# TUTORIEL Java SE

# JPA2.0 et MySQL

# Connexion sécurisée SSL

# Introduction

Ce Tutoriel relativement basique n'a pour seul but que de présenter une démarche de mise en œuvre d'une connexion sécurisée SSL entre un programme Java, et une base de données hébergée sur un serveur MySQL en partant de zéro.

L'environnement d'exécution du code décrit dans ce tutoriel est le suivant :

- Un seul PC installé sous Windows 7 Edition intégrale 64 bits (Le « client »).
- Une machine virtuelle VirtualBox sous Linux Ubuntu 12.04 32 bits (Le « serveur »).

Sur la machine Windows, NetBeans 7.3 + JDK1.7.
Sur la machine Linux, pas de Java, uniquement un serveur MySQL 5.5.31 qui intègre openssl de base, et phpmyadmin, pour simplifier l'administration.

L'objectif est de créer un logiciel Java dans l'environnement NetBeans Windows qui devra se connecter à une base de données hébergée sur le serveur MySQL Linux en sécurisant sa connexion à l'aide du cryptage SSL afin d'avoir une circulation de l'information sur le réseau cryptée, non seulement pour la connexion, mais pour tous les échanges de requêtes/réponses.

*ATTENTION !*

*En effet, ce genre de transaction passant par des phases de cryptage/décryptage n'est pas sans effets. Les algorithmes permettant de sécuriser ces transactions sont gourmands en ressources. Il ne faut pas utiliser ce genre de connexion à tout va sans réflexions. Soit la sécurité est nécessaire, dans ce cas, la question ne se pose pas, soit pour les opérations de lecture de grande quantité de données, lorsque l'on lit une table de plusieurs dizaines de milliers d'enregistrements par exemple, on se retrouve à patienter de longues minutes avant de voir sa table se remplir…*

Quoi qu'il en soit, cette réalisation passe par deux étapes bien distinctes. La préparation de l'environnement SSL, puis l'exploitation des fichiers d'identification dans le code Java. La partie la plus ardue correspondant à l'environnement SSL puisqu'elle concerne les deux parties Client et Serveur.

# ETAPE 1
# Préparation de l'environnement SSL

## Côté Serveur :

Pour mettre en œuvre SSL, il faut commencer par créer un certain nombre de fichiers représentant des certificats d'authentification et des clés de cryptage. Les algorithmes de cryptage/décryptage s'appuient sur ces fichiers pour garantir la sécurité et l'intégrité des données.

Je n'entrerai pas dans les détails des mécanismes SSL, je vous invite cependant à visiter le lien suivant pour découvrir ce protocole : http://fr.wikipedia.org/wiki/Transport_Layer_Security

Les lignes de commandes décrites dans la suite de cette partie ont été exécutées sur la machine Linux via openssl, un utilitaire libre intégré dans l'environnement Ubuntu 12.04.

*Création d'un dossier pour stocker les fichiers d'identification :*

```
mkdir Certificats && cd Certificats
```

*Création du certificat d'authentification nommé CA :*

```
openssl genrsa 2048 > ca-key.pem
openssl req -new -x509 -nodes -days 3600 -key ca-key.pem -out ca-cert.pem
```

*Note :*
*Lors de l'exécution de la seconde ligne "openssl req...", le système vous pose quelques questions afin de définir l'identité de certification.*
*Vous devrez alors répondre aux questions suivantes :*

```
Country Name (2 letter code) [AU]:_                 // FR pour la France par exemple.
State or Province Name (full name) [Some-State]:_   // La commune, ou le quartier par exemple.
Locality Name (eg, City) []:_                       // Ici, on peut préciser la Ville par exemple.
Organization Name (eg, company) [blablabla...]:_    // Le générique "PRIVE" par exemple.
Organizational Unit Name (eg, section) []:_         // Le générique "Domicile" par exemple.
Common Name (e.g. server FQDN or Your Name) []:_    // Son prénom par exemple.
Email Address []:_                                  // Sans commentaire...
```

*Création du certificat serveur et signature de ce dernier :*

```
openssl req -newkey rsa:2048 -days 3600 -nodes -keyout server-key.pem -out server-req.pem
```

*Note :*
*Lors de l'exécution de cette ligne, le système vous pose quelques questions afin de définir l'identité associée. Vous devrez alors répondre aux questions suivantes*

*Le même lot de question que précédemment, puis les deux questions suivantes :*

*Please enter the following 'extra' attributes*
*to be sent with your certificate request*
*A challenge password []:_ // Tapez alors un mot de passe. Je vous conseil d'utiliser le même pour toutes les autres manipulations nécessitant un mot de passe.*
*An optional company name []:_ // Ce que vous voulez, ou "." pour rien.*

*On poursuit avec :*

```
openssl rsa -in server-key.pem -out server-key.pem
openssl x509 -req -in server-req.pem -days 3600 -CA ca-cert.pem -CAkey ca-key.pem -
set_serial 01 -out server-cert.pem
```

*Création du certificat client et signature de ce dernier :*

```
openssl req -newkey rsa:2048 -days 3600 -nodes -keyout client-key.pem -out client-req.pem
```

*Note :*
*Une fois de plus, la même série de question vous est posée, répondez cette fois en vous mettant dans la peau du client.*

```
openssl rsa -in client-key.pem -out client-key.pem
openssl x509 -req -in client-req.pem -days 3600 -CA ca-cert.pem -CAkey ca-key.pem -
set_serial 01 -out client-cert.pem
```

*Le certificat client est maintenant généré, pour le vérifier faites :*

```
openssl verify -CAfile ca-cert.pem server-cert.pem client-cert.pem
```

*Si vous identifiez la présence des deux lignes suivante dans la réponse, c'est bon.*
*server-cert.pem: OK*
*client-cert.pem: OK*

Maintenant, dans le dossier exemple créé ci-dessus se trouve huit fichiers :

ca-cert.pem
ca-key.pem
client-cert.pem
client-key.pem
client-req.pem
server-cert.pem
server-key.pem
server-req.pem

Pour terminer la génération des fichiers d'authentification côté serveur, il en reste un à créer. Le fichier représentant le certificat client doit être au format DER pour pouvoir être utilisé par l'API JSSE de Java. Il faut donc procéder à une conversion à l'aide de la commande suivante :

```
Openssl x509 –outform DER –in client-cert.pem –out client.cert
```

Voilà, tous les fichiers pem nécessaires sont maintenant créés. De cet ensemble de fichiers, seuls quatre d'entre eux nous intéressent :

| | |
|---|---|
| ca-cert.pem | *le certificat d'authentification* |
| server-cert.pem | *le certificat du serveur* |
| server-key.pem | *la clé secrete du serveur* |
| client.cert | *le certificate du client* |

Pour exploiter ce résultat avec le serveur MySQL, il faut maintenant copier les trois fichiers ca-cert.pem, server-cert.pem et server-key.pem dans l'arborescence MySQL. Sur la machine Linux, il s'agit du dossier /etc/mysql
Dans ce dossier se trouve le fichier de configuration de mysql, il s'agit du fichier my.cnf dans ce cas. Il faut éditer ce fichier pour y cibler les lignes suivantes :

```
ssl-ca=/etc/mysql/ca-cert.pem
ssl-cert=/etc/mysql/server-cert.pem
ssl-key=/etc/mysql/server-key.pem
```

A la base, elles sont commentées (précédées du caractère #) et leur valeur nécessite d'être adaptée. Dans notre cas, le fichier my.cnf contient ces trois lignes telles quel.

Après cette modification, on redémarre le serveur mysql et ce dernier acceptera les connexions SSL et les acceptera si elles sont certifiées correctement. L'échange de données sera alors crypté sur le réseau.
Pour la partie Java, comme l'environnement n'existe pas sur la machine Linux de notre exemple, elle se fera sur la machine Windows. En effet, cette partie va nous faire utiliser un outil intégré au jdk, l'outil keytool, pour générer les fichiers truststore et keystore qu'utilisera JSSE lorsque le code sollicitera la connexion sécurisée. Et cet outil a besoin de deux fichiers, ca-cert.pem et client.cert
Il faut donc copier ces deux fichiers sur une clé USB, ou les déposer sur une ressource commune aux deux machines, le temps de les récupérer sur la machine Windows.

## Côté client :

Nous voilà sur la machine Windows, dans une fenêtre de commande MS-DOS. On se place dans notre dossier de profile, pour ma part, cela donne C:\Users\thierry

Ensuite, j'ai copié mes deux fichiers ca-cert.pem et client.cert. Les lignes qui suivent représentent les commandes exécutées à l'aide de l'outil keytool du jdk :

```
keytool -import -alias mysqlServerCACert -file ca-cert.pem -keystore trustsore

Puis :

Keytool –import –file client.cert –keystore keystore –alias mysqlClientCertificate
```

Lors de l'exécution de ces deux lignes, vous serez amené à fournir un mot de passe. Vous pouvez fournir le mot de passe que vous voulez, sachez simplement que ce sera le mot de passe à utiliser dans le code java, donc attention à ne pas se tromper.

Quoi qu'il en soit, la première étape consistant à créer nos cinq fichiers (ca-cert.pem, server-cert.pem, server-key.pem, truststore et keystore) est terminée. Nous pouvons passer à l'écriture du code de test de connexion sécurisée SSL avec notre serveur MySQL.

# ETAPE 2
# Création du Client Java

Du côté Java, c'est plus simple. Maintenant que nous avons à notre disposition  les fichiers keystore et truststore, il ne reste plus qu'à les fournir à l'API JSSE qui sera invoquée par la machine virtuelle lors de l'établissement de la connexion sécurisée. Rappelons que ces derniers se trouvent actuellement dans le dossier de profile de l'utilisateur : C:\Users\thierry

Pour pouvoir fournir ces fichiers à JSSE, il faut déclarer quatre variables système pour la JVM. Ces variables sont :

javax.net.ssl.keystore
javax.net.ssl.keyStorePassword
javax.net.ssl.trusstore
javax.net.ssl.trustStorePassword

Dans notre code exemple, on ajoutera ces variables comme suit :

```
System.setProperty("javax.net.ssl.keyStore","C:\\Users\\thierry\\keystore");
System.setProperty("javax.net.ssl.keyStorePassword","mot de passe");// le mdp du fichier keystore.
System.setProperty("javax.net.ssl.trustStore","C:\\Users\\thierry\\truststore");
System.setProperty("javax.net.ssl.trustStorePassword","mot de passe"); // le mdp du fichier truststore.
System.setProperty("javax.net.debug","all");
```

La dernière ligne, javax.net.debug, n'est pas indispensable pour le fonctionnement. Elle permet cependant d'obtenir les traces verbeuses du dialogue SSL vers le flux de sortie standard. Cela permet de pister d'éventuels problèmes dans la chaîne de transmission SSL.

Ensuite, il y a plusieurs possibilités. L'utilisation d'une connexion directe à travers le driver jdbcMySql en utilisant une instance de javax.sql.Connection, ou bien par une instance de classe EntityManager par exemple. Dans le cas d'une connexion directe par le driver, il suffit d'ajouter l'attribut « useSSL=true » à l'objet Properties fourni à la méthode DriverManager.getConnection(url, proprietesDeConnexion). Cela donne :

```
Properties proprietesDeConnexion = new Properties();
proprietesDeConnexion.setProperty("user", "utilisateur"); // Utilisateur MySQL.
proprietesDeConnexion.setProperty("password", "mot de passe); // Mot de passe de l'utilisateur MySQL.
proprietesDeConnexion.setProperty("useSSL", "true");
      try {
         String url = "jdbc:mysql://192.168.0.40:3306/bibdna";
         conn = DriverManager.getConnection(url, proprietesDeConnexion);
         System.out.println("Connection effective !");}
      catch (Exception e) {
         e.printStackTrace();}
```