Rapport de Projet

Développement d'une Application de Chat Sécurisée

Introduction à la Sécurité

Réalisé par :
Néha Sougoumar,Anyce Ekomono ,Frençois Godin,Valentin Guillon , Maëva Himeur, Dounia Hullot

Date de Soumission :
11 Décembre 2024

Table des matières

Introduction

Introduction

Le projet a pour objectif de concevoir et développer une application de mini-chat sécurisé permettant à un ensemble d'utilisateurs de s'échanger des messages sécurisés.

Les principales fonctionnalités recherchées étaient le chiffrement de bout en bout des messages échangés, l'ajout d'une interface minimaliste avec la possibilité d'initier une conversation avec un ou plusieurs usagers, la mise en place d'un serveur pour répondre aux différentes requêtes des clients, et enfin, la mise en place d'une application de client malhonnête.

Nous détaillerons dans ce rapport les fonctionnalités implémentées ainsi que les choix d'implémentation qui ont été fait.

1 Répartition des Contributions

L'équipe a collaboré sur divers aspects du projet, chacun apportant son expertise :
- **Interface Graphique** : Anyce, Dounia, Maëva.
- **Partage de clés** : Néha.
- **Implémentation principale (serveur, client, gestion des connexions)** : Valentin, François.
- **Rédaction du protocole** : L'ensemble de l'équipe.

2 Technologies utilisées

Le projet est développé en **Python**, en s'appuyant sur plusieurs modules et librairies :

Modules principaux :
- **Socket** : Établir les connexions clients / serveur et permettre les échanges d'informations via le réseau.
- **Threading** : Faire tourner plusieurs tâches en parallèle dans notre programme, notamment pour la gestion de plusieurs connexions réseaux.
- **PyNaCl** : Chiffrement symétrique des messages via SecretBox. (voir la documentation)
- **json** : Structurer nos données en texte pour faciliter les échanges.

Interface graphique :
- **Tkinter** : utilisé pour développer l'interface graphique de l'application. Cette bibliothèque native de Python offre des outils puissants et flexibles pour la création d'interfaces utilisateur intuitives.

3 Dépôt GitLab

Le code source complet du projet est hébergé sur GitLab, accessible via le lien suivant : `https://code.up8.edu/fgodin/p8-mini-chat`

4 Structure du code

— **serveur.py** : Ce fichier contient le code du serveur qui gère la connexion des clients, la réception et l'envoi de messages, ainsi que la gestion des groupes.

— **client.py** : Ce fichier contient le code du client, qui inclut l'interface graphique pour l'utilisateur, la gestion des connexions,de l'envoi des messages et de chiffrer les messages.

— **commonlib.py** : Ce fichier contient des fonctions utilitaires partagées entre le client et le serveur, telles que la gestion des dictionnaires de données échangées.

— **rsa.py** : Ce fichier implémente les fonctions liées à la cryptographie RSA, permettant de sécuriser les échanges de clés et plus tard les signatures de messages.

— **secret-box.py** : Ce fichier gère le chiffrement et déchiffrement des messages et des données à l'aide de techniques de cryptographie symétrique. Elle utilise la librarie PyNaCl et est utilsée pour sécuriser les messages envoyées dans les groupes.

5 Utilisation du chat

Pour utiliser le chat sécurisé, il est nécessaire de démarrer d'abord le serveur, puis de lancer les clients.
voici les étapes :
Pour démarrer le serveur, ouvrez un terminal et exécutez :

```
python3 serveur.py
```

Le serveur s'exécutera et attendra les connexions clients.

Pour démarrer le client, ouvrez un ou plusieurs terminaux et exécutez :

```
python3 client.py
```

Le client démarrera dans une interface graphique permettant de se connecter au serveur.

Fonctionnalités implémentées

6 Format des informations transmises

Les informations transitant au travers des sockets suivent un format structuré et uniforme. Chaque message est un **dictionnaire Python**, converti en chaîne de caractères avant d'être envoyé. Ce dictionnaire contient trois champs principaux :

— **L'émetteur** : identifiant ou pseudonyme représentant qui est à l'origine de la requête.
— **Le destinataire** : nom du groupe ou de l'utilisateur ciblé.
— **Le contenu du message** : texte ou données envoyées.

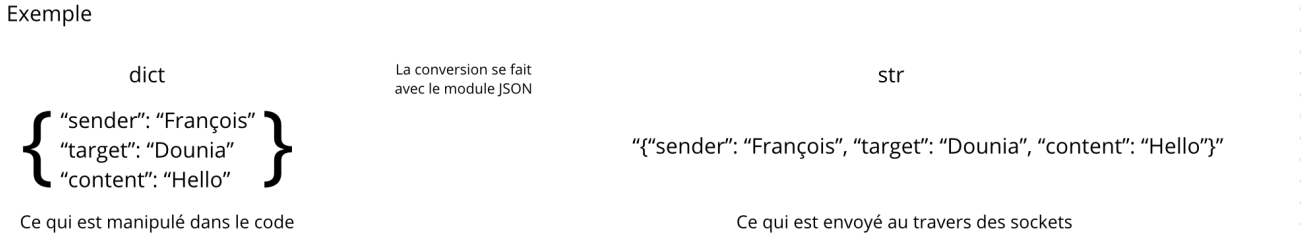


FIGURE 1 – Format des messages

7 Fonctionnalités

L'application implémente les fonctionnalités suivantes :

(a) Préconnexion

La préconnexion se déclenche automatiquement au lancement de l'application. Le client établit une connexion avec le serveur en utilisant des sockets. Lors de cette étape, le serveur crée une instance temporaire pour chaque client connecté. Un identifiant unique, est attribué au client comme nom temporaire. Cet identifiant est ensuite transmis au client, qui l'utilise jusqu'à ce qu'un nom d'utilisateur définitif soit validé.

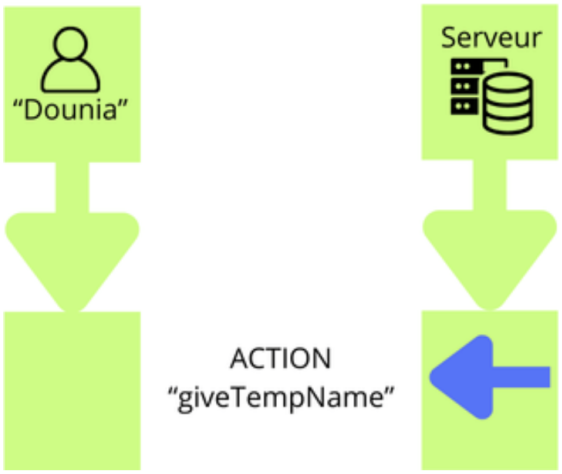


FIGURE 2 – Schéma du protocole de pré-connexion

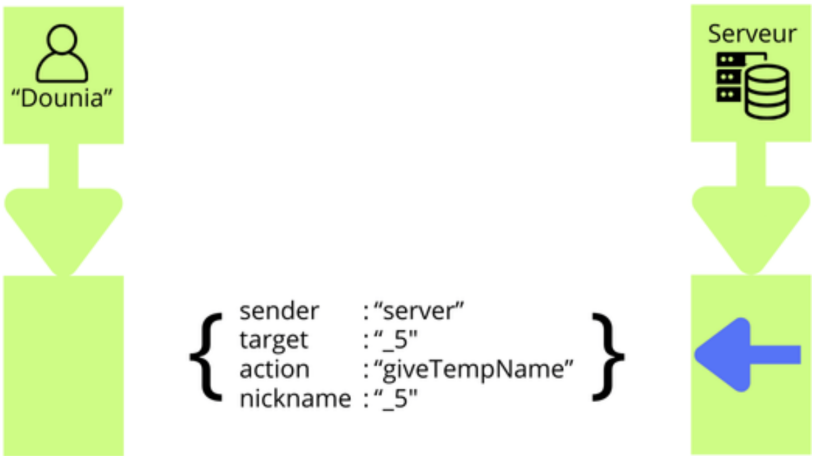


FIGURE 3 – Schéma du protocole de pré-connexion (avec format)

(b) Connexion

La connexion s'active lorsque l'utilisateur valide son pseudo via l'interface de connexion. Le client envoie une requête au serveur contenant le pseudo et une clé publique fraîchement générée. Le serveur analyse cette requête en vérifiant si un client portant le même pseudo est déjà connecté. En fonction des résultats :

— Si le pseudo est déjà utilisé et que le client est connecté, le serveur rejette la requête et retourne un message d'erreur.

— Sinon, le serveur valide la connexion, puis détermine s'il s'agit d'une première connexion ou d'une reconnexion. Dans chaque cas, les informations nécessaires sont transmises au client, permettant une gestion fluide des utilisateurs.

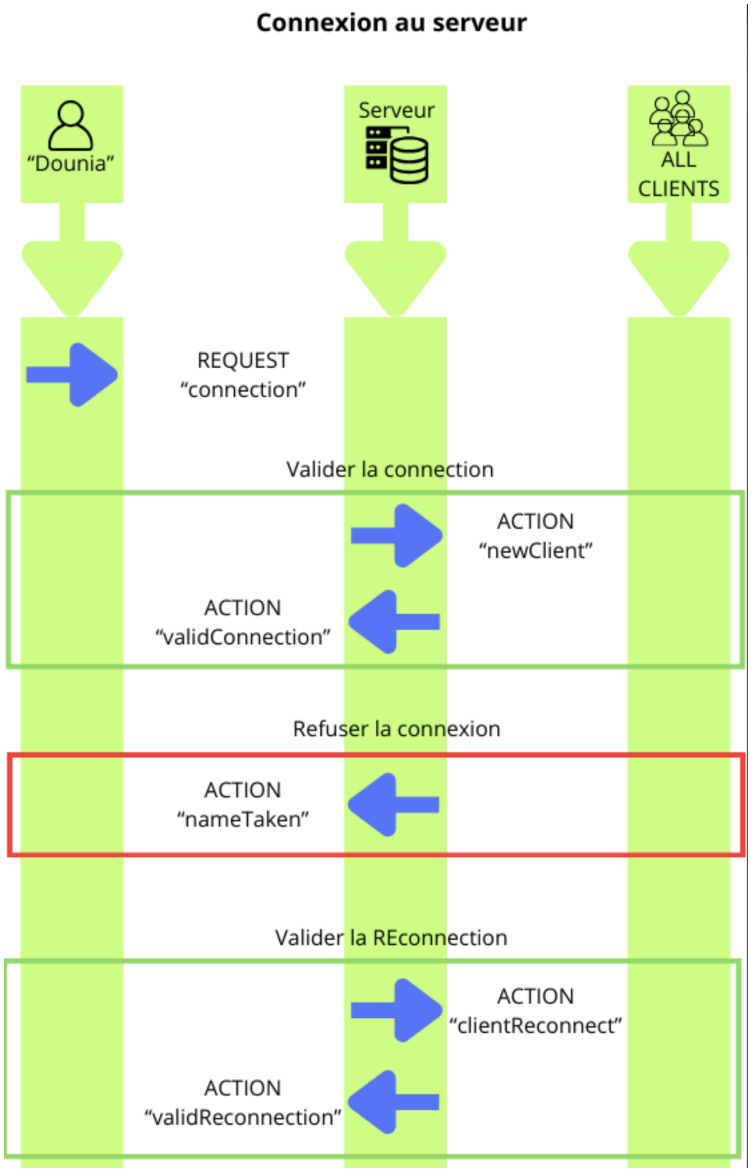
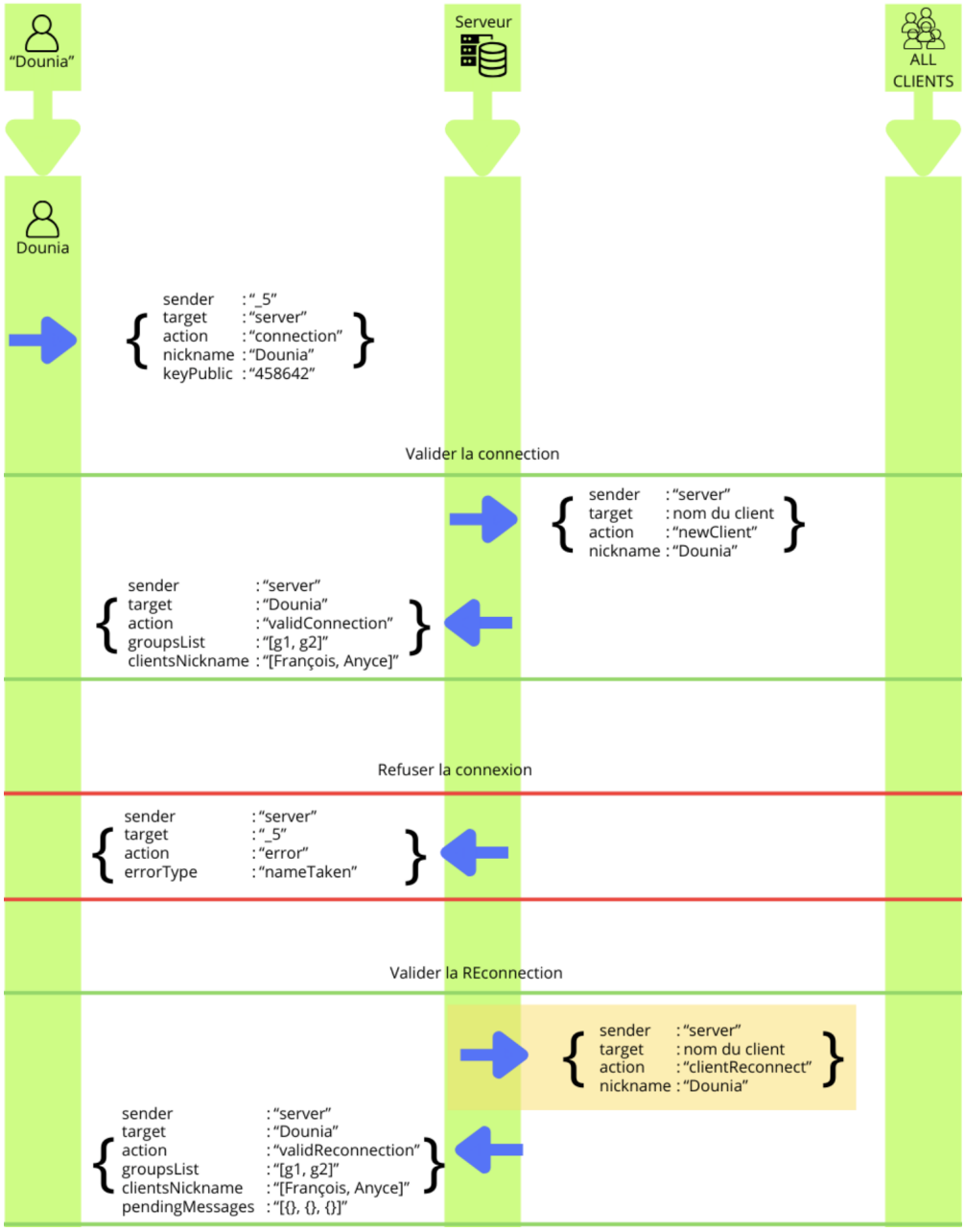


FIGURE 4 – Connexion au serveur


"Dounia"
Serveur
ALL CLIENTS
Dounia
sender : "_5"
target : "server"
action : "connection"
nickname : "Dounia"
keyPublic : "458642"
Valider la connection
sender : "server"
target : nom du client
action : "newClient"
nickname : "Dounia"
sender : "server"
target : "Dounia"
action : "validConnection"
groupsList : "[g1, g2]"
clientsNickname : "[François, Anyce]"
Refuser la connexion
sender : "server"
target : "_5"
action : "error"
errorType : "nameTaken"
Valider la REconnection
sender : "server"
target : nom du client
action : "clientReconnect"
nickname : "Dounia"
sender : "server"
target : "Dounia"
action : "validReconnection"
groupsList : "[g1, g2]"
clientsNickname : "[François, Anyce]"
pendingMessages : "[{}, {}, {}]"

(c) Déconnexion

La déconnexion se produit lorsque l'utilisateur ferme l'interface. Le client envoie un message au serveur pour indiquer son intention de se déconnecter. Le serveur met à jour son état interne pour marquer ce client comme déconnecté. Une fois cette étape terminée, le programme client se ferme.

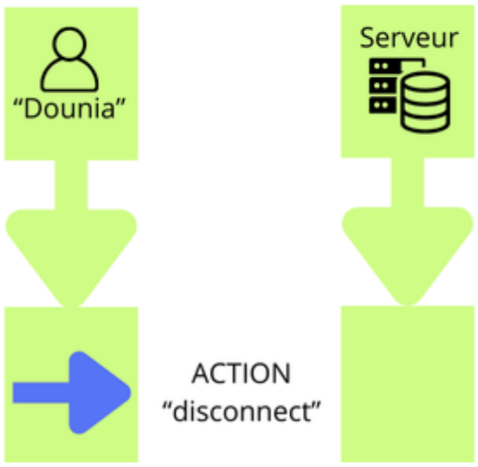


FIGURE 5 – Déconnexion

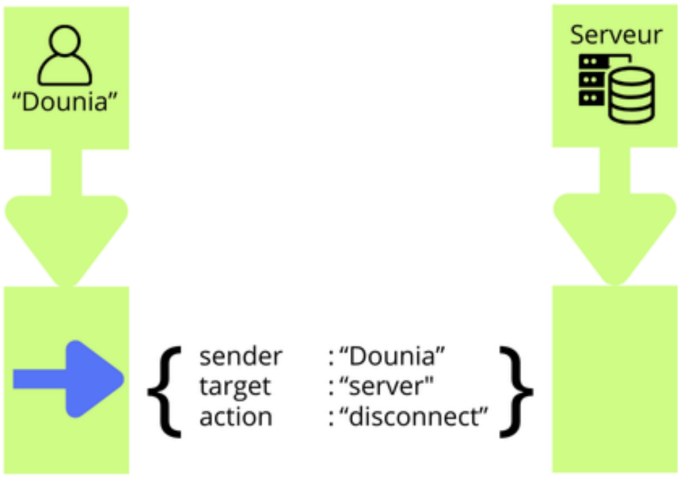


FIGURE 6 – Déconnexion (avec format)

(d) Envoi de Messages

L'envoi de messages est déclenché lorsque l'utilisateur valide un message via l'interface de messagerie. Les messages ne peuvent être échangés qu'à l'intérieur d'un groupe spécifique. Lorsqu'un utilisateur envoie un message, le client transmet ce message au serveur, qui se charge de le distribuer à tous les membres du groupe.

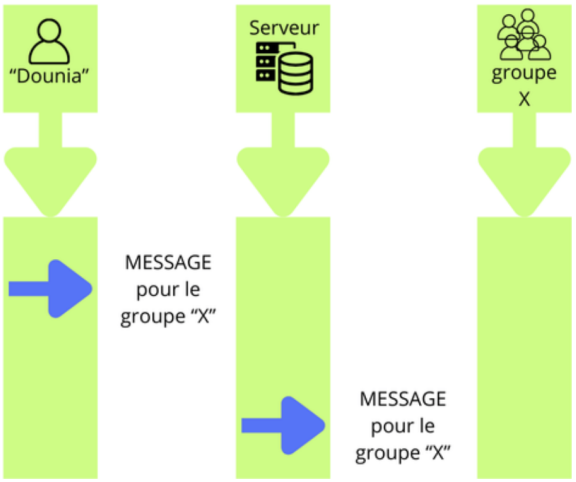


FIGURE 7 – Envoyer un message

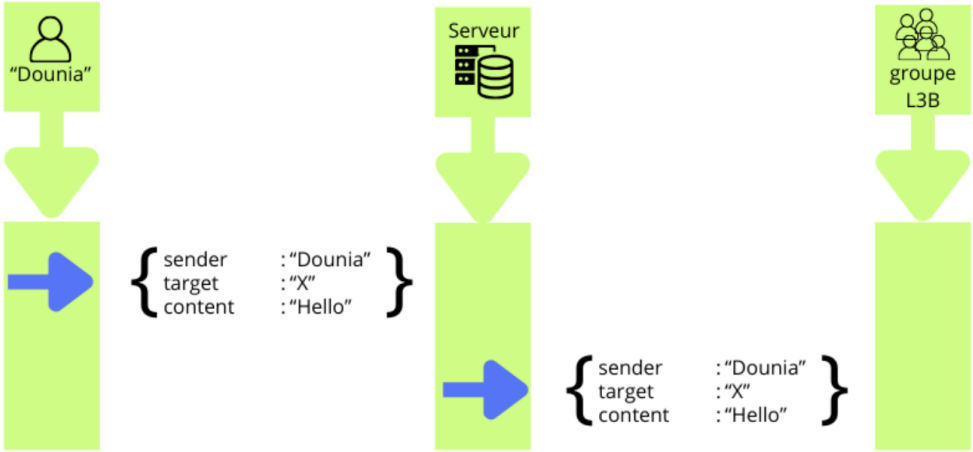


FIGURE 8 – Envoyer un message (avec format)

(e) Création de Groupes

La création de groupes est initiée lorsque l'utilisateur valide un nom via l'interface dédiée. Le client envoie une requête au serveur contenant le nom du groupe souhaité. Le serveur vérifie alors si un groupe portant le même nom existe déjà :

- Si le nom est déjà utilisé, le serveur retourne une erreur.
- Si le nom est disponible, le serveur crée le groupe, notifie tous les clients de son existence, et envoie une confirmation au créateur. Le client créateur génère ensuite une clé pour le groupe et la stocke localement.

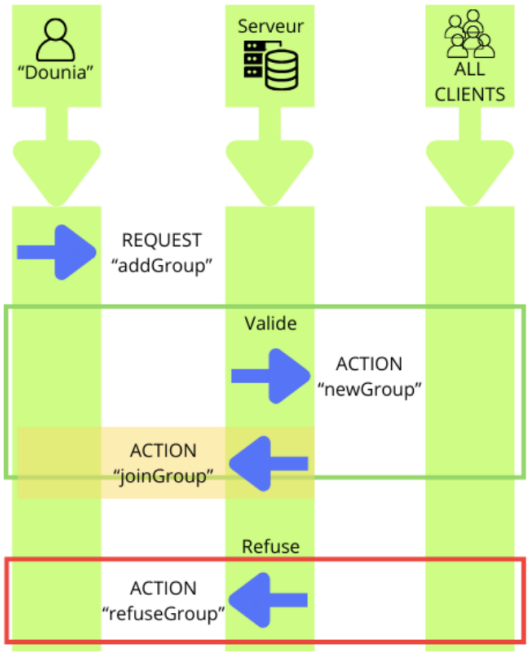


FIGURE 9 – Creation de Groupe

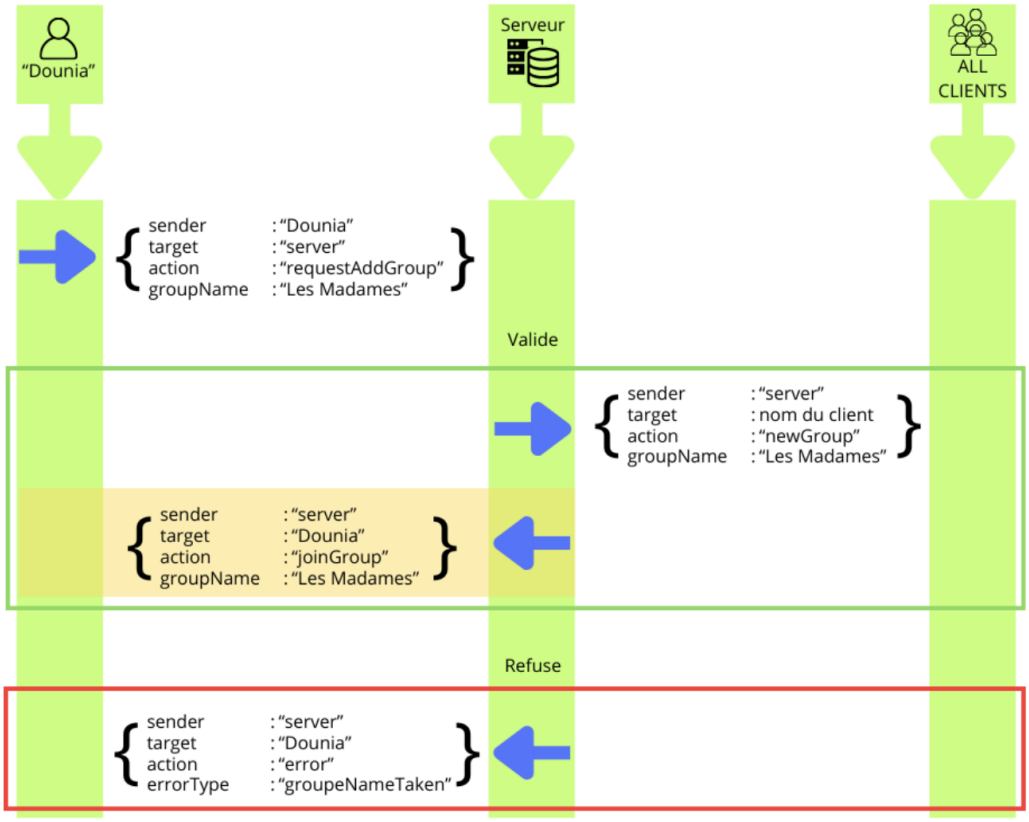


FIGURE 10 – Creation de Groupe (avec format)

Protocole de Sécurité

8 Chiffrement

Deux types de chiffrement sont utilisés dans ce projet :
- Chiffrement asymétrique RSA : Garantit un échange sécurisé de la clé d'un groupe de discussion.
- Chiffrement symétrique à l'aide de **PyNaCl** (Secret Box) : Permet le chiffrement et le déchiffrement des messages dans un groupe de discussion à l'aide d'une clé commune.

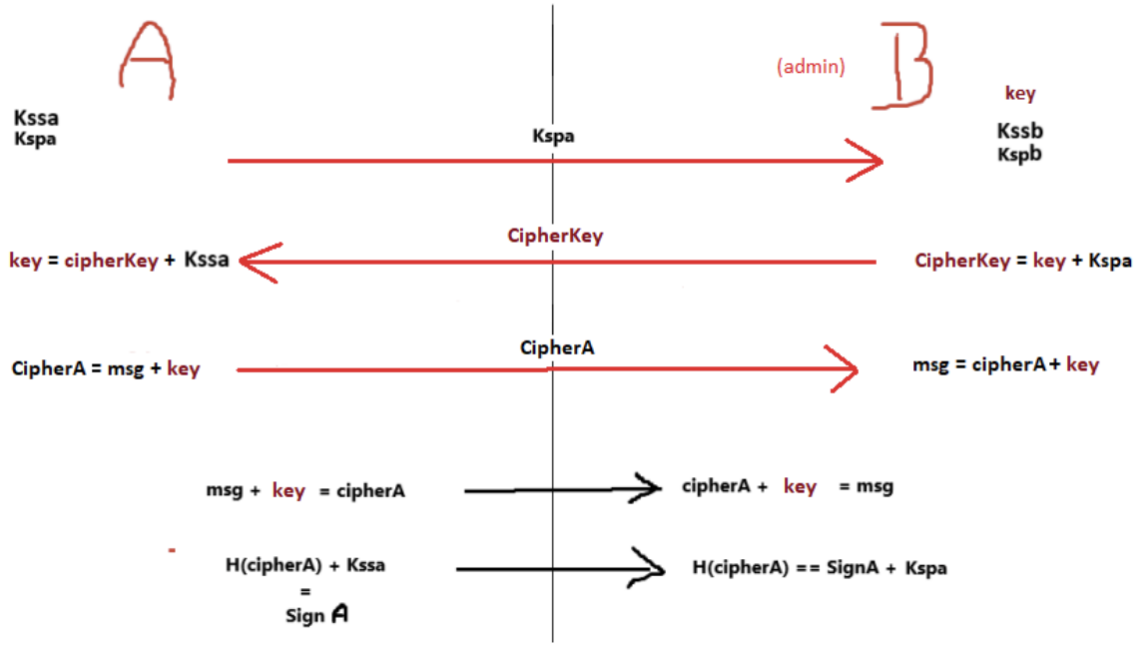


FIGURE 11 – Schéma du chiffrement entre deux clients

— Kss : Clé secrète signature
— Ksp : Clé publique signature
— H(...) : Fonction de hachage pour RSA
— key : Clé de chiffrement symétrique

(a) Chiffrement de la clé de groupe

Chaque client du chat possède une paire de clés RSA : une **clé publique** et une **clé privée**.

Un client transmet sa clé publique RSA lorsqu'il se connecte au serveur et ce dernier la stocke en mémoire.

Lorsqu'un client demande à rejoindre un groupe existant, notre serveur s'arrange pour transmettre la bonne clé publique à l'administrateur du groupe. Celui-ci l'utilise pour chiffrer et envoyer la clé de groupe.
Le client à l'origine de la demande, utilise sa clé secrète RSA pour déchiffrer cette même clé du groupe.

(b) Chiffrement des messages

Pour les échanges de messages, nous avons opté pour le chiffrement symétrique via **Secret Box** de la bibliothèque **PyNaCl**.

Cette librairie python nous permet de chiffrer et déchiffrer des messages à l'aide d'une clé secrète identique pour l'ensemble du groupe de discussion.

Un "nonce" est utilisé pour assurer l'unicité de chaque message chiffré et prévenir les attaques par réutilisation de clés.

Chaque client possède donc la clé symétrique des groupes auxquels il appartient.
Lors de la création d'un groupe, la clé symétrique est générée par l'administrateur (le créateur du groupe). Il la garde pour lui et la transmet lorsque le serveur lui soumet une requête de "demande" à rejoindre le groupe.

(c) Ce qu'il reste à faire !

Le projet manque toujours de messages **signés**. Nous devions utiliser la paire de clé RSA que dispose chaque client pour ajouter une couche de sécurité avec la signature RSA.

Ceci est une fonctionnalité qui n'a pu être livrée dans les temps mais que nous souhaiterions intégrer dans une version ultérieure.

Conclusion et Perspectives

9 Conclusion

Bien que le projet soit fonctionnel avec des scéanarios qui peuvent être testés de bout à bout, et avec une sécurité minimale, plusieurs améliorations peuvent être envisagées.

Tout d'abord, le projet ne comporte actuellement pas de composante pour tester comment réagirait notre système à des tentatives d'attaque concernant notre protocole de communication et de la cryptologie utilisée pour le sécuriser.
Nous envisagerons, lors de la présentation finale, d'en discuter malgré l'absence de notes dans ce rapport.

D'autres axes d'amélioration sont notables, comme intégrer des signatures RSA, tenter d'implémenter nous même une primitives de chiffrement symétrique comme PRESENT.

Concernant l'aspect fonctionnel, il nous manque certaines fonctionnalités que nous avions envisagé comme l'affichage de l'historique des messages lorsqu'un client arrive dans un groupe, ainsi que l'envoi de messages laissés "en attente" destinés à un client qui se reconnecte après un certain temps d'absence.

10 Figures et Liens

Document Canva pour illustrer notre protocole d'échange de données : `https://www.canva.com/design/DAGXBbY1c7I/NICZWVEpzDu46XDtHZWoFQ/view?utm_content=DAGXBbY1c7I&utm_campaign=designshare&utm_medium=link2&utm_source=uniquelinks&utlId=hb23c7780be`

Pour avoir l'interface graphique conçue à l'aide de figma : `https://www.figma.com/design/q2kvaDPG4YNI1hiwcr2ugg/Untitled?node-id=0-1&node-type=canvas&t=dNSDd36UtqXglvNp-0`

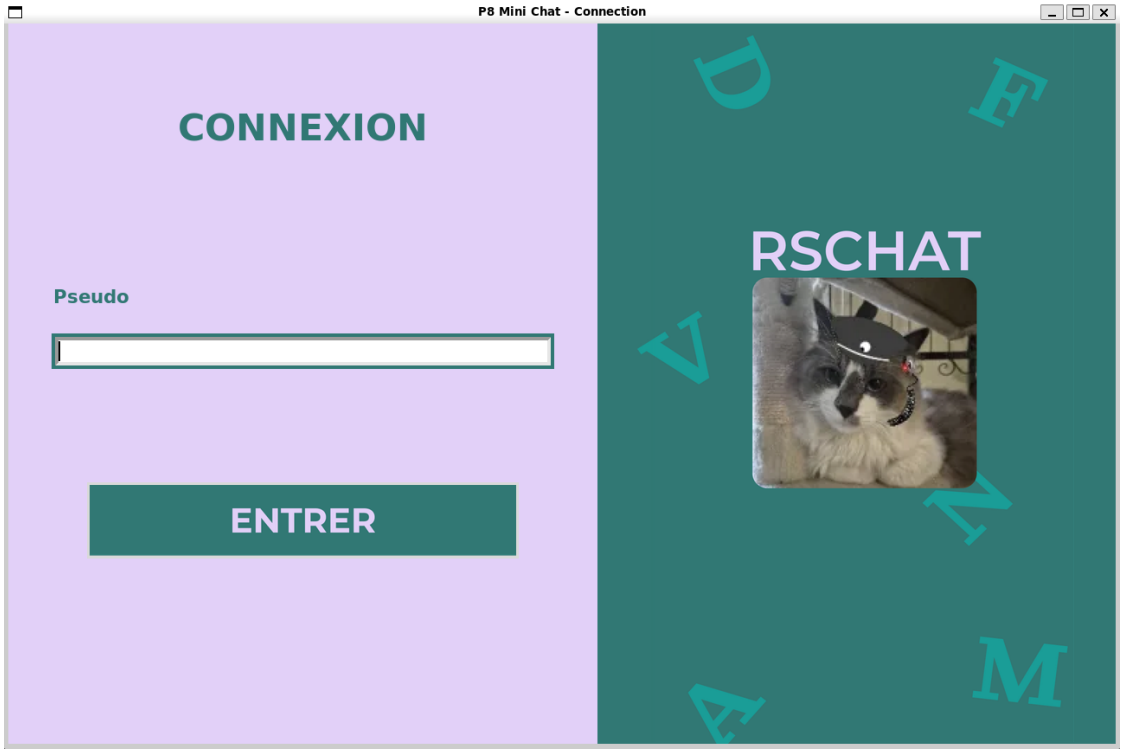


FIGURE 12 – Interface graphique de connexion