Sécurité Internet et Réseau

Rapport de TP

Application répartie et sécurisée

Auteurs

Christophe GUIEU

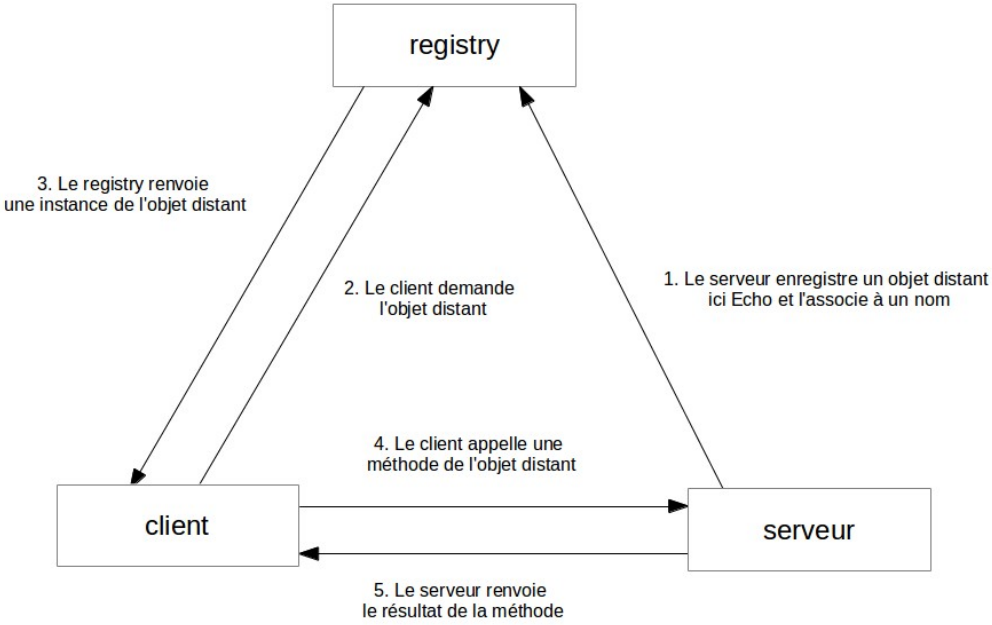
Cyriaque SERRE

Table des matières

1 Architecture rmi de base (en clair)

1.1 Vue globale

Voici l'architecture rmi que notre projet va tenter de sécuriser par l'ajout de l'authentification de toute les parties et de la confidentialité et l'intégrité de toutes les communications.



1.2 Présentation du serveur

1.2.1 Description du code

Le code du serveur se compose de deux fichiers : Echo.java et EchoImpl.java.

Echo.java :

```
import java.rmi.*;

public interface Echo extends Remote {
	public String echo(String str) throws RemoteException;
}
```

EchoImpl.java :

```
import java.rmi.*;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.*;

public class EchoImpl extends UnicastRemoteObject implements Echo {
	private static final long serialVersionUID = 1L;
	public EchoImpl() throws RemoteException {
		super() ;
	}
	public String echo(String str) throws RemoteException {
		String ret;
		System.out.println("Recu: " + str);
		ret = "Echo: " + str;
		System.out.println("Renvoi: " + ret);
		return ret;
	}
	public static void main (String args[]) throws Exception {
		Registry registry = LocateRegistry.getRegistry(null, 1099);
		EchoImpl obj = new EchoImpl();
		registry.rebind("rmi://localhost/EchoServer", obj);
		System.out.println("EchoServer bound in registry");
	}
}
```

1.2.2 Description de la commande

```
java EchoImpl
```

1.3 Présentation du client

1.3.1 Description du code

Le code du serveur se compose de deux fichiers : Echo.java et EchoClient.java.
Echo.java :
Cf. Echo.java du serveur.

EchoClient.java :

```
import java.rmi.registry.*;

public class EchoClient {
	public static void main(String args[]) throws Exception {
		Registry registry = LocateRegistry.getRegistry(null, 1099);
		String url = "rmi://localhost/EchoServer";
		Echo obj = (Echo)registry.lookup(url);
		String msg = obj.echo(args[0]);
		System.out.println("Envoi: "+args[0]);
		System.out.println(msg);
	}
}
```

1.3.2 Description de la commande

```
java EchoClient hello
```

1.4 Présentation du registry

1.4.1 Description du code

Le code du serveur se compose de deux fichiers : Echo.java et RmiRegistry.java.
Echo.java :
Cf. Echo.java du serveur.

RmiRegistry.java :

```
import java.rmi.registry.LocateRegistry;

public class RmiRegistry {
	public static void main (String[] args) throws Exception{
		LocateRegistry.createRegistry(1099);
		System.out.println("RMI registry running on port 1099");
		Thread.sleep(Long.MAX_VALUE);
	}
}
```

1.4.2 Description de la commande

```
java RmiRegistry
```

2 Choix de la chaîne de certification

2.1 Vocabulaire

la relation "A -> B" signifie que les informations contenues dans le certificat A permettent de vérifier l'authenticité du certificat B.

2.2 Chaîne de certification 1

godard-tp.pem -> g1010419.tp.pem;

2.2.1 Méthode de vérification

```
cat godard-tp.pem > chain.pem;
openssl verify -CAfile chain.pem g1010419.tp.pem;
```

2.2.2 Résultat obtenu

```
g1010419.tp.pem: OK
```

2.3 Chaîne de certification 2

godard.ca.pem -> godard-tp.ca.pem -> g1010419.tp.pem;

2.3.1 Méthode de vérification

```
cat godard.ca.pem godard-tp.ca.pem > chain.pem;
openssl verify -CAfile chain.pem g1010419.tp.pem;
```

2.3.2 Résultat obtenu

```
g1010419.tp.pem:
C = FR, ST = PACA, L = Marseille, O = Internet Widgits Pty Ltd, CN = GUIEU-SERRE,
emailAddress = cyriaque.serre@etu.univ-amu.fr
error 20 at 0 depth lookup:unable to get local issuer certificate
```

2.3.3 Problème

La vérification a échoué car le certificat "godard-tp.ca.pem" ne permet pas de vérifier l'authenticité de notre certificat alors qu'il possède a priori les mêmes informations que "godard-tp.pem" si l'on exclut l'autorité de certification.

2.3.4 Recherche de la cause

2.3.4.1 Comparaison des clés publiques

Hypothèse:
la vérification a échoué car la clé publique contenue dans "godard-tp.ca.pem" n'est pas associée à la clé privée qui a signé "g1010419.tp.pem" alors que celle de "godard-tp.pem" l'est.

Méthode:

```
openssl x509 -in godard-tp.pem -pubkey |head -8 | tail -n +2 > godard-tp.ca.txt;
openssl x509 -in godard-tp.ca.pem -pubkey | head -8 | tail -n +2 > godard-tp.txt;
diff -s godard-tp.ca.txt godard-tp.txt;
```

Résultat obtenu:

```
Les fichiers godard-tp.ca.txt et godard-tp.txt sont identiques.
```

Commentaire:
Cette opération nous permet d'affirmer que les clefs publiques sont identiques. Donc le problème se situe ailleurs, nous allons comparer le contenu des deux certificats afin d'y repérer des différences.

2.3.4.2 Comparaison des certificats

Hypothèse:
Les informations contenues dans "godard-tp.ca.pem" ne permettent pas de vérifier "g1010419.tp.pem". Elles sont soit erronées soit manquantes.

Méthode:

```
openssl x509 -text -in godard-tp.pem | head -41 > godard-tp.txt;
openssl x509 -text -in godard-tp.ca.pem | head -41 > godard-tp.ca.txt;
diff godard-tp.txt godard-tp.ca.txt;
```

Résultat obtenu:

ligne 4
godard-tp.txt:
Serial Number: 14329070079058082229 (0xc6db155177ef79b5)
godard-tp.ca.txt:
Serial Number: 14329070079058082230 (0xc6db155177ef79b6)
ligne 6
godard-tp.txt:
Issuer: C=FR, ST=PACA, O=M2 FSI, OU=TP SIR, CN=godard TP
godard-tp.ca.txt:
Issuer: C=FR, ST=PACA, L=Marseille, O=M2 FSI, CN=godard
ligne 8 et 9
godard-tp.txt:
Not Before: Nov 27 14:18:06 2014 GMT
Not After : Nov 26 14:18:06 2017 GMT
godard-tp.ca.txt:
Not Before: Nov 27 14:18:18 2014 GMT
Not After : Nov 27 14:18:18 2015 GMT
ligne 10
godard-tp.txt:
Subject: C=FR, ST=PACA, O=M2 FSI, OU=TP SIR, CN=godard TP
godard-tp.ca.txt:
Subject: C=FR, ST=PACA, **L=Marseille**, O=M2 FSI, OU=TP SIR, CN=godard TP
ligne 38
godard-tp.txt:
keyid:76:2F:58:BA:4E:DD:48:D3:E7:F8:67:78:7D:61:2B:B0:B1:C5:C6:29
godard-tp.ca.txt:
keyid:4B:17:85:6F:55:5C:6D:DE:B5:39:51:56:7F:A9:59:8B:19:86:58:A2

Commentaire:
La différence trouvée à la ligne 10 pourrait expliquer l'échec de la vérification car l'émetteur du certificat "g1010419.tp.pem" est défini par "C=FR, ST=PACA, O=M2 FSI, OU=TP SIR, CN=godard TP". Et cela ne correspond pas à ce qui est renseigné dans le champ subject du certificat "godard-tp.ca.pem" (Cf. ligne10).

2.4 Conclusion

Comme on ne possède pas la clé privée associée au certificat "godard.ca.pem", on ne peut pas modifier le certificat défaillant sans devoir modifier la chaîne de certification afin de vérifier si cette différence en est la cause. Nous choisirons donc la première chaîne de certification pour la suite du projet.

3 Ajout de l'authentification du serveur

3.1 Vocabulaire

keystore: base de données qui contient la clé privée et le certificat associé ainsi que sa chaîne de certification s'il n'est pas autosigné. -> permet de s'authentifier.
truststore: base de données qui contient uniquement les certificats racines des autorités de certification c'est à dire les certificats autosignés. -> permet d'authentifier autrui.

3.2 Changement pour le serveur

3.2.1 Modification du code

EchoImpl.java:
on remplace

```
public EchoImpl() throws RemoteException {
  super();
}
```

par

```
public EchoImpl() throws RemoteException {
  super(0,new SslRMIClientSocketFactory(), new SslRMIServerSocketFactory());
}
```

3.2.2 Création d'un keystore

Objectif: créer un keystore au format JKS qui contient la clé privée et la chaîne de certification qu'on a fabriqué précedement avec openssl.

Problème: keytool ne permet pas l'importation ou l'exportation de clé privée au format JKS.

Solution: créer un keystore au format PKCS12 et importer la totalité des entrées dans notre keystore au format JKS.

Méthode de création d'un keystore au format PKCS12 contenant la clé privée et sa chaîne de certification:

```
cat privkey.pem g1010419.tp.pem godard-tp.pem > mykeycertificate.pem;
openssl pkcs12 -export -in mykeycertificate.pem -out serverKeystore.pkcs12 -name serverkey -noiter -nomaciter;

"Enter pass phrase for mykeycertificate.pem:",
-> phrase de passe qui protége notre clé privée.
"Enter Export Password:",
-> choix d'un mot de passe pour protéger la base de donnée au format PKCS12.
"Verifying - Enter Export Password:",
-> confirmation du mot de passe.
```

Méthode pour importer le contenu d'une base de donnée au format PKCS12 dans une base de donnée au format JKS:

```
keytool -importkeystore -srckeystore serverKeystore.pkcs12 -srcstoretype PKCS12 -keystore serverKeystore.jks;

"Entrez le mot de passe du fichier de clés de destination :",
-> choix d'un mot de passe pour protéger la base de données au format JKS.
"Ressaisissez le nouveau mot de passe :",
-> confirmation du mot de passe.
"Entrez le mot de passe du fichier de clés source :",
-> mot de passe de la base de donnée au format PKCS12 qui servira aussi à protéger cette entrée dans la base de données au format JKS.
L'entrée de l'alias serverkey a été importée.
Commande d'import exécutée : 1 entrées importées, échec ou annulation de 0 entrées
```

3.2.3 Modification de la commande

```
java \
-Djavax.net.ssl.keyStore=../data/serverKeystore.jks \
-Djavax.net.ssl.keyStorePassword=ilfaitbeau \
EchoImpl
```

Commentaire: le mot de passe de l'entrée dans la base de données jks doit être celui de la base de données sinon cette commande échouera car la clé ne pourra être récupérée.

3.3 Changement pour le client

3.3.1 Création du truststore

Objectif: créer un truststore au format jks qui contient l'autorité de certification racine de la chaîne de certification du serveur.

Méthode:

```
keytool -import -file godard-tp.pem -alias godardtp -keystore clientTruststore.jks

"Entrez le mot de passe du fichier de clés : ",
-> choix d'un mot de passe pour protéger la base de données.
"Ressaisissez le nouveau mot de passe :",
-> confirmation du mot de passe.
```

3.3.2 Modification de la commande

```
java \
-Djavax.net.ssl.trustStore=../data/clientTruststore.jks \
-Djavax.net.ssl.trustStorePassword=cocolasticot \
EchoClient hello
```

3.4 Changement pour le registry

3.4.1 Création du truststore

Objectif: créer un truststore au format jks qui contient l'autorité de certification racine de la chaîne de certification du serveur.

Méthode: (**Cf.** 3.3.1).

3.4.2 Modification de la commande

```
java \
-Djavax.net.ssl.trustStore=../data/registryTruststore.jks \
-Djavax.net.ssl.trustStorePassword=cocolasticot \
RmiRegistry
```

4 Ajout de l'authentification du client et du registry

4.1 Changement pour le serveur

4.1.1 Modification du code

EchoImpl.java:
on remplace

```
public EchoImpl() throws RemoteException {
  super(0,new SslRMIClientSocketFactory(), new SslRMIServerSocketFactory());
}
```

par

```
public EchoImpl() throws RemoteException {
  super(0,new SslRMIClientSocketFactory(),
            newSslRMIServerSocketFactory(null,null,true));
}
```

Commentaire: par défaut en java 7,la version du protocole est tlsv1 et la suite cryptographique négociée est TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA. Cette dernière est la meilleure suite dont on dispose utilisant notre type de certificats (clé publique: RSA).

4.1.2 Création du truststore

Objectif: créer un truststore au format JKS qui contient l'autorité de certification racine de la chaîne de certification du client et du registry.

Méthode: (**Cf.** 3.3.1).

4.1.3 Modification de la commande

```
java \
-Djavax.net.ssl.keyStore=../data/serverKeystore.jks \
-Djavax.net.ssl.keyStorePassword=ilfaitbeau \
-Djavax.net.ssl.trustStore=../data/serverTruststore.jks \
-Djavax.net.ssl.trustStorePassword=cocolasticot \
EchoImpl
```

4.2 Changement pour le client

4.2.1 Création d'un keystore

Objectif: créer un keystore au format JKS qui contient sa clé privée et sa chaîne de certification associée.

Méthode:

```
keytool -genkey -keyalg RSA -keysize 2048 -alias clientKey \
-keystore clientKeystore.jks

Entrez le mot de passe du fichier de clés :
Ressaisissez le nouveau mot de passe :
Quels sont vos nom et prénom ?
  [Unknown]:  client
Quel est le nom de votre unité organisationnelle ?
  [Unknown]:  m2 fsi
Quel est le nom de votre entreprise ?
  [Unknown]:  amu
Quel est le nom de votre ville de résidence ?
  [Unknown]:  marseille
Quel est le nom de votre état ou province ?
  [Unknown]:  paca
Quel est le code pays à deux lettres pour cette unité ?
  [Unknown]:  fr
Est-ce CN=client, OU=m2 fsi, O=amu, L=marseille, ST=paca, C=fr ?
  [non]:  oui

Entrez le mot de passe de la clé pour <clientKey>
        (appuyez sur Entrée s'il s'agit du mot de passe du fichier de clés) :
```

4.2.2 Exportation du certificat autosigné au format PEM

Méthode:

```
keytool -export -rfc -alias clientKey -file clientCert.pem \
-keystore clientKeystore.jks
```

Attention: la commande retourne le premier certificat de la chaîne de certification associée qui est toujours le certificat contenant la clé publique associé à la clé privée de l'entrée. Donc, si la longueur de la chaîne de certification associé à l'entrée est supérieur à 1, on ne peut pas récupérer le certificat de l'autorité racine avec cette commande.

4.2.3 Modification de la commande

```
java \
-Djavax.net.ssl.keyStore=../data/clientKeystore.jks \
-Djavax.net.ssl.keyStorePassword=ilfaitbeau \
-Djavax.net.ssl.trustStore=../data/clientTruststore.jks \
-Djavax.net.ssl.trustStorePassword=cocolasticot \
EchoClient hello
```

4.3 Changement pour le registry

4.3.1 Création d'un keystore

Objectif: créer un keystore au format JKS qui contient sa clé privée et sa chaîne de certification associée.

Méthode: (**Cf.** 4.2.1).

4.3.2 Exportation du certificat autosigné au format PEM

(**Cf.** 4.2.2).

4.3.3 Modification de la commande

```
java \
-Djavax.net.ssl.keyStore=../data/registryKeystore.jks \
-Djavax.net.ssl.keyStorePassword=ilfaitbeau \
-Djavax.net.ssl.trustStore=../data/registryTruststore.jks \
-Djavax.net.ssl.trustStorePassword=cocolasticot \
RmiRegistry
```

5 Ajout de la protection du registry

5.1 Changement pour le serveur

5.1.1 Modification du code

EchoImpl.java:
on remplace

```
public static void main (String args[]) throws Exception {
  ...
  Registry registry = LocateRegistry.getRegistry(null, 1099);
  ...
}
```

par

```
public static void main (String args[]) throws Exception {
  ...
  Registry registry = LocateRegistry.getRegistry(null, 1099,
          new SslRMIClientSocketFactory());
  ...
}
```

5.2 Changement pour le client

5.2.1 Modification du code

EchoClient.java:
on remplace

```
public static void main (String args[]) throws Exception {
  ...
  Registry registry = LocateRegistry.getRegistry(null, 1099);
  ...
}
```

par

```
public static void main (String args[]) throws Exception {
  ...
  Registry registry = LocateRegistry.getRegistry(null, 1099,
                  new SslRMIClientSocketFactory());
  ...
}
```

5.2.2 Ajout du certificat du registry dans le truststore

Méthode: (**Cf.** 3.3.1).

5.3 Changement pour le registry

5.3.1 Modification du code:

RmiRegistry.java:
on remplace

```
public static void main (String args[]) throws Exception {
  ...
  LocateRegistry.createRegistry(1099);
  ...
}
```

par

```
public static void main (String args[]) throws Exception {
  ...
  LocateRegistry.createRegistry(1099, new SslRMIClientSocketFactory(),
          new SslRMIServerSocketFactory(null, null, true));
  ...
}
```

5.3.2 Ajout du certificat du client dans le truststore

Méthode: (**Cf.** 3.3.1).

6 Utilisation de TLS 1.2

6.1 Changement pour le serveur

6.1.1 Modification du code

EchoImpl.java:
on remplace

```
public EchoImpl() throws RemoteException {
  super(0,new SslRMIClientSocketFactory(),
        new SslRMIServerSocketFactory(null,null,true));
}
```

par

```
public EchoImpl() throws RemoteException {
  super(0,new SslRMIClientSocketFactory(),
        new SslRMIServerSocketFactory(
              null,
              new String[] {"TLSv1.2"},
              true));
}
```

6.1.2 Modification de la commande

```
java \
-Djavax.net.ssl.keyStore=../data/serverKeystore.jks \
-Djavax.net.ssl.keyStorePassword=ilfaitbeau \
-Djavax.net.ssl.trustStore=../data/serverTruststore.jks \
-Djavax.net.ssl.trustStorePassword=cocolasticot \
-Djavax.rmi.ssl.client.enabledProtocols=TLSv1.2 \
EchoImpl
```

6.2 Changement pour le client

6.2.1 Modification de la commande

```
java \
-Djavax.net.ssl.keyStore=../data/clientKeystore.jks \
-Djavax.net.ssl.keyStorePassword=ilfaitbeau \
-Djavax.net.ssl.trustStore=../data/clientTruststore.jks \
-Djavax.net.ssl.trustStorePassword=cocolasticot \
-Djavax.rmi.ssl.client.enabledProtocols=TLSv1.2 \
EchoClient hello
```

6.3 Changement pour le registry

6.3.1 Modification du code

RmiRegistry.java:
on remplace

```
public static void main (String args[]) throws Exception {
  ...
  LocateRegistry.createRegistry( 1099, new SslRMIClientSocketFactory(),
        new SslRMIServerSocketFactory( null, null, true));
  ...
}
```

par

```
public static void main (String args[]) throws Exception {
  ...
  LocateRegistry.createRegistry( 1099, new SslRMIClientSocketFactory(),
        new SslRMIServerSocketFactory( null, new String[] {"TLSv1.2"}, true));
  ...
}
```

6.3.2 modification de la commande

```
java \
-Djavax.net.ssl.keyStore=../data/registryKeystore.jks \
-Djavax.net.ssl.keyStorePassword=ilfaitbeau \
-Djavax.net.ssl.trustStore=../data/registryTruststore.jks \
-Djavax.net.ssl.trustStorePassword=cocolasticot \
-Djavax.rmi.ssl.client.enabledProtocols=TLSv1.2 \
RmiRegistry
```

7 Accès à des suites cryptographiques plus robustes

7.1 Objectif

Accéder à des suites cryptographiques utilisant des tailles de clés plus grandes pour leur algorithme de chiffrement symétrique. comme par exemple, la suite TLS_ECDHE_RSA_ WITH_AES_256_CBC_SHA afin d'améliorer la confidentialité des données échangées.

7.2 Problème

par défaut,les suites cryptographique proposées sont

```
[TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
TLS_RSA_WITH_AES_128_CBC_SHA,
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA,
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA,
TLS_DHE_RSA_WITH_AES_128_CBC_SHA,
TLS_DHE_DSS_WITH_AES_128_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA,
TLS_ECDHE_RSA_WITH_RC4_128_SHA, SSL_RSA_WITH_RC4_128_SHA,
TLS_ECDH_ECDSA_WITH_RC4_128_SHA, TLS_ECDH_RSA_WITH_RC4_128_SHA,
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,
SSL_RSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA,
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA,
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA, SSL_RSA_WITH_RC4_128_MD5,
TLS_EMPTY_RENEGOTIATION_INFO_SCSV]
```

donc le mieux qu'on puisse négocier avec notre type de certificat
est TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA.

7.3 Cause

java limite la taille des clés de certains algorithme de chiffrement comme on peut le constater à la lecture du fichier "default_local.policy" qui se trouve dans l'archive "jdk1.7.0_71/jre/lib/security/local_policy.jar" et dont voici le contenu:

```
grant {
   permission javax.crypto.CryptoPermission "DES", 64;
   permission javax.crypto.CryptoPermission "DESede", *;
   permission javax.crypto.CryptoPermission "RC2", 128,
                              "javax.crypto.spec.RC2ParameterSpec", 128;
   permission javax.crypto.CryptoPermission "RC4", 128;
   permission javax.crypto.CryptoPermission "RC5", 128,
        "javax.crypto.spec.RC5ParameterSpec", *, 12, *;
   permission javax.crypto.CryptoPermission "RSA", *;
   permission javax.crypto.CryptoPermission *, 128;
};
```

7.4 Solution

Modifier la politique de sécurité de java concernant la cryptographie.

7.5 Méthode

Télécharger l'archive "UnlimitedJCEPolicyJDK7.zip" dans le répertoire "jdk1.7.0_71/jre/lib/security/"
Lien: http://www.oracle.com/technetwork/java/javase/downloads/jce-7-download-432124.html
Dans le répertoire "jdk1.7.0_71/jre/lib/security/" faire :

```
unzip "UnlimitedJCEPolicyJDK7.zip";
mv local_policy.jar local_policy.bak.jar;
cp UnlimitedJCEPolicy/local_policy.jar ./;
rm -r UnlimitedJCEPolicy.zip UnlimitedJCEPolicy/;
```

Maintenant la taille des clés n'est plus limité car la politique de sécurité de java pour la cryptographie donne maintenant tous les droits:

```
grant {
   // There is no restriction to any algorithms.
   permission javax.crypto.CryptoAllPermission;
};
```

7.6 Résultat

Voici les suites cryptographiques qu'on nous propose par défaut:

```
[TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA,
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA,
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA,
TLS_DHE_RSA_WITH_AES_256_CBC_SHA,
TLS_DHE_DSS_WITH_AES_256_CBC_SHA,
…,
TLS_EMPTY_RENEGOTIATION_INFO_SCSV]
```

Nous pouvons désormais négocier des suites cryptographiques à base d'AES-256 qui ont l'avantage d'être plus difficile à casser.

la liste des suites cryptographiques supportés par java 7:
http://docs.oracle.com/javase/7/docs/technotes/guides/security/SunProviders.html
à "The SunJSSE Provider/Cipher Suites".

8 Utilisation de la suite : TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384

8.1 Changement pour le serveur

8.1.1 Modification du code:

EchoImpl.java:
on remplace

```
public EchoImpl() throws RemoteException {
  super(0,new SslRMIClientSocketFactory(),
        new SslRMIServerSocketFactory(
              null,
              new String[] {"TLSv1.2"},
              true));
}
```

par

```
public EchoImpl() throws RemoteException {
  super(0,new SslRMIClientSocketFactory(),
        new SslRMIServerSocketFactory(
              new String[] {"TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384"},
              new String[] {"TLSv1.2"},
              true));
}
```

8.1.2 Modification de la commande

```
java \
-Djavax.net.ssl.keyStore=../data/serverKeystore.jks \
-Djavax.net.ssl.keyStorePassword=ilfaitbeau \
-Djavax.net.ssl.trustStore=../data/serverTruststore.jks \
-Djavax.net.ssl.trustStorePassword=cocolasticot \
-Djavax.rmi.ssl.client.enabledProtocols=TLSv1.2 \
-Djavax.rmi.ssl.client.\
enabledCipherSuites=TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 \
EchoImpl
```

8.2 Changement pour le client

8.2.1 Modification de la commande

```
java \
-Djavax.net.ssl.keyStore=../data/clientKeystore.jks \
-Djavax.net.ssl.keyStorePassword=ilfaitbeau \
-Djavax.net.ssl.trustStore=../data/clientTruststore.jks \
-Djavax.net.ssl.trustStorePassword=cocolasticot \
-Djavax.rmi.ssl.client.enabledProtocols=TLSv1.2 \
-Djavax.rmi.ssl.client.\
enabledCipherSuites=TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 \
EchoClient hello
```

8.3 Changement pour le registry

8.3.1 Modification du code

RmiRegistry.java:
on remplace

```
public static void main (String args[]) throws Exception {
  ...
  LocateRegistry.createRegistry(
                1099,
                new SslRMIClientSocketFactory(),
                new SslRMIServerSocketFactory(
                    null,
                    new String[] {"TLSv1.2"},
                    true));
  ...
}
```

par

```
public static void main (String args[]) throws Exception {
  ...
  LocateRegistry.createRegistry(
                1099,
                new SslRMIClientSocketFactory(),
                new SslRMIServerSocketFactory(
                    new String[] {"TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384",
                    new String[] {"TLSv1.2"},
                    true));
  ...
}
```

8.3.2 Modification de la commande:

```
java \
-Djavax.net.ssl.keyStore=../data/registryKeystore.jks \
-Djavax.net.ssl.keyStorePassword=ilfaitbeau \
-Djavax.net.ssl.trustStore=../data/registryTruststore.jks \
-Djavax.net.ssl.trustStorePassword=cocolasticot \
-Djavax.rmi.ssl.client.enabledProtocols=TLSv1.2 \
-Djavax.rmi.ssl.client.\
enabledCipherSuites=TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 \
RmiRegistry
```

9 Version finale

Afin de simplifier le lancement de chaque logiciel les options de lancement ont été déplacées à l'intérieur du code.

9.1 Changement pour le serveur

9.1.1 Modification du code

EchoImpl.java:
on ajoute le code suivant :

```
public static void main (String args[]) throws Exception {
	String keyPass= null;
	String trustPass= null;

	if(args.length != 2){
		System.out.println(
			"Usage :./RmiRegistry keyStorePasswordtrustStorePassword");
		System.exit(-1);
	}

	keyPass = args[0];
	trustPass = args[1];

	System.setProperty("javax.net.ssl.keyStore", "data/serverKeystore.jks");
	System.setProperty("javax.net.ssl.keyStorePassword", keyPass);

	System.setProperty("javax.net.ssl.trustStore",
		"data/serverTruststore.jks");
	System.setProperty("javax.net.ssl.trustStorePassword", trustPass);

	System.setProperty("javax.rmi.ssl.client.enabledProtocols", "TLSv1.2");
	System.setProperty("javax.rmi.ssl.client.enabledCipherSuites",
		"TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384");
	…
}
```

9.1.2 Modification de la commande

```
java -jar server.jar ilfaitbeau cocolasticot
```

9.2 Changement pour le client

9.2.1 Modification du code

EchoClient.java:
on ajoute le code suivant :

```
public static void main(String args[]) throws Exception {

	String keyPass= null;
	String trustPass= null;
	String msg= null;

	if(args.length != 3){
		System.out.println("Usage:EchoClient keyStorePassword
			trustStorePassword Message");
		System.exit(-1);
	}

	keyPass = args[0];
	trustPass = args[1];

	System.setProperty("javax.net.ssl.keyStore", "data/clientKeystore.jks");
	System.setProperty("javax.net.ssl.keyStorePassword", keyPass);

	System.setProperty("javax.net.ssl.trustStore", "data/clientTruststore.jks");
	System.setProperty("javax.net.ssl.trustStorePassword", trustPass);

	System.setProperty("javax.rmi.ssl.client.enabledProtocols", "TLSv1.2");
	System.setProperty("javax.rmi.ssl.client.enabledCipherSuites",
		"TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384");
	…
}
```

9.2.2 Modification de la commande

```
java -jar client.jar ilfaitbeau cocolasticot hello
```

9.3 Changement pour le client

9.3.1 Modification du code

RmiRegistry.java:
on ajoute le code suivant :

```
public static void main (String args[]) throws Exception {
        String keyPass= null;
        String trustPass= null;

        if(args.length != 2){
                System.out.println(
                        "Usage :./RmiRegistry keyStorePasswordtrustStorePassword");
                System.exit(-1);
        }

        keyPass = args[0];
        trustPass = args[1];

        System.setProperty("javax.net.ssl.keyStore", "data/serverKeystore.jks");
        System.setProperty("javax.net.ssl.keyStorePassword", keyPass);

        System.setProperty("javax.net.ssl.trustStore",
                "data/serverTruststore.jks");
        System.setProperty("javax.net.ssl.trustStorePassword", trustPass);

        System.setProperty("javax.rmi.ssl.client.enabledProtocols", "TLSv1.2");
        System.setProperty("javax.rmi.ssl.client.enabledCipherSuites",
                "TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384");
        …
}
```

9.3.2 Modification de la commande

```
java -jar registry.jar ilfaitbeau cocolasticot
```

10 Résumé

Notre architecture rmi utilise désormais le protocole TLS 1.2 avec la suite cryptographique TLS_ ECDHE_RSA_WITH_AES_256_CBC_SHA384 qui représente la meilleur suite qu'on peut négocier avec notre type de certificat dans la version 7 de java. En effet, dans java 8 nous pouvons accéder à des suites aussi robustes qui utilisent GCM(Galois Counter Mode) à la place de CBC(Cypher Block Chaining) dont la propriété intéressante est de fournir une confidentialité future parfaite.