**High-Level Design Document**

**Privacy Preserving Quantum Secure Federated Learning Framework based on Multikey CKKS Homomorphic Encryption**

**Version :** 0.1

**Date :** 03rd Sept 2025

**Authors :** Pranali Nikam, Prabhat Karlekar, Siddhant Bopche

**Mentors :** Puneet Bakshi, Vinodh Kumar M.

**Department :** HPC I&E, C-DAC Pune

1. **Introduction**

Large-scale systems and distributed environments produce vast amounts of data from various sources such as devices, applications, and monitoring components. Directly sharing this raw information across different locations raises significant privacy, security, and regulatory challenges, making conventional centralized data processing or machine learning approaches less suitable. To address these challenges, this project integrates Federated Learning (FL), Homomorphic Encryption (HE), and High-Performance Computing (HPC) into a unified framework:

Federated Learning (FL): Enables multiple data centers to collaboratively train machine learning models without moving raw data outside their local environment. Each client trains locally and only shares model updates.

Homomorphic Encryption (HE): Ensures that all shared model updates remain encrypted end-to-end. The central server can perform aggregation on encrypted weights, preserving confidentiality while still producing a usable global model.

High-Performance Computing (HPC): Provides the computational backbone to scale federated training across multiple nodes.

The architecture ensures:

* Client data never leaves its source in plaintext.
* Aggregation is done on encrypted weights.
* HPC cluster resources handle the scale of computation.

Together, these technologies create a secure, scalable, and privacy-preserving federated training environment suitable for modern data centers.

1. **Purpose**

To enable privacy-preserving training of machine learning model (time-series) on distributed data using federated learning.

1. **Objectives**

* Federated Learning: Local training on client nodes, global aggregation on the server.
* Privacy via Homomorphic Encryption (OpenFHE): Encrypting model updates, enabling secure aggregation.
* Orchestration Layer: Automating workflow, round-based execution, logging.
* Kafka Integration: Streaming telemetry data ingestion.
* HPC Utilization: Running clients, server, and orchestrator processes on distributed worker nodes.

1. **Requirements**

|  |  |
| --- | --- |
| Type | Requirement |
| Functional | 1. Clients must train GRU/LSTM models on local infrastructure data. 2. Clients must encrypt model weights before transmission using homomorphic encryption. 3. Server must support model aggregation via MK-CKKS-HE /FedAvg. 4. Server must re-encrypt ciphertexts to target the client's domain using PRE. 5. Aggregated model weights must be sent back to clients for local retraining. 6. Clients compute metrics (e.g., MSE, R²) after each round. |
| Security | 1. Raw infrastructure data must never leave client premises. 2. Keys and ciphertexts must be securely exchanged and stored. |
| Non-Functional | 1. Model convergence check must be built-in to stop redundant training rounds. |

1. **High-Level Architecture**

The architecture illustrates a centralized Federated Learning (FL) system in which multiple clients collaboratively train a global model under strong privacy guarantees using Homomorphic Encryption (HE) and Proxy Re-Encryption (PRE).

1. Clients

Each client represents an independent data holder, such as a data center, organization, or node in an HPC cluster.

Key Components per Client:

* Data: Private, sensitive datasets never leave the client machine in plaintext.
* Training (Local GRU/LSTM):
  + Each client trains its own local model (e.g., GRU/LSTM) on its data.
  + Produces local model weights w₁ⁿ, w₂ⁿ after each training round.
* Encrypt Weights:
  + The updated local weights are encrypted using the client’s public key (Pb₁, Pb₂).
  + Example: Enc₁(w₁ⁿ), Enc₂(w₂ⁿ).
* Decrypt Module:
  + After aggregation, each client decrypts the received global aggregated weights using its private key (Pr₁, Pr₂).
  + Example: Dec₁(Enc₁(waggⁿ)).

No raw data or plaintext weights are ever shared outside the client boundary.

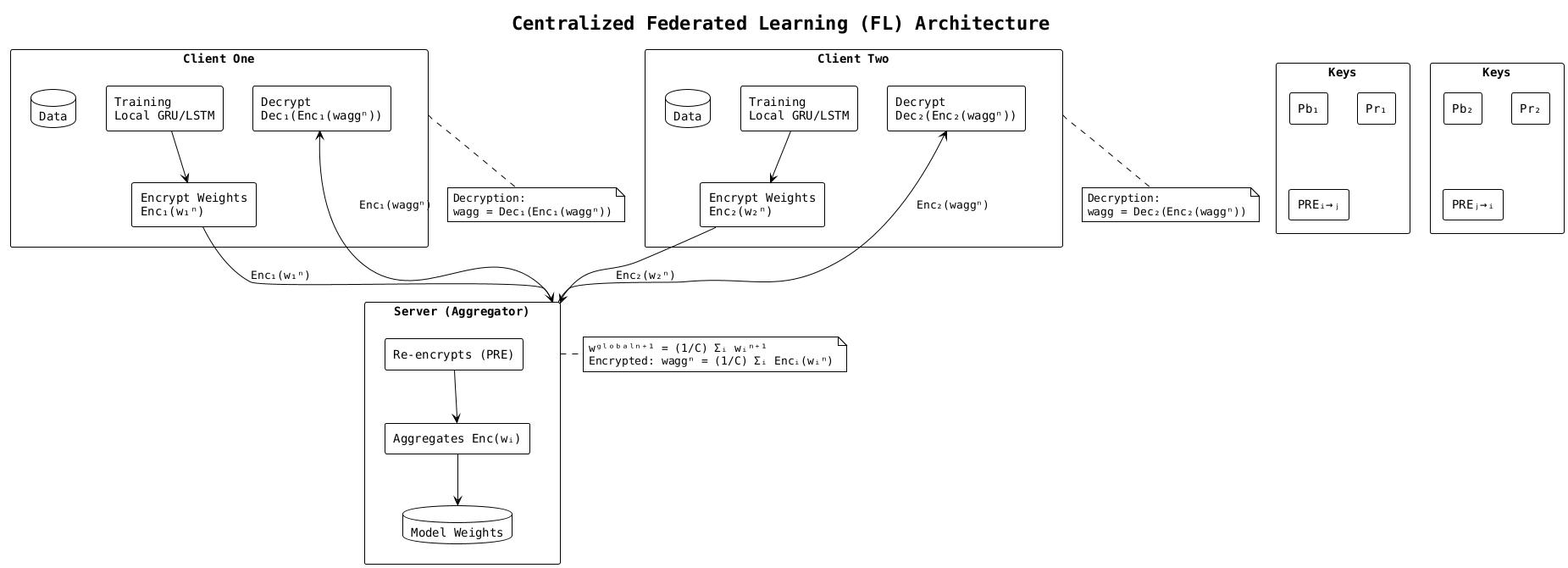


Figure 1: High-Level System Architecture

1. Keys & Proxy Re-Encryption (PRE)

Each client has a key pair:

* Public key (Pb) for encryption.
* Private key (Pr) for decryption.

Additionally, Proxy Re-Encryption (PRE) keys (PREᵢ→ⱼ, PREⱼ→ᵢ) allow the server to transform ciphertexts encrypted under one client’s key into another’s key domain without seeing the plaintext.

* Example: The server can re-encrypt Enc₁(w₁ⁿ) into a form that can be aggregated with Enc₂(w₂ⁿ).

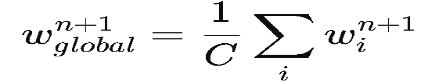
This step is crucial for enabling multi-client aggregation while keeping keys isolated.

1. Server (Aggregator)

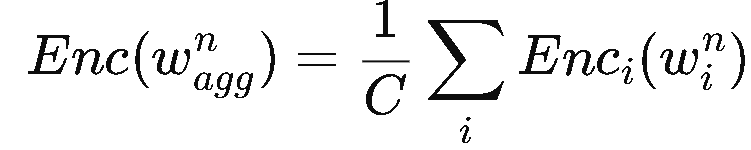
The central server coordinates FL rounds but never sees any plaintext data or weights.

Server Workflow

1. Receive Encrypted Updates:
   * Collects encrypted model updates from clients (Enc₁(w₁ⁿ), Enc₂(w₂ⁿ)).
2. Re-Encryption (PRE):
   * Uses proxy re-encryption keys to align ciphertexts into a common key space.
   * Ensures all encrypted weights are compatible for aggregation.
3. Aggregation on Ciphertexts:
   * Performs secure aggregation:



under encryption:



1. Distribute Global Encrypted Model:
   * Sends the aggregated encrypted global weights back to each client.
2. **High-Level Workflow (One Round)**
3. Orchestrator generates **CC.json** and distributes cryptographic keys.
4. Clients train local GRU/LSTM models on private telemetry data.
5. Local weights are **encrypted with OpenFHE**.
6. Server collects ciphertexts, applies PRE, and aggregates them homomorphically.
7. Aggregated encrypted model is redistributed to clients.
8. Client’s decrypt using their private keys and update their models.
9. Logs and round files are stored in **client-private directories.**

This process repeats until the global model converges.

1. **Technologies Used**

This project combines cryptography, machine learning, orchestration, and HPC resources to enable secure and scalable federated learning.

* + - 1. **OpenFHE (C++): Homomorphic encryption libraries**

Provides CKKS-based encryption and proxy re-encryption for securing model weights. Integrated into client (encrypt/decrypt) and server (aggregation, domain change) workflows.

* + - 1. **TensorFlow / Keras (Python) – Deep Learning Framework**

Implements GRU/LSTM models for time-series telemetry. Handles local training (c\_trainAndUpdate.py), evaluation, and weight export before encryption.

* + - 1. **Apache Kafka (Python, Java) – Pub/Sub Data Ingestion**

Reliable publish–subscribe middleware to stream telemetry data from distributed clients into local storage for training.

* + - 1. **Mongoose HTTP Server (C) – Lightweight Server**

Facilitates encrypted model weight exchange between clients and server via simple HTTP endpoints.

* + - 1. **Python + Shell Scripts – Orchestration Layer**

Automates FL workflow (round setup, execution, logging, key exchange) across nodes using SSH/MPI. Ensures repeatable experiments and convergence tracking.

* + - 1. **High-Performance Computing (HPC) Cluster**

Distributed multi-node infrastructure for running clients, server, orchestrator, and Kafka services. Enables scaling experiments and parallel execution.