CS438: Secure Distributed Computing

Mahdi Atallah
mahdi.atallah@epfl.ch

Ian Golob
ian.golob@epfl.ch

Yannik Tim Krone
yannik.krone@epfl.ch

I. Executive Summary

a) Background about the Project: The project focuses on developing a privacy-preserving federated learning system that enables multiple devices to collaboratively train a global model without exposing sensitive data. Federated learning is a distributed machine learning approach where individual devices train a model locally on their private data and share only the model updates (e.g., changes to the model weights) with a central server. This allows the server to aggregate updates from multiple devices to create an improved global model without ever accessing the raw data.

This design inherently addresses privacy concerns by keeping sensitive data on the devices. However, traditional federated learning systems are still vulnerable to information leakage through shared model updates. For instance, adversaries analyzing these updates can infer patterns or reconstruct private data under certain circumstances. To mitigate this risk, the project incorporates advanced techniques from secure multiparty computation and encryption, such as xMK-CKKS, a protocol that encrypts model updates before they are shared. This encryption ensures that even during aggregation, individual device contributions remain confidential. By combining federated learning with state-of-the-art encryption methods, the system guarantees robust data confidentiality and mitigates critical privacy concerns that arise in distributed learning environments.

b) Project Description and Scope: The system is designed to operate under an honest-but-curious threat model, targeting applications in decentralized environments. It exclusively builds upon the Peerster implementation from Homework 1, leveraging its decentralized communication framework for message exchange and coordination. The scope encompasses secure aggregation of model updates, and handling device failures while ensuring scalability and efficiency across varying network conditions. The key objective is to balance security and computational performance while maintaining compatibility with the original Homework 1 design principles.

c) Main Building Blocks of the Project:

- **Server:** Coordinates the training process by managing communication, encryption, and aggregation of model updates. It initiates and terminates aggregation rounds.
- **Device:** Participates by training models on its dataset and securely sharing encrypted updates and decryption shares with the server.

d) Component Descriptions:

- **Machine Learning Module**: Devices locally train their models and encrypt updates before securely transmitting them to the server. The server aggregates these updates to produce a global model for the next training round.
- **Encryption Module**: Implements homomorphic encryption to obfuscate model updates during aggregation. The server cannot access individual contributions.
- **Communication Module**: Coordinates the federated learning process through message exchanges, monitors device participation, and detects failures to handle them gracefully.

e) Implementation Summary: The implementation involves creating message handlers for core functionalities, including initiating aggregation rounds, managing encryption keys, aggregating updates, and detecting failures. Technical choices like asynchronous training, fault monitoring using a heartbeat mechanism. The diagram in Figure 1 provides an overview of the implementation.

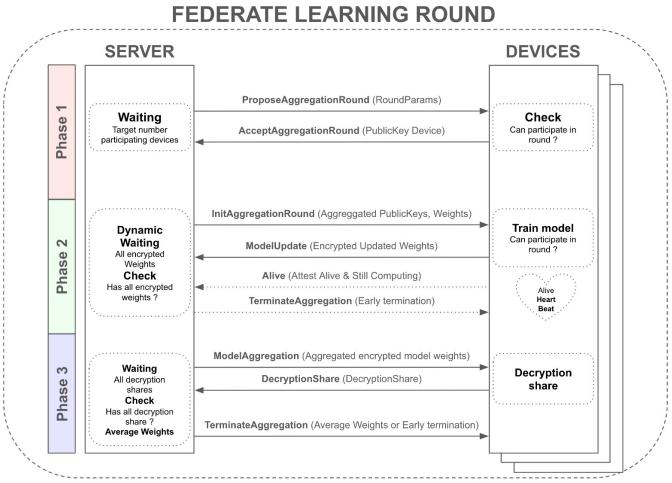


Fig. 1: Implementation Overview

f) Testing and Evaluation Summary: Testing was performed at multiple levels to ensure the reliability and security of the system. Unit tests validated core modules, including encryption and model updates, while integration tests evaluated the system under varying network topologies and conditions. The overall code coverage achieved during testing was 80.7%, ensuring the reliability of the implemented functionalities. Metrics such as training latency, encryption performance, and device failure detection time were measured to assess the system's efficiency and fault tolerance.

II. Related Work

Federated learning [1] allows multiple devices (e.g., smartphones or IoT devices) to collaborate on training a global model without sharing their raw data. However, sending even processed data like model updates could leak private information. Multi-key homomorphic encryption (MKHE) [2], [3] solves this by letting each device encrypt its updates so that the server can still combine them—just without ever seeing the raw values.

The xMK-CKKS protocol [4] is a way to privately aggregate these encrypted updates. A simplified view of the process is:

1) **Local Training:** Each device trains its model locally for several epochs, starting from the current global model weights w_t.
2) **Encryption:** The device then encrypts its updated model weights w_i^t into ciphertext form $(c_0^{d_i}, c_1^{d_i})$ using a shared public key that is derived from the aggregation of all devices' individual public keys.
3) **Aggregation at the Server:** The server collects all these encrypted weights from participating devices and adds them together homomorphically, resulting in an encrypted sum C_{sum}.
4) **Decryption Shares:** Each device helps decrypt this sum by sending a small decryption share D_i to the server. These shares do not reveal individual device weights.
5) **Averaged Model:** The server combines all the decryption shares and the encrypted sum, then recovers the aggregate model updates. By dividing this total by the number of devices, the server obtains the averaged weights w_{t+1} for the next round of training.

This approach ensures that the server never sees any single device's model weights in clear form. By performing these steps over polynomial rings and relying on RLWE assumptions [5], xMK-CKKS guarantees both security and functionality, making it well-suited for privacy-preserving federated learning.

III. Threat Model

Our system is designed to operate under an honest-but-curious adversary model, where participants (devices and server) follow the protocol but may attempt to infer sensitive information from shared data. The following adversarial capabilities, fault-tolerance assumptions, and security guarantees define the scope of the threat model and our system:

A. Adversary Capabilities

1) Access to All Messages: Both the server and each participating device can access all encrypted model updates and decryption shares. They may attempt to analyze this information to infer private data about individual devices.

2) Collusion: A subset of up to $k < N$ participating devices may collude with the server to compromise the privacy of other devices, where N is the total number of devices.

B. Fault Tolerance

If a device fails after its keys have been included in the public encryption parameters but before sending its decryption share, decryption becomes impossible [4]. As a result, the system is not fault-tolerant to such device failures.

C. Security Guarantees

1) Confidentiality of Model Updates: The system ensures that individual model updates remain private and cannot be decrypted by the server or other devices. Only the aggregated sum of all updates can be decrypted, obfuscating individual contributions.

2) Resistance to Collusion: The scheme is secure against collusion between up to $k < N - 1$ devices and the server [4].

IV. Design

This section describes the design of our system, focusing on the interaction between the server and devices during federated learning rounds. It provides an overview of the system's roles and details the communication messages used, integrated into the description of the main components.

A. High-Level System Overview

At a high level, the system consists of two primary roles:

- **Server:** Coordinates the training process by managing communication, encryption, and aggregation of model updates. It initiates and terminates aggregation rounds.
- **Device:** Participates in training by performing local updates on its dataset and securely sharing encrypted updates and decryption shares with the server.

B. Detailed Component Descriptions

The system design is based on the Peerster version of Homework 1, leveraging its decentralized communication framework to implement the messages required for secure federated learning. Below are the key message types and their roles in the system:

- **ProposeAggregationRoundMessage:** This message is sent by the server to initiate an aggregation round. It contains the server's identity, round parameters (e.g., encryption settings and model details), and a unique ID to identify the round. This message serves as the first step in initiating communication with devices that wish to participate in the round.
- **AcceptAggregationRoundMessage:** This message is sent by a device in response to the ProposeAggregationRoundMessage to indicate its participation in the proposed round. It contains the device's identity, the round ID for tracking, and the device's public key, which is required for the xMK-CKKS encryption scheme. This allows the server to register and authenticate participating devices.
- **InitAggregationRoundMessage:** After collecting responses from devices, the server sends this message to initiate the aggregation round formally. It includes

the server's identity, the round ID, the initial model weights to be trained upon, and the aggregated public keys of the participating devices. This message ensures that all devices begin the round with consistent starting parameters.

- **ModelUpdateMessage:** Devices send this message after completing their local training. It contains the device's identity, the round ID, and the encrypted model update in the form of a ciphertext pair as specified by the xMK-CKKS scheme. This message allows devices to securely share their contributions to the global model without exposing raw data.
- **ModelAggregationMessage:** Once the server receives all ModelUpdateMessages from participating devices, it aggregates the encrypted updates and sends this message. It contains the round ID and the aggregated model updates in the form of a ciphertext pair. This step enables the secure computation of the global model while maintaining privacy.
- **DecryptionShareMessage:** After receiving the ModelAggregationMessage, devices send this message to share their decryption shares. It includes the device's identity, the round ID, and the decryption share required to finalize the global model. This step ensures that the server can securely decrypt the aggregated updates without accessing individual device contributions.
- **TerminateAggregationRoundMessage:** This message is sent by the server to conclude an aggregation round. It is used in two scenarios: - If all required ModelUpdateMessages or DecryptionShareMessages are not received, the server terminates the round due to a failure. - If all required messages are received, the server successfully concludes the round and sends the averaged model weights to the participating devices. This message includes the round ID and the resulting model weights, if applicable.
- **AliveMessage:** Devices periodically send this message to the server to indicate their active status during an aggregation round. It includes the device's identity, allowing the server to monitor participation and detect any potential device failures in real-time.

V. Implementation

A. Device and server implementations

The device and server implementations are designed to handle these messages systematically, as described in detail in the accompanying diagrams:

- **Device Implementation:** Upon receiving the relevant messages, devices determine participation, train models, and share encrypted updates or decryption shares. Asynchronous training ensures responsiveness, and messages are processed based on the device's participation and round state.
- **Server Implementation:** The server operates in three phases, with each phase detailed in the corresponding diagram:

1) **Phase 1:** Proposes aggregation rounds, collects responses, and aggregates public keys (see Figure 2).

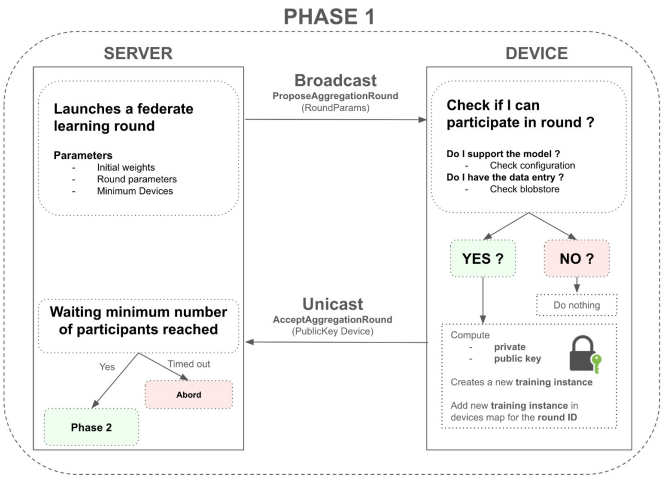


Fig. 2: Phase I: Propose and Accept Aggregation Round

2) **Phase 2:** Sends initialization messages, monitors device activity, and aggregates model updates (see Figure 3).

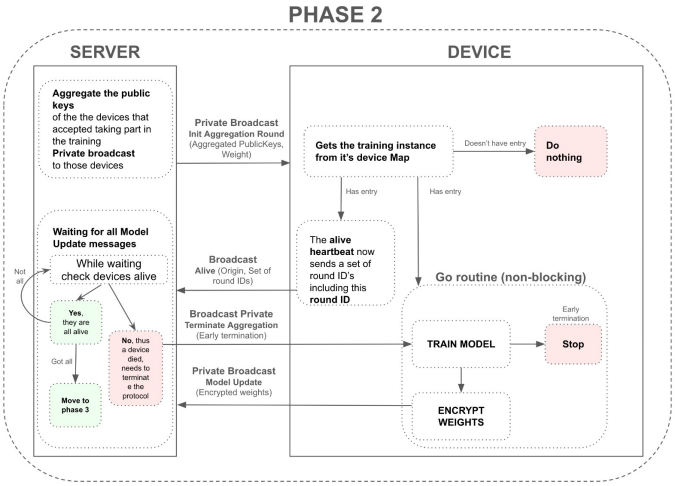


Fig. 3: Phase II: Initialize and Update Models

3) **Phase 3:** Manages decryption shares and finalizes model aggregation (see Figure 4).

B. Technical and Design Choices

- **Network Efficiency:** Unicast is used for accepting aggregation rounds to prevent congestion, while broadcasts ensure reliable delivery for other messages.
- **Asynchronous Training:** Training runs in a separate Go routine, allowing devices to handle other messages concurrently and improving responsiveness. The process can also be interrupted between epochs if a TerminateAggregationRoundMessage is received.

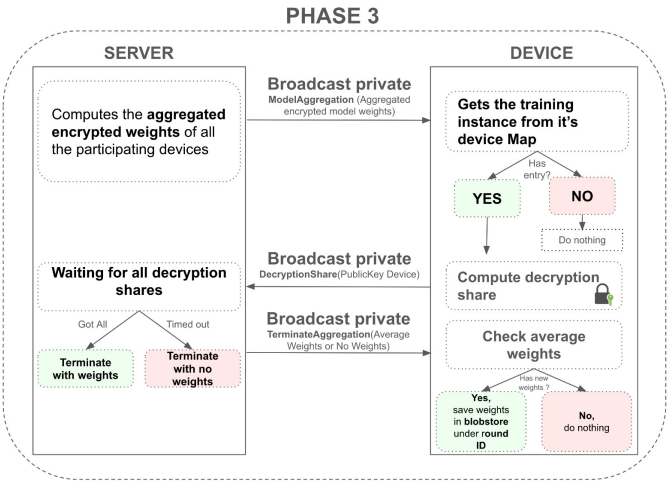


Fig. 4: Phase III: Aggregate and Decrypt Updates

- **Fault Tolerance:** The server monitors device activity using AliveMessages and terminates rounds early when failures occur, ensuring resource efficiency.

VI. Testing

This section outlines the testing strategy adopted for the system, including the types of tests conducted, the aspects of the implementation that were validated, and the overall coverage achieved.

A. Unit Testing

- **Message Handlers:** Unit tests cover every message handler, validating behavior for both invalid inputs and valid inputs with expected outputs.
- **Model Validation:** Unit tests ensure the correctness of model implementations.
- **Encryption:** Since encryption was implemented from scratch, dedicated unit tests verify its functionality and robustness.
- **Aggregation Round:** Simple cases of the entire aggregation round were tested, including:
  - Scenarios with insufficient devices to start the round.
  - Device failures during the aggregation process.
  - Small networks with few nodes to evaluate edge-case behavior.

B. Integration Testing

- **Multiple Rounds:** The system was tested over several rounds, mimicking real-life conditions.
- **Network Topologies:** Various topologies were tested under different conditions:
  - A standard topology with one server using different communication mediums (e.g., UDP, channels) subjected to normal, slightly jammed, or heavily jammed conditions.
  - Scenarios where the server runs two training rounds concurrently.
  - Scenarios with multiple servers running training rounds simultaneously.

The overall code coverage achieved during testing was 80.7% ensuring the reliability of the implemented functionalities.

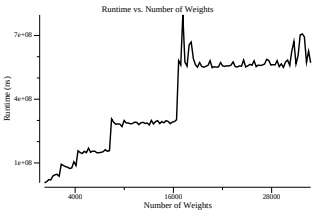
VII. Evaluation

This section presents the evaluation of the system, focusing on critical performance metrics. Each metric is discussed in detail in subsequent subsections, supported by graphs and tables for clarity. Reproducibility of results is ensured through detailed documentation.

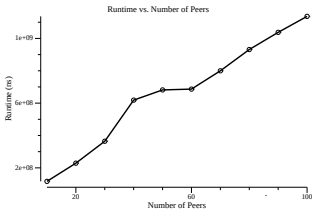
A. Encryption Implementation Performance

The encryption implementation was evaluated to assess its efficiency and robustness in various scenarios. The analysis focused on two main aspects:

- Analyzing the encryption latency, which increases with the number of peers or model complexity due to higher computational load (Figures 5a and 5b).
- Evaluating the system's variability, which decreases with more peers due to the averaging effect, smoothing individual discrepancies (Figure 6a), and with higher weight scaling factors as larger scales amplify consistent contributions while diminishing the relative impact of noise (Figure 6b).

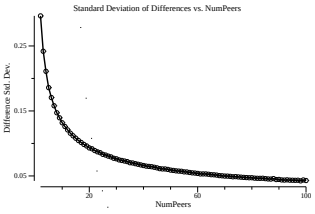


(a) Encryption time vs. model size.

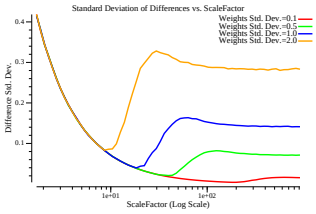


(b) Encryption time vs. number of participants.

Fig. 5: Performance evaluation of xMK-CKKS encryption: model size (left) and number of participants (right).



(a) Standard deviation vs. number of peers.

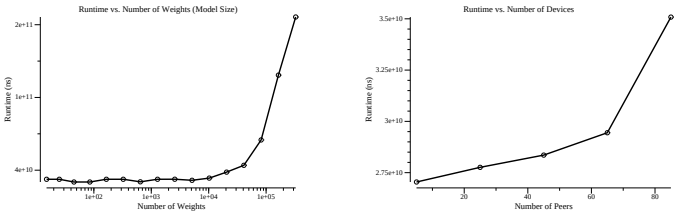


(b) Standard deviation vs. weight scaling.

Fig. 6: Variability analysis: number of peers (left) and weight scaling (right).

B. Training Latency

The training latency was analyzed based on two factors: the impact of increasing model sizes (Figure 7a) and the effect of varying the number of participating nodes (Figure 7b). Latency increases rapidly due to the added training time required for larger models and the computational cost of encrypting and transmitting heavier messages.



(a) Integration time vs. model size.

(b) Integration time vs. number of peers.

Fig. 7: Integration time comparison by model size (left) and number of peers (right).

C. Device Failure Detection Time

D. Device Failure Detection Time

The system's ability to detect device failures was evaluated by measuring the time it takes for the server to recognize a crashed device. Due to the encryption scheme, it is not possible to test training under churn conditions, as the training cannot continue if a node leaves mid-process. However, we can evaluate how quickly the system detects such failures. This evaluation focused on varying path lengths to simulate different network conditions and measure their impact on failure detection time, which increases nearly linearly with the path length due to the time required for communication between the node and the server.

Fig. 8: Topology used for evaluating device failure detection time under varying network conditions.

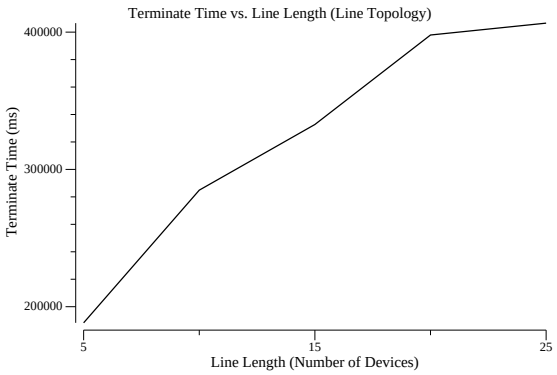


Fig. 9: Terminate Time vs. Line Length (Line Topology)

VIII. Limitations & Future work

This section highlights the current limitations of the project and outlines potential directions for future improvements.

- **Limited Model Complexity:** The project employs a simple linear regression and a simple neural network with a single hidden layer, with backpropagation computed manually. Future work should explore more complex models to improve scalability and performance.
- **Restricted Loss Functions and Optimization Methods:** The current implementation uses MSE loss with SGD. Expanding this to include a variety of loss functions and optimization algorithms could enhance model training efficiency and accuracy.
- **Rumor Size and Network Constraints:** The size of rumors grows significantly due to multiple broadcasts, exceeding the capacity of the UDP medium. Future efforts should focus on optimizing rumor propagation or exploring alternative gossiping methods to address this limitation.
- **Granularity in Supporting Models:** The current system does not support granular model configurations, such as limiting support for certain models based on the number of epochs or weights. While this was not a central focus of the project, enhancing the engine's viability would involve adding such features to better handle model-specific requirements.
- **Focus on Communication over Machine Learning Guarantees:** The project primarily focused on peer-to-peer communication and did not emphasize guarantees of machine learning performance. Future work could incorporate best practices in federated learning, such as exploring advanced machine learning methods tailored for distributed learning scenarios.

References

[1] K. e. a. Bonawitz, "Practical secure aggregation for privacy-preserving machine learning," in Proc. 2017 ACM SIGSAC Conf., 2017, pp. 1175–1191.

[2] J. H. e. a. Cheon, "Homomorphic encryption for arithmetic of approximate numbers," Proc. ACM Comput. Commun. Sec., vol. 6, pp. 1–35, 2019.

[3] S. e. a. Halevi, "An improved rns variant of the bfv homomorphic encryption scheme," Cryptography, vol. 7, no. 4, 2023.

[4] A. e. a. Kim, "Secure multi-key homomorphic encryption for federated learning," Int. J. Comput. Intell. Syst., vol. 13, pp. 1–12, 2021.

[5] A. e. a. Acar, "A survey on homomorphic encryption schemes: Theory and implementation," ACM Comput. Surv., vol. 54, no. 4, pp. 1–35, 2021.