# JWT Security

Dr. Philippe De Ryck

https://Pragmatic Web Security.com

Internet Engineering Task Force (IETF) M. Jones
Request for Comments: 7519 Microsoft
Category: Standards Track J. Bradley
ISSN: 2

Internet Engineering Task Force (IETF) M. Jones
Request for Comments: 7516 Microsoft
Category: Standards Track J. Hildebrand
ISSN: 2

Internet Engineering Task Force (IETF) M. Jones
Request for Comments: 7515 Microsoft
Category: Standards Track J. Bradley
ISSN: 2070-1721 Ping Identity

Internet Engineering Task Force (IETF) M. Jones
Request for Comments: 7517 Microsoft
Category: Standards Track May 2015
ISSN: 2070-1721

# JSON Web Key (JWK)


## Abstract

A JSON Web Key (JWK) is a JavaScript Object Notation (JSON) data structure that represents a cryptographic key. This specification also defines a JWK Set JSON data structure that represents a set of JWKs. Cryptographic algorithms and identifiers for use with this specification are described in the separate JSON Web Algorithms (JWA) specification and IANA registries established by that specification.

Internet Engineering Task Force (IETF)
Request for Comments: 7519
Category: Standards Track
ISSN: 2

M. Jones
Microsoft
J. Bradley

Abstrac

JSON
clai
are
Web
Encr
sign
(MAC

Internet Engineering Task Force (IETF)
Request for Comments: 7516
Category: Standards Track
ISSN: 2

M. Jones
Microsoft
J. Hildebrand

Abstrac

JSON
JSON
for
Web
that
Auth
JSON

Internet Engineering Task Force (IETF)
Request for Comments: 7515
Category: Standards Track
ISSN: 2070-1721

M. Jones
Microsoft
J. Bradley
Ping Identity

Abstrac

JSON
sign
data
with
Algo
spec
sepa

Internet Eng
Request f
Categ
ISSN:

M. Jones
Microsoft
May 2015

Abstrac

A JS ject Notation (JSON) data
struct phic key. This specification
also defines cture that represents a set of
JWKs. Cryptographic nd identifiers for use with this
specification are described in the separate JSON Web Algorithms (JWA)
specification and IANA registries established by that specification.

# 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**

Not Jim →

Jim →

https://pragmaticwebsecurity.com

1

JWT Signature Schemes

2

JWT Key Management

3

Ridiculous JWT vulnerabilities

4

Quiz & Summary

# JWT Signature Schemes

# By default, JWTs are …

A. Base64 encoded

B. Signed

C. Encrypted

# Encoded PASTE A TOKEN HERE


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
J1c2VyIjoiMSIsInRlbmFudCI6ImQ4Y2YzZmEzM
DFhMzRjOTY4NTAyYTcwNTFiZmRjMGE4IiwicmVz
dGF1cmFudCI6IjVlNGZkNjk5ZDZiODRjZDhiMWJ
lZTVmMDQyOGMwOTE4IiwidGVuYW50X25hbWUiOi
JUaGUgQnVyZ2VyIEdyb3VwIiwicmVzdGF1cmFud
F9uYW1lIjoiQnVyZ2VyIE1hc3RlciJ9.ag3PPzZ
c3x-jirZZn5ccsktC3PY-mW0UHwZ4-EnJ-IU
```


# Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE


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


PAYLOAD: DATA


```
  "user": "1",
  "tenant": "d8cf3fa301a34c968502a7051bfdc0a8",
  "restaurant": "5e4fd699d6b84cd8b1bee5f0428c0918",
  "tenant_name": "The Burger Group",
  "restaurant_name": "Burger Master"
}
```


VERIFY SIGNATURE

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

*Using the* java-jwt *library to decode a JWT*

```
1  String token = getTokenFromUrl(); //"eyJhbGciOiJIU...";
2  try {
3      DecodedJWT jwt = JWT.decode(token);
4  }
5  catch (JWTDecodeException exception) {
6      //Invalid token
7  }
```

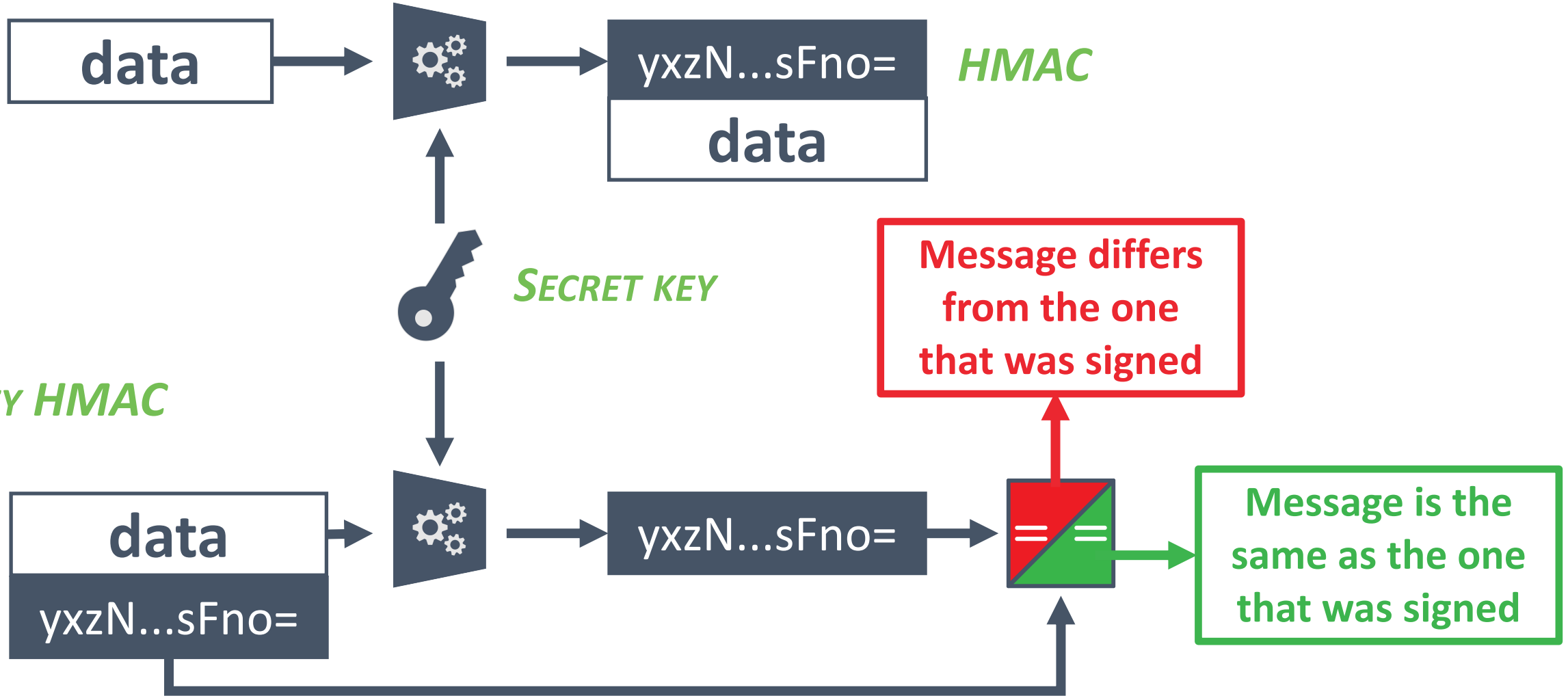
**The *decode* function returns the claims of the JWT, but does not verify the signature**

*Using the* java-jwt *library to verify the HMAC and decode a JWT*

```
1
2  String token = getTokenFromUrl(); //"eyJhbGciOiJIU...";
3  try {
4      Algorithm algorithm = Algorithm.HMAC256("secret");
5      JWTVerifier verifier = JWT.require(algorithm).build();
6      DecodedJWT jwt = verifier.verify(token);
7  }
8  catch (JWTVerificationException exception) {
9      //Invalid signature/claims
10 }
```

**The verify function on a verifier will only return the claims when the signature is valid**

## GENERATE HMAC



WEB APPLICATION SECURITY

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

OCTOBER 1, 2020 - 2 MINS READ

Decoded EDIT THE PAYLOAD AND SECRET

```
2a7051bfdc0a8",
8b1bee5f0428c0918",
up",
  restaurant_name : "Burger Master"
}
```

VERIFY SIGNATURE

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

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

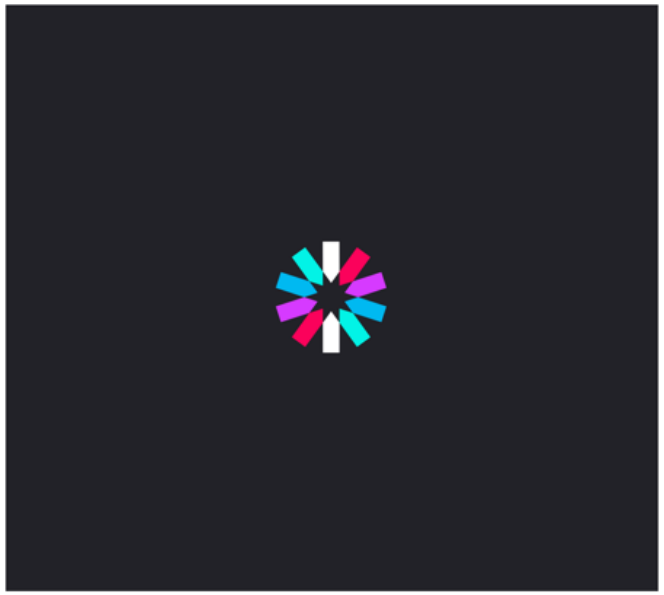
VULNERABILITIES

# 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"
```

D: DATA

```
"user": "1",
"tenant": "d8cf3fa301a34c968502a7051bfdc0a8",
"restaurant": "5e4fd699d6b84cd8b1bee5f0428c0918",
"tenant_name": "The Burger Group",
"restaurant_name": "Burger Master"
}
```

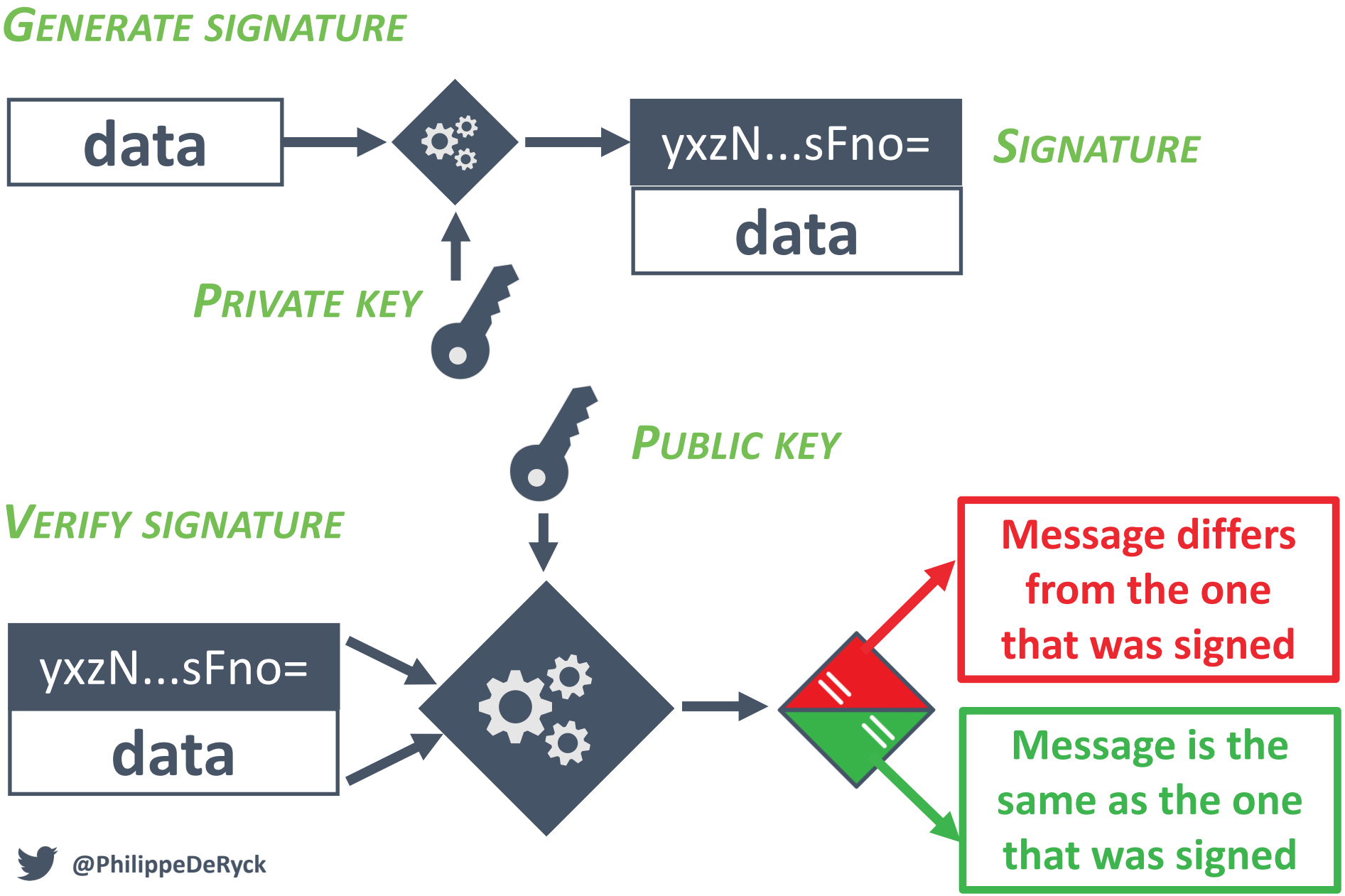
VERIFY SIGNATURE

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

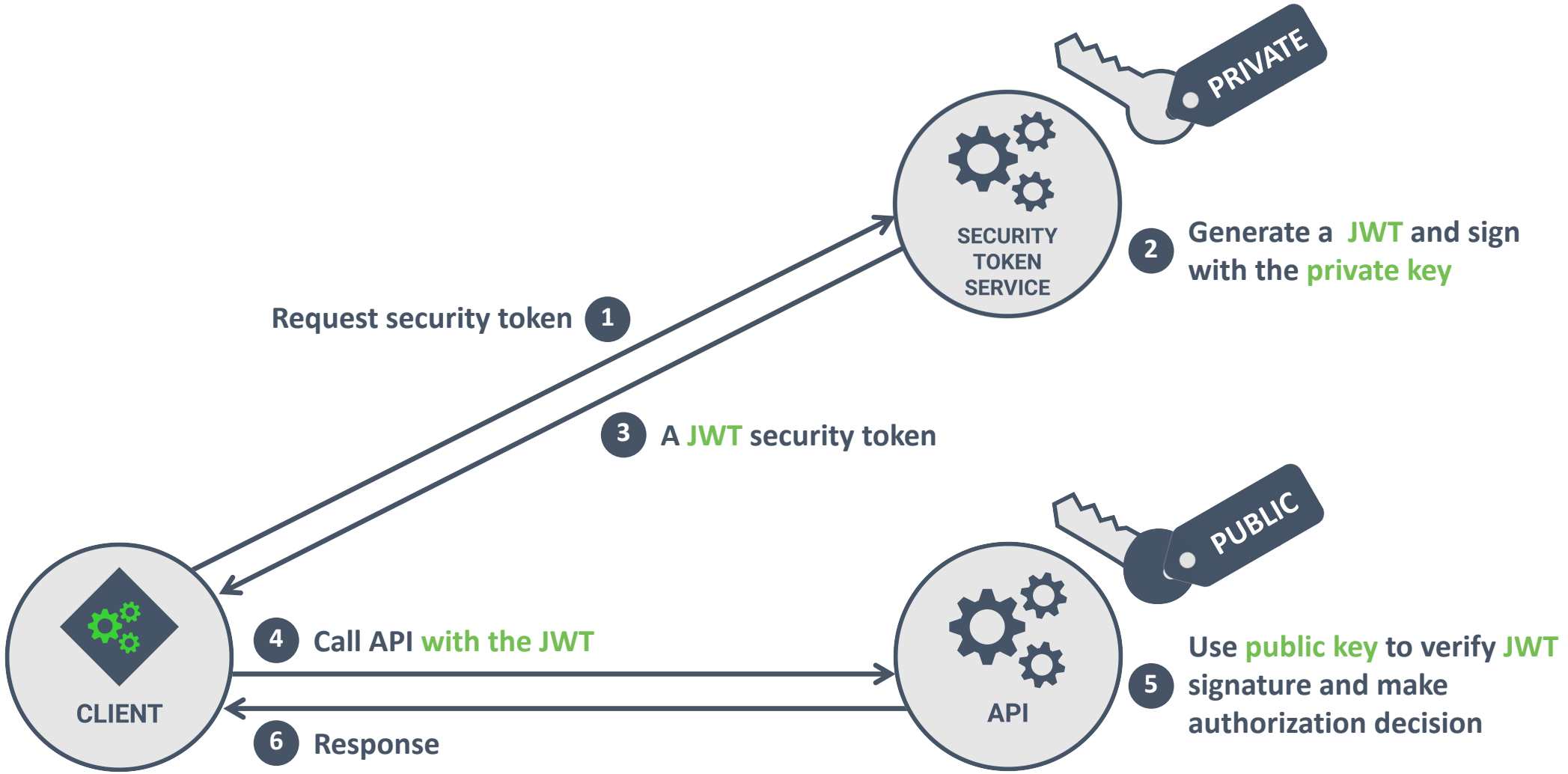
A key of the same size as the hash output (for instance, 256 bits for "HS256") or larger MUST be used with this algorithm.

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

# Asymmetric JWT signatures



# A DISTRIBUTED JWT USE CASE



1

JWT Signature Schemes

2

JWT Key Management

3

Ridiculous JWT vulnerabilities

4

Quiz & Summary

# JWT Key Management

# Which of these key distribution mechanisms are used by JWTs?

A. Static deployment (e.g., in an environment file)

B. Embedding the key in a JWT

C. Embedding the location of the key in a JWT

D. Not using keys at all

HEADER: ALGORITHM & TOKEN TYPE

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

What if there are multiple possible keys?

How does the receiver know which key to use to verify the signature?

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "d8cf3fa301a34c968502a7051bfdc0a8"
}
```

The reserved *kid* claim represents a key identifier, helping the receiver to find the right key

Useful to retrieve a key from a centralized key store

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "d8cf3fa301a34c968502a7051bfdc0a8",
  "jku": "https://sts.restograde.com/keys.json"
}
```

The reserved *jku* claim represents a URL pointing to a set of public keys that can be used to verify the signature

Since these keys are publicly available, the receiver can retrieve them from this location

The *kid* claim can be used to select the right key from the key set

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "RS256",
  "typ": "JWT",
 "x5u": "https://sts.restograde.com/cert.pem"
}
```

The reserved *x5u* claim represents the location of an X.509 certificate (TLS certificate)

Since the certificate is publicly available, the receiver can retrieve it from this location

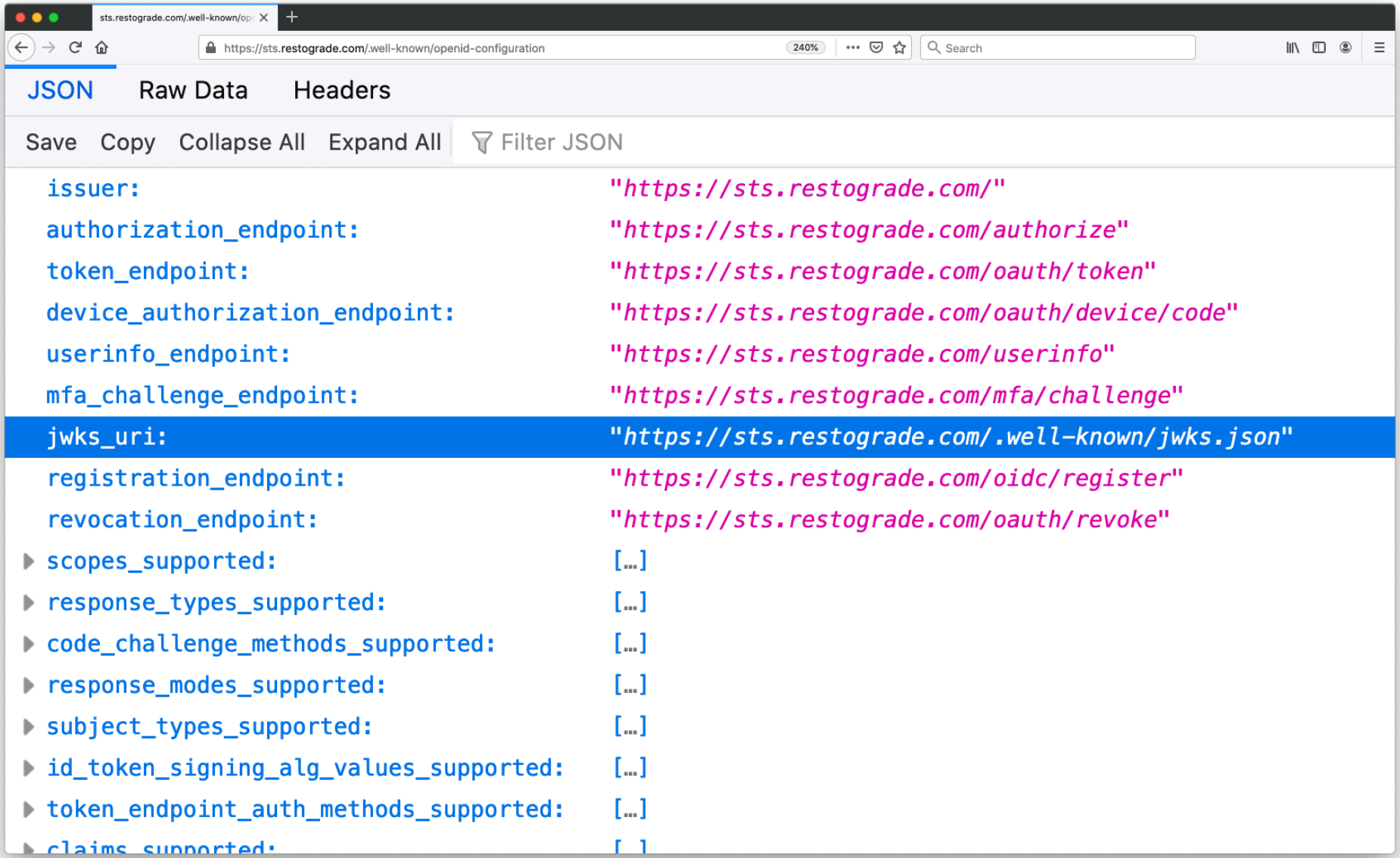
HEADER: ALGORITHM & TOKEN TYPE

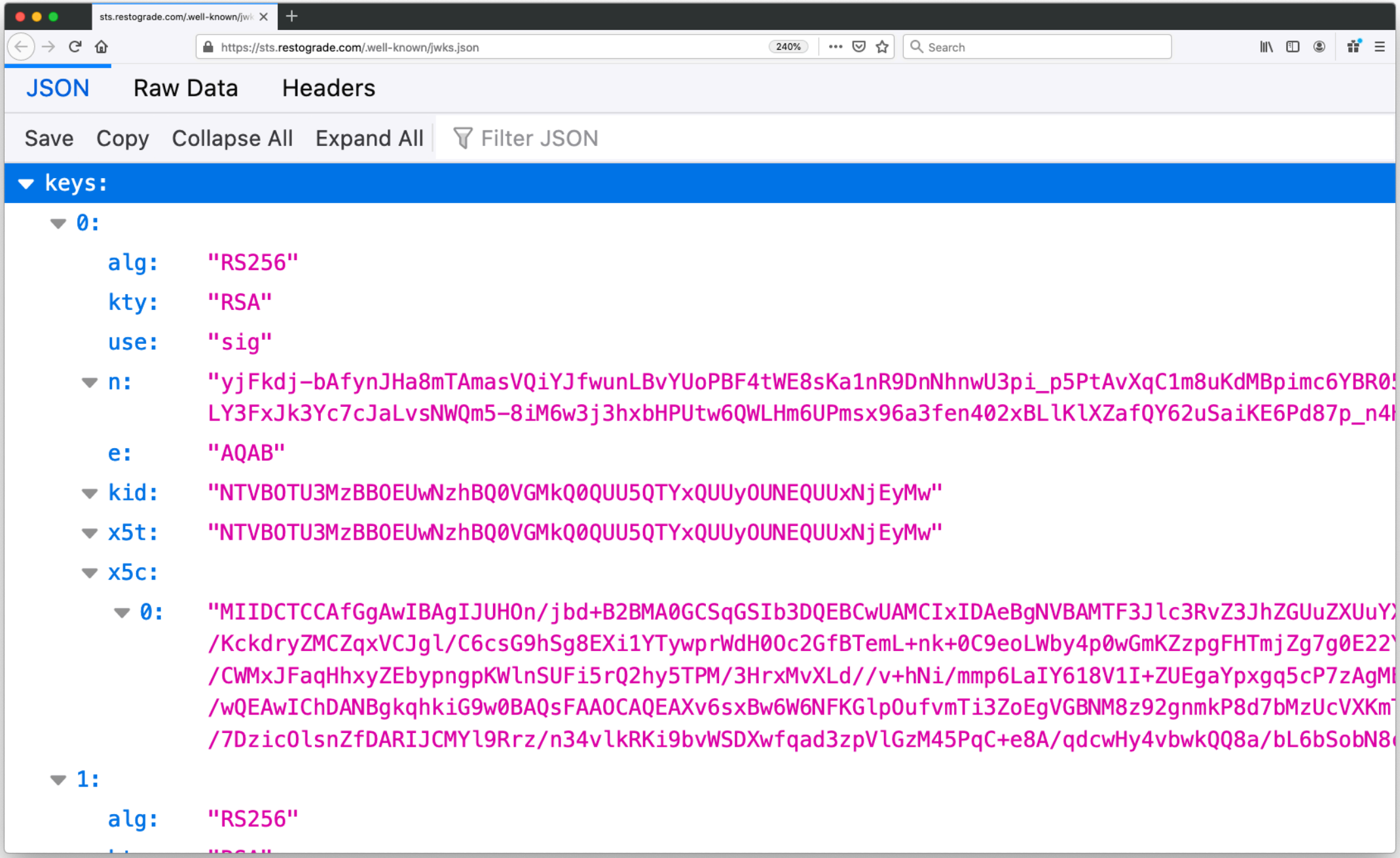
```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "666",
  "jku": "https://maliciousfood.com/evilkeyz.json"
}
```

Without proper verification, a gullible backend will retrieve the attacker's keys and use them to verify a malicious JWT token

This setup allows an attacker to provide arbitrary JWT tokens that will be considered valid, causing a major vulnerability

# .well-known/openid-configuration






@PhilippeDeRyck

1

JWT Signature Schemes

2

JWT Key Management

3

Ridiculous JWT vulnerabilities

4

Quiz & Summary

# RIDICULOUS JWT VULNERABILITIES

**HEADER:** ALGORITHM & TOKEN TYPE

```
{
  "alg": "none",
  "typ": "JWT",
  "kid": "Ae42SFaYAECQQ"
}
```

**PAYLOAD:** DATA

```
{
    "file_id": "502a7051bfdc0a8d8cf3fa301a34c968",
    "sub": "5e4fd699d6b84cd8b1bee5f0428c0918",
    "iss": "https://sts.restograde.com",
    "aud": "https://files.restograde.com",
    "iat": 1521314123,
    "exp": 1621314123
}
```

# 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.

”

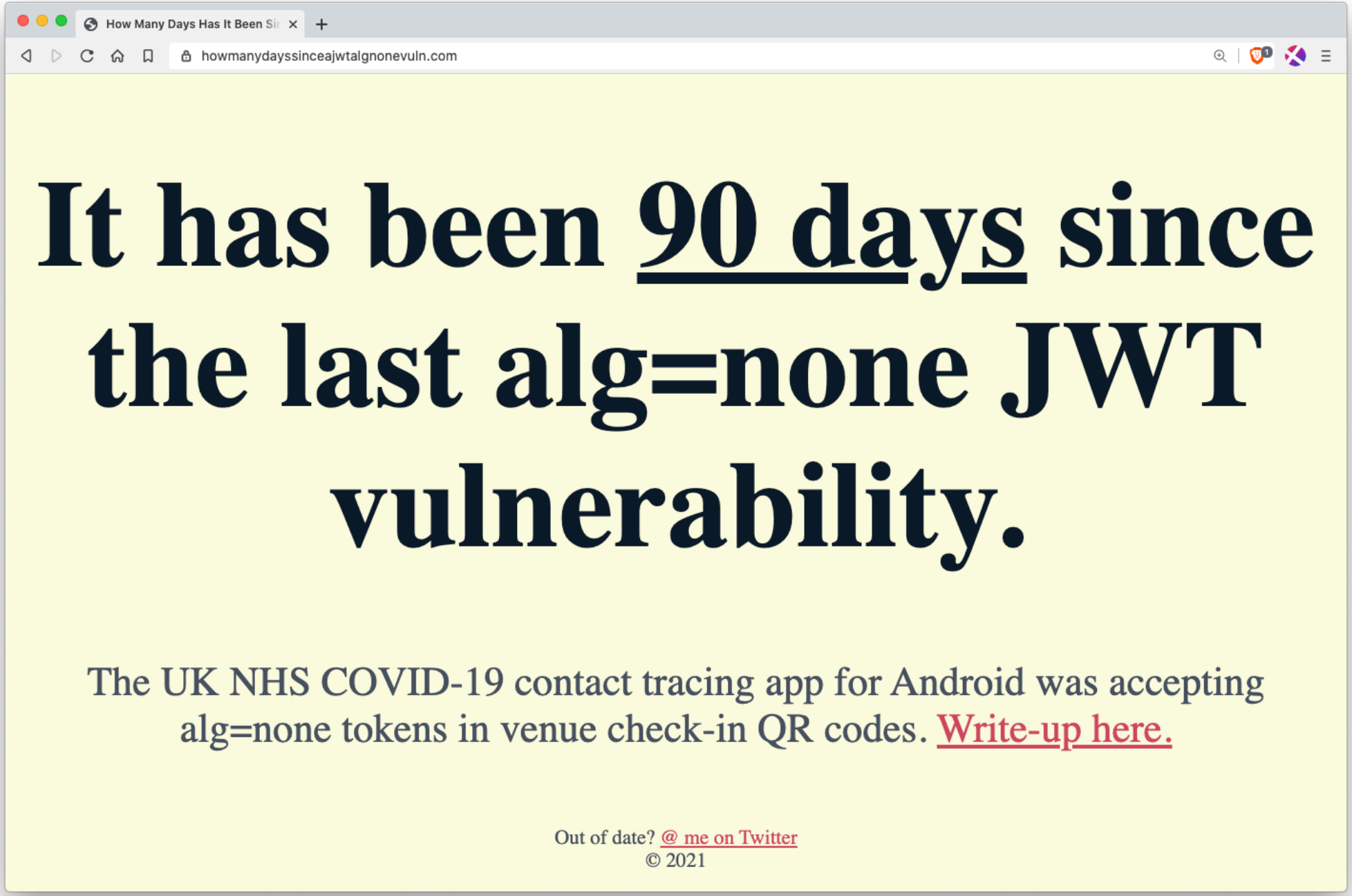
**Ben Knight** Senior Security Consultant

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*




@PhilippeDeRyck

# 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.

1

JWT Signature Schemes

2

JWT Key Management

3

Ridiculous JWT vulnerabilities

4

Quiz & Summary

# SUMMARY

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

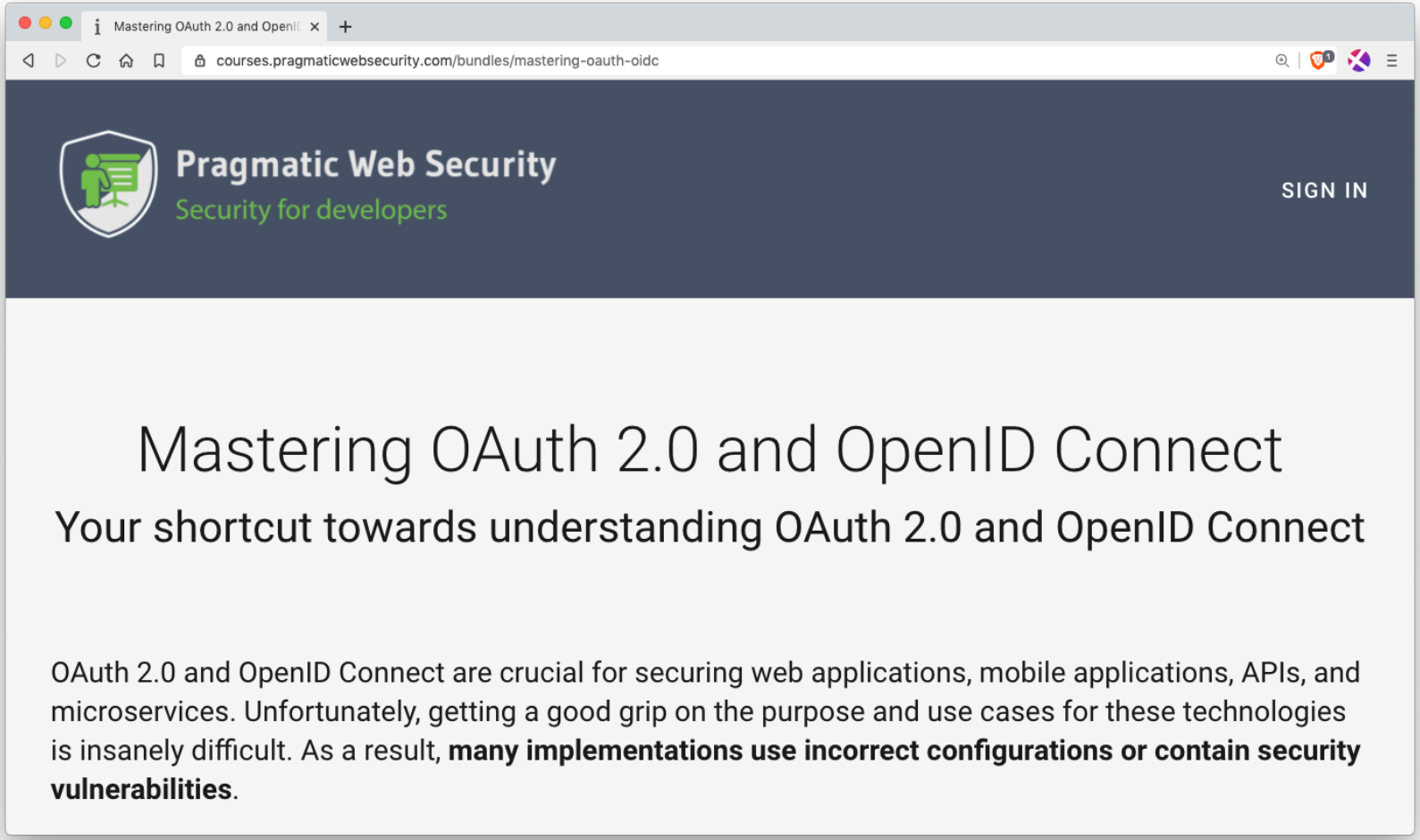
## Abstract

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.

# BEST PRACTICES JWT SECURITY

- Choose the proper signature algorithm
  - HMACs are only useful internally in an application
  - All other scenarios should rely on asymmetric signatures
    - Make sure you have a secure way to obtain the public keys of the sender
- Follow JWT security recommendations
  - Explicitly type your JWTs
  - Use strong signature algorithms
  - Use reserved claims and their meaning
- Explicitly verify the security of the backend application
  - Libraries should be actively supported and up to date
  - JWTs with *none* signatures should be rejected case-insensitively
  - JWTs with invalid signatures should be rejected

# 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*

# Thank you for watching!

## Connect on social media for more in-depth security content

← Still not Jim

@PhilippeDeRyck

/in/PhilippeDeRyck