**Installation & Dependencies**

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

**Version :** 0.1

**Date :** 2nd September 2025

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

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

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

This document provides step-by-step instructions for installing and configuring the dependencies required for the Privacy Preserving Quantum Secure FL Framework based on MultiKey CKKS HE. The setup covers system libraries, OpenFHE, Mongoose server, client dependencies, and Kafka components.

# 1. System Preparation (All VMs: Producer, Broker, Consumer)

# 

A. Update system packages:

sudo apt update && sudo apt upgrade -y

B. Install required tools and libraries:

sudo apt install -y openjdk-11-jdk wget curl net-tools python3 python3-pip unzip

build-essential cmake git libgmp-dev libmpfr-dev libmpc-dev

libssl-dev jq libcurl4-openssl-dev

C. Verify versions:

java -version

python3 –version

# 2. Kafka Setup (Broker, Producer, Consumer)

# 

## 2.1 Broker VM

2.1.1 Download and extract Kafka

wget https://downloads.apache.org/kafka/3.7.2/kafka\_2.13-3.8.0.tgz

tar -xzf kafka\_2.13-3.8.0.tgz

2.1.2 Create log/data dirs.

mkdir -p ~/kafka/logs ~/kafka/kafka-logs ~/kafka/zookeeper-data

2.1.3 Back up configs (safety)

cd ~/kafka/kafka\_2.13-3.8.0

cp config/server.properties config/server.properties.bak

cp config/zookeeper.properties config/zookeeper.properties.bak

2.1.4 Configure Kafka & ZooKeeper (simple and safe: append overrides)

cd ~/kafka/kafka\_2.13-3.8.0

cat >> config/server.properties <<'EOF'

# ---- appended for multi-VM setup ----

broker.id=0

listeners=PLAINTEXT://0.0.0.0:9092

advertised.listeners=PLAINTEXT:// 0.0.0.0:9092

listener.security.protocol.map=PLAINTEXT: PLAINTEXT

log.dirs=/home/$USER/hpcie/kafka/kafka-logs

zookeeper.connect=0.0.0.0:2181

# ---- end appended ----

EOF

cat >> config/zookeeper.properties <<'EOF'

# ---- appended to place zookeeper files under hpcie ----

dataDir=/home/$USER/kafka/zookeeper-data

clientPort=2181

# ---- end appended ----

EOF

2.1.5 Check port 2181 not already used (important)

sudo lsof -i :2181

2.1.6 If some process is already on 2181

sudo kill -9 <PID>

2.1.7 Start ZooKeeper (background, log saved)

cd ~/kafka/kafka\_2.13-3.8.0

nohup ./bin/zookeeper-server-start.sh config/zookeeper.properties > ~/hpcie/kafka/logs/zookeeper.log 2>&1 &

sleep 3

tail -n 80 ~/hpcie/kafka/logs/zookeeper.log

2.1.8 Start Kafka broker (background)

nohup ./bin/kafka-server-start.sh config/server.properties > ~/kafka/logs/kafka.log 2>&1 &

sleep 4

tail -n 120 ~/hpcie/kafka/logs/kafka.log

2.1.9 Create topics (run on broker)

cd ~/kafka/kafka\_2.13-3.8.0

./bin/kafka-topics.sh --create --bootstrap-server 0.0.0.0:9092 \ --replication-factor 1 --partitions 3 --topic client1-data

./bin/kafka-topics.sh --create --bootstrap-server 0.0.0.0:9092 \ --replication-factor 1 --partitions 3 --topic client2-data

2.1.10 Verify

./bin/kafka-topics.sh --list --bootstrap-server 0.0.0.0:9092

./bin/kafka-topics.sh --describe --bootstrap-server 0.0.0.0:9092 --topic client1-data

## 2.2 Producer VM

2.2.1 Make producer workspace & install Python libs

mkdir -p ~/producer/data ~/producer/logs

cd ~/producer

# install libs for current user

python3 -m pip install --user kafka-python pandas

export PATH=$PATH:~/.local/bin

2.2.2 Create producer.py

~/producer/producer.py

2.2.3 Test connectivity to broker

nc -vz 0.0.0.0 9092

# or

telnet 0.0.0.0 9092

2.2.4 Run producer - From 0.0.0.0

cd ~/producer

python3 producer.py 2>&1 | tee ~/producer/logs/producer.log

## 2.3 Consumer VM:

1. Prepare workspace & install libs

mkdir -p ~/consumer/data ~/consumer/logs

cd ~/consumer

python3 -m pip install --user kafka-python pandas

export PATH=$PATH: ~/. local/bin

2. Create consumer\_client1.py and consumer\_client2.py

~/consumer/consumer\_client1.py

~/consumer/consumer\_client2.py

3. Run consumers

nohup python3 consumer/consumer\_client1.py > consumer/logs/client1.log 2>&1 &

nohup python3 consumer/consumer\_client2.py > consumer/logs/client2.log 2>&1 &

4. Check logs

tail -f consumer/logs/client1.log

tail -f consumer/logs/client2.log

# 3. OpenFHE Installation, Compilation, and Client Setup

This section covers the installation and compilation of OpenFHE for both server and client environments. OpenFHE is a C++ library for homomorphic encryption, supporting schemes like CKKS for approximate computations. The instructions focus on Linux/Ubuntu.

3.1 Prerequisites and Dependencies

Install the following tools and libraries required for OpenFHE and related tasks:

sudo apt update

sudo apt install -y git cmake build-essential libgmp-dev libmpfr-dev libmpc-dev libssl-dev libntl-dev curl

* Explanation:
  + git: Clone source repositories.
  + cmake, build-essential: Build tools for compiling OpenFHE.
  + libgmp-dev, libmpfr-dev, libmpc-dev: Support multiprecision arithmetic (optional but recommended for NTL backend).
  + libssl-dev: Cryptographic operations.
  + libntl-dev: Optional for high-performance NTL backend (used for multiprecision arithmetic).
  + curl: Fetch files via HTTP (clients only).

Optional for scripting or testing:

sudo apt install -y python3 python3-pip

3.2 Installing OpenFHE (Server and Clients)

Step 1: Clone and Build OpenFHE

git clone https://github.com/openfheorg/openfhe-development.git

cd openfhe-development

mkdir build && cd build

cmake .. -DWITH\_NATIVEOPT=ON -DBUILD\_SHARED=ON -DWITH\_OPENMP=ON -DCMAKE\_BUILD\_TYPE=Release

make -j$(nproc)

sudo make install

* CMake Flags:
  + -DWITH\_NATIVEOPT=ON: Enables machine-specific optimizations.
  + -DBUILD\_SHARED=ON: Builds shared libraries (use -DBUILD\_STATIC=ON for static libraries).
  + -DWITH\_OPENMP=ON: Enables multi-threading (default).
  + -DWITH\_NTL=ON: Optional NTL backend (requires libntl-dev).
  + -DWITH\_TCM=ON: Optional tcmalloc for better performance (requires libgoogle-perftools-dev).
  + No explicit flag needed for CKKS (included by default).
  + Serialization uses bundled Cereal (no separate flag needed).
* Installation Paths:
  + Headers: /usr/local/include/openfhe
  + Libraries: /usr/local/lib

Step 2: Set Library Path

Ensure the system can find OpenFHE libraries:

export LD\_LIBRARY\_PATH=/usr/local/lib:$LD\_LIBRARY\_PATH

echo "export LD\_LIBRARY\_PATH=/usr/local/lib:$LD\_LIBRARY\_PATH" >> ~/.bashrc

Step 3: Verify Installation

ls /usr/local/include/openfhe

ls /usr/local/lib | grep openfhe

3.3 Compiling OpenFHE Applications

To compile an application using OpenFHE (e.g., your\_code.cpp), use:

g++ your\_code.cpp -o your\_program -std=c++17 -I/usr/local/include/openfhe -L/usr/local/lib -lopenfhecore -lopenfhepke -lopenfhebinfhe -fopenmp -lpthread

**Notes**:

* Add -lopenfhehexl if Intel HEXL optimizations are enabled (requires -DWITH\_INTEL\_HEXL=ON in CMake).
* For custom build paths, update -I and -L flags, e.g.:

-I/path/to/openfhe-development/src/core/include \

-I/path/to/openfhe-development/src/pke/include \

-L/path/to/openfhe-development/build/lib

* For static linking, use:

-Wl,--whole-archive -lOPENFHEpke -Wl,--no-whole-archive

3.4 Creating and Sharing a CKKS CryptoContext in JSON

OpenFHE supports serializing the CryptoContext to JSON for sharing encryption parameters across systems.

Why JSON?

* Platform-agnostic and human-readable.
* Simplifies debugging and cross-language compatibility.
* Allows clients to load the same encryption context without regenerating parameters.

Serialization Library

OpenFHE uses the bundled **Cereal** library for JSON serialization (not RapidJSON). No external JSON library is needed unless custom JSON handling is required (e.g., nlohmann/json).

Optional: Install nlohmann/json for advanced JSON processing:

git clone https://github.com/nlohmann/json.git

cd json

mkdir build && cd build

cmake ..

make -j$(nproc)

sudo make install

Explanation:

* Sets CKKS parameters: multDepth for computation levels, scaleModSize for precision, batchSize for vector packing.
* Enables features: PKE (public-key encryption), KEYSWITCH, and LEVELEDSHE (for CKKS operations).
* Serializes the context to cryptocontext.json using Cereal’s JSON archive.
* Clients can deserialize using cereal::JSONInputArchive to ensure compatible encryption/decryption.

Compile and run:

g++ cc.cpp -o cc\_program -std=c++17 -I/usr/local/include/openfhe -L/usr/local/lib -lopenfhecore -lopenfhepke -fopenmp -lpthread

./cc\_program

3.5 Server-Side Setup with Mongoose

Use **Mongoose**, a lightweight C-based HTTP server, to serve cryptocontext.json and handle client uploads (e.g., public keys or ciphertexts).

Step 1: Download Mongoose

cd /path/to/project/folder

wget https://raw.githubusercontent.com/cesanta/mongoose/master/mongoose.c

wget https://raw.githubusercontent.com/cesanta/mongoose/master/mongoose.h

Step 2: Adjust Buffer Size (Optional)

For large uploads (e.g., ciphertexts), edit mongoose.h:

#define MG\_MAX\_RECV\_SIZE (20UL \* 1024UL \* 1024UL) *// 20 MB*

Step 3: Compile and Run Server

Example server code (server.cpp) to serve files and handle POST requests (implement endpoints like /upload for client uploads). Compile:

g++ server.cpp mongoose.c -o server -pthread -std=c++17

./server

* Serves on port 8000 by default.
* Access files: http://[server-ip]:8000/cryptocontext.json
* Change port: ./server -listening\_port 8080

3.6 Client-Side Setup

Step 1: Install OpenFHE

Follow the same steps as in Section 3.2 to install OpenFHE on clients.

Step 2: Download CryptoContext

Fetch the JSON file from the server:

curl http://[server-ip]:8000/cryptocontext.json -o cryptocontext.json

Step 3: Upload Files to Server

Upload client data (e.g., public key):

curl -X POST "http://[server-ip]:8000/upload?client\_id=client1&type=pubkey" -F file=@client1-public.key

* Ensure client\_id and type match server requirements.
* Use -F for form data to handle file uploads securely.

3.7 Troubleshooting

* Library not found: Verify LD\_LIBRARY\_PATH includes /usr/local/lib.
* Mongoose buffer issues: Increase MG\_MAX\_RECV\_SIZE and recompile.
* Serialization errors: Ensure Cereal headers are accessible; rebuild OpenFHE if needed.
* Version mismatches: Use the same OpenFHE version on server and clients to avoid compatibility issues.
* Security: In production, secure the Mongoose server (e.g., HTTPS, authentication) to protect sensitive data like cryptocontext.json.

# 4. Directory Structure (Reference)

~/Desktop/PPFL/

├── Makefile

├── client

│   ├── build

│   │   ├── REkeyGen

│   │   ├── decryptModelWeights

│   │   ├── encryptModelWeights

│   │   └── keyGen

│   ├── config

│   │   ├── client\_1

│   │   └── client\_2

│   ├── src

│   │   ├── REkeyGen.cpp

│   │   ├── c\_trainAndUpdate.py

│   │   ├── decryptModelWeights.cpp

│   │   ├── encryptModelWeights.cpp

│   │   └── keyGen.cpp

│   └── storage

│   ├── client\_1

│   └── client\_2

├── documentation

├── lib

│   ├── base64\_utils.h

│   ├── mongoose

│   │   ├── mongoose.c

│   │   └── mongoose.h

│   └── openfhe-development

│   ├── CMakeLists.User.txt

│   ├── CMakeLists.txt

│   ├── CPPLINT.cfg

│   ├── LICENSE

│   ├── Makefile

│   ├── OpenFHEConfig.cmake.in

│   ├── OpenFHEConfigVersion.cmake.in

│   ├── PreLoad.cmake

│   ├── README.md

│   ├── benchmark

│   ├── build

│   ├── cicd

│   ├── configure

│   ├── demoData

│   ├── docker

│   ├── docs

│   ├── logo.png

│   ├── scripts

│   ├── src

│   ├── test

│   └── third-party

├── orchestration

│   ├── client\_fns.sh

│   ├── comm\_fns.sh

│   ├── helper\_fns.sh

│   ├── oConfig.json

│   ├── run.sh

│   └── server\_fns.sh

├── release

│   ├── client

│   └── server

├── server

│   ├── build

│   │   ├── aggregateEncryptedWeights

│   │   ├── changeCipherDomain

│   │   ├── genCC

│   │   └── runMserver

│   ├── config

│   │   ├── config\_cc.json

│   │   └── sConfig.json

│   ├── src

│   │   ├── aggregateEncryptedWeights.cpp

│   │   ├── changeCipherDomain.cpp

│   │   ├── genCC.cpp

│   │   └── runMserver.cpp

│   └── storage

│   ├── CC.json

│   ├── client\_1

│   └── client\_2

├── test

│   ├── client

│   │   ├── build

│   │   ├── config

│   │   └── src

│   ├── server

│   │   ├── build

│   │   ├── config

│   │   └── src

│   ├── test\_helper\_fns.hpp

│   └── test\_run.sh

└── utils

├── cereal

│   ├── CMakeLists.txt

│   ├── Config.cmake.in

│   ├── LICENSE

│   ├── README.md

│   ├── appveyor.yml

│   ├── build

│   ├── cereal.pc.in

│   ├── doc

│   ├── include

│   ├── sandbox

│   ├── scripts

│   └── unittests

└── json

├── BUILD.bazel

├── CITATION.cff

├── CMakeLists.txt

├── ChangeLog.md

├── FILES.md

├── LICENSE.MIT

├── LICENSES

├── MODULE.bazel

├── Makefile

├── Package.swift

├── README.md

├── build

├── cmake

├── docs

├── include

├── meson.build

├── nlohmann\_json.natvis

├── single\_include

├── src

├── tests

└── tools