JSON Web Token

a.k.a **JWT**

Robin CHALAS
@élao
github.com/chalasr

“ *JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties*

RFC 7519

- **Compact**

  Fit into *HTTP header*, *cookie*, *request body* parameter or *URI query* argument

- **Self-contained**

  Includes all the required informations about itself, including *what* and *why*

- **Cross-language**

  Work in *Python*, *Go*, *Node.js*, *PHP*, *Ruby*, *Javascript*, *Java* and *Haskell*

What does it look like?

```
// String composed of 3 parts separated by dot
aaaaaaaaa.bbbbbbbbbbbbb.cccccc
```

```
// Real world JWT
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
.eyJrZXkiOiJ2YWwiLCJpYXQiOjE0MjI2MDU0NDV9
.eUiabuiKv-8PYk2AkGY4Fb5KMZeorYBLw261JPQD5lM
```

Anatomy

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9        // header
.eyJrZXkiOiJ2YWwiLCJpYXQiOjE0MjI2MDU0NDV9    // claims
.eUiabuiKv-8PYk2AkGY4Fb5KMZeorYBLw261JPQD5lM // signature
```

- Header

```
// JWT's metadata
{
   "alg": "HS256", // the signature algorithm (here HMAC SHA-256)
   "typ": "JWT"    // the token type
}
```

- Claims

```
// JWT's claims
{
   "user": "chalasr",   // user-related claim (custom)
   "iat": "1484246137", // issued at (standard)
   "exp": "1484249737"  // expiration (standard)
}
```

- Signature

```
// JWT's Signature (HMAC SHA-256)
eUiabuiKv-8PYk2AkGY4Fb5KMZeorYBLw261JPQD5lM
```

Standard claims

iss (Issuer)	Who delivered the JWT
sub (Subject)	What is the subject of the JWT
aud (Audience)	Who can process the JWT
exp (Expiration time)	Until when is the JWT valid
nbf (Not before)	From when can the JWT be processed
iat (Issued at)	When the JWT has been delivered
jti (JWT ID)	What is the JWT unique identifier

Those claims are:

- defined by the RFC
- not mandatory
- recommended to use
- not relevant in all contexts

Custom claims

Share information between parties that agree on using them

```
// Common custom claims
{
    "username": "chalasr",
    "roles": [
        "ROLE_USER",
    ]
}
```

In PHP

Crypto engines

- *OpenSSL*
- *phpseclib*

JOSE libraries

- *spomky-labs/jose*
- *lcobucci/jwt*
- *namshi/jose*
- *firebase/php-jwt*

*** JOSE: Javascript Object Signing Encryption**

Creation

```php
// jwt.php
$base64UrlEncode = function (string $json) {
    return str_replace('=', '', strtr(base64_encode($json), '+/', '-_'));
};

$headers = json_encode(['typ' => 'JWT', 'alg' => 'RS256']);
$claims  = json_encode(['usr' => 'chalasr', 'exp' => time() + 3600, 'iat' => time()]);
$payload = [$base64UrlEncode($headers), $base64UrlEncode($claims)];

$privateKey = openssl_get_privatekey(file_get_contents(__DIR__.'/private.pem'), 'jwt-demo');
$signature = '';
openssl_sign(implode('.', $payload), $signature, $privateKey, OPENSSL_ALGO_SHA256);

$payload[] = $base64UrlEncode($signature);
$token = implode('.', $payload);

print $token; // JWT Created :)
```

```
$ php jwt.php

> eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
.eyJrZXkiOiJ2YWwiLCJpYXQiOjE0MjI2MDU0NDV9
.eUiabuiKv-8PYk2AkGY4Fb5KMZeorYBLw261JPQD5lM
```

Verification

```php
// verify.php
$base64UrlDecode = function (string $encoded) {
    if ($remainder = strlen($encoded) % 4) {
        $encoded .= str_repeat('=', 4 - $remainder);
    }

    return base64_decode(strtr($encoded, '-_', '+/'));
};

$token = $argv[1];
$payload = explode('.', $token);

$headers = (array) json_decode($base64UrlDecode($payload[0]));
$claims  = (array) json_decode($base64UrlDecode($payload[1]));
$signature = $base64UrlDecode($payload[2]);

$publicKey = openssl_get_publickey(file_get_contents('public.pem'));
$verified = openssl_verify($payload[0].'.'.$payload[1], $signature, $publicKey, OPENSSL_ALGO_SHA256);

if (!$verified || !isset($claims['exp']) || time() >= $claims['exp']) {
    die('Invalid JWT');
}

printf('Hello %s!', $claims['usr']); // JWT Verified :)
```

```
$ php verify.php eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
  .eyJrZXkiOiJ2YWwiLCJpYXQiOjE0MjI2MDU0NDV9
  .eUiabuiKv-8PYk2AkGY4Fb5KMZeorYBLw261JPQD5lM

> Hello chalasr!
```

Signature/Encryption algorithms

- **asymmetric**

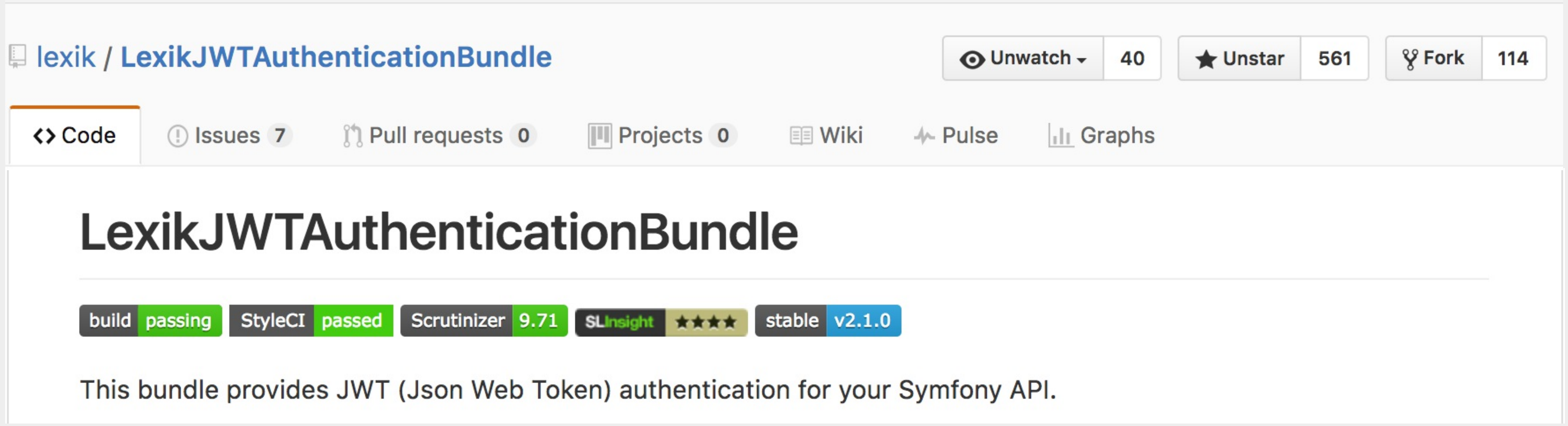
  Asymmetric cryptography is any cryptographic system that uses **pairs of keys**: *public keys* which may be disseminated widely, and *private keys* which are known **only to the owner**.
  e.g: *RS256 (RSA)*

- **symmetric**

  Symmetric cryptography is a cryptographic system in which both the sender and the receiver of a message share a **single, common key** that is used to encrypt and decrypt the message.
  e.g: *HS256 (HMAC)*

In Symfony

```
$ composer require lexik/jwt-authentication-bundle
```


lexik / LexikJWTAuthenticationBundle

LexikJWTAuthenticationBundle

This bundle provides JWT (Json Web Token) authentication for your Symfony API.

Demonstration

Thanks!

Slides: slides.com/chalasr/json-web-token

Sources: github.com/chalasr/jwt-prez