Towards Privacy-Preserving Federated Learning using Hybrid Homomorphic Encryption

Ivan Costa(✉), Pedro Correia, Ivone Amorim, Eva Maia, and Isabel Praça

GECAD, ISEP, Polytechnic of Porto, 4249-015 Porto, Portugal
ivcsi@isep.ipp.pt

Abstract. Federated Learning (FL) enables collaborative training while keeping sensitive data on clients' devices, but local model updates can still leak private information. Hybrid Homomorphic Encryption (HHE) has recently been applied to FL to mitigate client overhead while preserving privacy. However, existing HHE-FL systems rely on a single homomorphic key pair shared across all clients, which forces them to assume an unrealistically weak threat model: if a client misbehaves or intercepts another's traffic, private updates can be exposed. We eliminate this weakness by integrating two alternative key-protection mechanisms into the HHE-FL workflow. The first is masking, where client keys are blinded before homomorphic encryption and later unblinded homomorphically by the server. The second is RSA encapsulation, where homomorphically encrypted keys are additionally wrapped under the server's RSA public key. These countermeasures prevent key misuse by other clients and extend HHE-FL security to adversarial settings with malicious participants. We implement both approaches on top of the Flower framework using the PASTA/BFV HHE scheme and evaluate them on the MNIST dataset with 12 clients. Results show that both mechanisms preserve model accuracy while adding minimal overhead: masking incurs negligible cost, and RSA encapsulation introduces only modest runtime and communication overhead.

Keywords: Federated Learning · Privacy-Preserving Machine Learning · Hybrid Homomorphic Encryption · Secure Aggregation · Threat Models · Key Protection · Adversarial Clients · Masking · RSA Encapsulation · Internet of Things

1 Introduction

Federated learning (FL) is a distributed machine learning approach that allows a group of clients, such as Internet of Things (IoT) devices or healthcare entities, to collaboratively train a global model without sharing their sensitive data. In FL systems, clients train local models using their own data and are required to share only the model parameters. In a centralized FL system, the local models are sent to a server that is responsible for coordinating the training process by

aggregating the received updates to form an improved global model. Once updated, the global model parameters are sent back to the clients, allowing them to continue local training [26]. On the other hand, decentralized FL systems do not rely on a central server, and clients coordinate this process between them. In either case, raw data never leaves clients' devices, which is the reason why FL is often described as "privacy-preserving" [2]. However, recent studies have demonstrated that local model updates can still leak private information, enabling adversarial techniques such as gradient-based data reconstruction attacks, membership inference attacks, and property inference attacks [31,32,30,40]. Various techniques have been applied to overcome this problem, such as Differential Privacy (DP), Homomorphic Encryption (HE), and Multi-Party Computation (MPC) [37]. However, these techniques also bring their own limitations: MPC is not appropriate for resource-constrained settings with several clients, DP adds noise to the training process, and the main problems of HE, namely ciphertext expansion and noise growth, make it impractical for real world use cases [41,19]. Nonetheless, the property of computing over encrypted data, inherent to HE, is extremely valuable to ensure privacy-preserving settings. This has motivated the research for methods that employ HE more efficiently.

Recently, Hybrid Homomorphic Encryption (HHE) has emerged as a promising approach, combining a lightweight symmetric cipher with a HE scheme [1,12]. In this type of scheme, clients encrypt data symmetrically and send both the ciphertext and the key encrypted under HE to the server. The server homomorphically evaluates the symmetric decryption circuit to transform the symmetric ciphertext into a homomorphic one, enabling computations on encrypted data without exposing sensitive information. This reduction in client-side cost makes HHE especially attractive for IoT and other resource-constrained devices. The application of HHE in FL is still recent, and to the best of our knowledge only two works have proposed concrete systems in this direction. Correia et al. [9] introduced the first HHE-FL system, based on the combination of the PASTA cipher and the BFV scheme. Nguyen et al. [29] later proposed a similar design using the Rubato cipher together with FV and CKKS, relying on a trusted key dealer to reduce client responsibilities. While these works demonstrate the practicality of HHE-FL, they share a critical weakness: all clients rely on the same homomorphic key pair. As a result, their security depends on an unrealistically weak threat model in which no client acts maliciously. In practice, if one client intercepts another's communication, it could recover private model updates.

Our work addresses this fundamental liability. We propose two alternative key-protection mechanisms that eliminate the shared-key weakness, thereby extending HHE-FL security beyond the honest-but-curious assumption to settings with malicious clients. Concretely, our contributions are:

1. Identification and elimination of the shared-key liability present in prior HHE-FL systems, which made prior systems vulnerable to malicious clients intercepting communications.
2. Integration of two alternative countermeasures into the HHE-FL workflow: (i) a masking mechanism, where client keys are blinded before homomorphic

encryption and unblinded by the server; and (ii) RSA encapsulation, where homomorphically encrypted keys are additionally wrapped under the server's RSA public key.
3. Security analysis under an extended threat model that includes malicious clients capable of intercepting communications.
4. Full implementation and empirical evaluation of both approaches on the Flower framework with the PASTA/BFV HHE scheme, using the MNIST dataset. Results show that both mechanisms preserve model accuracy while adding negligible (masking) or modest (RSA encapsulation) overhead.

The remainder of the paper is organized as follows: Section 2 introduces background concepts on FL, HE, and HHE. Section 3 surveys related work and highlights common challenges. Section 4 presents our proposed approaches and threat model. Section 5 discusses implementation details, and Section 6 reports our experimental results. Finally, Section 7 concludes with a summary and directions for future work.

2 Preliminaries

This section reviews background needed for our setting: centralized federated learning and aggregation, homomorphic encryption with a focus on BFV scheme, and HHE with HE-friendly ciphers (PASTA) and the homomorphic evaluation of symmetric decryption (HESD).

2.1 Introduction to FL

FL is a distributed machine learning paradigm that enables multiple devices or organizations to collaboratively train a shared global model without exposing their local data. FL can be *centralized*, where a central server manages multiple clients and orchestrates the training process, or *decentralized*, where no central server is required [21]. In this work we focus on the centralized paradigm, in which clients share model updates with the server rather than raw data [2]. Although raw data stays on client devices, updates can still leak private information through attacks such as gradient inversion, membership inference, and property inference. This motivates the integration of privacy-preserving techniques into FL.

Aggregation algorithms are central to FL: they combine local model updates into a global model, and the choice of algorithm impacts both model accuracy and computational overhead [27]. The most common is *Federated Averaging* (FedAvg) [22]: with client $k \in \{1, \ldots, K\}$ holding n_k samples and sending local weights w_{t+1}^k, the global update at round $t+1$ is

$$w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k, \quad \text{with } n = \sum_{k=1}^{K} n_k.$$

Simple Averaging (SimpAvg) is the unweighted counterpart. Besides FedAvg and its unweighted variant SimpAvg, other aggregation methods have been explored in the literature, such as FedSGD and dimension-reduction techniques [39,16], but these are outside the scope of our work, since in our evaluation we employ centralized FL with FedAvg.

FL deployments can be categorized as *cross-silo*, involving a small number of institutional clients (e.g., hospitals), or *cross-device*, involving many resource-constrained clients such as IoT devices that connect intermittently. We emulate the latter scenario, where communication bandwidth, client drop-outs, and limited computational resources are the main bottlenecks [20,24].

Despite the fact that raw data never leaves client devices, FL must still contend with both external (e.g., eavesdroppers) and internal threats (e.g., malicious participants). A common model is the *honest-but-curious* server, which follows the protocol but attempts to infer private information from the updates it receives. Local model updates embed subtle patterns that may leak sensitive information [31,32,30,40], showing that the often-cited guarantee that "no raw data leaves the device" is insufficient for strong privacy.

2.2 Homomorphic Encryption

HE is a cryptographic technique that supports computation over encrypted data, resulting in ciphertexts that, when decrypted, produce the same result as if the operations had been applied to the plaintext [34]. The type and number of operations that an HE scheme can support determine its classification. Namely, a *Partially Homomorphic Encryption* (PHE) scheme allows a single type of arithmetic operation (e.g., addition or multiplication) an unlimited number of times; *Somewhat Homomorphic Encryption* (SWHE) supports a limited set of operations, with restrictions on how many times they can be applied; and *Fully Homomorphic Encryption* (FHE) enables multiple types of arithmetic operations to be performed an unlimited number of times.

Currently, there are four main FHE schemes: TFHE (Fast Fully Homomorphic Encryption over the Torus) [7], which operates at the bit level; BGV [5] and BFV [15], which are very similar and allow for exact computations over integers; and CKKS [6], which supports approximate computations over complex numbers. However, since the security of HE schemes relies on complex mathematical problems, they tend to be impractical for real-world use cases, especially FHE schemes [17]. Two of the main limitations of FHE are *ciphertext expansion* and *noise growth* [25]. The former refers to the increase in ciphertext size when encrypting a plaintext or after every computation, and the latter refers to the accumulation of noise after each operation, especially costly ones such as multiplication. These limitations cause two major problems for implementation: ciphertext expansion makes communication, storage, and computation significantly more expensive, while noise growth limits the depth of computable circuits, since correct decryption cannot be ensured if the noise grows beyond a certain threshold [12].

To mitigate these drawbacks, new approaches such as HHE have been proposed. The basic idea consists of combining HE with SE, which is known for its fast encryption, efficient transmission, and constant ciphertext expansion.

2.3 HE-friendly Ciphers

HHE is a cryptographic approach that combines symmetric ciphers with HE schemes to improve efficiency. In this setting, symmetric encryption is used for data encryption, while HE enables secure computations on the encrypted data. One notable example is the *transciphering* framework [28], which securely converts symmetric ciphertexts into homomorphic ciphertexts for further processing. This hybrid approach significantly reduces the client-side computational burden, as the client does not need to homomorphically encrypt all of its data. It also minimizes communication overhead, since symmetric ciphertexts are more compact than homomorphic ones. These advantages make HHE especially attractive for IoT and other resource-constrained devices. However, they come at the cost of increased computational load on the server, which must handle the more demanding homomorphic evaluation.

To alleviate the server-side overhead, several symmetric ciphers have been specifically designed for compatibility with HE. These so-called HE-friendly ciphers reduce the cost of the *Homomorphic Evaluation of Symmetric Decryption* (HESD) step, thereby improving the overall practicality of transciphering-based approaches. The most well-known HE-friendly ciphers include PASTA [12], HERA [8], and Elisabeth [10]. PASTA is a stream cipher designed to minimize multiplicative depth, optimized for integer use cases over $\mathbb{F}_p$ with p being a 16-bit prime. It is compatible with both the BGV and BFV schemes. HERA, in contrast, uses a randomized key schedule defined over $\mathbb{Z}_q$, with $q > 2^{16}$, and is primarily tailored for CKKS, though it remains compatible with BGV and BFV. Elisabeth targets TFHE and provides a wide range of operations for homomorphic evaluation on the server side. It is defined over $\mathbb{Z}_q$ with $q = 2^4$.

The authors of PASTA released an open-source C++ framework[1], developed specifically for benchmarking HHE schemes. It includes implementations of several HE-friendly ciphers, such as PASTA, together with various HE schemes, including BGV via HElib [18], BFV via Microsoft SEAL [13], and TFHE using the TFHE library[2].

3 Related Work

Several works have applied HE to enhance privacy and security in FL protocols. Some proposals rely on simplified schemes, namely PHE, as their main cryptographic primitive [33,36,42]. Other works employ FHE as the main component, typically within centralized topologies, since the substantial computational overhead of FHE requires a server to handle most of the processing. The advantage

[1] https://github.com/isec-tugraz/hybrid-HE-framework
[2] https://github.com/tfhe/tfhe

of this approach is flexibility: any aggregation algorithm can be used, as FHE supports arbitrary computations. For example, Ma et al. [23], Zhang et al. [43], and Duy et al. [14] apply FedAvg, while others have explored alternatives such as SimpAvg, FedSGD, or dimension-reduction techniques [38,16,39]. However, even in centralized designs, the computational burden on clients remains high, which makes these systems impractical for resource-constrained environments.

To address these limitations, recent works have proposed using HHE instead of FHE in FL, thereby reducing the client-side overhead. The first HHE-based FL system was introduced by Correia et al. [9], who designed a centralized topology that combines BFV with the HE-friendly cipher PASTA. Their implementation, built on the PASTA framework[3] and integrated into Flower [4], showed that model performance can be maintained while significantly reducing client communication costs. However, server-side computation increased substantially due to the HESD step, and, more critically, all clients shared the same HE key pair, which weakens the threat model by enabling key misuse if one client acts maliciously. For instance, a client with malicious intentions, Eve, intercepts the communication between another client, Alice, and the server. This exchange between Alice and the server are two ciphertexts: one PASTA ciphertext encrypting Alice's model weights, and a BFV ciphertext encrypting Alice's PASTA key. Eve then attempts to access the secret information hidden in these ciphertexts. Even though she can not decrypt the PASTA ciphertext directly, she can use the BFV secret key (Eve and Alice have commom BFV keys) to decrypt the BFV ciphertext, obtaining Alice's PASTA key. Following this, Eve can use this key to decrypt the PASTA ciphertext and retrieve Alice's model weights. She can then apply any of the well known attacks for classic FL systems, such as inference attacks.

At the same time, another work proposing a centralised HHE-FL system, that shares the above issue, was proposed by Nguyen et al. [29]. However, in this work the authors use the RtF framework, which combines the Rubato cipher with FV and CKKS. In their design, model parameters are encrypted under Rubato, while the Rubato key is encrypted under FV by a Trusted Key Dealer (TKD), which partially addresses the key management challenge but places strong trust in the trusted key dealer. The server then performs HESD and aggregation (via SimpAvg) on the converted ciphertexts. Their results were consistent with Correia et al., showing preserved accuracy and reduced client communication, at the expense of higher server-side computation.

Our work addresses this open problem by eliminating the vulnerabilities associated with shared-key usage in HHE-FL. We integrate two alternative key-protection mechanisms into the workflow: *masking*, where client keys are blinded before homomorphic encryption and later unblinded by the server, and *RSA encapsulation*, where homomorphically encrypted keys are additionally wrapped under the server's RSA public key. These approaches preserve the efficiency benefits of HHE while extending its security to adversarial settings with malicious participants.

[3] https://github.com/isec-tugraz/hybrid-HE-framework

4 Threat Model and Proposed Approaches

In this section, we present our threat model and describe the proposed countermeasures that eliminate the shared-key liability in prior HHE-FL systems. We then introduce the overall architecture, provide a logical decomposition of the entities and their components, and describe the complete workflow of our approaches.

4.1 Threat Model

The security of the proposed approaches is based on the following assumptions about the setup and participants:

A.1. Trusted setup phase: All cryptographic keys, certificates, and auxiliary values (e.g., masks) are generated and distributed securely, with no adversarial interference.

A.2. Clients are honest-but-curious but may maliciously intercept communications: They follow the protocol correctly but may attempt to infer private information from intercepted messages.

A.3. The server is honest-but-curious: It executes the protocol correctly but may try to infer private information.

A.4. No collusion occurs between the server and clients.

A.5. External adversaries may attempt to intercept communications but cannot compromise the underlying cryptographic primitives.

Under these assumptions, our system provides the following guarantees:

G.1. Confidentiality of individual client updates during transmission.

G.2. Confidentiality of individual client updates during aggregation at the server.

G.3. Confidentiality of the aggregated model weights.

G.1 is ensured by client-side encryption: each client encrypts its local update with a symmetric key, and this key is further protected via masking or RSA encapsulation before being encrypted under the homomorphic public key. Thus, even if a client intercepts another's message, the symmetric key cannot be misused. *G.2* is ensured because the server applies HESD to convert symmetric ciphertexts into homomorphic ciphertexts, but never learns any plaintext update; without the homomorphic secret key, the server cannot decrypt intermediate results. Finally, *G.3* follows from the fact that aggregation is performed directly over homomorphic ciphertexts, keeping the global model encrypted until decryption by authorized clients.

Comparison with Prior Threat Models. The threat models of existing HHE-FL systems are weaker. Correia et al. [9] assume honest clients and an honest-but-curious server, leaving the system vulnerable because all clients share the same HE key pair. Nguyen et al. [29] adopt a similar semi-honest model but rely on a trusted key dealer to manage key encryption, introducing an additional trusted party. In contrast, our model explicitly considers malicious clients capable of intercepting traffic, and our countermeasures (masking and RSA encapsulation) eliminate this liability without introducing new trusted entities.

4.2 System Overview

In this work, similarly to the work of Correia et al. [9], we consider three main entities: the *Third-Party Authority* (TPA), the *Server*, and the *Clients*. A high-level overview of the architecture is shown in Fig. 1, showcasing all three entities, with only a single client for illustration purposes.

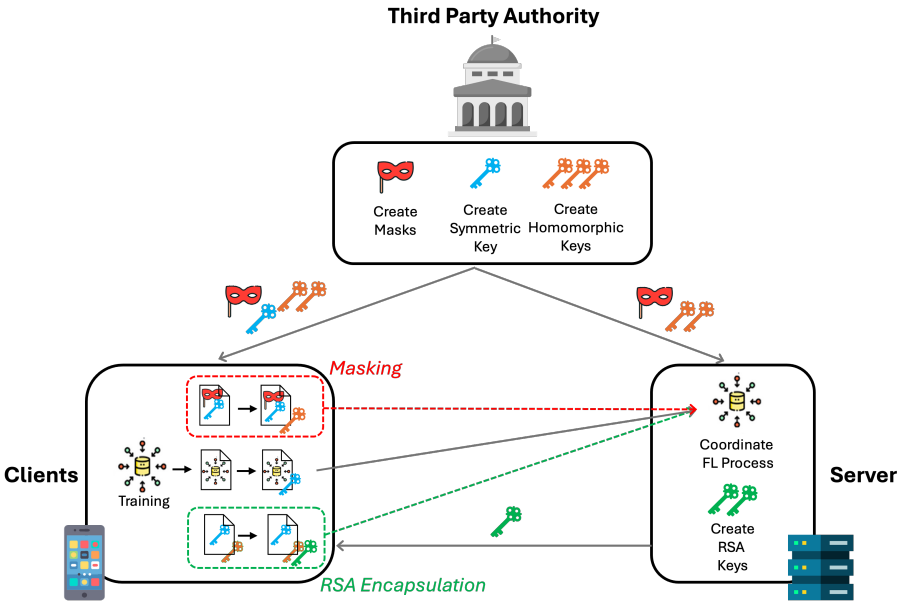

Fig. 1. Architecture of the proposed approaches

The TPA is fully trusted and is responsible for generating and securely distributing all PASTA keys and BFV key pair to the clients (one PASTA key for each client and one BFV key pair for all clients), as well as the masks, in the *Masking* scenario, or the server's RSA key pair, in the *RSA Wrapping* scenario. The *Server* coordinates the FL training process, gathers encrypted updates from *Clients*, performs secure aggregation, and distributes the updated global model parameters. In the *RSA encapsulation* approach, the *Server* also manages the RSA protocol. The *Clients* train their local models with private data, encrypt their local updates and symmetric key, send them to the *Server*, and then decrypt the received global model parameters after aggregation.

4.3 Logical View

A logical view of the system is presented in Fig. 2. This view decomposes the system into three main components, one for each entity. On the TPA side (blue

in Fig. 2), the *Key Management Module* is responsible for generating and securely distributing cryptographic keys to both client and server. In the *Masking* approach, it is also responsible for creating a mask for each client, which is sent to the server and the respective client. On the client side (orange), the *Training Module* trains the local model using data from the *Local Dataset*. The *Encryption Module* then encrypts the local model updates and the symmetric key, sending the encrypted data to the server. Once the updated global model is received, the client's *Encryption Module* decrypts it, and the *Training Module* incorporates the global weights into the local model. On the server side (green), the *HESD Module* performs HESD on the received model updates, generating homomorphic ciphertexts. These ciphertexts are then passed to the *Aggregator Module*, which computes the new global model. The aggregated global model is then sent back to the clients. In the approach using *RSA encapsulation*, the *Certificate Authority Module* generates certificates containing the server's RSA public key, ensuring its authenticity. On the server side, the *RSA Module* manages the server's RSA key pair and handles the distribution of the public key and its certificate to the clients.

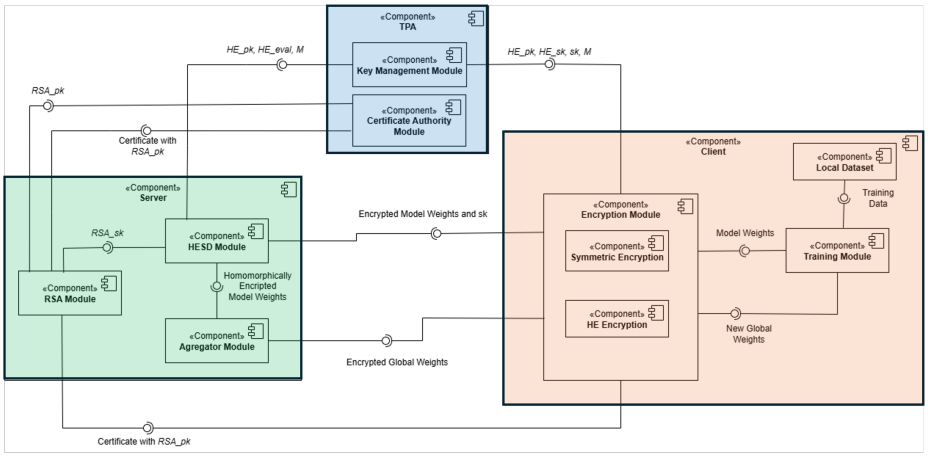

Fig. 2. Logical View of the Proposed System

4.4 Workflow

The workflow of our proposed approaches consists of five main phases: *Setup Phase*, *Client Training Phase*, *Server Aggregation Phase*, *Client Evaluation Phase*, and the *Server Evaluation Phase*. The full workflow is depicted in Fig. 3.

In the *Setup Phase*, the TPA generates the set of homomorphic keys, (*HE_pk*, *HE_sk*, *HE_eval*) where *HE_pk* is the homomorphic public key, *HE_sk* is the homomorphic secret key, and *HE_eval* is the homomorphic evaluation key. Both

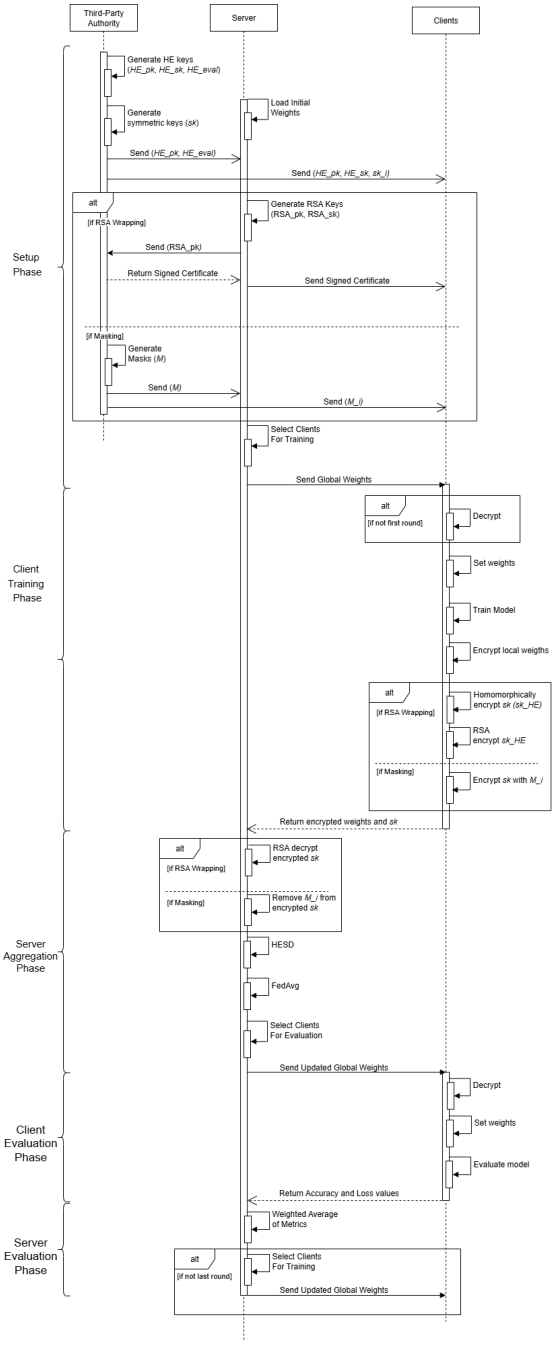

Fig. 3. Workflow of the proposed approaches

HE_eval and HE_pk are sent to the server, while HE_pk and HE_sk are sent to the clients. Additionally, the TPA generates a unique symmetric key sk_i for each client i, where $1 \leq i \leq N$ and $N \in \mathbb{N}$ denotes the total number of clients. In the approach using *Masking*, the TPA also generates a unique mask M_i for each client i and sends it to them. This mask is also shared with the server along with the corresponding client identifiers, enabling the server to associeate each client with its mask. Alternatively, in the approach using *RSA encapsulation*, the server is the one responsible for generating a single tuple of RSA keys, (RSA_pk, RSA_sk), where RSA_pk is the RSA public key and RSA_sk is the corresponding secret key. To guarantee the authenticity of the server, the server requests the TPA to sign the RSA_pk. After receiving the signed certificate, the server shares it with all clients. Then the server initializes a model and sends its weights to the clients, who use them as the starting point for local training.

After the initial phase, it follows the *Client Training Phase*, where each client trains its local model, and then encrypts the obtained model weights w_i using sk_i. In the approach using *Masking*, the clients compute

$$HE.Enc(sk + M).$$

In the *RSA encapsulation* approach the clients encrypt sk_i with HE_pk, producing sk^i_{HE}, which is then encrypted with the server's RSA_pk from the certificate, generating $RSA.Enc(sk^i_{HE})$. Note that we first encrypt with HE and then encapsulate with RSA; RSA cannot handle such large plaintexts directly, so wrapping the (already homomorphically encrypted) key in RSA avoids plaintext size limits. The resulting ciphertexts w^i_{SKE} and, $HE.Enc(sk+M)$ or $RSA.Enc(sk^i_{HE})$, depending on the approach, together with the number of training samples

$$n_i = \frac{training\_data_i}{batch\_size},$$

are sent to the server for aggregation.

In the *Server Aggregation Phase*, the server has to first obtain sk^i_{HE} from either $HE.Enc(sk + M)$ or $RSA.Enc(sk^i_{HE})$. In the approach using *Masking*, this is done by subtracting the masks M_i from the corresponding clients, and in the other approach, the server applies the RSA decryption function $RSA.Dec$ with the RSA secret key RSA_sk on the RSA encrypted keys $RSA.Enc(sk^i_{HE})$ to recover the homomorphically encrypted keys sk^i_{HE}. Then, the server applies HESD on each w_{SKE} using the associated sk^i_{HE}, obtaining w_{HE}. Once all updates are transformed, the server aggregates the homomorphic ciphertexts to compute the new global model weights. Finally, the server selects a new subset of clients and sends them the updated model for evaluation.

The fourth phase, *Client Evaluation*, consists of selecting a subset of clients to receive the newly aggregated global model weights and decrypt them using HE_sk. They update their local models with these weights and evaluate them on their local test dataset. Each client then sends the resulting accuracy and loss metrics to the server, along with the number of test samples.

The *Server Evaluation Phase* is the last phase in our workflow and it requires the server to aggregate the evaluation metrics received by the clients, using a chosen aggregation algorithm.

Masking and RSA Wrapping Correctness For the correctness of the *Masking* approach, we need to ensure that when a client sends an homomorphic ciphertext encrypting the masked key, the server can retrieve the homomorphic ciphertext corresponding to the key, by removing the respective mask. More specifically, client i sends

$$HE.Enc(sk_i + M_i)$$

to the server. Since the server knows the respective mask M_i, it computes

$$HE.Enc(sk_i + M_i) - M_i = HE.Enc(sk_i),$$

which is possible due to the homomorphic properties of $HE.Enc()$. This proves that the server can correctly retrieve $HE.Enc(sk_i)$ from $HE.Enc(sk_i + M_i)$.

For the correctness of the *RSA Wrapping* approach, the scenario is similar, but instead we want to prove that the server can correctly unwrap the RSA ciphertext to obtain the homomorphic ciphertext of the symmetric key. In this scenario, client i sends to the server

$$RSA.Enc(sk^i_{HE}),$$

which was encrypted using the server's RSA public key RSA_pk. This means that the server can just use its RSA secret key, RSA_sk, to decrypt this ciphertext as follows

$$RSA.Dec(RSA.Enc(sk^i_{HE})) = sk^i_{HE} = HE.Enc(sk_i).$$

This shows that the server can retrieve $HE.Enc(sk_i)$ correctly.

Discussion. Both proposed mechanisms strengthen HHE-FL against malicious clients, but with different trade-offs. Masking is lightweight and incurs negligible computational and communication overhead, but it requires careful management of masks by the server. RSA encapsulation builds on widely deployed public-key cryptography, reducing trust in mask management, but introduces additional cost for key encapsulation and decapsulation. Compared to Correia et al. [9], our approaches remove the unrealistic assumption that all clients are fully honest while still avoiding reliance on a trusted key dealer as in Nguyen et al. [29]. This positions our work as a more practical solution for adversarial FL settings, particularly in cross-device deployments with resource-constrained clients.

4.5 Security Analysis

We now analyze the security of the proposed mechanisms under the threat model defined in Section 4.1. Our goal is to show that the confidentiality guarantees (G.1–G.3) are satisfied assuming the hardness of the underlying cryptographic primitives.

Masking-based protection. Each client i transmits $(w^i_{SKE}, HE.Enc(sk_i + M_i))$, where w^i_{SKE} are model parameters encrypted under the symmetric key sk_i, and M_i is a random mask. Since M_i is uniformly sampled and unknown to other clients, $HE.Enc(sk_i + M_i)$ is computationally indistinguishable from an encryption of a random value under the semantic security of HE. Consequently, even if another client intercepts this message, it cannot extract sk_i or w^i_{SKE} without the secret mask M_i or the homomorphic secret key. At the server, the mask is subtracted in the homomorphic domain, yielding $HE.Enc(sk_i)$; security follows directly from the IND-CPA property of the BFV scheme.

RSA encapsulation. In this variant, each client sends w^i_{SKE}, together with $RSA.Enc(HE.Enc(sk_i))$. The symmetric key is first encrypted under HE and then encapsulated with the server's RSA public key. By the semantic security of RSA (under the RSA assumption in the random oracle model), only the server can recover $HE.Enc(sk_i)$. Therefore, even if another client intercepts the transmission, it cannot obtain sk_i or w^i_{SKE}. This removes reliance on mask management while preserving confidentiality.

Aggregation and server-side confidentiality. For both approaches, the server converts symmetric ciphertexts into homomorphic ciphertexts via HESD using only $HE.Enc(sk_i)$. At no point does the server learn sk_i or w^i in plaintext. By the security of the BFV scheme, intermediate ciphertexts remain IND-CPA secure, and aggregation is performed entirely in the homomorphic domain. Thus, both individual client updates and the aggregated model remain confidential until decryption by authorized parties.

Comparison to prior work. Correia et al. [9] and Nguyen et al. [29] rely on a single global HE key pair across all clients, leaving them vulnerable to key misuse if one client intercepts another's ciphertext. Our mechanisms eliminate this liability: masking relies on independent random masks, and RSA encapsulation ensures that only the server can access $HE.Enc(sk_i)$. Hence, our system achieves confidentiality against malicious clients, unlike prior HHE-FL systems.

Security claim. Formally, assuming (i) the IND-CPA security of the BFV scheme, (ii) the semantic security of the symmetric cipher (PASTA), and (iii) the RSA assumption, we claim that an adversary controlling a subset of clients and intercepting all network traffic cannot recover any other client's plaintext updates with probability non-negligibly better than random guessing. Additionally, since all this primitives were chosen to ensure 128-bit security, we can assume that the overall system also ensures the 128-bit security.

Analysis on Combined Security In the proposed system, specifically in the *RSA Wrapping* approach, we construct a ciphertext consisting of two layers of encryption: first a BFV encryption and then a RSA encryption, generating the ciphertext $RSA(BFV(m))$ encrypting m. Since the BFV ciphertexts have to be split into chunks before being encrypted with RSA, some of its security properties

might be lost. Therefore, we consider the security of these combined ciphertexts to be ensured by the security level of the RSA scheme, which is 128-bits. We also discuss the unwrapping and the *transciphering* steps to show that no leakage happens during these operations. The former consists of the transition

$$RSA(BFV(m)) \rightarrow BFV(m)$$

which happens by applying the RSA decryption algorithm using the server's RSA secret key. Due to the IND-CPA security of the BFV scheme, we know that the server can not distinguish the chunks from random noise, which ensures that there are no leakage. The latter consists of the transition

$$PASTA(m) \rightarrow BFV(m)$$

which happens by homomorphically evaluating PASTA's decryption algorithm using $BFV(k)$, which decrypts the PASTA ciphertext within BFV's ciphertext space, where k is PASTA's symmetric key. The IND-CPA security of BFV ensures that ciphertexts resulting from the evaluation algorithm satisfy the indistinguishability condition, which ensures that there is no leakage during the *transciphering* step.

5 Implementation Details

In this section, we present the details behind the implementation of our approaches. We use, as a foundation, the implementation of Correia et al., which was made available to us by the authors. This implementation uses the Flower framework for FL and the combination of PASTA and BFV as the HHE scheme. For the cryptographic layer, we rely on the code made available by the authors of PASTA[4], which is written in C++. For the FL layer, both server and client components are implemented within Flower, which uses gRPC[5] for communication. Each side also has specific subcomponents to coordinate the FL process: the server manages aggregation and evaluation, while clients manage local training and evaluation. Since the PASTA/BFV framework is implemented in C++ and Flower is a Python-based framework, a *pybind11*[6] bridge was used to connect both layers. RSA operations were implemented via the Python *cryptography library*[7], operating outside the C++ cryptographic layer.

Implementation trade-offs. Some design choices were made to balance efficiency and practicality. First, because division is not natively supported in the ciphertext space of BFV, we delegate the final division by the global sample size to the clients, after decryption, rather than performing it at the server. Second,

[4] https://github.com/isec-tugraz/hybrid-HE-framework
[5] https://grpc.io/
[6] https://github.com/pybind/pybind11
[7] https://cryptography.io/en/latest/

for the RSA encapsulation approach, the size of sk_{HE} exceeds the plaintext capacity of RSA. To address this, we apply chunking: the encrypted key is split into segments that fit within the modulus size, encrypted individually, and later recombined at the server after decryption. While this introduces overhead, it is unavoidable for correctness and is evaluated experimentally in Section 6.

Security considerations. Cryptographic material such as symmetric keys, masks, and homomorphic keys is never exposed in plaintext beyond the client or TPA environment. The pybind11 bridge only exchanges serialized ciphertexts, never raw keys, thereby preserving security across the Python/C++ boundary. On the server side, masks are stored only in encrypted form, and RSA key pairs are generated and managed following the recommendations of NIST [3].

Reproducibility. Our implementation builds on publicly available open-source frameworks: the PASTA HHE library and the Flower FL framework. We extended these with modules for RSA encapsulation, masking, and key management. The code, including integration with pybind11 and experimental scripts, will be made available upon publication to facilitate reproducibility and comparison with related work.

Server-Side Processing The server-side processing consists of first performing the HESD operations, which convert the PASTA ciphertexts into BFV ciphertexts, and then performing the *FedAvg* aggregation of model updates. Algorithm 1 shows the chunk-based HESD procedure, in which each chunk of the symmetrically encrypted weights w_{SKE} is converted into a BFV homomorphic ciphertext w_{HE} before aggregation. To perform this transformation, the server first obtains the usable homomorphically encrypted symmetric key sk_{HE} for each client. In the *RSA encapsulation* approach, the server decrypts the RSA-encrypted chunks of $RSA.Enc(sk_{HE})$ and reconstructs the original sk_{HE}. Alternatively, in the *Masking* approach, the server subtracts the known mask M from the ciphertext $HE.Enc(sk + M)$ to recover $sk_{HE} = HE.Enc(sk)$.

After HESD, the aggregation is performed using the FedAvg aggregation algorithm, which multiplies each weight vector by the client's training sample count, and sums the results homomorphically. Since division is not naturally supported in the ciphertext space of BFV, in our approaches we delegate the division by the global total n to the clients.

Client-Side Processing On each client device, processing begins with quantization and encryption of the local model updates using PASTA, and ends with decryption and de-quantization of the aggregated global model. In our implementation, the quantization and encryption strategy, as well as the de-quantization and decryption strategy, follow the approach of Correia et al. [9].

Besides these two steps, each client is also required to securely prepare its symmetric key sk_i for transmission to the server. In the approach using *RSA encapsulation*, the client first computes $sk^i_{HE} = HE.Enc(sk_i)$, splits it into

Algorithm 1 Chunk-Based HESD for a Single Client

Require: Client's encrypted symmetric model parameters w_{SKE}, Client's encrypted symmetric key sk_{HE}, BFV cipher instance `bfv_cipher`
Require: Protocol variant flag: `RSA` or `Masking`
Require: If `RSA`: RSA private key RSA_sk
Require: If `Masking`: Client's Mask M
if `RSA` **then**
  Decrypt sk_{HE}
    `decrypted_chunks` ← empty list
  for each `chunk` in sk_{HE} **do**
    `dec_chunk` ← `RSA.Decrypt(chunk,`RSA_sk`)`
    Append `dec_chunk` to `decrypted_chunks`
  end for
  Reconstruct sk_{HE}
    sk_{HE} ← `concat(decrypted_chunks)`
else if `Masking` **then**
  Remove mask M homomorphically
    $sk_{HE} \leftarrow sk_{HE} - M$
end if
Switch to client's encrypted key
  `BFV_CIPHER.SET_ENCRYPTED_KEY`(sk_{HE})
Perform HESD:
  `he_params` ← empty list
for each `chunk` in w_{SKE} **do**
  `he_chunk` ← `bfv_cipher.HESD(chunk)`
  Append `he_chunk` to `he_params`
end for
return `he_params`

chunks to fit RSA encryption limits, and then encrypts each chunk using the server's RSA_pk. Alternatively, in the approach using *Masking*, the client computes $HE.Enc(sk_i + M_i)$ and transmits this masked ciphertext to the server, as depicted in Algorithm 2.

In summary, our implementation extends the PASTA/BFV HHE framework and the Flower FL framework with two new key-protection mechanisms: masking and RSA encapsulation. Both client and server roles were modified to integrate these mechanisms, while maintaining compatibility with the underlying training pipeline. In the next section, we evaluate these implementations experimentally, focusing on their accuracy, communication, and computation costs, and compare them against the baseline system of Correia et al. [9].

6 Experimental Evaluation

This section presents the experimental evaluation of our proposed approaches. We first describe the experimental setup, dataset, model, and configuration parameters, and then benchmark RSA performance to select suitable key sizes

Algorithm 2 Symmetric Key Encryption

Require: Client Symmetric key sk_i, Homomorphic public key HE_pk, BFV cipher instance `BFV_cipher`
Require: Protocol variant flag: `RSA` or `Masking`
Require: If `RSA`: RSA public key RSA_pk, Max plaintext size `plaintext_size`
Require: If `Masking`: Client Mask M_i

if RSA **then**
  $enc\_key \leftarrow BFV\_cipher.encrypt(sk_i)$
  Split enc_key into chunks
    `chunked_key` $\leftarrow$ empty list
  for $i = 0$ to length(enc_key) step `plaintext_size` **do**
    `chunk` $\leftarrow enc\_key[i : i +$ `plaintext_size`$]$
    Append `chunk` to `chunked_key`
  end for
  Encrypt chunks with RSA_pk
    $sk^i_{HE} \leftarrow$ empty list
  for each $chunk$ in `chunked_key` **do**
    $enc\_chunk \leftarrow RSA.Enc(chunk, RSA\_pk)$
    Append enc_chunk to sk^i_{HE}
  end for
else if Masking **then**
  $sk^i_{HE} \leftarrow BFV\_cipher.encrypt(sk_i + M_i)$
end if
return sk^i_{HE}

for the *RSA encapsulation* variant. Next, we compare both of our mechanisms, *Masking* and *RSA encapsulation*, against the baseline system of Correia et al. [9]. The evaluation covers model quality, communication cost, and computation cost, with results reported for both clients and server.

6.1 Experimental Setup

All experiments were conducted on Google Colab's T4 High-RAM environment, which provides an Intel Xeon CPU @ 2.20GHz, an NVIDIA T4 GPU, 51GB of system RAM, and 15GB of GPU memory, offering hardware acceleration for training. The dataset used was MNIST [11], a widely adopted benchmark for handwritten digit recognition. It contains 70,000 grayscale images of digits (0–9), with 60,000 for training and 10,000 for testing, each of size 28×28 pixels. The training dataset was partitioned into N subsets using Flower's `IidPartitioner`, with each local dataset further split into 80% for training and 20% for local testing. During training, 20% of the local training set was reserved for validation. We used the same convolutional neural network (CNN) architecture as Correia et al. [9], based on the implementation by Sharma [35]. The model consists of two convolutional layers, each followed by LeakyReLU activation and batch normalization, with max-pooling for downsampling. Feature maps are flattened, followed by dropout, and a final dense layer predicts one of 10 classes. The network has approximately 8,000 trainable parameters.

Table 1 summarizes the configuration used across all experiments. The same configuration was adopted in Correia et al. [9], enabling a fair comparison.

Parameter constraints. The HHE scheme used in this work operates in $\mathbb{Z}_q$ with $q = 2^{16}+1$. As a result, server-side computations must remain within $[0, 2^{16}+1)$.

Table 1. Parameters used for the FL experiment

Config. Name	Value	Config. Name	Value
Clients	12	Rounds	10
Epochs	10	Classifier	CNN
Loss function	Categorical Cross-entropy	Batch size	64
Optimizer	Nadam	Learning Rate	0.001
Clients per Training Phase	4	Clients per Evaluation Phase	12
Clip Range (α)	5	Key size	256 bits
Plaintext size	128 bits	Ciphertext size	128 bits
Plaintext Modulus	65537	Polynomial Degree Modulus	16384
Security level	128-bit		

This constrains the product of quantized model weights, the number of participating clients, and the number of training batches per client to remain below 2^{16}. Formally,

$$\text{weights} \times \text{batches\_per\_client} \times \text{training\_clients} < 2^{16} + 1.$$

6.2 RSA Benchmarking

In the RSA encapsulation approach, the homomorphically encrypted key sk_{HE} must be chunked before RSA encryption due to plaintext size limits. To select suitable key sizes, we benchmarked RSA with 1024, 2048, 3072, and 4096-bit keys, measuring encryption and decryption time, ciphertext expansion, and number of chunks.

Results are reported in Table 2. Larger key sizes reduce the number of chunks and ciphertext expansion but increase decryption cost. While RSA-1024 offers the fastest decryption, it is insecure for our setting and produces large ciphertext expansion. RSA-2048 offers moderate trade-offs but does not meet the 128-bit security level required to match PASTA. RSA-3072 and RSA-4096 both achieve adequate security, with RSA-3072 balancing ciphertext size and runtime better, while RSA-4096 slightly reduces communication overhead at the cost of higher decryption latency.

Table 2. Average Performance and Output Size Comparison Across RSA Key Sizes (Ptx - Plaintext ; Ctx - Ciphertext)

RSA Key size (bits)	Enc. (s)	Dec. (s)	Ptx Size (bytes)	Max Ptx Size (bytes)	Total Chunks	Ctx Size (bytes)	Ctx Size (MB)
1024	0.51	3.3	1826326	62	29456	4742416	4.52
2048	0.43	7.91	1826049	190	9611	2777579	2.65
3072	0.31	11.74	1826287	318	5743	2394831	2.28
4096	0.35	18.72	1826243	446	4095	2231775	2.13

Based on these results, we focus our evaluation of the RSA encapsulation approach on RSA-3072 and RSA-4096, as they provide the necessary 128-bit security level.

6.3 Results

The results of the analysis are presented below, highlighting the key findings and their implications.

Model quality. Fig. 4 shows accuracy and loss over training rounds. Our approaches closely match or even slightly outperform the baseline of Correia et al. [9]. Final results are summarized in Table 3. Masking achieves the highest accuracy (98.35%), while RSA-4096 incurs only a minor drop (97.28%). All approaches preserve model quality effectively.

Table 3. Global Model Comparisons

	Accuracy(%)	Loss	Precision(%)	Recall(%)	F1-score(%)
Correia et al.	97.39	0.0962	97.50	97.39	97.39
RSA 3072	98.04	0.0737	98.09	98.04	98.05
RSA 4096	97.28	0.1152	97.38	97.28	97.28
Masking	98.35	0.0622	98.37	98.35	98.35

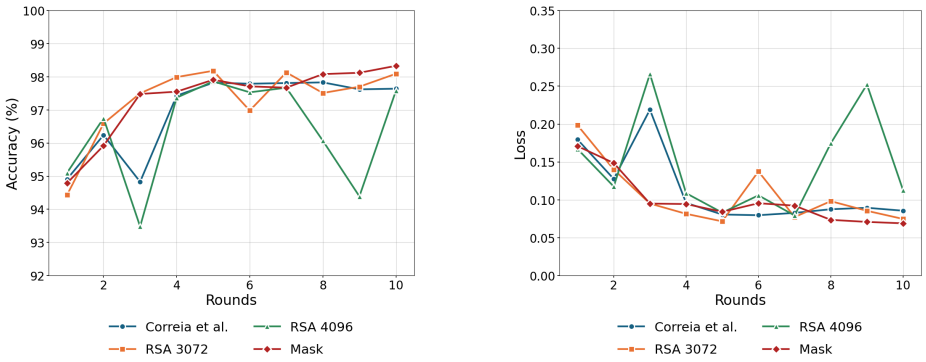

Fig. 4. Training Performance Across Rounds: (a) Accuracy; (b) Loss

Communication costs. For clients, masking incurs the same ciphertext size as Correia et al. [9] (≈1.79 MB per client). RSA encapsulation increases communication by 1.31× (RSA-3072) and 1.22× (RSA-4096). On the server side, communication overhead is identical across all approaches since aggregation occurs in BFV ciphertext space.

Computation costs (clients). Table 4 reports client runtimes. Masking adds negligible overhead (≈0.018s, performed at initialization). RSA encapsulation adds 0.35–0.38s for key chunking and encryption. Overall, client runtimes remain in the 11.9–12.4s range, only slightly higher than the baseline.

Table 4. Average Client Computation Cost

Operation (s)	Correia et al.	RSA-3072	RSA-4096	Masking
De-serialization	0.42480	0.48112	0.47115	0.44298
Decryption	0.42206	0.49451	0.48567	0.46385
Training	10.70805	10.70805	10.70805	10.70805
Evaluation	1.07516	1.07516	1.07516	1.07516
Weights Encryption	0.32867	0.33840447	0.33226	0.33888
RSA Key Encryption	-	0.35177	0.38373	-
Key Masking[a]	-	-	-	0.01845
Serialization	0.00128	0.00240	0.00212	0.00123
Training runtime	11.88486	12.37625	12.38298	11.95498
Evaluation runtime	1.92330	2.05319	2.03410	1.98321

[a] The time reported for Key Masking is not included in the phases runtime since it can be performed during client initialization.

Computation costs (server). Server runtime is dominated by HESD, which is common to all approaches (≈1400s per client). RSA encapsulation adds 12s (RSA-3072) or 19s (RSA-4096) for decryption per client, while masking adds only 0.001s. Table 5 summarizes the results. Although RSA introduces measurable overhead, it remains insignificant compared to HESD.

Table 5. Average Server Aggregation Phase Computation Cost Comparison

Operation (s)	Correia et al.	RSA-3072	RSA-4096	Masking
De-serialization	0.0009	0.0009	0.0008	0.0010
HESD per client	1451.0	1431.5	1439.8	1400.0
RSA Key Decrypt per client	-	12.0	19.3	-
Unmasking per client	-	-	-	0.0014
Aggregation	0.2174	0.2222	0.2244	0.2212
Serialization	0.3629	0.3626	0.3620	0.3528
Total runtime	5804.5803	5774.5857	5836.9872	5600.5806

6.4 Discussion

The results show that both mechanisms preserve accuracy while significantly improving the threat model. Masking is extremely lightweight, with negligible communication and computation overhead. RSA encapsulation provides stronger guarantees without reliance on mask management but at the cost of moderate

runtime and communication overhead. Both approaches outperform prior HHE-FL systems in terms of security while maintaining practicality for cross-device federated learning with resource-constrained clients.

Regarding scalability, all the bottlenecks are expected to appear on the server side, since it has to manage more clients, whereas the client protocols can happen simultaneously. The main issue will be on the HESD step, which is by far the most expensive operation performed on the server side, as the experimental results showed. However, regarding the proposed techniques, we expect the *Masking* approach to outperform the *RSA Wrapping* approach, since it is a much more lightweight solution, taking only 0.0014s per client to remove the masks on the server side. Both RSA-3072 and RSA-4096 introduce a runtime cost of 12s and 19.3s per client, respectively, to perform RSA decryption, which is 8500× and 13500× more than the masking approach, making them obviously worse choices in settings with a large number of clients, since these costs scale proportionally to the number of clients.

7 Conclusion

In this work, we revisited the security assumptions of existing HHE-based federated learning approaches and showed that the common practice of sharing a single homomorphic key pair across all clients leaves prior systems exposed to malicious participants. To address this, we integrated two alternative key-protection mechanisms, *masking* and *RSA encapsulation*, into the HHE-FL workflow. Both mechanisms prevent key misuse by other clients while preserving the efficiency advantages of HHE.

We implemented both variants on top of Flower using the PASTA/BFV stack and evaluated them on MNIST with 12 clients. The results indicate that model quality is maintained (up to 98.35% accuracy with masking) while overheads remain modest: masking incurs negligible runtime and communication cost; RSA-3072/4096 increases client communication by only 1.31 × /1.22× and adds a small client-side time penalty (≈ 1.04–1.05×). On the server, the dominant cost continues to be HESD (shared by all systems), with RSA decapsulation adding a comparatively minor overhead.

Overall, our study demonstrates that it is possible to harden HHE-FL against malicious clients without sacrificing practicality for cross-device deployments. Future work includes (i) exploring lightweight multi-key HE or key-homomorphism to remove residual trust in a single HE secret key, (ii) dropout-robust training via secret sharing or coded computation, and (iii) parallel and hardware-assisted HESD to reduce the server bottleneck.

Acknowledgments. This work has been supported by the PC2phish project, which has received funding from FCT with Ref[a]: 2024.07648.IACDC. Furthermore, this work also received funding from the project UID/00760/2025.

References

1. Abdinasibfar, H., Nuoskala, C., Michalas, A.: The HHE Land: Exploring the Landscape of Hybrid Homomorphic Encryption (2025), https://eprint.iacr.org/2025/071
2. Aledhari, M., Razzak, R., Parizi, R.M., Saeed, F.: Federated Learning: A Survey on Enabling Technologies, Protocols, and Applications. IEEE Access **8**, 140699–140725 (2020). https://doi.org/10.1109/ACCESS.2020.3013541
3. Barker, E.: Recommendation for Key Management Part 1: General. Tech. Rep. NIST SP 800-57pt1r4, National Institute of Standards and Technology (Jan 2016). https://doi.org/10.6028/NIST.SP.800-57pt1r4
4. Beutel, D.J., Topal, T., Mathur, A., Qiu, X., Fernandez-Marques, J., Gao, Y., Sani, L., Li, K.H., Parcollet, T., de Gusmão, P.P.B., Lane, N.D.: Flower: A Friendly Federated Learning Research Framework. arXiv preprint arXiv:2007.14390 (Mar 2022). https://doi.org/10.48550/arXiv.2007.14390
5. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Trans. Comput. Theory **6**(3) (Jul 2014). https://doi.org/10.1145/2633600
6. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017. pp. 409–437. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_15
7. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast fully homomorphic encryption over the torus. J. Cryptology **33**(1), 34–91 (Jan 2020). https://doi.org/10.1007/s00145-019-09319-x
8. Cho, J., Ha, J., Kim, S., Lee, B., Lee, J., Lee, J., Moon, D., Yoon, H.: Transciphering Framework for Approximate Homomorphic Encryption (Full Version) (2020), https://eprint.iacr.org/2020/1335
9. Correia, P., Silva, I., Amorim, I., Maia, E., Praça, I.: Federated learning: An approach with hybrid homomorphic encryption (2025), https://arxiv.org/abs/2509.03427
10. Cosseron, O., Hoffmann, C., Méaux, P., Standaert, F.X.: Towards Globally Optimized Hybrid Homomorphic Encryption - Featuring the Elisabeth Stream Cipher (2022), https://eprint.iacr.org/2022/180
11. Deng, L.: The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine **29**(6), 141–142 (2012). https://doi.org/10.1109/MSP.2012.2211477
12. Dobraunig, C., Grassi, L., Helminger, L., Rechberger, C., Schofnegger, M., Walch, R.: Pasta: A case for hybrid homomorphic encryption. Cryptology ePrint Archive, Paper 2021/731 (2021), https://eprint.iacr.org/2021/731
13. Dowlin, N., Gilad-Bachrach, R., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Manual for using homomorphic encryption for bioinformatics. Proceedings of the IEEE **105**(3), 552–567 (2017). https://doi.org/10.1109/JPROC.2016.2622218
14. Duy, P.T., Quyen, N.H., Khoa, N.H., Tran, T.D., Pham, V.H.: FedChain-Hunter: A reliable and privacy-preserving aggregation for federated threat hunting framework in SDN-based IIoT. Internet of Things **24**, 100966 (Dec 2023). https://doi.org/10.1016/j.iot.2023.100966
15. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144 (2012), https://eprint.iacr.org/2012/144

16. Fontenla-Romero, O.e.a.: FedHEONN: Federated and homomorphically encrypted learning method for one-layer neural networks. Future Generation Computer Systems **149**, 200–211 (Dec 2023). https://doi.org/10.1016/j.future.2023.07.018
17. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing. p. 169–178. STOC '09, Association for Computing Machinery, New York, NY, USA (2009). https://doi.org/10.1145/1536414.1536440
18. Halevi, S., Shoup, V.: Design and implementation of HElib: a homomorphic encryption library. Cryptology ePrint Archive, Paper 2020/1481 (2020), https://eprint.iacr.org/2020/1481
19. Jin, W., Yao, Y., Han, S., Joe-Wong, C., Ravi, S., Avestimehr, S., He, C.: FedML-HE: An efficient homomorphic-encryption-based privacy-preserving federated learning system. In: International Workshop on Federated Learning in the Age of Foundation Models in Conjunction with NeurIPS 2023 (2023), https://openreview.net/forum?id=PuYD0fh5aq
20. Kairouz, P., McMahan, H.B., et al.: Advances and Open Problems in Federated Learning. arXiv preprint arXiv:1912.04977 (2021), https://arxiv.org/abs/1912.04977
21. Li, Q., Yu, W., Xia, Y., Pang, J.: From Centralized to Decentralized Federated Learning: Theoretical Insights, Privacy Preservation, and Robustness Challenges. arXiv preprint arXiv:2503.07505 (Mar 2025). https://doi.org/10.48550/arXiv.2503.07505
22. Li, Y., Gui, J., Wu, Y.: An experimental study of different aggregation schemes in semi-asynchronous federated learning. arXiv preprint arXiv:2405.16086 (2024), https://arxiv.org/abs/2405.16086
23. Ma, J., Naas, S.A., Sigg, S., Lyu, X.: Privacy-preserving federated learning based on multi-key homomorphic encryption. International Journal of Intelligent Systems **37**(9), 5880–5901 (2022). https://doi.org/10.1002/int.22818
24. Majeed, U., Hassan, S.S., Hong, C.S.: Cross-Silo Model-Based Secure Federated Transfer Learning for Flow-Based Traffic Classification. In: 2021 International Conference on Information Networking (ICOIN). pp. 588–593. IEEE, Jeju Island, Korea (South) (Jan 2021). https://doi.org/10.1109/ICOIN50884.2021.9333905
25. Marcolla, C., Sucasas, V., Manzano, M., Bassoli, R., Fitzek, F.H.P., Aaraj, N.: Survey on fully homomorphic encryption, theory, and applications. Proceedings of the IEEE **110**(10), 1572–1609 (2022). https://doi.org/10.1109/JPROC.2022.3205665
26. McMahan, H.B., E, M., D, R., S., H., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data (2023), https://arxiv.org/abs/1602.05629
27. Moshawrab, M., Adda, M., Bouzouane, A., Ibrahim, H., Raad, A.: Reviewing federated learning aggregation algorithms; strategies, contributions, limitations and future perspectives. Electronics **12**(10) (2023). https://doi.org/10.3390/electronics12102287
28. Naehrig, M., Lauter, K., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop. p. 113–124. CCSW '11, Association for Computing Machinery, New York, NY, USA (2011), https://doi.org/10.1145/2046660.2046682
29. Nguyen, K., Khan, T., Abdinasibfar, H., Michalas, A.: A privacy-centric approach: Scalable and secure federated learning enabled by hybrid homomorphic encryption (2025), https://arxiv.org/abs/2507.14853

30. Niu, J., Liu, P., et al.: A survey on membership inference attacks and defenses in machine learning. Journal of Information and Intelligence **2**(5), 404–454 (2024). https://doi.org/10.1016/j.jiixd.2024.02.001
31. Ovi, P.R., Gangopadhyay, A.: A comprehensive study of gradient inversion attacks in federated learning and baseline defense strategies. In: 2023 57th Annual Conference on Information Sciences and Systems (CISS). pp. 1–6 (2023). https://doi.org/10.1109/CISS56502.2023.10089719
32. Qian, J., Wei, K., Wu, Y., Zhang, J., Chen, J., Bao, H.: Gi-smn: Gradient inversion attack against federated learning without prior knowledge. In: Huang, D.S., Chen, W., Pan, Y. (eds.) Advanced Intelligent Computing Technology and Applications. pp. 439–448. Springer Nature Singapore, Singapore (2024). https://doi.org/10.1007/978-981-97-5603-2_36
33. Rabieinejad, E., Yazdinejad, A., Dehghantanha, A., Srivastava, G.: Two-Level Privacy-Preserving Framework: Federated Learning for Attack Detection in the Consumer Internet of Things. IEEE Transactions on Consumer Electronics **70**(1), 4258–4265 (Feb 2024). https://doi.org/10.1109/TCE.2024.3349490
34. Sathya, S.S., Vepakomma, P., Raskar, R., Ramachandra, R., Bhattacharya, S.: A review of homomorphic encryption libraries for secure computation. arXiv preprint arXiv:1812.02428 (2018). https://doi.org/10.48550/arXiv.1812.02428
35. Sharma, G.: Mnist cnn with 8k parameters (2020), https://www.kaggle.com/code/gauravsharma99/mnist-8k-parameters, accessed: 2025-06-11
36. Song, C., Wang, Z., Peng, W., Yang, N.: Secure and Efficient Federated Learning Schemes for Healthcare Systems. Electronics **13**(13), 2620 (Jan 2024). https://doi.org/10.3390/electronics13132620
37. Stan, O., Thouvenot, V., Boudguiga, A., Kapusta, K., Zuber, M., Sirdey, R.: A Secure Federated Learning: analysis of different cryptographic tools. In: di Vimercati, S.D.C., Samarati, P. (eds.) SECRYPT 2022 - 19th International Conference on Security and Cryptography. vol. 1, pp. 669–674. Lisbonne, Portugal (Jul 2022). https://doi.org/10.5220/0011322700003283
38. Tan, Z.S., See-To, E.W., Lee, K.Y., Dai, H.N., Wong, M.L.: Privacy-preserving federated learning for proactive maintenance of IoT-empowered multi-location smart city facilities. Journal of Network and Computer Applications **231**, 103996 (Nov 2024). https://doi.org/10.1016/j.jnca.2024.103996
39. Xu, Y., Mao, Y., Li, J., Chen, X., Wu, S.: Edge server enhanced secure and privacy preserving federated learning. Computer Networks **249**, 110465 (Jul 2024). https://doi.org/10.1016/j.comnet.2024.110465
40. Yu, S., Cui, L.: Inference Attacks and Counterattacks in Federated Learning, pp. 13–36. Springer Nature Singapore, Singapore (2023). https://doi.org/10.1007/978-981-19-8692-5
41. Zhang, L., Xu, J., Vijayakumar, P., Sharma, P.K., Ghosh, U.: Homomorphic Encryption-Based Privacy-Preserving Federated Learning in IoT-Enabled Healthcare System. IEEE Transactions on Network Science and Engineering **10**(5), 2864–2880 (Sep 2023). https://doi.org/10.1109/TNSE.2022.3185327
42. Zhang, M., Chen, S., Shen, J., Susilo, W.: Privacyeafl: Privacy-enhanced aggregation for federated learning in mobile crowdsensing. IEEE Transactions on Information Forensics and Security **18**, 5804–5816 (2023). https://doi.org/10.1109/TIFS.2023.3315526
43. Zhang, Y., Zhang, W., Shen, C.: A fault-tolerant federated learning scheme based on multi-key homomorphic encryption. In: Proceedings of the 2024 International Academic Conference on Edge Computing, Parallel and Distributed Computing. p.

96–101. ECPDC '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3677404.3677421