PDF PROJET CHAT

Manuel d'utilisation :

- **L'adresse IP** (réglée par défaut sur le localhost) peut être modifiée à la ligne 31 du fichier Client.java.

- Comment utiliser l'application :
  - 1ere étape : Exécuter le fichier **Serveur.java**, le serveur se lance et attend d'éventuelles connexions.
  - 2e étape : Exécuter le fichier **Client.java**. Un client se connecte au serveur, et 2 interfaces graphiques se lancent (une pour le serveur et une pour le client). Le client et le serveur peuvent alors communiquer de façon sécurisée grâce à la clé de **cryptage** qu'ils se sont échangés au moment de la connexion. Il suffit de taper son message (**composé de caractères ASCII uniquement**) au milieu de la fenêtre, puis de cliquer sur envoyer ou d'appuyer sur Entrée pour envoyer son message.
  - 3e étape (facultative, **multicast**) : Il est possible de lancer une autre instance de la classe Client.java afin de créer un autre client. Ce client se connecte alors au serveur et peut alors discuter avec les autres clients et le serveur, également de façon sécurisée. Il est possible de créer autant de clients que l'on veut, **tant que l'on patiente au moins 1 sec. entre chaque lancement.**

- L'en-tête *Personnes connectées* indique aux clients et au serveur le nombre de personnes connectées, et donc le nombre de personnes qui peuvent communiquer ensemble.

Explication sur le projet :

Pour le **multicast**, lorsque le serveur reçoit un message venant d'un client, il diffuse ce message à tous les autres clients.

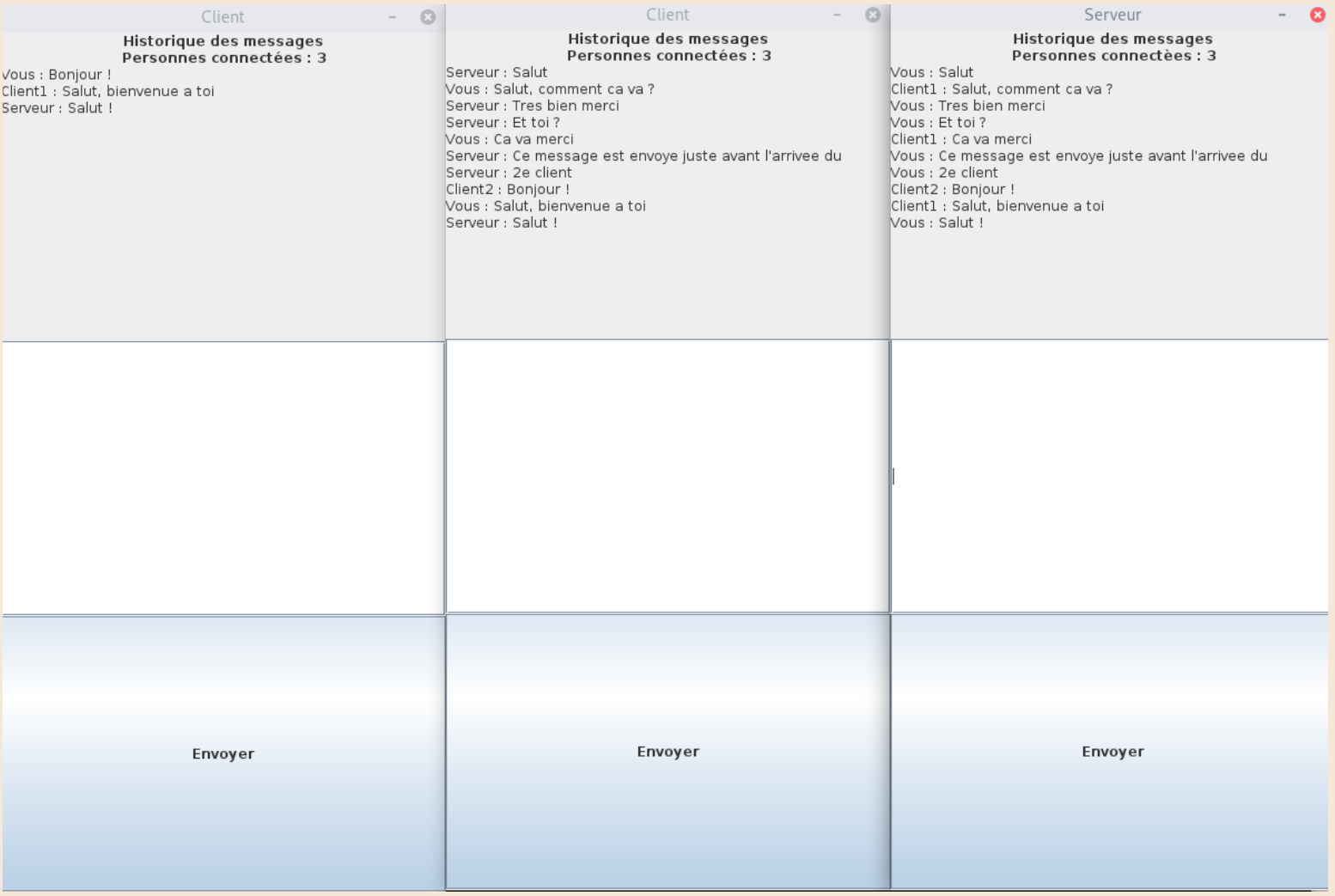
Pour la gestion des **clés de cryptage**, nous avons décidé de choisir une clé par client, sinon, à chaque nouvelle connexion d'un client, il fallait changer la clé pour tous les autres clients. Il est plus simple pour tout le monde que le serveur garde une **liste** avec une clé de cryptage pour chaque client.

Pour l'échange de la première clé, nous avons suivi le protocole d'échange **Diffie-Hellman** ; il faut que les clients et le serveur choisissent un nombre à l'avance et un groupe (difficilement inversible de préférence). Nous avons choisi le nombre 428 et le groupe (Z*, .). Le client et le serveur choisissent 2 nombres aléatoirement dans ce groupe, multiplient ce nombre par 428 et l'envoient au correspondant. Chacun multiplie à nouveau par son nombre et les 2 utilisateurs sont sûrs d'avoir la même clé, sans avoir donné le moindre indice à un éventuel pirate.

Chaque message échangé entre le client et le serveur possède un **en-tête** (envoyé en clair) afin que le réceptionnaire connaisse la nature du message (mise à jour du nombre de personnes connectées, message, ou information de déconnexion). Seul le contenu des messages est crypté.

Une difficulté côté serveur aura été de trouver une façon d'attendre une éventuelle connexion d'un autre client tout en continuant de mettre à jour l'interface graphique. La présence de la classe *Mon_Thread* nous permet de **faire tourner 2 boucles simultanément** : une pour gérer les connexions, et l'autre pour gérer l'interface graphique.

Lorsque le serveur est fermé, les clients ne peuvent plus communiquer, donc l'interface graphique de tous les clients se ferme.


Client
Historique des messages
Personnes connectées : 3
Vous : Bonjour !
Client1 : Salut, bienvenue a toi
Serveur : Salut !
Envoyer
Client
Historique des messages
Personnes connectées : 3
Serveur : Salut
Vous : Salut, comment ca va ?
Serveur : Tres bien merci
Serveur : Et toi ?
Vous : Ca va merci
Serveur : Ce message est envoye juste avant l'arrivee du
Serveur : 2e client
Client2 : Bonjour !
Vous : Salut, bienvenue a toi
Serveur : Salut !
Envoyer
Serveur
Historique des messages
Personnes connectèes : 3
Vous : Salut
Client1 : Salut, comment ca va ?
Vous : Tres bien merci
Vous : Et toi ?
Client1 : Ca va merci
Vous : Ce message est envoye juste avant l'arrivee du
Vous : 2e client
Client2 : Bonjour !
Client1 : Salut, bienvenue a toi
Vous : Salut !
Envoyer