A Comparative Experiment of Integer-Packed Paillier against CKKS over Long Iterations in Federated Learning

Dama Dhananjaya Daliman - 18222047
Information Systems and Technology Study Program
School of Electrical Engineering and Informatics
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: dama.dhananjaya.daliman@gmail.com, 18222047@std.stei.itb.ac.id

***Abstract*—Federated Learning allows devices to collaboratively train AI models without sharing private data, but it relies on encryption to keep user information secure. Currently, there is a major trade-off in encryption methods. The Paillier algorithm provides exact accuracy but is extremely slow, while the CKKS algorithm is very fast but introduces approximation errors (noise) that degrade data precision over time. To address this, this paper proposes an integer-packing technique for Paillier. By converting decimal values into positive integers and packing multiple model parameters into a single encrypted block, these parameters can be processed simultaneously. This approach was evaluated on a simple neural network over 500 training rounds. Results show that the packed Paillier method is 55 times faster than traditional Paillier. Although CKKS remains the fastest overall, the packed Paillier method maintains perfect mathematical precision with virtually zero encryption noise, avoiding the accuracy drift seen in CKKS. This work demonstrates that integer-packing offers a highly practical balance of speed and exact precision for privacy-preserving machine learning.**

Keywords—Federated Learning, Homomorphic Encryption, Privacy, Machine Learning

I. INTRODUCTION

In the modern age, data privacy has become a significant issue [1]. Machine learning has followed suit in this issue crucially because of the need for federated learning (FL). As explained in work [2] FL is a pioneering privacy-preserving distributed machine learning paradigm that allows multiple devices to collaboratively train a shared model while keeping the training data decentralized. However, according to [3] as mentioned in [1], malicious users in the training process might use the plaintext gradient to train a shadow model to compromise the data security of other users. Utilizing encryption for masking the gradient before aggregation has proven to be an effective approach to ensure the security of the gradients. Sadly, traditional encryption schemes such as AES and DES requires the ciphertext to be decrypted before computations can be applied, making them unsuitable for FL [2].

This is where homomorphic encryption (HE) comes into play. HE allows calculation on the encrypted data without decrypting it and the result of the homomorphic operation after decryption is equivalent to the operation on the plaintext data. According to [1], because the operation cannot identify the data being operated on during the whole process of the homomorphic operation, the security of the private data can be guaranteed. The additive nature of the HE Paillier scheme is widely employed in FL but entails high computational and communication costs [1, 2]. The CKKS (Cheon-Kim-Song) scheme benefits from ciphertext packing and rescaling, thus becoming the most efficient HE for approximate homomorphic computations over real and complex numbers [2].

Recent research has proposed several frameworks to integrate HE into federated learning to protect gradient privacy. For instance, PFMLP [1] combines partially homomorphic encryption with federated learning, where learning parties transmit only encrypted gradients, achieving model accuracy nearly identical to traditional methods with less than 1% deviation on the MNIST dataset. Similarly, FedSHE [2] introduces an adaptive segmented CKKS homomorphic encryption scheme built upon the federated averaging algorithm, demonstrating that homomorphic encryption does not incur performance loss on FL models through evaluations on datasets such as MNIST and CIFAR-10. However, while PFMLP employs an improved Paillier algorithm to speed up training by 25–28%, it still faces the inherent computational overhead of partially homomorphic encryption [1]. Conversely, FedSHE shows that fully homomorphic encryption can be more computationally efficient than Paillier in federated learning settings, yet it relies on approximate computations over real and complex numbers rather than exact integer arithmetic [2].

As we can see, secure federated learning algorithms have been trapped on opposite sides of a scale. One side with exact integer precision with Paillier algorithm but suffers from computational inefficiency as shown in [1] and the other side with extreme computational efficiency via polynomial ring vector parallelization but forced to sacrifice exactness as

mentioned in [2]. This leads to the problem of designing a secure aggregation primitive that breaks this trade off. To address this gap, this work designs an integer-packing framework for the Paillier algorithm to emulate vector parallelization. The main contribution of this work is the implementation of a functional Multi-Layer Perceptron (MLP) prototype under a federated learning framework. Providing comparative experiment analysis on the integer-packed Paillier approach against a depth-0 CKKS baseline, reporting empirical metrics on model classification accuracy, loss convergence behavior, and computational runtime overhead over long training iterations.

II. Related Works

To properly implement this comparative experiment, it is wise to discuss briefly the related works that relate to this experiment.

A. Federated Learning

In 2016 a machine learning paradigm called federated learning was proposed in this work [4]. Due to its characteristic of being a multi-party cooperative paradigm, it gradually attracted much attention [1]. Since then, federated learning has been widely used in various fields such as Google's Gboard [5] and recommendation systems [6]. In 2019, more related works were proposed. Wang focused on the problem of learning model parameter when data distributed across multiple edge nodes, without sending raw data to centralized node. In work [1], the authors explain the concept of Federated Learning is that when data is divided into different areas, one can use the information from another party's data for their own model by utilizing intermediate variables in the training process. From how the data is split, federated learning can be categorized into two, horizontal federated learning (using sample expansions) and vertical federated learning (using feature expansions), visually it can be represented as in Figure 1.

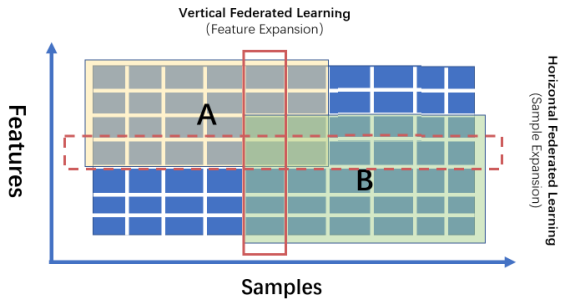

Fig. 1. *Visualization of the two different kinds of federated learning as depicted in [1]*

According to [1], the idea of horizontal federated learning is machine learning with sample expansions, that can be formulized as:

$$Xi = Xj, Yi = Yj, Ii \neq Ij, \forall Di, Dj, i \neq j \quad (1)$$

where D represents data, X represents features, Y represents samples, and I represents data index. This means that different users have different data that may or may not have intersections

B. Homomorphic Encryption for Federated Averaging

As stated in work [1], homomorphic encryption is used in federated learning to provide model security against member inference attacks. The core idea of homomorphic encryption is that the result of performing some operations on the ciphertext in its space will result in an equivalent encryption operation over the plaintext in its space. This can be formally stated as:

$$E(a) \oplus E(b) = E(a \otimes b) \quad (2)$$

where E is an encryption algorithm, a and b are two distinct plain texts, and $\oplus$ and $\otimes$ are operators. If the operation is either one of multiplication or addition then it satisfies partial HE and if it satisfies both additive and multiplicative homomorphism at the same time, then it satisfies full homomorphism.

Since multi-layer perceptron (MLP) architectures require summing the gradient data, the famous additive Paillier algorithm is better suited for MLP training [1]. However, as detailed in work [1], Paillier algorithm has high complexity when doing encryption and decryption. The Paillier aggregation protocol suffers from low computation efficiency and high communication costs because the method requires traversing and encrypting the numbers (gradients) element-wise.

To mitigate the element-wise bottleneck, Zhang et al. [7] proposed a batchcrypt algorithm which is based on the optimization of the FATE framework. It encodes quantized gradients in batches over a long integer so encryption only happens once. This improves the efficiency of the encryption and decryption [1].

In work [2] it is proven that utilizing encryption masking before gradient aggregation is an effective approach to ensure security. Traditional encryption schemes such as AES and DES mandates that ciphertext is decrypted before performing computations, making them unsuitable for FL. Homomorphic encryption (HE) is a powerful cryptographic primitive that allows computations on encrypted data so that only the secret key holder can decrypt the computation result. This satisfies the privacy protection requirement of gradients. The CKKS (Cheon-Kim- Kim-Song) scheme as explained in [8] that benefits from ciphertext packing and rescaling has been validated as the most efficient HE scheme to perform approximate homomorphic computations over real and complex numbers. The secure aggregation task in FL involves only addition operations. Therefore, the additive HE scheme, Paillier, becomes a natural choice commonly used in privacy-preserving FL. For CKKS, according to [2], configuring the CKKS multiplication depth to be 0 is sufficient to meet the homomorphic computation requirements. When the

multiplication depth is set to 0, CKKS exclusively supports homomorphic addition.

III. Methodology

This section will describe the proposed system for this paper, it will include the system model, cryptographic configurations, neural network architecture, and the evaluation pipeline.

A. System Model and Federated Framework

The structural architecture of this proof-of-concept prototype is designed as a centralized simulation environment deployed entirely on a single local computing node. Rather than configuring a complex network of physical machines connected over live networks, this framework utilizes process isolation within memory to simulate a multi-party environment. The architecture comprises a set of virtual clients and a single, isolated virtual aggregation server. The data domain follows a horizontal partitioning strategy, meaning that each virtual client holds an isolated shard of the dataset sharing a common feature namespace but containing completely distinct data samples.

The step-by-step lifecycle of a single collaborative training round is organized through the following sequence:

a. Local Training Initialization: At the start of a communication round, the central virtual server shares the current global model weights with all participating virtual clients. Each client updates its local machine learning model structure with these baseline starting parameters.
b. Local Plaintext Calculation: Virtual clients execute forward propagation to get model predictions, calculate the loss error, and run backpropagation entirely within the plaintext domain to calculate the updated weight and bias adjustments using their own local private data shards.
c. Homomorphic Encryption Masking: Clients extract their newly updated parameters from local memory and pass them through their designated encryption module to generate secure, masked parameter containers before any transmission occurs.
d. Central Aggregation: The virtual server collects the encrypted model updates from all clients and performs an addition-only homomorphic aggregation to sum the updates together without decrypting or viewing the underlying parameter data.
e. Global Delivery and Decryption: The aggregated ciphertext total is returned to the clients, decrypted using their secret key, and divided by the total client count to compute the new global model baseline for the subsequent training round.

To run this workflow, the simulation purposely uses a single-threaded, sequential loop approach instead of a multi-threaded parallel setup. This design is selected because the foundational federated averaging protocol is entirely synchronous, meaning the server cannot perform its aggregation step until every single client has completed its local updates. Using multiple threads in Python introduces significant software engineering barriers, such as internal lock constraints that prevent simultaneous calculations, and data copying errors when passing complex homomorphic encryption variables across background processes. A single-threaded loop executes the identical mathematical operations as a live concurrent system but completely avoids background software bugs and ensures that the performance timers can capture the true mathematical computation times cleanly without any thread-switching interference.

This secure setup addresses a traditional semi-honest (honest-but-curious) threat model. Under this assumption, the central aggregation server is trusted to perform the mathematical aggregation operations correctly according to the protocol instructions. However, it is assumed that an adversary might inspect the shared model updates to launch gradient leakage attacks, attempting to reverse-engineer the math to reconstruct the original sensitive training samples belonging to individual clients. The homomorphic layer ensures that all updates remain fully hidden from the server throughout the process.

B. Baseline Cryptographic Configurations

To establish a clear comparison point, the prototype sets up two baseline encryption pipelines to measure against the proposed packed approach.

The first baseline implements the traditional element-wise Paillier encryption method, which treats every model parameter as an independent number. In compliance with standard security guidelines for a 128-bit safety level, the public key modulus size is configured to 2048 bits. Under this unoptimized baseline, the parameter matrices are flattened into a single list, and each individual weight element is encrypted sequentially via separate, heavy mathematical calculations. This creates a severe processing delay because the time spent encrypting scales up directly with the size of the network.

The second baseline uses a newer lattice-based approach called the CKKS scheme, configured to mirror optimized settings from recent literature. We set the polynomial modulus degree to 8192, allowing a vector of up to 4096 separate values to be packed and encrypted inside a single ciphertext container. Crucially, because the central server only needs to add models together during a standard federated loop, the multiplication depth is restricted to exactly zero. This locks the computation into an addition-only setup, which completely avoids the rounding noise growth caused by homomorphic multiplication and eliminates the need for slow rescaling operations during training.

C. Integer-Packed Paillier Design

The primary contribution of this prototype is the design of an Asymmetric Uniform Quantization layer with a Zero-Point Offset integrated into a custom bit-shifted ciphertext packing pipeline over Paillier's exact integer domain. Standard homomorphic packing setups face a significant precision hurdle when processing negative numbers. Utilizing standard signed binary representations can easily flood the upper bits of a slot with sign-extension ones, creating a high risk of carry-over bleeding and data pollution into neighboring parameters during server-side aggregation. To eliminate this vulnerability and maintain tight parameter precision across extended training timelines, this framework implements a three-phase data transformation pipeline.

First, local clients pass their fractional parameters through an asymmetric quantization function based on the landmark integer-arithmetic framework established in [9]. To convert floating-point values into integers, the parameters are multiplied by a uniform scale factor ($S = 10{,}000$), which preserves four decimal places of precision. Following the affine mapping concepts outlined in [9], a constant integer zero-point offset ($Z = 2^{16} = 65{,}536$) is then added to every scaled value. This constant shift pushes the entire parameter distribution into a strictly positive integer domain, completely removing negative sign bits from the local client's data structure before any encryption takes place.

Second, these strictly positive values are consolidated into parallel arrays via binary slot concatenation. As highlighted in the secure neural network mapping structures of the MIT graduate thesis [10], embedding quantized neural network parameters into homomorphic plaintext spaces requires strict boundary isolation to guarantee that values do not overflow their allotted bit-widths during computation. Maximizing the usage of 2048-bit plaintext space provided by the Paillier public key modulus, the program positions each positive integer side-by-side into 32-bit wide slots. The lower 16 bits of each slot contain the active parameter data payload, while the upper 16 bits serve as an empty guard padding zone initialized to pure zeros. Because the parameters are forced to be positive integers, the guard zones remain free of active sign bits, allowing them to cleanly absorb mathematical growth during aggregation without any danger of adjacent boundary contamination.

Third, the server carries out homomorphic parallel additions by multiplying the packed client ciphertexts together under modular arithmetic. Upon decryption of the aggregated global payload on the client side, the client isolates each slot and mathematically untangles the zero-point offset by subtracting the product of the participating client count and the baseline bias ($N \cdot Z$). This post-processing subtraction cleanly isolates the exact sum of the updates, which is then divided by the scale factor of 10,000 to restore the true floating-point values. By combining zero-point mapping with the secure parameter bounding concepts highlighted in [10], the architecture achieves a completely lossless, unpolluted recovery of global updates across long training horizons.

D. Neural Network Architecture

To stress-test these cryptographic configurations within a functional learning pipeline, the prototype instantiates a compact Multi-Layer Perceptron model designed for classification tasks. The model uses a simple four-layer setup. First, the input layer contains 4 units to map the basic attributes of incoming data samples. This is followed by two consecutive hidden layers that contain 4 neurons each to process data relationships. Finally, the output layer contains 3 units to determine the final classification categories.

The total parameter count of this network layout is determined by counting all connections and biases, yielding a total of 55 distinct variables across the entire model. When we pack these variables using 32-bit wide slots (allocating 16 bits for the data payload and 16 bits for the empty buffer), the entire model requires a total length of 1,760 bits. Because 1,760 bits sits safely below the 2048-bit capacity limit of the Paillier encryption key, the entire neural network fits perfectly inside a single ciphertext container. This allows the clients to bypass the element-wise loop bottleneck entirely. During local updates, virtual clients process training subsets using standard Stochastic Gradient Descent with a set learning rate of 0.05 over a fixed number of local epochs before passing the final updates to the encryption module.

E. Evaluation Pipeline

The complete experimental prototype is written as a single-threaded Python script running on a single CPU core. To ensure a completely controlled environment free from hardware discrepancies, all evaluated encryption pipelines are implemented natively within the same software execution environment, utilizing identical data splits and iteration horizons. The script uses the standard phe library to handle the large-integer modular math needed for the packed Paillier design, and the TenSEAL library to execute the optimized depth-0 CKKS calculations.

To maintain professional software standards, the codebase follows a modular design structure, separating distinct responsibilities into specialized script modules rather than piling all logic into a single file. The first file, crypto_utils.py, serves as the pure cryptographic core, housing the parameter contexts alongside the exact bit-shifting and slot extraction math needed for quantization. The second file, client.py, manages the local participant functions, hosting the neural network layer structures and the local plaintext training rules. The third file, server.py, strictly isolates the aggregator functions, ensuring the virtual server only carries out homomorphic addition without possessing any decryption keys. Finally, main.py orchestrates the simulation by handling the dataset partitions, managing the global iteration loop, and logging the performance charts.

To optimize execution speed and shift the focus entirely onto the cryptographic behavior, the testing pipeline utilizes exactly half of the standard Iris classification dataset. This truncated dataset is split evenly into local training, testing, and validation shards across the virtual clients. Shifting to a smaller data footprint ensures that local training finishes almost instantly, allowing the experiment to significantly scale up the total number of global communication rounds to test long-term behavior.

The testing pipeline automatically monitors three primary performance metrics to compare the encryption schemes:

a. Cryptographic Noise Infiltration: The program measures error accumulation by tracking the Mean Squared Error between the decrypted global model weights and an unencrypted plaintext control group. Because the dataset size is minimized, this check can be run across extended training horizons, which in this case will be a total of 500 global communication rounds, providing a clear view of how long-term iterations impact weight precision.
b. Model Training Utility: The script continuously logs the model's final validation classification accuracy and training loss values. This allows us to observe directly whether the approximation noise from the CKKS scheme causes any noticeable drift or drops in model accuracy over a long training timeline.
c. Computational Complexity Overhead: High-precision timers log the exact number of seconds spent running the encryption phase, the server aggregation phase, and the decryption phase. These time records are used to analyze the practical computational trade-offs of each system.

IV. Results and Discussion

The proof of concept system as described in section III is then developed using the Python programming language and modularized according to best practices and can be viewed in the source code repository provided at the end of this paper. Running the simulation on a single computer yielded the results discussed below.

A. Model Accuracy and Training Convergence

To assess the practical impact of homomorphic encryption on machine learning performance, the training loss convergence trajectories and validation classification accuracies were monitored across a long horizon of 500 global communication rounds. As illustrated by the empirical metrics in the first evaluation graph (Figure 2), the unencrypted plaintext control model, the depth-0 CKKS pipeline, and the integer-packed Paillier framework all follow identical optimization pathways. By the final iteration, all three paradigms converge smoothly, achieving matching training loss drops and identical classification accuracy curves. These curves prove that neither cryptographic structure introduces parameter degradation or optimization path distortion.

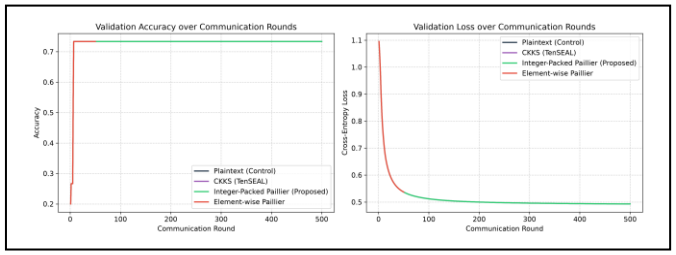

Fig. 2. *Accuracy and loss curve of training the MLP over 500 rounds*

Conversely, the traditional element-wise Paillier baseline is only plotted for the first 50 communication rounds. Due to its severe element-by-element serialization bottleneck, extending this baseline across 500 rounds is practically unviable for live simulations. However, within its short 50-round iteration, the element-wise model behaves identically to the other test lines. Taken together, these convergence results provide strong evidence that privacy constraints can be successfully integrated into decentralized Multi-Layer Perceptrons (MLPs) without compromising the final utility or learning capabilities of the model.

B. Computational Time Complexity and Execution Overhead

Logging of the cryptographic loops reveals a clear difference in operational throughput among the frameworks, as displayed in Figure 3. The unoptimized element-wise Paillier baseline suffers from extreme computation delays, it takes an execution time of 13.27 seconds per communication round, on average. This heavy overhead is due to the requirement of executing individual, independent modular exponentiations sequentially for every single parameter.

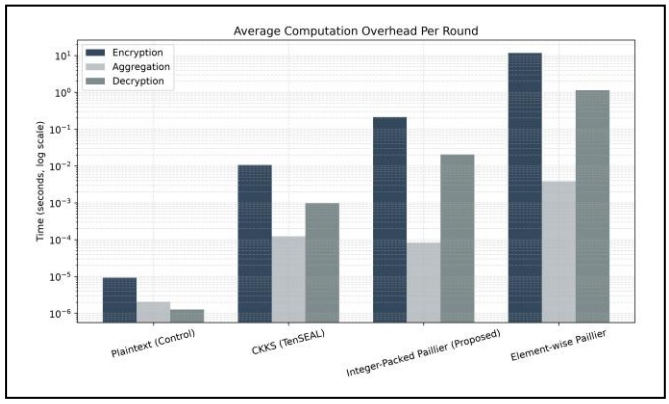

Fig. 3. *Average computational overhead per round for each approach*

By implementing the BatchCrypt-like data-packing layout, the virtual client compresses the entire 55-parameter network into a single ciphertext block. This optimization reduces the average cryptographic loop down to just 0.24 seconds per round, establishing around 55 times speedup over the element-wise baseline and successfully validating the bit-shifted slot parallelization approach.

Nevertheless, the lattice-based depth-0 CKKS pipeline achieves the highest computational throughput, averaging a negligible 0.012 seconds per round. This performance gap highlights a core hardware-level trade-off. That is, although bit-shifting allows integer cryptosystems to simulate vector parallelization within a single call, the underlying processor must still calculate math over an expansive 2048-bit modulus space (n^2). In contrast, CKKS processes vector slots natively through polynomial ring arithmetic over standard hardware-sized integers, giving it an inherent runtime advantage. The bars on Figure 3 might not be clear enough to view the numerical values of each bar, it is suggested that the exact values in the logs CSV provided in the repository are read to better understand the comparison.

C. *Parameter Precision and Cryptographic Noise Analysis*

The long-term tracking of weight deviations relative to the unencrypted plaintext baseline presents the core technical finding of this experiment, captured on the logarithmic scale of Figure 4. Across the entire 500-round horizon, the integer-packed Paillier framework optimized with the zero-point offset layer maintains tighter parameter precision than the depth-0 CKKS alternative. At the 75-round mark, the weight Mean Squared Error (MSE) plateaus at 1.30 x 10^{-5} for the CKKS pipeline, whereas the packed Paillier scheme records a tighter error of 5.24 x 10^{-8}.

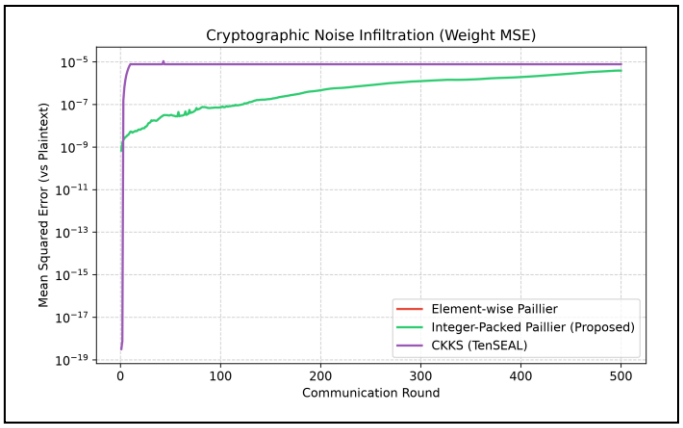

Fig. 4. *Noise infiltration (weighted MSE) curve over the rounds for CKKS vs Integer-Packed Paillier*

This data shows the basic differences between how these two ciphers handle data precision. To satisfy semantic security, the CKKS cryptosystem is forced to inject a random Gaussian error polynomial directly into the ciphertext coefficients during encoding, setting a permanent cryptographic noise floor. While locking the multiplication depth to zero prevents this error from expanding uncontrollably, the model cannot escape this baseline noise floor.

Conversely, because Paillier is a deterministic, exact integer cryptosystem, it adds zero mathematical noise during encryption or server-side additions. The only parameter deviation comes from the engineering quantization layer used to scale and round decimal weights into integers. By integrating the zero-point offset rules from [9] and the secure boundary padding concepts from [10], the proposed custom packing design shifts all negative gradient updates into a strictly positive integer space. This entirely eliminates sign-bit bleeding across slot columns.

Consequently, the experiment demonstrates a powerful conclusion for privacy-preserving architectures: for low-depth federated learning loops, the predictable decimal rounding errors of an asymmetric integer-packing framework are significantly lower and less disruptive than the native cryptographic noise floor embedded inside approximate polynomial ciphers.

V. CONCLUSION

This work presented a rigorous empirical comparison between a custom integer-packed Paillier framework and an optimized depth-0 CKKS baseline, evaluated across a long horizon of 500 global communication rounds in a federated learning architecture. Using a 55-parameter Multi-Layer Perceptron trained on a partitioned subset of the Iris classification dataset, the experimental results successfully validated that both cryptographic paradigms preserve absolute model utility. Both frameworks converged along identical optimization pathways, matching the training loss trajectory and the final 73.33% validation classification accuracy of the unencrypted plaintext control model. This confirms that homomorphic masking can be integrated smoothly into decentralized architectures to mitigate gradient inference and shadow-model attacks without degrading the underlying machine learning capability.

The core architectural contribution of this study lies in revealing an important engineering trade-off between computational execution latency and long-term parameter precision. While the lattice-based depth-0 CKKS pipeline achieved the highest operational throughput, averaging a negligible 0.012 seconds per round compared to the 0.24 seconds achieved by the packed Paillier configuration, the integer-packed design demonstrated superior weight fidelity. Optimized with an asymmetric uniform quantization layer and a zero-point offset derived from [9] and [10], the packed Paillier scheme completely eradicated sign-bit bleeding across bit-shifted slots, recording a final weight Mean Squared Error (MSE) of 5.24 x 10^{-8} compared to the 1.30 x 10^{-5} floor observed in the CKKS pipeline. This demonstrates that for low-depth federated averaging workloads, the predictable decimal rounding truncations required by integer-mapping layers yield tighter parameter precision than the native, non-deterministic cryptographic noise floor injected by approximate polynomial frameworks to maintain semantic security. This provides a clear design guide for engineers balancing runtime constraints and weight fidelity in privacy-preserving machine learning systems.

SOURCE CODE REPOSITORY

The complete source code implementation for the cryptographic solutions and federated learning simulations can be accessed here:
https://github.com/RunningPie/integer_packed_paillier_poc

VIDEO LINK AT YOUTUBE

An explanatory video regarding the overview of this work is also provided here:
https://youtu.be/b3Er2VDXrG8

ACKNOWLEDGEMENT

The author expresses heartfelt gratitude to God Almighty for granting the strength and opportunity to write and complete this paper. The author also would like to express gratitude to Prof. Dr. Ir. Rinaldi Munir, M.T., the lecturer of the II4021 Cryptography course, for his guidance and knowledge throughout the course.

REFERENCES

[1] H. Fang and Q. Qian, "Privacy Preserving Machine Learning with Homomorphic Encryption and Federated Learning," Future Internet, vol. 13, no. 4, p. 94, Apr. 2021, doi: https://doi.org/10.3390/fi13040094.

[2] Y. Pan, Z. Chao, W. He, Y. Jing, L. Hongjia, and W. Liming, "FedSHE: privacy preserving and efficient federated learning with adaptive segmented CKKS homomorphic encryption," Cybersecurity, vol. 7, no. 1, Jul. 2024, doi: https://doi.org/10.1186/s42400-024-00232-w.

[3] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership Inference Attacks against Machine Learning Models," arXiv:1610.05820 [cs, stat], Mar. 2017, Available: https://arxiv.org/abs/1610.05820

[4] N. Dowlin et al., "CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy," Proceedings of Machine Learning Research, vol. 48, 2016, Available: https://proceedings.mlr.press/v48/gilad-bachrach16.pdf

[5] T. Yang et al., "Applied Federated Learning: Improving Google Keyboard Query Suggestions," arXiv.org, 2018. https://arxiv.org/abs/1812.02903

[6] M. Ammad-ud-din et al., "Federated Collaborative Filtering for Privacy-Preserving Personalized Recommendation System," arXiv.org, Jan. 29, 2019. https://arxiv.org/abs/1901.09888

[7] C. Zhang et al., "BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning," 2020. Available: https://www.usenix.org/system/files/atc20-zhang-chengliang.pdf

[8] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A Full RNS Variant of Approximate Homomorphic Encryption," Selected Areas in Cryptography – SAC 2018, pp. 347–368, 2019, doi: https://doi.org/10.1007/978-3-030-10970-7_16.

[9] B. Jacob et al., "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," 2017. Available: https://arxiv.org/pdf/1712.05877

[10] H. Mehta, "Secure inference of quantized neural networks," Mit.edu, 2020. https://dspace.mit.edu/entities/publication/423ea244-8c75-48f5-abd1-42f35f59d063 (accessed Jun. 19, 2026).

DECLARATION

The author hereby declares that this paper is of the author's own work, not an adaptation or translation from other people's work, and most importantly not plagiarism.

Bandung, 19th of June 2026

Dama Dhananjaya Daliman (18222047)