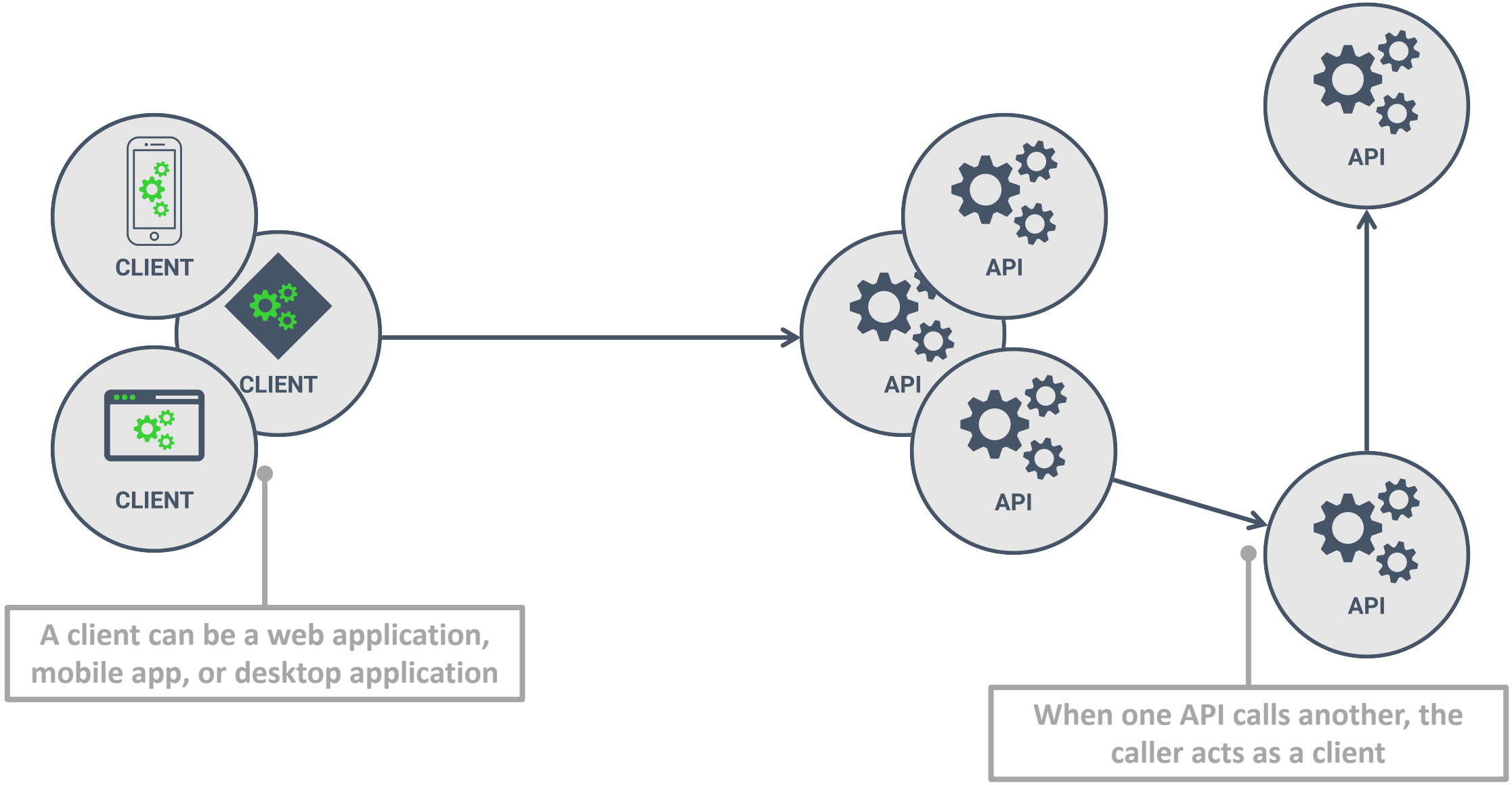
Serving the right recipe for API authentication

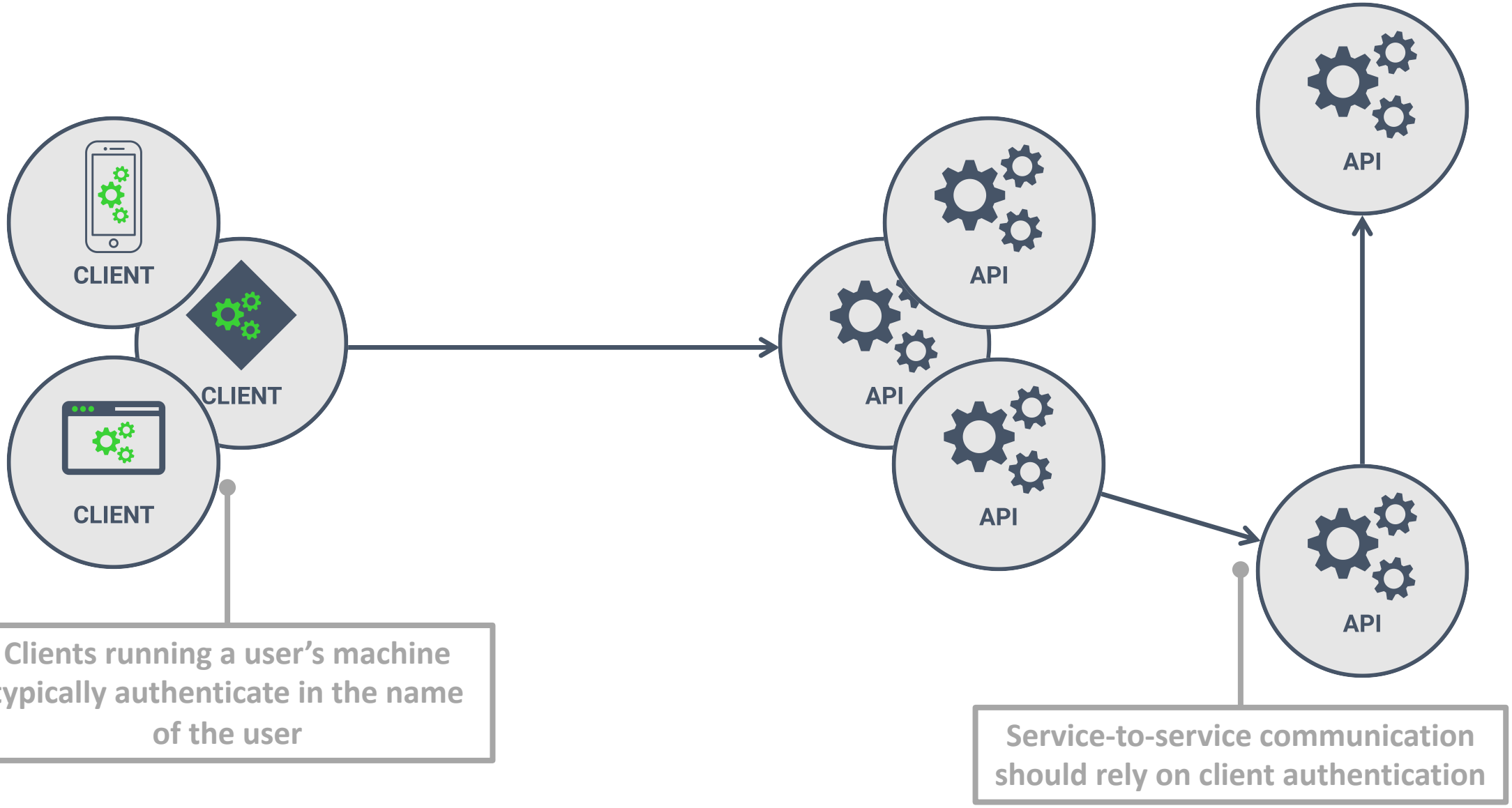
Dr. Philippe De Ryck

https://Pragmatic Web Security.com

Authentication is providing proof that
a party is who they claim to be

Authorization relies on authentication
to decide if an operation is allowed or not


CLIENT
CLIENT
CLIENT
API
API
API
API
API
A client can be a web application, mobile app, or desktop application
When one API calls another, the caller acts as a client


CLIENT
CLIENT
CLIENT
API
API
API
API
API
Clients running a user's machine typically authenticate in the name of the user
Service-to-service communication should rely on client authentication

r/shittyfoodporn · Crossposted by u/supergamer1313 1 hour ago

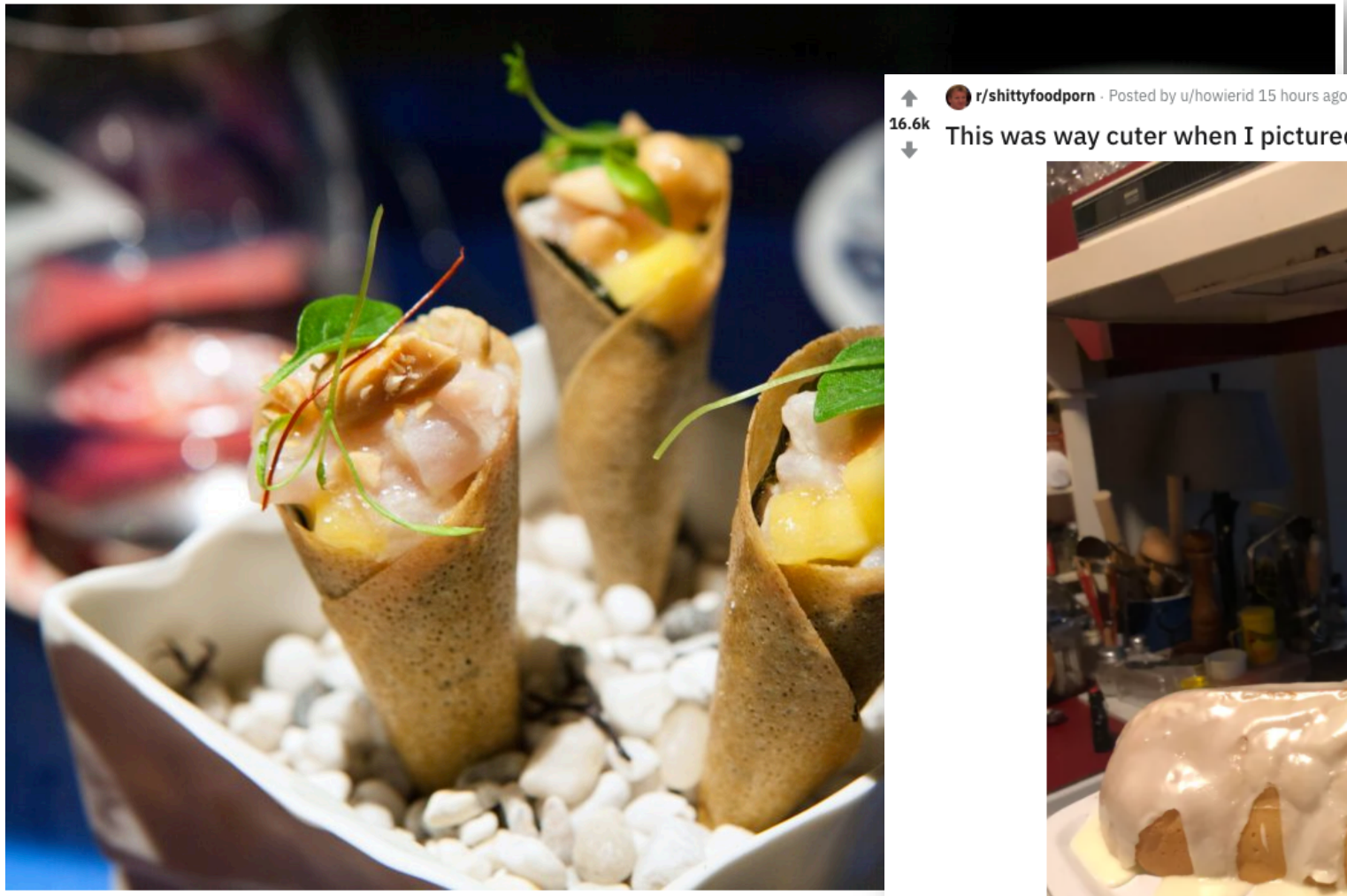
$1 NYC Pizza Slice

r/FoodPorn · Posted by u/BravoVogue 7 hours ago

$1 NYC Pizza Slice

6.7k points · 325 comments

3 Comments Give Award Share S



Tickets (Barcelona, Spain) – Fun, Innovative T
Hype

16.6k

r/shittyfoodporn · Posted by u/howierid 15 hours ago 2 3 2 5 3 3

This was way cuter when I pictured it in my head

420 Comments Give Award Share Save Hide Report 96% Upvoted

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 / Expert

Secure Application Development

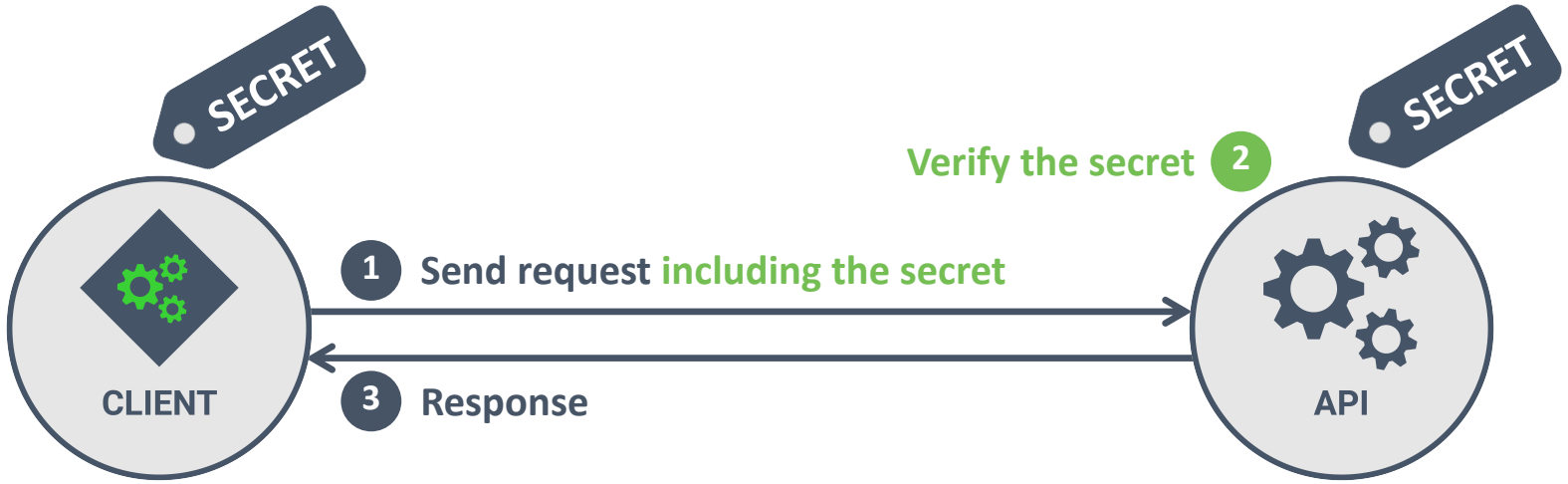
SecAppDev organizer

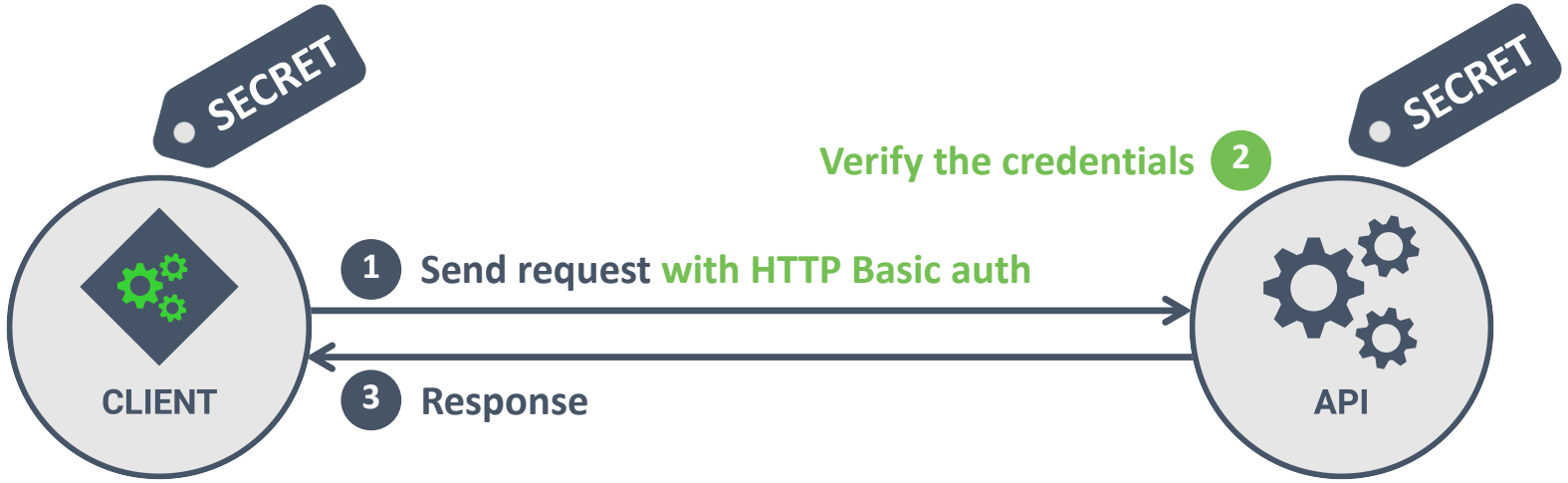
I help developers with security

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

https://pragmaticwebsecurity.com

Basic client authentication


SECRET
SECRET
Verify the secret 2
1 Send request including the secret
3 Response
CLIENT
API

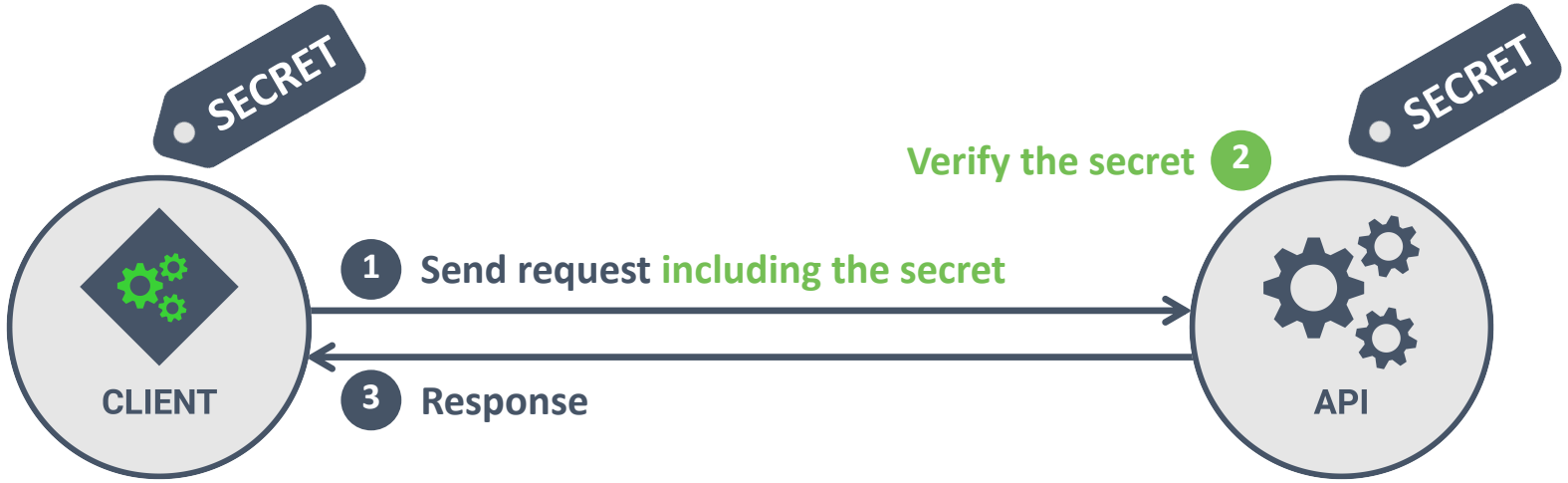


1 *The client includes their credentials on every request*

```
1  GET /restaurants HTTP/1.1
2  Host: restograde.com
3  Authorization: Basic bXlDbGllbnQ6dGhlQ2xpZW50U2VjcmV0
```

The client credentials are included in the HTTP Authorization header

The value consists of "username:password" in base64



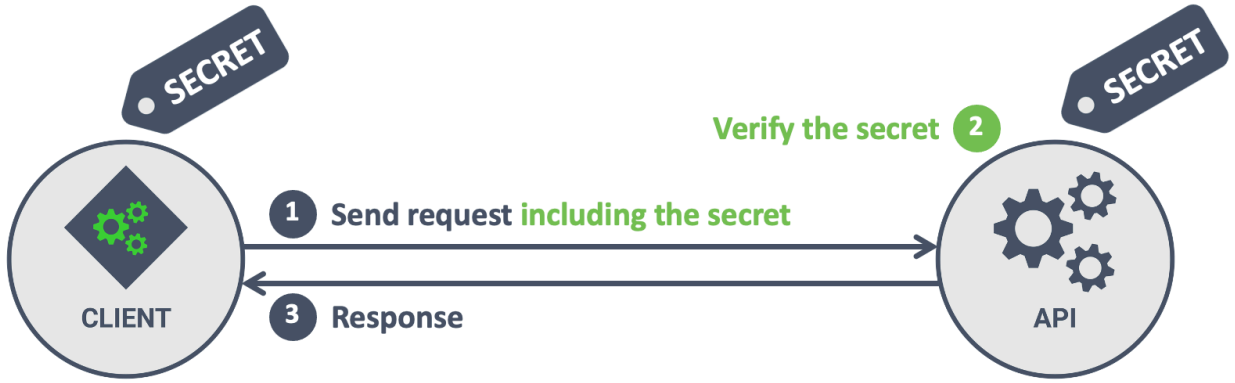
1 *The secret is an API key included in every request*

```
GET /restaurants HTTP/1.1
Host: restograde.com
X-API-Key: fd2bcd6eab56417f81332c109e0d67eb
```

The API key is included in a custom request header

SENDING A SHARED SECRET

- *Basic authentication / API keys*
- *Secret added by sender, verified by API*
- *Secret is often hardcoded*
- *Works well between services*



BENEFITS

Lightweight mechanism with minimal overhead

Easy to implement

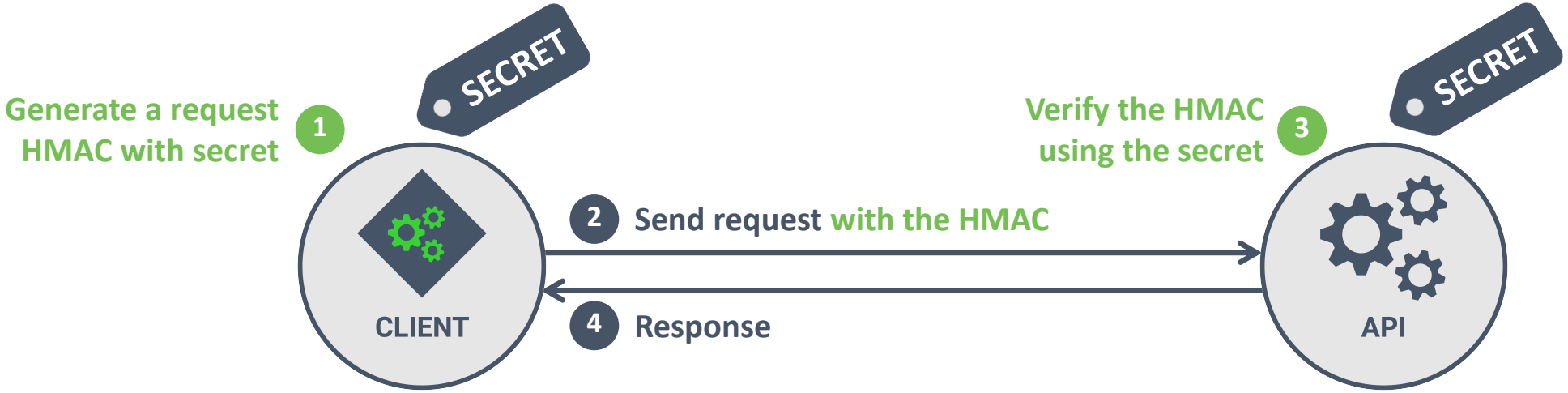
Works well within a single "*trust zone*"

DRAWBACKS

Secret has to be known by all involved parties

Scalable secret management is challenging

Secret is not linked to the request in any way



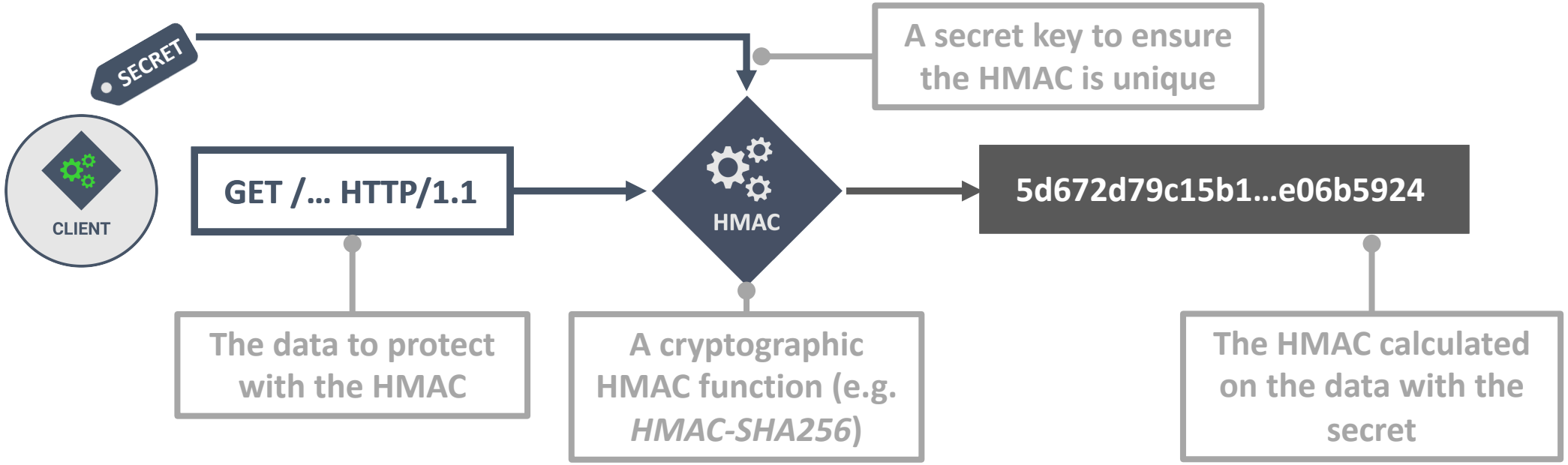
2 *The client authenticates the request with an HMAC*

```
GET /restaurants HTTP/1.1
Host: restograde.com
X-Req-Sig: 5d672d79c15b13162d927…e06b5924a6f2b5d7
```

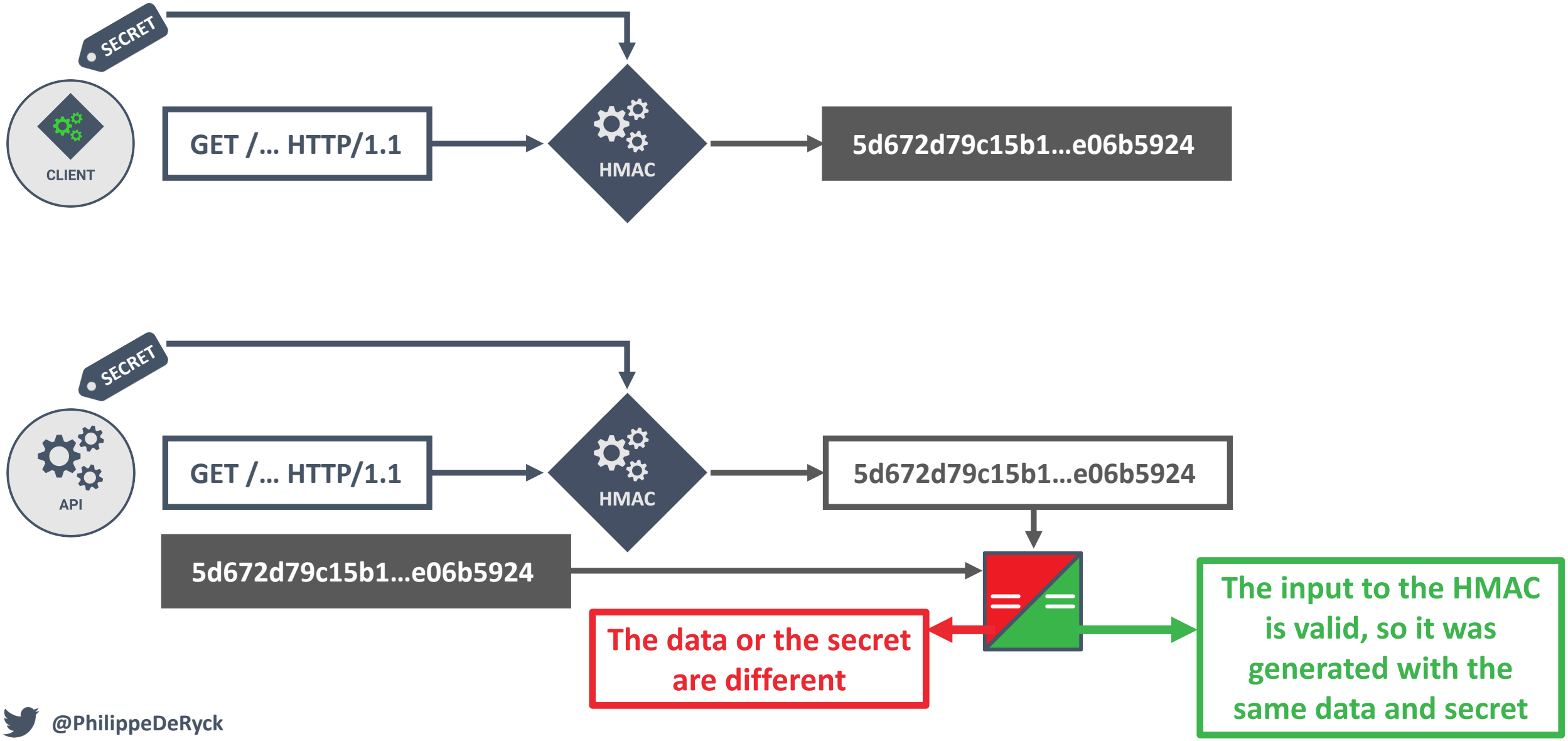
The HMAC is included in a custom request header

The HMAC is based on the request and the secret value

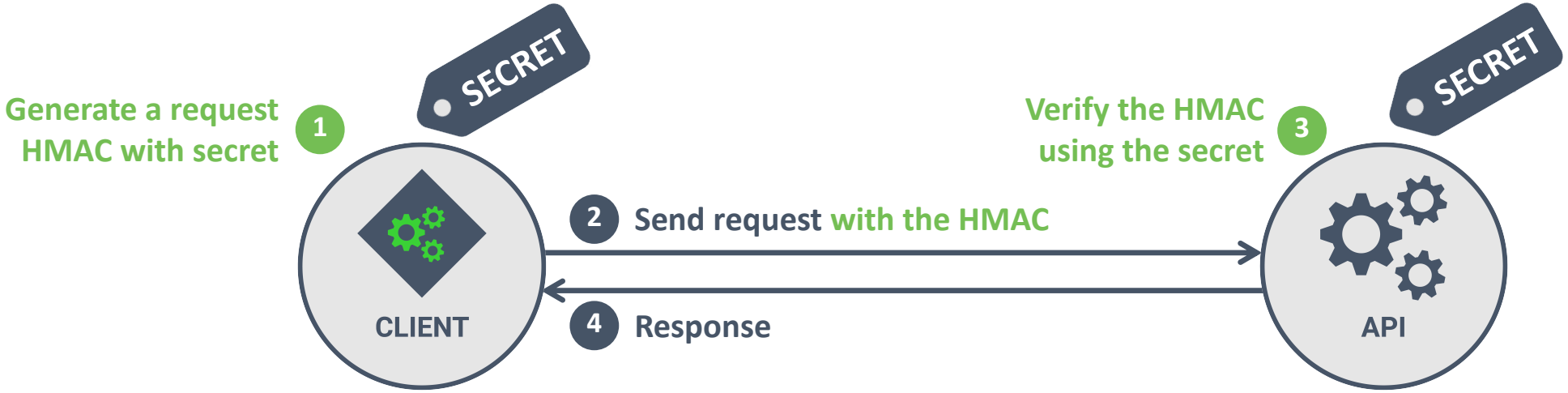
INTERMEZZO: HMACS



INTERMEZZO: HMACS



@PhilippeDeRyck



2 *Signing AWS requests with Signature Version 4*

```
GET /?Action=ListUsers&Version=2010-05-08 HTTP/1.1
Host: iam.amazonaws.com
X-amz-date: 20150830T123600Z
Authorization: AWS4-HMAC-SHA256 Credential=AKIDEXAMPLE/20150830/us-east-1/iam/aws4_request,
SignedHeaders=content-type;host;x-amz-date,
Signature=5d672d79c15b13162d9279b0855cfba6789a8edb4c82c400e06b5924a6f2b5d7
```

The Authorization header also includes the metdata about the HMAC

2 *The client authenticates the request with a JWT*

```
GET /restaurants HTTP/1.1
Host: restograde.com
X-Req-JWT: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey…
```

The JWT is included in a custom request header

INTERMEZZO: JWTS

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IlBoaWxpcHBlIERlIFJ5Y2siLCJyb2xlcyI6InVzZXIgcmVzdGF1cmFudG93bmVyIiwiaWF0IjoxNTE2MjM5MDIyfQ.KPjhyE9oi83uehgw6Lm_0yAZzRuJhcUqXETD2AIrF2A

Encoded PASTE A TOKEN HERE


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
zdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IlBoaWx
pcHBlIERlIFJ5Y2siLCJyb2xlcyI6InVzZXIgcmV
zdGF1cmFudG93bmVyIiwiaWF0IjoxNTE2MjM5MDI
yfQ.KPjhyE9oi83uehgw6Lm_0yAZzRuJhcUqXETD
2AIrF2A
```


Base64-encoded

Contains a set of claims

Integrity-protected with a signature

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE


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


PAYLOAD: DATA

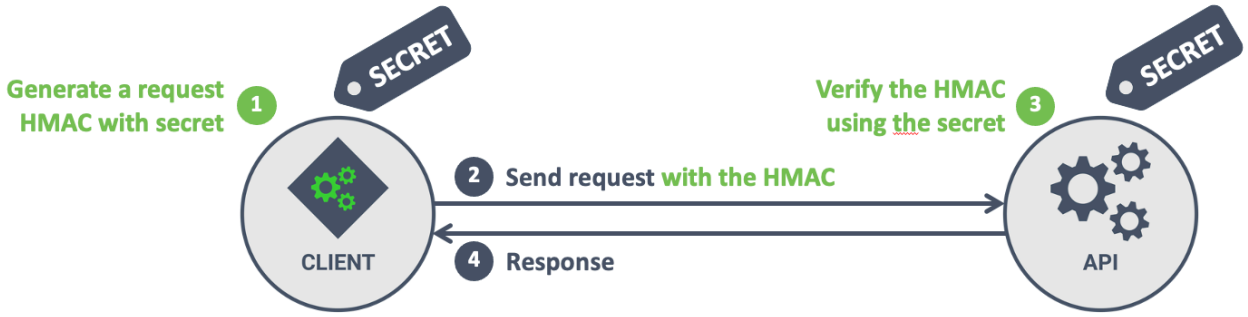

```
{
  "sub": "1234567890",
  "name": "Philippe De Ryck",
  "roles": "user restaurantowner",
  "iat": 1516239022
}
```


VERIFY SIGNATURE

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

ADDING AN HMAC IN THE REQUEST

- *HTTP Signatures / Custom JWTs*
- *HMACs are calculated on a piece of data using a shared secret*
- *HMACs ensure the integrity of the data*



BENEFITS

HMACs provide data authenticity and integrity

Relatively easy to implement

Signature can be uniquely tied to a specific request

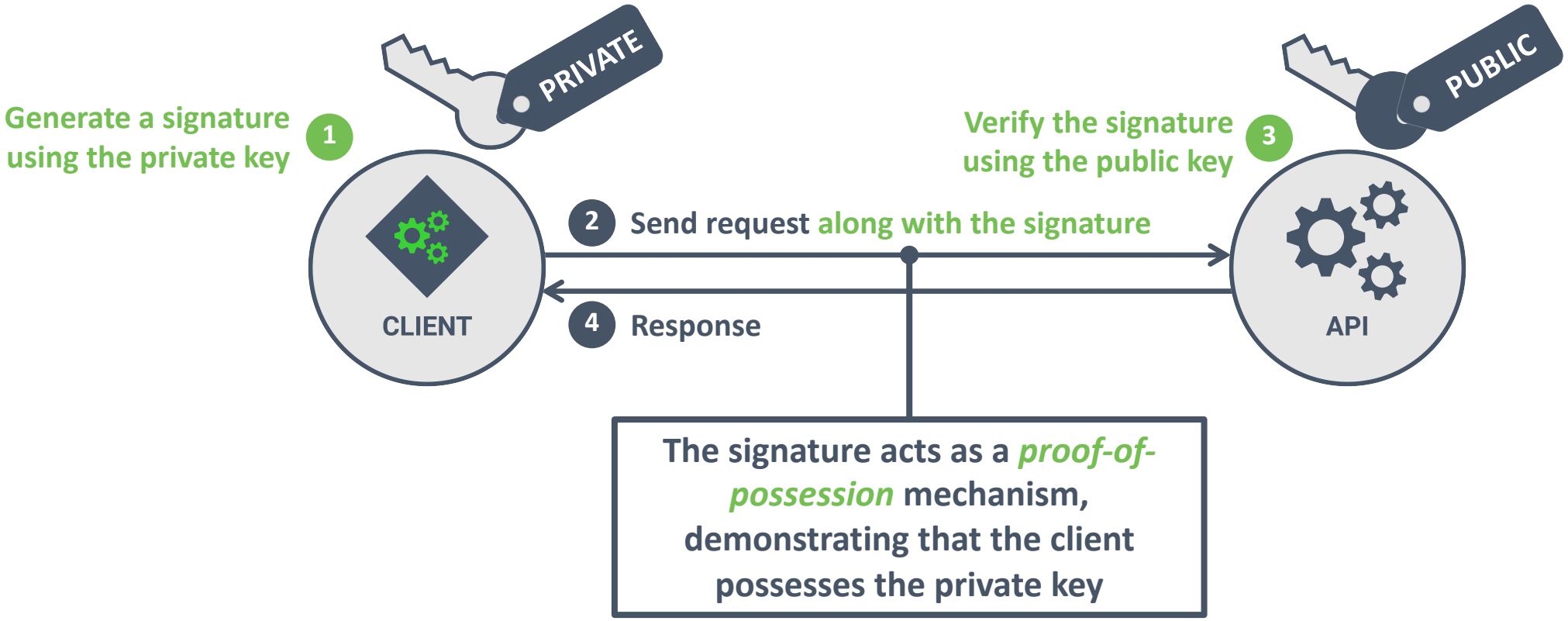
DRAWBACKS

Secret has to be known by all involved parties

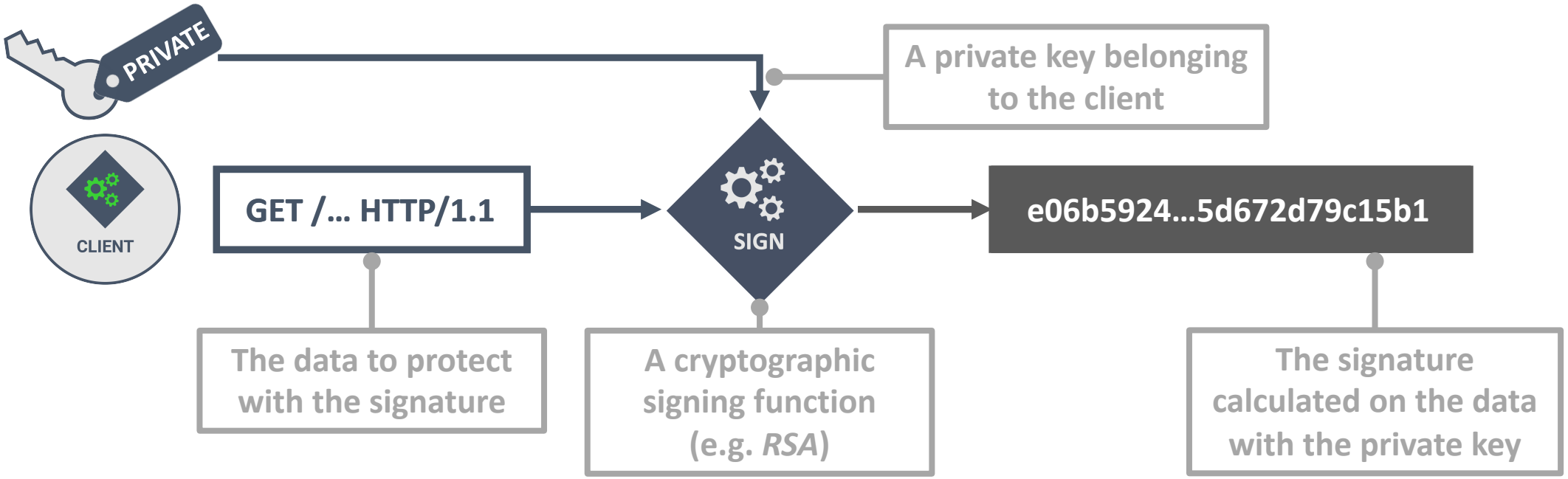
Scalable secret management is challenging

HMAC verification requires (application) code

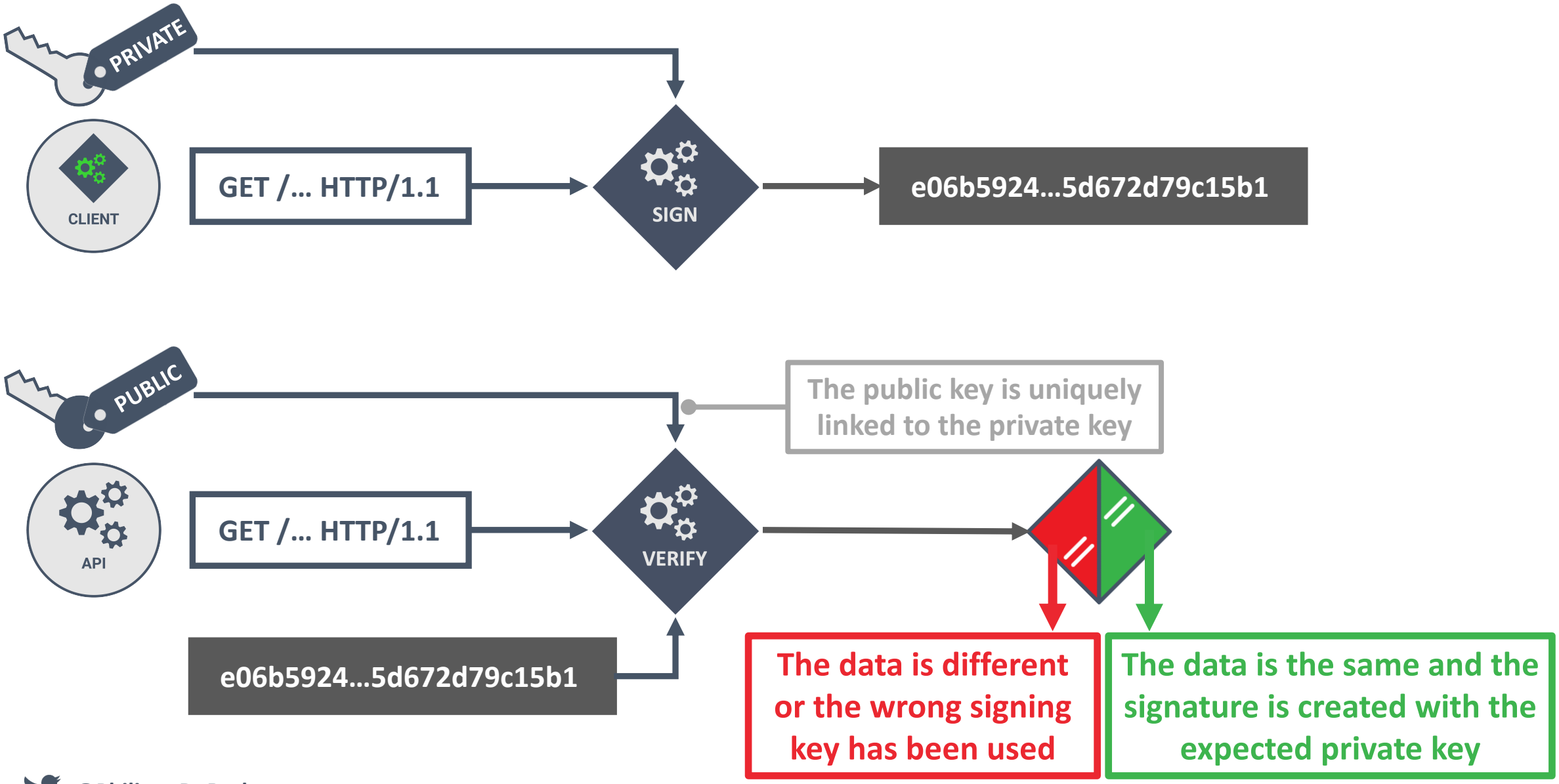
Advanced client authentication


PRIVATE
PUBLIC
Generate a signature using the private key
1
Verify the signature using the public key
3
2
Send request along with the signature
CLIENT
4
Response
API
The signature acts as a *proof-of-possession* mechanism, demonstrating that the client possesses the private key

INTERMEZZO: DIGITAL SIGNATURES



INTERMEZZO: DIGITAL SIGNATURES



2 *Signing requests with the HTTP Signature specification*

```
GET /restaurants HTTP/1.1
Host: restograde.com
Date: Thu, 29 Oct 2020 07:28:00 GMT
Signature: keyId="clientPubKey", algorithm="rsa-sha256", created=1402170695, expires=1402170995,
headers="host date", signature="T1l3tWH2cSP31nfuvc3nVaHQ6IAu9YLEXgTXnlWbgKtBTa…gd9rGnCHtAg=="
```

The Signature header also includes the metdata about the used key and the signature contents

2 *The client authenticates the request with a JWT*

```
GET /restaurants HTTP/1.1
Host: restograde.com
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey…
```

The Authorization header includes a client-generated JWT

The client can include arbitrary request metadata in the JWT

Internet Engineering Task Force (IETF)
Request for Comments: 7523
Category: Standards Track
ISSN: 2070-1721

M. Jones
Microsoft
B. Campbell
Ping Identity
C. Mortimore
Salesforce
May 2015

JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants

Abstract

This specification defines the use of a JSON Web Token (JWT) Bearer Token as a means for requesting an OAuth 2.0 access token as well as for client authentication.

ASYMMETRIC REQUEST SIGNATURES

- *HTTP Signatures / Custom JWTs*
- *Created with the sender's private key*
- *Verified with the sender's public key*
- *Signatures ensure the validity of the data*

BENEFITS

Only the public key needs to be shared (no secrets)

Works well when one client relies on multiple APIs

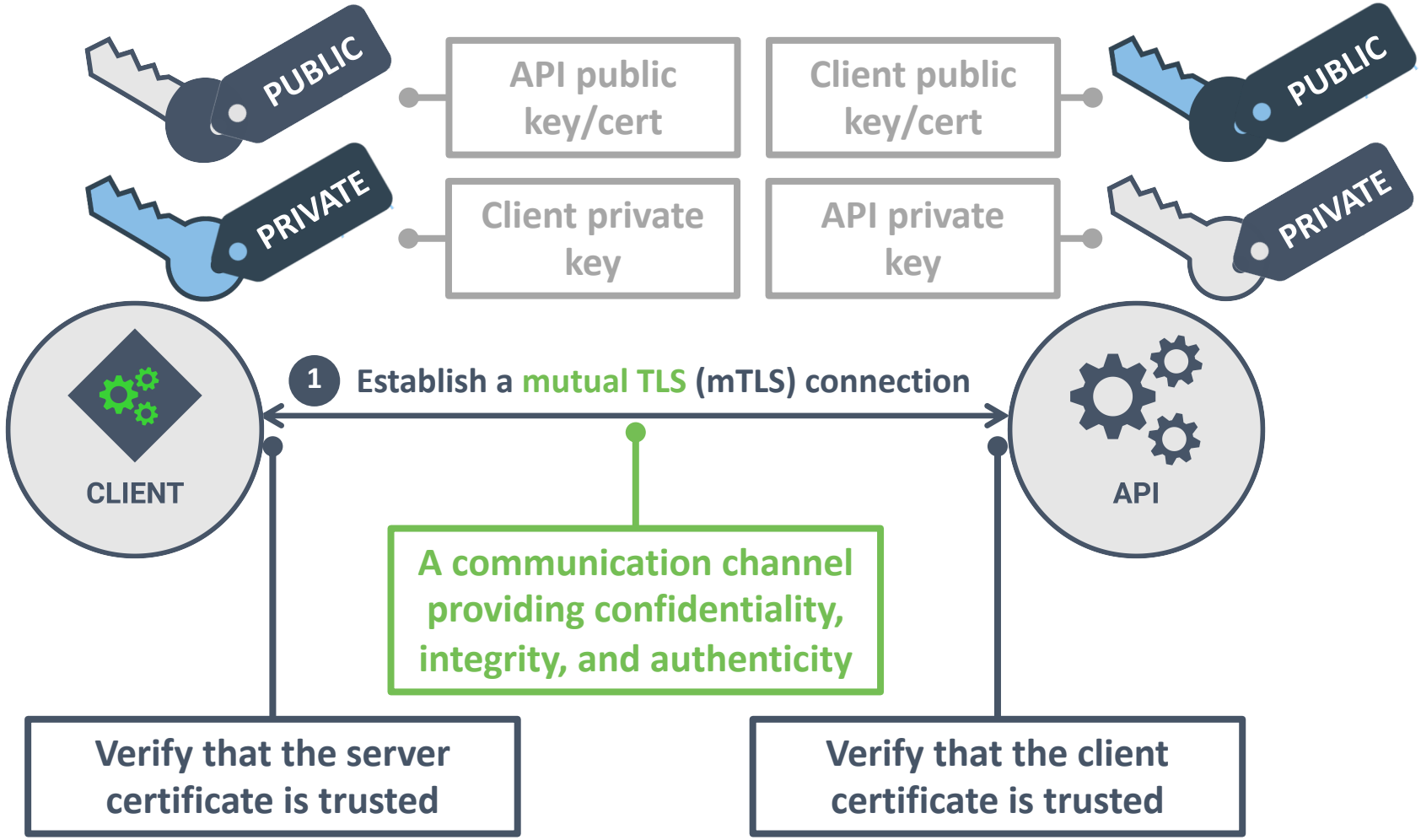
Cryptographic keys can be stored securely

DRAWBACKS

Key management / trustworthiness is challenging

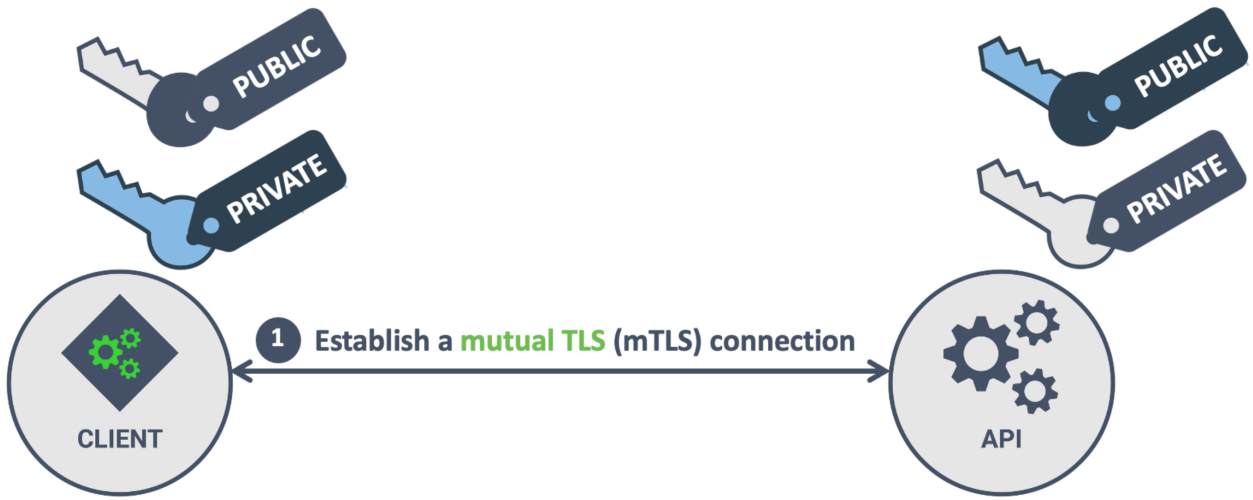
Only provides authenticity (and data integrity)

Integrity protection only applies to the signed data


PUBLIC
PRIVATE
API public key/cert
Client public key/cert
Client private key
API private key
PUBLIC
PRIVATE
CLIENT
1 Establish a mutual TLS (mTLS) connection
API
A communication channel providing confidentiality, integrity, and authenticity
Verify that the server certificate is trusted
Verify that the client certificate is trusted

USING AN MTLS CONNECTION

- *Client and server have a TLS certificate*
- *During the handshake, client and server verify trustworthiness of certificates*
- *Recommended for native applications*



BENEFITS

mTLS offers confidentiality, integrity, and authenticity

Supported in most languages / frameworks

Works with self-signed certificates if they are trusted

DRAWBACKS

mTLS does not work well with browser-based apps

Certificate and key management (PKI) is challenging

No further data besides the info from the certificate

RFC 8705
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.

Amazon API Gateway now supports mutual TLS authentication

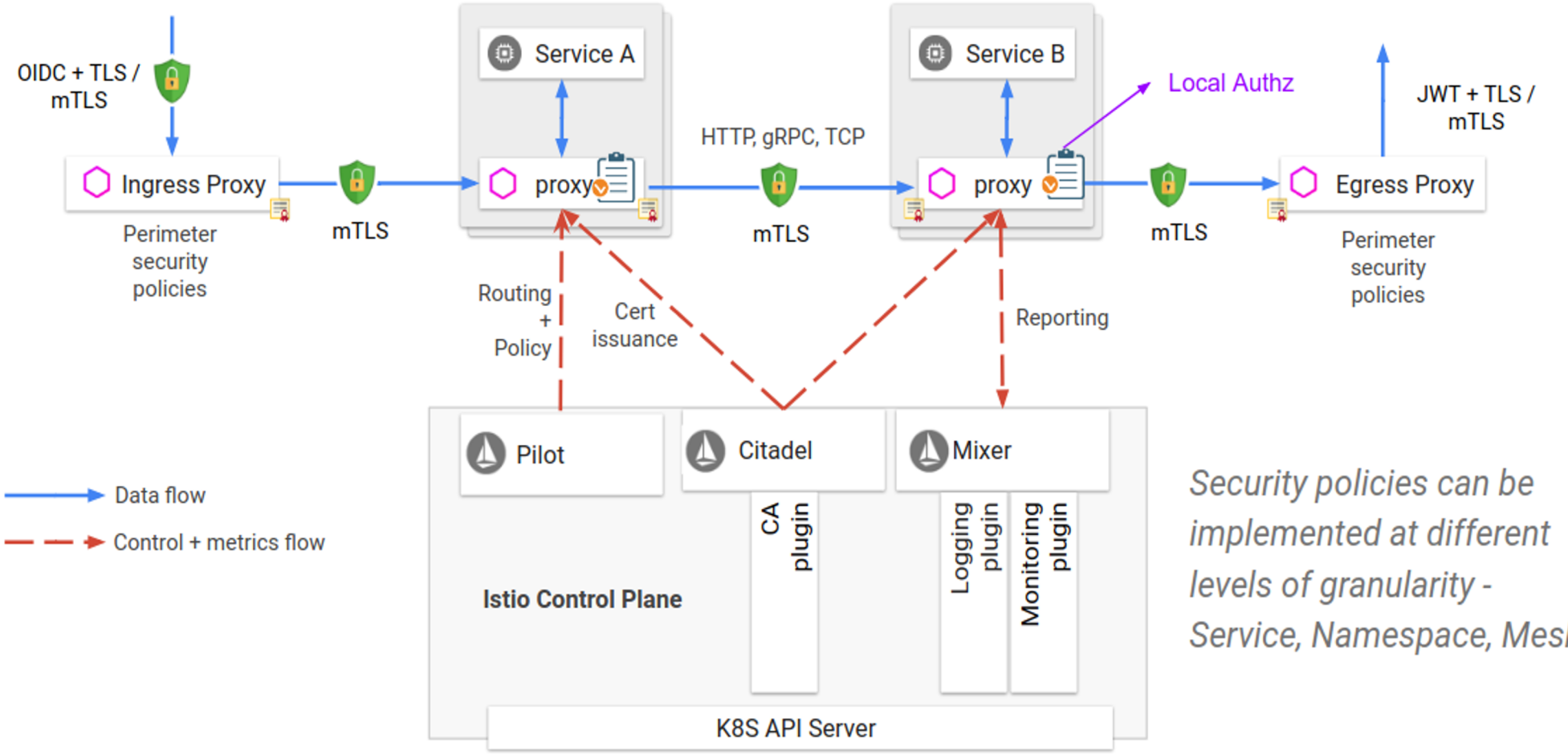
Posted On: Sep 17, 2020

Amazon API Gateway now supports mutual TLS (mTLS) authentication. Customers can now enable mTLS on custom domain names for regional REST and HTTP APIs at no additional cost. Mutual TLS enhances the security of your API and helps protect your data from attacks such as client spoofing or man-in-the middle attacks.

Historically, API Gateway has supported one-way TLS to ensure that API clients are able to verify API Gateway's identity by validating its public certificate. With this new feature, customers can now configure a custom domain name to enforce two-way TLS or mTLS which enables certificate-based authentication both ways: client-to-server and server-to-client. This helps you comply with security requirements for your Open Banking solution or easily authenticate devices in an IOT solution.

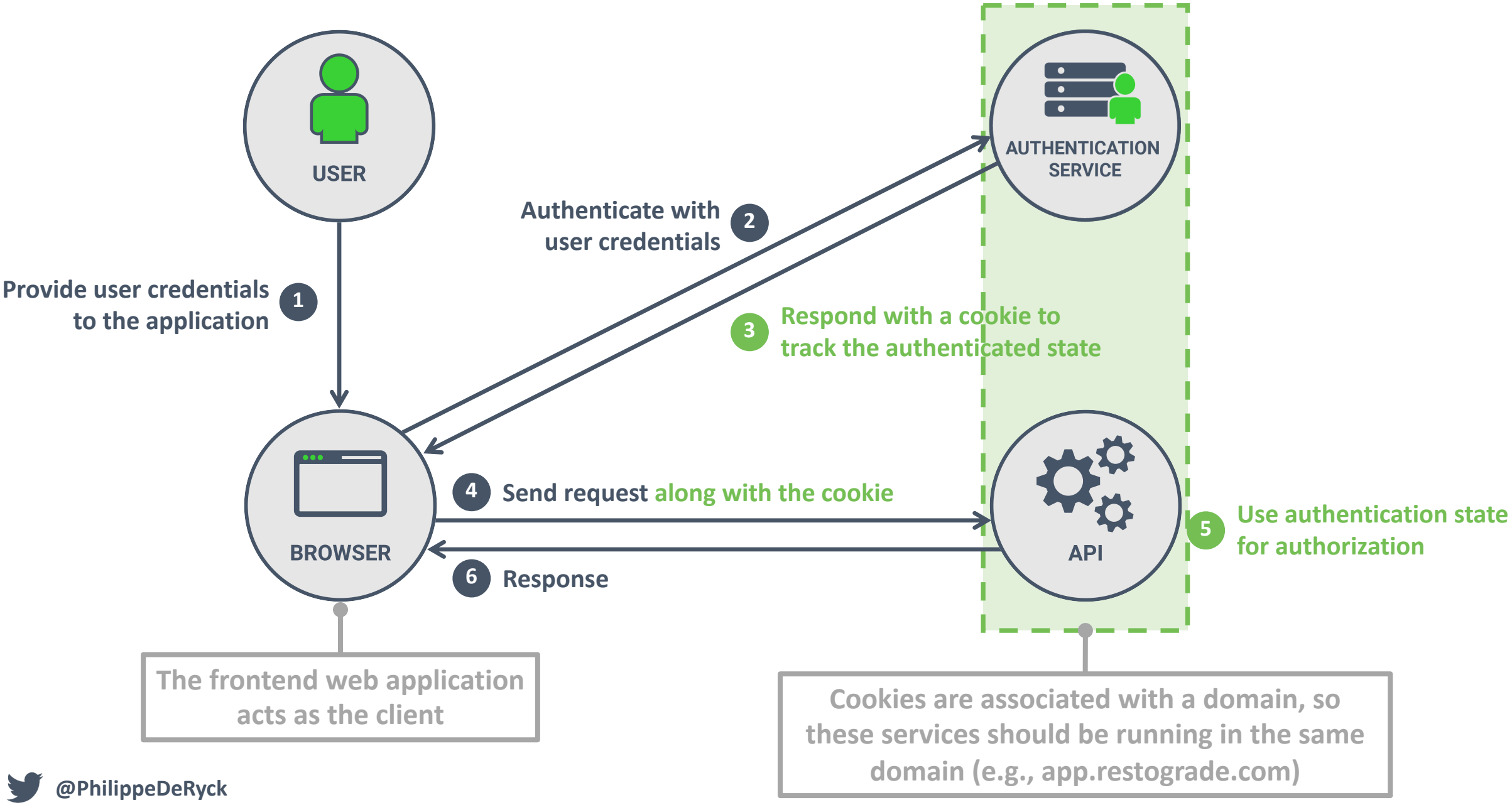
This new feature is generally available in all regions where API Gateway is available. To learn more you can read the documentation. For more information about Amazon API Gateway, visit our product page.

ISTIO SUPPORTS AUTOMATIC MTLS CONFIGURATIONS



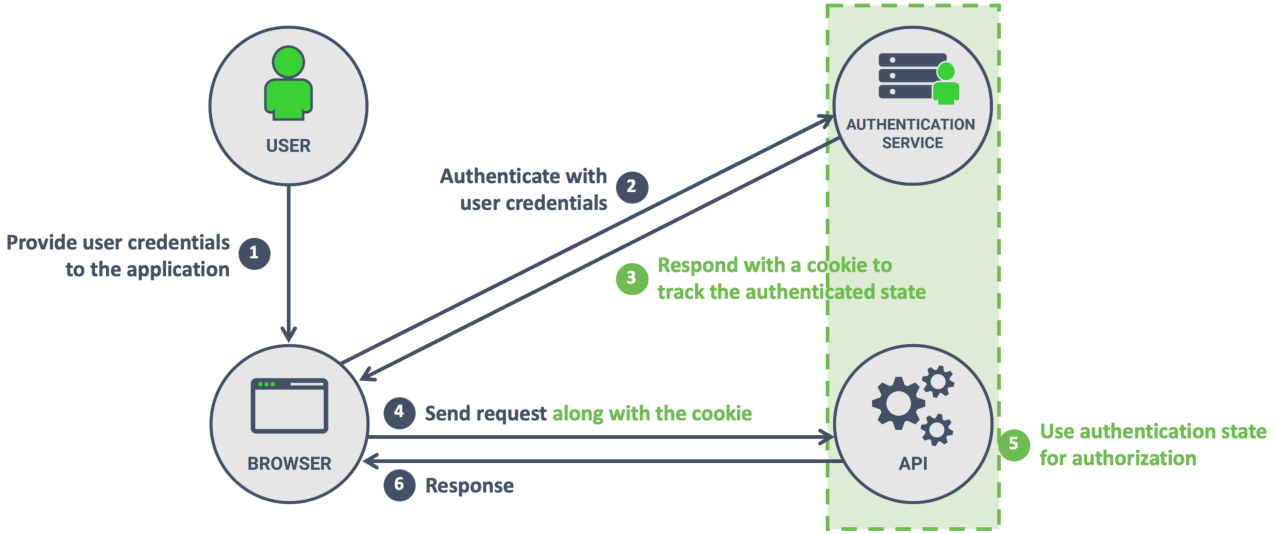
Security policies can be implemented at different levels of granularity - Service, Namespace, Mesh.

User authentication


USER
AUTHENTICATION SERVICE
Authenticate with user credentials
2
Provide user credentials to the application
1
3
Respond with a cookie to track the authenticated state
BROWSER
4
Send request along with the cookie
API
5
Use authentication state for authorization
6
Response
The frontend web application acts as the client
Cookies are associated with a domain, so these services should be running in the same domain (e.g., app.restograde.com)

COOKIE-BASED "AUTHENTICATION"

- *The user authenticates once*
- *Authentication state is tracked for the duration of a "session"*
- *Supports both stateful and stateless backend scenarios*



BENEFITS

Cookies are handled automatically by the browser

Supported by most backend frameworks

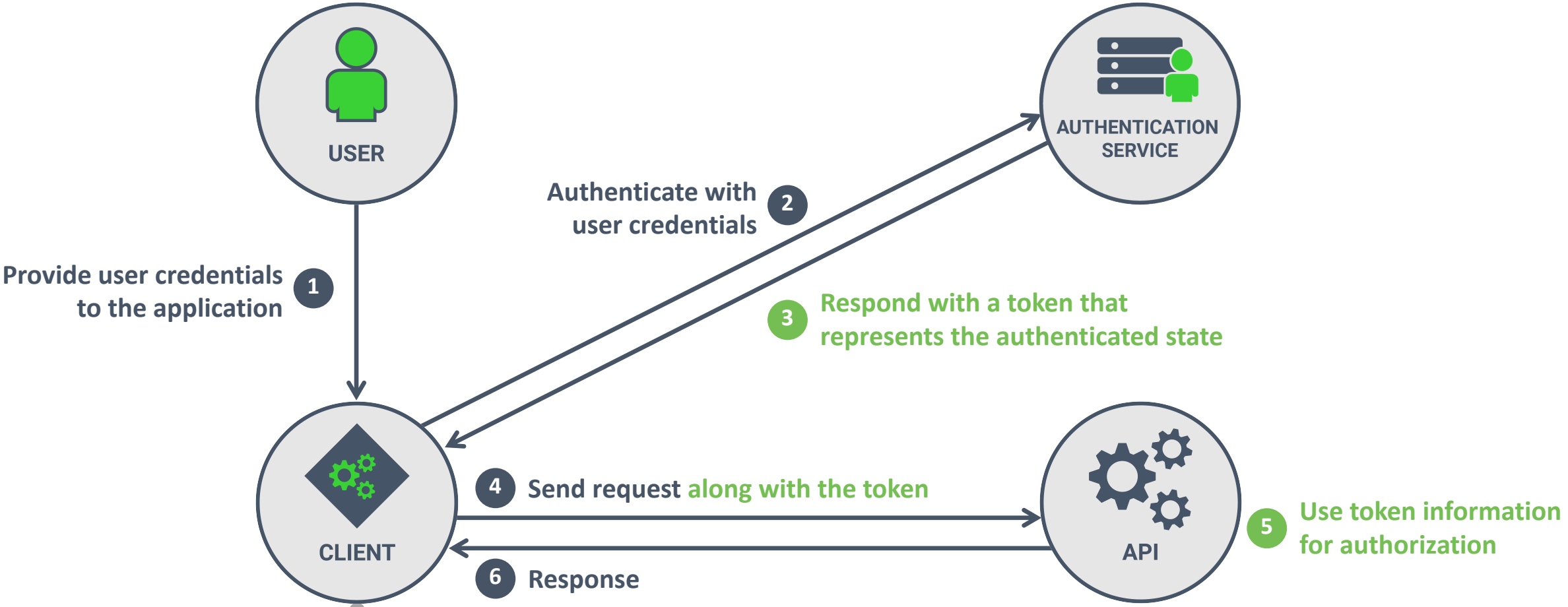
Modern browsers support advanced cookie security

DRAWBACKS

Cookies only work well in browser-based applications

Cookies only work well within a single domain

Suggesting the use of cookies makes you look uncool



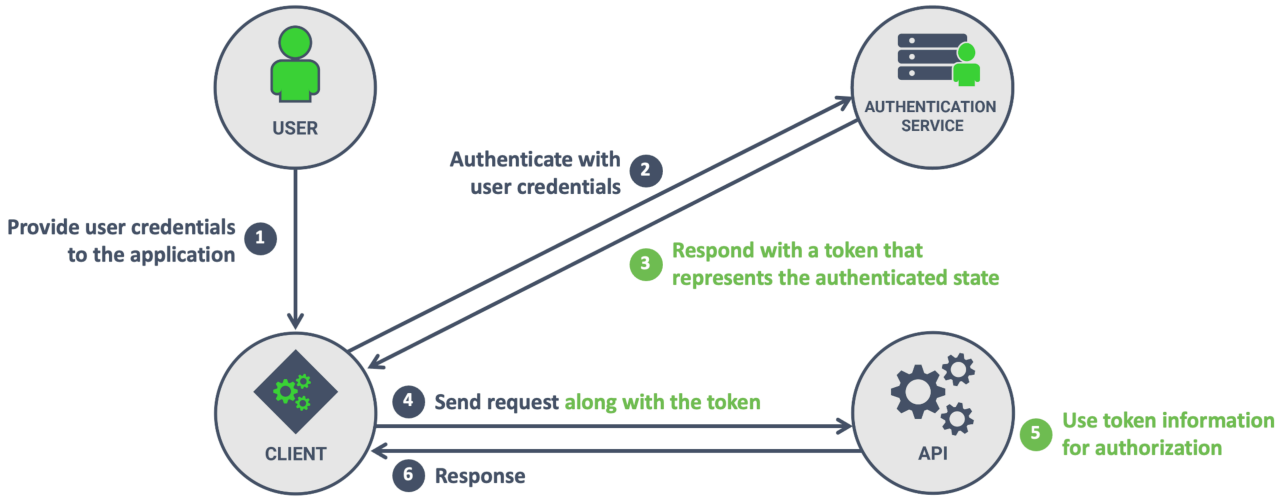
The client can be a frontend web app, a backend web app, or a native application

4 *The client includes the token in each request*

```
GET /restaurants HTTP/1.1
Host: restograde.com
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey…
```

TOKEN-BASED "AUTHENTICATION"

- *The user authenticates once*
- *Authentication state is represented by a token (typically a JWT)*
- *The client sends the token on every request to the API*



BENEFITS

Works well for all clients, including JS applications

Works well within a single application

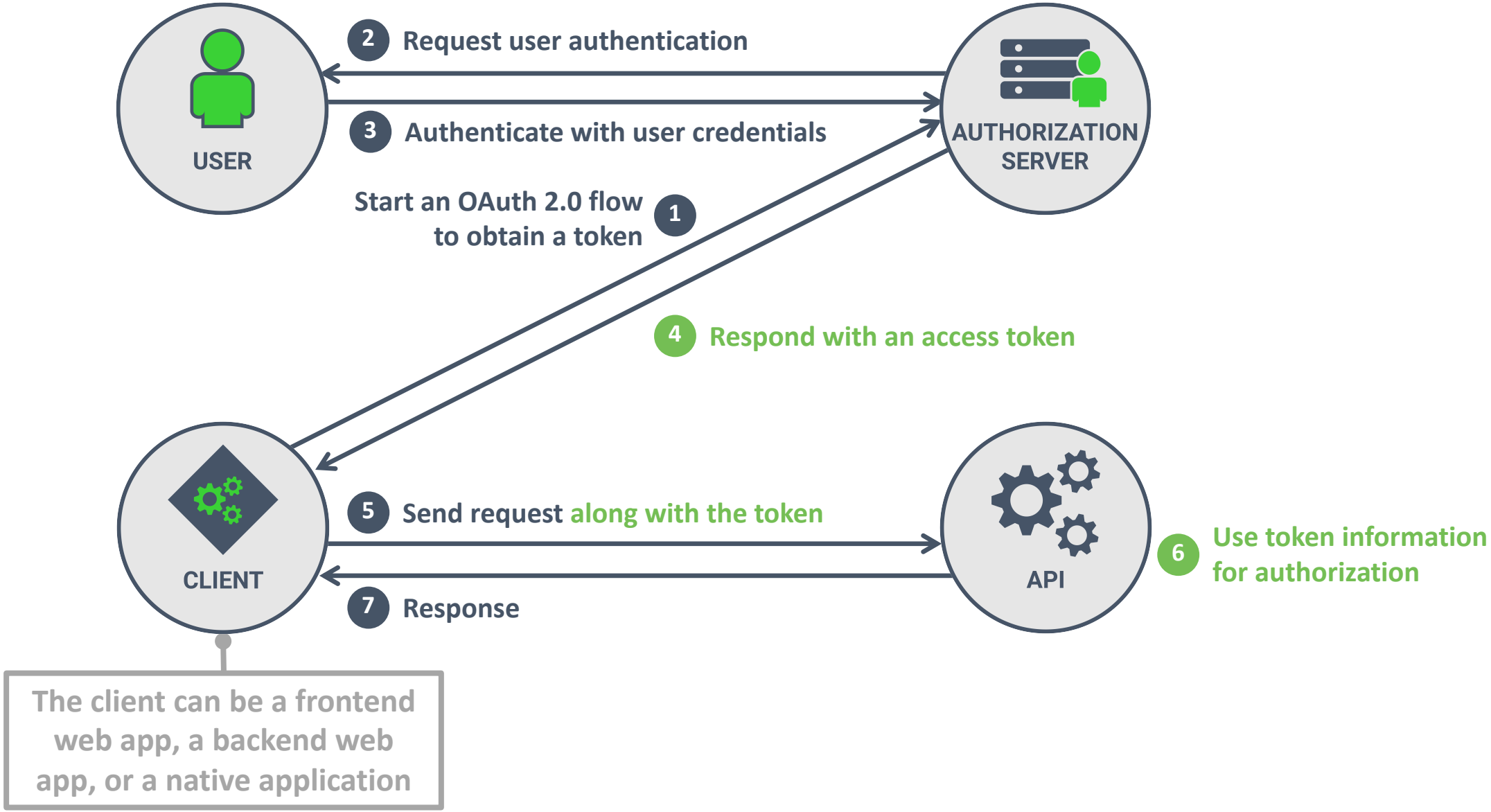
Suggesting the use of tokens makes you look cool

DRAWBACKS

Typically requires custom backend code to handle

Long-lived tokens are dangerous

Token security in browser-based applications is hard


USER
AUTHORIZATION SERVER
CLIENT
API
1 Start an OAuth 2.0 flow to obtain a token
2 Request user authentication
3 Authenticate with user credentials
4 Respond with an access token
5 Send request along with the token
6 Use token information for authorization
7 Response
The client can be a frontend web app, a backend web app, or a native application

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6Ik5UVkJPVFUzTXpCQk9FVXdOemhCUTBWR01rUTBRVVU1UVRZeFFVVXlPVU5FUVVVeE5qRXlNdyJ9.eyJpc3MiOiJodHRwczovL3N0cy5yZXN0b2dyYWRlLmNvbS8iLCJzdWIiOiJhdXRoMHw1ZWI5MTZjMjU4YmRiNTBiZjIwMzY2YzYiLCJhdWQiOlsiaHR0cHM6Ly9hcGkucmVzdG9ncmFkZS5jb20iLCJodHRwczovL3Jlc3RvZ3JhZGUuZXUuYXV0aDAuY29tL3VzZXJpbmZvIl0sImlhdCI6MTU4OTc3NTA3MiwiZXhwIjoxNTg5ODYxNDcyLCJhenAiOiJPTEtObjM4OVNVSW11ZkV4Z1JHMVJpbExTZ2RZeHdFcCIsInNjb3BlIjoiib3BlbmlkIHByb2ZpbGUgZW1haWwgb2ZmbGluZV9hY2Nlc3MifQ.XzJOXtTXOGOSbCFvp4yZGJzh7XhMmOmI2XxtjWdlODz_siI-u8h11elcr8LwX6-hL20QOW0eStzBzmm1FM_tS7MxuKkYx8QlTWOURPembVKZOhNi8kN-1j0pyc0uzve7Jib5vcxmkPwqpcVDFACgP85_0NYe4zXHKxCA5_8VOn05cRCDSkNMTFzGJCT9ipCcNXaVGdksojYGqQzezjpzzzwrtPEkiyFLFtDPZAl0MleF3oFAOCBK0UKuNjJ_cSBbUsaIwfvK0WH47AwFrRn_TxL4S1P3j3b1GgBm8tAqXysY84VZu0rSg3zrZj1PnoqPD4mbOXds20xafCr9wR4WTQ

HEADER: ALGORITHM & TOKEN TYPE


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


PAYLOAD: DATA


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

HEADER: ALGORITHM & TOKEN TYPE


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


PAYLOAD: DATA


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


The *sub* claim represents the user's unique identifier

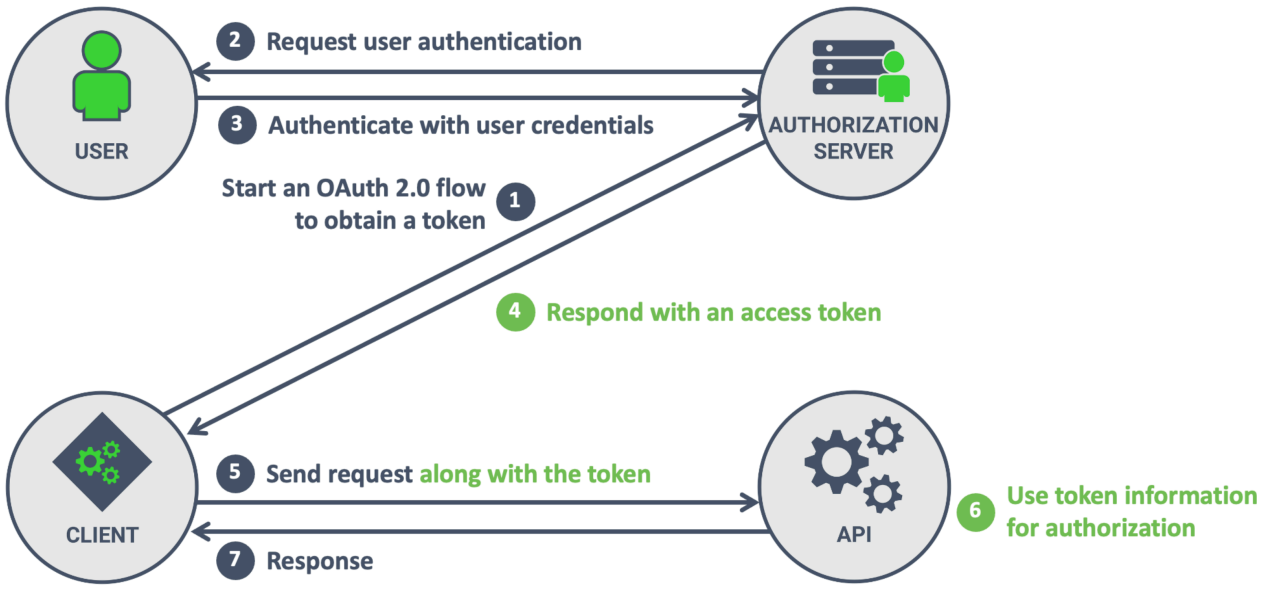
The *aud* claim represents the target APIs for this access token

The *exp* claim represents the lifetime of the token

The *azp* claim represents the client to which this token was issued

Using OAuth 2.0 access tokens

- *OAuth 2.0 allows clients to access APIs on behalf of users*
- *OAuth 2.0 supports access & refresh tokens*
- *OAuth 2.0 offers uniform support for different types of clients*



BENEFITS

Uniform authorization framework for various clients

Well-defined threat model / security considerations

Ecosystem of libraries to simplify implementation

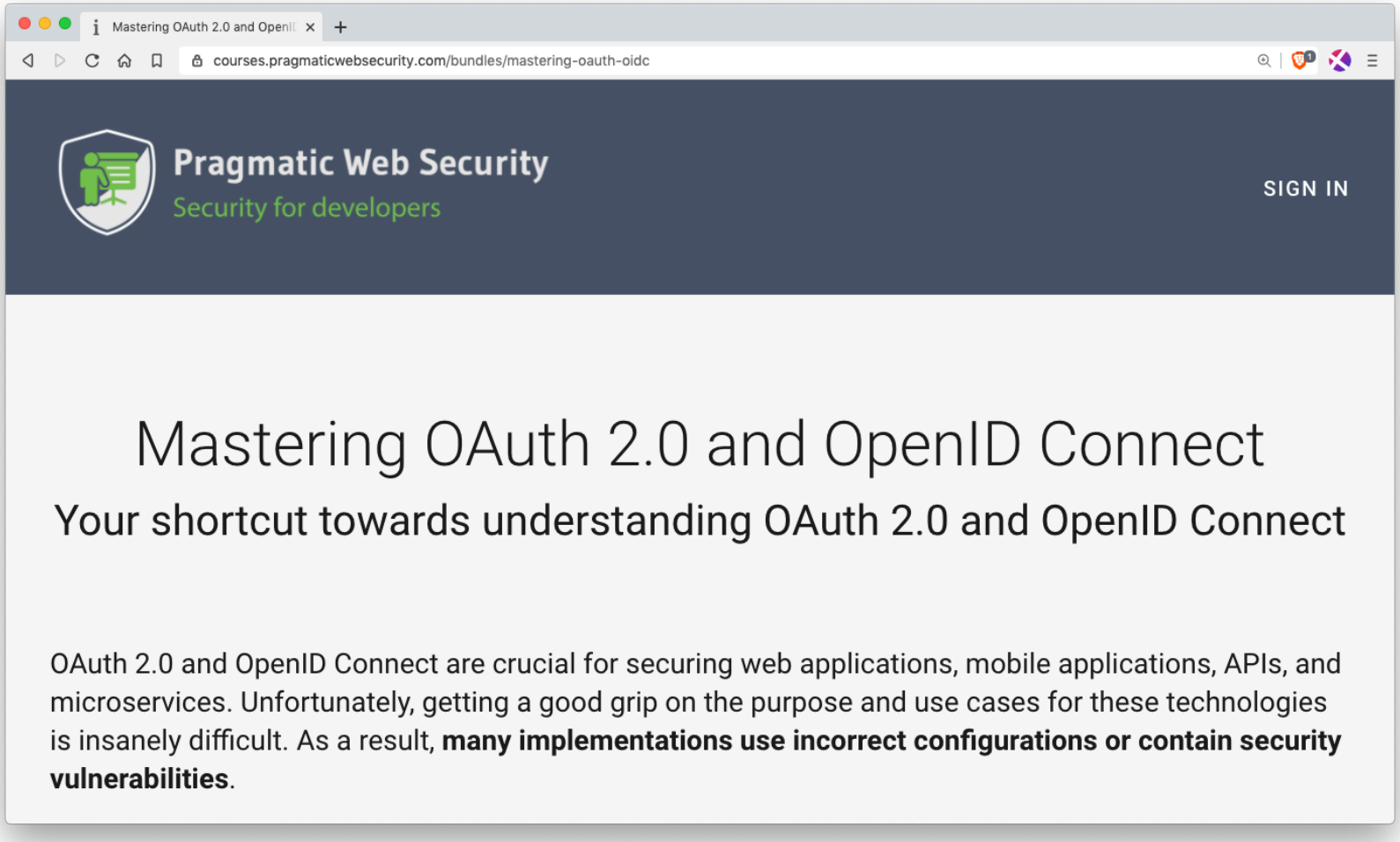
DRAWBACKS

Complex to manage in a simple architecture

User authentication typically involves the browser

OAuth 2.0 is an extensive but complicated framework

This online course condenses dozens of confusing specs into a crystal-clear academic-level learning experience



25% discount

Use coupon code

VIRTUAL_OWASP

Offer expires Feb 25th, 2021

@PhilippeDeRyck

https://courses.pragmaticwebsecurity.com

Sending a shared secret **Adding an HMAC in the request**	Simple client authentication mechanism that only works well in a single trust zone
Asymmetric request signatures	Simple client authentication mechanism that does not rely on shared secrets
Using an mTLS connection	**Recommended for service-to-service communication** to establish a secure channel
Cookie-based "authentication"	**Recommended to track authentication state within a single web application architecture**
Token-based "authentication"	Mainly useful within a single app. **Not recommended without revocable refresh tokens/sessions**
Using OAuth 2.0 access tokens	Complex but extensive authorization framework. **Recommended to support multiple clients and APIs**

USEFUL REFERENCES

- AWS request signing: https://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html
- HTTP Signatures: https://tools.ietf.org/html/draft-ietf-httpbis-message-signatures-00
- Client authentication with JWT in OAuth 2.0: https://tools.ietf.org/html/rfc7523
- Client authentication with mTLS in OAuth 2.0: https://tools.ietf.org/html/rfc8705
- Istio security architecture: https://istio.io/v1.3/docs/concepts/security/
- Additional talks on API security: https://pragmaticwebsecurity.com/talks.html
- Online courses: https://pragmaticwebsecurity.com/courses.html

Thank you for watching!

Connect on social media for more in-depth security content

@PhilippeDeRyck

/in/PhilippeDeRyck