Assignment 1: Simple Client-Server Communication

Introduction

In this assignment, you will implement a basic client-server model using **TCP sockets** to facilitate communication between a server and multiple clients. The **server** will listen for incoming connections, process messages sent by clients, log these messages, and send an acknowledgment back to the respective client. The **client** will establish a connection with the server, send a user-provided message, and receive/display the server's response.

This project aims to introduce you to **socket programming**, **multi-threaded or asynchronous network handling**, and the **basics of encryption** in network communication. By working on this assignment, you will gain practical experience with **TCP/IP protocols**, **secure messaging**, and **software development best practices** in a collaborative environment using **GitHub**.

Guidelines

Project Requirements:

1. **GitHub Repository**:
    - Your project must be stored in a private repository on **GitHub**.
    - Proper **commit history** should be maintained, demonstrating progressive development.
2. **TCP Server Implementation:**
    - Listens on a specific **port**.
    - Accepts **multiple client connections**.
    - Receives messages from clients and **logs them**.
    - Sends an **acknowledgment** response back to the client.
    - Uses **basic encryption** for message transmission (e.g., OpenSSL, PyCrypto, or equivalent libraries for your chosen language).
3. **TCP Client Implementation:**
    - Connects to the **server**.
    - Sends a **user-provided message**.
    - Receives and **displays the acknowledgment** from the server.
    - Implements **basic encryption** to securely send the message.
4. **Error Handling:**
    - Implement **error detection** and proper handling for failed connections.
    - Ensure the server can gracefully handle **multiple clients** and unexpected disconnections.
5. **Multi-Threading/Async Handling:**
    - The server must support **multiple clients** using **multi-threading** or an **asynchronous approach**.

Project Structure and Submission:

- Include a **Makefile** with the following commands:
  - `make build` - Compile the project.
  - `make run` - Start the server and client.
  - `make clean` - Remove compiled files.
- Include a **README.md** file with:
  - Instructions to build and run the project.
  - Required libraries and installation steps.
  - Explanation of dependencies.
- Include a **Design Explanation Document** (`design_explanation.pdf` or `.md`):
  - Describe how the client and server communicate.
  - Explain the threading/async model chosen.
  - Provide an overview of how encryption is implemented.

Grading Rubric

Criteria	Points	Description
Simplicity & Cleanliness of Design	25	Code should be well-structured, modular, and easy to understand.
Documentation & Explanation	20	README.md and Design Explanation Document should clearly describe the implementation and setup.
Proper Use of GitHub & Make Utility	20	Clear commit history, correct Makefile usage, and repository structure.
Encryption Implementation	15	Messages should be securely transmitted using a basic encryption library.
Successful Execution Against Test Cases	20	The project should function correctly, handling multiple clients and expected errors.
Total	**100**	

Bonus (+5 points): Implement an additional encryption feature, such as a hashed message authentication (HMAC) or message integrity check.

Additional Notes:

- You may use any **programming language**, but ensure that the **required libraries** are clearly mentioned in the README file.
- Consider using **Wireshark** or equivalent tools to analyze and demonstrate encrypted traffic.
- Ensure your server does not crash when handling multiple clients simultaneously.

Deadline: [Insert Due Date]

By completing this assignment, you will enhance your understanding of **network programming**, **secure communications**, and **multi-client handling**, key concepts that are foundational for cybersecurity and software engineering. Good luck!