09

Para saber mais: usando chaves públicas

Nesse curso, o servidor do *Blog do Código* que gera os tokens JWT é o mesmo que verifica eles, por isso nós usamos um algoritmo simétrico (https://pt.wikipedia.org/wiki/Algoritmo_de_chave_sim%C3%A9trica) de assinatura (HMAC + SHA256), que usa apenas *uma* chave secreta, para assinar o token. Esse é um método mais fácil de implementar e muito mais rápido que outros métodos.

Entretanto, se você estiver numa situação com um servidor que gera os tokens e um ou mais servidores diferentes que verificam os tokens, então é necessário utilizar um algoritmo assimétrico (https://pt.wikipedia.org/wiki/Criptografia_de_chave_p

%C3%BAblica) para assinatura.
Os mais comuns são o RS256 (assinatura do RSA (https://pt.wikipedia.org/wiki/RSA_%28sistema_criptogr%C3%A1fico%29) + SHA256) e ES256 (assinatura do ECDSA (https://pt.wikipedia.org/wiki/ECDSA#:~:text=Em%20criptografia%2C%20o%20Elliptic%20Curve,usa%20criptogr%C3%A1fica.) + SHA256).
Como escolher eles?
Basicamente, o RSA é mais rápido mas o ECDSA permite chaves menores, então é uma escolha que depende do seu caso. De qualquer forma, ambos os métodos são bem mais lentos e complexos que o HMAC.

Vamos ver então um exemplo de JWT com o algoritmo `RS256`. Primeiro, vamos gerar as chaves pública e privada. Para isso, podemos criar um programa `generate-keys.js` como o abaixo:

```
const fs = require('fs
const { generateKeyPai

// substituir 'senha s
// e guardada em variá
const senha = 'senha s

generateKeyPair('rsa',
  modulusLength: 4096,
  publicKeyEncoding: {
    type: 'spki',
    format: 'pem'
  },
  privateKeyEncoding:
    type: 'pkcs8',
    format: 'pem',
    cipher: 'aes-256-c
    passphrase: senha
  }
}, (erro, chavePublica
    fs.writeFileSync('
    fs.writeFileSync('
    }
);
```

COPIAR CÓDIGO

VOLTAR AO TOPO

Com isso, ao r
com `node generate-keys.js`
no seu servidor de
autenticação, você vai gerar
dois arquivos:

- `public.pem` , com a chave

pública;

- `private.key` , com a chave privada criptografada.

Assim, para gerar os tokens, seu servidor de autenticação deverá fazer algo da forma

```
// [...]
const fs = require('fs
const jwt = require('j

// o arquivo 'private.
const privateKey = fs.

// substituir 'senha s
// das chaves e guarda
const senha = 'senha s

const token = jwt.sign
    payload,
    { key: privateKey,
    { algorithm: 'RS25
);

// [...]
```


e, para verificar o token, os outros servidores podem

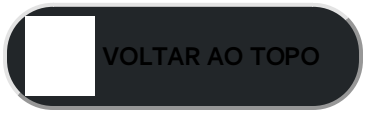
executar

```
// [...]
const fs = require('fs
const jwt = require('j

// o modo como o servi
// variar de acordo co
const publicKey = fs.r

const payload = jwt.ve
// [...]
```

COPIAR CÓDIGO

É importante notar que essa é uma implementação mais complexa, principalmente pela administração adicional das chaves que é preciso ser feita. Por isso, é recomendado ler esse artigo da Ping identity (https://www.pingidentity.com/en/company/blog/posts/2019/jwt-security-nuts-and-bolts-revealed.html)(em inglês) que explica as considerações adicionais de segurança que você precisará ter.

VOLTAR AO TOPO


VOLTAR AO TOPO