

# Spring OAuth2

Mise en place d'OAuth2 avec Spring.

[OAuth2](#) est un protocole d'autorisation et non d'authentification. Il permet de vérifier l'accès à des ressources.

OAuth2 utilise des jetons, matérialisant l'accès autorisé. L'avantage des tokens JWT (JSON Web Token) est qu'ils permettent de mémoriser de manière sécurisée des informations dans le jeton délivré.

## Dépendances

A ajouter dans **pom.xml**

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
</dependency>
```

## Sécurisation

Les clés RSA (publique et privée) sont utilisées dans le contexte d'un serveur OAuth 2 pour sécuriser les échanges de données entre les différentes parties impliquées dans le processus d'authentification et d'autorisation.

### Génération des clés

Générer une clé publique et une privée avec openssl (à installer éventuellement) :

Créer un dossier certs dans le dossier ressources de votre projet Spring.

Créer la clé privée dans **certs**

```
openssl genpkey -algorithm RSA -out private-key.pem
```

Extraire la clé publique à partir de la clé privée :

```
openssl rsa -pubout -in private-key.pem -out public-key.pem
```

Convertir la clé privée au format PKCS :

```
openssl pkcs8 -topk8 -inform PEM -outform PEM -in private-key.pem -out private-key-used.pem -nocrypt
```

La clé privée est gardée secrète et ne devra jamais être partagée, tandis que la clé publique pourra être distribuée librement.

## Usage des clés RSA

### Signature des JWT (JSON Web Tokens)

Lorsqu'un client demande un jeton d'accès au serveur OAuth, celui-ci génère un JWT contenant des informations telles que l'identifiant du client, les autorisations demandées et une empreinte temporelle. Ce JWT est signé à l'aide de la clé privée du serveur OAuth, garantissant ainsi son authenticité et son intégrité.

### Vérification des JWT

Lorsqu'un jeton d'accès est présenté pour accéder à une ressource protégée, le serveur OAuth vérifie la signature du JWT à l'aide de la clé publique associée. Si la signature est valide, cela prouve que le jeton d'accès a été émis par le serveur OAuth et qu'il n'a pas été modifié depuis sa création.

En résumé, les clés RSA (publique et privée) sont utilisées dans le cadre d'OAuth 2 pour garantir l'authenticité, l'intégrité et la sécurité des échanges de jetons d'accès entre les clients et le serveur OAuth.

## Intégration RSA/Spring

Créer une classe pour gérer les propriétés à ajouter pour stocker les 2 clés :

Dans un package **security** à créer :

```
import org.springframework.boot.context.properties.ConfigurationProperties
import java.security.interfaces.RSAPrivateKey
import java.security.interfaces.RSAPublicKey

@ConfigurationProperties(prefix = "rsa")
@JvmRecord
data class RsaKeyConfigProperties(val publicKey: RSAPublicKey, val privateKey: RSAPrivateKey)
```

Activer cette classe de propriétés directement sur la classe de votre application Spring :

```
import fr.zerp.api.security.RsaKeyConfigProperties
import org.springframework.boot.autoconfigure.SpringBootApplication
import org.springframework.boot.context.properties.EnableConfigurationProperties
import org.springframework.boot.runApplication
```

```
@SpringBootApplication
@EnableConfigurationProperties(RsaKeyConfigProperties::class)
class MyApplication
```

Ajouter les 2 clés à **application.properties** :

```
#JWT
rsa.private-key=classpath:certs/private-key-used.pem
rsa.public-key=classpath:certs/public-key.pem
```

From:  
<http://slamwiki2.kobject.net/> - **SlamWiki 2.1**

Permanent link:  
<http://slamwiki2.kobject.net/web/framework/spring/oauth2?rev=1713263265>

Last update: **2024/04/16 12:27**

