QFLARE: Project Description

Abstract

Title: QFLARE: A Quantum-Resistant Federated Learning System with Secure Enclave Aggregation

Abstract: The proliferation of edge computing devices has created new opportunities for large-scale machine learning, but it has also introduced significant challenges in data privacy and system security. Traditional centralized models require sensitive user data to be sent to a central server, while the rise of quantum computing threatens to break the classical cryptographic protocols that protect these systems. This project, **QFLARE (Quantum-Safe Federated Learning & Authentication for Resilient Edge)**, presents a novel framework for performing privacy-preserving machine learning that is secure against both classical and quantum attacks.

QFLARE integrates **Federated Learning (FL)** with a robust, multi-layered security architecture. At its core, the system uses a **Post-Quantum Cryptography (PQC)** suite for all communication, combining a Key Encapsulation Mechanism (KEM) with digital signatures to ensure confidentiality, integrity, and authentication. All sensitive operations, particularly the aggregation of model updates, are architecturally designed to be performed within a **Trusted Execution Environment (TEE)**, or secure enclave, to protect against server-side compromises. Furthermore, the system incorporates a secure, token-based device enrollment process and implements defensive measures against model poisoning attacks. This holistic approach ensures that only authenticated devices can participate, that their data privacy is preserved, and that the integrity of the global model is maintained in a quantum-resistant environment.

Methodology

The QFLARE system operates through a sequence of secure, well-defined steps that guide a device from initial enrollment to participation in the federated learning process.

Step 1: Secure Device Enrollment

The foundation of the system's trust is a secure enrollment process that prevents unauthorized devices from joining the network.

1. **Token Generation**: An administrator generates a unique, single-use enrollment token. This token is delivered securely to the legitimate operator of a new edge device through an out-of-band channel.
2. **Initial Handshake**: The new edge device connects to a dedicated /enroll endpoint on the server. It presents the one-time token as proof of authorization.
3. **Key Pair Generation**: Upon successful token validation, the edge device generates two sets of Post-Quantum key pairs:

    - A **KEM key pair** (e.g., FrodoKEM) for secure key exchange.
    - A **digital signature key pair** (e.g., CRYSTALS-Dilithium) for authentication.
4. **Public Key Registration**: The device transmits its **public keys** (both KEM and signing keys) to the server. The server stores these public keys and associates them with the device's unique ID. The **private keys never leave the edge device**.
5. **Token Revocation**: The enrollment token is immediately invalidated by the server to prevent reuse.

Step 2: Authenticated Session Establishment

For each training round, a device must establish a secure session with the server using a challenge-response mechanism that ensures Perfect Forward Secrecy (PFS).

1. **Challenge Request**: The edge device initiates a connection and requests a challenge from the server.
2. **Encapsulated Session Key**: The server uses the device's registered public KEM key to **encapsulate** a new, random, single-use session key. It sends this encrypted key (ciphertext) to the device.
3. **Session Key Decryption**: The device uses its private KEM key to **decapsulate** the ciphertext, revealing the session key. Now both the device and server share a symmetric session key that is unique to this interaction.
4. **Proof of Possession**: All further communication in the session is encrypted with this temporary session key.

Step 3: Local Model Training

With a secure session established, the device can participate in the federated learning process.

1. **Global Model Download**: The device securely downloads the current version of the global machine learning model from the server.
2. **Local Training**: The device trains this model on its own local, private data. This training produces a "model update" containing the learned weights or gradients.
3. **Update Signing**: The device calculates a hash of its model update and signs the hash with its **private signing key**, creating a digital signature.

Step 4: Secure Model Update Submission

The device securely transmits its results back to the server.

1. **Secure Transmission**: The device sends the model update and its digital signature to the server through the encrypted session channel.

2. **Server-Side Verification**: The server receives the update. It uses the device's public signing key to verify the digital signature. This proves that the update came from the legitimate device and was not tampered with in transit.

Step 5: Secure Aggregation within the Enclave

The server does not process the model updates in its main memory. All sensitive aggregation logic is handled by the secure enclave.

1. **Data Forwarding**: The server forwards the verified model updates from multiple devices to the secure enclave.
2. **Poisoning Defense**: Inside the enclave, a preliminary check is performed on each update. For example, its cosine similarity is compared against the current global model. Updates that are drastic statistical outliers are flagged and rejected, providing a defense against model poisoning.
3. **Federated Averaging**: The enclave securely averages the valid model updates to compute a new, improved set of weights for the global model.
4. **Global Model Update**: The enclave returns the new global model to the server, which can then be distributed to devices in the next training round.

Root Directory

- QFLARE/
    - .env: **(New)** Stores secrets and environment-specific configurations, like the SECRET_KEY. This file is not committed to version control.
    - .env.example: Provides a template for the .env file, showing what variables are needed.
    - docker-compose.yml: Defines and orchestrates the multi-container Docker application, managing the server and edge_node services and their networking.
    - LICENSE: Contains the software license for the project.
    - README.md: The main project documentation, updated to reflect the new secure architecture and setup instructions.
    - requirements.txt: Lists the high-level Python dependencies for the entire project.

Configuration

- config/

- global_config.yaml: Holds static, non-secret configurations for the application, such as the chosen Post-Quantum Cryptography (PQC) algorithm and federated learning parameters.

Documentation

- docs/
  - api_docs.md: Documentation for the server's REST API endpoints.
  - architecture.png: A visual diagram of the system's architecture.
  - system_design.md: A detailed document explaining the overall system design, data flow, and security model.

Edge Node

- edge_node/
  - Dockerfile: Instructions for building the Docker container for the edge node application.
  - main.py: The main entry point for the edge node. It orchestrates the process of connecting to the server, training the model, and submitting updates. It has been **updated to use HTTPS**.
  - requirements.txt: A list of Python packages required specifically for the edge node.
  - secure_comm.py: Contains the logic for handling the secure session establishment with the server, including key exchange and encryption.
  - trainer.py: Simulates the local machine learning model training on the edge device's private data.

Enclaves

- enclaves/
  - enclave_code.c: A placeholder for the C code that would run inside a real hardware-based Trusted Execution Environment (TEE) like Intel SGX.
  - enclave_config.json: Configuration file for the secure enclave.

Models

- models/
  - cnn_model.py: A stub for defining the architecture of the machine learning model (e.g., a Convolutional Neural Network).

    - model_utils.py: Placeholder for utility functions related to the model, such as serialization or evaluation metrics.

Scripts

- scripts/
    - enroll_device.py: A client-side script that a new device would run. It uses a one-time token to securely enroll with the server via the /enroll endpoint.
    - federated_start.py: A stub for a script that could be used to orchestrate and kick off a federated learning round across all devices.
    - generate_token.py: **(New)** An administrative script used to generate secure, single-use enrollment tokens for new devices.

Server

- server/
    - Dockerfile: **(Updated)** A robust Dockerfile that reliably installs the necessary PQC cryptographic libraries from a trusted package archive.
    - main.py: **(Updated)** The main FastAPI application file. It has been updated with rate limiting to prevent DoS attacks and now includes the secure /enroll endpoint, replacing the insecure public registration form.
    - requirements.txt: **(Updated)** Lists all Python dependencies for the server, including new additions like redis, slowapi, and pqcrypto.
    - registry.py: A simple in-memory store for tracking registered device IDs.
    - api/
        - routes.py: **(Updated)** Defines the core API endpoints (/enroll, /request_qkey, /submit_model). The logic has been updated to use the real PQC functions from pqcrypto_utils.py.
        - schemas.py: Contains Pydantic models for request and response data validation.
    - auth/
        - key_handler.py: Contains legacy, insecure key handling logic (kept for reference, but no longer used by the main secure flow).
        - pqcrypto_utils.py: **(Updated)** A critical security file. It has been rewritten to implement a real PQC algorithm (like FrodoKEM), replacing the insecure placeholder hash function.

- enclave/
    - mock_enclave.py: **(New)** A Python class that simulates a secure enclave. It isolates the sensitive model aggregation logic and includes a basic defense against model poisoning, demonstrating the correct system architecture.
- fl_core/
    - aggregator.py: **(Updated)** This file's logic has been refactored. It now acts as a controller that receives model updates and delegates the actual secure aggregation to the mock_enclave.
    - client_manager.py: A module for managing the status and activity of connected edge clients.
    - fl_controller.py: A stub for a higher-level controller that could manage the entire lifecycle of federated learning rounds.
- ledger/
    - audit_logger.py: A simple logger for recording important system events.
    - block_commit.py: A stub for logic that could commit audit logs to a blockchain for immutable record-keeping.
- templates/
    - devices.html: A simple HTML page to display the list of registered devices.
    - index.html: The main landing page for the server's web interface.
    - register.html has been removed, as the insecure public registration process has been replaced.

Tests

- tests/
    - test_auth.py: A stub for writing tests for the authentication mechanisms.
    - test_fl_training.py: A stub for testing the federated learning training process.
    - test_key_rotation.py: A stub for testing key rotation logic.

QFLARE/

```
│
├── .env              # Stores environment variables like SECRET_KEY (Not committed to Git)
├── .env.example      # Example environment file
├── docker-compose.yml  # Orchestrates the server and edge node services
├── LICENSE
├── README.md         # Updated to reflect the new secure architecture
└── requirements.txt  # Main project requirements (includes new dependencies)
│
├── config/
│   └── global_config.yaml  # Main configuration (PQC algorithm, etc.)
│
├── docs/
│   ├── api_docs.md
│   ├── architecture.png
│   └── system_design.md
│
├── edge_node/
│   ├── Dockerfile
│   ├── main.py         # Updated to use HTTPS for all server communication
│   ├── requirements.txt
│   ├── secure_comm.py  # Handles secure session logic on the client side
│   └── trainer.py
│
├── enclaves/
│   ├── enclave_code.c  # Stub for a real hardware-based TEE implementation
│   └── enclave_config.json
│
```

```
├── models/
│   ├── cnn_model.py
│   └── model_utils.py
│
├── scripts/
│   ├── enroll_device.py    # Script for a device to use a one-time token to enroll
│   ├── federated_start.py
│   └── generate_token.py   # New: Admin script to create secure one-time enrollment tokens
│
├── server/
│   ├── Dockerfile          # Updated to reliably install PQC cryptographic libraries
│   ├── main.py             # Updated with rate limiting and the new secure /enroll endpoint
│   ├── requirements.txt    # Includes new dependencies like redis, slowapi, pqcrypto
│   ├── registry.py         # Manages the list of registered devices
│   │
│   ├── api/
│   │   ├── routes.py       # API logic, updated to use the real PQC functions
│   │   └── schemas.py
│   │
│   ├── auth/
│   │   ├── key_handler.py
│   │   └── pqcrypto_utils.py # Updated: Implements a real PQC algorithm (e.g., FrodoKEM)
│   │
│   ├── enclave/
│   │   └── mock_enclave.py   # New: Simulates the secure enclave for aggregation and poison defense
```

```
│   │
│   ├── fl_core/
│   │   ├── aggregator.py   # Updated: Delegates aggregation to the secure enclave
│   │   ├── client_manager.py
│   │   └── fl_controller.py
│   │
│   ├── ledger/
│   │   ├── audit_logger.py
│   │   └── block_commit.py
│   │
│   └── templates/
│       ├── devices.html
│       └── index.html
│       └── # register.html has been removed as it's no longer secure
│
└── tests/
    ├── test_auth.py
    ├── test_fl_training.py
    └── test_key_rotation.py
```