UFR-SEA Licence-Informatique

L3S5

Université Joseph Ki-Zerbo

Programmation Réseau :

Mise en Place d'un Serveur de communication Multi-Clients avec Interface console

Nom de l'enseignant :

- Dr. O BARRA

Membres du groupe :

- LANKOANDE Soumaila
- SAWADOGO Amidou
- SAWADOGO Loukouman

PLAN

Introduction Générale

- Contexte du projet
- Objectifs pédagogiques
- Présentation globale du travail réalisé

Partie I :Étude Théorique : Concepts de Programmation Réseau

1. Définition de la Programmation Réseau
2. Les Sockets en Python
3. Communication Multi-Client
4. Sécurité dans la Communication Réseau

Partie II :Réalisation Pratique : Système de Chat Sécurisé

1. Cahier des charges
2. Architecture du système
3. Mise en œuvre du Serveur
4. Mise en œuvre du Client
5. Module de Chiffrement
6. Captures d'écran et démonstration

Partie III : Documentation et Analyse

1. Manuel utilisateur
2. Analyse des problèmes rencontrés
3. Limitations et perspectives

Conclusion Générale

Annexes

- Code source
- Bibliothèques utilisées
- Documentation API si utilisée

Bibliographie / Références

- Cours ou livres utilisés
- Documentation Python
- Articles ou tutoriels utiles

Quelques abréviations en programmation réseaux

API: Application Programming Interface
ARPANET: Advanced Research Projects Agency Network
ARP: Address Resolution Protocol
BSD: Berkeley Software Distribution
BOOTP: Bootstrap Protocol
CORBA: Common Object Request Broker Architecture
DARPA: Defense Advanced Research Projects Agency
DNS: Domain Name System
DHCP: Dynamic Host Configuration Protocol
FDDI: Fiber Distributed Data Interface
FTP: File Transfer Protocol
HTTP: Hypertext Transfer Protocol
HTML: HyperText Markup Language
IP: Internet Protocol

JADE: Java Agent Development Framework
JRMP: Java Remote Method Invocation
JSP: JavaServer Pages
J2E : Java Platform, Enterprise Edition
JVM: Java Virtual Machine
JSON: JavaScript Object Notation
LAN: Local Area Network
MDA: Model Driven Architecture
NFS: Network File System
NTP: Network Time Protocol
OMG: Object Management Group
OSI: Open Systems Interconnection
PHP: Hypertext Preprocessor langage
RARP: Reverse Address Resolution Protocol
RMP: Remote Method Protocol
RPC: Remote Procedure Call

RMI: Remote Method Invocation
SDK: Software Development Kit
SMTP: Simple Mail Transfer Protocol
SOAP: Simple Object Access Protocol
TCP : Transmission Control Protocol
Telnet: Telecommunication network
UDP: User Datagram Protocol
WAN: Wide Area Network
W3C: World Wide Web Consortium
XDR: eXternal Data Representation
XML: Extensible Markup Language

Introduction Générale

Dans le cadre de l'unité d'enseignement « Programmation Réseau » du semestre 5 de la Licence 3 Informatique, il nous est demandé de mettre en pratique les concepts étudiés à travers la réalisation d'un projet concret. La communication entre machines, la gestion de connexions multiples, ainsi que les enjeux de la sécurité des échanges sont aujourd'hui au cœur des systèmes informatiques modernes, en particulier dans les applications de messagerie et de collaboration à distance.

Ce projet s'inscrit donc dans un contexte où la **maîtrise des protocoles réseau** et des **techniques de sécurisation des données** devient indispensable pour tout futur ingénieur ou développeur logiciel.

Les objectifs principaux de ce projet sont les suivants :

- Comprendre et manipuler les **sockets réseau en Python** pour établir une communication entre plusieurs machines.
- Mettre en œuvre une **architecture client-serveur multi-clients**, en utilisant des concepts de **programmation concurrente** (threads).
- Intégrer des **mécanismes de sécurité** dans les échanges, à travers un **chiffrement des données** (cryptographie symétrique).
- Appliquer une **méthodologie de développement logicielle complète**, incluant conception, implémentation, test et documentation.

Le projet a été divisé en deux grandes parties : une première partie **théorique**, où nous présentons les concepts fondamentaux de la programmation réseau et de la sécurité des communications, et une deuxième partie **pratique**, dans laquelle nous développons une application Python complète.

Cette application est un **système de chat en ligne de commande**, qui repose sur une architecture **client-serveur TCP**. Le **serveur** accepte plusieurs connexions simultanées,

chaque **client** peut envoyer des messages qui seront **chiffrés à l'envoi** et **déchiffrés à la réception** grâce à l'algorithme AES.

Le système a été conçu pour fonctionner via des terminaux (CMD ou PowerShell), sans interface graphique, en se concentrant sur les **fondamentaux de la communication réseau sécurisée**. L'ensemble du code est documenté, testé, et accompagné d'une analyse critique des limites du projet et des pistes d'amélioration possibles.

Partie I – Étude Théorique : Concepts de Programmation Réseau

1. Définition de la Programmation Réseau

La **programmation réseau**, ou **programmation des réseaux**, consiste à écrire du code informatique pour créer des programmes capables de communiquer sur un réseau(**LAN** ,**WAN**,**MAN**,…). Cela implique l'utilisation de langages de programmation (**python**, **java**, …), de bibliothèques et de protocoles spécifiques pour permettre l'échange de données entre différents systèmes informatiques. Dans le monde moderne, la plupart des applications logicielles (navigateur web, messagerie, jeux en ligne, services cloud…) reposent sur cette capacité à échanger des données à travers des réseaux informatiques.

Deux modèles dominent l'organisation des échanges sur un réseau :

- Le **modèle client-serveur** : un programme dit "client" initie une communication avec un programme "serveur" qui attend les requêtes et y répond.
- Le **modèle pair-à-pair (peer-to-peer)** : chaque programme agit à la fois comme client et serveur.

Le modèle client-serveur est celui utilisé dans ce projet.

2. Les Sockets en Python

Un socket en Python est un point d'extrémité dans une liaison de communication bidirectionnelle entre deux programmes fonctionnant sur le réseau. Il est utilisé pour envoyer et recevoir des données entre un client et un serveur sur un port spécifique.

Types de sockets :

- Socket TCP (SOCK_STREAM) : protocole fiable, orienté connexion (ex : HTTP, FTP, etc.)
- Socket UDP (SOCK_DGRAM) : protocole plus rapide, sans connexion, mais non fiable

Fonctionnement de base d'un socket TCP :

Voici le cycle typique de connexion :

Côté Serveur :

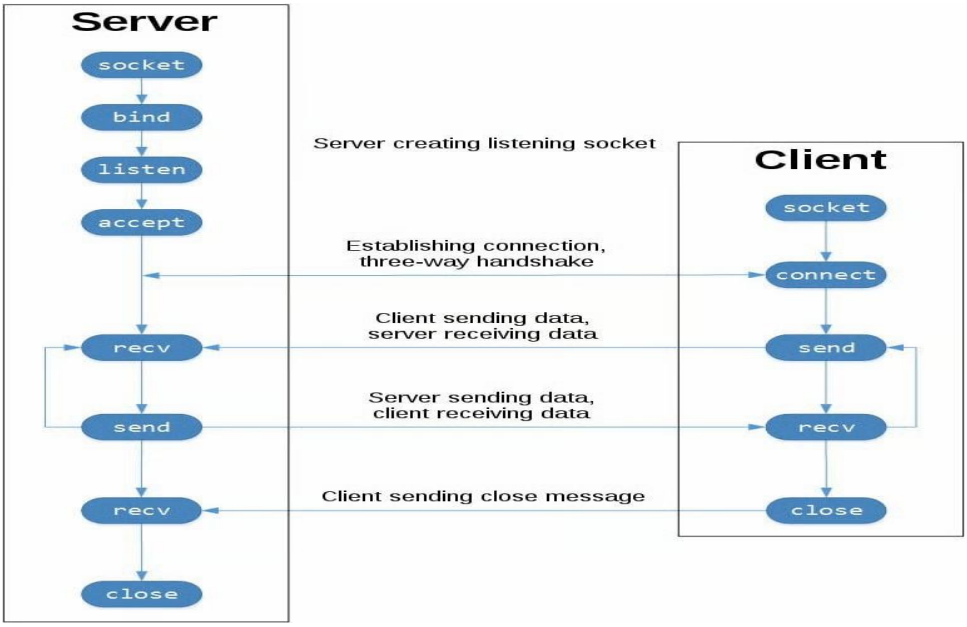
- `socket()` ⇨ Création du socket
- `bind()` ⇨ Attachement à une adresse IP et un port
- `listen()` ⇨ Mise en attente de connexions
- `accept()` ⇨ Acceptation d'une connexion entrante

Côté Client :

- `socket()` ⇨ Création du socket
- `connect()` ⇨ Connexion à un serveur

Ensuite, les deux côtés peuvent échanger des données avec :

- `send()` / `recv()` ⇨ pour envoyer et recevoir des données
- `close()` ⇨ pour fermer la connexion



3. Communication Multi-Client

Dans une architecture réseau de type client-serveur, le **serveur doit être capable de gérer plusieurs clients simultanément**. Il existe deux principales techniques en Python :

Le multi-threading :

- **Définition** :
  Le multi-threading (ou multithread en français, multi-fil d'exécution) est une technique de programmation qui permet à un programme **(**ou processus**)** de lancer plusieurs tâches **(**threads**)** en parallèle.

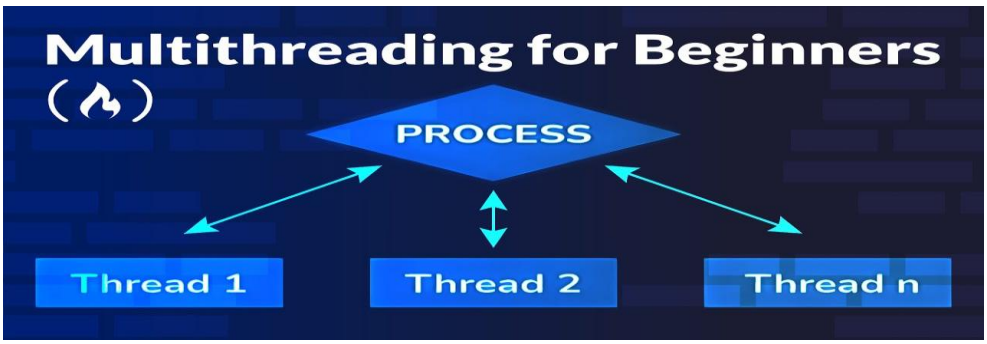
  Un thread est une unité légère d'exécution, qui fait partie d'un processus. Chaque thread peut s'exécuter indépendamment, tout en partageant la même mémoire du programme principal.

  Dans un programme serveur, chaque client peut être géré par un thread différent, ce qui permet d'éviter que l'un bloque les autres.

- **Utilité dans la programmation des sockets**

  - Servir plusieurs clients simultanément
  - Améliorer la réactivité du serveur
  - Séparer la logique de chaque client

- *Image :*



Le module `select` :

Permet au serveur de surveiller plusieurs sockets à la fois, sans créer de threads. Plus complexe, mais plus efficace pour des centaines de connexions.

Dans ce projet, nous utiliserons le **multi-threading** pour sa simplicité.

4. Sécurité et Chiffrement dans les Communications Réseau

Lorsqu'on transmet des données sur un réseau, il est important d'assurer la **confidentialité** et **l'intégrité** de ces données.

Les risques sans chiffrement :

- Interception des messages par des attaquants (attaque Man-In-The-Middle)
- Fuite de données sensibles (identifiants, messages privés)

Cryptographie symétrique : AES

Le **chiffrement symétrique** utilise **la même clé** pour chiffrer et déchiffrer les données.
L'algorithme **AES (Advanced Encryption Standard)** est une norme très utilisée.

Fonctionnement typique :

- Génération d'une clé secrète partagée (128, 192 ou 256 bits)
- Utilisation d'un vecteur d'initialisation (IV) pour sécuriser les blocs
- Chiffrement/déchiffrement via une librairie comme `pycryptodome`

Dans notre projet, chaque message est **chiffré côté client**, puis **déchiffré côté serveur**, ou inversement.

Partie II – Réalisation Pratique : Système de Chat Sécurisé

1. Cahier des charges

Le but du projet est de créer un système de messagerie **(chat)** sécurisé, basé sur une architecture client-serveur**.**
L'utilisateur exécute le serveur et les clients via des terminaux **(CMD** ou **PowerShell)**, sans interface graphique.

Fonctionnalités principales :

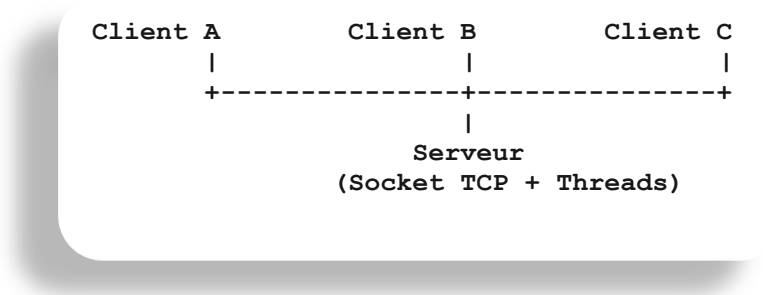
- Un **serveur** capable de gérer plusieurs clients simultanément (multi-threading)
- Des **clients** pouvant se connecter et échanger des messages entre eux
- Les **messages doivent être chiffrés** avant envoi et déchiffrés à la réception
- Communication uniquement en **ligne de commande**

Contraintes techniques :

- Langage : **Python**
- Protocole réseau : **TCP/IP (via `socket`)**
- Sécurité : **Chiffrement symétrique avec AES** (bibliothèque `pycryptodome`)
- Exécution via : **CMD ou PowerShell** (pas d'interface graphique)

2. Architecture du système

Schéma simplifié :



- `Chaque client établit une connexion TCP avec le serveur`
- Le serveur écoute et crée un thread par client
- Le chiffrement/déchiffrement **AES** s'effectue à chaque étape de communication

3. Mise en œuvre du Serveur

Fonctionnalités du serveur :

- Démarre en écoutant sur un port défini (ex: 5555)
- Accepte les connexions entrantes
- Crée un thread pour chaque client
- Reçoit les messages chiffrés, les déchiffre et les retransmet à tous les clients connectés (broadcast)

*Exemple de code – **`server.py`***

```python
import socket
import threading
from crypto_utils import AESCipher, generate_key

HOST = '127.0.0.1'
PORT = 5555
clients = []
client_names = {}
key = generate_key()
cipher = AESCipher(key)

def handle_client(client):
    name = None
    try:
        # Recevoir le nom du client
        name_msg = client.recv(4096)
        if not name_msg:
            return
        name = cipher.decrypt(name_msg).decode('utf-8')
        client_names[client] = name
        print(f"{name} a rejoint la conversation.")
        broadcast(cipher.encrypt(f"{name} a rejoint la conversation.".encode('utf-8')), client)

        while True:
            msg = client.recv(4096)
            if not msg:
                break

            decrypted_msg = cipher.decrypt(msg).decode('utf-8')
```

```python
            c.close()

def admin_commands():
    while True:
        cmd = input("Commande admin (/list, /kick <nom>, /quit): ")
        if cmd.lower() == '/quit':
            print("Arrêt du serveur...")
            for c in clients:
                c.close()
            os._exit(0)
        elif cmd.lower() == '/list':
            print("Clients connectés:")
            for name in client_names.values():
                print(f"- {name}")
        elif cmd.startswith('/kick '):
            name_to_kick = cmd[6:]
            for client, name in client_names.items():
                if name == name_to_kick:
                    try:
                        client.send(cipher.encrypt("Vous avez été expulsé par l'admin.".encode('utf-8')))
                        client.close()
                        print(f"{name} expulsé.")
                    except:
                        pass
                    break
            else:
                print(f"Client {name_to_kick} introuvable.")

def start_server():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((HOST,PORT))
    server.listen()

    print(f"Serveur démarré sur {HOST}:{PORT}")
    print(f"Clé partagée: {key.decode()}")
```

```python
            # Vérifier les commandes spéciales
            if decrypted_msg.lower() == '/quit':
                break

            print(f"{name}: {decrypted_msg}")
            broadcast(cipher.encrypt(f"{name}: {decrypted_msg}".encode('utf-8')), client)

    except Exception as e:
        print(f"Erreur avec {name if name else 'client inconnu'}: {e}")
    finally:
        if client in clients:
            clients.remove(client)
        if client in client_names:
            name = client_names[client]
            del client_names[client]
            print(f"{name} a quitté la conversation.")
            broadcast(cipher.encrypt(f"{name} a quitté la conversation.".encode('utf-8')), client)
        client.close()

def broadcast(msg, sender=None):
    for c in clients:
        if c != sender:
            try:
                c.send(msg)
            except:
                if c in clients:
                    clients.remove(c)
                if c in client_names:
                    del client_names[c]
                c.close()
```

```python
    # Démarrer le thread pour les commandes admin
    admin_thread = threading.Thread(target=admin_commands, daemon=True)
    admin_thread.start()

    try:
        while True:
            client, addr = server.accept()
            print(f"Connexion entrante de {addr}")
            clients.append(client)
            thread = threading.Thread(target=handle_client, args=(client,))
            thread.start()
    except KeyboardInterrupt:
        print("Arrêt du serveur...")
        for c in clients:
            c.close()
        server.close()

if __name__ == "__main__":
    start_server()
```

4. Mise en œuvre du Client

Fonctionnalités du client :

- Se connecte au serveur via IP et port
- Envoie des messages chiffrés
- Reçoit et affiche les messages des autres clients

Exemple de code – `client.py`

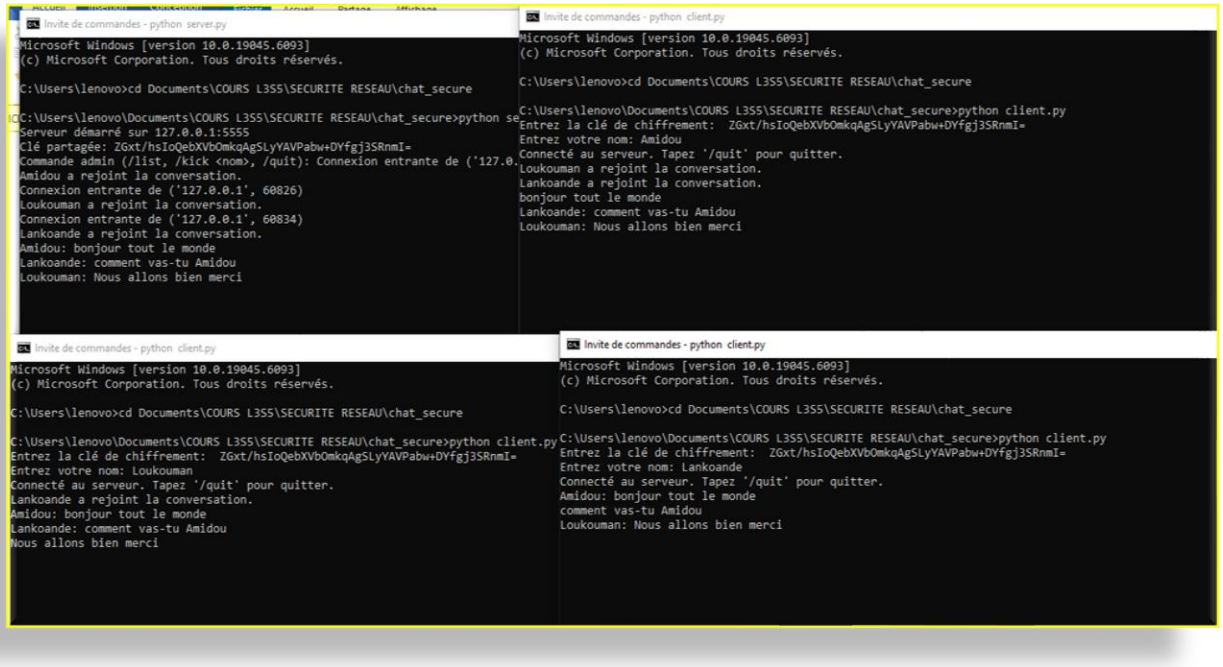
```python
import socket
import threading
from crypto_utils import AESCipher

HOST = '127.0.0.1'
PORT = 5555

def receive_messages(client, cipher):
    while True:
        try:...
        except:
            print("Erreur de connexion.")
            client.close()
            break

def start_client():
    key = input("Entrez la clé de chiffrement: ").encode('utf-8')
    cipher = AESCipher(key)

    name = input("Entrez votre nom: ")

    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        client.connect((HOST, PORT))
        # Envoyer le nom au serveur
        client.send(cipher.encrypt(name.encode('utf-8')))

        # Démarrer le thread pour recevoir les messages
        receive_thread = threading.Thread(target=receive_messages, args=(client, cipher))
        receive_thread.start()

        print("Connecté au serveur. Tapez '/quit' pour quitter.")
```

```python
        while True:
            msg = input()
            if msg.lower() == '/quit':
                client.send(cipher.encrypt('/quit'.encode('utf-8')))
                break
            client.send(cipher.encrypt(msg.encode('utf-8')))

    except Exception as e:
        print(f"Erreur de connexion: {e}")
    finally:
        client.close()
        print("Déconnecté.")

if __name__ == "__main__":
    start_client()
```

5. Module de chiffrement – `crypto_utils.py`

```python
ypto_utils.py > AESCipher > decrypt
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
import os
import base64

def generate_key():
    return base64.b64encode(os.urandom(32))

class AESCipher:
    def __init__(self, key):
        self.key = base64.b64decode(key)
        self.bs = AES.block_size

    def encrypt(self, raw):
        raw = pad(raw, self.bs)
        iv = os.urandom(self.bs)
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        return base64.b64encode(iv + cipher.encrypt(raw))

    def decrypt(self, enc):
        enc = base64.b64decode(enc)
        iv = enc[:self.bs]
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        return unpad(cipher.decrypt(enc[self.bs:]), self.bs)
```

6. Captures d'écran et Démonstration



- **Attendu :**

```
Serveur démarré sur 127.0.0.1:5555
Clé partagée :ZGxt/hsIoQ…==
Connexion entrante : ('127.0.0.1', 60826)
Loukouman a rejoint la conversation.
Connexion entrante : ('127.0.0.1', 60834)
Lankoande a rejoint la conversation.
Amidou :Bonjour tout le monde
Lankoande : Comment vas-tu Amidou ,Nous allons bien merci
```

Connexion de plusieurs clients

- **Commande :**

```
$ python client.py
```

- **Attendu :**

```
Connecté à 127.0.0.1:58372
Vous :
```

- **Dans un autre terminal (PowerShell ou CMD)** :

```
$ python client.py
```

Échange de messages

- **Client Soumaila envoie :**

```
Vous : Salut, je suis Lankoande soumaila étudiant en informatique L3S5 à
l'UJKZ et vous.
```

- **Client Amidou voit apparaître :**

```
Message reçu : Salut, je suis Lankoande Soumaila étudiant en informatique
L3S5 à L'UJKZ et vous.
```

- **Client Loukouman répond :**

```
Vous : Bienvenue Lankoande Soumaila! Moi c'est Loukouman Sawadogo
```

- **Client Soumaila voit :**

```
Message reçu : Bienvenue Lankoande Soumaila! Moi c'est Loukouman Sawadogo
```

Partie III – Documentation et Analyse

1. Manuel utilisateur

Installation préalable

- Installer Python 3.x
- Installer les dépendances :

```
$ sudo apt update
$ sudo apt install python3 python3-pip
$ pip install pycryptodome
```

Exécution

Ouvrir un terminal

Lancer le serveur :

```
$ python server.py
```

Dans un autre terminal, lancer un ou plusieurs clients :

```
$ python client.py
```

Les messages peuvent alors être échangés en toute sécurité.

2. Problèmes rencontrés

Problème	Cause	Solution
Blocage du serveur	Création excessive de threads	Utilisation d' un pool de threads
Message illisible	Mauvais encodage ou IV manquant	Envisager des techniques de multiplexage d'E/S comme **`select`** ou **`poll`**
Mauvais déchiffrement	Mauvais découpage du message (IV/CT)	Séparation propre dans **`decrypt`**`()`

3. Limitations et perspectives

Limitations actuelles :

- Pas d'authentification ni de gestion des utilisateurs
- La clé AES est codée en dur (pas d'échange sécurisé)
- Aucune persistance des messages
- Pas d'interface graphique

Améliorations possibles :

- Ajouter un système d'inscription/login
- Générer dynamiquement les clés avec échange sécurisé via Diffie-Hellman
- Stocker l'historique des messages
- Intégrer une interface graphique avec `Tkinter` ou `Kivy`
- Ajouter le chiffrement RSA pour un échange de clés sécurisé

Conclusion Générale

Ce projet a permis de mettre en œuvre les principes fondamentaux de la programmation réseau, notamment l'usage des sockets TCP, la gestion multi-client via les threads, et l'intégration de mécanismes de sécurité avec le chiffrement AES.

Au-delà des aspects techniques, ce projet développe également des compétences pratiques en débogage, structure de code, et documentation, essentielles pour tout développeur.

Annexes

Code source

L'ensemble du code source du projet est organisé de la manière suivante :

- `server.py` ⇨ Code du serveur (écoute, multi-clients, déchiffrement, redistribution)
- `client.py` ⇨ Code du client (connexion, envoi/réception chiffrés)
- `crypto_utils.py` ⇨ Module de chiffrement AES (cryptographie symétrique)

Structure du projet :

- `client.py`
- `server.py`
- `crypto_utils.py`
- `README.md`

Bibliothèques Python utilisées

Les bibliothèques suivantes n'ont pas besoin d'installation car elles sont intégrées à Python **:**

- **`socket`** : Module de base pour la programmation réseau (inclus dans Python)
- **`threading`** : Pour la gestion des connexions multiples (inclus)
- **`Crypto`** (via **`pycryptodome`**) : Pour le chiffrement **AES**
- **`get_random_bytes`**, **`pad`**, **`unpad`** : Utilitaires pour gérer les blocs **AES**
- **os** : Pour l'interaction avec le système d'exploitation.
- **Base64** :Pour le codage et le décodage des données en **Base64**
- **Sys :** Pour gérer les arguments ou quitter le programme

Documentation et Outils utilisés

- **Documentation** de **PyCryptodome**: *https://www.pycryptodome.org/src/cipher/aes*
- **Documentation du module socket** : *https://docs.python.org/3/library/socket.html*
- **Documentation du threading (multithreading Python)** *https://docs.python.org/3/library/threading.html*

Bibliographie / Références

Cours utilisés

- Support de cours "Sécurité de réseaux" – L3 S5 Informatique, Université Joseph Ki-Zerbo
- Cours de Dr. Ousmane Barra

Documentation Python

- Documentation officielle Python 3 : https://docs.python.org/fr/3/
- Module **`socket`** : *https://docs.python.org/3/library/socket.html*

Articles et tutoriels utiles

- Tutoriel Socket Python (RealPython) :
  https://realpython.com/python-sockets/
- Tutoriel AES PyCryptodome :
  https://nitratine.net/blog/post/python-encryption-and-decryption-with-pycryptodome/
- Tutoriel multiclient TCP :
  https://www.geeksforgeeks.org/python/socket-programming-multi-threading-python/

Real Python :