Design Specification

Project 3 – Salt Vault – Encrypted Client/Server using TweetNaCl

1. Overview

This project involves developing a secure client-server application that enables encrypted file transfers. The client (written in Python) uploads files of arbitrary length to a C-based server while ensuring **Confidentiality, Integrity, and Authentication (CIA triad)** using **TweetNaCl** on the server side and **PyNaCl** on the client side. The system must also support **forward secrecy** by using **ephemeral session keys**.

---

2. Functional Requirements

2.1 Client (Python)

- Provides a **Command-Line Interface (CLI)** using `cmd` or `cmd2` for interactive operation.
- Uses **PyNaCl** for cryptographic operations, ensuring data is encrypted before transmission.
- Supports **large file uploads** by breaking them into encrypted chunks.
- Displays a **progress bar** using `tqdm` to show the upload progress in bytes.
- Establishes a **secure connection** to the server and performs a handshake to derive session keys.
- Can use a **configuration tool** for pre-sharing authentication challenges or key material.

2.2 Server (C)

- Accepts incoming **secure connections** from multiple clients.
- Uses **TweetNaCl** for cryptographic operations (encryption, authentication).
- Supports **secure storage** of received files, ensuring integrity and confidentiality.
- Implements **session key derivation** for forward secrecy.
- Efficiently handles large file transfers by **decrypting incoming encrypted chunks**.
- Logs connection attempts and successful file transfers.

---

3. Security Requirements

1. **Confidentiality** – All data transferred between the client and server must be encrypted using **authenticated encryption** (e.g., `crypto_secretbox` in NaCl).
2. **Integrity** – Each encrypted chunk must include an **authentication tag (MAC)** to detect tampering.
3. **Authentication** – The server must verify the client's legitimacy before accepting file uploads.
4. **Forward Secrecy** – Each session must use **ephemeral session keys**, preventing exposure of past communications if a long-term key is compromised.
5. **Pre-shared Authentication or Key Agreement** – The client and server must establish a secure key **before transmission**, either through a challenge-response mechanism or a Diffie-Hellman exchange.

---

4. System Architecture

4.1 Client (Python)

- **CLI Interface** (`cmd`/`cmd2`) → Reads the filename input.
- **Configuration Module** → Handles pre-shared authentication keys or challenge-response.
- **Encryption Module (PyNaCl)**
  - Encrypts file data in chunks.
  - Generates authentication tags.
- **File Transfer Module**
  - Reads file in **fixed-size blocks**.
  - Sends encrypted chunks to the server.
  - Displays upload progress using `tqdm`.
- **Networking Module (Sockets)**
  - Establishes **secure session** with the server.
  - Handles key exchange and authentication.

4.2 Server (C)

- **Connection Handler**
  - Listens for incoming connections.
  - Verifies client authentication.
- **Session Key Management**
  - Uses **ephemeral session keys** for encryption.
- **Decryption Module (TweetNaCl)**
  - Verifies integrity and authenticity of received data.
  - Decrypts file chunks.
- **File Storage Module**
  - Stores received files securely.

---

5. Communication Protocol

5.1 Secure Handshake

1. **Client Initiates Connection**
   - Sends a client hello with a **public key** or a challenge response.
2. **Server Responds**
   - Server verifies client authentication.
   - Generates **ephemeral session key**.
3. **Key Exchange**
   - Both sides agree on a **shared session key**.
4. **Confirmation**
   - Client encrypts a nonce + test message and sends it.
   - Server decrypts and verifies.

5.2 Encrypted File Transfer

1. **Client reads the file** and encrypts **fixed-size chunks**.
2. **Each chunk contains:**

    - Encrypted data
    - Authentication tag
    - Sequence number (for ordering)
3. **Server receives and verifies** each chunk.
4. **Decrypted data is stored securely**.

---

6. Error Handling & Resilience

- **Network Failures** – Implement **timeouts and retries**.
- **Authentication Failures** – Reject unauthorized clients.
- **File Integrity** – Validate with **authentication tags**.
- **Concurrency** – Server should handle **multiple clients** concurrently.

---

7. Tools & Libraries

Component	Technology
Encryption (Client)	PyNaCl
Encryption (Server)	TweetNaCl
CLI	Python `cmd` / `cmd2`
Progress Bar	`tqdm`
File Transfer	Python Sockets (TCP)
Server	C (Sockets + Threads)

---

8. Deliverables

- **Python Client:** Secure CLI-based file uploader.
- **C Server:** Encrypted file receiver.
- **Configuration Tool:** Optional key-sharing mechanism.
- **Documentation:** README with setup & usage instructions.