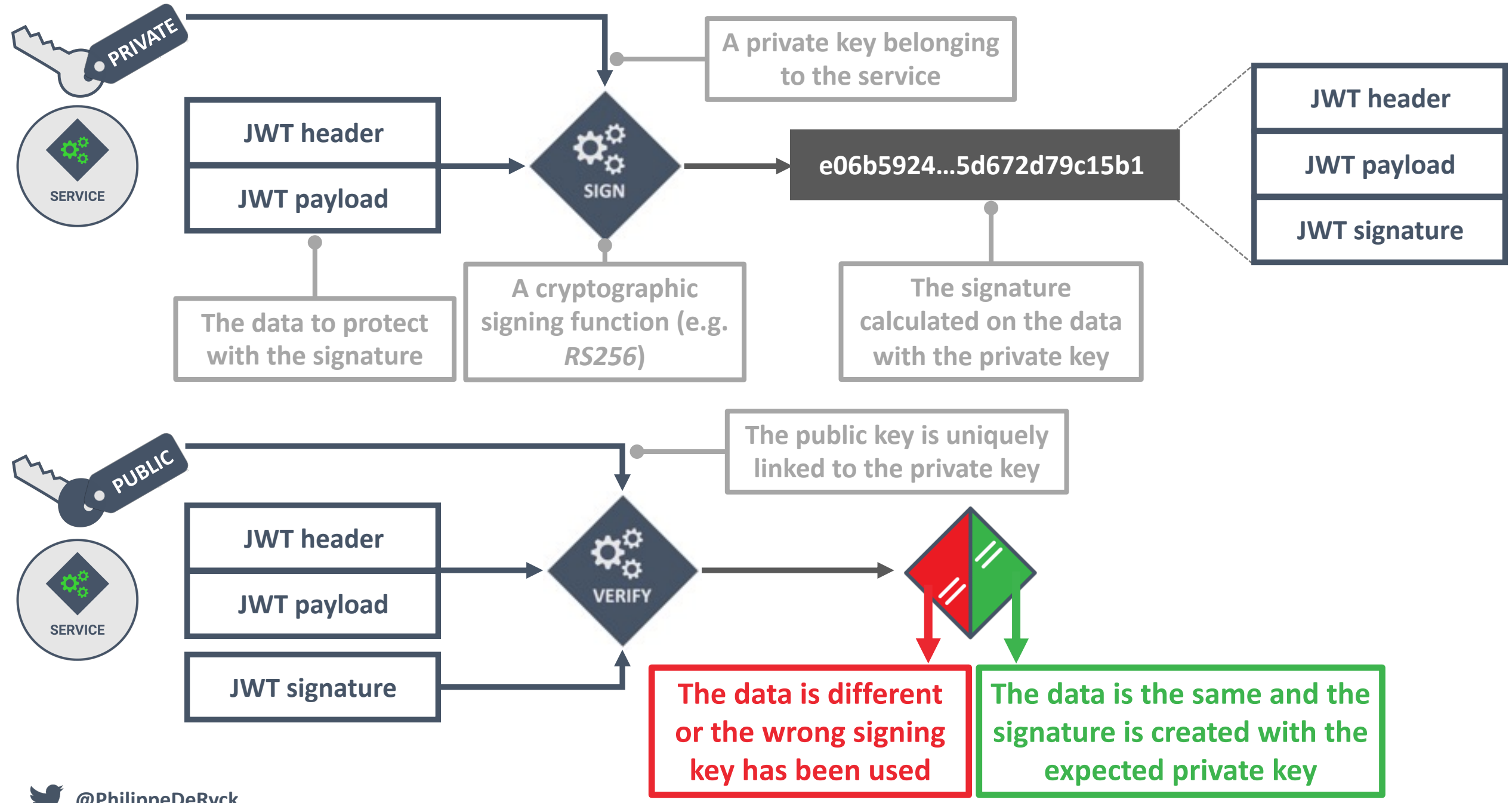
Your secret should be more random, and should not be published on a Powerpoint slide

https://auth0.com/blog/brute-forcing-hs256-is-possible-the-importance-of-using-strong-keys-to-sign-jwts/

AVOID HMACS AS MUCH AS POSSIBLE

HMACs with long-lived keys have fundamental weaknesses, so it's better to use public/private key signatures


PRIVATE
SERVICE
A private key belonging to the service
JWT header
JWT payload
SIGN
e06b5924...5d672d79c15b1
JWT header
JWT payload
JWT signature
The data to protect with the signature
A cryptographic signing function (e.g. *RS256*)
The signature calculated on the data with the private key



@PhilippeDeRyck

Which signature algorithm should you use?

WHICH SIGNING ALGORITHM SHOULD YOU USE?

- In the rare case that HMACs suit your needs, the default ***HS256*** is a solid choice

- For asymmetric signatures, the story is a bit more complicated
  - RS256 is most widely supported and used, and is still considered secure for signatures
    - In light of future-proofing implementations, the RSA crypto spec has deprecated *RS256*
      - *RS256* is actually JWT's shorthand for *RSASSA-PKCS1-v1_5*
    - Instead, the spec recommends the use of *RSASSA-PSS*, known in the JWT world as ***PS256***
      - *PS256*, *PS384*, and *PS512* are well supported by common JWT libraries
  - An even better alternative to RSA are elliptic curve digital signatures (ECDSA)
    - JWT libraries support ES256, which is unfortunately easy to misuse
    - Instead, you should use of ***EdDSA***, which has unfortunately little to no library support

- **TL;DR: Use *HS256* for HMACs and *PS256* for asymmetric signatures**

https://www.scottbrady91.com/JOSE/JWTs-Which-Signing-Algorithm-Should-I-Use

Use JWTs signed with a private key

The generator of a JWT uses the private key, but the verifiers all use the public key. PS256 is a robust choice for the signature algorithm.

Critical Vulnerabilities Affect JSON Web Token Libraries

Author:
Chris Brook

April 1, 2015 / 2:58 pm

3:30 minute read

Share this article:

https://threatpost.com/critical-vulnerabilities-affect-json-web-token-libraries/111943/

“

The Authentication API prevented the use of "alg: none" with a case sensitive filter. This means that simply capitalising any letter ("alg: nonE"), allowed tokens to be forged.

”

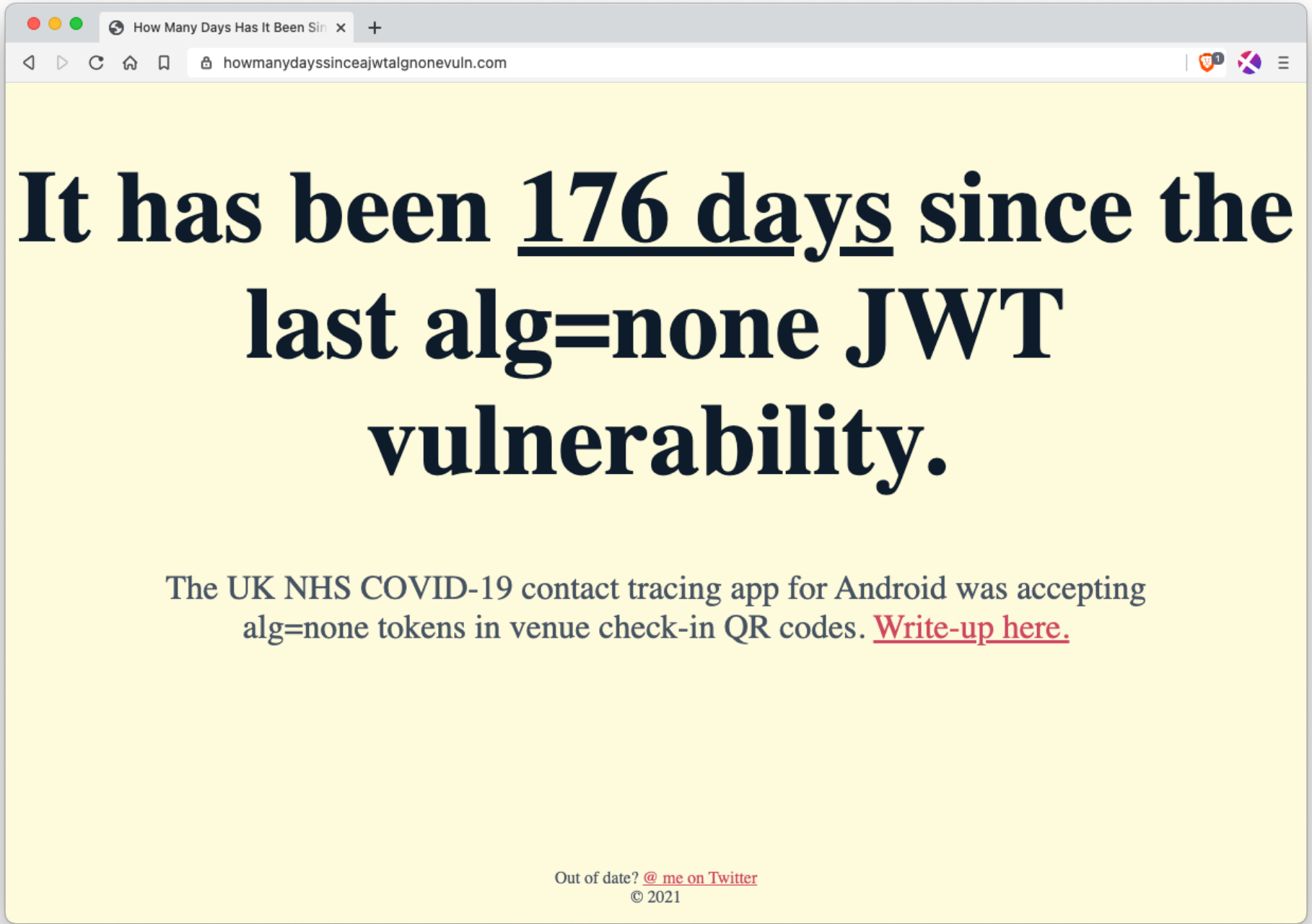
Ben Knight Senior Security Consultant

April 16, 2020

JSON Web Token Validation Bypass in Auth0 Authentication API

Ben discusses a JSON Web Token validation bypass issue disclosed to Auth0 in their Authentication API.

https://insomniasec.com/blog/auth0-jwt-validation-bypass



https://www.howmanydayssinceajwtalgnonevuln.com/

JSON Web Token Attacker

JOSEPH - JavaScript Object Signing and Encryption Pentesting Helper

This extension helps to test applications that use JavaScript Object Signing and Encryption, including JSON Web Tokens.

Features

- Recognition and marking
- JWS/JWE editors
- (Semi-)Automated attacks
  - Bleichenbacher MMA
  - Key Confusion (aka Algorithm Substitution)
  - Signature Exclusion
- Base64url en-/decoder
- Easy extensibility of new attacks

Author Dennis Detering

Version 1.0.2

Rating ☆ ☆ ☆ ☆ ☆

Popularity

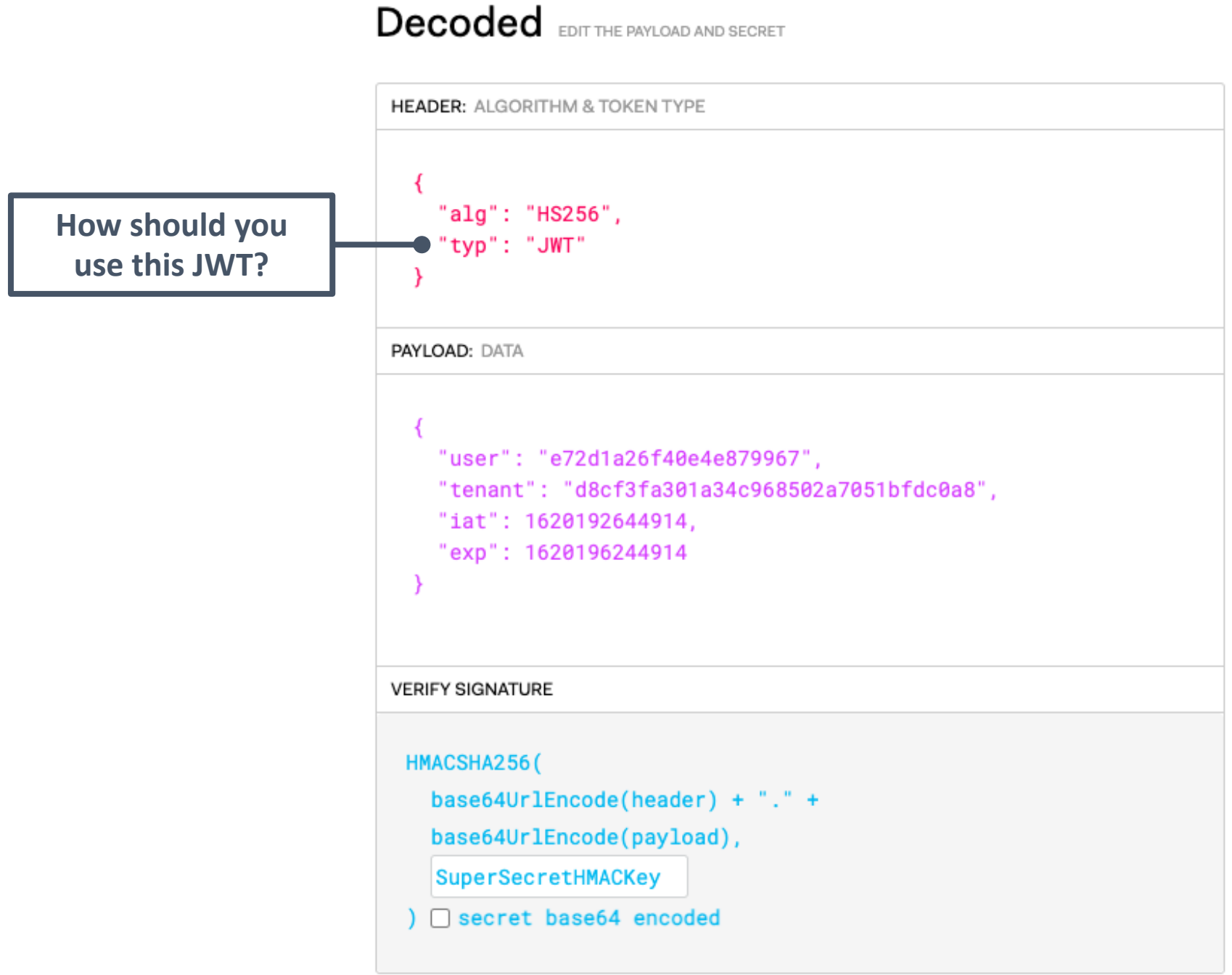
Last updated 08 February 2019

You can install BApps directly within Burp, via the BApp Store feature in the Burp Extender tool. You can also download them from here, for offline installation into Burp.

ASSERT THE *ALG* CLAIM MAKES SENSE

The* alg *claim in the header indicates how the token is signed. Ensure the claim corresponds to an expected value (or hardcode the algorithm)

How should you use this JWT?



HEADER: ALGORITHM & TOKEN TYPE


```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "NTVBOTU3MzBBOEUwNzhBQ0VYxQUUyOUNEQUUxNjEyMw"
}
```


JWTs should be explicitly typed. For an access token, the ***typ*** should be set to ***at+jwt*** to avoid token type confusion


```
  "iss": "https://sts.restograde.com/",
  "sub": "auth0|5ef6ef551b24320013b6c638",
  "aud": [
    "https://api.restograde.com",
    "https://restograde.eu.auth0.com/userinfo"
  ],
  "iat": 1599250282,
  "exp": 1599336682,
  "azp": "DtsTliLAWq3JXIwaoPQzl8vXhNI6qGnb",
  "scope": "openid email read:reviews delete:reviews"
}
```


HEADER: ALGORITHM & TOKEN TYPE


```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "NTVBOTU3MzBBOEUwNzhBQ0VYxQUUyOUNEQUUxNjEyMw"
}
```


PAYLOAD: DATA


```
{
  "email": "philippe@pragmaticwebsecurity.com",
  "email_verified": true,
  "iss": "https://sts.restograde.com/",
  "sub": "auth0|5ef6ef551b24320013b6c638",
  "aud": "DtsTliLAWq3JXIwaoPQzl8vXhNI6qGnb",
  "iat": 1599250282,
  "exp": 1599286282
}
```


Which one is the OAuth 2.-0 access token and which one is the OIDC identity token?

Explicit typing for JWTs

- JWTs are just a data representation and can be used for different scenarios
  - Due to reserved claims, many JWTs contain similar values
  - It can become tricky to differentiate between JWTs from the same service
    - OAuth 2.0 access tokens and OIDC identity tokens are issued by the same server
    - While both tokens contain similar claims, they serve a completely different purpose
    - An attacker could gain API access by using an identity token, which should never happen
- JWT best practices recommend explicit JWT typing
  - Instead of the generic ***JWT*** type, applications should use a custom type
  - E.g., the recommendation for OAuth 2.0 access tokens is to use ***at+jwt***
- Explicit typing is highly recommended for custom JWTs
  - Only accept JWTs with proper typing and reject everything else

Use explicit typing for JWTs

The* typ *claim in the header indicates the type of JWT token. Verify the type after having verified the signature to avoid token confusion.

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "userid": "12",
  "name": "Philippe De Ryck",
  "admin": true,
  "exp": 1620196244914
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  SuperSecretHMACKey
) ☐ secret base64 encoded
```

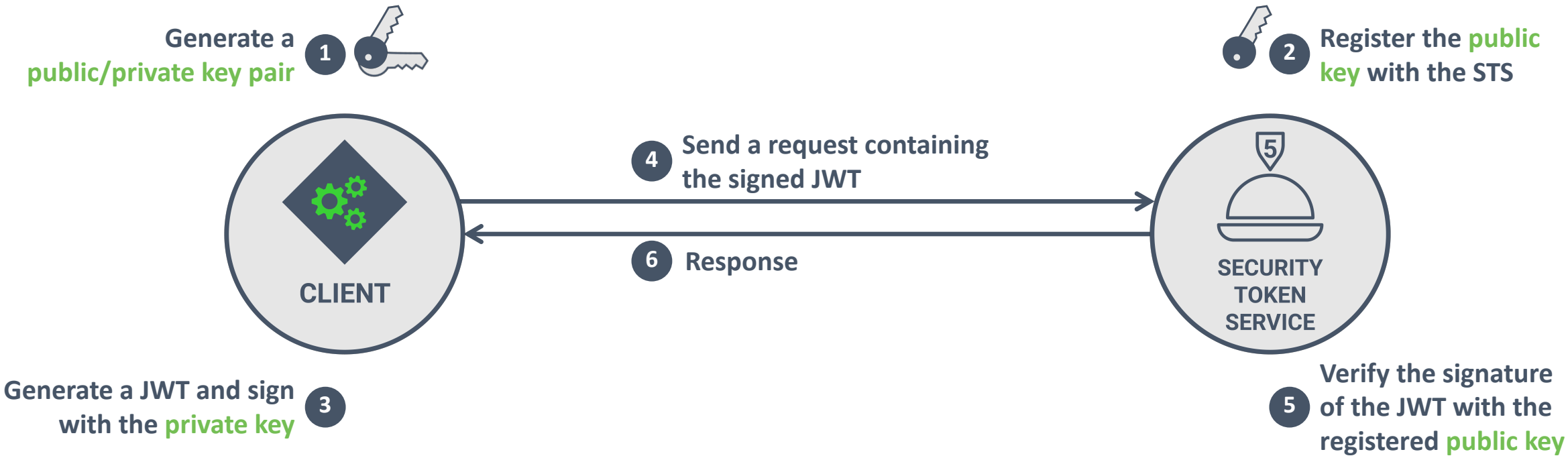
JWTs ARE JUST A REPRESENTATION OF CLAIMS

JWTs support the secure representation of claims, nothing more. You are responsible for what you build with JWTs and how you handle them.

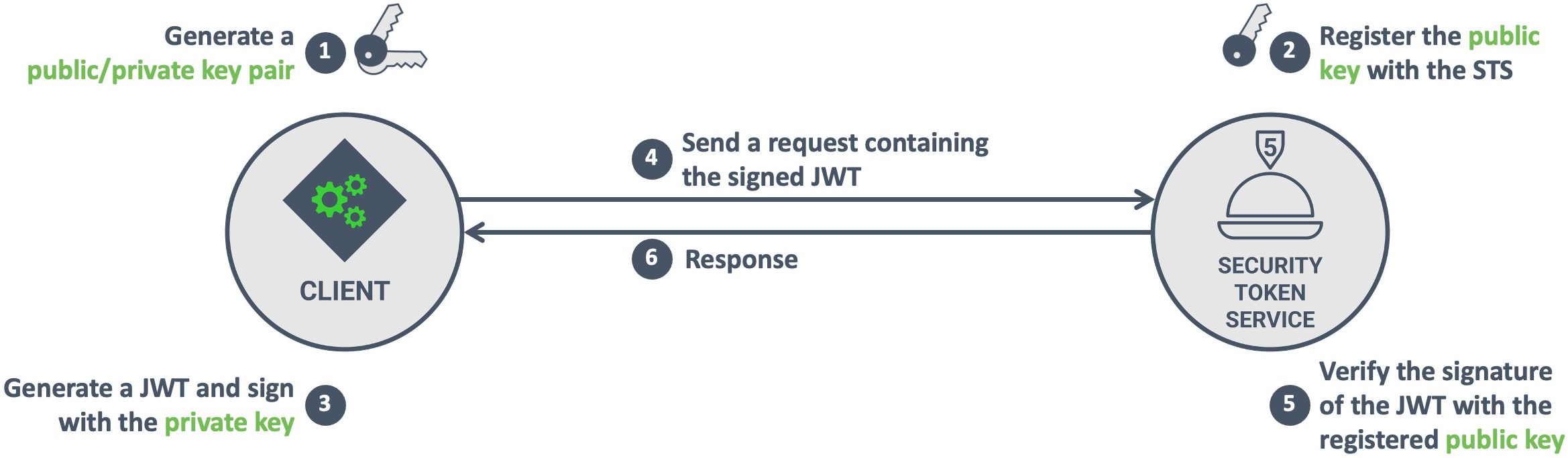
USING JWTS IN PRACTICE

JWTs play a crucial role in the OAuth 2.0 and OpenID Connect ecosystem

KEY-BASED CLIENT AUTHENTICATION WITH JWT TOKENS



KEY-BASED CLIENT AUTHENTICATION WITH JWT TOKENS



The payload of the generated JWT

```
{
  "iss": "test_client_jwt",
  "sub": "test_client_jwt",
  "aud": "https://sts2.restograde.com/auth/realms/Restograde",
  "iat": 1590316085,
  "exp": 1590316100,
  "jti": "77bef630-361c-486b-bc68-763c6c1d8900"
}
```

- "sub": The ID of the authenticating client
- "aud": The identifier of the STS
- "iat": The generation time of the JWT
- "exp": The expiration time of the JWT
- "jti": A unique value to prevent replay

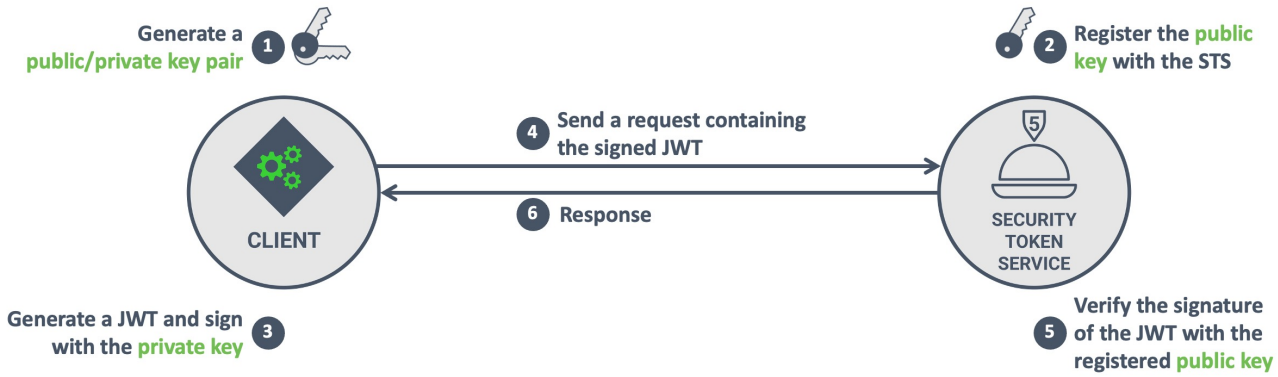
KEY-BASED CLIENT AUTHENTICATION WITH JWT TOKENS

The authentication request containing the JWT

```
1  POST /auth/realms/Restograde/protocol/openid-connect/token
2  Host: sts2.restograde.com
3
4   grant_type=client_credentials
5  &client_id=test_client_jwt
6  &client_assertion_type=urn:ietf:params:oauth:client-
7   assertion-type:jwt-bearer
8  &client_assertion=eyJhbGciOiJSUz…ZuTnMNQ
```

- Running the *client credentials* flow
- The ID of the authenticating client
- Indicating JWT-based authentication
- The JWT signed by the client

KEY-BASED CLIENT AUTHENTICATION WITH JWT TOKENS



JWTs CAN BE USED FOR CLIENT AUTHENTICATION

RFC 7523 defines how to use JWTs for key-based OAuth 2.0 client authentication (along with a custom grant based on JWTs).

An OAuth 2.0 initialization URI

```
1  https://sts.restograde.com/authorize
2    ?response_type=code
3    &client_id=lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv
4    &scope=read
5    &redirect_uri=https://app.restograde.com/callback
6    &state=s0wzojm2w8c23xzprkk6
7    &code_challenge=JhEN0Amnj7B…Wh5PxWitZYK1woWh5PxWitZY
8    &code_challenge_method=S256
```

- Indicates the *authorization code flow*
- The client requesting access
- Where the STS should send the code
- The PKCE code challenge
- The PKCE hash function

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

These shortcomings can result in advanced attacks, such as **Redirection URL rewriting** or **Mix-up attacks**

JWT Secured Authorization Requests (JAR)
use JWTs to generate initialization URIs

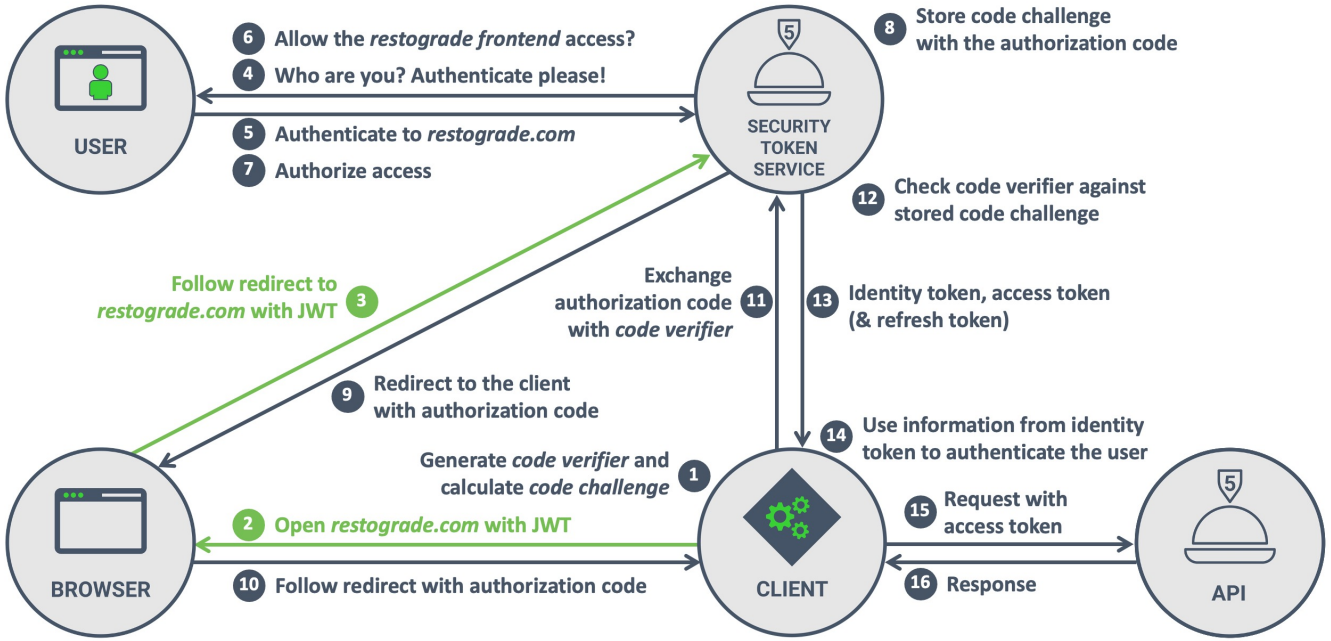
An OAuth 2.0 initialization URI

```
1  https://sts.restograde.com/authorize
2    ?client_id=lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv
3    &request=eyJhbGciOiJQUzI1NiIsInR5cCI6Im9hdXRoLWF1dGh6LX
4    JlcStqd3QifQ.eyJpc3MiOiJsWTVnMEJLQjdNb3c0eURsYjZyZEdQc0
5    8yaTFnN09zdiIsImF1ZCI6Imh0dHBzOi8vc3RzLnJlc3RvZ3JhZGUuY
6    …
7    a8JSiQtbP4IKzGXvHoJvPh-T4OxgA9QZj9erIT2wEVBcieAO0340zl2
8    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


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


The payload of the decoded JWT object


```json
{
  "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

```
1  {
2    "alg": "PS256",
3    "typ": "oauth-authz-req+jwt",
4    "kid": "hhbGxlbmdlIjoiSmhFT"
5  }
```

The JWT is explicitly typed

The payload of the decoded JWT object

```
1  {
2    "iss": "lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv",
3    "aud": "https://sts.restograde.com",
4    "response_type": "code",
5    "client_id": "lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv",
6    "redirect_uri": "https://app.restograde.com/callback",
7    "scope": "read",
8    "state": "s0wzojm2w8c23xzprkk6",
9    "code_challenge": "JhEN0Amnj … xWitZYK1woWh5PxWitZY",
10   "code_challenge_method": "S256"
11 }
```

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

JWT Secured Authorization Request (JAR)

- JAR allows the client to provide the flow configuration as a JWT
  - Contrary to plain URL parameters, the JWT is signed by the client
    - A signed JWT provides both data integrity and authenticity
  - If preferred, the client can also encrypt the request JWT for confidentiality

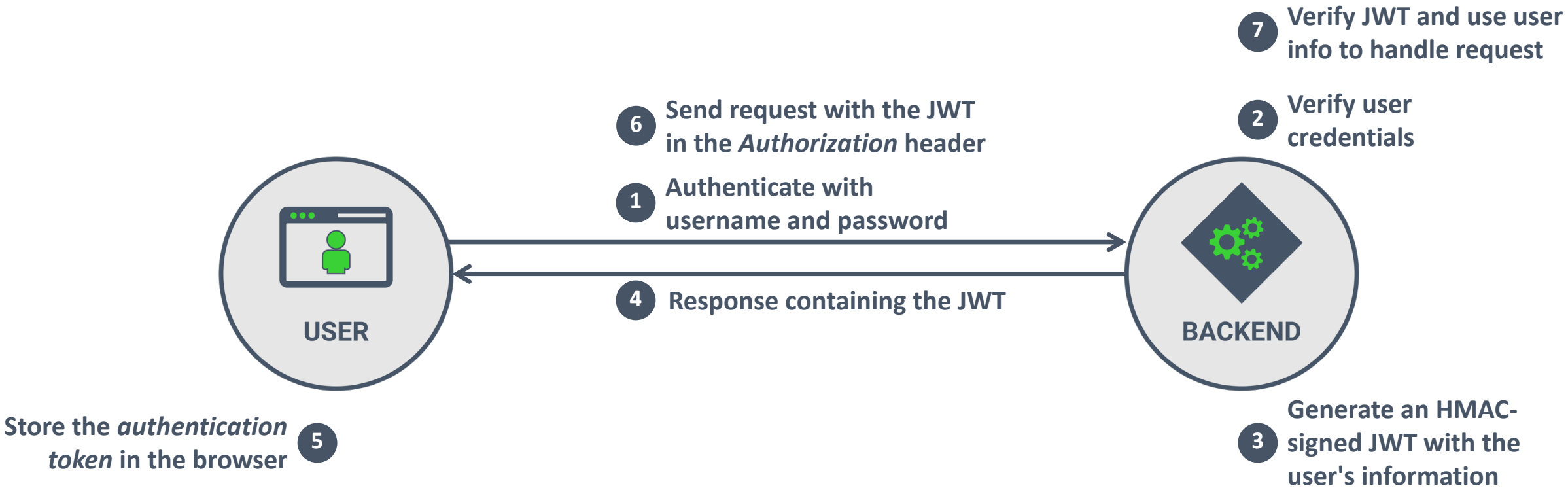
- The JWT signing key of the client must be registered with the STS
  - For confidential clients, this happens during client registration
  - For native public clients, this can be done with *dynamic client registration*
  - Web-based public clients do not benefit from JAR, since they already run in the browser

- The JAR specification is currently a draft, with limited implementation support
  - JAR is considered extremely useful and will become widely supported when finalized

JWTs CAN BE USED FOR PARAMETER INTEGRITY

The upcoming JAR specification defines how to use JWTs to guarantee the integrity of URL parameters in redirect-based mechanisms.

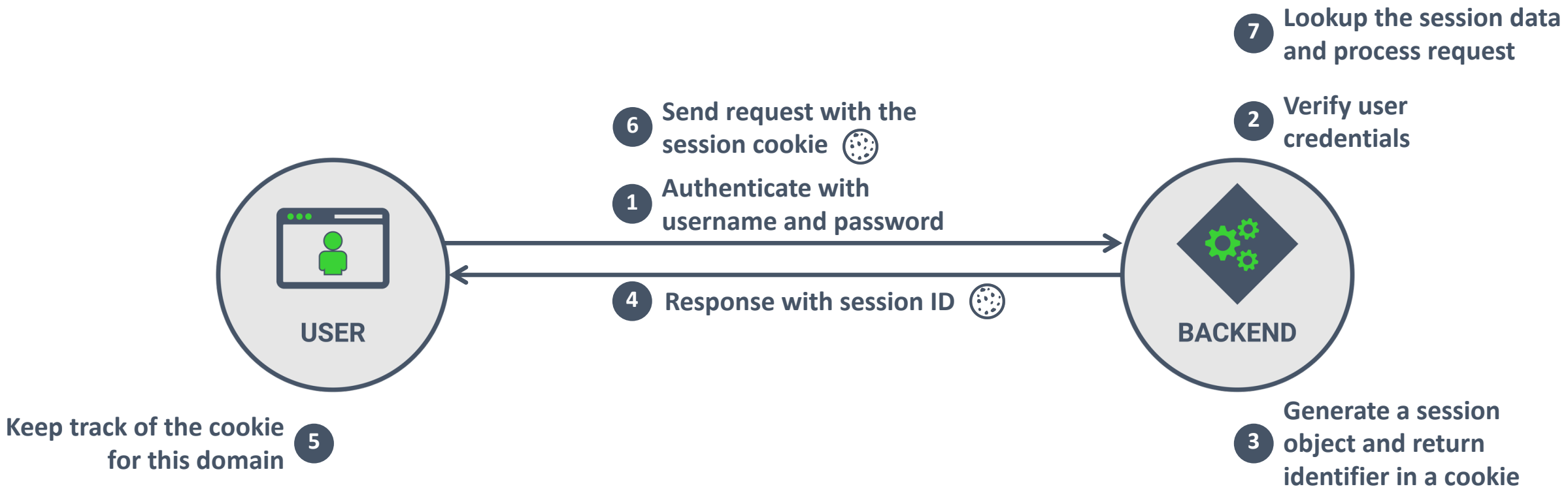
What about *token-based authentication?*

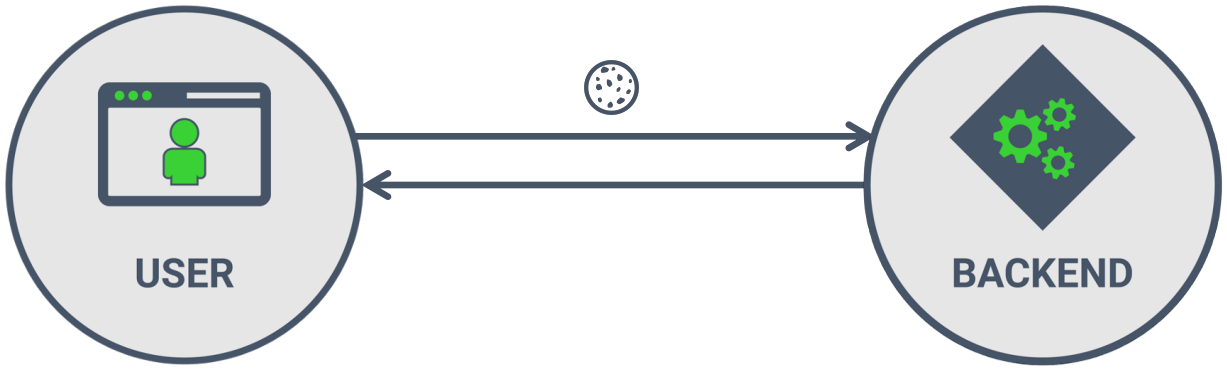


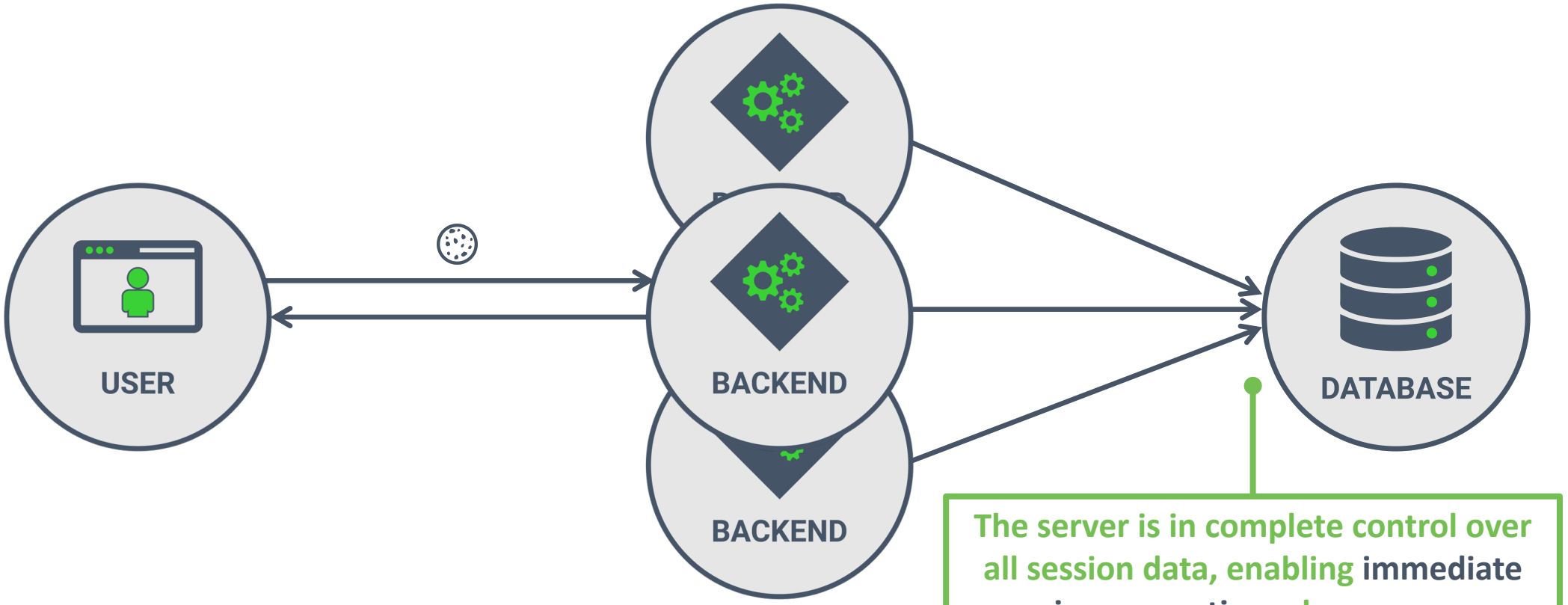

```
{
  "userid": "12",
  "name": "Philippe De Ryck",
  "admin": true,
  "exp": 1620196244914
}
```

Many applications use a JWT as a replacement for a traditional server-side session object …

MANAGING THE TOKEN LIFECYCLE

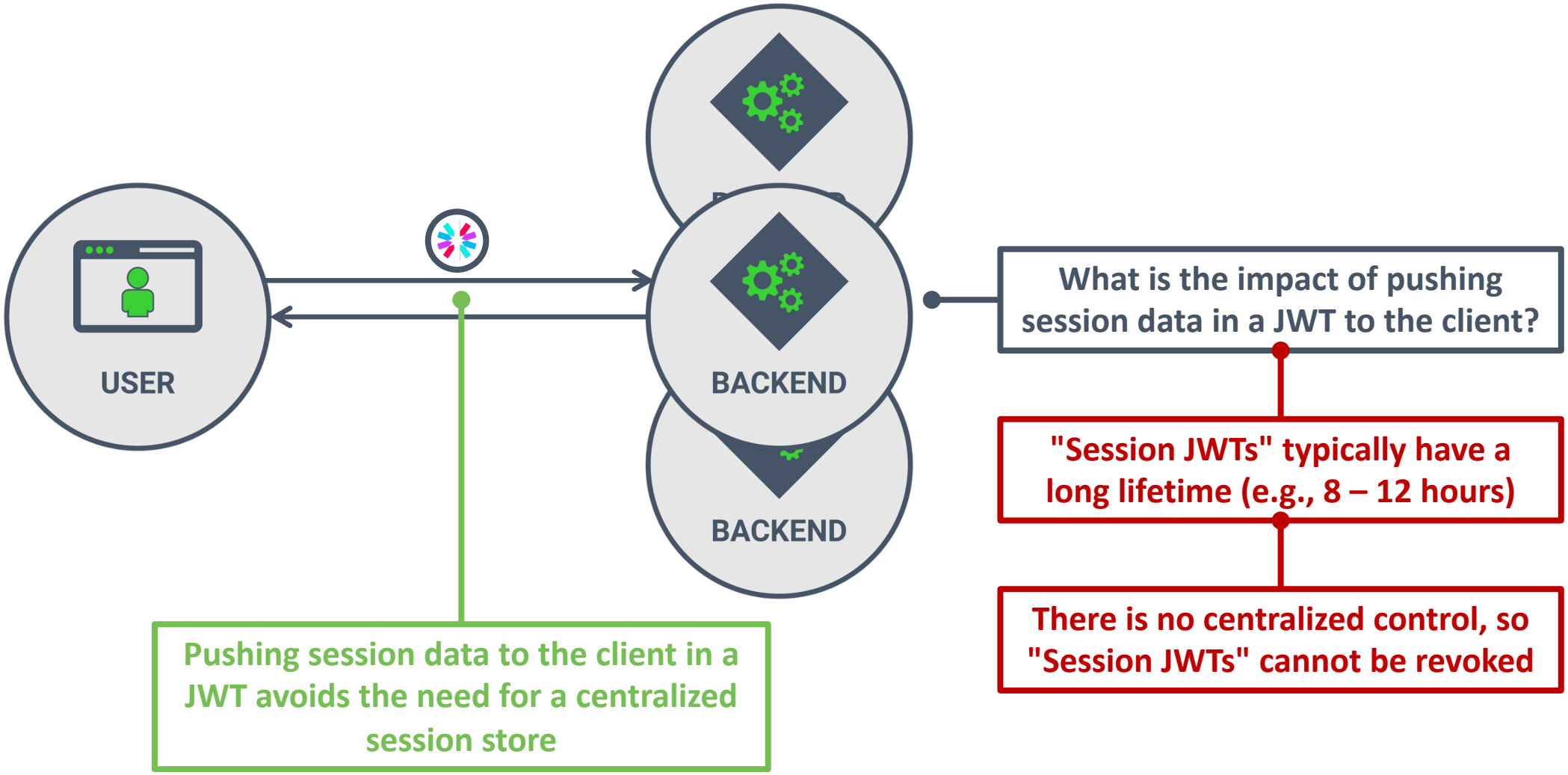

7 Lookup the session data and process request
6 Send request with the session cookie
2 Verify user credentials
1 Authenticate with username and password
USER
BACKEND
4 Response with session ID
Keep track of the cookie for this domain
5
3 Generate a session object and return identifier in a cookie

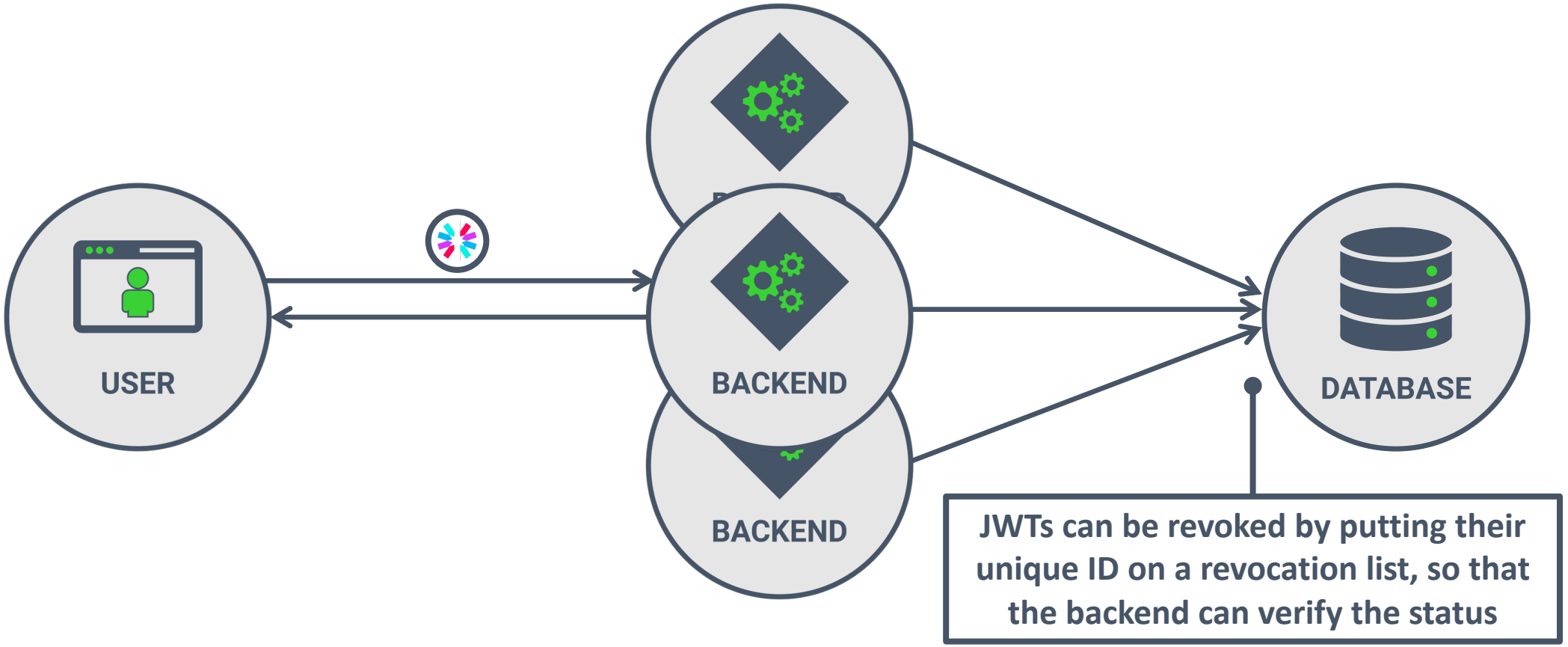

USER
BACKEND

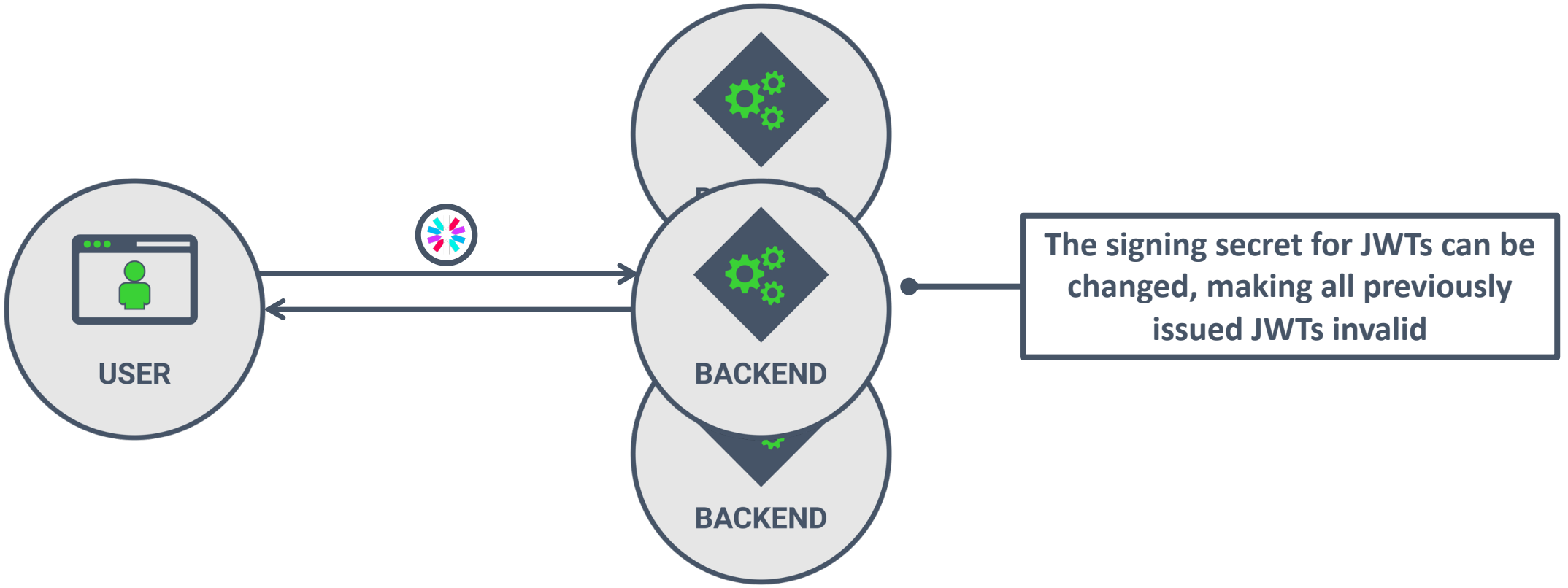


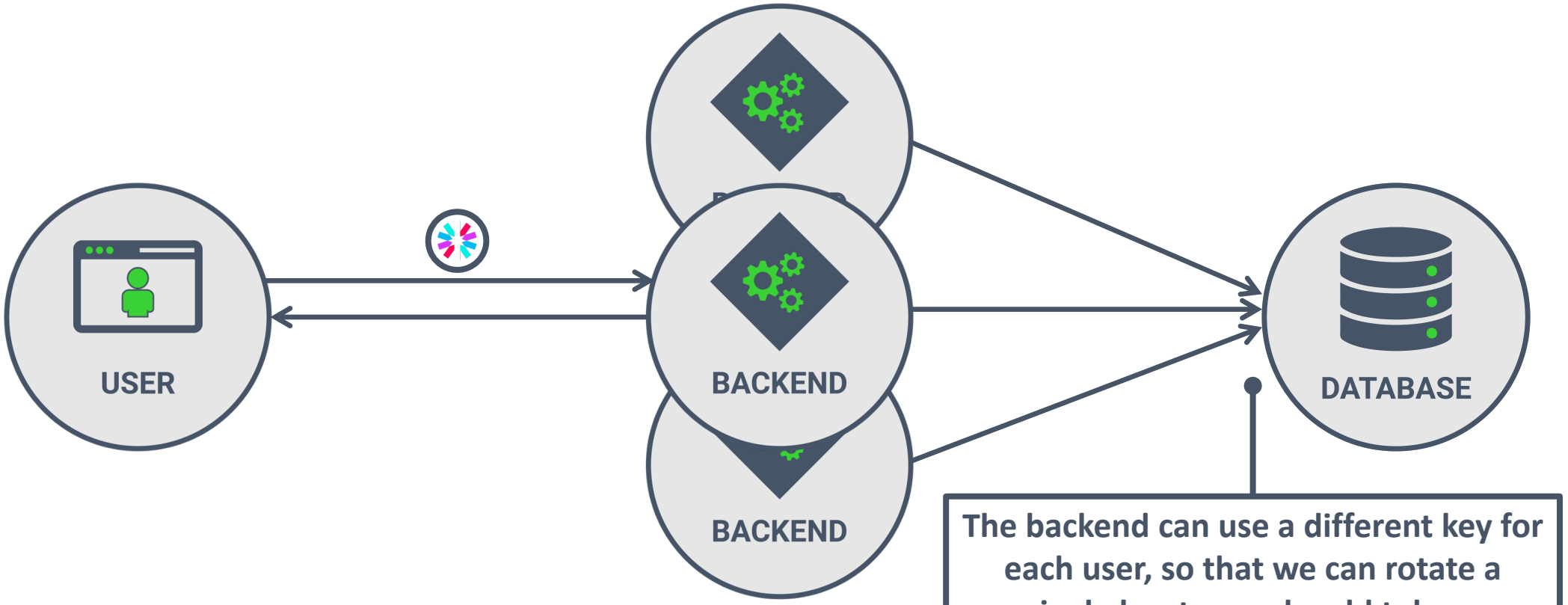
The server is in complete control over all session data, enabling immediate session revocation when necessary

Scaling a system with server-side sessions requires session replication or sticky sessions


USER
BACKEND
BACKEND
What is the impact of pushing session data in a JWT to the client?
"Session JWTs" typically have a long lifetime (e.g., 8 – 12 hours)
There is no centralized control, so "Session JWTs" cannot be revoked
Pushing session data to the client in a JWT avoids the need for a centralized session store


USER
BACKEND
BACKEND
DATABASE
JWTs can be revoked by putting their unique ID on a revocation list, so that the backend can verify the status


USER
BACKEND
BACKEND
The signing secret for JWTs can be changed, making all previously issued JWTs invalid



The backend can use a different key for each user, so that we can rotate a single key to revoke old tokens

This pattern requires using data from the JWT before it is verified to retrieve the correct signature verification key

joepie91's Ramblings RSS

Stop using JWT for sessions

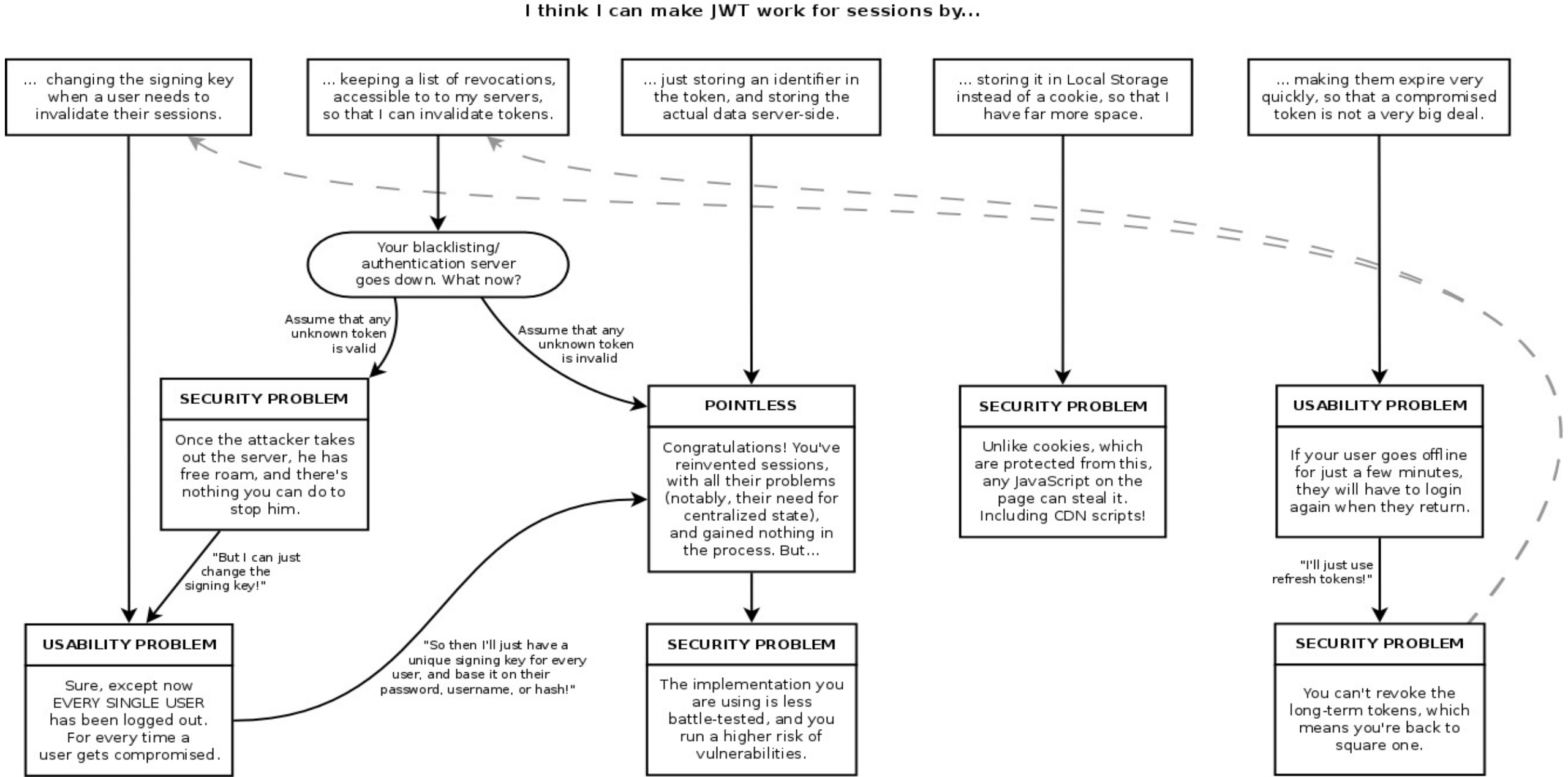
13 Jun 2016

Update - June 19, 2016: A lot of people have been suggesting the same "solutions" to the problems below, but none of them are practical. I've published a new post with a slightly sarcastic flowchart - please have a look at it before suggesting a solution.

> This article does *not* argue that you should *never* use JWT - just that it isn't suitable as a session mechanism, and that it is dangerous to use it like that. Valid usecases *do* exist for them, in other areas.

Stop using JWT for sessions, part 2

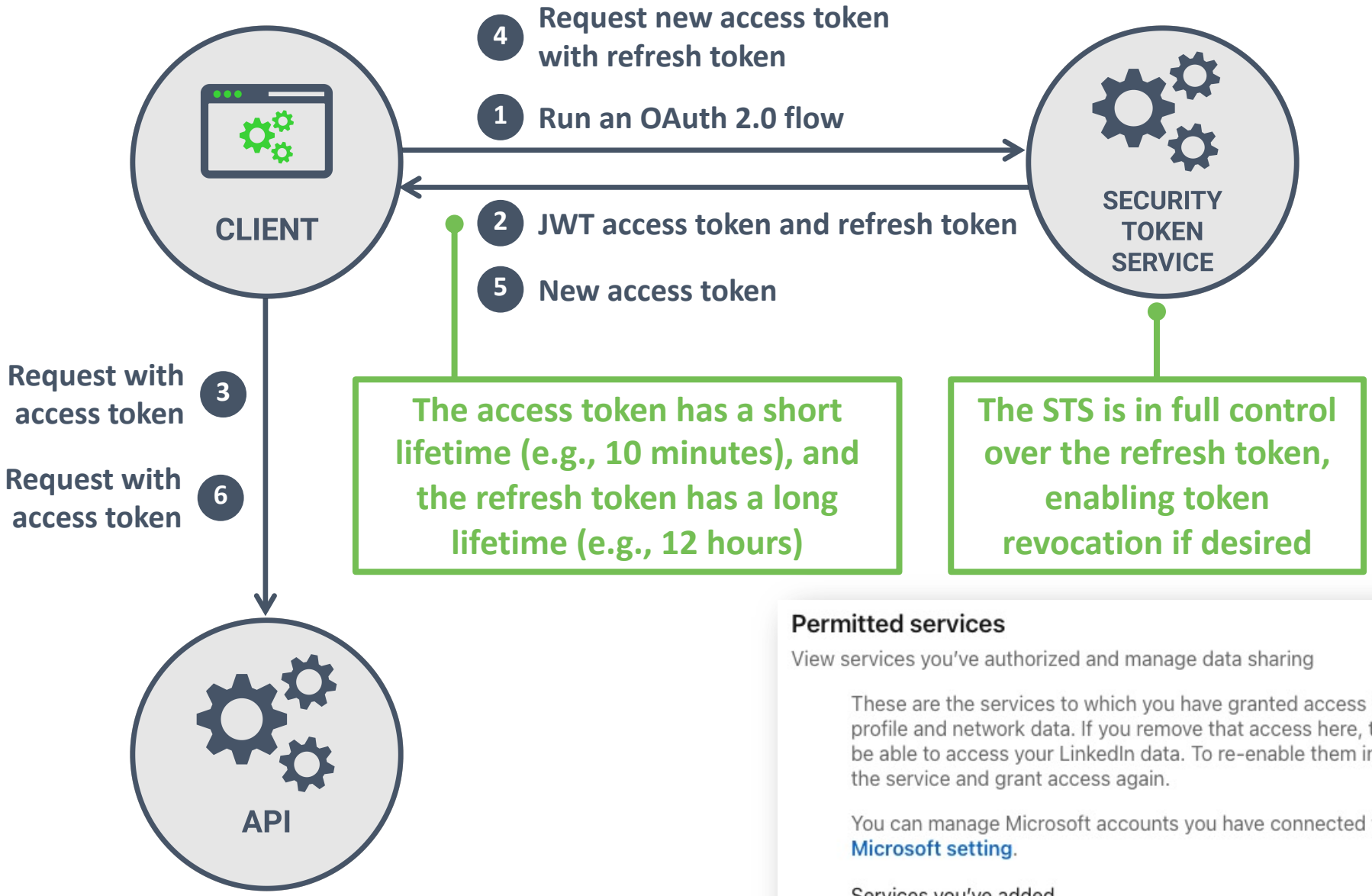
A handy dandy (and slightly sarcastic) flowchart about why your "solution" doesn't work

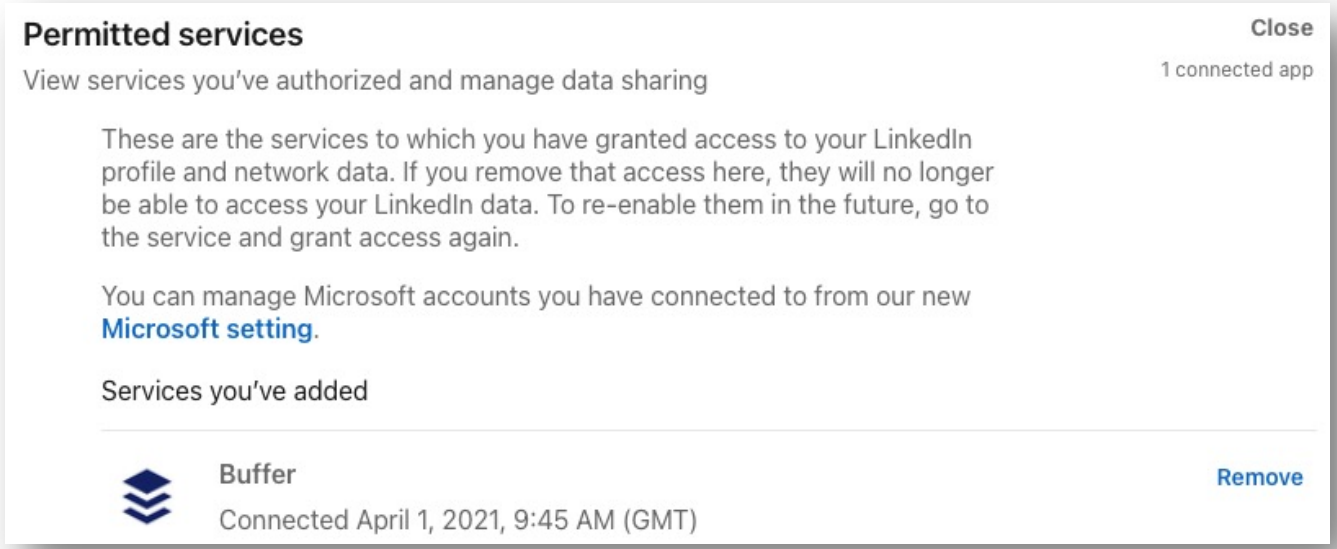


JWTs ARE JUST A WAY TO REPRESENT CLAIMS

JWTs are not a session mechanism and should not be used as one. Using JWTs as authorization tokens requires a supporting ecossytem.

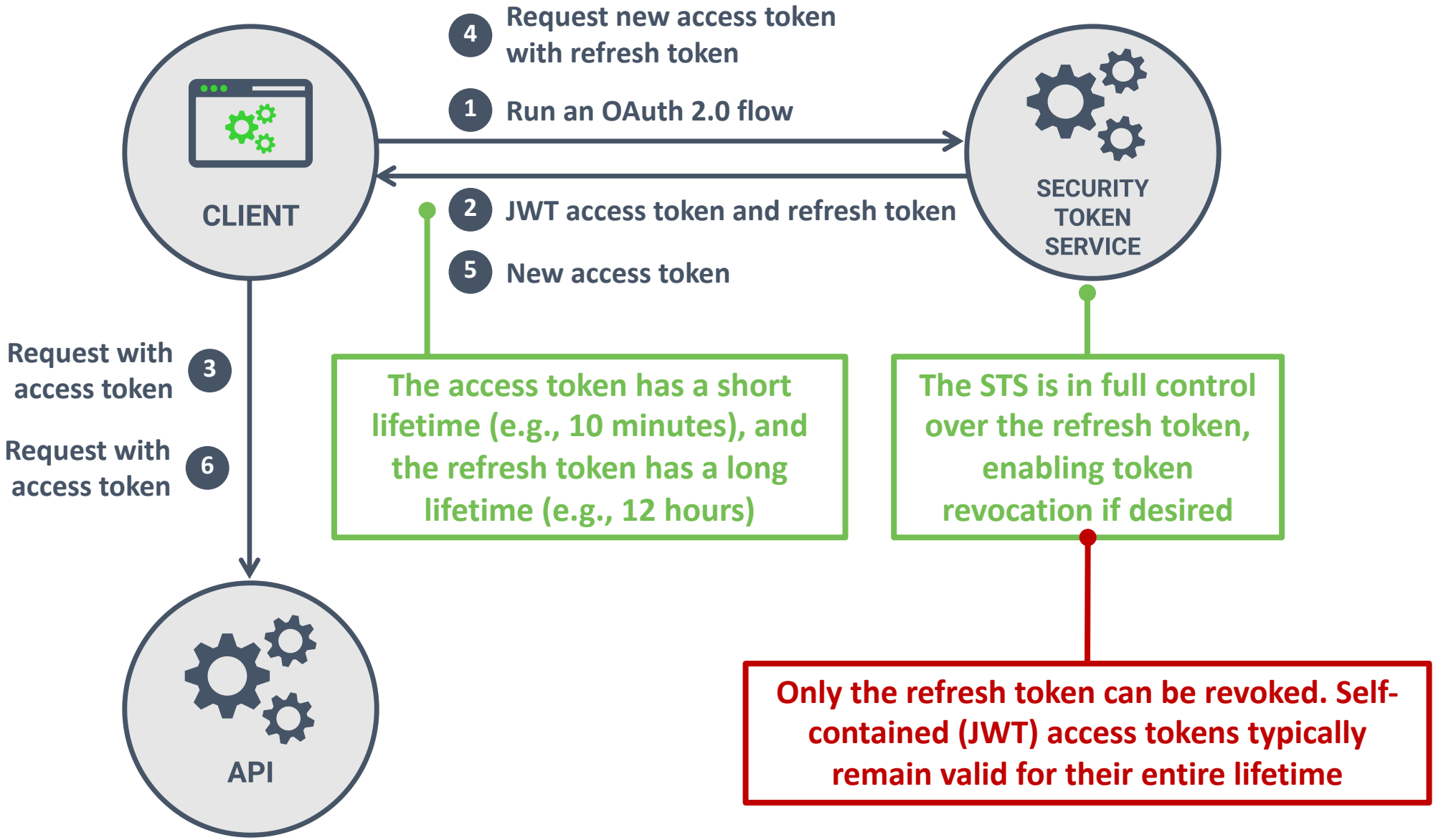
OAuth 2.0 refresh tokens are crucial to improve the properties of access tokens

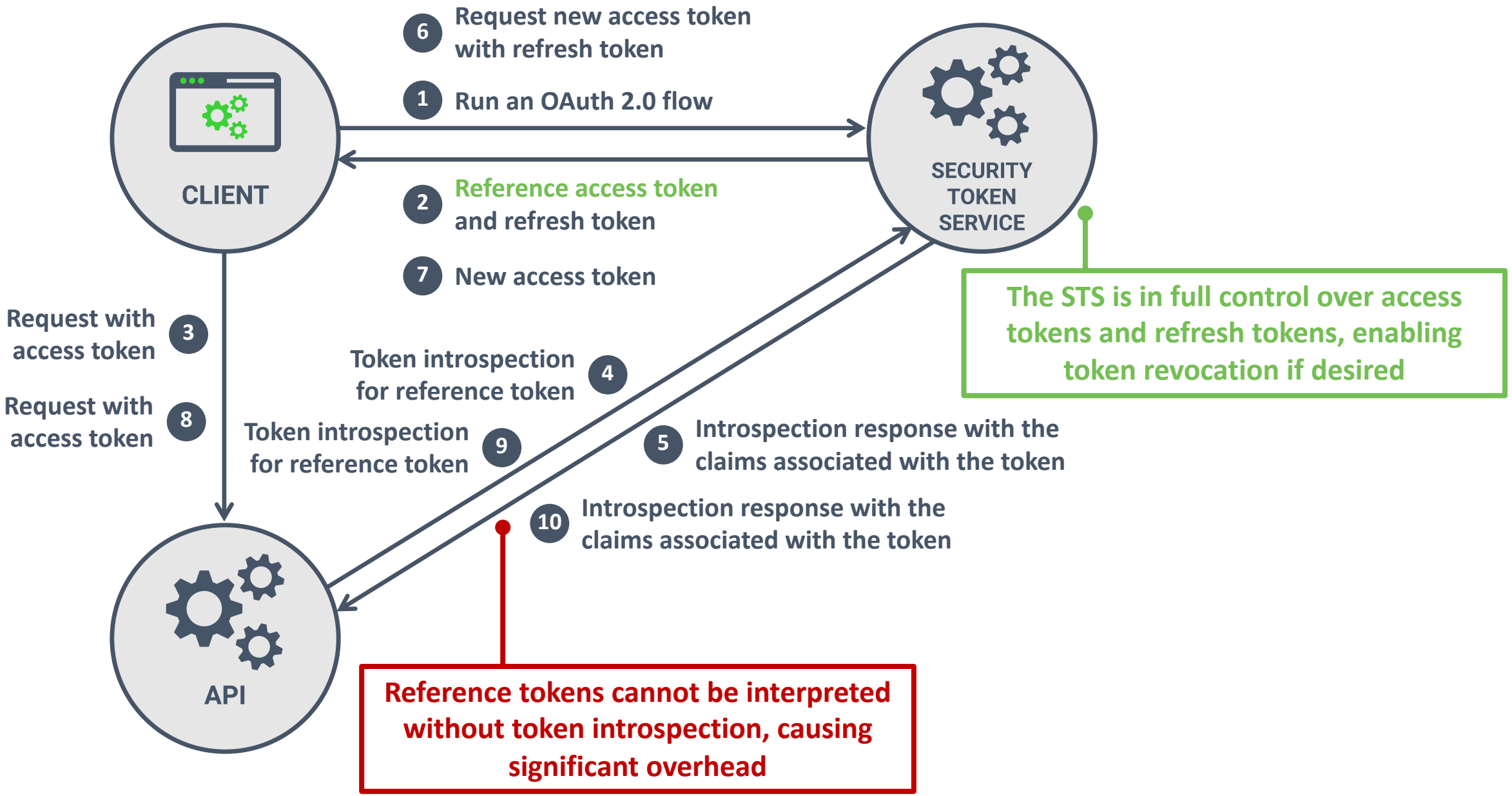

CLIENT
SECURITY TOKEN SERVICE
API
4 Request new access token with refresh token
1 Run an OAuth 2.0 flow
2 JWT access token and refresh token
5 New access token
Request with access token 3
Request with access token 6
The access token has a short lifetime (e.g., 10 minutes), and the refresh token has a long lifetime (e.g., 12 hours)
The STS is in full control over the refresh token, enabling token revocation if desired


Permitted services
Close
View services you've authorized and manage data sharing
1 connected app
These are the services to which you have granted access to your LinkedIn profile and network data. If you remove that access here, they will no longer be able to access your LinkedIn data. To re-enable them in the future, go to the service and grant access again.
You can manage Microsoft accounts you have connected to from our new Microsoft setting.
Services you've added
Buffer
Remove
Connected April 1, 2021, 9:45 AM (GMT)

JWT ACCESS TOKENS ARE *SELF-CONTAINED*

JWT access tokens are also known as self-contained access tokens. They can be independently verified by APIs using the public key of the STS.


4
Request new access token with refresh token
1
Run an OAuth 2.0 flow
CLIENT
SECURITY TOKEN SERVICE
2
JWT access token and refresh token
5
New access token
Request with access token
3
Request with access token
6
The access token has a short lifetime (e.g., 10 minutes), and the refresh token has a long lifetime (e.g., 12 hours)
The STS is in full control over the refresh token, enabling token revocation if desired
API
Only the refresh token can be revoked. Self-contained (JWT) access tokens typically remain valid for their entire lifetime


6
Request new access token
with refresh token
1
Run an OAuth 2.0 flow
CLIENT
SECURITY
TOKEN
SERVICE
2
Reference access token
and refresh token
7
New access token
The STS is in full control over access tokens and refresh tokens, enabling token revocation if desired
Request with
access token
3
Token introspection
for reference token
4
Request with
access token
8
Token introspection
for reference token
9
5
Introspection response with the
claims associated with the token
10
Introspection response with the
claims associated with the token
API
Reference tokens cannot be interpreted without token introspection, causing significant overhead

Reference tokens sound awesome, let's GOOOOOO!

How fast can you revoke an access token?

PRACTICAL GUIDELINES ON ACCESS TOKEN TYPES

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

ACCESS TOKEN TYPES

- The STS decides on the security properties of access tokens
  - Clients only send access tokens, so they are agnostic of the token type and its properties
  - The API will need to understand how to process different token types

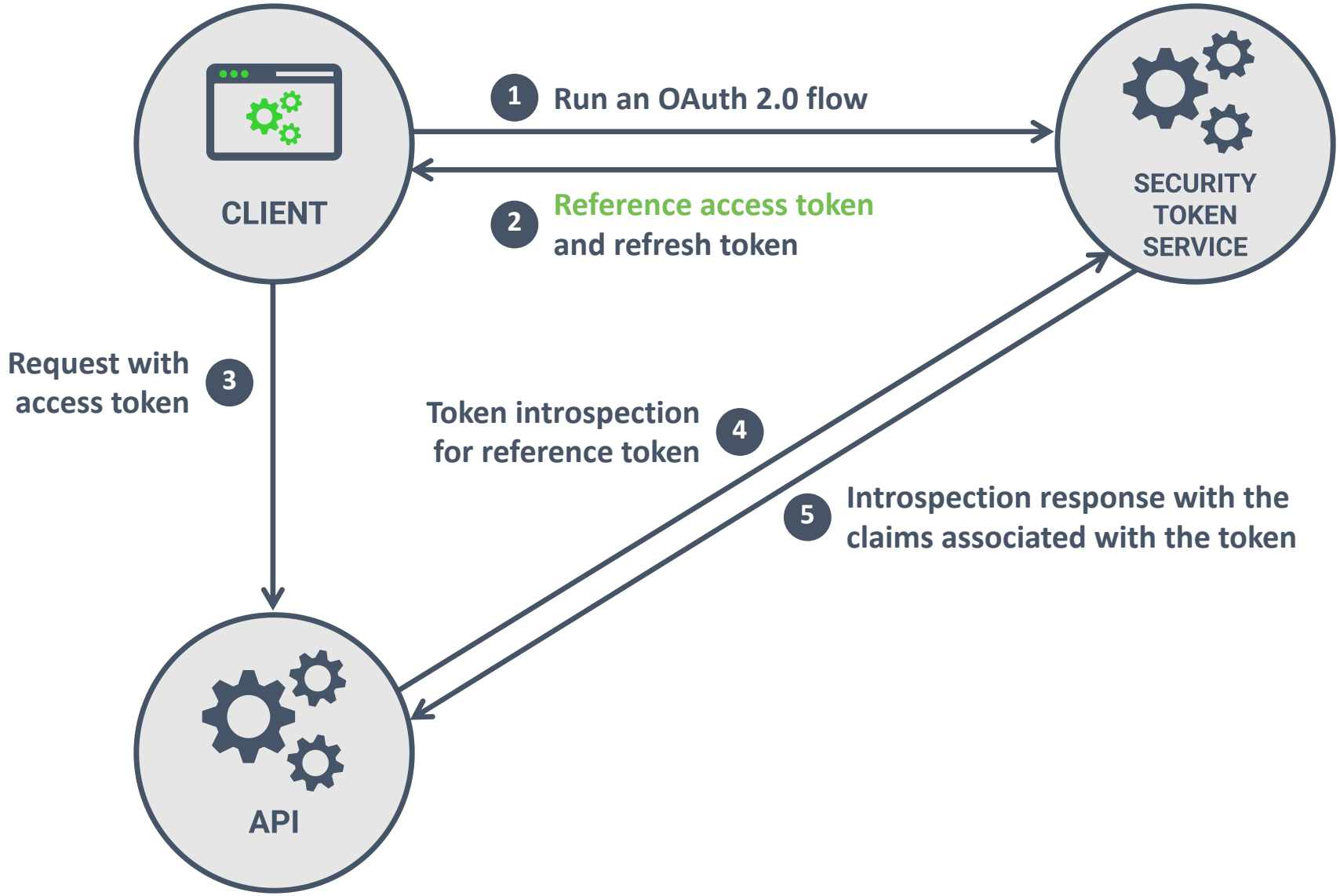
- In practice, self-contained JWT tokens are common for distributed scenarios
  - Running token introspection between different parties is often difficult
  - Keep token lifetimes as short as possible

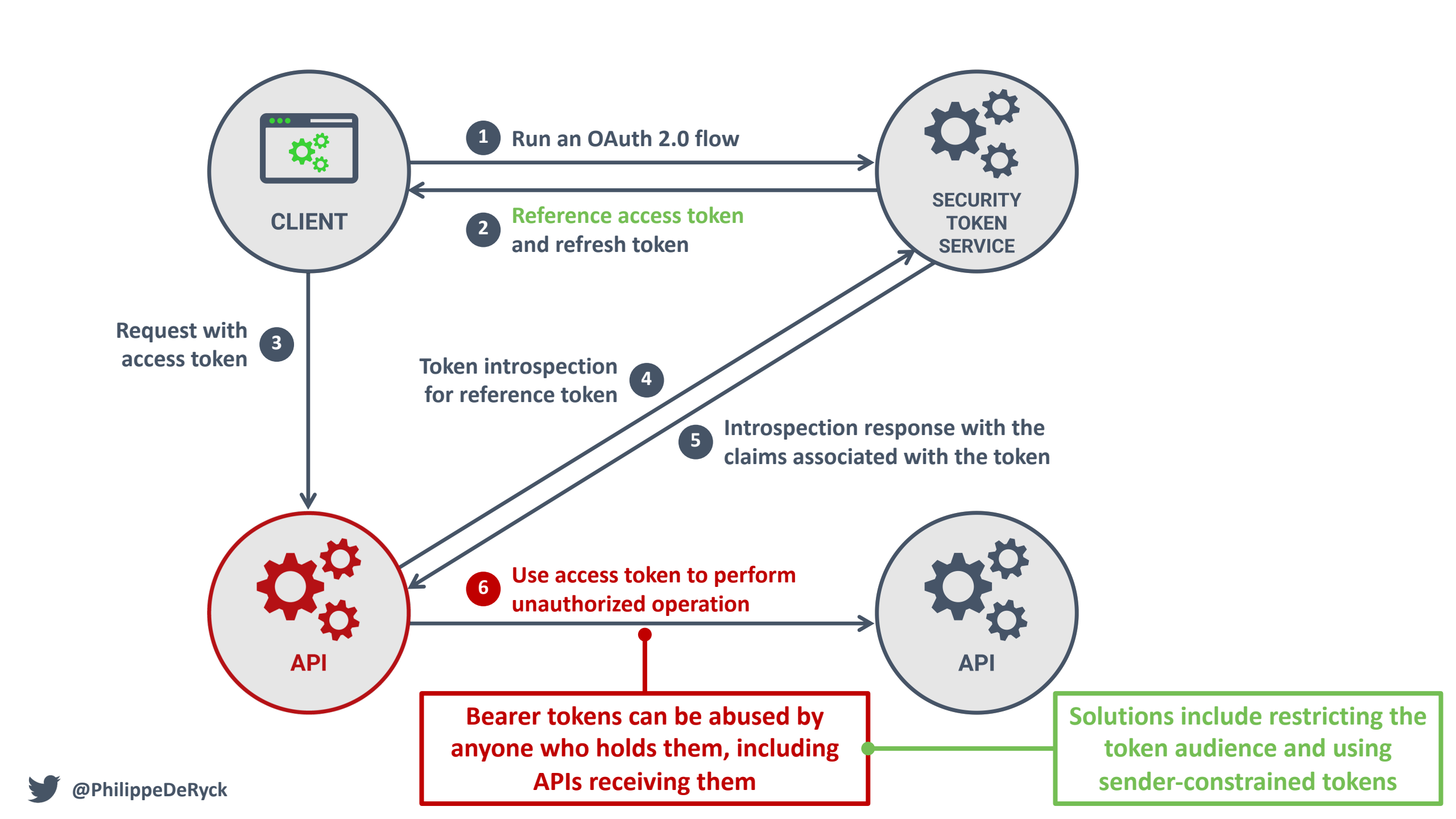
- Reference tokens are often used for internal systems
  - *On-premise* token introspection is easier to implement
  - Can also be implemented with an API gateway that translates tokens

Look at the full picture of a token lifecycle

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

Advanced token security


CLIENT
1
Run an OAuth 2.0 flow
2
Reference access token
and refresh token
SECURITY
TOKEN
SERVICE
Request with
access token
3
Token introspection
for reference token
4
5
Introspection response with the
claims associated with the token
API


CLIENT
1
Run an OAuth 2.0 flow
2
Reference access token
and refresh token
SECURITY
TOKEN
SERVICE
Request with
access token
3
Token introspection
for reference token
4
5
Introspection response with the
claims associated with the token
API
6
Use access token to perform
unauthorized operation
API
Bearer tokens can be abused by
anyone who holds them, including
APIs receiving them
Solutions include restricting the
token audience and using
sender-constrained tokens

An OAuth 2.0 initialization URI

```
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)

Requesting access tokens with a specific resource

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

 grant_type=authorization_code
&client_id=lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv
&code=SplxlOBeZQQYbYS6WxSbIA
&resource=https://api.restograde.com/reviews
& […  redirect_uri / code_verifier …]
```

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

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

Requesting access tokens with a specific resource

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

 grant_type=authorization_code
&client_id=lY5g0BKB7Mow4yDlb6rdGPsO2i1g7Osv
&code=SplxlOBeZQQYbYS6WxSbIA
&resource=https://api.restograde.com/reviews
& [… redirect_uri / code_verifier …]
```

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

USING RESOURCE INDICATORS

- A client can use the ***resource*** parameter to indicate the target audience
  - The client requests a set of resources when initializing the *Authorization Code* flow
  - The exchange of an authorization code/refresh token is done with a specific resource

- Resource indicators are URIs which are defined by the STS
  - It is recommended to use the full URL of an API to identify a resource
    - E.g., ***https://api.restograde.com***, ***https://api.restograde.com/reviews***
  - When not possible, the use of URNs allows for a more flexible naming scheme
    - E.g., ***urn:restograde:reviews***

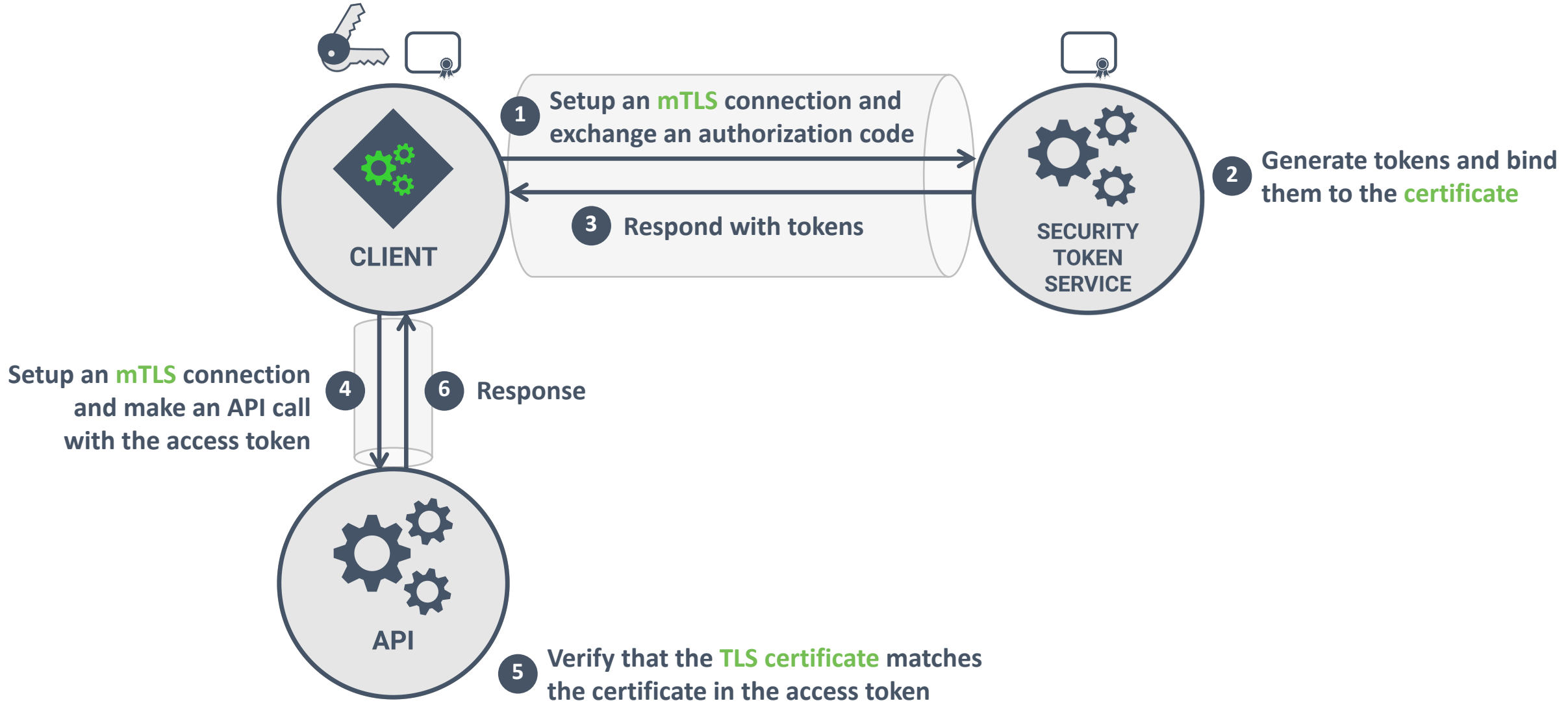
- The resulting access token will be tailored towards the requested resource
  - The audience will contain the resource indicator (***aud*** claim in a JWT)
  - The scopes will typically be limited to relevant scopes for the audience (downscoping)

Resource indicators support limiting audiences

This new OAuth 2.0 specification (RFC 8707) allows clients to request permission to access multiple APIs, but only request access tokens for a single API

Sender-constrained tokens can be used to link access/refresh tokens to a specific client

PROOF-OF-POSSESSION THROUGH TLS CERTIFICATES



Public key Private key Certificate

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

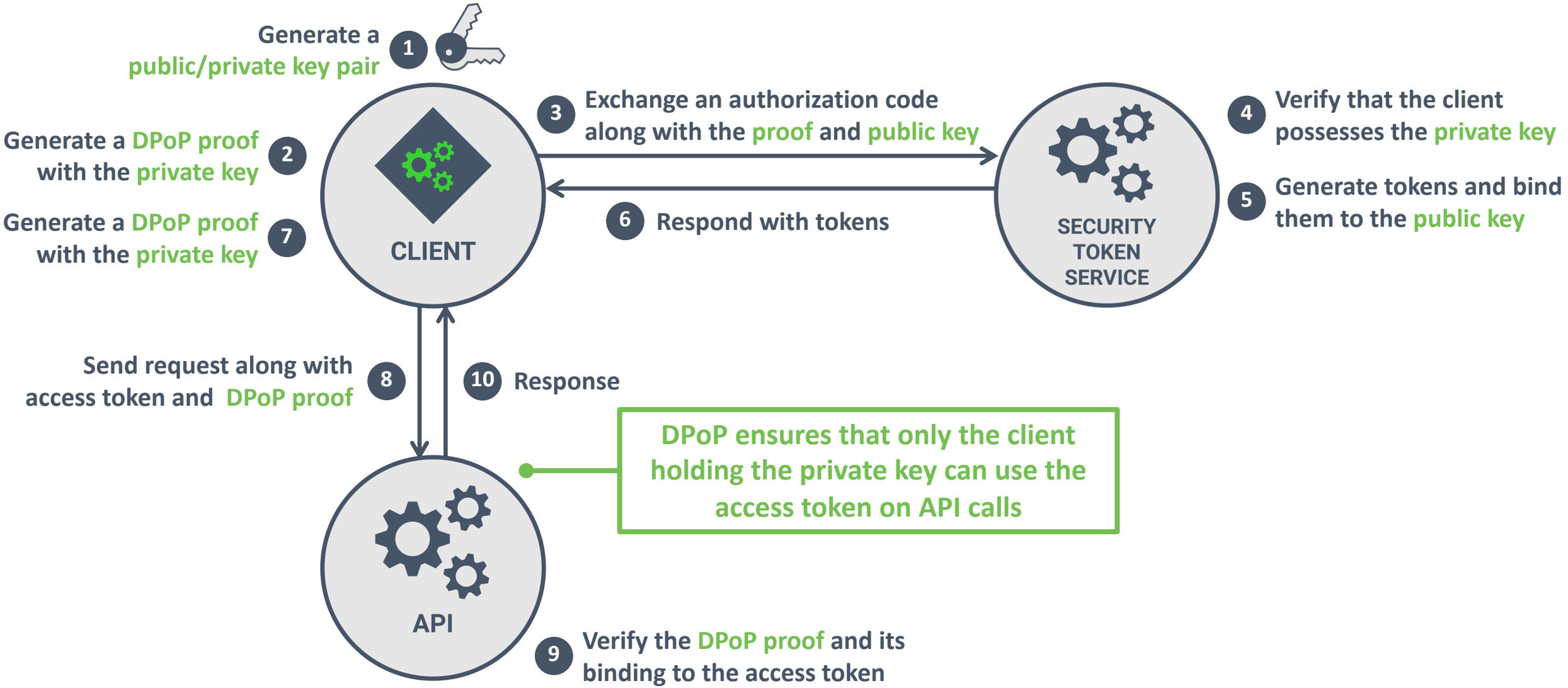
SENDER-CONSTRAINED TOKENS WITH MTLS

- The ***cnf*** claim contains information about the proof-of-possession key
  - JWT access tokens directly embed the ***cnf*** claim in the token
  - For reference access tokens, the STS provides the ***cnf*** claim during introspection

- The only responsibility for a client is using mTLS with a client certificate
  - An STS that supports sender constrained access tokens will use the certificate fingerprint
    - The hash in the ***x5t#S256*** value uniquely identifies the certificate and its public key
  - An API enforcing proof-of-possession will look for the ***cnf*** claim can verify the fingerprint
    - If the connection is setup with the right certificate, the client must possess the private key

- Sender constrained access tokens are much harder to abuse
  - An attacker would need to completely compromise a client to abuse access tokens

The concept of DPoP



Public key

Private key

Demonstration of Proof-of-Possession (DPoP)

- DPoP is an application-layer PoP mechanism relying on JWTs
  - The client generates a private/public key pair and proves possession of the private key
  - The STS links access tokens / refresh tokens to the public key of the client
  - The client provides a signed JWT along with any request carrying a DPoP token

- Possession of the private key is done through a DPoP proof JWT
  - The JWT has type ***dpop+jwt*** and contains metadata about the request being sent
  - The metadata includes the HTTP method and HTTP endpoint that the request is going to
  - Further details (request headers, request body, …) are not included in the DPoP proof

- With DPoP, access tokens and refresh tokens become sender-constrained
  - They no longer act as bearer tokens, making it significantly harder to abuse stolen tokens

USE SENDER-CONSTRAINED TOKENS

The use of sender-constrained tokens is considered a best practice for sensitive OAuth 2.0 clients (both backend and native)

PASETO AS AN ALTERNATIVE TO JWT

Platform-Agnostic Security Tokens (PASETO)

- PASETO is explicitly developed to counter vulnerabilities in the JWT spec
  - The use of the ***none*** algorithm
  - The potential confusion between HMACs and digital signatures
  - The pitfalls with handling encrypted JWTs
  - The wide range of supported signing and encryption algorithms

- The goal of PASETO is to offer a secure-by-design standard to represent tokens
  - Versioned tokens instead of *algorithm agility*
  - Fixed algorithm selection for each version to avoid confusion
  - Specification of the purpose of a token (local use or distributed use)

A local PASETO token (corresponds to an HMAC-signed JWT)

```
v2.local.sIgVm0es9uswZliPdyXOOi99czPbpl41KOUu45e62BvCaL5H3kHNibrbRZkM1-wW091ARzNexLY8g0GZA0-
WCNsgs8GZLClEk5TJbgQjf__yExZRh2qMnqxfVr_KS9WoqKVlU-
WrAG6TRUXZo43OSJQkeNBnB8Gq4rN2A8HYeA3ms20up80dgz2rpY79F9ILvPrAIzxNkDSE51vAxv50BTShuel3F3hXgReHsDv2PJCn
MBnMyE_AfePxJ6WJ1obXSIUpSsOQX6wjwdQdOIcXZ853c-NPYMVU-abXJhhLVvvHyNZPi1wcEvjt.eyJraWQiOiAiMTIzNDUifQ
```

A public PASETO token (corresponds to a JWT signed with a private key)

```
v2.public.eyJpZCI6ICI0MTBkZjI5Ni04OWQ1LTQzODAtODQyMy02ZjJkNzMwNDA3NDQiLCAibmFtZSI6ICJSYW5kYWxsIERlZ2dl
cyIsICJleHAiOiAiMjAxOS0xMC0xMFQxMTowMzoyNC0wNzowMCJ9xe6hZBYn8IZoJmgL9k1VjTcl7Dz4T-
lo2FvIxeFXQNtNY3QAyCaa5XW-29n-9nV-beU6z7P-YF97lPFvnPfnDA.eyJraWQiOiAiMTIzNDUifQ
```

A local PASETO token (corresponds to an HMAC-signed JWT)

```
v2.local.sIgVm0es9uswZliPdyXOOi99czPbpl41KOUu45e62BvCaL5H3kHNibrbRZkM1-wW091ARzNexLY8g0GZA0-
WCNsgs8GZLClEk5TJbgQjf__yExZRh2qMnqxfVr_KS9WoqKVlU-
WrAG6TRUXZo43OSJQkeNBnB8Gq4rN2A8HYeA3ms20up80dgz2rpY79F9ILvPrAIzxNkDSE51vAxv50BTShuel3F3hXgReHsDv2PJCn
MBnMyE_AfePxJ6WJ1obXSIUpSsOQX6wjwdQdOIcXZ853c-NPYMVU-abXJhhLVvvHyNZPi1wcEvjt.eyJraWQiOiAiMTIzNDUifQ
```

The version indicates how the token is structured and which algorithm is used (currently v2)

A public PASETO token (corresponds to a JWT signed with a private key)

```
v2.public.eyJpZCI6ICI0MTBkZjI5Ni04OWQ1LTQzODAtODQyMy02ZjJkNzMwNDA3NDQiLCAibmFtZSI6ICJSYW5kYWxsIERlZ2dl
cyIsICJleHAiOiAiMjAxOS0xMC0xMFQxMTowMzoyNC0wNzowMCJ9xe6hZBYn8IZoJmgL9k1VjTcl7Dz4T-
lo2FvIxeFXQNtNY3QAyCaa5XW-29n-9nV-beU6z7P-YF97lPFvnPfnDA.eyJraWQiOiAiMTIzNDUifQ
```

A local PASETO token (corresponds to an HMAC-signed JWT)

```
v2.local.sIgVm0es9uswZliPdyXOOi99czPbpl41KOUu45e62BvCaL5H3kHNibrbRZkM1-wW091ARzNexLY8g0GZA0-
WCNsgs8GZLClEk5TJbgQjf__yExZRh2qMnqxfVr_KS9WoqKVlU-
WrAG6TRUXZo43OSJQkeNBnB8Gq4rN2A8HYeA3ms20up80dgz2rpY79F9ILvPrAIzxNkDSE51vAxv50BTShuel3F3hXgReHsDv2PJCn
MBnMyE_AfePxJ6WJ1obXSIUpSsOQX6wjwdQdOIcXZ853c-NPYMVU-abXJhhLVvvHyNZPi1wcEvjt.eyJraWQiOiAiMTIzNDUifQ
```

The purpose indicates how the token is secured (HMAC vs digital signature) and makes explicit how the token should be used

A public PASETO token (corresponds to a JWT signed with a private key)

```
v2.public.eyJpZCI6ICI0MTBkZjI5Ni04OWQ1LTQzODAtODQyMy02ZjJkNzMwNDA3NDQiLCAibmFtZSI6ICJSYW5kYWxsIERlZ2dl
cyIsICJleHAiOiAiMjAxOS0xMC0xMFQxMTowMzoyNC0wNzowMCJ9xe6hZBYn8IZoJmgL9k1VjTcl7Dz4T-
lo2FvIxeFXQNtNY3QAyCaa5XW-29n-9nV-beU6z7P-YF97lPFvnPfnDA.eyJraWQiOiAiMTIzNDUifQ
```

A local PASETO token (corresponds to an HMAC-signed JWT)

```
v2.local.sIgVm0es9uswZliPdyXOOi99czPbpl41KOUu45e62BvCaL5H3kHNibrbRZkM1-wW091ARzNexLY8g0GZA0-
WCNsgs8GZLClEk5TJbgQjf__yExZRh2qMnqxfVr_KS9WoqKVlU-
WrAG6TRUXZo43OSJQkeNBnB8Gq4rN2A8HYeA3ms20up80dgz2rpY79F9ILvPrAIzxNkDSE51vAxv50BTShuel3F3hXgReHsDv2PJCn
MBnMyE_AfePxJ6WJ1obXSIUpSsOQX6wjwdQdOIcXZ853c-NPYMVU-abXJhhLVvvHyNZPi1wcEvjt.eyJraWQiOiAiMTIzNDUifQ
```

V2 local tokens provide data integrity and data confidentiality for the payload (using authenticated encryption)

A public PASETO token (corresponds to a JWT signed with a private key)

```
v2.public.eyJpZCI6ICI0MTBkZjI5Ni04OWQ1LTQzODAtODQyMy02ZjJkNzMwNDA3NDQiLCAibmFtZSI6ICJSYW5kYWxsIERlZ2dl
cyIsICJleHAiOiAiMjAxOS0xMC0xMFQxMTowMzoyNC0wNzowMCJ9xe6hZBYn8IZoJmgL9k1VjTcl7Dz4T-
lo2FvIxeFXQNtNY3QAyCaa5XW-29n-9nV-beU6z7P-YF97lPFvnPfnDA.eyJraWQiOiAiMTIzNDUifQ
```

V2 public tokens provide data integrity using digital signatures

PASETO PROS AND CONS

Robust and unambiguous algorithms

Purpose of tokens is easier to understand

No specific payload format (e.g., JSON)

No official specification, just an expired draft

No guidance on explicit typing for tokens

No support for encrypting public tokens

Library devs are still responsible for security

PASETO ADDRESSES JWT INSECURITIES

PASETO is simpler and less ambiguous than the JWT specifications, but the lack of use/support makes it less suited than JWTs

Fixing JWTs in your architecture

1. Avoid HMACs and hardcode one digital signature algorithm
2. Use explicit typing to indicate the purpose of a token
3. Write a wrapper library to encapsulate the dirty details

Internet Engineering Task Force (IETF)
Request for Comments: 8725
BCP: 225
Updates: 7519
Category: Best Current Practice
ISSN: 2070-1721

Y. Sheffer
Intuit
D. Hardt

M. Jones
Microsoft

February 2020

JSON Web Token Best Current Practices

Abstract

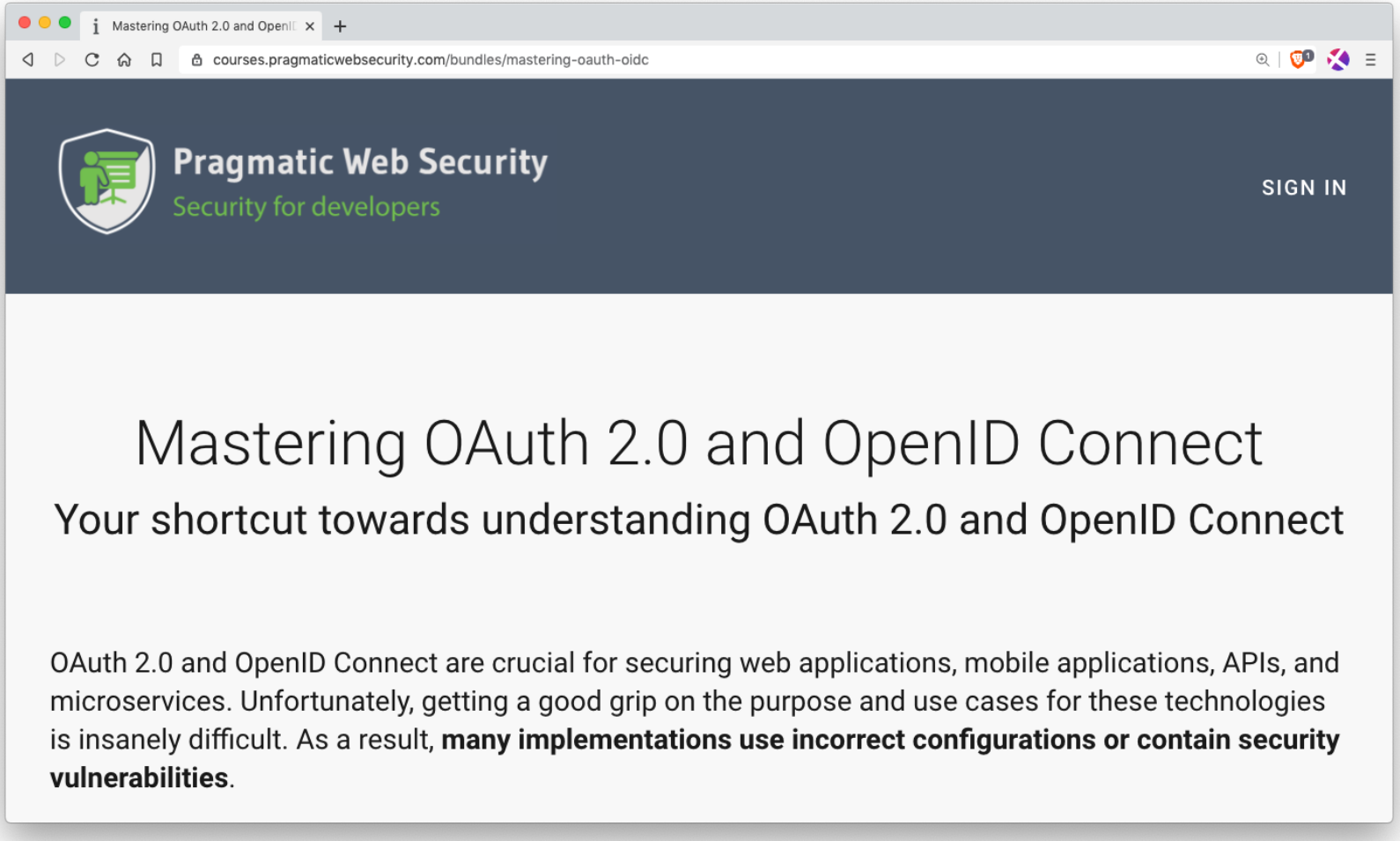
JSON Web Tokens, also known as JWTs, are URL-safe JSON-based security tokens that contain a set of claims that can be signed and/or encrypted. JWTs are being widely used and deployed as a simple security token format in numerous protocols and applications, both in the area of digital identity and in other application areas. This Best Current Practices document updates RFC 7519 to provide actionable guidance leading to secure implementation and deployment of JWTs.

Conclusion

Key takeaways

1. Follow current best practices for handling JWTs (or use PASETO)
2. Remember that JWTs/PASETOs only represent claims, nothing else
3. Carefully analyze token security requirements in your architecture

This online course helps you understand the details of OAuth 2.0 and OpenID Connect



Thank you for watching!

Connect on social media for more in-depth security content

@PhilippeDeRyck

/in/PhilippeDeRyck