Thibault MALESPINE

Armand LECOURT

Rapport Projet Chat Sécurisé

Table des matières

1/ Arborescence

Notre projet java consiste en 7 classes différentes :

-AES

- RSA

-Server

-Client

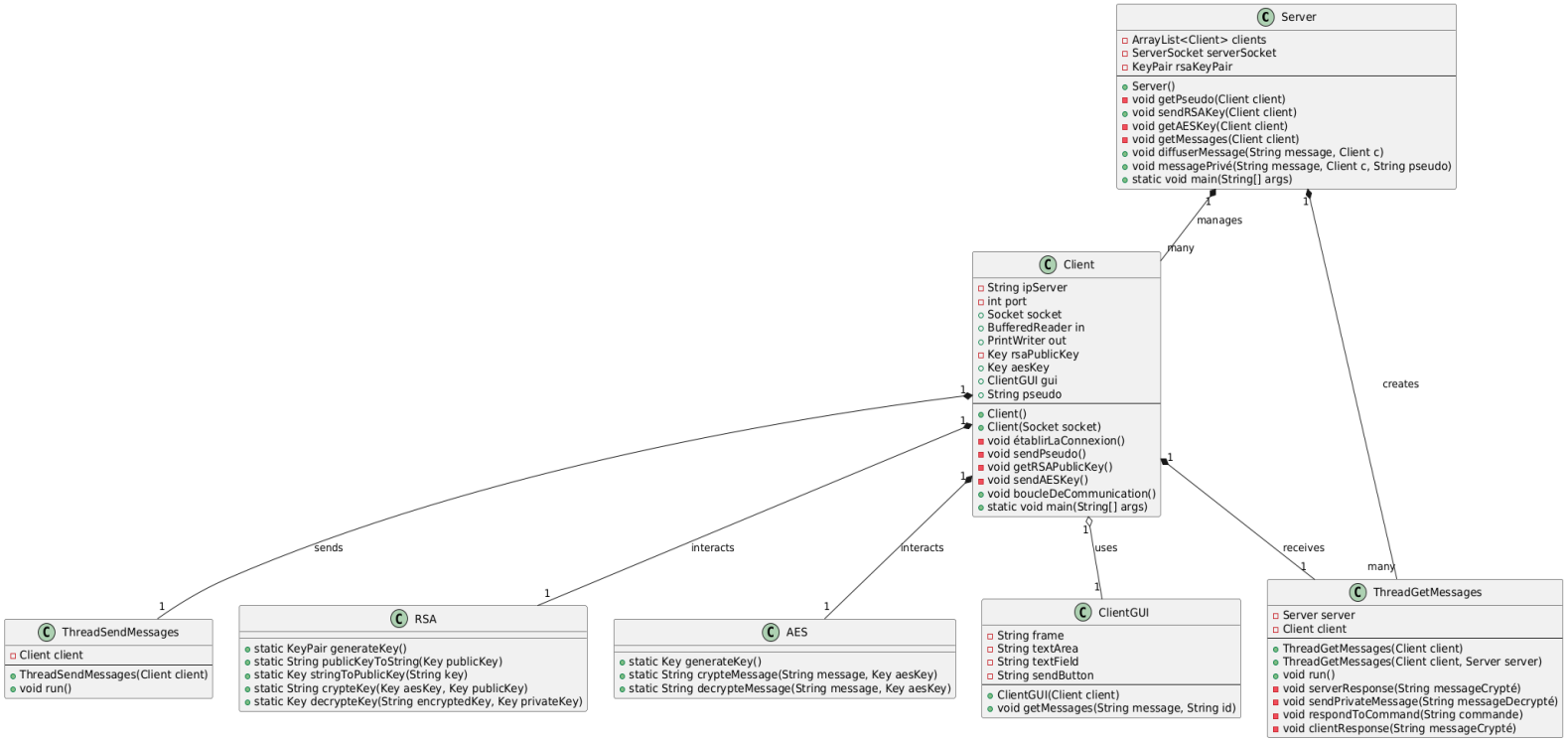
-ClientGUI

-ThreadGetMessages

-ThreadSendMessages

2/ Diagramme UML

Voici le diagramme UML du projet :



3/ Serveur

La classe Server constitue le cœur de l'architecture du projet en gérant la communication entre les différents clients connectés. Elle repose sur plusieurs composantes clés pour assurer la gestion des connexions, la sécurité des échanges et la diffusion des messages.

3.1. Fonctionnalité principale

Le serveur attends qu'un client se connecte puis effectue les étapes suivantes :

-Génération de clés RSA : Le serveur crée une paire de clés RSA (*rsaKeyPair*) pour sécuriser les échanges initiaux avec les clients.

-Ouverture d'un socket serveur : Un objet *ServerSocket* écoute les connexions entrantes sur le port 4444.

-Gestion des connexions clients : Lorsqu'un client se connecte, le serveur :

- Accepte la connexion via un Socket.
- Crée une instance de la classe Client associée à ce socket.
- Récupère le pseudo du client (*getPseudo()*).
- Envoie la clé publique RSA au client pour sécuriser les échanges ultérieurs (*sendRSAKey()*).

- Reçoit une clé AES chiffrée en utilisant RSA, afin d'établir une communication sécurisée avec ce client (getAESKey()).
- Lance un thread pour recevoir et traiter les messages du client (getMessages()).

Pour chaque nouveau client, ce processus sera répété.

3.2. Gestion des messages

Pour gérer les messages, le serveur crée un Thread pour chaque client.

Le serveur offre deux modes principaux de gestion des messages :

- Diffusion publique : Lorsqu'un message est reçu, il est chiffré avec la clé AES de chaque client et envoyé à tous les autres clients connectés via la méthode diffuserMessage().
- Messages privés : Si un message est destiné à un client spécifique, il est transmis uniquement au destinataire grâce à la méthode messagePrivé().

4/ Client

La classe Client représente l'élément côté client dans le projet de chat. Elle gère l'établissement de la connexion avec le serveur, la sécurisation des échanges et la transmission des messages. Cette classe est conçue pour offrir une interface utilisateur graphique et permettre des communications sécurisées.

4.1. Fonctionnalité principale

La classe Client réalise plusieurs étapes essentielles :

Établissement de la connexion :

- Elle se connecte au serveur via un socket (Socket) à une adresse IP (ipServer) et un port défini.
- Les flux de communication (BufferedReader() et PrintWriter()) sont initialisés pour l'envoi et la réception des messages.

4.2. Authentification et sécurisation

- Pseudo : Le client envoie son pseudo au serveur via la méthode sendPseudo().
- RSA : Il récupère la clé publique RSA du serveur (getRSAPublicKey()) et l'utilise pour chiffrer une clé AES générée localement.
- AES : La clé AES chiffrée est envoyée au serveur pour permettre des communications sécurisées (sendAESKey()).

4.3. Gestion des communications

Une boucle de communication est lancée via la méthode boucleDeCommunication(). Deux threads sont créés :

- ThreadGetMessages : Pour écouter et afficher les messages reçus.
- ThreadSendMessages : Pour envoyer des messages à travers l'interface graphique du client.

5/ Threads

Deux classes, ThreadGetMessages et ThreadSendMessages gérent l'envoie et la réception des messages, ce sont des classes qui hérite de la classe Thread de Java.

6/ Cryptage

Le projet intègre deux algorithmes de cryptographie, AES et RSA, pour garantir la sécurité des échanges entre le client et le serveur. Chaque classe gère un aspect spécifique du chiffrement hybride utilisé.

6.1. Modèle de chiffrement hybride

Le projet combine AES et RSA pour tirer parti des points forts de chaque algorithme :

- Le serveur génère une paire de clés RSA et transmet la clé publique au client.
- Le client génère une clé AES, la chiffre avec la clé publique RSA, et l'envoie au serveur.
- Les deux parties utilisent ensuite la clé AES pour échanger des messages chiffrés.

6.2. Algorithme AES (Advanced Encryption Standard)

L'AES est un algorithme de cryptage symétrique utilisé pour chiffrer et déchiffrer les messages échangés entre le client et le serveur.

- **Génération de clés :**
  La méthode generateKey() génère une clé AES aléatoire, utilisée pour chiffrer les messages échangés. Cette clé est partagée avec le serveur après avoir été chiffrée avec RSA.
- **Chiffrement des messages :**
  La méthode crypteMessage() prend un message en clair et le chiffre avec une clé AES en utilisant l'objet Cipher. Le résultat est encodé en Base64 pour faciliter sa transmission.
- **Déchiffrement des messages :**
  La méthode decrypteMessage() reçoit un message chiffré (encodé en Base64), le décode et le déchiffre à l'aide de la clé AES correspondante.

L'AES est utilisé pour le chiffrement des messages. Les clés AES sont générées dynamiquement pour chaque session, garantissant une sécurité renforcée.

6.3. Algorithme RSA

RSA est un algorithme de cryptage asymétrique utilisé dans ce projet pour sécuriser la transmission de la clé AES.

- **Génération de paires de clés :**
  La méthode generateKey() génère une paire de clés RSA (publique et privée) de 2048 bits. La clé publique est utilisée pour chiffrer, tandis que la clé privée sert à déchiffrer.
- **Chiffrement des clés AES :**
  La méthode crypteKey() chiffre une clé AES avec la clé publique RSA. Ce chiffrement asymétrique garantit que seule la clé privée associée peut déchiffrer la clé AES.
- **Déchiffrement des clés AES :**
  La méthode decrypteKey() déchiffre une clé AES chiffrée avec RSA à l'aide de la clé privée. Une fois déchiffrée, la clé AES est utilisée pour sécuriser les échanges ultérieurs.
- **Conversion des clés :**
  publicKeyToString() convertit une clé publique en chaîne Base64 pour faciliter son envoi au client.
  stringToPublicKey() permet de reconstruire une clé publique à partir d'une chaîne Base64 reçue.

7/ Fenêtre graphique

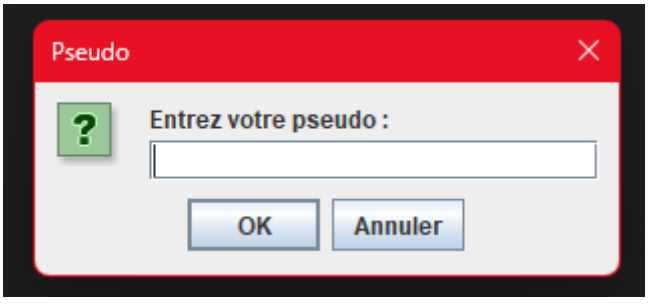
7.1. Interface graphique

La classe ClientGUI crée une interface graphique pour le client.
Elle consiste en un textArea qui à l'aide de la méthode getMessages() permet au client d'afficher les messages en les ajoutant au textArea.

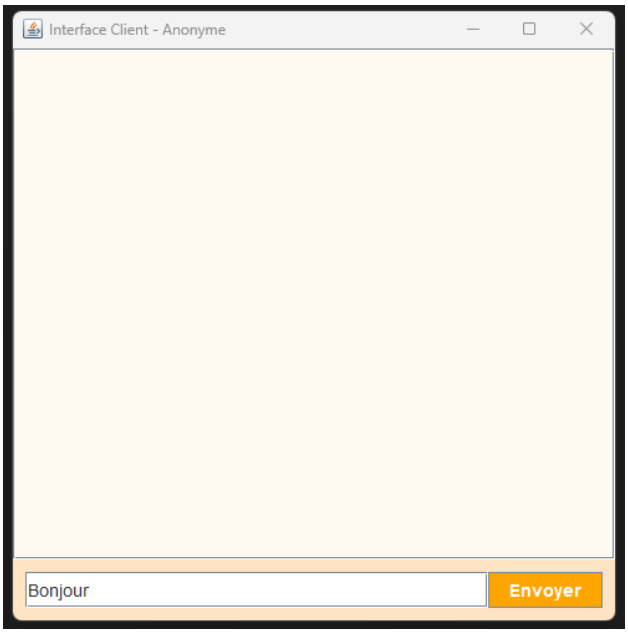
7.2. Guide pas à pas

Il faut tout d'abord s'assurer que le serveur est démarré.

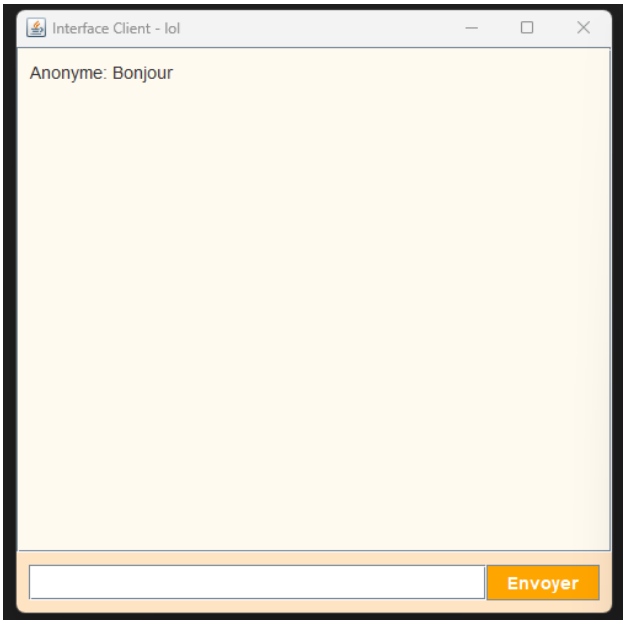
Puis lancer le client, une première fenêtre s'affiche alors, avec le pseudo à entrer :



Ensuite l'interface de messagerie s'affiche avec le pseudo choisi en haut et un textArea en bas pour écrire le message souhaité suivi d'un bouton envoyer :



Le message sera affiché sur tous les autres clients connectés avec le pseudo de l'envoyeur :



8/ Fonctionnalités supplémentaires

8.1. Message privé

Nous avons intégré une fonctionnalité de message privé, si le client souhaite envoyer un message privé à un seul client au lieu de tous, dans ce cas-là le serveur vérifie que le message envoyé commence par « @ ».

La méthode envoie ensuite le message uniquement au pseudo indiquer après le « @ ».

8.2. Commandes

Un système de commande existe, par exemple en tapant « \all » le serveur renvoie la liste de tous les clients.

Cette fonctionnalité peut donc être enrichie avec l'ajout d'autres commandes.