



Institut National
Universitaire
Champollion

Projet chat sécurisé

Compte rendu

Gaël Burguès
Laurian Dufrechou
Lucàs Vabre

Proposé par: Monsieur POUIT

Table des matières

1. Introduction	3
2. Communication client / serveur	3
3. Génération de clé AES	3
4. Utilisation du système RSA	4
5. Multi-clients	5
6. Interface et manuel d'utilisation	6
Annexe	8
Diagramme de classe du projet chat sécurisé:	8

1. Introduction

Au cours de notre 5ème semestre de Licence Informatique, Monsieur POUIT nous a proposé un projet lors de ses cours de réseau. Celui-ci avait pour but de nous initier à la communication sécurisée entre 2 personnes ou plus. Le projet impliquait donc l'utilisation de sockets, de clés de cryptage et de threads.

2.Communication client / serveur

Afin de réaliser la communication client / serveur, nous avons fait dans un premier temps deux programmes, un lancé par le serveur et un par le client. Celui du serveur crée un `ServerSocket` et recherche un client.

Du côté du client, on crée un nouvel objet `Socket` avec l'host et le bon port (dans notre cas "localhost" et 4444). Si ces deux paramètres sont les bons du côté serveur, la connexion sera établie. Puis le serveur et le client se mettront respectivement en mode réception et envoi, le serveur va donc attendre un message envoyé depuis le client à travers l'`InputStream` du socket du client. Après avoir reçu un message, il l'affiche sur la console et attend un message (un input) de l'utilisateur. Pendant ce temps le client est bloqué en réception et ce processus est répété jusqu'à ce que le message d'un des deux partis soit "bye".

3.Génération de clé AES

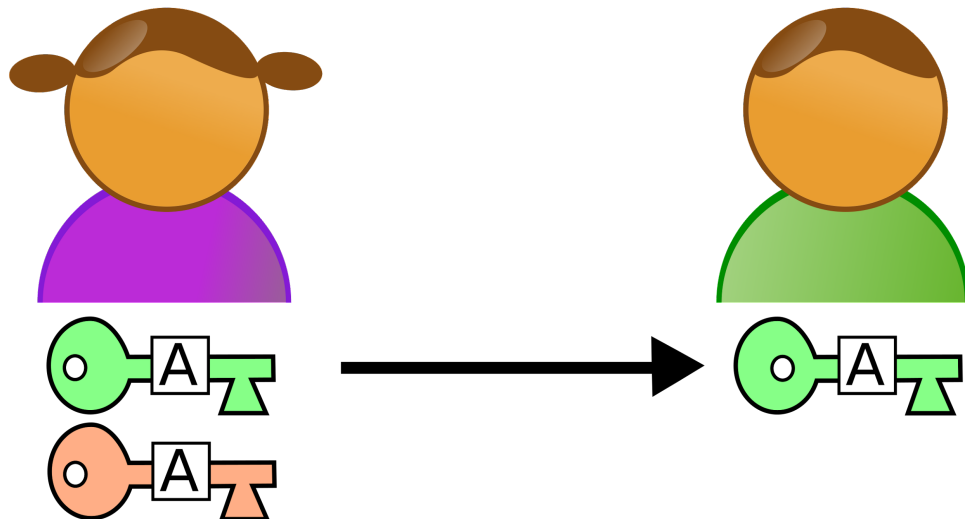
Pour un échange sécurisé il faut pouvoir chiffrer et déchiffrer les messages, pour cela nous allons créer une fonction capable de créer une clé AES, de sauvegarder une clé dans un fichier et d'en charger une depuis un fichier. Pour créer la clé nous avons utilisé la librairie `javax.crypto` qui nous permet d'utiliser la méthode `KeyGenerator` nécessitant seulement en paramètre le nom de l'algorithme utilisé pour la créer (dans notre cas le DES) et qui renvoie la clé générée.

Pour la charger et sauvegarder dans un fichier il suffit d'utiliser des `ObjectInputStream` et `ObjectOutputStream` qui vont venir sérialiser ou désérialiser la clé.

4. Utilisation du système RSA

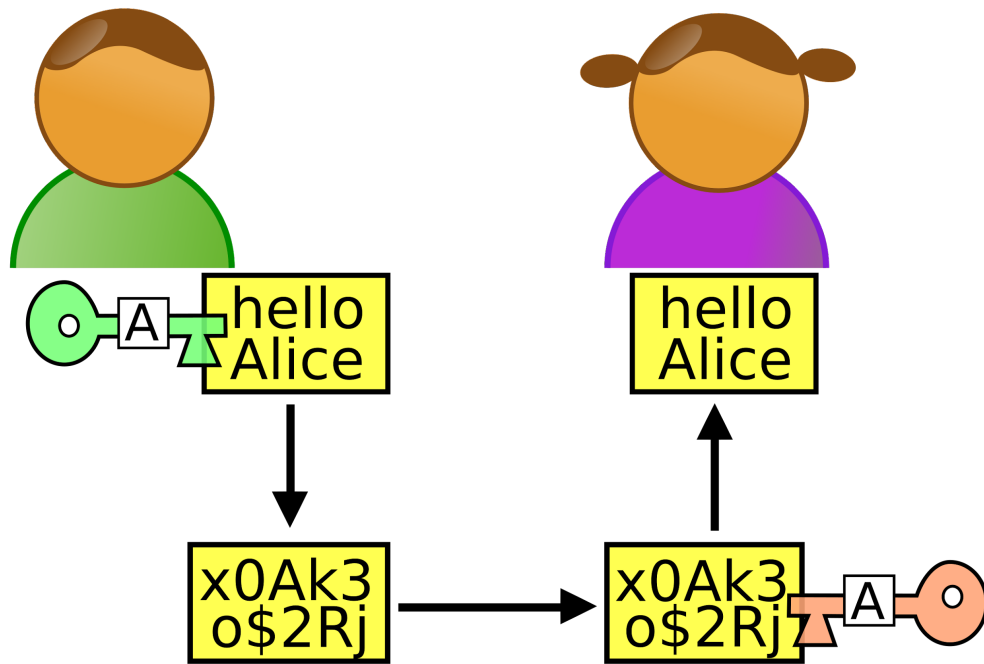
Les clés AES peuvent nous servir à réaliser un transfert de message sécurisé, cependant l'envoi de cette dite clé ne peut pas s'envoyer de manière sécurisée sans réaliser quelque chose de complexe.

Nous avons donc préféré utiliser le système RSA qui consiste à avoir une clé privée et une clé publique pour chaque parti et d'envoyer la clé publique à l'autre parti.



Puis lors de l'envoi d'un message, nous le chiffons en utilisant la clé publique de l'expéditeur et lors de la réception, nous déchiffrons le message avec la clé privée de l'expéditeur du message.

La clé publique n'est donc pas chiffrée lorsqu'on la transfère au destinataire mais elle ne permet pas à elle seule de trouver la clé privée.



5. Multi-clients

Jusqu'à maintenant, on ne pouvait envoyer que des messages chacun son tour. Pour avoir plusieurs clients il faudra absolument éviter ça.

Nous avons donc décidé que le serveur ne pourrait plus envoyer de message rédigé par un utilisateur mais seulement des messages systèmes et aussi transférer les messages d'un client aux autres. Nous utilisons donc les threads qui permettent de séparer l'exécution d'un programme afin que ces parties ne soient pas bloquantes.

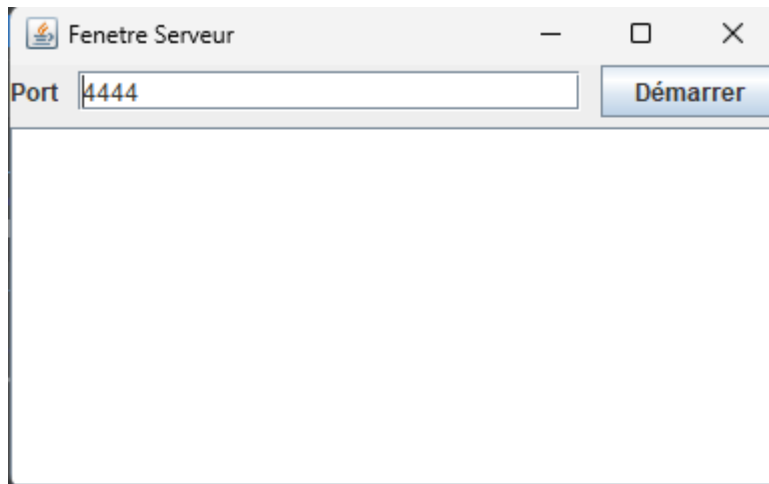
Chaque `threadServer` est affecté à une connexion et donc un client. Ce thread récupère le pseudo envoyé par le client et l'affiche à tout le monde (parcours de tous les threads du serveur et envoi à chacun d'eux) de même pour les messages qui seront envoyés. Le chiffage et déchiffage est encore présent du côté serveur et client.

Quant aux clients, les threads sont aussi utilisés. Après l'envoi du pseudo au serveur, un `listenThread` est créé celui-ci cherchant en boucle un nouveau message du serveur.

Pour l'envoi d'un message depuis le client, il n'a pas changé, on attend l'écriture d'un message, on le chiffre et l'envoi au serveur (celui-ci le transférera à tous les autres clients).

6.Interface et manuel d'utilisation

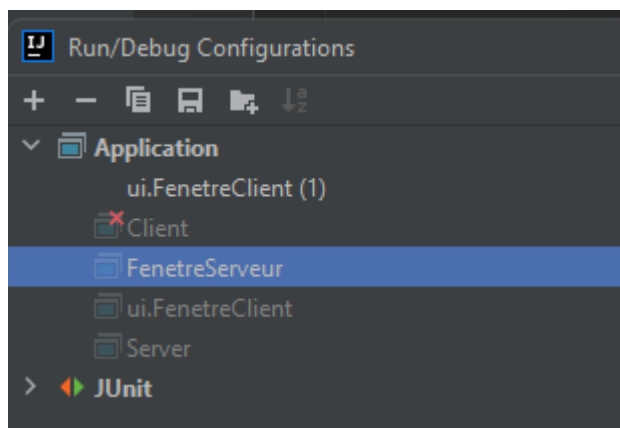
Premièrement, il faut lancer le main de ui.FenetreServeur.



Vous obtiendrez cette fenêtre, il faut choisir le port du serveur qu'on va créer.

Ensuite il faut démarrer le serveur avec le bouton.

Ensuite il faut lancer 2 fois la méthode main de ui.FenetreClient.



Sur IntelliJ :

liste des run (à côté du bouton build)

→ "Edit Configurations"

→ le "+" en haut à gauche

→ application

→ choisir le nom et la "main class" dans le menu de droite

Sur eclipse :

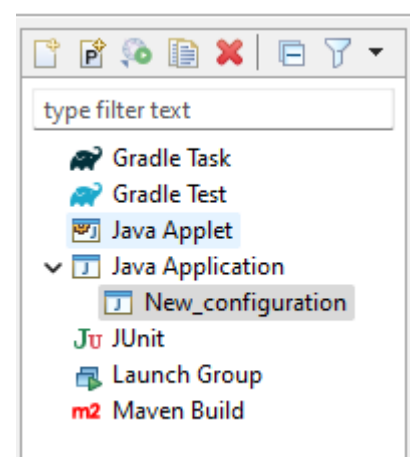
flèche à côté de Run




→ "Run Configurations"

→ "Java Application"

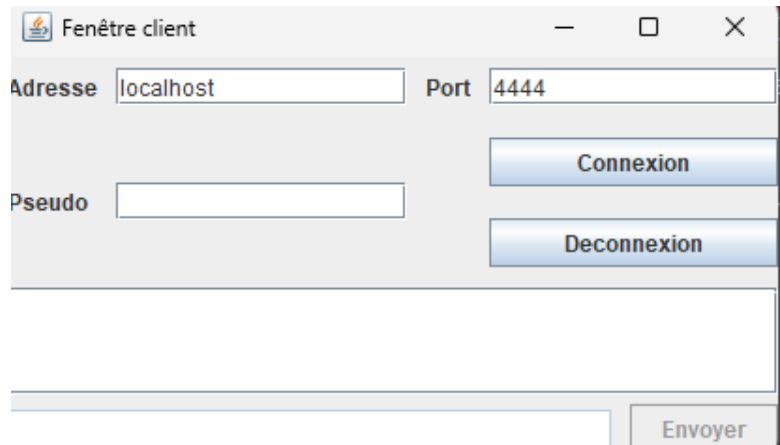
→ choisir le nom et la "main class" dans le menu de droite



Puis pour le run, Run simple  et choisissez la configuration créée.

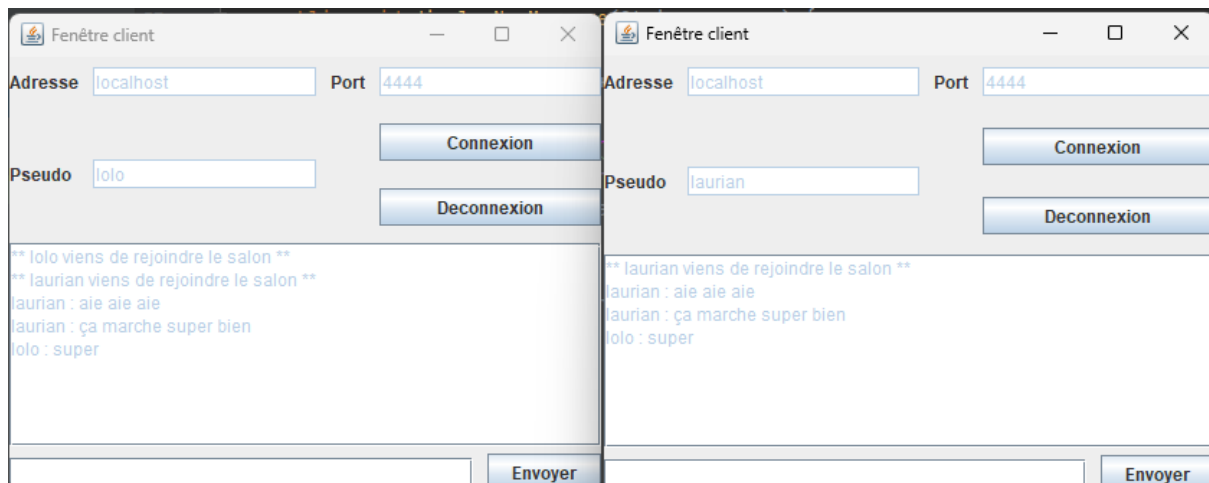
Lorsqu'on a lancé une FenetreClient, nous obtenons la fenêtre suivante :

Il suffit maintenant de rentrer la bonne adresse et le bon port (dans notre cas localhost et le port saisi dans la FenetreServeur).



Cliquez ensuite sur connexion et vous pouvez ensuite saisir vos messages depuis le champ de texte en bas de la fenêtre, envoyer avec le bouton du même nom.

Si vous faites ces manipulations sur la deuxième fenêtre, vous verrez que les messages s'afficheront sur l'autre (exemple ci-dessous)..



Annexe

Diagramme de classe du projet chat sécurisé:

