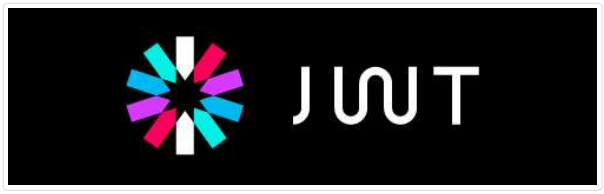
PHP

JWT – JSON Web Token em PHP

Rodrigo Aramburu

19/06/2022



JWT – JSON Web Token em PHP

Em uma aplicação web um ponto importante é a segurança, é muito comum realizarmos a autenticação de usuários no nosso sistema utilizando um formulário e gravarmos esta informação em um sessão. Mas em alguns tipos de aplicações como *APIs* este procedimento não é possível, sendo necessário utilizar outras abordagem como *JWT*(*JSON Web Token*).

O que é JWT?

JSON Web Token (*JWT*) é um padrão aberto (RFC 7519) que define uma maneira compacta e segura de transmitir objetos *JSON* entre duas partes. ***JWTs*** podem ser assinados utilizando uma chave secreta (com o uso do algoritmo **HMAC**) ou com um par de chaves publico/privada usando **RSA** or **ECDSA**.

Quando utilizar JWT

Existem dois cenários onde o *JSON Web Token* é util, no processo de autorização(em geral em APIs), e na toca de informações.

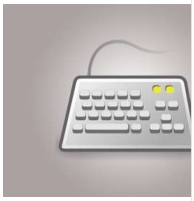
- **Autorização**: é o cenário mais comum de uso, uma vez que o usuário está logado no sistema, em cada requisição subsequente é incluido o *JSON Web Token*, permitindo o acesso a rotas serviços e recursos permitidos para aquele *token*.
- **Troca de informações**: JWT é uma boa maneira segura de trocar informações entre duas partes. Já que o *JSON Web Token* é assinado (por exemplo com um par de chaves publico/privada) você pode ter certeza de que o remetente é quem diz ser.

A Estrutura do *JSON Web Token*

O *JWT* é composto de três partes dividas por ponto (.) que são ***header***, ***payload*** e ***signature***, sendo cada parte encodada com *Base64Url*. Então o *token* se parece com ***xxxxx.yyyyy.zzzzz***.

*** Obs.**: o Base64URL é uma modificação do padrão Base64, para permitir que o resultado possa ser utilizado como nome de arquivo ou endereço de URL. A diferença é que os caracteres "+" são substituido por "-", "/" por "_" e o espaço é omitido.

MAIS POPULARES



Teclas / (barra) e (interrogação) no Lenovo S400

Criando uma janela modal simples com jQuery

jQuery UI datepicker em Português

Manipulando datas em Java

Gravando e lendo dados de um arquivo em Java

POSTS RECENTES

Manipulando imagens com Intervention Image

Header

O *header* consiste normalmente em duas partes o tipo de *token*, o qual é *JWT* e o algoritmo de assinatura que se esta usando como *HMAC SHA256* ou *RSA*. Veja um exemplo:

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

Sendo encodado em *base64url* para eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.

Payload

A segunda parte do *token* é o *payload* que contem as *claims*. As *claims* são as declarações sobre uma entidade(normalmente o usuário) com dados adicionais. Existem 3 tipo de claims: *Reserved*, *Public*, *Private*.

Reserved claims: atributos não obrigatórios (mas recomendados) que são usados na validação do *token* pelos protocolos de segurança das APIs. Alguns deles são:

- **sub**: entidade à quem o token pertence, nomalmente o id do usuário;
- **iss**: emissor do token;
- **exp**: *timestamp* de quando o token irá expirar;
- **aud**: destinatário do *token*, representa a aplicação que irá usá-lo

Public claims: atributos que usamos em nossas aplicações. Normalmente armazenamos as informações do usuário autenticado na aplicação. Exemplos ter *name*, *role*, *permissions*, etc.

Private claims: atributos definidos especialmente para compartilhar informações entre aplicações.

Um exemplo de *payload*.

```
{
    "sub": "1234567890",
    "name": "Rodrigo",
    "role": "admin"
}
```

O payload também é encodado em *base64url*. E lembrando que, **por questões de segurança, não devemos colocar nenhuma informação sensível no *payload***, já que ele é apenas encodado em *base64* e é adicionado no *token*, sendo muito fácil reverter se o *token* for interceptado.

Signature

Para criar a assinatura devemos pegar o *header* e o *payload* encodados em *base64url* e concatená-los usando um ponto (.) e usar o algoritmo *HMAC SHA256* com uma chave secreta ou certificado *RSA*.

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

Esta assinatura garante a integridade do *token*, garantindo que quem o gerou tinha a chave secreta e que as informações do *payload* não foram alteradas, já que mudar o *payload* irá alterar esta assinatura que precisa da chave para ser gerada.

Executando comandos shell pelo PHP

Editor Wysiwyg em blocos com Editor.js

Minicli – Aplicativos linha de comando em PHP

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 . eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IlJvZHJpZ28iLCJyb2xlIjoiYWRtaW4ifQ . vqoDiXce_zgbzRq4k2B70UIr1NuZ08Z_4

Você pode utilizar o Debuger do JWT conferir e gerar *tokens* manualmente para testar.

Exemplo em PHP

Para exemplificar vamos criar uma classe(muito básica) para criar e validar *tokens* JWT.

```
1  class MyJWT
2  {
3      private static function base64url_encode($data)
4      {
5          return str_replace(['+','/','='], ['-','_',''], base64_encod
6      }
7
8      private static function base64_decode_url($string)
9      {
10         return base64_decode(str_replace(['-','_'], ['+','/'], $stri
11     }
12
13     // retorna JWT
14     public static function encode(array $payload, string $secret): s
15     {
16
17         $header = json_encode([
18             "alg" => "HS256",
19             "typ" => "JWT"
20         ]);
21
22         $payload = json_encode($payload);
23
24         $header_payload = static::base64url_encode($header) . '.'.
25                           static::base64url_encode($payload);
26
27         $signature = hash_hmac('sha256', $header_payload, $secret, t
28
29         return
30             static::base64url_encode($header) . '.' .
31             static::base64url_encode($payload) . '.' .
32             static::base64url_encode($signature);
33     }
34
35     // retorna payload em formato array, ou lança um Exception
36     public static function decode(string $token, string $secret): ar
37     {
38         $token = explode('.', $token);
39         $header = static::base64_decode_url($token[0]);
40         $payload = static::base64_decode_url($token[1]);
41
42         $signature = static::base64_decode_url($token[2]);
43
44         $header_payload = $token[0] . '.' . $token[1];
45
46         if (hash_hmac('sha256', $header_payload, $secret, true) !==
47             throw new \Exception('Invalid signature');
48         }
49         return json_decode($payload, true);
50     }
51
52 }
```

Primeiro criamos os métodos auxiliares `base64url_encode` e `base64url_decode`.

Para gerar o *token* criamosum método chamado **encode**(linha 14) que recebe o *payload* em formato de *array* e a chave secreta como parâmetros. Nele criamos o *header* na linha 17 e convertemos o *payload* recebido para o formato *JSON* na linha

para gerar a assinatura *HMAC*, passando para ela o algoritmo *sha256* e valor a ser feito o *hash* que será o valor do *header* e *payload* que concatenamos, também passamos a nossa chave secreta. O retorno da função **hash_hmac** codificamos em base64url. Por fim realizamos o *return* na linha 29 concatenando o *header* e o *payload* e a assinatura encodados.

Para validar o *token* criamos o método **decode**(linha 36) que recebe o *token* e a chave secreta e irá retornar o *payload* no formato de array. Na linha 38 separamos o *token* em partes pelo caractere de ponto (.). Então com "desencodamos" o *header*, *payload* e a *signature* nas linhas 39, 40 e 42 com o método *base64url_encode*. Na linha 44 concatemanos as partes de *header* e *payload* do *token* (que ainda estão encodados com base64url). Na linha 46 usamos a função `hash_hmac` para gerar um *hash* com os dados de *header* e *payload* que extraimos do *token* recebido e que concatenamos na lina 44 utilizando a chave secreta recebida pelo método, se o *hash* for diferente da *signature* recebido no *token* então ele é inválido e lançamos uma exceção na linha 47. Se forem iguais retornamos o valor do *payload*.

Podemos testar com o seguinte código

```
1  $secret = "secret-muito-secret";
2
3  $payload = [
4      "sub" => "1234567890",
5      "name" => "Rodrigo",
6      "role" => "admin"
7  ];
8  echo MyJWT::encode($payload, $secret);
```

Para testar você pode usar o Debuger.

Para validar um token gerado podemos utilizar o seguinte código.

```
1  $secret = "secret-muito-secret";
2  $token = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3OD
3
4  print_r(MyJWT::decode($token, $secret));
```

Utilizando uma biblioteca

Existem diversas bibliotecas para criarmos JWT e validarmos *tokens* nas mais diversas linguagens, em PHP podemos utilizar a firebase/php-jwt que é muito simples e poderosa.

Para instalar basta utilizar o composer.

```
1  composer require firebase/php-jwt
```

Para gerar o JWT utilizando uma chave secreta basta chamar o método `JWT::encode` com os argumentos de *payload*, chave e algoritmos, no caso *HS256*. Para verificar chama-se o método `JWT::decode` passando o *token* JWT e um objeto *Key* que recebe em seu construtor a chave e o algoritmo. Veja o exemplo abaixo.

```php
require './vendor/autoload.php';

use Firebase\JWT\JWT;
use Firebase\JWT\Key;

$key = 'chave-de-exemplo';
$payload = [
    "sub" => "1234567890",
    "name" => "Rodrigo",
    "role" => "admin"
];

$token= JWT::encode($payload, $key, 'HS256');

$decoded = JWT::decode($token, new Key($key, 'HS256'));

print_r($decoded);
```

Para criar um *token* através de **chaves publico/privada** é bastante semelhante, basta passar a chave privada para o método de `encode` com o algoritmo ***RS256***. E para verificar chama-se o método `decode` com a chave publica. Veja um exemplo.

```php
require './vendor/autoload.php';

use Firebase\JWT\JWT;
use Firebase\JWT\Key;

//certificate.key
$privateKey = <<<EOD
-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDAdh3CtveK0IaR
RkkCtVrIrvy8GJMLjxbqhtOjnPl+m30R9AxIO3fGgQNVW3y85XRFqSr1xSmpSyDh
Y3rzO81rvqjJYfv4UnyOGeJc9MyLyfIixHQcRGNtqkFF7t1LRmoPJm1fxIaXY3fU
uhfkx72t9G8Xs/U3HRYa0SsyJpwL1Zlny1Ln22AUTjAkOLn7+WKhN3T2BoJ+pqEF
rq9OHi7JiplIernZfqrbze5Z4+ln/PATt0iF9NOrKL4uWxwpi5ueOUwaIzfeD5JL
DuGSMyzQVOw1Vkc9vXcEV4hjdDHFb6N3PyH30FoSINpHz4uVTMPCtsB+HRAOej0a
DfodxWIjAgMBAAECggEBAIBJ+5qAZc9VIm4R4Lb6djJRWkxzQQkybz8Tcl2cVD6N
nLfr6of3g+aAtt0d567ucCvDEMzUiOm9F4TFz/30fgB1DYN3WgLz0OzT2izGTNZd
XZbJRvBUscY2992K1F3J6QaCQePVVzXwar8U8LpiXlBnH+o33ZvIOqTTiqyaSegk
Zxaxv8hVDulmVypxyIfWGgarxEzmUd6luo5lmzVeMCYyi3iGDmHLA1KCwItRQATd
4oaGKP9+FYORUxrZ0g4sPaYj3x21F38y6gDFj2NMnaR6Of7Vcv35nsAQOS/ajR8f
mWbA++OP2+a8r7Mj8TeEFCLmGjcqNJgJgb0rRSqU+JkCgYEA9JAiT6zWxSbdZPax
q9zAVTJ/IIUy0zlv53vavADb+Z3qHzHCsUxgMIAKf1cNx+S8bvrMJHznMB40MuPX
4UMJTsjaVAJ9qUmz4gTC9Iji4j1myMajR8ddwcr+1HFCFyFvkp0qZQTn2wpcyaW6
T/ZFCeoXz+UeC+Z8V0aA3OsNa1cCgYEAyXY7BVz3eKAdWQ8PnrGX9spPLn4RMaiX
oHNOdXYeeDVGQZc9gp/mAk996h7R3l3Y6BG5X5mNUEe2fLjEALEWFQHyKxh4xclB
VBi1EUm6bK9sR+lR5v/s21KWFJvKILmwtG37kiYrE4vNkUsdYdscxxEX9PG9VNZO
iNbzNpXIjBUCgYAn77mKpUDVJQmWSSquU/gccYiM+PNZxtFdTb5kYQeK9Zybx3Zx
mXVDtkUAkmivscyLmuznMHZYRzZi8Q35vQayFN6CRX3bvStgx0JGyGu0Yi58ruNO
/2FyGgIPEwel21Hq8TtRD8IE5OZOD4AjOqX/fniw/EsxykuXvO2iJcglNQKBgE+s
ID9Iut3LhI+58xVxaoXBEHt0g9w9rmlX8IlngzlK6FP8Oek0z0POqB80vQ7R0nxE
tijmkwpSsgq1D16uqer48Aq3DNw7cUiO1NzXaZCd95ag4TEXuVYrXQsdaVxz0zwn
2ru7uIFsYom5SQ9wFftr+St3hsbMUhav21OU/NDZAoGBAMYstEHC/DkeOfWL5xoI
/kZd3xQ7xYn0F1DLh77rvoYRxmH8/5Y3MKaV6GmWMOCF6LmsukGhIbopK8HV8zZy
vofXGbK1qj1Yfimdalp+riQ2g8GUL9VSNGpWnrDb7yGHhT1uKLWx9h5FCQNlsY/t
CJecs4YdpGdPT/t2TX3XXfIH
-----END PRIVATE KEY-----
EOD;

// certificate.crt
$publicKey = <<<EOD
-----BEGIN CERTIFICATE-----
MIIDYDCCAkigAwIBAgIJAMnG6sKtFwEqMA0GCSqGSIb3DQEBCwUAMEUxCzAJBgNV
BAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEwHwYDVQQKDBhJbnRlcm5ldCBX
aWRnaXRzIFB0eSBMdGQwHhcNMjIwNjE4MTkxNjEzWhcNMjUwNDA3MTkxNjEzWjBF
MQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UECgwYSW50
ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIB
CgKCAQEAwHYdwrb3itCGkUZJArVayK78vBiTC48W6obTo5z5fpt9EfQMSDt3xoED
VVt8vOV0Rakq9cUpqUsg4WN68zvNa76oyWH7+FJ8jhniXPTMi8nyIsR0HERjbapB
Re7dS0ZqDyZtX8SGl2N31LoX5Me9rfRvF7P1Nx0WGtErMiacC9WZZ8tS59tgFE4w
JDi5+/lioTd09gaCfqahBa6vTh4uyYqZSHq52X6q283uWePpZ/zwE7dIhfTTqyi+
LlscKYubnjlMGiM33g+SSw7hkjMs0FTsNVZHPb13BFeIY3QxxW+jdz8h99BaEiDa
R8+LlUzDwrbAfh0QDno9Gg36HcViIwIDAQABo1MwUTAdBgNVHQ4EFgQUH9PWBAHi
eL3KeE3REc/2s7LdZE0wHwYDVR0jBBgwFoAUH9PWBAHieL3KeE3REc/2s7LdZE0w
DwYDVR0TAQH/BAUwAwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAjAd7Y1eVWkFImGNp
4C2dfr6XHraN/EfxV9oL20/8rjCUGf7mt8kauJScmZw6gxRVQJ44oYwPcDE1uhYK
rJZjK9AE0yPeP39T4p1khqhUoeoiqGpb6xZGjYFZprex7+DoSGjbqaltB3iv9OrW
Bh0H9oSCxDbswWQh6QFOvOjkOVy97DHLv20gDOpxMm/lVaFAPm3Zijt2cxejTWzT
VecQpbEUnrXjX1/0XTjCBFxPjCocszseMjRPnjopjdtWpcpY782dlKlPPemo4SJ1
pA+rScswUQrXTyUvCvSAcFJdmbZTPcDcB5h/zB3nDLpXcMY+1YRTdjdYLlrNPmw5
sfE6rg==
-----END CERTIFICATE-----
EOD;

$payload = [
    "sub" => "1234567890",
    "name" => "Rodrigo",
    "role" => "admin"
];

$jwt = JWT::encode($payload, $privateKey, 'RS256');
echo $jwt . PHP_EOL;

$decoded = JWT::decode($jwt, new Key($publicKey, 'RS256'));

print_r($decoded);
```

Usando o token

Quando é realizada a autenticação em um sistema é gerado o *token* e retornado para o cliente. Este *token* deve ser enviado novamente no *header* `Authorization` das próximas requisições HTTP com a *flag* **Bearer**

```
Authorization: Bearer <token>
```

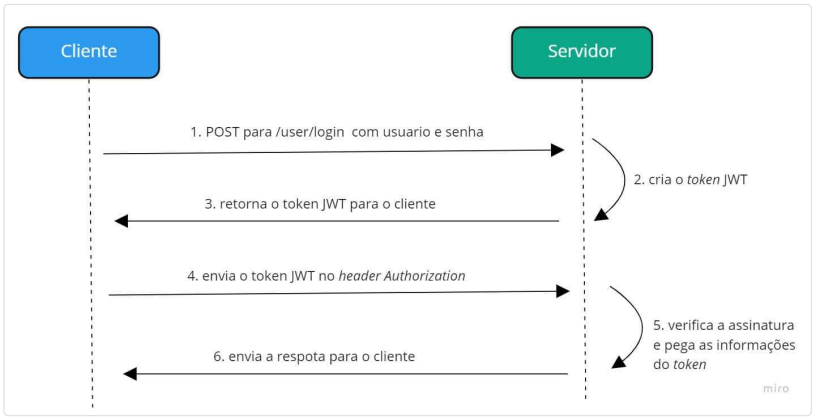


Diagrama de uso de JWT

Bom era isso. T++

autenticacao jwt