**Test Document**

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

**Version :** 0.1 **Date :** 03rd Sept 2025

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

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

**I. Overview**

This project leverages OpenFHE for cryptographic operations within a federated learning framework. To ensure reliability and robustness, we use **GoogleTest** for automated unit and integration testing of the C++ components.

**Unit Test Cases**

|  |  |  |
| --- | --- | --- |
| **Test Module** | **Test Case** | **Description** |
| genCC (test\_s\_CC.cpp) | ConfigFileExists | Verify that config\_cc. json exists at the expected location. |
| SchemaHasAllKeys | Verify that runtime config has required keys: MultiplicativeDepth, ScalingModSize, BatchSize, PREMode. |
| MultiplicativeDepthValid | Verify that depth ∈ [1, 20]. Prevents runtime blow-up. |
| ScalingModSizeValid | Verify that scaling modulus ∈ (30, 100). Prevents inefficiency or invalid crypto context. |
| BatchSizeValid | Verify that Batch size > 0 and ≤ 8192. Prevents invalid/inefficient training. |
| PREModeValid | Verify that PREMode must be INDCPA or INDCCA. Any other value is invalid. |
| BinaryExists | Verify that compiled genCC binary exists. |
| CCFileExists | Verify that genCC generated CC.json exists |
| CCFileNotEmpty | Verfiy that CC.json is not empty and is readable. |
| keyGen (test\_c\_keyGen.cpp | |  | | --- | | Client1ConfigExists |  |  | | --- | |  | | Verify that ConfigFile\_Client1 exists. |
| Client2ConfigExists | |  | | --- | |  |  |  | | --- | | Verify that ConfigFile\_Client2 exists. | |
| Client1SchemaHasAllKeys | Verify that client config contains CC\_PATH, PUBKEY\_PATH, PRIVKEY\_PATH. |
| Client2SchemaHasAllKeys | Verify that client config contains CC\_PATH, PUBKEY\_PATH, PRIVKEY\_PATH. |
| KeyGenBinaryExists | Verify that keyGen binary is present |
| PubKeyFileExists | Verify that public key file exists after run. |
| PrivKeyFileExists | Verify that private key file exists after run. |
| PubKeyFileNotEmpty | Ensure public key file is non-empty. |
| PrivKeyFileNotEmpty | |  | | --- | | Ensure private key file is non-empty. |  |  | | --- | |  | |
| PubKeyParseableJSON | |  | | --- | | Validate public key file parses as valid JSON. |  |  | | --- | |  | |
| PrivKeyParseableJSON | |  | | --- | | Validate private key file parses as valid JSON. |  |  | | --- | |  | |
|  |  |  |

**Note:** Similar structured tests are implemented for:

* REkeyGen
* encryptModelWeights
* changeCipherDomain
* aggregateEncryptedWeights
* decryptModelWeights

**II. Testing Philosophy**

* Separation: Tests are separate of orchestration (run.sh).
* Granularity: Modular targets (make test\_cpp, make test\_integration, etc.).
* Test Types:
* Unit tests → binaries, config params, JSON schema validation.
* Integration tests → full workflow (keygen, training, aggregation) using test\_run.sh

**III. Directory Structure**

Tests are organized in a dedicated test/ directory to maintain clarity and separation from production code:

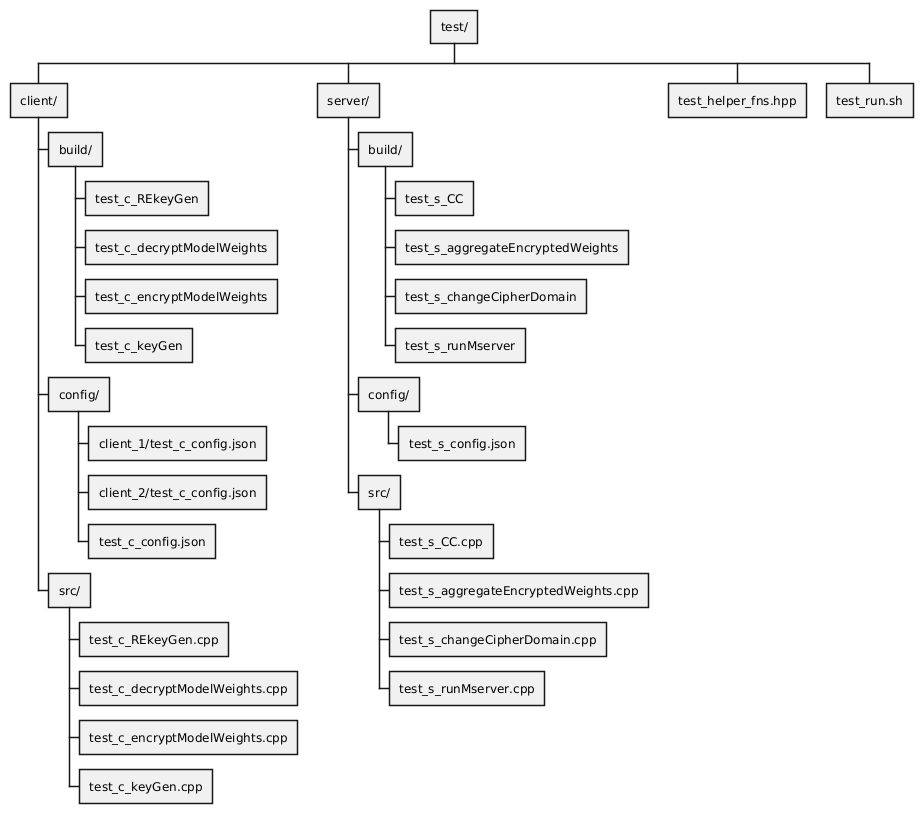
* **test/src/**: Contains C++ unit test source files (e.g., test\_s\_CC.cpp).
* **test/**: Contains integration test scripts (e.g., test\_s\_CC.sh, test\_run.sh).
* **test/build/**: Stores compiled test binaries (e.g., test/build/test\_s\_CC).
* **utils/**: Houses utility functions (e.g., helper\_fns.hpp for file existence checks).

Figure 1: Directory Structure

**IV. GoogleTest Usage**

GoogleTest is used for unit testing C++ components, leveraging its robust assertion mechanisms and test fixture support. Key aspects:

* ASSERT\_: Stops test if condition fails (e.g., for preconditions like file existence).
* EXPECT\_: Logs failure but continues (e.g., for output validation).
* Current Tests:
  + Precondition checks (ASSERT\_TRUE for directories, binaries, configs).
  + Functional checks (EXPECT\_TRUE for outputs, JSON parsing).
  + Output validation (check file existence, non-empty, valid JSON).
  + Pipeline simulation (CLI args, dry-runs).
  + Postcondition checks (verify created/used files).

**V. Test Execution**

Tests are executed via the Makefile and test\_run.sh:

|  |  |
| --- | --- |
| **Command** | **What It Does** |
| make test\_all | Runs all unit tests. |
| make test\_cpp | Compiles and runs C++ tests in test/src/ (e.g., test\_s\_CC). |
| test\_run.sh | End-to-end workflow test |

**VI. GoogleTest Installation**

To ensure consistent builds, we use a system-installed GoogleTest. This simplifies dependency management and avoids issues with missing libraries (e.g., libgtest.a, libgtest\_main.a).

**1. Installation Steps (Ubuntu/Debian)**

1. **Install Dependencies**:

sudo apt-get update

sudo apt-get install -y libgtest-dev cmake g++

**2. Build GoogleTest Libraries**: The libgtest-dev package provides source files only. Build and install the libraries:

cd /usr/src/gtest

sudo cmake .

sudo make

sudo cp lib/\*.a /usr/lib

**3. Verify Installation**: Check for libgtest.a and libgtest\_main.a:

ls -l /usr/lib/libgtest\*.a

Ensure headers are available:

ls -l /usr/include/gtest/gtest.h

**VII. Makefile Updates**

The Makefile is updated to support the new test structure and system-installed GoogleTest. Key changes:

* Test directories: test/client/, test/server/, test/build/.
* GoogleTest linking: Use -lgtest -lgtest\_main -pthread.
* Targets: test, test\_cpp, test\_run.

**VIII. Future Directions**

* Expand integration tests with controlled fixture data.
* Add Python-based testing (pytest) for c\_trainAndUpdate.py.