



Filière : Master Systèmes Distribués et Intelligence  
Artificielle (SDIA)

---

## Sécurité des systèmes multi-agents

---

Préparé par :

BOUZYAN WAHIBA

## Table des matières

<b>INTRODUCTION</b> .....	<b>2</b>
<b>PARTIE 1 : RSA</b> .....	<b>3</b>
Classe GenerateRSAKey .....	3
Classe CryptoGraphUtils .....	4
Classe MainContainer.....	4
Classe SimpleClientContainer.....	4
Classe SimpleServerContainer.....	5
Classe ClientAgent.....	6
Classe ServerAgent.....	6
Exemple d'exécution .....	8
<b>PARTIE 1 : RSA</b> .....	<b>9</b>
Classe SimpleServerContainer .....	9
Classe SimpleClientContainer.....	9
Classe ServerAgent.....	10
Classe ClientAgent.....	11
Exemples d'exécution .....	12
<b>Conclusion</b> .....	<b>13</b>

## **I. INTRODUCTION**

AES sont deux algorithmes de chiffrement utilisés pour sécuriser les communications et protéger les données sensibles. Ils sont largement utilisés dans les systèmes de sécurité informatique pour garantir la confidentialité et l'intégrité des messages échangés entre des agents ou des entités.

RSA, qui signifie Rivest, Shamir, et Adleman, est un algorithme de chiffrement asymétrique, ce qui signifie qu'il utilise une paire de clés distinctes pour chiffrer et déchiffrer les données. Une clé est utilisée pour le chiffrement, tandis que l'autre clé correspondante est utilisée pour le déchiffrement. La clé de chiffrement peut être rendue publique, tandis que la clé de déchiffrement doit rester privée. RSA est souvent utilisé pour échanger les clés de session dans les protocoles de sécurité en ligne, car il offre une méthode sécurisée pour établir des clés de chiffrement symétriques pour une communication sécurisée.

En revanche, AES, qui signifie Advanced Encryption Standard, est un algorithme de chiffrement symétrique, ce qui signifie qu'il utilise une seule clé pour chiffrer et déchiffrer les données. Cette clé est partagée entre les deux agents qui communiquent et doit rester confidentielle pour garantir la sécurité des messages échangés. AES est considéré comme l'un des algorithmes de chiffrement les plus robustes et les plus largement utilisés dans le monde pour sécuriser les données sensibles, et il est souvent utilisé pour chiffrer les données de manière efficace et rapide.

L'utilisation de RSA et AES ensemble peut offrir un niveau de sécurité élevé dans les communications en ligne. Par exemple, RSA peut être utilisé pour échanger les clés de session AES, qui sont ensuite utilisées pour chiffrer et déchiffrer les messages entre les agents de manière efficace et sécurisée. Cette combinaison de chiffrement asymétrique et symétrique permet de garantir la confidentialité, l'intégrité et l'authenticité des messages échangés entre les agents, offrant ainsi une communication sécurisée dans les systèmes de sécurité informatique.

## II. PARTIE 1 : RSA

### 1- Classe GenerateRSAKey:

Cette classe contient une méthode statique "main()" pour générer une paire de clés RSA, puis affiche les clés sous forme encodée en base64. Au début, On va faire appel à la méthode statique generateRSAKeys() de la classe CryptographUtils pour générer une paire de clés RSA, et stocke la paire de clés générée dans un objet KeyPair appelé keyPair. En suite , on récupère la clé privée de la paire de clés générée à partir de l'objet KeyPair et la stocke dans un objet PrivateKey appelé privateKey. Et on récupère en plus la clé publique de la paire de clés générée à partir de l'objet KeyPair et la stocke dans un objet PublicKey appelé publicKey et encode la clé privée en base64 en utilisant la classe Base64 et la méthode encodeToString() de l'encodeur de base64, puis stocke le résultat dans une chaîne de caractères appelée encodedPK puis Encoder la clé publique en base64 de la même manière que la clé privée, puis stocker le résultat dans une chaîne de caractères appelée encodedPubK et en fin on affiche les clés privée et public.

```
package ma.enset.rsa;
import java.security.KeyPair;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.util.Base64;

public class GenerateRSAKey {
    public static void main(String[] args) throws NoSuchAlgorithmException
    {
        KeyPair keyPair = CryptographUtils.generateRSAKeys();
        PrivateKey privateKey = keyPair.getPrivate();
        PublicKey publicKey = keyPair.getPublic();
        String encodedPK =
Base64.getEncoder().encodeToString(privateKey.getEncoded());
        String encodedPubK =
Base64.getEncoder().encodeToString(publicKey.getEncoded());
        System.out.println("===Private Key===");
        System.out.println(encodedPK);
        System.out.println("===Public key===");
        System.out.println(encodedPubK);
    }
}
```

### 2- Classe CryptographUtils :

Cette classe contient une méthode statique "generateRSAKeys()" pour générer une paire de clés RSA. Alors, au début on crée un nouvel objet KeyPairGenerator en utilisant l'algorithme de chiffrement RSA. L'algorithme "RSA" est passé en paramètre à la méthode getInstance() pour spécifier que nous voulons générer une paire de clés RSA. Puis, on Initialise l'objet KeyPairGenerator avec une taille de clé de 512 bits. Cela signifie que les clés générées auront une longueur de 512 bits et on va Générer une paire de clés RSA à l'aide de l'objet

KeyPairGenerator initialisé précédemment et stocke la paire de clés générée dans un objet KeyPair appelé keyPair. La méthode va Retourner l'objet KeyPair contenant la paire de clés générée.

```
package ma.enset.rsa;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
public class CryptographUtils {
    public static KeyPair generateRSAKeys() throws NoSuchAlgorithmException
    {
        KeyPairGenerator keyPairGenerator =
KeyPairGenerator.getInstance("RSA");
        keyPairGenerator.initialize(512);
        KeyPair keyPair = keyPairGenerator.generateKeyPair();
        return keyPair; }
}
```

### 3- Classe MainContainer :

Le bloc du code ci-dessous représente un exemple de démarrage d'un conteneur principal (Main Container) dans un environnement JADE (Java Agent Development Framework) pour exécuter des agents Jade dans une plateforme multi-agents.

```
package ma.enset.rsa;

import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.wrapper.AgentContainer;
import jade.wrapper.ControllerException;

public class MainContainer {
    public static void main(String[] args) throws ControllerException {
        Runtime runtime = Runtime.instance();
        ProfileImpl profile = new ProfileImpl();
        profile.setParameter(ProfileImpl.GUI , "true");
        AgentContainer mainContainer =
runtime.createMainContainer(profile);
        mainContainer.start();

    }
}
```

### 4- Classe SimpleClientContainer :

Le code représente un exemple de démarrage d'un conteneur d'agents Jade pour exécuter un agent client dans un environnement RSA (système de cryptographie à clé publique).

```
package ma.enset.rsa;

import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.wrapper.AgentContainer;
import jade.wrapper.AgentController;
```

```

public class SimpleClientContainer {
    public static void main(String[] args) throws Exception {
        Runtime runtime = Runtime.instance();
        ProfileImpl profile = new ProfileImpl();
        profile.setParameter(ProfileImpl.MAIN_HOST, "localhost");
        AgentContainer clientContainer =
runtime.createAgentContainer(profile);
        String encodedPubKey =
"MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAAKkWsAIC8pBY811dP2ImA48QK2EeLr0/LQHZb6h4/
Iinh5+PmFI/vAgGd340IsSSIQkEoDiBbQKlP7sMFNmuTMCAwEAAQ==";
        AgentController agentController =
clientContainer.createNewAgent("client", "ma.enset.aes.ClientAgent", new
Object[]{encodedPubKey});
        agentController.start();
    }
}

```

### 5- Classe SimpleServerContainer :

Le code représente un exemple de démarrage d'un conteneur d'agents Jade pour exécuter un agent serveur dans un environnement RSA (système de cryptographie à clé publique).

```

package ma.enset.rsa;

import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.wrapper.AgentContainer;
import jade.wrapper.AgentController;
public class SimpleServerContainer {
    public static void main(String[] args) throws Exception {
        Runtime runtime = Runtime.instance();
        ProfileImpl profile = new ProfileImpl();
        profile.setParameter(ProfileImpl.MAIN_HOST, "localhost");
        AgentContainer clientContainer =
runtime.createAgentContainer(profile);
        String encodedPK =
"MIIBVgIBADANBgkqhkiG9w0BAQEFAASCauAwggE8AgEAAkEAopawAhwjykFjzXV0/YiYDjxArY
R4uvT8tAdlvqHj8iKeHn4+YUj+8CAZ3fjQixJIhCQSgOIFtAqU/uwwU2a5MwIDAQABAKEAgu4mP
wy6JZ4S6Nm1pJXjHVOYIiOTtbqj5bYvURY9m7bEPKMyidjpES7iyrBtK9fUoYZYCWuGJI+cvieY
FTQ6gQIhAP6lr4bReNxV2aK2FUXS921fTLlCa8LNUJX/ICmp5QApAiEAo3PN5/dyIlIi0pzfMkK
7mhBChZmPXuDIMP3V9JMxKfsCIQDUUWsnmrIN7Fq4yt6zyDhXnskG0nWtB//2uLCgAQdZ+QIgOr
y/zs82Pa0khGkJMCEZAQk8zoPbaK4AYtKlE2ndrgcCIQDj/VbTQMnrftbN3ssbQUuw1U51RPFwi
rQDfuVjdXTdcw=";
        AgentController agentController =
clientContainer.createNewAgent("server", "ma.enset.aes.ServerAgent", new
Object[]{encodedPK });
        agentController.start();
    }
}

```

## 6- Classe ClientAgent :

Le code ci-dessous représente un agent client dans un système multi-agents. Cet agent est responsable d'envoyer un message chiffré à un agent serveur.

Il récupère la clé publique encodée en base64, chiffre le message "Hello Server!!!" en utilisant cette clé, encode le message chiffré en base64, crée un message ACL avec le message chiffré comme contenu, et l'envoie au serveur en utilisant la méthode send(). Les octets du message chiffré et la chaîne de caractères encodée en base64 du message chiffré sont également imprimés à des fins de débogage.

```
package ma.enset.rsa;

import jade.core.AID;
import jade.core.Agent;
import jade.lang.acl.ACLMessage;
import javax.crypto.Cipher;
import java.security.KeyFactory;
import java.security.PublicKey;
import java.security.spec.X509EncodedKeySpec;
import java.util.Arrays;
import java.util.Base64;

public class ClientAgent extends Agent {
    @Override
    protected void setup() {
        String encodedPubKey = (String) getArguments()[0];
        String message = "Hello Server!!!";
        try {
            byte[] encodedPBK = Base64.getDecoder().decode(encodedPubKey);
            KeyFactory keyFactory = KeyFactory.getInstance("RSA");
            PublicKey publicKey = keyFactory.generatePublic(new
X509EncodedKeySpec(encodedPBK));
            Cipher cipher = Cipher.getInstance("RSA");
            cipher.init(Cipher.ENCRYPT_MODE, publicKey);
            byte[] cryptMsg = cipher.doFinal(message.getBytes());
            String cryptEncodedMsg =
Base64.getEncoder().encodeToString(cryptMsg);
            ACLMessage aclMessage = new ACLMessage(ACLMessage.INFORM);
            aclMessage.addReceiver(new AID("server", AID.ISLOCALNAME));
            aclMessage.setContent(cryptEncodedMsg);
            send(aclMessage);
            System.out.println(Arrays.toString(cryptMsg));
            System.out.println(cryptEncodedMsg);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

## 7- Classe ServerAgent :

Le code représente un agent serveur pour recevoir et décrypter un message chiffré envoyé par l'agent client. L'agent serveur récupère la clé privée encodée en base64, crée un

comportement cyclique qui attend la réception d'un message ACL, extrait le message chiffré du contenu du message reçu, décode la chaîne de caractères encodée en base64 du message chiffré en octets, déchiffre le message chiffré en utilisant la clé privée, et imprime le message déchiffré. Si aucun message n'est reçu, le comportement cyclique bloque l'agent serveur en attendant de recevoir un message.

```
package ma.enset.rsa;

import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;

import javax.crypto.Cipher;
import java.security.KeyFactory;
import java.security.PrivateKey;
import java.security.spec.PKCS8EncodedKeySpec;
import java.util.Base64;

public class ServerAgent extends Agent {
    @Override
    protected void setup() {
        String encodedPK = (String) getArguments()[0];
        addBehaviour(new CyclicBehaviour() {
            @Override
            public void action() {
                ACLMessage receive = receive();
                if (receive != null) {
                    String cryptMsg = receive.getContent();
                    byte[] cryptMsg = Base64.getDecoder().decode(cryptMsg);
                    try {
                        byte[] encodedPrivateKey = Base64.getDecoder().decode(encodedPK);
                        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
                        PrivateKey privateKey = keyFactory.generatePrivate(new PKCS8EncodedKeySpec(encodedPrivateKey));
                        Cipher cipher = Cipher.getInstance("RSA");
                        cipher.init(Cipher.DECRYPT_MODE, privateKey);
                        byte[] decryptMsg = cipher.doFinal(cryptMsg);
                        System.out.println(new String(decryptMsg));
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                } else {
                    block();
                }
            }
        });
    }
}
```

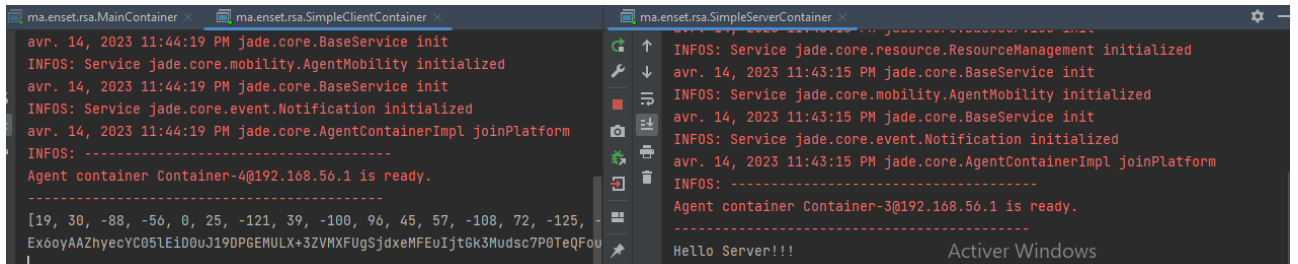


## 8- Exemples d'exécution :

Voici les clés générés :

```
"C:\Program Files\Java\jdk1.8.0_211\bin\java.exe" ...  
===Private Key===  
MIIBVAIBADANBgkqhkiG9w0BAQEFAASCAT4wggE6AgEAAkEAiP/LXkvd2gowJv3iwutZMYJ00R9k5Cj0o7vy7+QU4XUY/5g2EHvP3YAGs1C  
===Public key===  
MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAlj/y15L3doKMcB94sLrWT6CdNEfZ0Qo9K078u/kF0F16P+YNhB7z92ABrNRgx1+5Xff9fZmhBE  
|  
Process finished with exit code 0
```

Comme nous pouvons voir dans la capture ci-dessous, le serveur a bien reçu le message et il a réussi à le décrypté en utilisant **RSA** et l'afficher.



The screenshot displays three console windows from an IDE, showing the logs of a client-server application. The windows are titled 'ma.enset.rsa.MainContainer', 'ma.enset.rsa.SimpleClientContainer', and 'ma.enset.rsa.SimpleServerContainer'. The logs show the initialization of Jade services (BaseService, AgentMobility, Notification) and the joining of the platform. The client container sends a message 'Hello Server!!!' to the server container, which successfully receives and displays it. The server container also shows the message 'Hello Server!!!' and the text 'Activer Windows'.

```
ma.enset.rsa.MainContainer  
avr. 14, 2023 11:44:19 PM jade.core.BaseService init  
INFOS: Service jade.core.mobility.AgentMobility initialized  
avr. 14, 2023 11:44:19 PM jade.core.BaseService init  
INFOS: Service jade.core.event.Notification initialized  
avr. 14, 2023 11:44:19 PM jade.core.AgentContainerImpl joinPlatform  
INFOS: -----  
Agent container Container-4@192.168.56.1 is ready.  
-----  
[19, 30, -88, -56, 0, 25, -121, 39, -100, 96, 45, 57, -108, 72, -125,  
Ex6oyAAZhyecYC051EiD0uJ19DPGEMULX+3ZVMXFUgSjdxMFEuIjtGk3Mudsc7P0TeQFou  
|  
ma.enset.rsa.SimpleClientContainer  
avr. 14, 2023 11:44:19 PM jade.core.BaseService init  
INFOS: Service jade.core.mobility.AgentMobility initialized  
avr. 14, 2023 11:44:19 PM jade.core.BaseService init  
INFOS: Service jade.core.event.Notification initialized  
avr. 14, 2023 11:44:19 PM jade.core.AgentContainerImpl joinPlatform  
INFOS: -----  
Agent container Container-4@192.168.56.1 is ready.  
-----  
[19, 30, -88, -56, 0, 25, -121, 39, -100, 96, 45, 57, -108, 72, -125,  
Ex6oyAAZhyecYC051EiD0uJ19DPGEMULX+3ZVMXFUgSjdxMFEuIjtGk3Mudsc7P0TeQFou  
|  
ma.enset.rsa.SimpleServerContainer  
avr. 14, 2023 11:43:15 PM jade.core.BaseService init  
INFOS: Service jade.core.resource.ResourceManagement initialized  
avr. 14, 2023 11:43:15 PM jade.core.BaseService init  
INFOS: Service jade.core.mobility.AgentMobility initialized  
avr. 14, 2023 11:43:15 PM jade.core.BaseService init  
INFOS: Service jade.core.event.Notification initialized  
avr. 14, 2023 11:43:15 PM jade.core.AgentContainerImpl joinPlatform  
INFOS: -----  
Agent container Container-3@192.168.56.1 is ready.  
-----  
Hello Server!!!  
Activer Windows
```

### III. PARTIE 2 : AES

#### 1- Classe SimpleServerContainer :

Le code qu'on voit ci-dessous représente un conteneur d'agents Jade pour exécuter un agent serveur AES (Advanced Encryption Standard). Le conteneur est créé en utilisant le runtime Jade, avec un profil d'hôte principal défini sur "localhost". Un agent serveur est ensuite créé en utilisant la classe "ServerAgent" dans le package "ma.enset.aes", avec un argument supplémentaire de clé secrète "secret" passée au constructeur de l'agent. Enfin, l'agent serveur est démarré en appelant la méthode "start()" sur l'objet AgentController.

```
package ma.enset.aes;

import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.wrapper.AgentContainer;
import jade.wrapper.AgentController;

public class SimpleServerContainer {
    public static void main(String[] args) throws Exception {
        Runtime runtime = Runtime.instance();
        ProfileImpl profile = new ProfileImpl();
        profile.setParameter(ProfileImpl.MAIN_HOST, "localhost");
        AgentContainer clientContainer =
runtime.createAgentContainer(profile);
        String secret = "1234561234560000";
        AgentController agentController =
clientContainer.createNewAgent("server", "ma.enset.aes.ServerAgent", new
Object[]{secret });
        agentController.start();
    }
}
```

#### 2- Classe SimpleClientContainer :

De la même manière on a créé un conteneur d'agents Jade pour exécuter un agent client AES. Nous avons utilisé la même clé secrète.

```
package ma.enset.aes;

import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.wrapper.AgentContainer;
import jade.wrapper.AgentController;
public class SimpleClientContainer {
    public static void main(String[] args) throws Exception {
        Runtime runtime = Runtime.instance();
        ProfileImpl profile = new ProfileImpl();
        profile.setParameter(ProfileImpl.MAIN_HOST, "localhost");
```

```

        AgentContainer clientContainer =
runtime.createAgentContainer(profile);
        String secret = "1234561234560000";
        AgentController agentController =
clientContainer.createNewAgent("client", "ma.enset.aes.ClientAgent", new
Object[]{secret});
        agentController.start();

    }
}

```

### 3- Class ServerAgent :

On parle du même principe que la classe précédente ou nous avons utilisé 'RSA' mais dans ce cas on déchiffre le message en utilisant une clé secrète AES définie par la chaîne de caractères "secret" passée en argument lors de la création de l'agent.

```

package ma.enset.aes;

import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;

import javax.crypto.Cipher;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.security.KeyFactory;

import java.security.PrivateKey;
import java.security.spec.PKCS8EncodedKeySpec;
import java.util.Base64;

public class ServerAgent extends Agent {
    @Override
    protected void setup() {
        String secret = (String) getArguments()[0];
        addBehaviour(new CyclicBehaviour() {
            @Override
            public void action() {
                ACLMessage receive = receive();
                if(receive!=null){
                    String crypteMsg = receive.getContent();
                    byte[] cryptMsg =
Base64.getDecoder().decode(crypteMsg);
                    try {
                        SecretKey secretKey = new
SecretKeySpec(secret.getBytes(), "AES");
                        Cipher cipher = Cipher.getInstance("AES");
                        cipher.init(Cipher.DECRYPT_MODE, secretKey);
                        byte[] decryptMsg = cipher.doFinal(cryptMsg);
                        System.out.println(new String(decryptMsg));
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }else{
                    block();
                }
            }
        });
    }
}

```

```
    }  
    });}
```

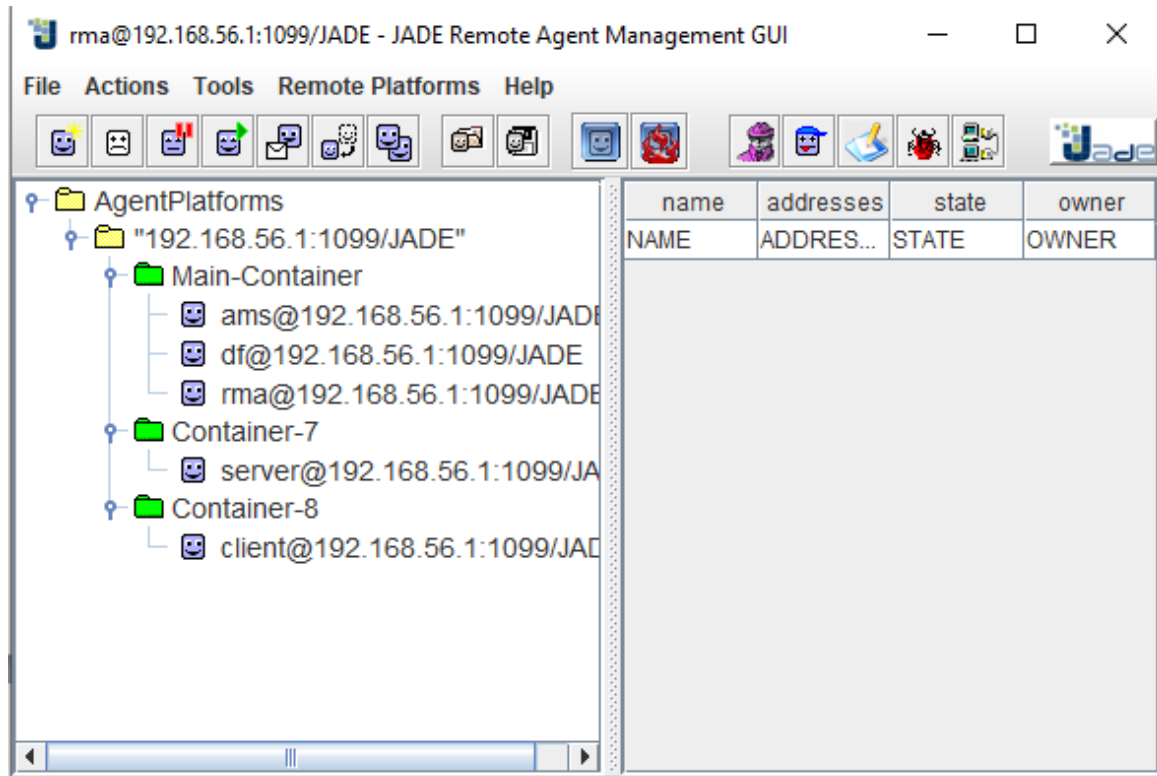
#### 4- Classe ClientAgent :

Le code représente un agent client AES dans le cadre de l'architecture d'agents Jade. Le message « hello server !! » Est chiffré en utilisant une clé secrète AES définie par la chaîne de caractères "secret" passée en argument lors de la création de l'agent.

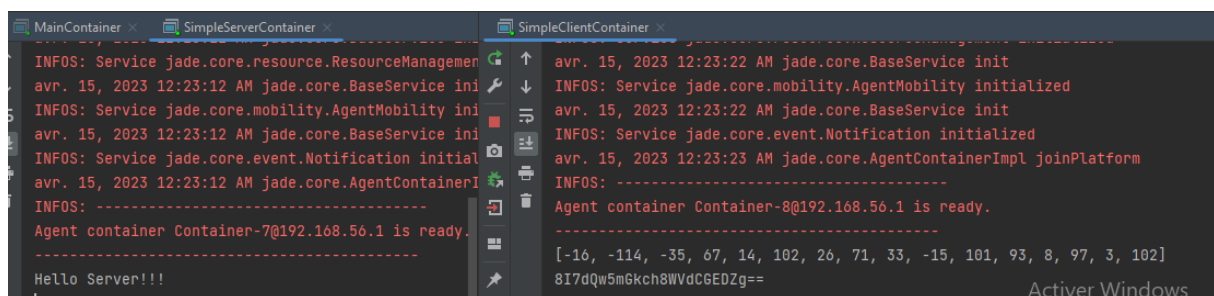
```
package ma.enset.aes;  
  
import jade.core.AID;  
import jade.core.Agent;  
import jade.lang.acl.ACLMessage;  
  
import javax.crypto.Cipher;  
import javax.crypto.SecretKey;  
import javax.crypto.spec.SecretKeySpec;  
import java.util.Arrays;  
import java.util.Base64;  
  
public class ClientAgent extends Agent {  
    @Override  
    protected void setup() {  
        String secert = (String) getArguments()[0];  
        String message = "Hello Server!!!";  
        try {  
            SecretKey secretKey = new SecretKeySpec(secert.getBytes() ,  
"AES");  
            Cipher cipher = Cipher.getInstance("AES");  
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);  
            byte[] cryptMsg = cipher.doFinal(message.getBytes());  
            String cryptedExceptionMsg =  
Base64.getEncoder().encodeToString(cryptMsg);  
            ACLMessage aclMessage = new ACLMessage(ACLMessage.INFORM);  
            aclMessage.addReceiver(new AID("server" , AID.ISLOCALNAME));  
            aclMessage.setContent(cryptedExceptionMsg);  
            send(aclMessage);  
            System.out.println(Arrays.toString(cryptMsg));  
            System.out.println(cryptedExceptionMsg);  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

## 5- Exemples d'exécution :

Voici La plateforme du jade :



Et voilà le message a été bien reçu par le serveur :



#### **IV. Conclusion**

En conclusion, AES (Advanced Encryption Standard) et RSA (Rivest, Shamir, Adleman) sont deux algorithmes de chiffrement largement utilisés dans le domaine de la sécurité informatique. AES est un algorithme de chiffrement symétrique, ce qui signifie qu'une seule clé est utilisée pour le chiffrement et le déchiffrement, tandis que RSA est un algorithme de chiffrement asymétrique, ce qui signifie qu'une paire de clés (clé publique et clé privée) est utilisée pour le chiffrement et le déchiffrement.

AES est généralement utilisé pour le chiffrement de données sensibles ou confidentielles, notamment dans les communications sécurisées, les systèmes de fichiers chiffrés, et les protocoles de sécurité. Il offre une sécurité élevée et est efficace en termes de performances, ce qui en fait un choix populaire pour de nombreuses applications.

RSA, quant à lui, est principalement utilisé pour la sécurisation des communications et l'échange sécurisé de clés dans des environnements où la sécurité et l'authentification sont des préoccupations majeures. RSA offre une sécurité asymétrique en utilisant une paire de clés publique/privée, où la clé publique peut être partagée publiquement pour le chiffrement et la clé privée est gardée secrète pour le déchiffrement. RSA est souvent utilisé en combinaison avec d'autres algorithmes pour établir des canaux de communication sécurisés.

Les systèmes multi-agents, tels que l'architecture Jade utilisée dans les exemples de code fournis, sont des systèmes distribués dans lesquels plusieurs agents autonomes interagissent pour atteindre un objectif commun. Les agents peuvent communiquer entre eux en utilisant différents protocoles de communication, y compris le chiffrement des données pour garantir la confidentialité et la sécurité des informations échangées. L'utilisation d'algorithmes de chiffrement tels qu'AES et RSA dans les systèmes multi-agents peut contribuer à renforcer la sécurité des communications et à protéger les données sensibles échangées entre les agents.