FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

A Hybrid Homomorphic Encryption approach for Federated Learning

Pedro Miguel da Silva Correia

Master in Informatics and Computing Engineering

Supervisor: Ivone De Fátima Da Cruz Amorim

Second Supervisor: Eva Catarina Gomes Maia

September 30, 2025

A Hybrid Homomorphic Encryption approach for Federated Learning

Pedro Miguel da Silva Correia

Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

President: Prof. João Carlos Viegas Martins Bispo
Referee: Dr. Pierrick Roger Fernand Méaux
Referee: Prof. Ivone De Fátima Da Cruz Amorim

September 30, 2025

Resumo

A Aprendizagem Federada (FL) é uma abordagem de aprendizagem automática distribuída que permite o treino colaborativo de modelos sem necessidade de centralizar dados sensíveis, prometendo privacidade, embora se tenha demonstrado vulnerável a ataques de inferência de associação e de reconstrução de dados. A Criptografia Homomórfica (HE) pode mitigar estes problemas ao permitir a realização de operações sobre dados cifrados. Contudo, o seu elevado custo computacional e a expansão do texto cifrado complicam a implementação em ambientes de recursos limitados. A Criptografia Híbrida Homomórfica (HHE) oferece uma alternativa mais prática ao combinar cifras simétricas leves com HE, reduzindo a sobrecarga nos clientes e mantendo a privacidade.

Esta tese investiga a aplicação da HHE em FL e propõe uma nova plataforma de FL baseada em HHE que combina a cifra simétrica PASTA com o esquema HE BFV. Os clientes cifram as atualizações locais do modelo com PASTA e enviam-nas juntamente com a chave PASTA cifrada com a cifra BFV para o servidor. O servidor realiza então uma avaliação homomórfica do circuito de decifrar do PASTA e agrega os dados cifrados resultantes. Para mitigar os riscos de confidencialidade decorrentes do uso de uma chave HE partilhada, foram desenvolvidas duas estratégias: encapsulamento RSA, que cifra a chave PASTA (já cifrada com BFV) com a chave pública RSA do servidor, e o uso de máscaras, em que os clientes enviam uma chave com uma máscara que o servidor posteriormente retira durante a agregação.

O sistema foi integrado na plataforma Flower e avaliado com uma versão independente e identicamente distribuída do conjunto de dados MNIST, distribuída por 12 clientes ao longo de 10 rondas de treino. Os resultados demonstram que os esquemas HHE alcançaram um nível de precisão comparável ao treino em dados não cifrados (97,58 % a 98,33% face a 98,93%), reduziram o tamanho do modelo cifrado por um fator de 2.077×, o tráfego de envio dos clientes até 61,73× e o tempo de execução do cliente em até 30%, em comparação com um sistema baseado exclusivamente no esquema HE BFV. No entanto, o custo computacional do servidor aumenta aproximadamente 6.700× por cada cliente participante na fase de treino.

No geral, o estudo demonstra que a HHE proporciona um equilíbrio eficaz entre privacidade, eficiência e precisão em FL para dispositivos com recursos limitados. Embora a escalabilidade do lado do servidor continue a ser um desafio, a abordagem proposta constitui uma das primeiras demonstrações práticas de HHE aplicada a FL.

Keywords: Aprendizagem Federada, Agregação Segura de Modelos, Criptografia Homomórfica Híbrida, Aprendizam Automática com Preservação da Privacidade, Transciphering, BFV, PASTA, Internet das Coisas

Abstract

Federated Learning (FL) is a distributed machine learning approach that enables collaborative model training without centralizing raw data, promising privacy but having been shown to be vulnerable to membership inference and reconstruction attacks. Homomorphic Encryption (HE) can address those privacy concerns by allowing computation on encrypted updates. However, its high computational cost and ciphertext expansion hinder deployment in resource-constrained environments. Hybrid Homomorphic Encryption (HHE) offers a more practical alternative by combining lightweight symmetric ciphers with HE, reducing client overhead while maintaining end-to-end privacy.

This thesis investigates the application of HHE in FL and proposes a novel HHE-based FL framework that combines the PASTA symmetric cipher with the BFV HE scheme. Clients encrypt local model updates with PASTA and send both the lightweight ciphertext and the BFV encryption of the PASTA key to the server. The server then performs a homomorphic evaluation of the decryption circuit of PASTA and aggregates the resulting BFV ciphertexts. To mitigate confidentiality risks stemming from the use of a shared HE key, two mitigation strategies were developed: RSA wrapping, which re-encrypts the BFV-encrypted PASTA key under the server's RSA public key, and Masking, where clients encrypt a masked key that the server later unmasks during aggregation.

The system was integrated into the Flower framework and evaluated under an independent and identically distributed partitioned version of the MNIST dataset with 12 clients across 10 training rounds. The results demonstrate that the HHE schemes achieved accuracy comparable to plaintext training (97.58%–98.33% vs. 98.93%), while reducing the encrypted model size by a factor of 2,077×, client upload traffic by up to 61.73×, and cutting client runtime by at most 30% compared to a system based solely on the BFV HE scheme. However, server computational cost increases by roughly 6700× for each client participating in the training phase.

Overall, the study shows that HHE provides an effective balance between privacy, efficiency, and accuracy in FL for resource-constrained devices. While server-side scalability remains a challenge, the proposed approach constitutes one of the first practical demonstration of HHE for FL.

Keywords: Federated Learning, Secure Model Aggregation, Hybrid Homomorphic Encryption, Privacy-Preserving Machine Learning, Transciphering, BFV, PASTA, Internet of Things

UN Sustainable Development Goals

The United Nations Sustainable Development Goals(SDGs) provide a global framework to achieve a better and more sustainable future for all. It includes 17 goals to address the world's most pressing challenges, including poverty, inequality, climate change, environmental degradation, peace, and justice.

This work contributes to global sustainability efforts by advancing secure and efficient federated learning. The proposed hybrid homomorphic encryption framework reduces computational and communication overhead on the client side, making privacy-preserving machine learning more practical. By lowering resource requirements, the approach enables broader participation from organizations and devices with limited capabilities, fostering inclusivity in the development of data-driven technologies.

The specific Sustainable Development Goals mentioned have the following names:

SDG 8 Promote sustained, inclusive and sustainable economic growth, full and productive employment and decent work for all

SDG 9 Build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation

SDG	**Target**	**Contribution**	**Performance Indicators and Metrics**
8	8.2	Reduces computational and communication overhead for clients, enabling organizations in resource-constrained environments to participate in federated learning, fostering inclusive economic growth.	Reduction in client upload traffic (up to 61.73×); Reduction in encrypted model size (2,077× smaller); Reduction in client runtime (up to 30%)
9	9.5	Proposes one of the first practical implementations of Hybrid Homomorphic Encryption in Federated Learning, advancing privacy-preserving machine learning.	Novelty of combining PASTA with BFV for FL; Integration with Flower for practical FL experimentation; Open-source release of implementation and benchmarking framework

Acknowledgement

I would like to express my gratitute to all the people that helped me complete this work, either directly or indirectly involved.

I would like to express my sincere thanks to my supervisors, Dr. Ivone Amorim and Dr. Eva Maia, for their invaluable guidance and insights, which kept this thesis on the right track, and for their support and availability that helped me overcome challenges during its development. I also apologize for any difficulties caused by my writing.

To Ivan Silva, I am truly grateful for your help, not only in understanding HHE but also with any other topics I struggled with during this work. I am also thankful for your encouragement during the more challenging phases of the development.

To Dr. Isabel Praça, for placing your trust in me and allowing me to take part in activities I never thought I would have the chance to experience.

To my friends, for being there during both the good and bad moments, and for always checking in to see if I was keeping up with my work

Lastly, and most importantly, to my family, especially my parents, for always believing in me and allowing me to pursue an academic life.

Pedro Correia

Contents

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
ML	Machine Learning
DML	Distributed Machine Learning
IoT	Internet of Things
FL	Federated Learning
DP	Differential Privacy
SMC	Secure Multi-Party Computation
HE	Homomorphic Encryption
HHE	Hybrid Homomorphic Encryption
TEE	Trusted Execution Environment
IID	Independent and Identically Distributed
FedAvg	Federated Averaging
FedSGD	Federated Stochastic Gradient Descent
HFL	Horizontal Federated Learning
VFL	Vertical Federated Learning
FTL	Federated Transfer Learning
PHE	Partially Homomorphic Encryption
SWHE	Somewhat Homomorphic Encryption
FHE	Fully Homomorphic Encryption
MKHE	Multi-key Homomorphic Encryption
HESD	Homomorphic Evaluation of Symmetric Decryption
CNN	Convolutional Neural Network
HE_pk	Homomorphic Encryption Public Key
HE_sk	Homomorphic Encryption Secret Key
HE_evk	Homomorphic Encryption Evaluation Key
sk	Symmetric Encryption Secret Key
RSA_pk	RSA Public Key
RSA_sk	RSA Secret Key
m	Plaintext Message
c_{HE}	Homomorphic Ciphertext
C_{HE}	Set of homomorphic ciphertexts
c_{SKE}	Symmetric ciphertext
sk_{HE}	Homomorphically encrypted symmetric encryption secret key
M	Random Mask used in the Masking scheme
TPA	Third-Party Authority
N	Total number of clients in the system
i	Index of a client, $i = 1, \ldots, N$
sk_i	Symmetric encryption secret key of client i
sk^i_{HE}	Homomorphically encrypted symmetric secret key of client i
M_i	Random Mask of client i
n_i	Number of Training Samples of client i
w_i	Local Model Weights of client i
w_{SKE}	Symmetrically encrypted local model weights
w^i_{SKE}	Symmetrically encrypted local model weights of client i

Chapter 1

Introduction

1.1 Context and Motivation

Artificial Intelligence (AI) and Machine Learning (ML) are transforming a wide range of fields, from the food industry and e-commerce to chemical manufacturing [1], and even healthcare, where it has the potential to enhance quality of life [2]. Their success, however, relies on access to large and diverse datasets, as well as the ability to train increasingly complex models [3, 4, 5].

These requirements pose two major challenges. On the one hand, many devices, particularly those in the Internet of Things (IoT), wearable sensors, and remote monitors, suffer from limited computational and energy resources, making it difficult to train or deploy large-scale models locally [6]. However, even when computational power is available, access to sensitive data is often restricted due to privacy concerns [7]. Such restrictions prevent researchers from using valuable data for their studies and obstruct healthcare professionals from offering more personalised analyses and treatment plans [8]. Creating an effective AI model requires access to large amounts of data, and if most of the data needed is private and not available for use, the resulting model may not perform well when faced with new situations [9].

To address the computational burden of training large models on single machines or constrained devices, the Distributed Machine Learning (DML) paradigm emerged [10]. DML uses distributed computing to spread the training process across multiple machines, or edge devices, allowing the development of models previously thought impossible in centralised settings [10].

However, while DML addresses computational scalability, it does not alone solve the equally critical issue of data privacy, since sensitive data often cannot be centralised, even in distributed setups [11]. To overcome this limitation, Federated Learning (FL) has been introduced as a specialised form of DML that allows a group of clients, such as hospitals or mobile devices, to collaboratively train a global AI model without directly sharing their raw data. In typical FL setups, each client trains a local model using its own data and shares only the model parameters with a central server that coordinates the training. The central server then updates the global model by aggregating the local model parameters using techniques such as weighted averaging. Once updated, the new global parameters are sent back to the clients, allowing them to continue training locally. This

approach aims to preserve the privacy of the data, as raw data is not exchanged directly between clients or with the server [12].

However, recent research has shown that local model updates can unintentionally leak information from local data, potentially exposing participants to various privacy attacks, including data reconstruction, membership inference, and property inference attacks [13, 14]. To solve those issues, several techniques have been integrated into FL systems, such as Differential Privacy (DP), Homomorphic Encryption (HE), and Secure Multi-Party Computation (SMC) [15]. DP adds noise to data, making inference harder for attackers, but possibly reduce model accuracy [16]. SMC protocols allow multiple clients to collaboratively perform aggregation without trusting the central server, but require high computational and communication overhead [17]. HE allows servers to perform computations directly on encrypted data, generating an encrypted result that, when decrypted, matches the result of the same operations on the corresponding unencrypted data, allowing for a secure aggregation process [18, 19]. Unfortunately, due to the high computational cost of the homomorphic operations and ciphertext expansion [20], a HE-based FL scheme is much less efficient than its plaintext counterpart [19].

These limitations make it challenging to implement secure FL schemes in resource-constrained environments. Hybrid Homomorphic Encryption (HHE) offers a balanced solution tailored to such constraints [21]. Instead of relying solely on HE for the entire model, which is often computationally expensive and slow, HHE combines multiple encryption methods to improve efficiency. As demonstrated by Dobraunig et al. [22], with HHE, client can reduce their computational requirements by encrypting data with a fast symmetric cipher. The encrypted data, along with the symmetric key encrypted using homomorphic encryption, is sent to the server. The server homomorphically evaluates the symmetric decryption function on the symmetric ciphertext, transforming it into a homomorphic ciphertext. This enables computations to be carried out directly on encrypted data while ensuring the raw data remains protected. Although this approach increases the computational load on the server, it reduces communication overhead, making HHE a promising solution for privacy-preserving FL in IoT environments with limited computational resources.

1.2 Problem

HE is a widely studied privacy-preserving technique in FL [15], and its integration into IoT devices has naturally followed. However, practical limitations have emerged, primarily due to the high computational and communication costs associated with HE [23].

With the development of HHE schemes, the costs associated with HE on the client side start to decrease drastically [22], making HHE a promising alternative for resource-constrained FL scenarios. However, to the best of our knowledge, the application of HHE in FL has not been explored.

This research aims to investigate the use of HHE to enhance privacy and efficiency in FL environments involving computationally limited devices. The primary research question guiding

this study is: "How can HHE be applied to FL to enhance privacy and efficiency in IoT devices?". To address this question, narrower-sub questions were established:

RQ1 What are the current methods used to protect data privacy in FL?

RQ2 How do different homomorphic encryption approaches operate in FL for IoT devices, and how effective are they in enhancing privacy and security?

RQ3 How does an HHE approach compare to existing privacy-preserving techniques in FL for IoT settings?

1.3 Objectives

Considering the problem stated, the main objective of this research is to develop and evaluate a HHE approach for FL in IoT environments, with the goal of enhancing both privacy and efficiency for resource-constrained devices.

To achieve this, the study pursues the following specific objectives

- **O1 - Review and analyze existing privacy-preserving methods in FL** to establish a foundation for comparison.
- **O2 - Investigate how different HE approaches are applied in FL for IoT devices**, focusing on their effectiveness in ensuring privacy and security.
- **O3** - **Design, implement, and evaluate an HHE-based FL scheme**, comparing its performance and privacy guarantees with existing techniques.

1.4 Contributions

This thesis makes several key contributions to the field of privacy-preserving FL.

First, it provides a comprehensive survey of the most widely used privacy-preserving techniques in FL. This review highlights the strengths and limitations of current approaches, offering a clear understanding of the existing landscape and identifying areas that require further research.

Second, the thesis presents a detailed analysis of the use of HE in FL, with particular focus on its application to IoT environments. This analysis covers multiple dimensions, including system architectures, the entities involved, assumed threat models, aggregation methods, evaluation datasets, and the HE schemes employed. It also examines key management practices, the specific data encrypted, and the entities responsible for decryption. Furthermore, it reviews additional techniques designed to strengthen security, such as defenses against poisoning and collusion attacks, as well as mechanisms for enhancing robustness and communication efficiency, including strategies for managing user dropout, filtering irrelevant updates, and compressing data.

Third, the thesis introduces a novel HHE-based FL scheme. To balance efficiency and strong security guarantees, the framework employs the PASTA [22] stream cipher for lightweight and

fast encryption on the client side, making it suitable for resource-constrained IoT devices. On the server side, the BFV [24] HE scheme is integrated to enable secure computation and aggregation directly over encrypted data. This combination ensures end-to-end privacy while maintaining practical feasibility in real-world deployments.

Fourth, the proposed HHE scheme is designed to be independent of any specific FL framework, but was implemented and evaluated in this thesis using the Flower [25] framework. The implementation includes a homomorphic evaluation routine for Federated Averaging (FedAvg), operating under a shared HE key setup since multi-key HE was not feasible in the selected framework.

Fifth, because all clients share the same HE key pair, a client could potentially intercept and decrypt data transmitted by others. To mitigate this risk, two strategies were designed and implemented: RSA wrapping and Masking, ensuring that client data remains private during transmission even when multiple clients share the same HE key pair.

Sixth, the thesis reports on a comprehensive experimental evaluation. The scheme was tested on an Independent and Identically Distributed (IID) partitioned version of the MNIST [26] dataset using 12 clients over 10 training rounds. The results demonstrated that the HHE approaches achieved accuracy comparable to plaintext training (97.58%–98.33% vs. 98.93%), while reducing the encrypted weights size by a factor of 2,077× and total client upload traffic by up to 61.73×. However, server computational overhead increased by approximately 6,700× per client participating in the training phase.

Finally, the software developed for this thesis is made publicly available. The Python library developed is available on GitHub[1] , while the code used to conduct the experiments presented is accessible in a Google Drive folder[2].

The work done for this thesis resulted in a scientific publication:

- **Pedro Correia, Ivan Silva, Ivone Amorim, Eva Maia, and Isabel Praça** *Federated learning: An approach with hybrid homomorphic encryption*. 6th International Workshop on Cyber-Physical Security for Critical Infrastructures Protection (CPS4CIP 2025), an workshop of the 30th European Symposium on Research in Computer Security (ESORICS 2025), September 2025 [27]. Accepted for presentation and publication.

1.5 Thesis Structure

This document is organized as follows: Chapter 1, *Introduction*, provides the context and motivation behind this work, introducing the research questions, the main objectives, and the contributions of this thesis. Chapter 2, *Preliminaries*, expands on concepts introduced earlier, such as FL, HE, and HHE, to support understanding of the techniques discussed in later chapters. Chapter 3, *Literature Review*, presents existing studies that address the aforementioned research questions.

[1] https://github.com/PedroCorreia56/hhe_fl

[2] https://drive.google.com/drive/folders/19miLI9eniVmRq4e0ME6NShVxHDH5l2AI?usp=sharing

Chapter 4, *Proposed Solution*, describes the architecture and the workflow of the proposed solution, ending with the threat model considered for this work. Chapter 5, *Implementation Details*, presents the cryptographic tools and FL framework used in this work, and the resulting prototype architecture. It also describes the algorithms implemented on both the client and server sides and the parameters constraints forced on this work. Chapter 6, *Experimental Results*, presents the results of the experimental evaluation, analyzing the performance and effectiveness of the proposed approach. Finally, Chapter 7, *Conclusion and Future Work*, concludes the thesis by summarizing key findings, discussing encountered limitations, and outlining directions for future work.

Chapter 2

Preliminaries

This chapter presents the foundational concepts and key technologies relevant to this work. It starts with an overview of FL, a modern approach to addressing privacy concerns in distributed AI model training. Key challenges in FL, such as heterogeneity of data and system characteristics, as well as privacy and security threats, are discussed. Following this, HE is introduced as a cryptographic technique that allows computations on encrypted data, enhancing privacy in FL. Finally, HHE is covered as a recent advancement that combines different encryption methods to improve efficiency and practicality in privacy-preserving FL systems.

2.1 Federated Learning

FL is a DML paradigm that enables multiple devices or organizations to collaboratively train a shared global model without sharing each participant's local data, and it can be implemented in either a *centralized* or *decentralized* manner [28].

In *centralized* FL, a central server manages multiple clients, orchestrating the training process as depicted in Figure 2.1. The training occurs in rounds: during each round, clients train their local models on local data and send only the local model parameters to the server. The server then updates the global model by aggregating the local model parameters. Once updated, the new global parameters are sent back to the clients to use in the following round. This process continues for a fixed number of rounds or until the model converges [29]. This approach preserves privacy, as no sensitive local data is exchanged between clients or with the server [12]. However, because the server is solely responsible for coordinating training and aggregating updates, its failure would halt the entire process, making it a single point of failure [28].

In contrast, *decentralized* FL does not rely on a central server. Instead, the clients communicate and exchange model updates directly with each other, typically in a peer-to-peer fashion, as shown in Figure 2.2. This approach removes the single point of failure brought by the server but introduces other challenges, such as slower convergence when compared to centralized FL [28].

Within these architectures, aggregation algorithms are used to combine the local model updates into a global model. The choice of aggregation method can impact both the accuracy of

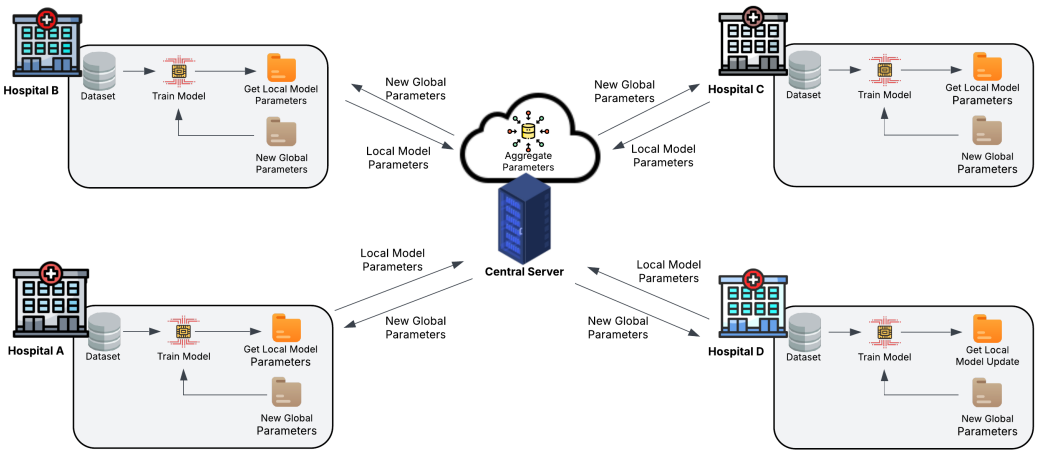

Figure 2.1: Centralized Federated Learning Architecture

the resulting global model and the computational overhead [30]. Two of the most widely used algorithms are *Federated Stochastic Gradient Descent* (FedSGD) and *Federated Averaging* (FedAvg) [31].

FedSGD extends the classic stochastic gradient descent algorithm [32] to the federated setting. In this algorithm, the aggregator aggregates local gradients computed by the clients. In each training round, a fraction C of clients is selected. Each participating client computes the gradient of its local objective function on its dataset using the current global model, and sends this gradient to the server [33].

Let K be the total number of clients, and $S_t \subseteq \{1,\dots,K\}$ be the set of clients selected at round t, with $|S_t| = C \cdot K$ being the number of participating clients. For each client $k \in S_t$, let n_k denote the number of local training samples of client k, and $n_{S_t} = \sum_{k \in S_t} n_k$ be the total number of training samples of all clients in S_t. Denote by g_t^k the gradient computed by client k at round t by $g_t^k = \nabla F_k(w_t)$ where F_k is the local objective function of client k and w_t being the current global model of round t. The global model is then updated using a weighted average of the clients' gradients:

$$w_{t+1} = w_t - \eta \sum_{k \in S_t} \frac{n_k}{n_{S_t}} g_t^k,$$

where η is the learning rate. When $C = 1$ the procedure reduces to full-batch gradient descent, which is non-stochastic because the update is based on the complete set of training data rather than a random subset. FedSGD is a computationally efficient algorithm but may require a high number of rounds to produce good results [33].

FedAvg extends FedSGD by allowing clients to perform multiple local optimization steps before the global aggregation. In each round, each client $k \in S_t$ performs several local updates on

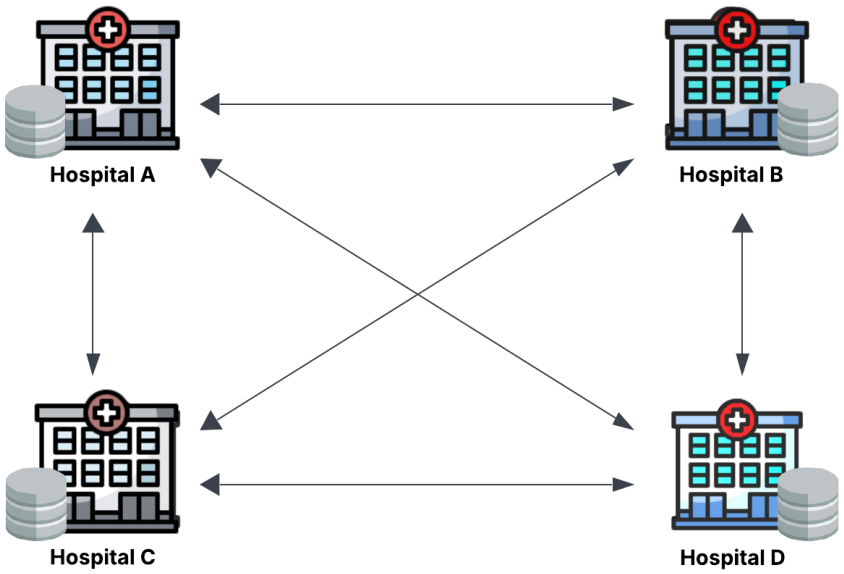

Figure 2.2: Decentralized peer-to-peer FL

its local model w^k using its own dataset. The local updates are computed iteratively as:

$$w^k \leftarrow w^k - \eta \nabla F_k(w^k)$$

This procedure can be repeated for multiple iterations or epochs over the local dataset. After completing the local updates, the clients send their updated models w^k_{t+1} to the server. The global model is then updated using a weighted average of the clients' models:

$$w_{t+1} = \sum_{k \in S_t} \frac{n_k}{n_{S_t}} w^k_{t+1},$$

FedAvg, by allowing each client to perform more computations locally, reduces the number of communication rounds needed to achieve higher accuracy, thereby lowering the overall communication cost compared to FedSGD [33].

While these algorithms improve aggregation efficiency, FL continues to face significant challenges, most notably heterogeneity across clients and persistent privacy and security threats, which complicate both its design and deployment.

2.1.1 Heterogeneity

Unlike traditional centralized training environments where data, models, and computational resources are controlled and standardized, FL involves diverse clients with different characteristics [34]. This heterogeneity typically arises from three main factors: data distribution, model configurations and client-side system resources [35].

2.1.1.1 Data Distribution Heterogeneity

In traditional ML, data is typically assumed to be Independently and Identically Distributed (IID), meaning that all clients possess datasets with similar features and sample distributions [34]. In FL scenarios, however, the diversity of clients often leads to Non-IID data [36], where different clients may have datasets that vary both in feature composition and in the number of samples. For example, Google [37] used FL for next-word prediction in Gboard, a Google keyboard for mobile devices. Different languages and unique writing patterns of users lead to variations in both features and sample distributions across clients. This data can negatively affect the FL model by causing biased outcomes, slower convergence, lower accuracy, and increased communication overhead due to more rounds being needed for training. Additionally, this variability complicates privacy guarantees, as certain data patterns may unintentionally expose sensitive client information [38]. To handle this, FL is commonly categorized into three main settings based on the overlap in feature and sample spaces between clients.

Horizontal Federated Learning (HFL), also known as sample-based FL, applies when client datasets share the same feature space but consist of different samples [12], as shown in Figure 2.3 (a). For instance, in healthcare, hospitals may store patient records containing similar attributes, such as age, gender, height, and weight, but for entirely different patients.

Vertical Federated Learning (VFL), or feature-based FL, is applied when clients have datasets with the same samples but distinctive features [39], as seen in Figure 2.3 (b). For example, in finance, one client might hold a customer's transaction history while another has their credit scores and demographic information.

Federated Transfer Learning (FTL) is used in scenarios where datasets differ in both sample and feature spaces [40], as illustrated in Figure 2.3 (c). In such cases, traditional horizontal or vertical federated learning approaches are insufficient due to the lack of alignment between clients' data. To solve this issue, FTL leverages transfer learning techniques, allowing the model to learn across disparate datasets, even when the intersection in the sample space is minimal [39].

2.1.1.2 Model and System Heterogeneity

Real-world applications of FL should not assume that all clients have the same computational capabilities or network conditions. For instance, in FL scenarios involving mobile devices, computational power can vary from high-end smartphones with multiple cores and GPUs to older devices with limited processing capabilities, memory can range from several gigabytes to just a few hundred megabytes, and connectivity can fluctuate from stable 5G connections to irregular Wi-Fi access, leading to system heterogeneity [35, 41]. Limited computational resources or unstable connections may prevent some clients from completing training at the same pace as others, or even cause them to drop out during a training round. Such constraints can also restrict the complexity of models that clients can run. For example, some clients may only be able to train a neural

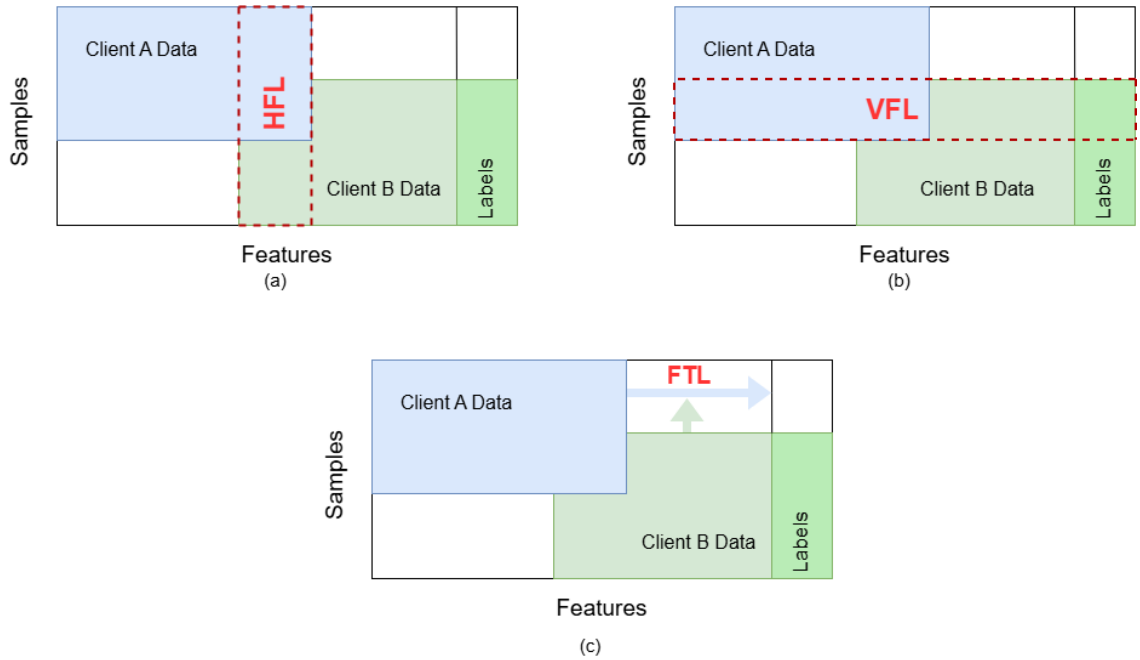

Figure 2.3: FL Settings: (a) Horizontal Federated Learning, (b) Vertical Federated Learning, (c) Federated Transfer Learning

network with three layers, while others can train networks with six layers, introducing model heterogeneity. This complicates the aggregation process, as combining updates from heterogeneous models requires specialized techniques to be effective [35].

These issues have led to the emergence of two main FL deployment schemes. *Cross-Silo FL* involves a relatively small number of reliable clients (typically 2–100), such as organizations or institutions. These clients are generally stable and consistently available, allowing them to participate in each round of computation and carry over information from one round to the next [42]. Cross-Silo FL is compatible with HFL, VFL, and FTL approaches, depending on the alignment of data across clients and the specific application [42, 43]. The main challenges of this approach arise from both performance and collaboration aspects. From a performance perspective, it is necessary to obtain global and local models that remain accurate even when clients have different degrees of heterogeneity, while also ensuring that these models can be obtained with minimal computational and communication costs. From a collaboration perspective, since the participants are typically institutions with long-term goals, maintaining cooperation is essential. To achieve this, mechanisms that provide incentives and reward consistent contributions are often required [44].

In contrast, *Cross-Device FL* involves a large number of clients, often ranging from thousands to millions, such as mobile or IoT devices. These clients are highly unreliable, with frequent dropouts due to issues like battery depletion or unstable network connections. The number of participating clients also varies significantly at any given time. As a result, each client generally participates only once in a task and does not retain information across rounds. The primary challenge in this setup is communication, which must operate under highly unstable conditions [42].

2.1.2 Privacy and Security Threats

FL enables collaboration with different clients while ensuring that data remains stored on the individual devices, rather than being in a centralized environment. This setup offers advantages for data privacy, as sensitive information never leaves the client side. Nonetheless, privacy concerns remain, as both clients and the server may assume different roles: honest, honest-but-curious, or malicious [19].

An *honest entity* adheres to the FL protocol and perform computations as intended [39, 40]. *Honest-but-curious entities* also follow the protocol without interfering with the training process, but may attempt to infer sensitive information from received data [45]. *Malicious entities* actively deviate from the protocol, aiming both to compromise model performance and to extract private information from honest participants [13].

Depending on participant behavior, FL is susceptible to two primary categories of attacks. Poisoning attacks are typically carried out by malicious clients to degrade the global model's integrity [36], while inference attacks are often conducted by honest-but-curious or malicious entities seeking to extract sensitive information from data or model updates [13, 14].

2.1.2.1 Poisoning Attacks

These attacks aim to compromise the integrity of the global model and can vary in severity, from reducing the accuracy of the FL model, to manipulating the model into producing specific target outputs, which is generally more challenging to achieve. These attacks are typically categorized into two types: data poisoning attacks and model poisoning attacks [13].

Data poisoning attacks occur when an attacker manipulates the training data to influence the model's behavior without directly altering the local model parameters. These attacks fall mainly into two categories: *clean-label* and *dirty-label* attacks [13, 46]. Clean-label attacks assume that the adversary cannot alter the labels of training samples due to a verification process that ensures label integrity. Instead, the adversary alters the training input, for example, by injecting noise, to influence the model's behavior without changing the class labels. There are also techniques where adversaries modify training samples to inject a backdoor into the global model [13, 46]. Dirty-label attacks, in contrast, allow the adversary to inject training samples with incorrect labels into the dataset. A common example is label flipping, where labels of one class are replaced with those of another. For instance, in a dataset of animal images, an attacker might relabel "bird" samples as "dog," causing the model to predict "dog" whenever it sees an image of a "bird" [13, 46]. An adversary can also embed backdoors into the model by modifying specific regions of the training data, causing the model to behave as the attacker intends when inputs contain certain backdoor features [13].

Model poisoning attacks involve manipulating a client's local model parameters before they are sent to the server, aiming to degrade the global model's performance, prevent convergence, or embed hidden backdoors. These attacks can range from injecting random noise into local

updates to crafting carefully designed malicious models that replace or distort the global model update [13, 46].

2.1.2.2 Inference Attacks

As mentioned earlier, recent studies have shown that local model updates can still leak private information. Inference attacks aim to extract sensitive data from the shared local model updates or from the global model itself [13].

In a membership inference attack, the adversary aims to determine whether a specific data sample was included in the training set. By analyzing the model's responses to queries, the attacker can infer whether an individual's data was part of the training process. These attacks can be passive or active. In passive attacks, the adversary observes the model's responses without altering the training process. In contrast, active attacks involve manipulating the training protocol, such as injecting malicious updates [14].

In a class representative attack, the adversary aims to infer the prototypical samples of a target label used in training, which the adversary does not have access to. Prototypical samples are considered representative examples that capture the typical features of a class. This attack assumes that all class members are similar and that the adversary has knowledge of the victim's data labels [14].

Property inference attacks aim to infer meta-properties of other participants' training data that are not directly related to the primary learning task [47]. The adversary typically requires auxiliary data correctly labeled with the target property. In passive attacks, the adversary merely observes model updates and trains a binary classifier to distinguish between data with and without the property. In active attacks, the adversary manipulates the FL model toward better separating data with and without the training property. The effectiveness of this attack is heavily dependent on the availability of auxiliary labeled data, which limits its applicability in cross-device scenarios, where such data may not be easily accessible [13, 14].

Training sample and label inference attacks, also known as *data reconstruction attacks* [14], attempt to reconstruct participants' original training inputs and their associated labels by analyzing gradients shared during the FL process [13, 14]. These attacks typically require that the adversaries are selected for multiple rounds of training and possess substantial computational resources, making them more feasible in cross-silo settings rather than in cross-device scenarios [13].

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 [48]. Formally, an encryption function Enc is said to be homomorphic over a set of plaintexts M and an operation $\odot$ if it satisfies:

$$\forall m_1, m_2 \in M, \quad Enc(m_1 \odot m_2) = Enc(m_1) \odot Enc(m_2) \tag{2.1}$$

To realize this property in practice, modern HE schemes are typically structured as a tuple of four Probabilistic Polynomial-Pime (PPT) algorithms, HE = (KeyGen, Enc, Dec, Eval), such that:

- HE.KeyGen (1^{λ}): Given a security parameter λ, generate a public key *HE_pk*, a secret key *HE_sk* and an evaluation key *HE_evk*.
- HE.Enc(m, *HE_pk*): Encrypts the message m with the public key *HE_pk*, returning a ciphertext c_{HE}.
- HE.Eval(C_{HE}, f, *HE_evk*): Applies the function f over a set of ciphertexts $C_{HE} = \{c_{HE_1}, \ldots, c_{HE_n}\}$ using the evaluation key *HE_evk*, producing a new set of ciphertexts C'_{HE}. This set C'_{HE} is the encrypted result of $f(m_1, \ldots, m_n)$, the application of function f to the underlying plaintexts $\{m_1, \ldots, m_n\}$, thereby enforcing the homomorphic property defined in Equation 2.1.
- HE.Dec(c_{HE}, *HE_sk*): Decrypts the ciphertext c_{HE} with the secret key *HE_sk*, returning a plaintext m.

The type and number of operations $\odot$ that a HE scheme can support determine its classification. Namely, a *Partially Homomorphic Encryption* (PHE) scheme allows only a single type of arithmetic $\odot$ (e.g., addition or multiplication) but permits it to be applied an unlimited number of times. *Somewhat Homomorphic Encryption* (SWHE) supports a limited set of operations $\odot$, with restrictions on how many times they can be applied. *Fully Homomorphic Encryption* (FHE) enables multiple types of arithmetic operations $\odot$ to be performed an unlimited number of times.

This classification has led to the development of several HE schemes have been developed to support different types and levels of homomorphic operations. One of the earliest public-key cryptosystems with homomorphic properties is RSA [49], which supports multiplicative homomorphism. Although RSA was not originally designed as a HE scheme, it demonstrated that meaningful computations could be carried out on ciphertexts. Building on this idea, other PHE schemes were developed, most notably ElGamal [50] and Paillier [51]. ElGamal was derived from the Diffie-Hellman Key Exchange [52] algorithm, enabling multiplicative homomorphism, while Paillier is based on the composite residuosity problem and supports additive homomorphism over integers.

To overcome the functional limitations of PHE and enable more expressive computations, SWHE schemes were developed. One of the earliest SWHE schemes is the Polly Cracker [53] scheme, which relies on polynomial ideal theory to support both addition and multiplication over integers. However, ciphertext size grows rapidly with each operation, and multiplication is particularly costly, making the scheme impractical for real-world use [54]. A more efficient step towards FHE was the Boneh–Goh–Nissim (BGN) scheme [55], based on the subgroup decision problem [56]. BGN allows an arbitrary number of additions and a single multiplication on ciphertexts while keeping the ciphertext size constant [54].

A major breakthrough came with Gentry et al. [57], which presented the first construction of a FHE scheme using ideal lattices. This work also introduced the concept of bootstrapping, a

technique that homomorphically evaluates the decryption circuit to reduce noise and thus enable unlimited computations on ciphertexts [57]. While Gentry's original scheme was largely impractical due to high computational costs, it inspired subsequent schemes such as BGV [58] and FV [24], the latter now commonly referred to as BFV due to its combination of techniques from BGV. Both schemes rely on the Ring Learning with Errors (RLWE) problem and allow for computations over integers modulo q, supporting modular arithmetic over finite fields and enabling faster evaluation without the need for immediate bootstrapping.

In contrast, the CKKS scheme [59] supports approximate homomorphic computations over real or complex numbers, enabling efficient encrypted vector operations through techniques such as batching. Unlike other schemes that guarantee exact decryption, CKKS provides approximate results, making it well-suited for applications where exact arithmetic is not critical [60].

Another prominent FHE scheme is TFHE (Fast Fully Homomorphic Encryption over the Torus) [61], which operates at the bit level and is particularly well-suited for boolean operations. It is distinguished by its support for fast bootstrapping, enabling efficient and precise computation over encrypted data [60].

Table 2.1 summarizes the main characteristics of the homomorphic encryption schemes discussed above.

Scheme	HE Type	Data Type	Supported Operations	Decryption	Bootstrapping
ElGamal [50]	PHE	Integer	Multiplication	Exact	Not required
Paillier [51]	PHE	Integer	Addition	Exact	Not required
Polly Cracker [53]	SWHE	Integer	Add + Mult	Exact	Not required
BGN [55]	SWHE	Integer	Add + Mult	Exact	Not required
BGV/BFV [58, 24]	FHE	Integer	Add + Mult	Exact	Optional (expensive)
CKKS [59]	FHE	Floating point	Add + Mult	Approximate	Required (expensive)
TFHE [61]	FHE	Bits	Bitwise	Exact	Required (fast)

Table 2.1: Comparison of Some Homomorphic Encryption Schemes

A simple implementation of HE in FL involves clients encrypting their local model parameters before sending them to the server. The server then aggregates all the homomorphic ciphertexts and performs an aggregation compatible with the chosen HE scheme. After homomorphically computing the new global model, the server sends it back to the clients, who decrypt it before updating their local models. This process is illustrated in Figure 2.4.

In this architecture, all clients use the same homomorphic key pair, as the HE schemes discussed earlier were originally designed for single-key operations, meaning all ciphertexts are encrypted under the same public key. While this simplifies evaluation and key management, it poses limitations in multi-party settings such as FL. For instance, if a malicious client intercepts messages from an honest client, it could decrypt their ciphertexts if both use the same keys. This highlights the need for schemes that support computations over ciphertexts encrypted under different homomorphic keys.

Multi-key Homomorphic Encryption (MKHE) extends traditional homomorphic encryption by enabling computations on ciphertexts encrypted under different public keys. Classical MKHE

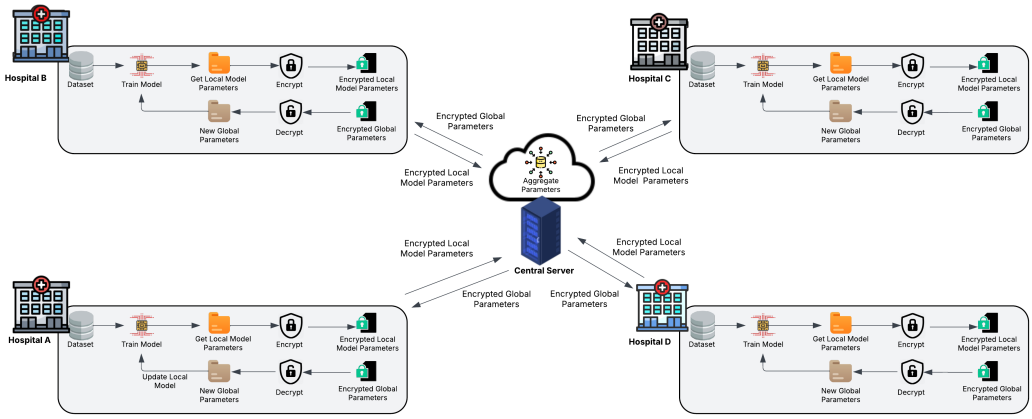

Figure 2.4: Federated Learning Architecture with Homomorphic Encryption

schemes produce a ciphertext that can only be decrypted through joint cooperation among all parties holding the secret keys [62]. More recent work in FL has explored simplified variants, where each client can decrypt individually rather than relying on collective decryption [63]. These two approaches entail different trade-offs: joint decryption introduces coordination and communication overhead [62], whereas individual decryption avoids this requirement but demands aggregation and decryption mechanisms capable of handling ciphertexts encrypted under multiple public keys.

Figure 2.5 illustrates possible HE key pair configurations in FL scenarios. In the base case (a), all the clients share the same homomorphic key pair, simplifying computations but posing security risks in adversarial environments. In (b), each client uses a distinct key pair, ensuring stronger privacy but requiring the server and the clients to be able to handle ciphertexts encrypted with different public keys. Setting (c) shows an approach where clients contain individual key pairs but collaborate to derive a shared encryption key. In this case, decryption typically requires each client to contribute their share of the secret key to recover the final result [20].

Despite significant advancements and various HE designs, practical deployments of HE, especially in real-world applications such as FL, still face challenges that limit scalability, efficiency, and usability. The most notable issues include computational complexity, ciphertext expansion, and noise accumulation.

Computational complexity in HE is influenced by the fact that arithmetic operations on ciphertexts are several orders of magnitude slower than on plaintext, which results in considerable computational overhead, burdening clients with limited computational resources [20]. Ciphertext expansion further aggravates this problem, as HE ciphertexts are significantly larger than the original plaintexts, thereby increasing communication, storage, and processing requirements. This overhead is particularly problematic in FL, where frequent exchanges of large model updates occur between clients and server [20]. In addition, noise accumulation presents a fundamental limita-

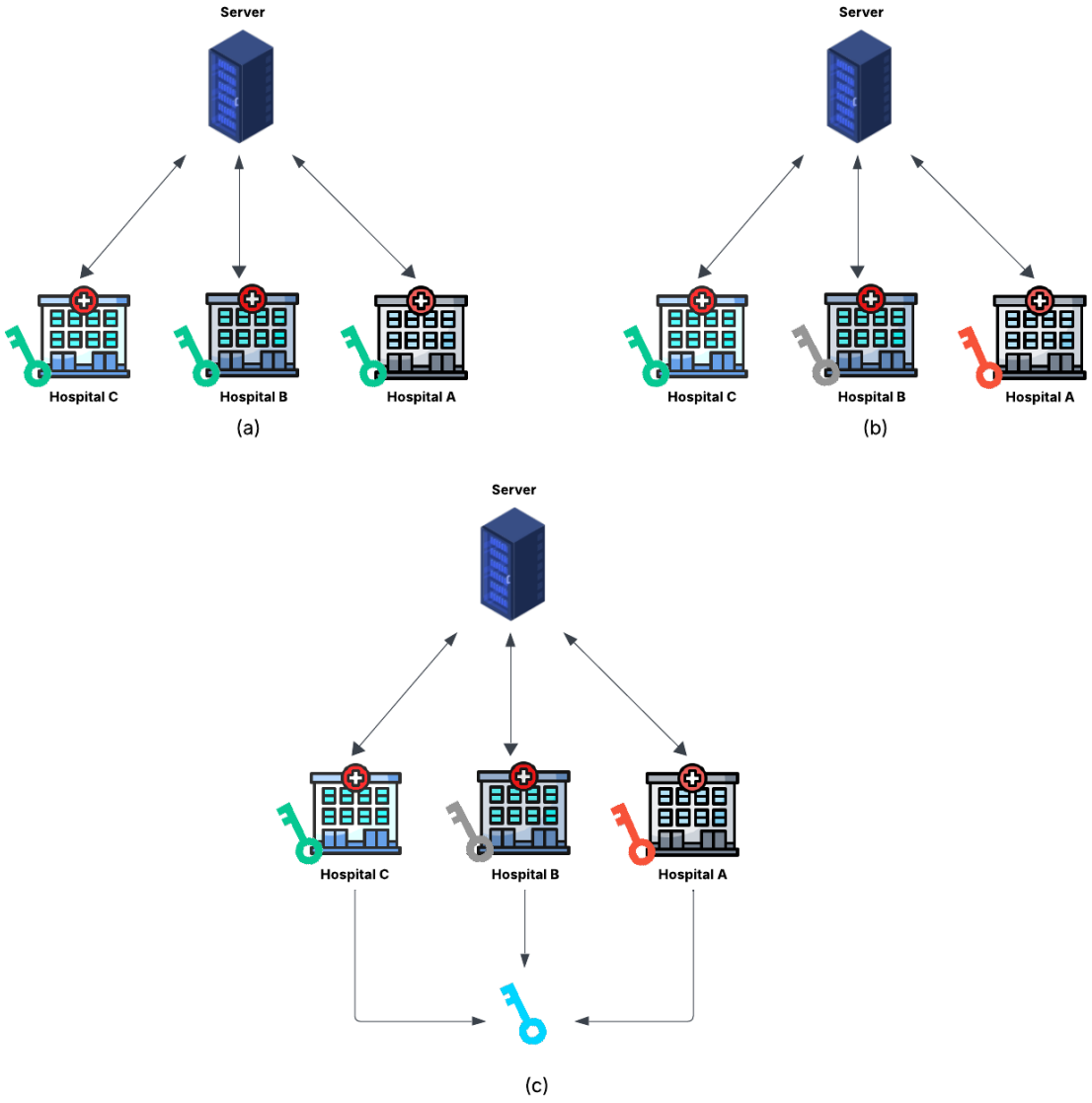

Figure 2.5: Different HE key setups in FL scenarios: (a) all clients share the same key pair (b) each client uses a distinct key pair without cross-key operations (c) clients collaborate to form a shared public key

tion. Most HE schemes add noise during encryption to ensure semantic security. As homomorphic operations, especially multiplications, are performed, this noise grows and may eventually lead to decryption failure or incorrect results [20].

2.2.1 Software Libraries for Homomorphic Encryption

Implementing HE schemes from scratch is a challenging and time-consuming task due to their mathematical complexity. To simplify their use, specialized software libraries have been developed, providing implementations of various HE schemes along with tools for testing and experimentation.

Microsoft SEAL [64], developed by Microsoft, is one of the most well-know HE libraries, currently supporting the BGV, BFV and CKKS schemes. Although it has a steep learning curve, especially for users unfamiliar with HE concepts, the available documentation and examples make it a user-friendly library. The library is implemented in C++ and also provides .NET wrappers.

The HElib [65] library, by IBM, supports the BGV and CKKS schemes. It applies various optimizations to make the HE operations faster, namely the Smart-Vercauteren [66] ciphertext packing techniques and the Gentry-Halevi-Smart [67] optimizations. Ongoing development focuses on enhancing its reliability, robustness, serviceability, performance, and usability. The library is implemented in C++ and relies on the NTL[1] mathematical library.

The HEAAN [68] library was the reference implementation of the CKKS scheme. The initial versions of the library did not include bootstrapping, but the most recent ones do. It is written in C++ and is no longer actively maintained. An implementation of version 1.0 can be found online[2].

The TFHE [69] library was released as the reference implementation of the scheme originally proposed by its authors. It specializes in fast bitwise operations, allowing any binary gate to be evaluated homomorphically in about 13 milliseconds on a single core. This library, written in C/C++, is no longer actively maintained. Further optimizations of the TFHE scheme are available in the TFHE-rs [70] library. Developed by Zama in Rust, TFHE-rs extends the original implementation by providing improved performance and additional integer capabilities.

The OpenFHE [71] library, written in C++, is a community-driven project developed by authors of other well-known HE libraries, such as HElib and HEAAN. It supports all major HE schemes and provides several bootstrapping designs. In addition, it includes extensions such as threshold FHE and proxy re-encryption for the BGV, BFV, and CKKS schemes, and interactive bootstrapping for threshold CKKS.

The TenSEAL [72] library, developed by OpenMined, is a Python library built on top of Microsoft SEAL. It focuses on homomorphic operations on tensors for privacy-preserving machine learning. Supports both the BFV and CKKS schemes.

Pyfhel [73] is another Python-based library. It supports the BGV, BFV, and CKKS schemes, leverages Microsoft SEAL and OpenFHE as backends, and is compatible with machine learning algorithms.

Table 2.2 provides a summary of the libraries described.

2.3 Hybrid Homomorphic Encryption

HHE is a cryptographic approach that combines multiple encryption techniques with HE schemes to improve efficiency, with the concept initially proposed by Naehrig et al. [74]. Recent research has focused on integrating lightweight symmetric ciphers with HE to mitigate the substantial computational and communication overhead typically associated with HE.

[1] https://libntl.org/
[2] https://github.com/kimandrik/HEAAN?tab=readme-ov-file

Library	Developer	Language	BGV	BFV	CKKS	TFHE
SEAL	Microsoft	C++	✓	✓	✓	
HElib	IBM	C++	✓		✓	
HEAAN	CRYPTOLAB	C++			✓	
TFHE	-	C/C++				✓
TFHE-rs	ZAMA	Rust				✓
OpenFHE	-	C++	✓	✓	✓	✓
TenSEAL	OpenMined	Python		✓	✓	
Pyfhel	-	Python	✓	✓	✓	

Table 2.2: Features of Fully Homomorphic Encryption Software Libraries

To better understand this combination, a definition of a symmetric encryption scheme is needed. Unlike public-key schemes, symmetric encryption relies on a single key for both encryption and decryption. A symmetric encryption scheme is typically defined as a tuple of three PPT algorithms, SKE = (KeyGen, Enc, Dec), such that:

- SKE.KeyHen(1^λ): Given a security parameter λ, generates a secret key sk.
- SKE.Enc(m, sk): Encrypts the message m with the secret key sk, returning a ciphertext c_{SKE}.
- SKE.Dec(c_{SKE}, sk): Decrypts the ciphertext c_{SKE} with the secret key sk, returning a plaintext m.

This symmetric encryption scheme serves as a basis for the hybrid strategy. In practice, HHE utilizes the efficiency of symmetric encryption for data encryption, while using HE to securely perform computations on that encrypted data. One notable example of this strategy is the *transciphering* framework [74], which provides a secure method to convert symmetric ciphertexts into homomorphic ciphertexts for further processing.

Figure 2.6 illustrates the core components of a basic transciphering framework. On the client side, a plaintext message m is encrypted using a symmetric secret key sk and a symmetric encryption function *SKE.Enc*, producing the symmetric ciphertext c_{SKE}. Simultaneously, the symmetric key sk is encrypted under a homomorphic public key *HE_pk*, using a homomorphic encryption function *HE.Enc*, resulting in a homomorphic ciphertext sk_{HE}. Both of these ciphertexts are then sent to the server.

On the server side, the symmetric ciphertext c_{SKE} is encrypted under the homomorphic public key *HE_pk* with the homomorphic encryption function *HE.Enc*, producing a homomorphic ciphertext c_{HE}. Using the homomorphic evaluation key *HE_evk*, the server then uses the homomorphic evaluation function *HE.Eval* to homomorphically apply the symmetric decryption function *SKE.Dec*, operating on the homomorphically encrypted symmetric key sk_{HE} and the homomorphic ciphertext c_{HE}. This evaluation simulates the decryption of the symmetric ciphertext c_{SKE} under the symmetric key sk, but entirely within the encrypted domain, resulting in a homomorphic ciphertext of the original plaintext m_{HE}. In this work, this step will be referred to as Homomorphic Evaluation of Symmetric Decryption (HESD) for ease of reference.

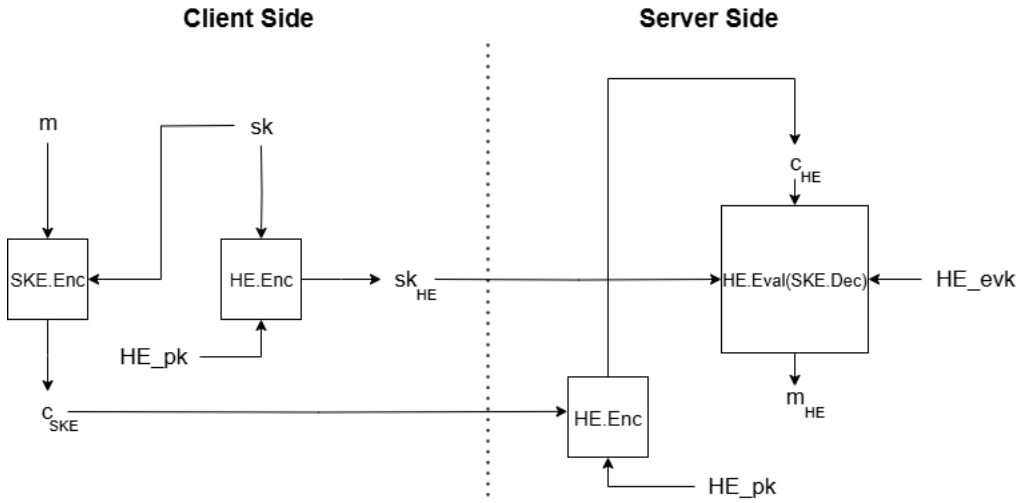

Figure 2.6: Transciphering Framework

This 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, given that symmetric ciphertexts are much smaller than their homomorphic counterparts. However, these benefits come at the cost of increased computational load on the server, which must handle the more demanding homomorphic evaluation.

To further alleviate the server-side computational overhead, several symmetric ciphers have been specifically designed for compatibility with homomorphic ciphers. These so-called HE-friendly ciphers help reduce the cost of the HESD step, thereby improving the overall practicality of transciphering-based approaches.

The most well-known HE-friendly ciphers include the previously mentioned PASTA [22], HERA [75] and Elisabeth [76]. PASTA is a stream cipher designed to minimize multiplicative depth, being optimized for integer use cases over $\mathbb{F}_p$ with p 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 $q = 2^4$. Table 2.3 summarizes the key properties of these ciphers.

	PASTA [22]	**HERA [75]**	**Elisabeth [76]**
Data Type	Integers modulo p	Floating point	Bits
HE Schemes	BGV/BFV	CKKS	TFHE
Defined Over	$\mathbb{Z}_p$ with p a 16-bit prime	$\mathbb{Z}_q$ with $q > 2^{16}$	$\mathbb{Z}_q$ with $q = 2^4$

Table 2.3: Properties of some HE-friendly Ciphers

An important property of the HE schemes BGV, BFV, CKKS and TFHE is that they support homomorphic operations between a ciphertext and a known constant from the underlying ring

or field, allowing operations such as $Enc(m) \odot k$, where k is known. BGV, BFV, and CKKS are based on the RLWE problem, in which ciphertexts and plaintexts are represented as polynomials in a cyclotomic ring. This representation allows ciphertext and plaintext operations to be performed simply by adding or multiplying by a constant in the ring [58, 24, 59]. Similarly, TFHE is based on a torus-based generalization of the Learning with errors (LWE) problem, called TLWE, in which both ciphertexts and plaintexts are represented in the same structure, enabling efficient operations between them [61]. As a result of this property, in most HHE schemes the intermediate step of re-encrypting the symmetric ciphertext c_{SKE} into a homomorphic ciphertext c_{HE} is not required. The server can instead operate directly on c_{SKE} during the HESD step, which reduces server-side computational load and simplifies the overall process. This streamlined approach is illustrated in Figure 2.7.

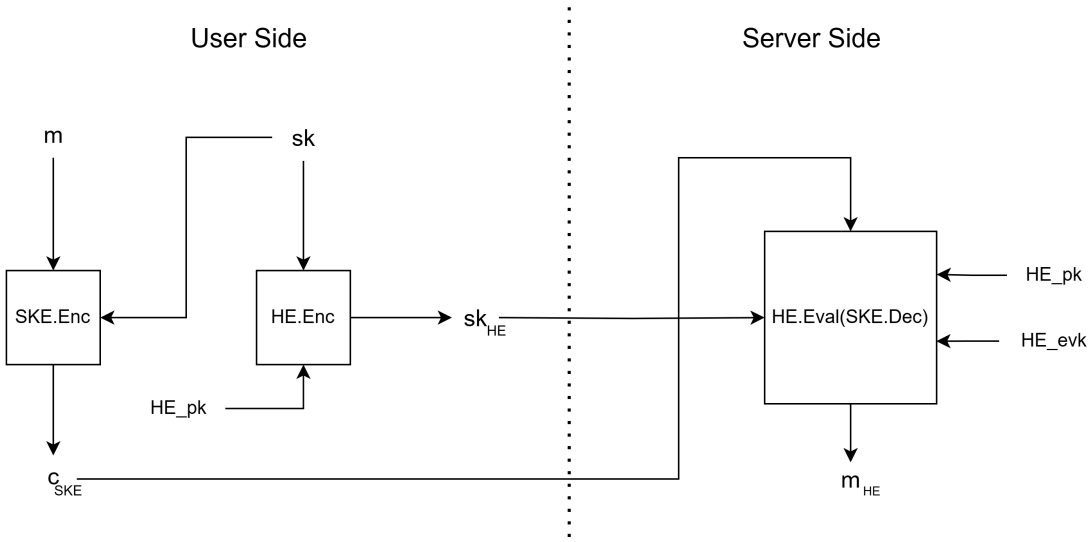

Figure 2.7: Transciphering Framework for HE schemes allowing ciphertext-plaintext operations

2.3.1 Software Libraries for Hybrid Homomorphic Encryption

Several software libraries have been developed to provide public implementations of the HE-friendly ciphers introduced earlier. These libraries enable researchers to experiment with HHE schemes and integrate the ciphers into practical applications.

The authors of HERA introduced the RtF Transciphering Framework [77], a HHE implementation in Go utilizing the lattigo [78] library. This framework integrates the HERA cipher with the CKKS scheme. Additionally, it offers benchmarking capabilities, allowing users to evaluate performance with various parameter configurations. The Elisabeth authors released an implementation [79] of the cipher in Rust, paired with the TFHE cipher through the Concrete [80] library, which is built on top of TFHE-rs. They also provided a use case [81], demonstrating homomorphic inference on encrypted data generated by the HESD step, derived from the Fashion-MNIST dataset, which consists of images of clothing items. Finally, the PASTA authors introduced the

HHE framework [82]. This C++ framework was developed to benchmark HHE schemes and includes implementations of several HE-friendly ciphers, such as PASTA and HERA, integrated with various HE schemes: BGV via HElib, BFV via Microsoft SEAL, and TFHE via the TFHE library. It provides tools for testing and benchmarking, enabling reproducible performance comparisons across different cipher combinations.

The software libraries and frameworks discussed above move HHE beyond theoretical proposals into real-world experimentation. They showcase a variety of approaches for integrating HE-friendly ciphers with different HE schemes, supporting development for diverse application scenarios. This chapter also highlighted the key challenges in FL, particularly heterogeneity and privacy threats, demonstrating why these issues continue to be main obstacles to its broader use. To address privacy concerns, HE was presented as a cryptographic technique that enables computations on encrypted data, thereby reinforcing privacy during transmission and aggregation. However, its high computational and communication costs limit its practical applicability in resource-constrained environments such as IoT. HHE addresses these limitations by combining the efficiency of symmetric encryption with the security of HE, reducing client-side overhead while preserving privacy. Together, these concepts provide the foundation for the solutions developed and assessed throughout the remainder of this thesis.

Chapter 3

Literature Review

This chapter presents the literature review conducted to answer the aforementioned research questions, following the PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) guidelines for systematic review reporting and conduct [83].

3.1 Privacy methods in FL

To answer RQ1: "What are the current methods used to protect data privacy in FL", this section reviews various privacy-preserving mechanisms that have been implemented to address the inherent privacy concerns in FL.

3.1.1 Data Sources and Search Strategy

To gather relevant documents, the following databases were searched: Scopus[1], ACM[2], IEEE Xplore[3], PubMed[4] and Web of Science[5]. The search strategy involved choosing specific fields within these databases, namely the title, abstract, and keywords.

Given the broad scope of this research question, *reviews* and *surveys* were the focus of this research. The search terms used were derived based on the research question itself, like *federated learning* and *privacy-preserving*. To ensure that HE-based methods were adequately captured, particularly any references to HHE, an additional search including *homomorphic encryption* was conducted as a verification step to confirm that relevant surveys were not missed by the broader query. Based on this strategy, the search queries listed below were constructed. To focus on surveys and reviews specifically related to FL and privacy-preserving methods, any articles containing *blockchain* in the title were excluded, due to the large number of such articles returned by the initial search. This exclusion was applied at the title level only, so as not to remove papers that

[1] https://www.scopus.com
[2] https://dl.acm.org/
[3] https://ieeexplore.ieee.org
[4] https://pubmed.ncbi.nlm.nih.gov/
[5] https://www.webofscience.com/

might discuss blockchain-related methods within the body of the text. The queries used were the following:

- *("federated learning" AND ("privacy-preserving") AND ("survey" OR "review" OR "overview" OR "synthesis")) AND NOT ("blockchain*")*
- *"homomorphic encryption" AND "federated learning" AND ("privacy-preserving") AND ("survey" OR "review" OR "overview" OR "synthesis") AND NOT ("blockchain*")*

3.1.2 Studies' Selection

The process of selecting the final articles involved three main phases: identification, screening, and eligibility. Each phase is designed to narrow down the results systematically and ensure the inclusion of only relevant studies.

3.1.2.1 Identification Phase

This phase involved collecting studies from the selected databases and then removing duplicates. To evaluate the quality of a study, two factors were considered: the number of citations and the ranking of the journal in which it was published . For the citations, the highest number found across all databases was used. Journal rankings were determined based on the 2023 SJR score from Scimago Journal & Country Rank[6].

3.1.2.2 Screening Phase

After gathering all the studies, the documents were screened by reviewing their titles and abstracts. During this phase, the inclusion and exclusion criteria presented in Table 3.1 were applied to select relevant studies for further analysis.

Inclusion Criteria	**Exclusion Criteria**
IC1- Date of the publication between 2020-2025	EC1 - Article not focused on FL
IC2 - Available in English	EC2 - Article focused on FL within a specific domain or application area (e.g., healthcare, IoT, finance) rather than providing an overview of concepts, challenges, and/or methodologies
	EC3 - Article has 0 citations, or the publishing journal does not have a ranking in SJR

Table 3.1: Inclusion and Exclusion Criteria for Screening Phase

[6] https://www.scimagojr.com/

3.1.2.3 Eligibility Phase

For the remaining documents, a full text analysis was made. This phase involved using the exclusion criteria presented in Table 3.2.

	Exclusion Criteria
EC4	Article not discussing HE
EC5 a)	For articles published in 2025 or earlier: Exclude documents with fewer than 50 citations. From those, select the 2 with the best combination of citation count and SJR score.
EC5 b)	For articles published in 2025 or later: Exclude documents with fewer than 10 citations (unless published within the last 2 months). From those, select the 5 with the best combination of citation count and SJR score.

Table 3.2: Inclusion and Exclusion Criteria for Eligibity Phase

Focusing on reviews means that the information can be repetitive. To counter this, the EC5 criteria was created. Only picking the best reviews from each year ensures a high-quality selection of articles, preventing redundancy and prioritizing the most relevant and impactful studies. By selecting the articles with the highest combination of citations and SJR score, those with major influence in the field are prioritized, while also considering the academic reputation of the journals in which they were published. Note that the minimum number of citations required for an article was determined based on the documents that passed the screening phase. Additionally, the number of articles selected is higher for 2025, as they are the most recent studies.

3.1.3 Results

Figure 3.1 illustrates the flowchart of the systematic review process. The databases searches resulted in a total 973 records. The screening process began with the removal of 363 duplicate records, followed by an examination of the titles and abstracts, which led to the exclusion of 556 records. Finally, the full texts of the remaining 54 records were analysed, resulting in the exclusion of an additional 41 records. Therefore, 13 studies were deemed eligible and included in this study.

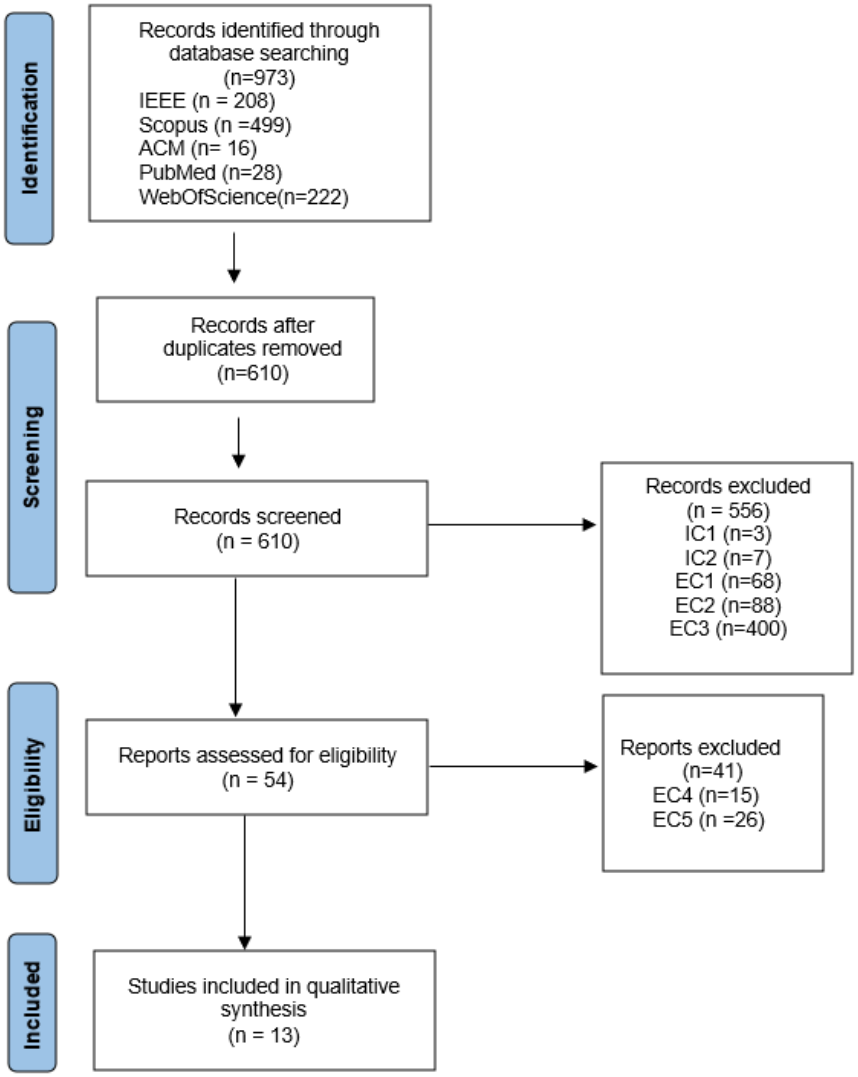

Figure 3.1: PRISMA Flow Diagram for RQ1

Table 3.3 summarizes the privacy-preserving methods mentioned in the reviewed articles. The three main methods are Differential Privacy, Homomorphic Encryption, and Secure Multi-Party Computation. In addition, hybrid protocols have been proposed which combine multiple privacy-preserving techniques. While some methods, such as locality-sensitive hashing, have become less used over the years, others, like blockchain, have gained prominence due to advancements in technology.

Method	**[84]**	**[16]**	**[85]**	**[86]**	**[39]**	**[87]**	**[88]**	**[17]**	**[89]**	**[19]**	**[90]**	**[91]**	**[92]**
DP	x	x	x	x	x	x	x	x	x	x	x	x	x
HE	x	x	x	x	x	x	x	x	x	x	x	x	x
SMC	x	x	x	x	x	x	x	x	x	x	x	x	x
Locality-Sensitive Hashing	x												
Anonymization Techniques				x									x
Secret sharing				x					x				
Additive/ Multiplicative perturbation				x									
Trusted Execution Environment						x	x					x	
One-Time Pad													
Zero-Knowledge Proof								x					
Blockchain									x			x	
Hybrid	x	x	x	x	x	x	x	x	x				

Table 3.3: Privacy-Preserving Methods Mentioned in the Eligible Studies

Even though a variety of privacy-preserving methods have been proposed, the following discussion focuses on those approaches most frequently addressed in the literature, reflecting their relevance and practical significance in FL research, namely Differential Privacy, Homomorphic Encryption, Secure Multi-Party Computation, Anonymization Techniques, Secret Sharing, Trusted Execution Environment, Blockchain, and Hybrid Protocols.

3.1.3.1 Differential Privacy

Differential Privacy is a technique designed to prevent data leakage by introducing artificial noise to the data, making it difficult for attackers to infer sensitive information. Adding too much noise can negatively affect model accuracy, while too little noise may compromise privacy [90]. In FL, DP is typically divided into two primary categories: *Global (or Centralized) Differential Privacy* and *Local Differential Privacy* [16, 86, 17, 91]. These approaches mainly differ in where the noise is applied and the level of trust assumed in the aggregator.

Global Differential Privacy: In Global Differential Privacy (GDP), a trusted server receives the clients' local updates in plaintext, and applies noise to the aggregated global model. This ensures that malicious clients cannot infer sensitive information about other clients from the shared global model [86, 91]. By applying noise only to the aggregated global model, GDP schemes can preserve data privacy while maintaining good accuracy, as the noise is added in a controlled manner that protects sensitive information without significantly affecting the model's performance [86]. However, GDP schemes are vulnerable to malicious servers, as the server has access to the "clean" local updates of all clients. [86, 17]. GDP is suited for cross-device environments for it to be able to converge and obtain an acceptable trade-off between privacy and accuracy [17].

Local Differential Privacy: Local Differential Privacy (LDP) provides stronger privacy guarantees than GDP by requiring clients to add noise to their local model updates before sending them to an untrusted aggregator [86]. Because of this, the amount of noise added by clients is significantly higher than in GDP schemes [86], which can lead to a notable reduction in model accuracy, particularly in cross-device scenarios [17].

Overall, DP schemes are computationally efficient and provide strong privacy protection. GDP achieves a good balance between privacy and model accuracy by adding noise to the aggregated global model, while LDP offers stronger privacy guarantees by adding noise locally, at the cost of reduced accuracy. Both approaches effectively mitigate membership inference attacks, although only GDP can defend against property inference attacks, typically with some loss of model performance [91, 17].

3.1.3.2 Homomorphic Encryption

As explained before, HE is a cryptographic technique that allows computations to be performed directly on encrypted data without needing to decrypt it [18], and can either be partially, somewhat, or fully homomorphic depending on the operations they support [17].

In FL, HE ensures the privacy of data during processing and aggregation. Clients can encrypt their local model parameters or gradients before transmitting them to the server using a public key. Afterwards, the encrypted global model is sent back to the clients, who can decrypt it using the private key and continue with local training [19]. The two most popular HE schemes for FL are the PHE scheme Paillier and the FHE scheme CKKS [89, 19]. The former is due to its ability to encrypt integers and perform homomorphic functions over them. The latter is due to its capability of performing homomorphic addition and multiplication in floating point values, which is preferred for machine learning scenarios [89, 19].

Despite the strong privacy guarantees, HE presents several challenges. The computational overhead of encryption, particularly for FHE, can be significant for clients, requiring substantial local resources. Encrypted data also results in larger ciphertexts, increasing communication costs [17]. Secure key management is another critical concern, usually determining the threat model and application scenarios [39]. Furthermore, HE systems often face scalability issues, as the combined overhead of encryption and transmission can create challenges in cross-device scenarios [88].

3.1.3.3 Secure Multi-Party Computation

Secure Multi-Party Computation encompasses cryptographic techniques that enable multiple parties to collaboratively compute a function over their inputs without revealing those inputs to each other [86].

In the context of FL, SMC provides a mechanism for privacy-preserving aggregation of local models updates or gradients, ensuring that sensitive data from clients remains confidential throughout the training process, making it suitable for honest-but-curious client scenarios. However, SMC does not protect the final model, which remains vulnerable to inference and poisoning attacks. To address these vulnerabilities, additional techniques, such as DP, may be necessary [87, 17].

The use of SMC in FL implies high computational and communication overhead, although not as high as HE. Another limitation is that all clients need to remain online and actively involved during the computation process. Dropouts or failures can disrupt the protocol, requiring mechanisms to handle participant unavailability. All of these factors make SMC less suitable for cross-device scenarios [17].

3.1.3.4 Anonymization Techniques

Anonymization techniques are primarily used to achieve anonymity by removing any personal information from data that could be used to identify a specific user, while also ensuring the data can still be useful for further operations [86, 92]. In the context of FL, this could involve transforming a client's local model updates to prevent the server from inferring details about the local dataset, while still being usable for aggregation.

The three most commonly used anonymization techniques are *k-anonymity*, *l-diversity*, and *t-closeness* [86]. These schemes vary in how they protect different aspects of the data, such as unique identifiers and sensitive attributes. *k-anonymity* ensures that each client's data is indistinguishable from at least $k-1$ others, which protects records against re-identification based on unique identifiers. However, it can be vulnerable to inference attacks on the sensitive attributes of the records [86]. To mitigate this, *l-diversity* extends *k-anonymity* by ensuring that sensitive attributes within each group are sufficiently diverse. Nonetheless, this diversity can sometimes still allow inference attacks if the distribution of values is predictable [86]. To further enhance protection, *t-closeness* was proposed as an extension of *l-diversity* by maintaining the distribution of sensitive attributes. This method is often more effective than the other two approaches when applied to numeric attributes. However, strict application of *t-closeness* tends to degrade the usefulness of data [86].

A more recent variation is the *(k,km)-anonymity* approach. In this method, k ensures that each group contains at least k indistinguishable records, while km guarantees a minimum number of distinct values for sensitive attributes within each group, providing protection against re-identification based on the unique identifiers and inference of the sensitive attributes [92].

3.1.3.5 Secret Sharing

Secret Sharing is a cryptographic technique that protects sensitive information by splitting it into multiple shares. The original secret can be reconstructed only when a sufficient number of shares, called the threshold, are combined [86, 89].

In the context of FL, secret sharing can be applied to secure model updates during aggregation. Instead of sending raw updates to the server, each participant sends shares of their update. The server can then aggregate the shares without learning any individual update, ensuring privacy even if some participants collude [86].

Nevertheless, secret sharing is not immune to threats. Reconstruction relies on at least a threshold number of honest participants submitting correct shares, and malicious behavior by enough participants or the server can compromise the aggregation process [86]. Furthermore, as a cryptographic technique, secret sharing introduces additional computational and communication overhead, which can impact the performance of cross-device FL systems [86].

3.1.3.6 Trusted Execution Environment

A Trusted Execution Environment (TEE) is a secure computing area that isolates sensitive data and code from the rest of the system, ensuring that data can be stored and processed in a trusted environment independent of the main operating system [91].

In FL, TEEs ensures data privacy and integrity during aggregation. They can be used to process local updates in a secure and isolated environment, allowing for privacy-preservation aggregation [87]. TEEs can also be used to prevent information leakage [88, 91] and potentially protect against model poisoning attacks [91].

TEEs are particularly useful in scenarios where clients do not fully trust the aggregator. However, a major limitation is hardware dependency, as TEEs require specific implementations that restrict their use in cross-device FL setups. Additionally, they often have limited computational resources, such as memory and processing power, which can create performance bottlenecks for resource-intensive tasks like large machine learning models [91].

3.1.3.7 Blockchain

Blockchain is a decentralized, transparent and tamper-proof ledger system designed to record transactions in an immutable and ordered manner. Transactions are verified by untrustworthy blockchain nodes through a decentralized consensus protocol before being aggregated into blocks, which are then added to the blockchain. Each block contains a cryptographic hash of the previous block, ensuring traceability of data, thus preserving the integrity of the chain [89].

Blockchain brings several advantages for FL. Decentralization eliminates the need for a centralized authority, as transactions require confirmation from a majority of blockchain nodes. Immutability ensures that once a transaction is recorded, it cannot be altered unless a majority of nodes are compromised, with tampering made visible to all network participants. Finally, blockchain's transparency allows all transactions to be visible and traceable for verification, while

digital signature techniques enable anonymous transactions without relying on trusted third parties [89, 91].

Blockchain's decentralization benefits FL by distributing the aggregation task across nodes, reducing risks from single points of failure in centralized FL systems. It also ensures the auditability of processes such as gradient collection and model aggregation, thus enhancing security and preventing poisoning attacks from malicious clients. Additionally, consensus mechanisms keep data authenticity and offer incentives for participation, which can help identify reliable and trusted clients [89, 91].

3.1.3.8 Hybrid Protocols

Hybrid protocols in FL combine multiple privacy-preserving techniques to capitalize on the benefits of different approaches while mitigating their individual limitations. Such methods are designed to balance data privacy, computational and communication efficiency, and model performance, addressing the problems that arise from deploying the individual privacy solutions [86].

Hybrid approaches usually combine cryptographic methods like HE and SMC with techniques based on perturbation like DP. This integration enables robust protection against privacy attacks while reducing the computational and communication overhead typically associated with pure cryptographic protocols [86]. For instance, SMC can be combined with DP to reduce the noise required for privacy preservation while maintaining high model utility, ensuring that no sensitive information is disclosed during communication and mitigating inference attacks [86]. Other prominent approaches involve using HE alongside secret sharing to enhance confidentiality during aggregation. While HE ensures the encryption of local models, secret sharing facilitates dropout resilience by handling the absence of certain users during the training process [86]. Compression techniques are often integrated to alleviate the communication and computational costs associated with HE, SMC, and DP protocols, further improving the efficiency of hybrid methods [88].

To sum up, hybrid protocols, by definition, combine distinct cryptographic and perturbation techniques, such as HE, SMC, DP as well as infrastructures like TEE and Blockchain to enhance the privacy guarantees and robustness of a FL scheme. These protocols leverage the strengths of diverse approaches to address the limitations of using a single method.

To answer RQ1, the current methods used to protect data privacy in FL can be broadly categorized into *cryptographic approaches* such as HE, SMC, and secret sharing, *perturbation-based methods* including DP and anonymization, and *trusted infrastructures* like TEE and blockchain. *Hybrid protocols* are increasingly gaining popularity, as they combine these techniques to balance privacy guarantees, computational and communication efficiency, and model performance. Among these, DP, HE, and SMC remain the most prominent due to their strong theoretical foundations and practical deployment. In contrast, infrastructures such as TEE and blockchain are mainly complementary, providing secure environments and decentralized trust. Overall, hybrid protocols represent a promising direction, as they leverage the strengths of multiple methods to mitigate the limitations of using a single technique.

In the context of this thesis, the *cryptographic approaches* studied in the literature are generally not designed for resource-constrained FL environments due to their elevated computational and communication overhead. This gap motivates the exploration of HHE as a promising solution, offering a balance between strong privacy guarantees and efficiency suitable for such settings.

3.2 Homomorphic Encryption in FL for IoT

This section aims to answer RQ2: "How do different homomorphic encryption approaches operate in federated learning for IoT devices, and how effective are they in enhancing privacy and security? ", reviewing state-of-the-art FL protocols that incorporate HE in IoT environments. These protocols address the complex challenge of securing distributed learning on resource-constrained devices, striving to balance privacy, security, and efficiency.

3.2.1 Data Sources and Search Strategy

The data sources and search strategy used in this study were the same as in the previous research, which involved searching the following databases: Scopus[7], ACM[8], IEEE Xplore[9], PubMed[10], and WebofScience[11]. The search focused on the fields title, abstract, and keywords.

The search terms for this study were updated to align with the new research question. These terms include *homomorphic encryption*, *federated learning* and *IoT* and all its variations. Articles containing *blockchain* in the title continued to be excluded. Unlike earlier work, the focus was not limited to surveys and reviews. The search query used is presented below, with the exclusion of *blockchain* applied only to the article titles:

- "homomorphic encryption" AND ("federated learning") AND (iot OR "internet of things" OR "internet-of-things")) AND NOT ("blockchain*")

3.2.2 Studies' Selection

The process of selecting the final articles followed the same three-phase approach as outlined in our previous research: identification, screening, and eligibility. Each phase is designed to narrow down the results systematically and ensure the inclusion of only relevant studies.

3.2.2.1 Identification Phase

This phase involved collecting studies from the selected databases and removing duplicates. To evaluate the quality of a study, journal rankings were determined based on the 2023 SJR score from Scimago Journal & Country Rank[12], as done in the previous study.

[7] https://www.scopus.com
[8] https://dl.acm.org/
[9] https://ieeexplore.ieee.org
[10] https://pubmed.ncbi.nlm.nih.gov/
[11] https://www.webofscience.com/
[12] https://www.scimagojr.com/

3.2.2.2 Screening Phase

Once all the studies were gathered, the documents were screened by reviewing their titles and abstracts. During this phase, new inclusion and exclusion criteria were applied to select relevant studies for further analysis, as presented in Table 3.4.

Inclusion Criteria	**Exclusion Criteria**
IC1- Date of the publication between 2020-2025	EC1 - Article conference proceeding
IC2 - Available in English	EC2 - Article publishing journal does not have a ranking in SJR
	EC3 - Article primarily summarizes existing research
	EC4 - Article only discusses HE for comparison purposes, without implementing it as part of its proposed solution.

Table 3.4: Inclusion and Exclusion Criteria for Screening Phase

3.2.2.3 Eligibility Phase

For the remaining documents, a full text analysis was made. This phase introduced a new exclusion criteria:

- **EC5** - Articles with insufficient information

This criterion excludes articles that present gaps in detail, such as a lack of sufficient methodological information, absence of data that would allow for result replication, or performance evaluations that rely solely on metrics like accuracy without considering other relevant factors such as computation time or communication cost. These factors are important because computation and communication overhead are well-known challenges in the application of HE in FL, and addressing them is crucial for a comprehensive evaluation of the effectiveness of HE in FL.

3.2.3 Results

Figure 3.2 illustrates the flowchart of the systematic review process. The database search yielded a total of 185 studies. During the screening process, 88 duplicates were removed. Next, a review of the titles and abstracts led to the exclusion of 51 studies. Afterward, the full-text examination resulted in the exclusion of 25 more studies. Ultimately, 21 studies were deemed eligible and included in this analysis.

To support this analysis, two summary tables are provided. Table 3.5 outlines the general features of the FL systems when using HE, including their architectures, the entities considered,

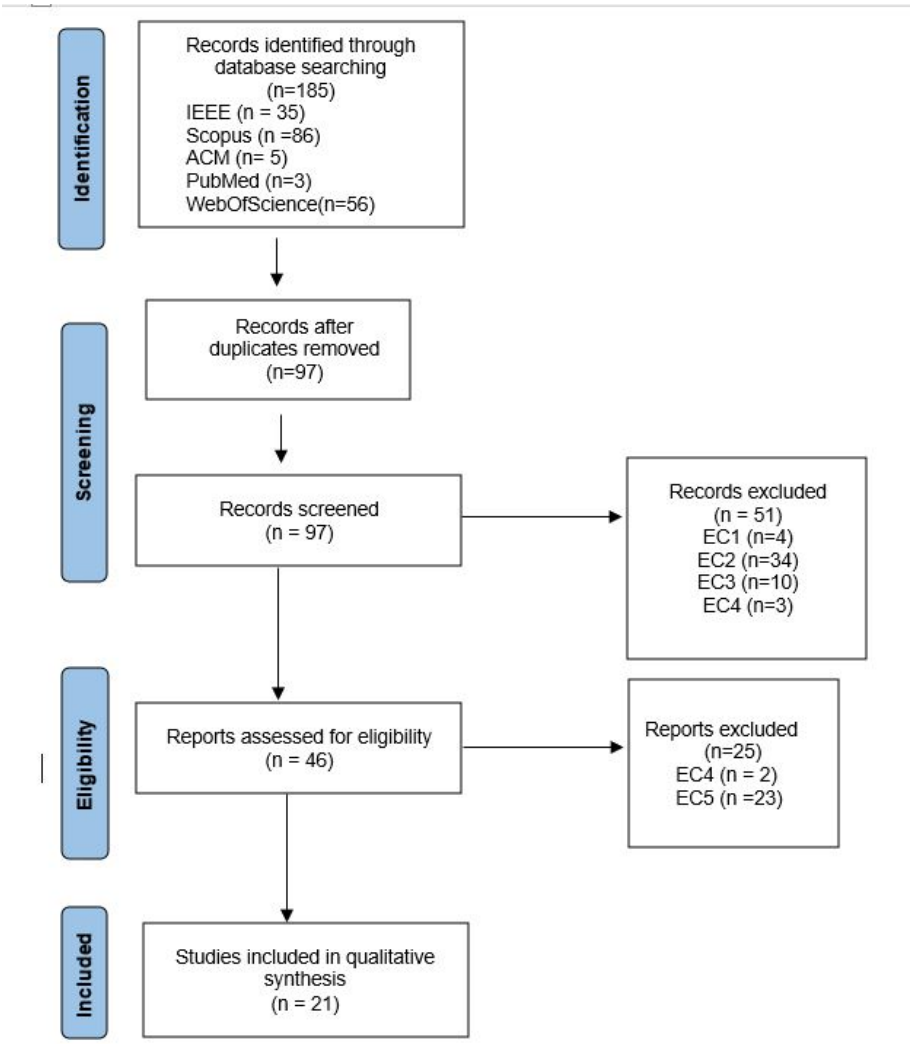

Figure 3.2: Prisma Flow Diagram for RQ2

the assumed threat model, aggregation methods, and the evaluation datasets. Table 3.6 details the homomorphic encryption schemes employed, key distribution mechanisms, encrypted data, and decryption responsibilities. It also highlights additional techniques used by the protocols to enhance security, such as defenses against poisoning and collusion attacks, and ensure robustness and communication efficiency, including mechanisms for handling user dropout, ignoring irrelevant updates, and compressing data.

As shown in Table 3.5, the majority of the protocols adopt a centralized FL architecture, where a central server is responsible for aggregating model updates. It is also common to include a trusted third-party authority to handle the key generation process, even in cases where identical keys are distributed to all users, as seen in Jalali et al. [93]. Only two works, Hijazi et al. [94] and Duy et al. [95], apply HE in decentralized environments. Among them, Hijazi et al. employs a fully decentralized scheme, where no central server exists. Instead, a capable user is selected in each round to coordinate and aggregate local updates. In contrast, Duy et al. scheme

is considered semi-decentralized, as it integrates a blockchain infrastructure for task coordination and procedure management, while still relying on a central server to perform model aggregation. Regarding the threat model, most protocols assume that both clients and the server are honest-but-curious [96, 97, 98, 99, 100], although not all allow for collusion between them [98, 100]. Third-party adversaries are typically modeled as eavesdroppers who can observe the data in transit. Only Ren et al. [101] considers a more active adversary capable of tampering with the messages exchanged by participants. The choice of aggregation method plays a role not only in the model's final accuracy but also in the system's efficiency and robustness. A robust aggregation strategy can help mitigate the effects of user dropouts and filter out irrelevant or malicious updates. The most commonly used aggregation algorithm is FedAvg [93, 63, 96, 97, 102]. Lastly, the datasets used in these works reveal that the underlying models often contain tens of thousands of parameters, which increases communication overhead, especially when paired with HE. This highlights the need for more efficient data handling techniques prior to encryption.

Table 3.6 shows that all articles adopting a fully homomorphic encryption scheme utilize CKKS or a multi-key variant of it, while those employing PHE tend to favor schemes optimized for addition, such as Paillier. Regarding key distribution, schemes where each client holds a distinct key offer stronger protection against collusion between a client and the server. This is exemplified by comparing Tan et al. [103] and Jalali et al. works with Ma et al. [96]. All three adopt a fully homomorphic encryption scheme and follow a similar process: clients train local models, encrypt their weights, send them to the server for aggregation, and then receive the global update. However, in Tan et al. and Jalali et al. works, all clients share the same keys, which means a colluding server and client can jointly decrypt others' updates. In contrast, Ma et al. requires clients' decryption shares, preventing such collusion from compromising the privacy of honest users, showcasing the benefits of multi-key HE.

Several studies apply additional techniques, such as cosine similarity, to assess the relevance of client updates [104], which can be combined with other techniques such as K-means clustering to protect against poisoning attacks [94], before aggregation.

In parallel, some works employ dimensionality reduction or gradient filtering methods to improve communication efficiency. For instance, Fontenla-Romero et al. [105] uses Singular Value Decomposition (SVD), Song et al. [98] applies Gradient Filtering Compression, and Cai et al. [106] adopts Principal Component Analysis (PCA). These methods are often implemented before encryption to alleviate the overhead associated with HE.

To address challenges arising from client dropouts, secret sharing is the most used technique, with Xu et al. [107] being able to even resist Byzantine attacks. Fang et al. [108] also shows that Sparse Bidirectional Compression can be further used to improve communication overhead.

To further maintain client data privacy, works apply Double Masking [99, 101] or Differential Privacy [95, 109], to apply noise to the data before it is encrypted and then shared.

Some works also focus on ensuring the authenticity of the communicating parties. This can be done by the use of a simple TLS protocol, as shown by Tan et al. and Jalali et al., or by the use of digital signatures by Ren et al. and Wu et al. [110].

It is also worth noting that Chen et al. [111] introduces a different form of data transformation by applying a blinding factor. In this work, clients are not responsible for decryption, instead, a proxy server performs this task. To ensure that client privacy is still preserved, the clients apply a blinding factor to the aggregated results before forwarding them to the proxy. After decryption, the clients remove the blinding factor to recover the updated global model.

3.3 Hybrid Homomorphic Encryption in FL for IoT

This section addresses RQ3: "How does an HHE approach compare to existing privacy-preserving techniques in FL for IoT devices?"

3.3.1 Data Sources and Search Strategy

The data sources and search strategy followed the same approach as in the earlier researches. The databases Scopus[13], ACM[14], IEEE Xplore[15], PubMed[16], and Web of Science[17] were queried, with the search focused on the fields title, abstract, and keywords.

The search terms from the previous study were updated, replacing *homomorphic encryption* with *hybrid homomorphic encryption* and introducing *transcipher* to include studies that may refer to an HHE scheme as a transciphering scheme. As previously done, any articles containing *blockchain* in the title were excluded. The final search query, with the exclusion of *blockchain* applied only to article titles, is shown below:

- ("hybrid homomorphic encryption" OR "transcipher*") AND ("federated learning") AND (iot OR "internet of things" OR "internet-of-things") AND NOT ("blockchain*")

3.3.2 Studies' Selection

The planned three-phase process of identification, screening, and eligibility could not be carried out, as the search strategy returned zero results across all databases. Consequently, no studies were available for further selection.

3.3.3 Results

The database search yielded no articles matching the defined query, and therefore no studies could be included in the review. This absence of literature highlights a significant research gap in the application of HHE within FL, particularly in IoT contexts, and highlights that no prior studies have applied HHE in this context, making this work the first to do so.

[13] https://www.scopus.com
[14] https://dl.acm.org/
[15] https://ieeexplore.ieee.org
[16] https://pubmed.ncbi.nlm.nih.gov/
[17] https://www.webofscience.com/

Article	FL Scheme	Entities	Threat Model	Aggregation Algorithm	Dataset(s) Used
[103]	Centralized	Clients and Server	Server is honest-but-curious	Average (b)	Self-Collected
[93]	Centralized	Clients, Central Server and Trusted Authority	Eavesdroppers and unauthorized access.	FedAvg	MNIST
[63]	Centralized	Clients, Server, Third party	(*)	FedAvg	UNSW-NB15, Bot-IoT
[96]	Centralized	Clients and Server	Server and Clients are honest-but-curious and may collude.	FedAvg	UP-FALL
[97]	Centralized	Clients and Server	Clients and Server are honest-but-curious and may collude	FedAvg	(*)
[102]	Centralized	Clients and Server	(*)	FedAvg	N-BaIoT
[104]	Centralized	Clients, Server and Trusted Third-Party	(*)	Gradient Similarity Model Aggregation (GSA)	Edge-IIoTset, CIC IoT 2023
[94]	Decentralized	Users (Normal User and User Head)	(*)	FedAvg	WSN, MNIST, CIFAR-10, N-BaIoT
[105]	Centralized	Clients and Aggregation Server (Coordinator)	Eavesdroppers and Non-trustworthy Server	Incremental SVD aggregation	MiniBoone, DryBean×10,Skin, SUSY, HEPMASS, Higgs, Higgs×4
[98]	Centralized	Clients, Server and Key Generation Server	Clients and Server are honest-but-curious. An external adversary can eavesdrop and infer data.	FedAvg	MNIST, CIFAR-10
[106]	Centralized	Guest, Host and Arbiter	Guest and Host are honest-but-curious	Homomorphic Gradient Aggregation (b)	MIMIC-III, Epsilon, NUS-WIDE
[107]	Centralized	Clients, Edge Server and Cloud Server	Server is honest-but-curious. Clients are malicious	FedSGD	MNIST, CIFAR-10
[108]	Centralized	Clients, Server and Trusted Authority	Clients and Server are honest-but-curious and may collude. Eavesdroppers.	Additive Homomorphic Aggregation (b)	MNIST, Shakespeare play
[99]	Centralized	Clients, Server and Third Party Trusted Authority	Clients and Server are honest-but-curious and may collude. Passively malicious adversaries and eavesdroppers.	Weighted averaging based on Data Quality (b)	HAM10000
[101]	Centralized	Clients, Server and Trusted Authority	Trusted Authority is honest and does not collude with any other participant. Clients and Server are malicious and may collude. Third-Party Attackers may eavesdrop, analyze and tamper with the messages	Privacy-Enhanced Traceable Aggregation	MNIST
[110]	Centralized	Clients, Cloud Server and Trusted Authority	Clients are honest-but-curious and Cloud Server can be honest-but-curious or malicious.	AdaGS	Fashion-MNIST (F-MNIST) and KDD CUP 99
[95]	Semi-Decentralized	Threat Hunting Agents (Clients), Aggregation Server, Blockchain Network, Trusted Server	Threat Hunting Agents can be malicious.	FedAvg	InSecLab-IDS-2021
[100]	Centralized	Clients, Server and Auditor	Server and Clients are honest-but-curious. Eavesdroppers.	Hybrid LEGATO	MNIST, CIFAR-10
[112]	Centralized	Clients, Crowdsensing Platform (CSP), Task Requester (TR)	All entities are honest but curious. Server colludes with participants	FedAvg	MNIST, Car Evaluation
[109]	Centralized	Clients and Server	(a)	Federated Stochastic Modification Minimized Gradient (FSMMG)	018 n2c2 Track 2 (Adverse Drug Events and Medication Extraction)
[111]	Centralized	Clients, Proxy Server, Parameter Server	All entities can be malicious. Clients and Parameter Server may collude, but the Parameter Server and Proxy Server do not collude. Eavesdroppers.	Homomorphic Stochastic Gradient Descent (b)	MNIST

(*) Not mentioned in the original paper (a) Not specified in the original paper (b) The original work did not specify an explicit algorithm name, so an interpretative label has been assigned in this review

Table 3.5: Taxonomy of the FL Systems using HE in the Literature

Article	HE Scheme Type	HE Scheme Used	HE Key Distribution	What was encrypted?	Who decrypts the data?	Other Techniques	Protects Against[a]	Handles Drop-out Users?	Handles Irrelevant Updates?	Data Compression
[103]	FHE	Not mentioned	Shared	Local Model Weights	Clients	SSL/TLS	-	×	×	×
[93]	FHE	CKKS	Shared	Local Model Weights	Clients	iTLS Protocol	-	×	✓	×
[63]	PHE	Paillier	Distinct	Local Model Parameters	Clients	-	Data Leakage, Inference Attacks	×	×	×
[96]	FHE	xMK-CKKS	Distinct	Local Model Weights	Server	-	Collusion, Data Leakage	×	×	×
[97]	FHE	ftxMK-CKKS	Distinct	Local Model Weights	Server	-	Collusion Attacks	✓	×	×
[102]	FHE	CKKS[b]	Shared	Local Model Weights / Cluster Matrices	Clients	Clustering	Data Leakage	×	×	×
[104]	PHE	Paillier	Shared	Local Gradients	Server+Clients	Cosine Similarity	Parameter Leakage	×	✓	×
[94]	FHE	CKKS	Not mentioned	Local Model Weights	Users	Cosine Similarity, K-means	Model Poisoning	✓	✓	×
[105]	FHE	CKKS	Shared	Weighted Data Vector	Clients[c]	SVD	Inference Attacks	×	✓	✓
[98]	PHE	Paillier	Shared	Local Gradients	Server	ZKP, Gradient Filtering	Inference Attacks	×	✓	✓
[106]	PHE	Paillier	Shared Public Key	Intermediate Results	Arbiter	PCA, Straggler-Resistance	-	✓	×	✓
[107]	FHE	CKKS	Shared Secret Key	Local Gradients	Edge+Cloud	Threshold Sharing	Inference, Byzantine	✓	✓	×
[108]	PHE	Additive ElGamal	Shared (via Secret Sharing)	Compressed Gradients	Server+Clients	Threshold Sharing, Sparse Compression	Collusion	✓	✓	✓
[99]	PHE	Additive ElGamal	Distinct	Gradient Quality	Server+Clients	Double Masking, Shamir, DH Exchange	Inference, Collusion	✓	×	×
[101]	PHE	Compact-LWE	Distinct	Local Gradients	Clients	Double Masking, DH, Secret Sharing, Hashing, Signatures	Collusion, Poisoning	✓	✓	×
[110]	FHE	CKKS	Distinct	Sparse + Full Parameters	Clients	AdaGS, BLS Signatures	CPA, Poisoning	×	✓	✓
[95]	FHE	CKKS	Shared	Noisy Local Weights	Clients	Blockchain, DP	-	×	×	×
[100]	PHE	TTHE Paillier	Distinct	Local Gradients	Not mentioned	Differential Privacy	Forgery, Replay, Inference, Collusion	✓	✓	×
[112]	PHE	MK TTHE Paillier	Distinct	Local Model + Sample Count	CSP+Clients	DH Exchange, Data Packing	Collusion	×	×	✓
[109]	PHE	Paillier	Shared Public Key	Prediction Errors	Clients	DGK, DP, FSMMG, Newton Estimation	Inference, Replay	×	×	✓
[111]	PHE	ElGamal	Distinct	Local Gradients	Proxy Server	Proxy Re-Encryption, Blinding	Collusion	×	×	×

(a) Only protections beyond inherent guarantees of HE (confidentiality, eavesdropping resistance) are listed. (b) Info obtained from GitHub code. (c) Not explicitly mentioned, inferred since coordinator does not decrypt post-aggregation.

Table 3.6: Comparison of FL Protocols Employing HE for IoT, Highlighting Scheme Details, Security Measures, and Robustness Capabilities

3.4 Chapter Remarks

This chapter presents a systematic review of privacy-preserving techniques in FL, highlighting their strengths, limitations, and applicability in enhancing data privacy. Major approaches, such as DP, HE, and SMC, were analyzed in detail, as well as the new hybrid protocols that combine different strategies to improve both efficiency and security in FL.

The review highlights the trade-offs between privacy, computation efficiency, and model performance in the existing methods. On one hand, DP guarantees strong privacy but at the cost of model accuracy. On the other hand, HE guarantees strong data security but suffers from high computational and communication overheads. Hybrid protocols, which are the combination of two or more privacy-preserving techniques, seem to be a promising direction in leveraging the strengths of various methods to achieve a more balanced approach.

A review of current implementations of HE in FL schemes in IoT scenarios was also performed. It highlighted the most commonly used HE algorithms and their typical application contexts. The findings indicate that, while HE offers a solid foundation for privacy, its effectiveness in real-world FL deployments often depends on integrating additional techniques, from secret sharing and digital signatures to DP, each introducing its own set of trade-offs. Besides security concerns, data reduction methods also need to be considered for HE to be viable in a resource-constrained IoT environment. Overall, an optimal solution that balances both privacy and efficiency in HE-based FL schemes has yet to be achieved.

No studies applying HHE in FL were identified in the database searches, indicating that the practical impact and suitability of HHE in FL for resource-constrained IoT scenarios remains largely unexplored

Chapter 4

Proposed Solution

This chapter is dedicated to presenting the proposed solution. It begins by outlining the problem addressed in this work, followed by a detailed description of the proposed system model, including the overall architecture, the key management strategies considered, and the mitigation mechanisms developed to address potential issues arising from the chosen key configuration. Finally, it describes the proposed workflow, covering each phase of the FL process and concluding with the threat model considered.

4.1 Problem Statement

HE provides strong privacy guarantees in FL, but its computational and communication overhead can be prohibitive for resource-constrained devices, such as those in IoT environments. While some studies have applied HE in IoT FL scenarios, they often rely on complementary techniques, such as differential privacy and secret sharing, to enhance security and robustness to client dropouts, and on data packing strategies to reduce the overhead imposed by HE. Nevertheless, the challenges of applying HE in resource-constrained FL environments remain largely unresolved.

HHE has the potential to address this overhead by offloading the most complex homomorphic operations to the server. However, its feasibility, performance, and practical impact in resource-constrained FL settings are still largely unexplored.

This thesis addresses these gaps by designing and implementing a FL architecture based on HHE, focusing on reducing computational and communication overhead on resource-constrained devices while maintaining strong privacy guarantees.

4.2 Proposed System Model

This section introduces the proposed system architecture for integrating HHE into FL. It first provides a high-level overview of the proposed architecture, then examines key management approaches relevant to this implementation, and finally presents mitigation measures to address potential threats arising from the chosen key configuration.

4.2.1 Architecture Overview

The proposed solution is a centralized FL HHE scheme in which clients encrypt their data using a symmetric cipher, while the server performs the computationally expensive homomorphic operations. A centralized approach was chosen because the HESD step imposes significant computational overhead, making it impractical in a decentralized FL environment, particularly in scenarios involving mobile or low-power devices.

The architecture consists of three main entities: the *Third-Party Authority* (TPA), the *Server* and the *Clients*. The TPA is an independent trusted authority, responsible for generating and distributing all cryptographic keys. The *Server* is responsible for coordinating the FL training process, collecting the encrypted updates from *Clients*, performing secure aggregation and distributing the updated global model. At last, the *Clients* train their local models with their private data, encrypt their local updates and symmetric key, send them to the *Server*, and decrypt the received global model. A high-level overview of the architecture is shown in Figure 4.1, considering a single client for illustration purposes.

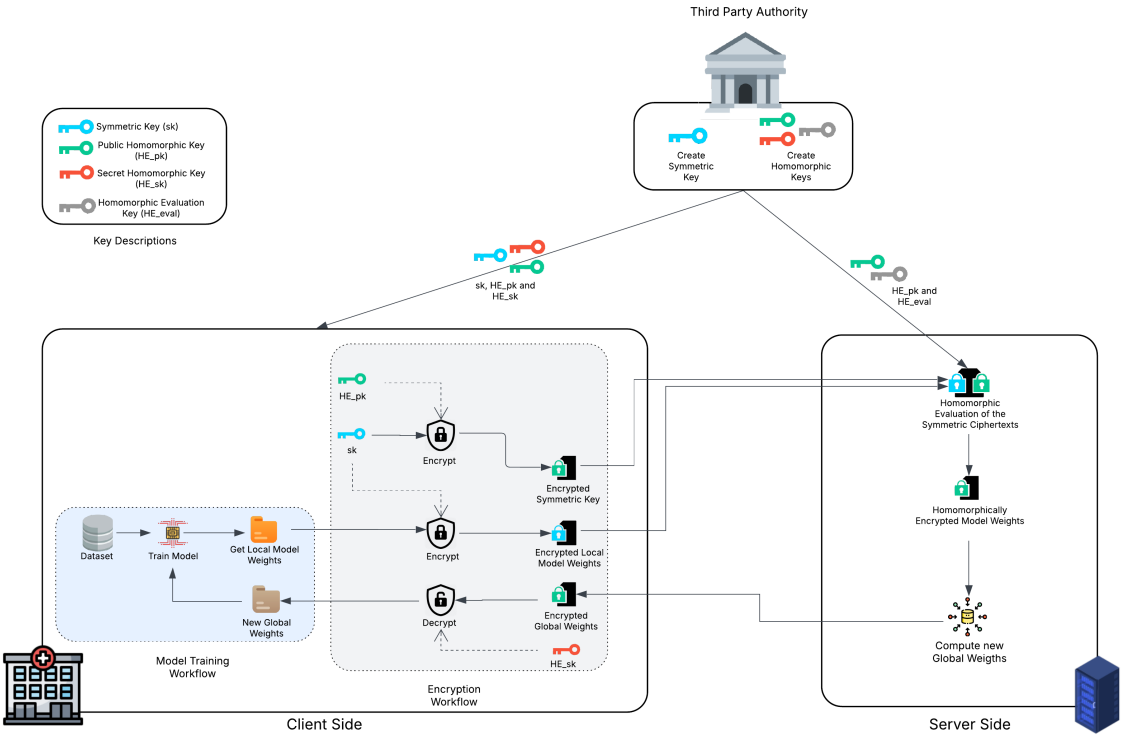

Figure 4.1: FL Architecture with HHE

A logical view of the system is presented in Figure 4.2. This view decomposes the system into key components for each entity. On the TPA side, the *Key Management Module* is responsible for generating and securely distributing cryptographic keys to both client and server. On the client side, 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*

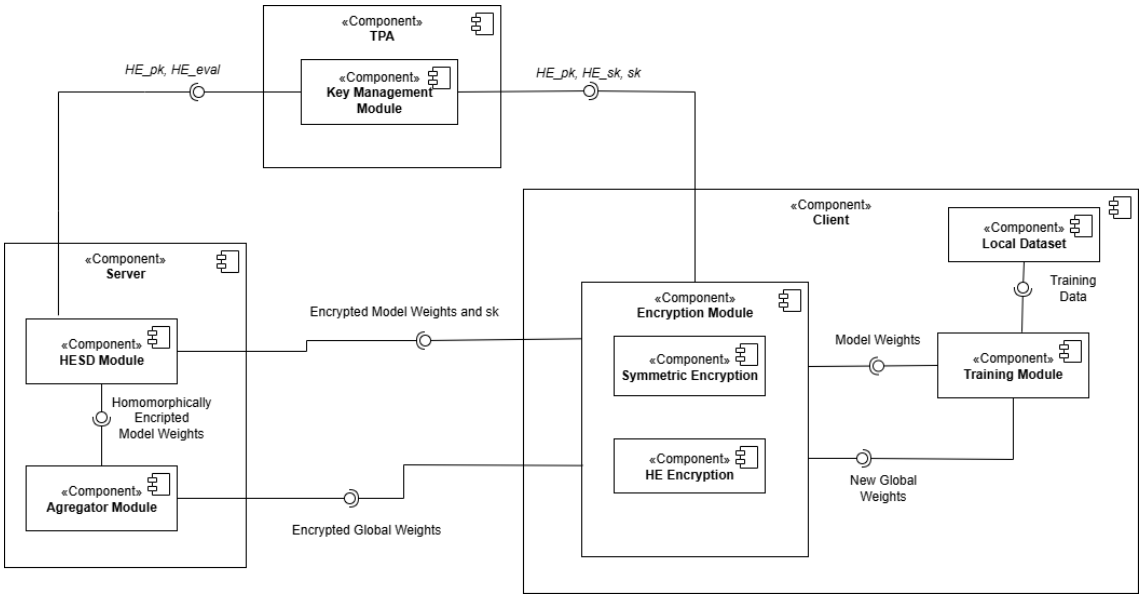

Figure 4.2: Logical View of the Proposed System

Module decrypts it, and the *Training Module* incorporates the global weights into the local model. On the server side, 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 client.

4.2.2 Key Management

As seen in Section 3.2.3, key management remains one of the main challenges of HE-based schemes, and this persists in the proposed HHE-based approach. Since the security and feasibility of the proposed solution depend heavily on how keys are managed, several configurations were carefully evaluated, taking into account the participant roles in FL defined in Section 2.1.2:

- *Shared symmetric and homomorphic keys*: In this setting, all clients share a symmetric key *sk* and the same homomorphic key pair (HE_pk, HE_sk), where HE_pk is the homomorphic public key and HE_sk is the corresponding secret key. This simplifies key management but introduces serious security risks. A malicious client eavesdropping on the network could decrypt ciphertexts sent by honest clients using the shared *sk*. Likewise, an honest-but-curious server collaborating with a client might recover either the *sk* or HE_sk, depending on the system design. Consequently, client privacy is compromised, making the system vulnerable to collusion and inference attacks.

- *Distinct symmetric keys, shared homomorphic keys*: Each client uses a unique *sk* but shares the homomorphic keys. This reduces risks on the client side during data transmission. However, if the server and clients are honest-but-curious and collude, the server may obtain HE_sk. This would allow it to decrypt the encrypted symmetric keys and ultimately recover

client model parameters. Likewise, a malicious client capable of eavesdropping on an honest client's transmission could also use HE_sk to decrypt the symmetric key and access the model parameters.

– *Shared symmetric keys, distinct homomorphic keys*: Clients share a *sk* but have different homomorphic keys, enabling a multi-key HE scheme. In this case, even if the server and the client collude, a client's HE_sk alone is insufficient to decrypt the data. However, the risks associated with the shared symmetric key remain.

– *Distinct symmetric and homomorphic keys*: Each client holds unique *sk* and homomorphic keys, eliminating the vulnerabilities related to key sharing, while preventing client-server collusion attacks, making it the most secure option.

As observed, the most secure configuration requires each client to possess distinct symmetric and homomorphic keys. However, the HHE libraries discussed in Section 2.3.1 were primarily designed with shared homomorphic keys in mind. Since the focus of this work is to evaluate the impact of HHE on client-side overhead, extensive modifications to the underlying libraries are beyond the scope of this study. Consequently, in the implemented architecture, each client maintains a distinct symmetric key to protect local updates, while a single homomorphic key pair is used between the clients and the server. This design balances practical feasibility with a reasonable level of security.

4.2.3 Mitigation Strategies

The choice of using distinct symmetric keys alongside a shared homomorphic key pair introduces a potential vulnerability in the presence of honest-but-curious and malicious clients. A client who manages to intercept another client's transmission to the server could exploit the shared homomorphic secret key HE_sk to attempt recovery of the other client's symmetric key *sk*. If successful, this would allow the attacker to decrypt the intercepted local model update, thereby compromising confidentiality. To address this weakness two strategies were designed:

1. *RSA Wrapping*: This solution is based on the RSA public-key cryptosystem. The server contains a public-private key pair and sends a certificate with its public key to all clients. Each client encrypts their homomorphic encryption of the symmetric key sk_{HE} with the server's RSA public key using a RSA encryption function $RSA.Enc$ producing a ciphertext $RSA.Enc(sk_{HE})$. Since only the server holds the corresponding RSA private key, it is the only entity capable of recovering sk_{HE}. This prevents other clients, even with access to HE_sk, from learning another client's symmetric key.

2. *Masking*: The second approach avoids public-key wrapping and instead introduces a random mask. Each clients adds a mask *M* to the *sk* before encryption producing:

$$sk_{HE} = HE.Enc(sk + M).$$

The server, knowing the masks for all clients, can remove them within the homomorphic domain by computing:

$$\begin{aligned} sk_{HE} - M &= HE.Enc(sk + M) - M \\ &= HE.Enc(sk) \end{aligned}$$

leveraging the property that HE schemes support operations between a ciphertext and a known constant. For any other client intercepting the ciphertext, the value remains meaningless without knowledge of the corresponding mask.

In addition to the original logical view, Figure 4.3 presents an updated system view that incorporates both mitigation strategies. On the TPA side, the *Key Management Module* now not only generates cryptographic keys but also creates a mask for each client, which is sent to the server to support the Masking scheme. To support RSA Wrapping, 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.

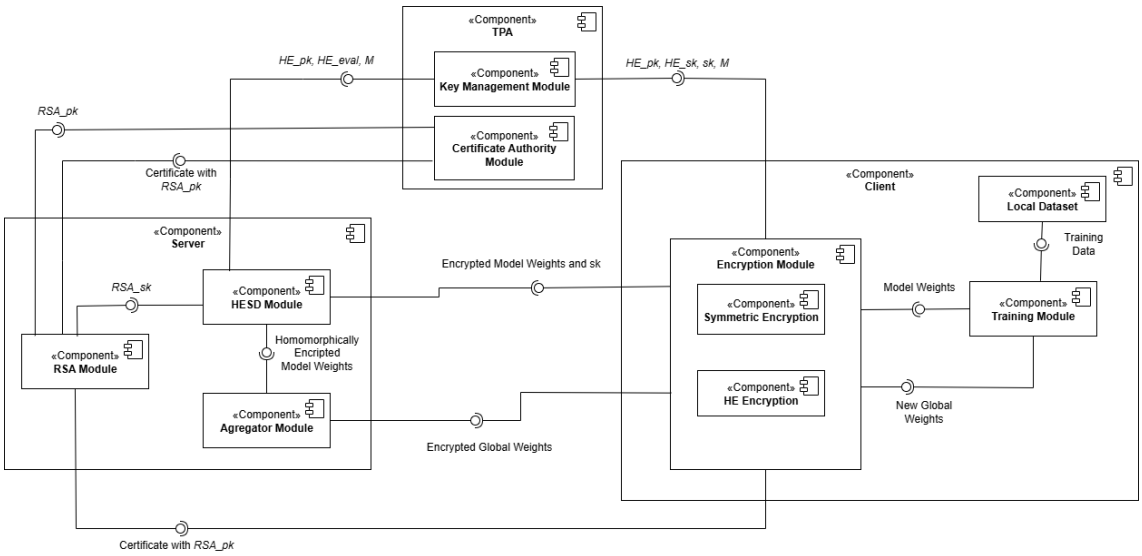

Figure 4.3: Logical View of the Proposed System with Mitigation Strategies

Both approaches strengthen confidentiality during transmission and aggregation, although through different trade-offs: *RSA Wrapping* relies on well-established public-key cryptography but introduces additional computational and communication overhead on the clients and the server, while *Masking* offers a lightweight alternative at the cost of requiring secure mask management by the server. These mitigation strategies form the foundation for the workflow described in the following section.

4.3 Workflow of the proposed solution

The workflow of the proposed solution consists of five sequential phases: *Setup*, *Client Training*, *Server Aggregation*, *Client Evaluation*, and *Server Evaluation*, which are described in detail below and illustrated in Figure 4.4. All phases, except for the *Setup phase*, are repeated for a fixed number of rounds.

4.3.1 Setup Phase

The *Setup Phase* occurs once at the beginning of the protocol. During this phase, the TPA generates and distributes the necessary cryptographic keys to both the clients and the server. This includes a single tuple 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. Additionally, the TPA generates a unique symmetric key sk_i for each client i, where $1 \leq i \leq N$ and N is the total number of clients in the system. The TPA then sends (HE_pk, HE_eval) to the server and (HE_pk, HE_sk, sk_i) to each client i. If the masking strategy is used, the TPA also generates a unique mask M_i for each client i and sends it to them. In this case, the TPA additionally provides the server with the list of masks together with the corresponding client identifiers, enabling the server to remove the correct mask from each ciphertext during aggregation. The server initializes a model and sends its weights to the clients, who use them as the starting point for local training during the training phase. In the case of RSA wrapping, the server also generates 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.

4.3.2 Client Training Phase

The *Client Training Phase* takes place immediately after the *Setup Phase* in the first round, during which clients receive the global model weights from the server in plaintext. In subsequent rounds, it follows the *Server Evaluation Phase*, where clients first decrypt the received encrypted weights using the HE secret key HE_sk. Each client then trains their local model with the received weights and encrypts the updated weights w_i with sk_i, producing the symmetric ciphertext w^i_{SKE}. In the RSA wrapping strategy, 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 $RSA.Enc(sk^i_{HE})$. In the masking strategy, the clients first add M_i to sk_i before encrypting it with HE_pk, producing sk^i_{HE}. Finally, the clients send the encrypted weights, the encrypted symmetric key, and the number of training samples, n_i, used.

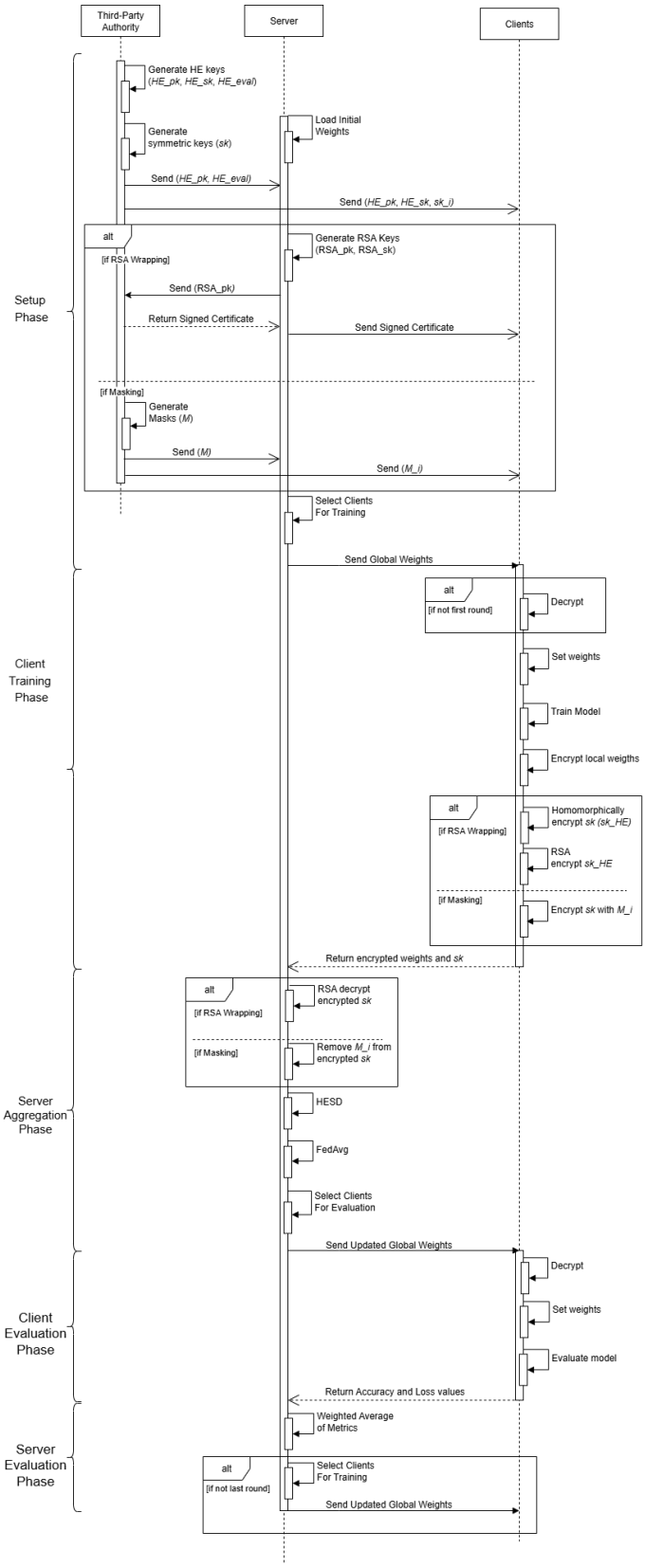

Figure 4.4: Workflow of the proposed solution

4.3.3 Server Aggregation Phase

The *Server Aggregation Phase* occurs after each *Client Training Phase* in every round of the FL process. During this phase, the server receives the encrypted weights w_{SKE} along with the corresponding encrypted symmetric keys from the selected clients. When using RSA wrapping, the server applies the RSA decryption function *RSA.Dec* with the RSA secret key *RSA_sk* on the RSA encrypted keys $RSA.Enc(sk_{HE})$ to recover the homomorphically encrypted keys sk_{HE}. In the masking strategy, the server simply subtracts the masks M_i from the corresponding clients' sk_{HE}. The server then applies HESD on each w_{SKE} using the associated sk_{HE}. Once all updates have been transformed, the server aggregates the homomorphic ciphertexts to compute the new global model weights. In the current implementation, any client that drops out during the *Client Training Phase* is excluded from aggregation for that round but may participate in subsequent rounds. Finally, the server selects a new subset of clients and sends them the updated model for evaluation.

4.3.4 Client Evaluation Phase

This phase follows the *Server Aggregation Phase* in each round of the FL process. The selected clients receive the newly aggregated global weights and decrypt them using HE_sk. They update their local models with these weights and evaluate them on their local test datasets. After evaluation, each client sends the computed accuracy and loss metrics back to the server, along with the number of test samples.

4.3.5 Server Evaluation Phase

The *Server Evaluation Phase* occurs after the *Client Evaluation Phase* and concludes each round of the federated process. Upon receiving evaluation metrics from all clients, the server aggregates these metrics using a chosen aggregation algorithm. If additional training rounds remain, the server selects a new subset of clients for the upcoming *Client Training Phase* and distributes the updated global weights accordingly.

4.4 Threat Model

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

A.1 *The setup phase is trusted*: 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*: they follow the protocol correctly but may attempt to infer private information from intercepted messages.

A.3 *The server is honest-but-curious*: it correctly executes the protocol but may attempt to infer private information from received data.

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

A.5 *External adversaries may attempt to infer information from participating clients but are unable to disrupt message transmission or inject poisoned data.*

Under these assumptions, the system provides the following security guarantees:

SP.1 Confidentiality of individual client updates during transmission.

SP.2 Confidentiality of individual client updates during aggregation on the server.

SP.3 Confidentiality of the aggregated model weights.

SP.1 is ensured by client-side encryption. Each client encrypts its local model update with its *sk*, and the *sk* is further protected via the chosen countermeasure (RSA wrapping or masking). Therefore, the transmitted data remains confidential, preventing any eavesdropper, including other clients, from recovering the local update.

Upon receiving the encrypted client updates, the server performs HESD to generate the homomorphic ciphertexts used for aggregation. Since the updates remain encrypted throughout this process, the server never has access to the plaintext of any individual client update. Furthermore, because the server does not possess the HE_sk, it cannot decrypt the resulting homomorphic ciphertexts, thereby ensuring **SP.2**.

Finally, because aggregation is performed over homomorphic ciphertexts, the resulting aggregated model weights also remain encrypted. As the server does not possess the HE_sk, it is unable to decrypt this aggregated result, thereby ensuring **SP.3**.

In summary, this chapter described the design of the proposed HHE-based FL scheme, which aims to reduce the computational and communication overhead of HE in resource-constrained scenarios like IoT devices. In this approach, clients encrypt their local model updates with a lightweight symmetric cipher, and the server aggregates them using HE after applying HESD. The system follows a centralized architecture, with a trusted third-party authority handling key generation and distribution, a server coordinating aggregation, and multiple clients performing local training. To balance security and practical constraints, each client uses a distinct symmetric key alongside a shared homomorphic key pair. Two mitigation strategies protect against risks from the shared key: RSA Wrapping, which encrypts a client's sk_{HE} under the server's RSA public key, and Masking, which introduces a client-specific random value before homomorphic encryption. The workflow unfolds across five phases: *Setup*, *Client Training*, *Server Aggregation*, *Client Evaluation*, and *Server Evaluation*. Finally, under a threat model assuming honest-but-curious clients and server with no collusion, the system ensures the confidentiality of both individual client updates and the global model during transmission and aggregation.

Chapter 5

Implementation Details

This chapter presents the implementation details of the proposed HHE-based FL scheme, with a focus on resource-constrained FL scenarios. It begins by introducing the cryptographic components, including the chosen HE-friendly symmetric cipher and the HE cipher. Next, it introduces the FL framework chosen for this work. The chapter then presents the prototype architecture followed by a detailed explanation of the server-side processing, including the HESD operations and aggregation using the FedAvg algorithm. This is succeeded by a description of the client-side processing, covering the encryption and decryption procedures. Finally, the chapter concludes with a discussion of the parameter constraints imposed by the cryptographic setup.

5.1 Cryptographic Components

In this section, the cryptographic components used to implement the HHE protocol introduced in Chapter 4 are described. Given that the implementation focuses on cross-device FL scenarios, in which clients may have limited computational resources and must encrypt their model updates efficiently, the chosen HHE scheme needs to be able to efficiently reduce client-side overhead.

Among the HE-friendly symmetric ciphers introduced in Section 2.3 (HERA, PASTA, and Elisabeth), all could, in principle, be used for this implementation. To the best of our knowledge, HERA has not been applied in any ML context and has been shown to be less efficient than both PASTA and Elisabeth [22, 76], making it a less attractive option. A recent study [113] implemented an HHE scheme for FL using Rubato [114], a HE-friendly cipher similar to HERA, together with CKKS, demonstrating the feasibility of HERA-like ciphers in FL. Nevertheless, our choice of cipher was guided by the established performance of PASTA and Elisabeth in ML scenarios prior to this work, as HERA had not been applied in such contexts.

Elisabeth was evaluated in a machine learning scenario by Cosseron et al. [76], where it was used for inference on the Fashion-MNIST dataset, achieving an accuracy of 84.18%. PASTA, similarly, was used for inference on the MIT-BIH dataset in Frimpong et al. [115] and Nguyen et al. [116] works, reaching an accuracy of 87.06% in both cases. In terms of server computation cost, Elisabeth requires, on average, 421.49 seconds per sample to perform the HESD procedure,

with each sample sized at approximately 0.0004 MB. Using optimal parameters, this time can be reduced to 71.46 seconds per sample, roughly a 6× speedup. In comparison, PASTA requires only 11.9 seconds per sample, with each sample sized at 0.0002 MB. For two PASTA samples (totalling approximately 0.0004 MB, equivalent to Elisabeth's sample size), the server would need 23.8 seconds to perform the HESD procedure. This makes PASTA roughly 18× faster than standard Elisabeth and 3× faster than Elisabeth with optimal parameters.

Based on these results, PASTA was selected as the HE-friendly symmetric cipher for this work. It was paired with BFV, which, compared to BGV, offers simpler parameter selection and more straightforward support for common operations such as encoding and noise management. Furthermore, BFV supports native plaintexts in $\mathbb{Z}_p$, which align well with PASTA's design. PASTA and BFV were integrated into the prototype using the PASTA authors' open-source framework described in Section 2.3.1.

5.2 Federated Learning Framework

To implement and evaluate the proposed FL scheme, several frameworks were considered, as summarized in Table 5.1. The comparison highlights differences in data partitioning strategies, supported deployment modes, privacy-preserving mechanisms, supported HE schemes, scalability, and cloud compatibility.

Regarding deployment and partitioning modes, all reviewed frameworks support HFL, and most also support VFL. Only Flower [25] and TensorFlowFederated [117] natively support cross-device scenarios. PySyft [118], in contrast, does not natively support cross-device scenarios, but it has been applied in such settings in the literature [119].

Every framework provides DP as a privacy-preserving mechanism, with most also support- ing SMC and HE. Only OpenFL [120] and SecretFlow [121] support the use of TEE. The available HE schemes differ differ significantly across frameworks. FATE [122], PySyft, and SecretFlow support PHE schemes, with SecretFlow offering a broader set, including Paillier and ElGamal. In terms of FHE, only Nvidia FLARE [123] provides direct support through the CKKS and BFV schemes, while Flower includes an implementation of CKKS based on the approach proposed by Catalfamo et al. [124].

Regarding scalability and cloud compatibility, Flower, Nvidia FLARE, and SecretFlow are cloud-friendly and scalable, making them suitable for experiments with a large number of clients. In terms of aggregation algorithms and machine learning libraries, Flower provides the greatest flexibility, supporting numerous aggregation strategies and a wide range of ML tools including PyTorch and TensorFlow.

The proposed solution specifically targets cross-device FL scenarios, where clients with limited resources must encrypt their model updates with minimal overhead, while a well-provisioned server performs the heavy homomorphic operations. To implement and evaluate this scheme, Flower was selected as the framework, given its high customizability, support for cross-device

scenarios, scalability, cloud compatibility, and the availability of an existing CKKS implementation [124].

Framework	Developer	Operating Systems	Supported Data Partition Modes	Supported Deployment Modes	Supported Machine Learning Tools	Aggregation Algorithms	Privacy-Preserving Mechanisms	Supported HE Schemes	Cloud Friendly	Scalability
FATE [122]	Webank	Linux, macOS, Windows	HFL, VFL	Cross-silo	TensorFlow, PyTorch	FedAvg, SSHE, SecuroBoost, Custom Made	DP, MPC, HE	Partial (Paillier, Affine, Iterative Affine)	✓	✓
Flower [25]	FlowerLab	Linux, macOS, Windows	HFL, VFL	Cross-silo, Cross-device	PyTorch, TensorFlow, HuggingFace, JAX, Pandas, fastai, PyTorch-Lightning, MXNet, scikitlearn, XGBoost	FedAVG, FedMedian, FedAvgM, FedAdam, FedYogi, FedAdagrad, FedTrimmedAvg, FedXgbBagging, FedXgbCyclic, Krum, QFedAvg, FedAvgAndroid, FedProx, FaultTolerantFedAvg, Bulyan, Custom Made	DP, MPC, HE (*) [124]	Fully (CKKS [124]*)	✓	✓
PySyft [118]	OpenMined	Linux, macOS, Windows	HFL	Cross-silo, Cross-device* [119]	PyTorch	Custom Made	DP, MPC, HE	Partial (Paillier) Fully (CKKS)	-	-
TensorFlow Federated [117]	Google	Linux, macOS	HFL	Cross-silo, Cross-device	TensorFlow	FedAvg, FedProx, Mime Lite, Custom Made	DP, MPC, HE [95]	-	✓	-
SecretFlow [125]	Ant Group	Linux, macOS, Windows	HFL, VFL	Cross-silo	PyTorch, TensorFlow	FedAvg, FedProx, FedSCR, FedSTC, Custom Made	DP, MPC, HE, TEE	Partial (Paillier, Okamoto-Uchiyama, ElGamal, Damgard-Jurik, Damgard-Geisler-Krøigaard (DGK))	✓	✓
NvidiaFlare [123]	Nvidia	Linux, macOS	HFL, VFL	Cross-silo	PyTorch, RAPIDS, Nemo, TensorFlow, sickit-learn, XGBoost, MONAI	FedAvg, FedOpt, FedProx, Scaffold, Ditto, FedSM, Fed AutoRL, Federated Horizontal/Vertical XGBoost¶ Custom Made	DP, HE	Fully (CKKS, BFV)	✓	✓
Open FL [120]	Intel	Linux, macOS, Windows	HFL (Partial VFL)	Cross-silo	Keras, TensorFlow, PyTorch	FedAvg, FedProx, FedOpt(Adam, Yogi, Adagrad), FedCurv SecAgg, Custom Made	DP, TEE	-	-	-

*: The study cited applied the indicated functionality in the framework.

Table 5.1: FL Frameworks Comparison

5.3 Prototype Architecture

With the cryptographic components and FL framework chosen, an architecture of the implemented prototype can be seen in Figure 5.1, consisting of two layers: the *Flower Layer* and the *Cryptographic Layer*.

The *Flower Layer* includes both server and client components. For communication between client and server, Flower uses gRPC. gRPC is a communication framework [1] based on the Remote Procedure Call (RPC) mechanism that allows a program on one machine to invoke functions on another machine as if they were local. In a client–server scheme, the server exposes services and waits for incoming requests, while clients call these services and process the responses [126]. Within Flower, the *SuperLink* acts as a gRPC server, managing communication and coordinating messages with clients, while the *ServerApp* is responsible for orchestrating training, aggregation, and evaluation. On the client side, the design is mirrored through the *SuperNode*, which acts as a gRPC client that connects to the server and exchanges updates, and the *ClientApp*, which handles local training and evaluation. During development, the main application logic is implemented in the *ServerApp* and *ClientApp* modules

The *Cryptographic Layer* provides the backend for most cryptographic operations in the prototype. Since the PASTA/BFV stack is implemented in C++ and Flower is a Python-based framework, both the *ServerApp* and the *ClientApp* connect to this layer through a *pybind11* [2] bridge, which exposes the necessary cryptographic functions to Python. It should be noted that, RSA operations were implemented via Python *cryptography* [3] library, operating outside the C++ cryptographic layer.

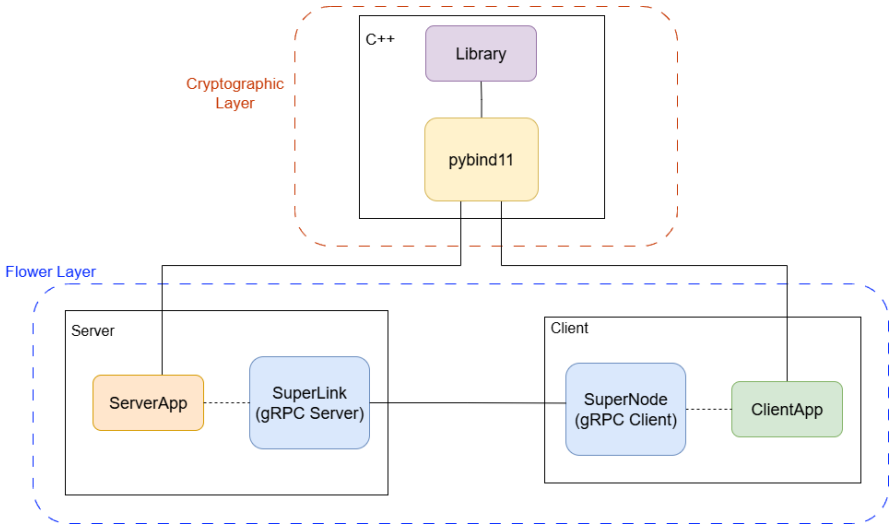

Figure 5.1: Prototype Implementation Architecture

[1] https://grpc.io/
[2] https://github.com/pybind/pybind11
[3] https://cryptography.io/en/latest/

This architecture introduced slight modifications in the workflow described in Section 4.3. Before any data can be transmitted, it must first be serialized, meaning it is converted into a byte format suitable for communication. Once received, the data is then deserialized to reconstruct its original structure. Figures 5.2 and 5.3 illustrate how this integration plays out in the *Client Training Phase* and the *Server Aggregation Phase*, respectively. Both of these workflows do not incorporate the masking and RSA wrapping methods described previously, in order to simplify the illustration of the modifications.

In Figure 5.2, during the first training round, the global model weights are transmitted in plaintext and can be deserialized using standard Python utilities. In subsequent rounds, however, the model is sent as a homomorphic ciphertext. This ciphertext cannot be processed in Python, so deserialization and decryption are delegated to the C++ backend through `pybind11`.

After local training, the client encrypts the updated weights and the symmetric key using the C++ library. The encrypted weights are returned as lists of unsigned integers, which can be serialized directly in Python. However, the symmetric key must be serialized using the C++ backend due to its incompatible internal representation.

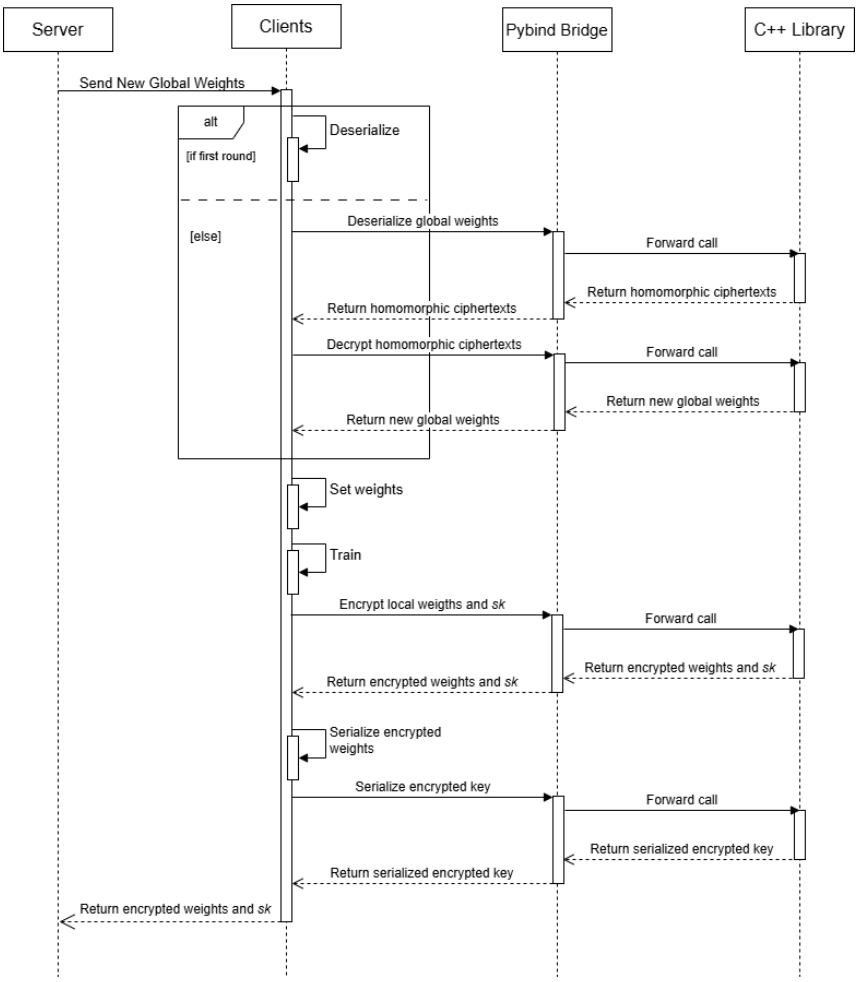

Figure 5.2: Client Training Phase Workflow in Flower

On the server side, as shown in Figure 5.3, the encrypted weights and symmetric keys are deserialized using the C++ backend. The server then performs the HESD operation and aggregates the results using the Federated Averaging (FedAvg) algorithm. The resulting global model is subsequently serialized through the C++ library and returned to the clients for the evaluation phase.

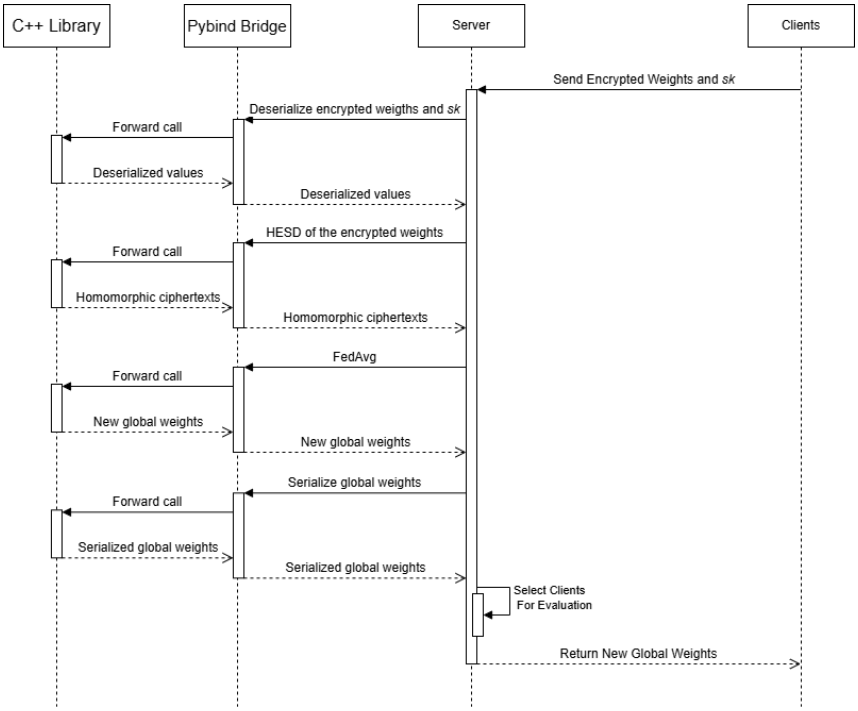

Figure 5.3: Server Aggregation Phase Workflow in Flower

5.4 Server-Side Processing

The server-side processing consists of two main steps: the HESD operations and federated averaging of the 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. When RSA wrapping is employed, the server decrypts the RSA-encrypted chunks of $RSA.Enc(sk_{HE})$ and reconstructs the original sk_{HE}. When masking is applied, the server subtracts the known mask M from the ciphertext $sk_{HE} = HE.Enc(sk+M)$ to recover $sk_{HE} = HE.Enc(sk)$.

After HESD, FedAvg multiplies each weight vector by the client's training sample count, $n_i = training\_data_i/batch\_size$, and sums the results homomorphically. Here, $training\_data_i$ denotes the number of training samples available to client i, and batch_size refers to the number of samples processed together in one forward/backward pass during local model training. Division by the global total n is deferred to the clients, because BFV does not support exact division in ciphertext space.

Algorithm 1 Chunk-Based HESD for a Single Client

Require: Client's encrypted symmetric 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}`)`
Performe 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`

5.5 Client-Side Processing

On each client device, the processing begins with quantization and encryption of the local model updates using PASTA, and ends with decryption and dequantization of the aggregated global model. Quantization reduces both memory and computational overhead by representing model weights in lower-precision data types, at the cost of a small reduction in model accuracy [127].

To encrypt model parameters with PASTA, original 32-bit floating point number (float32) values $[-\alpha, \alpha]$ are converted into int8 8-bit integer (int8) values in $[-2^{b-1}, 2^{b-1}-1]$ using scale quantization [127]. Each value $x \in w_i$ is first clipped to $[-\alpha, \alpha]$ and then scaled and rounded according to

$$quantized\_x = round\left(x * \frac{2^{b-1}}{\alpha}\right)$$

with $b = 8$ for int8 precision. Since PASTA requires unsigned 64-bit integer (uint64) inputs, the quantized int8 values are converted to uint64 before chunk-based encryption, as detailed in Algorithm 2.

In addition to encrypting model weights, each client securely prepares its symmetric key sk_i for transmission to the server. When RSA wrapping is used, the client first computes $sk_{HE}^i = HE.Enc(sk_i)$, splits it into chunks to fit RSA encryption limits, and then encrypts each chunk using the server's *RSA_pk*. Alternatively, when masking is applied, the client computes $sk_{HE}^i = HE.Enc(sk_i + M_i)$ and transmits this masked ciphertext to the server, as outlined in Algorithm 3.

Algorithm 2 Quantize and Encrypt Local Model Weights

Require: Model Weights w_i, Clipping Bound α, Chunk Size `chunk_size`, PASTA cipher instance `PASTA_cipher`
Flatten all parameters into a single vector:
    `flat_params` ← concatenate(flatten(*layer*) for *layer* in w_i)
Clip each value x in `flat_params` to $[-\alpha, \alpha]$:
    `clipped` ← clip(`flat_params`, $-\alpha, \alpha$)
Define scale factor for int8 quantization:
    `scale_factor` ← $\frac{2^{8-1}-1}{\alpha} = \frac{127}{\alpha}$
Quantize each clipped value:
    `quantized_int8` ← round(`clipped` × `scale_factor`)
Convert quantized values to unsigned 64-bit integers:
    `quantized_uint64` ← convert_to_uint64(`quantized_int8`)
Encrypt Weights:
    `list_sym_ciphertext` ← empty list
for $i = 0$ to length(`quantized_uint64`) step `chunk_size` **do**
    `chunk` ← `quantized_uint64`$[i : i +$ `chunk_size`$]$
    `encrypted_chunk` ← `PASTA_cipher.encrypt(chunk)`
    Append `encrypted_chunk` to `list_sym_ciphertext`
end for
return `list_sym_ciphertext`

Algorithm 3 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 ← $BFV\_cipher.encrypt(sk_i)$
    Split *enc_key* into chunks
        `chunked_key` ← empty list
    for $i = 0$ to length(*enc_key*) step `plaintext_size` **do**
        `chunk` ← *enc_key*$[i : i +$ `plaintext_size`$]$
        Append `chunk` to `chunked_key`
    end for
    Encrypt chunks with RSA_pk
        sk^i_{HE} ← empty list
    for each *chunk* in `chunked_key` **do**
        enc_chunk ← $RSA.Enc(chunk, RSA\_pk)$
        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}

Upon receiving the aggregated ciphertexts, each client decrypts the global weights in chunks using HE_sk. The decrypted uint64 values are converted back to signed 64-bit integers (int64), restoring negative values by subtracting the modulus from any value greater than half the modulus. Clients then apply the averaging step from FedAvg by dividing the aggregated parameters by n in float32 to preserve numerical accuracy. Finally, the results are dequantized by dividing by the quantization scale used during encryption, as shown in Algorithm 4.

Algorithm 4 Decrypt and Dequantize Global Weights

Require: Encrypted Global Weights w_{HE}, Decryption Key HE_sk, Total Number of Examples n, Clipping Bound α, Chunk Size `chunk_size`, Modulus `modulus` , HE cipher instance `bfv_cipher`

Decrypt the encrypted global weights:
  `decrypted_chunks` ← empty
for each `chunk` in w_{HE} **do**
  $c \leftarrow$ `bfv_cipher.decrypt(chunk,`HE_sk`)`
  Append c to `decrypted_chunks`
end for
Concatenate all chunks into one vector:
  `uint64_params` ← concatenate(`decrypted_chunks`)
Convert to int64:
  `int64_params` ← convert_to_int64(`uint64_params`)
Restore negative values:
for each x in `int64_params` **do**
  if $x >$ `modulus//2` **then**
    $x \leftarrow x -$ `modulus`
  end if
end for
Compute average:
  `float_params` ← `int64_params` $\div n$
Define scale factor for dequantization:
  `scale_factor` $\leftarrow \frac{2^{8-1}-1}{\alpha} = \frac{127}{\alpha}$
Dequantize new global parameters:
  `decrypted_params` ← `float_params` $\div$ `scale_factor`
return `decrypted_params`

5.6 Parameter Constraints

The HHE scheme used in this work operates in $\mathbb{Z}_q$ with $q = 2^{16}+1$. As a result, all server-side computations must remain within the range $[0, 2^{16}+1[$. This imposes a constraint on the product of the quantized model weights, the number of clients participating in training, and the number of training batches per client. Specifically, the following condition must be satisfied:

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

Given that model weights are quantized to int8 the above constraint reduces to:

$$2^8 \times 2^{x_1} \times 2^{x_2} < 2^{16}+1.$$

This means that the valid parameter combinations are those where $2^{x_1} \times 2^{x_2} \leq 2^8$. Table 5.2 lists all the valid combinations of x_1, the maximum number of training batches per client, and x_2, the maximum number of training clients, that satisfy the previous constraint.

To conclude, this chapter described the implementation details of the proposed HHE-based FL solution. PASTA was selected as the symmetric cipher due to its efficiency and strong privacy

x_1	**Max Batches per Client**	x_2	**Max Training Clients**
7	128	1	2
6	64	2	4
5	32	3	8
4	16	4	16
3	8	5	32
2	4	6	64
1	2	7	128

Table 5.2: Valid Combinations

guarantees. On the server side, the BFV cipher was used due to its compatibility with PASTA's design. The framework was implemented using Flower, a highly customizable FL framework compatible with resource-constrained device scenarios, and the prototype architecture was subsequently presented. Since the cryptographic functions were developed in C++, pybind11 was employed to bridge Python and C++ integration. The chapter also outlines the server and client-side processing steps. The server, before aggregation, needs to perform chunk-based HESD on each client's local update. On the client-side, the clients perform quantization on the model weights before encrypting them in a chunk-based manner. Finally, the parameter constraints of this work were presented, which restrict the experimental evaluation scenarios.

Chapter 6

Experimental Results

This section presents the experimental evaluation of the proposed HHE-based FL schemes using the MNIST dataset. To assess their effectiveness, seven approaches are compared: (i) traditional FL without encryption, (ii) FL with the BFV scheme, (iii) FL using a hybrid approach combining PASTA and BFV, (iv–vi) FL schemes integrating PASTA, BFV, and RSA with key sizes of 2048, 3072, and 4096 bits, respectively, and (vii) an FL scheme using PASTA, BFV and the key masking strategy.

Similar to the proposed HHE-based schemes, the BFV-based scheme applies the same homomorphic keys across all clients, encrypts the quantized local model weights, and performs division and dequantization after decryption. The evaluation considers multiple metrics throughout the training process, including accuracy, loss, communication cost, and computation cost. After training is completed, a final model with the final global weights is assessed using additional metrics such as precision, recall, and F1-score.

6.1 Experimental Setup

The experiments were run on Google Colab's T4 High-RAM environment, which provides an Intel Xeon CPU @ 2.20GHz, a NVIDIA T4 GPU, 51GB of system RAM, and 15GB of GPU memory, offering hardware acceleration for training. The second scheme, based solely on the BFV scheme, was implemented using the TenSEAL [72] library, which provides a Python interface to Microsoft SEAL and supports encrypted tensor operations.

The dataset used for this experiment was MNIST [26], a widely used benchmark for image classification tasks. It consists of 70000 grayscale images of handwritten digits (0-9), with 60000 images for training and 10000 for testing, each sized 28×28 pixels. In the context of HE, MNIST was also used by Bourse et al. [128] to evaluate discretized neural networks homomorphically. The training dataset was split into N partitions using `IidPartitioner` from Flower. This partitioner assigns samples to clients, resulting in equally sized local datasets that approximate an IID setting. However, due to the randomness of the sampling process, perfect class balance across clients is not guaranteed. Each local dataset was further split into 80% for training and 20% for local testing.

During training, an additional 20% of the local training set was reserved for validation, to monitor model performance and support early stopping during each local training round.

A convolutional neural network (CNN) based on the implementation by Gaurav Sharma [129] was employed. It contains two convolutional layers, each followed by LeakyReLU activations and batch normalization. Max-pooling layers were used after each convolution for spatial downsampling. The resulting feature maps are flattened and sent through a dropout layer before the final dense layer generates predictions for 10 classes. The model contains approximately 8000 trainable parameters. A summary of the model can be seen in Figure 6.1.

```
Model: "cnn_8k"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
leaky_relu_1 (LeakyReLU)     (None, 26, 26, 32)        0
_________________________________________________________________
batchnorm_1 (BatchNormalizat (None, 26, 26, 32)        128
_________________________________________________________________
max_pool_1 (MaxPooling2D)    (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 14)        4046
_________________________________________________________________
leaky_relu_2 (LeakyReLU)     (None, 11, 11, 14)        0
_________________________________________________________________
batchnorm_2 (BatchNormalizat (None, 11, 11, 14)        56
_________________________________________________________________
max_pool_2 (MaxPooling2D)    (None, 5, 5, 14)          0
_________________________________________________________________
flatten_layer (Flatten)      (None, 350)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 350)               0
_________________________________________________________________
dense_out (Dense)            (None, 10)                3510
=================================================================
Total params: 8,060
Trainable params: 7,968
Non-trainable params: 92
```

Figure 6.1: CNN Model Summary

6.1.1 Runtime Constraints

In addition to the theoretical bounds discussed in Section 5.6, practical runtime considerations further limited the number of configurations that could be evaluated.

Table 6.1 presents the average execution times for PASTA encryption and HESD operations, measured over 50 runs across different implementations and platforms. The Python implementation with `pybind11` performs nearly the same as the original C++ implementation, indicating that `pybind11` introduces negligible overhead. In contrast, execution on the Google Colab environment results in significantly higher runtimes, which is critical given its 24-hour runtime limit.

These differences are mainly due to Google Colab running on virtualized Intel Xeon CPUs @ 2.20 GHz, which offer significantly lower performance due to lower clock speeds and shared resources, whereas the local PC benefits from dedicated hardware and higher single-core performance. Additionally, the Google Colab results exhibit some variability between test runs, likely due to resource contention and dynamic allocation in the shared cloud environment. As a result, the estimated HESD time for 8000 parameters can vary between roughly 1400 and 1550 seconds.

Environment	Pasta Encrypt 1 Chunk (s)	Pasta Encrypt 2 Chunks (s)	HESD 1 Chunk (s)	HESD 2 Chunks (s)	8k Parameters Chunk Number	Estimated HESD 8k Parameters (s)	Estimated HESD 8k Parameters (min)
C++ (Local PC)	0.00126 ± 0.00011	0.00249 ± 0.00021	13.89690 ± 0.60661	27.58640 ± 0.32804	63	875.5047	14.5917
Python (Local PC)	0.00125 ± 0.00007	0.002460 ± 0.00007	13.52102 ± 0.50632	26.98048 ± 0.42148	63	851.8242	14.1971
Python (Google Colab)	0.00539 ± 0.00034	0.01064 ± 0.00040	24.46735 ± 0.71388	48.88514 ± 1.48935	63	1541.4431	25.6907

Table 6.1: Average execution times for Pasta encryption and HESD, and the estimated total time of HESD for 8000 parameters, across different implementations and platforms. Times represent the mean of 50 runs, reported as mean ± standard deviation. The estimated total time is derived by multiplying the number of chunks required for 8,000 parameters by the average time for HESD on a single chunk. Local PC is a Windows 10 machine with an Intel Core i7-11370H CPU and 16 GB RAM, using a Conda environment with Python 3.11 for the Python tests.

Using these estimates, the project's minimum runtime for each configuration was calculated and summarized in Table 6.2 for a varying number of rounds and clients participating in the training phase. Only configurations with estimated durations below the 24-hour limit of Google Colab were considered for experimental evaluation.

Max Training Clients	Average Time per round (mins)	Time for 5 rounds (hours)	Time for 10 rounds (hours)	Time for 15 rounds (hours)	Time for 20 rounds (hours)
2	51.38	4.28	8.56	12.85	17.13
4	102.76	8.56	17.13	25.69	34.25
8	205.53	17.13	34.25	51.38	68.51
16	411.05	34.25	68.51	102.76	137.02
32	822.10	68.51	137.02	205.53	274.03
64	1644.20	137.02	274.03	411.05	548.07
128	3288.41	274.03	548.07	822.10	1096.14

Table 6.2: Runtime Estimates Based on Number of Clients and Training Rounds. Times were estimated assuming each HESD operation takes 25.6907 minutes.

6.2 Results

This section presents the results obtained from the experiments conducted in this study. It begins with a description of the configuration used for the tests. Next, the evaluation metrics employed to assess performance, efficiency, and accuracy are outlined. Finally, the experimental outcomes are analyzed and discussed, highlighting key observations, and any notable differences between the tested setups.

6.2.1 Configuration Used

Based on the constraints described in Sections 5.6 and 6.1.1, the configuration selected for experimentation consisted of 12 clients, each with 63 training batches, where 4 were used for training and 12 for evaluation, over 10 federated rounds. Table 6.3 lists all parameters used in all the experiments. The value for α was chosen based on empirical observations of the model's typical output range across several runs.

Config. Name	**Value**
Clients	12
Rounds	10
Epochs	10
Classifier	CNN
Batch size	64
Loss function	Categorical Cross-entropy
Optimizer	Nadam
Learning Rate	0.001
Clients used per Training Phase	4
Clients used per Evaluation Phase	12
Clip range (α)	5
PASTA-3 Key size	256 bits
PASTA-3 Plaintext size	128 bits
PASTA-3 Ciphertext size	128 bits
BFV Plaintext Modulus	65537
BFV Polynomial Degree Modulus	16384
Security level	128-bit

Table 6.3: Parameters used for the federated learning experiments

Regarding the RSA schemes, multiple key sizes were benchmarked to assess their performance. The evaluation measured the encryption time of sk_{HE} and the decryption time of $RSA.Enc(sk_{HE})$. Since sk_{HE} exceeds the maximum size allowed for standard RSA key sizes, encryption was performed in chunks, therefore, the number of chunks produced and the resulting ciphertext sizes were also measured.

The results presented in Table 6.4 show that larger RSA key sizes allow encrypting bigger plaintexts, thereby reducing the total number of chunks needed and consequently speeding up the encryption process. However, this improvement in encryption time is offset by a significant increase in decryption time as the key size grows. Concerning the resulting ciphertext size, the smallest key size, 1024 bits, results in an expansion of approximately 2.6x the original plaintext, whereas 2048, 3072, and 4096-bit keys result in progressively smaller expansion factors of about 1.52x, 1.30x, and 1.22x, respectively. Due to the large expansion and lower security level, the 1024-bit key is not recommended for resource-constrained environments. The other key sizes offer a better balance between size overhead and performance, although it is important to note that only key sizes of 3072 bits or higher provide the 128-bit security level compatible with PASTA [130].

RSA Key size (bits)	Encryption (s)	Decryption (s)	Original Plaintext Size (bytes)	Max Plaintext Size (bytes)	Total Chunks to encrypt	Ciphertext Size (bytes)	Ciphertext 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

Table 6.4: Average Performance and Output Size Comparison Across RSA Key Sizes

6.2.2 Evaluation metrics

This work considers a variety of metrics to assess both the performance and efficiency of the FL system. These include standard classification metrics such as accuracy, loss, precision, recall, and F1-score, as well as system-level metrics such as computation and communication cost.

Accuracy [131] is the proportion of all classifications that were correct. Let `TP` represent true positives, `TN` true negatives, `FP` false positives and `FN` false negatives, accuracy can be defined as:

$$\texttt{Accuracy} = \frac{\texttt{correct\_classifications}}{\texttt{total\_classifications}} = \frac{\texttt{TP + TN}}{\texttt{TP + TN + FP + FN}}$$

Loss is a measure of how far the model's predictions are from the true labels. In this experiment, categorical cross-entropy loss [132] was used, which is defined as:

$$\texttt{Loss} = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

where C is the number of classes, y_i is the true label (one-hot encoded), and $\hat{y}_i$ is the predicted probability for class i.

Precision [131] measures the proportion of correctly predicted positive observations among all predicted positives:

$$\texttt{Precision} = \frac{\texttt{TP}}{\texttt{TP + FP}}$$

Recall [131] measures the proportion of actual positives that were correctly identified:

$$\texttt{Recall} = \frac{\texttt{TP}}{\texttt{TP + FN}}$$

F1-score [131] is the harmonic mean of precision and recall, providing a balance between the two:

$$\texttt{F1-score} = 2 \times \frac{\texttt{Precision} \times \texttt{Recall}}{\texttt{Precision + Recall}}$$

Communication Cost refers to the amount of data transmitted between a single client and the server during training or evaluation. This includes local model weights sent to the server, encrypted symmetric key and global model weights received from the server.

Computation Cost refers to the CPU time taken by both the client and the server to perform their respective operations during each round. On the client side, this includes serialization, weight encryption, training, evaluation, decryption, and deserialization. Additional operations such as RSA key encryption or key masking are included when applicable. On the server side, this includes deserialization, aggregation, serialization, and, when applicable, operations like HESD, RSA key decryption, and unmasking.

While accuracy and loss can be reliably computed at each round by aggregating results from individual clients using weighted averaging, this approach does not generalize well to precision, recall, and F1-score. These metrics depend not only on the number of correct predictions but also on the distribution of labels within each client's local dataset. Since FL often involves data that is non-IID across clients, averaging these metrics may lead to misleading or inconsistent results. For this reason, precision, recall, and F1-score are computed only after training is complete, using a model with final global weights evaluated on the entire MNIST test set.

6.2.3 Results

Regarding the model metrics, Figure 6.2 and Figure 6.3 show the evolution of accuracy and loss over the training rounds, respectively. As expected, the proposed hybrid schemes perform similarly to the BFV-based scheme throughout the process. Their accuracies range from 97.58% to 98.33%, closely matching the BFV scheme's 98.04%. This variation is influenced by both noise introduced during the HESD step and the selection of clients in each training phase. All encrypted schemes approach the performance of the non-encrypted baseline, which achieved 98.93% accuracy. Overall, the accuracy loss introduced by quantization remains below 1.5%, with the masking scheme showing only a 0.6% decrease. The final evaluation results in Table 6.5 confirm the efficiency of the proposed schemes, demonstrating effective model quality preservation while ensuring the protection of model updates.

	Accuracy(%)	**Loss**	**Precision(%)**	**Recall(%)**	**F1-score(%)**
FL without encryption	99.00	0.0360	99.00	99.00	99.00
BFV	98.21	0.0702	98.23	98.21	98.21
PASTA+BFV	97.39	0.0962	97.50	97.39	97.39
RSA 2048	98.26	0.0691	98.27	98.26	98.26
RSA 3072	98.04	0.0737	98.09	98.04	98.05
RSA 4098	97.28	0.1152	97.38	97.28	97.28
Masking	98.35	0.0622	98.37	98.35	98.35

Table 6.5: Global Model Comparisons

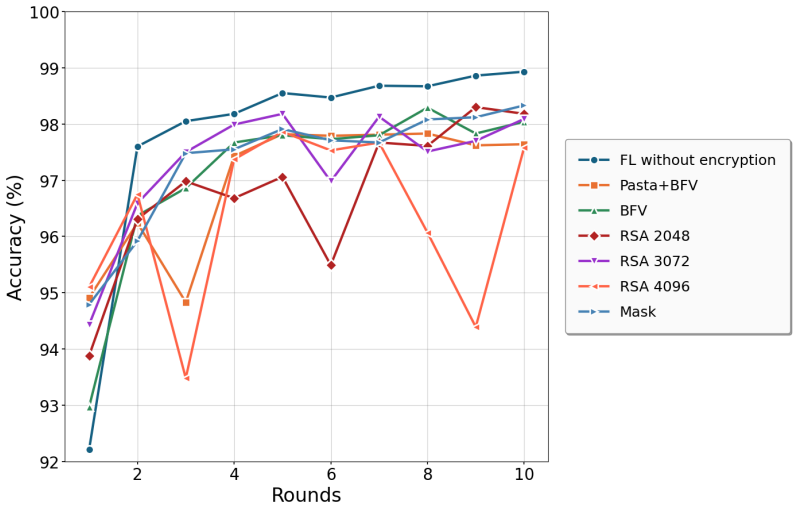

Figure 6.2: Accuracy of the FL schemes over the rounds

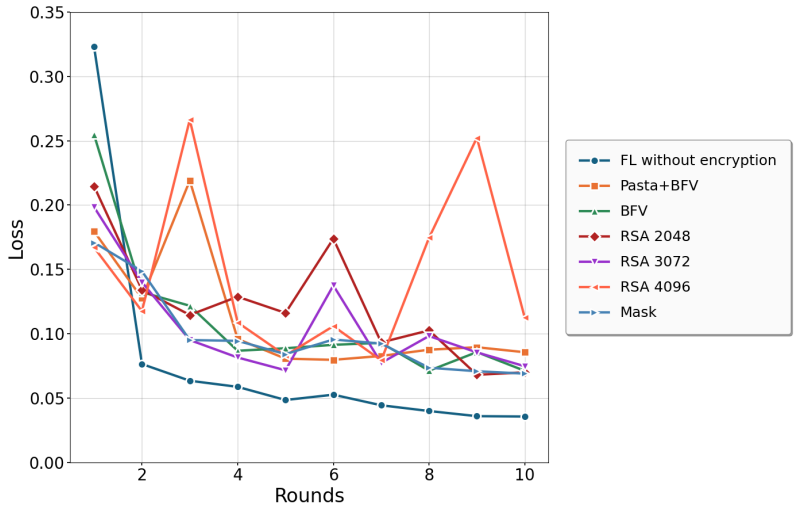

Figure 6.3: Loss of the FL schemes over the rounds

While model performance is an essential aspect, it is not the only factor to consider when evaluating the practicality of a privacy-preserving FL scheme. Communication cost, in particular, plays a crucial role in real-world deployments. Table 6.6 compares client and server communication costs across schemes. On the client side, PASTA-based schemes reduce the encrypted weight size by 2077× but require an additional encrypted key. The PASTA+BFV and Masking schemes share a similar encrypted key size since masking is applied before encryption. The use of RSA increases the total data sent by 1.51× with a 2048-bit key and by only 1.22× with a 4096-bit key. Overall, hybrid schemes reduce client-side communication overhead by up to 61.31× compared to BFV alone. On the server side, all schemes incur identical overhead, as they all rely on BFV. Taken together, hybrid schemes achieve nearly half the total communication cost of a BFV-only approach. These results highlight the effectiveness of using a homomorphic-friendly symmetric cipher to substantially reduce client-side communication cost.

Scheme	**Client**			**Server**
	Encrypted Weights Size (bytes)	Encrypted Key Size (bytes)	Total Size (MB)	Aggregated Weights Size (MB)
BFV	115055367	-	109.74	109.74
PASTA+BFV	55383	1826183	1.79	109.74
RSA 2048	55383	2777579	2.70	109.74
RSA 3072	55383	2394831	2.34	109.74
RSA 4096	55383	2231775	2.18	109.74
Masking	55383	1826422	1.79	109.74

Table 6.6: Comparison of Client and Server Communication Cost between schemes

Computation cost is another important metric when evaluating the practicality of a FL scheme. The client-side results, shown in Table 6.7, demonstrate that using PASTA significantly improves encryption and decryption times by approximately 9.7× and 6.9×, respectively, compared to BFV. Training remains the most time-consuming operation at around 10.7s. Note that training and evaluation times are identical across all schemes since these operations are performed in plaintext and are not the main focus of comparison. Serialization in the PASTA+BFV scheme is notably faster than the BFV scheme (0.0013s versus 0.0814s), due to the reduced ciphertext size introduced by PASTA. In contrast, deserialization is slower (0.425s versus 0.083s), despite receiving the same amount of data, mainly because of inefficient serialization of homomorphic ciphertexts on the server, which complicates client-side deserialization. For the RSA-based schemes, the additional RSA key encryption step adds an overhead ranging from 0.35s to 0.44s, slightly increasing total runtime compared to the PASTA+BFV scheme but still averaging around 12.4s (a 1.04× increase). Similarly, the Masking scheme introduces a masking step of about 0.018s, which is negligible as it can be performed during client initialization and is excluded from the reported runtimes. Overall, hybrid schemes achieve total client runtimes between 11.88s and 12.45s, making them approximately 1.36× to 1.43× faster than the BFV-only scheme (16.95s) while maintaining comparable model performance. Minor variations among hybrid schemes in encryption, decryption, and deserialization times are primarily due to runtime variability on Google Colab, as all schemes process the same data volume in these steps.

Operation	**BFV**	**PASTA+BFV**	**RSA 2048**	**RSA 3072**	**RSA 4096**	**Masking**
Deserialization (s)	0.08270	0.42480	0.47379	0.48112	0.47115	0.44298
Decryption (s)	2.90076	0.42206	0.48014	0.49451	0.48567	0.46385
Training (s)	10.70805	10.70805	10.70805	10.70805	10.70805	10.70805
Evaluation (s)	1.07516	1.07516	1.07516	1.07516	1.07516	1.07516
Weights Encryption (s)	3.18009	0.32867	0.34869	0.33840447	0.33226	0.33888
RSA Key Encryption (s)	-	-	0.43752	0.35177	0.38373	-
Key Masking[a] (s)	-	-	-	-	-	0.01845
Serialization (s)	0.08138	0.00128	0.00249	0.00240	0.00212	0.00123
Training runtime (s)	16.95298	11.88486	12.45068	12.37625	12.38298	11.95498
Evaluation runtime (s)	4.140	1.92330	2.03158	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.

Table 6.7: Average Client Computation Cost

Although the client-side computation costs differ by only about 1.4×, the server-side results show a much larger discrepancy. As shown in Table 6.8, the hybrid schemes are faster at deserialization but slower during aggregation and serialization. This may be due to these functions being partially implemented in C++, which can introduce delays between operations, though nothing that negatively impacts the overall process. The main bottleneck lies in the HESD operation, which took an average of 1446.48s (24.1 minutes) per client with 8000 parameters. This represents an increase of roughly 6700x compared to BFV, and with four clients, a PASTA+BFV server becomes approximately 26727x slower than a BFV-only server for a single aggregation phase. Additionally, RSA key decryption adds between 7.9s for a 2048-bit key and 19.3s for a 4096-bit key. In contrast, unmasking a key takes only about 0.0014s, 13786x faster than decrypting with a 4096-bit RSA key. While RSA decryption overhead is non-negligible, it becomes insignificant when compared to the HESD operation.

Overall, these results show that HHE significantly reduces both communication and computation overhead on the client side, suggesting that HHE is a preferable choice over pure HE in resource-constrained environments like IoT. However, due to increase in the computational overhead of the server, the application of HHE in a decentralized FL setting is only practical when clients have substantial computational resources and stable network connections.

Operation	**BFV**	**PASTA+ BFV**	**RSA 2048**	**RSA 3072**	**RSA 4096**	**Masking**
Deserialization(s)	0.0584	0.0009	0.0009	0.0009	0.0008	0.0010
HESD per client(s)	-	1451.0	1510.1	1431.5	1439.8	1400.0
RSA Key Decrypt. per client(s)	-	-	7.9	12.0	19.3	-
Unmasking per client(s)	-	-	-	-	-	0.0014
Aggregation(s)	0.092898	0.2174	0.2355	0.2222	0.2244	0.2212
Serialization(s)	0.065235	0.3629	0.3827	0.3626	0.3620	0.3528
Total runtime(s)	0.2165	5804.5803	6072.6191	5774.5857	5836.9872	5600.5806

Table 6.8: Average Server Aggregation Phase Computation Cost Comparison

To better understand how the computation cost scales with model size, Fig. 6.4 shows the time required to perform HESD for different numbers of parameters. As observed, the execution time grows linearly with the number of parameters. This behavior is due to the chunk-based processing of the encrypted parameters, where each chunk ends up being padded if it does not have enough size. Let T_{chunk} be the time to process a single chunk, and N_{chunks} be the total number of chunks needed to represent all parameters, the total HESD time needed can be estimated as:

$$T_{\text{HESD}} = T_{\text{chunk}} \times N_{\text{chunks}}$$

Given this, even well-known small CNN models like LeNet-5 [133], which contains approximately 60000 weights, would result in a server-side overhead of roughly 11490 seconds (3.2 hours) per client participating in the training phase. This estimate is based on the average time to perform HESD on a single chunk, shown in Table 6.1, and the number of chunks required to represent the model's weights.

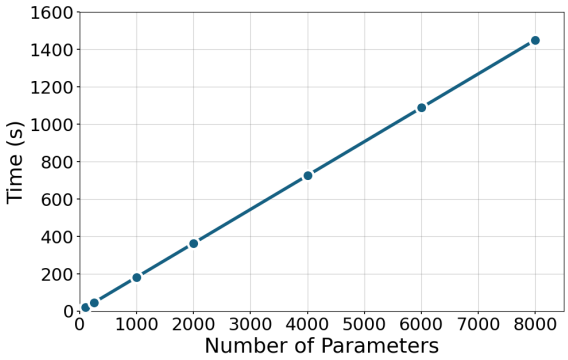

Figure 6.4: Runtime Overhead of HESD with Increasing Parameters

An additional observation during experimentation was the occurrence of memory-related failures in the BFV-based scheme. As seen in Table 6.9, starting from round 3, the BFV-based scheme had clients dropping out due to an out-of-memory (OOM) error during the client evaluation phase. These were triggered by Flower's underlying task management system, Ray, which automatically terminates processes when node memory usage exceeds a predefined threshold. In this case, memory usage reached approximately 48.5GB out of 51GB (95%), causing Ray to shut down the most recently scheduled client processes. In contrast, the hybrid schemes completed all rounds without any memory-related issues, demonstrating that the HHE schemes are significantly more memory efficient compared to a pure HE scheme.

Scheme	**Rounds with Dropout**	**Successful Clients (Avg)**	**Cause of Failure**
Pasta+BFV based	0 / 10	12 / 12	None
BFV	8 / 10	7 / 12	OOM in Flower

Table 6.9: Evaluation reliability comparison

In summary, this chapter presented a comprehensive experimental evaluation of the proposed HHE-based FL schemes on the MNIST dataset, comparing them against plaintext, pure BFV, and variant mitigation strategies. The results demonstrated that the hybrid schemes effectively balanced privacy, efficiency, and model performance, with minimal accuracy loss compared to the plaintext baseline. The use of the PASTA cipher significantly reduced client-side communication and computational overhead, making the approach practical for resource-constrained clients. Meanwhile, server-side computations, particularly the HESD operation, remained a substantial bottleneck. The evaluation also characterized the performance trade-offs of the two mitigation strategies, with RSA key sizes impacting encryption/decryption times and ciphertext expansion, while the Masking strategy proved to be computationally negligible. Overall, the findings validate HHE as a highly effective solution for resource-constrained clients in FL, while also clearly identifying the substantial server-side computational burden as the primary challenge for future work.

Chapter 7

Conclusions and Future Work

This final chapter presents the conclusions drawn from the developed work. It reflects on the achieved objectives, discusses the strengths and limitations of the proposed approach, and highlights the key contributions of this thesis. Additionally, this chapter discusses potential directions for future research and improvements that could improve the solution but were beyond the scope of this project.

7.1 Work Overview

This thesis explored the application of hybrid homomorphic encryption in FL to enhance privacy and efficiency, focusing on resource-constrained IoT devices. The primary objective was to design and implement an HHE-based FL scheme that reduces computational and communication overhead on client devices while maintaining robust security guarantees.

The work begins with a survey of the main privacy-preserving methods used in FL, highlighting the strengths and limitations of techniques such as Differential Privacy, Secure Multi-Party Computation, Homomorphic Encryption, Secret Sharing, Blockchain, and Trusted Execution Environments. It also emphasizes the emergence of hybrid protocols, which combine two or more privacy-preserving methods to strengthen security guarantees; for example, integrating DP with HE to prevent leakage of local gradients and to protect shared models in adversarial settings.

Subsequently, a focused investigation on the application of HE in FL within IoT scenarios was conducted to understand current practices. This study revealed that HE can be applied in diverse ways depending on the considered threat model. To defend against collision attacks, some works utilize multi-key HE schemes [96] or secret sharing schemes [99]. Cosine similarity has also been employed to filter out irrelevant updates [104]. Additionally, data compression techniques such as gradient compression [98] are applied to reduce communication costs associated with HE. However, none of the surveyed studies implemented HHE in FL, leaving this promising approach unexplored.

The main contribution of this thesis is the development of an HHE scheme for FL tailored to resource-constrained environments. By combining the PASTA symmetric cipher with the BFV

fully homomorphic encryption scheme, a centralized FL protocol was devised to reduce computation and communication overhead on the client side. Since PASTA is a single-key cipher, the homomorphic key pair is identical for all clients, which introduces a vulnerability if malicious clients attempt to eavesdrop on honest clients' transmissions. To address this issue, two mitigation techniques were proposed. The first employs RSA public-key encryption to encrypt the homomorphic encryption of each client's symmetric key with the server's public key. The second uses masking, where a mask is added to the client's symmetric key prior to encryption, the server subsequently removes the mask homomorphically.

The developed schemes were experimentally compared against a baseline FL scheme and a pure BFV-based scheme. The experiment involved 12 clients training on the MNIST dataset over 10 rounds, with 4 clients participating in training and all 12 used for evaluation. Results demonstrated that the quantization method utilized reduced accuracy by at most 1.5%, indicating that HHE maintains effective model performance. Regarding communication costs, the PASTA cipher reduced encrypted weight sizes by about 2000x compared to the BFV scheme. Although adding an encrypted key increased the total data transmitted by clients from 1.79 MB to between 2.70 MB (depending on RSA key size), the hybrid HHE scheme still reduced total communication overhead by up to 61.31x relative to the BFV scheme. Server-side costs remained comparable to the BFV baseline, as all schemes use the BFV cipher. Computationally, the HHE schemes achieved a 9.7x speedup in encryption and a 6.9x speedup in decryption compared to pure BFV, making them, overall, 1.43x faster. The RSA encryption process imposed a minor slowdown of only about 1.04x compared to a pure HHE-scheme, while the masking process incurred negligible overhead. However, on the server side, the hybrid schemes must perform an homomorphic evaluation of the symmetric decryption function over symmetric ciphertexts to obtain homomorphic ciphertexts for aggregation. This introduced an overhead of approximately 1450 seconds per client during training, increasing server computation cost by roughly 6700x per client compared to the BFV-only scheme. Finally, the experiments demonstrated that the HHE schemes are significantly more memory-efficient compared to the BFV scheme.

Overall, this study demonstrates the benefits of employing HHE for resource-constrained clients, while also highlighting significant computational challenges on the server side that must be addressed for practical deployment.

7.2 Challenges and Limitations

Although the proposed scheme met its objectives, several challenges emerged, both from the HHE design itself and from practical constraints imposed by the experimental environment.

The main limitation of an HHE scheme lies in its security assumptions. The homomorphic-friendly ciphers reviewed in this work were not designed with multi-key HE in mind, which introduces vulnerabilities in settings involving malicious clients that are able to eavesdrop on honest clients' transmissions. To solve this, this study used the RSA public-key scheme and masking but

future work could explore the development of homomorphic-friendly ciphers that are compatible with multi-key HE schemes.

A practical constraint specific to this work arose from the use of Google Colab as the experimental environment. Google Colab offers limited hardware resources, especially in CPU performance, and enforces strict runtime and memory constraints that limit the scale and complexity of the experiments. As a result, it was not possible to evaluate the scheme in more realistic scenarios involving dozens or hundreds of clients during training. This constraint also influenced the choice of the machine learning model. Due to the high computational cost of the HESD step, a smaller model had to be used to ensure compatibility with Colab's resource limits.

Finally, limitations also emerged from the choice of the aggregation method. The use of FedAvg requires division by a scalar to compute the weighted average of model updates. However, since the BFV scheme does not support division or multiplication by real numbers, this operation had to be performed by the clients. As a result, the server was required to send the global scalar to all clients, introducing a potential privacy limitation, especially in scenarios where clients prefer not to disclose their local data volume. This issue could be addressed by using a scheme like CKKS, which supports approximate arithmetic over real numbers.

7.3 Future Work

Even though this study already provides a baseline of the improvements of HHE compared to HE on the client side, several areas remain open for improvement.

As mentioned previously, a primary direction for improvement involves the development of an efficient homomorphic-friendly cipher compatible with a multi-key scheme. This would help mitigate collusion attacks between clients and server, as demonstrated by Ma et al. [96].

Additionally, the current protocol does not address the issue of client dropouts during training rounds. Future work could integrate dropout-resilient strategies, such as secret sharing, to allow clients to rejoin or to enable the server to reconstruct their contributions for aggregation [101]. Furthermore, incorporating techniques like cosine similarity [104] could help identify and filter out tampered or irrelevant model updates before aggregation. These approaches also highlight the importance of defending the system against poisoning attacks from malicious participants.

Authentication mechanisms, such as TLS [103], could also be incorporated to verify the identities of participating entities. This could be easily solved, as the Flower framework already provides built-in support for TLS in a project, however it was not compatible with the Google Colab environment. In addition, the protocol could be extended to allow clients to verify whether the received model genuinely reflects the aggregated updates.

Finally, to alleviate the server's computational burden, future work could investigate parallelization strategies. Specifically, parallelization may be applied at two levels during the aggregation phase: concurrently processing multiple clients, and parallel processing of ciphertext chunks within each client's data.

References

[1] Shiza Malik, Khalid Muhammad, and Yasir Waheed. Artificial intelligence and industrial applications-a revolution in modern industries. *Ain Shams Engineering Journal*, 15(9):102886, 2024.

[2] Shuroug A. Alowais, Sahar S. Alghamdi, Nada Alsuhebany, Tariq Alqahtani, Abdulrahman I. Alshaya, Sumaya N. Almohareb, Atheer Aldairem, Mohammed Alrashed, Khalid Bin Saleh, Hisham A. Badreldin, Majed S. Al Yami, Shmeylan Al Harbi, and Abdulkareem M. Albekairy. Revolutionizing healthcare: The role of artificial intelligence in clinical practice. *BMC Medical Education*, 23(1):689, September 2023.

[3] Xia Hu, Lingyang Chu, Jian Pei, Weiqing Liu, and Jiang Bian. Model complexity of deep learning: A survey. arXiv preprint arXiv:2103.05127, 2021.

[4] Alexandre Bailly, Corentin Blanc, Élie Francis, Thierry Guillotin, Fadi Jamal, Béchara Wakim, and Pascal Roy. Effects of dataset size and interactions on the prediction performance of logistic regression and deep learning models. *Computer Methods and Programs in Biomedicine*, 213:106504, 2022.

[5] Zhiqiang Gong, Ping Zhong, and Weidong Hu. Diversity in machine learning. *IEEE Access*, 7:64323–64350, 2019.

[6] Wei Cao, Wenting Shen, Zhixiang Zhang, and Jing Qin. Privacy-preserving healthcare monitoring for iot devices under edge computing. *Computers and Security*, 134:103464, 2023.

[7] European Parliament and Council. Regulation (EU) 2016/679 (General Data Protection Regulation), Article 6 – Lawfulness of processing. Official Journal of the European Union, L119, 4 May 2016, pp. 1–88, 2016. Art. 6.

[8] S. Liu and L. Guo. Data ownership in the ai-powered integrative health care landscape. *JMIR Medical Informatics*, 12:e57754, 2024.

[9] Yunke Wang, Yanxi Li, and Chang Xu. Position: Ai scaling: From up to down and out. arXiv preprint arXiv:2502.01677, 2025.

[10] Bapi Chatterjee. Distributed machine learning. In *Proceedings of the 25th International Conference on Distributed Computing and Networking*, ICDCN '24, page 4–7, New York, NY, USA, 2024. Association for Computing Machinery.

[11] Ji Liu, Jizhou Huang, Yang Zhou, Xuhong Li, Shilei Ji, Haoyi Xiong, and Dejing Dou. From distributed machine learning to federated learning: a survey. *Knowledge and Information Systems*, 64(4):885–917, March 2022.

[12] Mohammed Aledhari, Rehma Razzak, Reza M. Parizi, and Fahad Saeed. Federated Learning: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Access*, 8:140699–140725, 2020.

[13] Lingjuan Lyu, Han Yu, and Qiang Yang. Threats to federated learning: A survey. arXiv preprint arXiv:2003.02133, March 2020.

[14] Pengrui Liu, Xiangrui Xu, and Wei Wang. Threats, attacks and defenses to federated learning: Issues, taxonomy and perspectives. *Cybersecurity*, 5(1):4, February 2022.

[15] Oana Stan, Vincent Thouvenot, Aymen Boudguiga, Katarzyna Kapusta, Martin Zuber, and Renaud Sirdey. A Secure Federated Learning: analysis of different cryptographic tools. In Sabrina De Capitani di Vimercati and Pierangela Samarati, editors, *SECRYPT 2022 - 19th International Conference on Security and Cryptography*, volume 1, pages 669–674, Lisbonne, Portugal, July 2022.

[16] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Processing Magazine*, 37(3):50–60, May 2020.

[17] Lingjuan Lyu, Han Yu, Xingjun Ma, Chen Chen, Lichao Sun, Jun Zhao, Qiang Yang, and Philip S. Yu. Privacy and Robustness in Federated Learning: Attacks and Defenses. *IEEE Transactions on Neural Networks and Learning Systems*, 35(7):8726–8746, July 2024.

[18] Sai Sri Sathya, Praneeth Vepakomma, Ramesh Raskar, Ranjan Ramachandra, and Santanu Bhattacharya. A review of homomorphic encryption libraries for secure computation. arXiv preprint arXiv:1812.02428, 2018.

[19] Joshua Zhao, Saurabh Bagchi, Salman Avestimehr, Kevin Chan, Somali Chaterji, Dimitris Dimitriadis, Jiacheng Li, Ninghui Li, Arash Nourian, and Holger Roth. The federation strikes back: A survey of federated learning privacy attacks, defenses, applications, and policy landscape. *ACM Comput. Surv.*, 57(9), April 2025.

[20] Chiara Marcolla, Victor Sucasas, Marc Manzano, Riccardo Bassoli, Frank H. P. Fitzek, and Najwa Aaraj. Survey on fully homomorphic encryption, theory, and applications. *Proceedings of the IEEE*, 110(10):1572–1609, 2022.

[21] Khoa Nguyen, Mindaugas Budzys, Eugene Frimpong, Tanveer Khan, and Antonis Michalas. A pervasive, efficient and private future: Realizing privacy-preserving machine learning through hybrid homomorphic encryption. arXiv preprint arXiv:2409.06422, 2024.

[22] Christoph Dobraunig, Lorenzo Grassi, Lukas Helminger, Christian Rechberger, Markus Schofnegger, and Roman Walch. Pasta: A case for hybrid homomorphic encryption. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(3):30–73, Jun. 2023.

[23] Abhishek Vyas, Po-Ching Lin, Ren-Hung Hwang, and Meenakshi Tripathi. Privacy-preserving federated learning for intrusion detection in iot environments: A survey. *IEEE Access*, 12:127018–127050, 2024.

[24] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012.

[25] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, and Nicholas D. Lane. Flower: A Friendly Federated Learning Research Framework. arXiv preprint arXiv:2007.14390, March 2022.

[26] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[27] Pedro Correia, Ivan Silva, Ivone Amorim, Eva Maia, and Isabel Praça. Federated learning: An approach with hybrid homomorphic encryption. arXiv preprint arXiv:2509.03427, 2025.

[28] Qiongxiu Li, Wenrui Yu, Yufei Xia, and Jun Pang. From Centralized to Decentralized Federated Learning: Theoretical Insights, Privacy Preservation, and Robustness Challenges. arXiv preprint arXiv:2503.07505, March 2025.

[29] Mónica Ribero and Haris Vikalo. Reducing communication in federated learning via efficient client sampling. *Pattern Recognition*, 148:110122, 2024.

[30] Mohammad Moshawrab, Mehdi Adda, Abdenour Bouzouane, Hussein Ibrahim, and Ali Raad. Reviewing federated learning aggregation algorithms; strategies, contributions, limitations and future perspectives. *Electronics*, 12(10), 2023.

[31] Yunbo Li, Jiaping Gui, and Yue Wu. An experimental study of different aggregation schemes in semi-asynchronous federated learning. arXiv preprint arXiv:2405.16086, 2024.

[32] Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400 – 407, 1951.

[33] H. Brendan McMahan, Moore E, Ramage D, Hampson S., and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. arXiv preprint arXiv:1602.05629, 2023.

[34] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. arXiv preprint arXiv:1909.06335, 2019.

[35] Bingyan Liu, Nuoyan Lv, Yuanchun Guo, and Yawen Li. Recent advances on federated learning: A systematic survey. *Neurocomputing*, 597:128019, September 2024.

[36] Priyanka Mary Mammen. Federated learning: Opportunities and challenges. arXiv preprint arXiv:2101.05428, 2021.

[37] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. arXiv preprint arXiv:1811.03604, 2019.

[38] Daniel M. Jimenez G, David Solans, Mikko Heikkila, Andrea Vitaletti, Nicolas Kourtellis, Aris Anagnostopoulos, and Ioannis Chatzigiannakis. Non-IID data in Federated Learning: A Survey with Taxonomy, Metrics, Methods, Frameworks and Future Directions. arXiv preprint arXiv:2411.12377, December 2024.

[39] Syreen Banabilah, Moayad Aloqaily, Eitaa Alsayed, Nida Malik, and Yaser Jararweh. Federated learning review: Fundamentals, enabling technologies, and future applications. *Information Processing & Management*, 59(6):103061, November 2022.

[40] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications, January 2019.

[41] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. A survey on federated learning. *Knowledge-Based Systems*, 216:106775, 2021.

[42] Peter Kairouz, H. Brendan McMahan, and et al. Advances and Open Problems in Federated Learning. arXiv preprint arXiv:1912.04977, 2021.

[43] Umer Majeed, Sheikh Salman Hassan, and Choong Seon Hong. Cross-Silo Model-Based Secure Federated Transfer Learning for Flow-Based Traffic Classification. In *2021 International Conference on Information Networking (ICOIN)*, pages 588–593, Jeju Island, Korea (South), January 2021. IEEE.

[44] Chao Huang, Jianwei Huang, and Xin Liu. Cross-silo federated learning: Challenges and opportunities. arXiv preprint arXiv:2206.12949, 2022.

[45] Mohammad Malekzadeh, Anastasia Borovykh, and Deniz Gündüz. Honest-but-Curious Nets: Sensitive Attributes of Private Inputs Can Be Secretly Coded into the Classifiers' Outputs. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 825–844, November 2021.

[46] Geming Xia, Jian Chen, Chaodong Yu, and Jun Ma. Poisoning Attacks in Federated Learning: A Survey. *IEEE Access*, 11:10708–10722, 2023.

[47] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting Unintended Feature Leakage in Collaborative Learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 691–706, San Francisco, CA, USA, May 2019. IEEE.

[48] Ronald L. Rivest, Leonard Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–180, 1978.

[49] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[50] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[51] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, page 223–238, Berlin, Heidelberg, 1999. Springer-Verlag.

[52] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[53] Michael R. Fellows and Neal Koblitz. Combinatorial cryptosystems galore! In *Contemporary Mathematics*, volume 168, pages 51–72. American Mathematical Society, 1994.

[54] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. A Survey on Homomorphic Encryption Schemes: Theory and Implementation. *ACM Computing Surveys*, 51(4):1–35, July 2019.

[55] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Proceedings of the Second Theory of Cryptography Conference (TCC 2005)*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 2005.

[56] Kristian Gjøsteen. *Subgroup Membership Problems and Public Key Cryptosystems*. PhD thesis, Norwegian University of Science and Technology, 2004.

[57] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery.

[58] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory*, 6(3), July 2014.

[59] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437, Cham, 2017. Springer International Publishing.

[60] Pedro Barbosa, Ivone Amorim, Eva Maia, and Isabel Praça. ENNigma: A framework for Private Neural Networks. *Future Generation Computer Systems*, 166:107719, May 2025.

[61] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *J. Cryptology*, 33(1):34–91, January 2020.

[62] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. Multikey Fully Homomorphic Encryption and Applications. *SIAM Journal on Computing*, 46(6):1827–1892, January 2017.

[63] Elnaz Rabieinejad, Abbas Yazdinejad, Ali Dehghantanha, and Gautam Srivastava. 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, February 2024.

[64] Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL, January 2023. Microsoft Research, Redmond, WA.

[65] Helib. https://github.com/homenc/HElib, 2013.

[66] N. P. Smart and F. Vercauteren. Fully homomorphic SIMD operations. Cryptology ePrint Archive, Paper 2011/133, 2011.

[67] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, 2012.

[68] Jung Hee Cheon et al. Heaan. https://github.com/snucrypto/HEAAN, 2016.

[69] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption library, August 2016. https://tfhe.github.io/tfhe/.

[70] Zama. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data, 2022. https://github.com/zama-ai/tfhe-rs.

[71] Ahmad Al Badawi, Andreea Alexandru, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Carlo Pascoe, Yuriy Polyakov, Ian Quah, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. OpenFHE: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915, 2022. https://eprint.iacr.org/2022/915.

[72] Ayoub Benaissa, Bilal Retiat, Bogdan Cebere, and Alaa Eddine Belfedhal. TenSEAL: A Library for Encrypted Tensor Operations Using Homomorphic Encryption. arXiv preprint arXiv:2104.03152, April 2021.

[73] Alberto Ibarrondo and Alexander Viand. Pyfhel: Python for homomorphic encryption libraries. In *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 11–16, 2021.

[74] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, page 113–124, New York, NY, USA, 2011. Association for Computing Machinery.

[75] Jihoon Cho, Jincheol Ha, Seongkwang Kim, Byeonghak Lee, Joohee Lee, Jooyoung Lee, Dukjae Moon, and Hyojin Yoon. Transciphering Framework for Approximate Homomorphic Encryption (Full Version), 2020.

[76] Orel Cosseron, Clément Hoffmann, Pierrick Méaux, and François-Xavier Standaert. Towards Globally Optimized Hybrid Homomorphic Encryption - Featuring the Elisabeth Stream Cipher, 2022.

[77] Jihoon Cho et al. Rtf transciphering framework. https://github.com/KAIST-CryptLab/RtF-Transciphering, 2020.

[78] Lattigo v6. Online: https://github.com/tuneinsight/lattigo, August 2024. EPFL-LDS, Tune Insight SA.

[79] Orel Cosseron et al. Elisabeth repository. https://github.com/princess-elisabeth/Elisabeth, 2022.

[80] Zama. Concrete: TFHE Compiler that converts python programs into FHE equivalent, 2022. https://github.com/zama-ai/concrete.

[81] Orel Cosseron et al. Elisabeth use case. https://github.com/princess-elisabeth/elisabeth_usecase/tree/master, 2022.

[82] Christoph Dobraunig et al. Framework for hybrid homomorphic encryption. https://github.com/isec-tugraz/hybrid-HE-framework, 2021.

[83] Matthew J Page, Joanne E McKenzie, Patrick M Bossuyt, Isabelle Boutron, Tammy C Hoffmann, and Cynthia D et al. Mulrow. The prisma 2020 statement: an updated guideline for reporting systematic reviews. *BMJ*, 372, 2021.

[84] Li Li, Yuxi Fan, Mike Tse, and Kuo-Yi Lin. A review of applications in federated learning. *Computers & Industrial Engineering*, 149:106854, November 2020.

[85] Sawsan Abdulrahman, Hanine Tout, Hakima Ould-Slimane, Azzam Mourad, Chamseddine Talhi, and Mohsen Guizani. A Survey on Federated Learning: The Journey From Centralized to Distributed On-Site Learning and Beyond. *IEEE Internet of Things Journal*, 8(7):5476–5497, April 2021.

[86] Xuefei Yin, Yanming Zhu, and Jiankun Hu. A Comprehensive Survey of Privacy-preserving Federated Learning: A Taxonomy, Review, and Future Directions. *ACM Computing Surveys*, 54(6):1–36, July 2022.

[87] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection. *IEEE Transactions on Knowledge and Data Engineering*, 35(4):3347–3366, April 2023.

[88] Yang Liu, Yan Kang, Tianyuan Zou, Yanhong Pu, Yuanqin He, Xiaozhou Ye, Ye Ouyang, Ya-Qin Zhang, and Qiang Yang. Vertical federated learning: Concepts, advances, and challenges. *IEEE Transactions on Knowledge and Data Engineering*, 36(7):3615–3634, 2024.

[89] Chuan Chen, Tianchi Liao, Xiaojun Deng, Zihou Wu, Sheng Huang, and Zibin Zheng. Advances in robust federated learning: A survey with heterogeneity considerations. *IEEE Transactions on Big Data*, 11(3):1548–1567, 2025.

[90] Afsana Khan, Marijn ten Thij, and Anna Wilbik. Vertical federated learning: A structured literature review. *Knowledge and Information Systems*, 67(4):3205–3243, April 2025.

[91] Yanli Li, Zhongliang Guo, Nan Yang, Huaming Chen, Dong Yuan, and Weiping Ding. Threats and defenses in the federated learning life cycle: A comprehensive survey and challenges. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2025.

[92] Dhanya Shenoy, Radhakrishna Bhat, and K. Krishna Prakasha. Exploring privacy mechanisms and metrics in federated learning. *Artificial Intelligence Review*, 58(8):223, May 2025.

[93] Nasir Ahmad Jalali and Hongsong Chen. Federated Learning Security and Privacy-Preserving Algorithm and Experiments Research Under Internet of Things Critical Infrastructure. *Tsinghua Science and Technology*, 29(2):400–414, April 2024.

[94] Neveen Mohammad Hijazi, Moayad Aloqaily, and Mohsen Guizani. Collaborative iot learning with secure peer-to-peer federated approach. *Computer Communications*, 228:107948, 2024.

[95] Phan The Duy, Nguyen Huu Quyen, Nghi Hoang Khoa, Tuan-Dung Tran, and Van-Hau Pham. FedChain-Hunter: A reliable and privacy-preserving aggregation for federated threat hunting framework in SDN-based IIoT. *Internet of Things*, 24:100966, December 2023.

[96] Jing Ma, Si-Ahmed Naas, Stephan Sigg, and Xixiang Lyu. Privacy-preserving federated learning based on multi-key homomorphic encryption. *International Journal of Intelligent Systems*, 37(9):5880–5901, 2022.

[97] Yiming Zhang, Wei Zhang, and Cong Shen. 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*, ECPDC '24, page 96–101, New York, NY, USA, 2024. Association for Computing Machinery.

[98] Cheng Song, Zhichao Wang, Weiping Peng, and Nannan Yang. Secure and Efficient Federated Learning Schemes for Healthcare Systems. *Electronics*, 13(13):2620, January 2024.

[99] Li Zhang, Jianbo Xu, Pandi Vijayakumar, Pradip Kumar Sharma, and Uttam Ghosh. Homomorphic Encryption-Based Privacy-Preserving Federated Learning in IoT-Enabled Healthcare System. *IEEE Transactions on Network Science and Engineering*, 10(5):2864–2880, September 2023.

[100] Abbas Yazdinejad, Ali Dehghantanha, Gautam Srivastava, Hadis Karimipour, and Reza M. Parizi. Hybrid Privacy Preserving Federated Learning Against Irregular Users in Next-Generation Internet of Things. *Journal of Systems Architecture*, 148:103088, March 2024.

[101] Yanli Ren, Yerong Li, Guorui Feng, and Xinpeng Zhang. Privacy-Enhanced and Verification-Traceable Aggregation for Federated Learning. *IEEE Internet of Things Journal*, 9(24):24933–24948, December 2022.

[102] Neveen Mohammad Hijazi, Moayad Aloqaily, Mohsen Guizani, Bassem Ouni, and Fakhri Karray. Secure Federated Learning With Fully Homomorphic Encryption for IoT Communications. *IEEE Internet of Things Journal*, 11(3):4289–4300, February 2024.

[103] Zu-Sheng Tan, Eric W.K. See-To, Kwan-Yeung Lee, Hong-Ning Dai, and Man-Leung Wong. Privacy-preserving federated learning for proactive maintenance of IoT-empowered multi-location smart city facilities. *Journal of Network and Computer Applications*, 231:103996, November 2024.

[104] JiaMing Wang, Kai Yang, and MinJing Li. Nids-fgpa: A federated learning network intrusion detection algorithm based on secure aggregation of gradient similarity models. *PLOS ONE*, 19, 10 2024.

[105] O. et al. Fontenla-Romero. FedHEONN: Federated and homomorphically encrypted learning method for one-layer neural networks. *Future Generation Computer Systems*, 149:200–211, December 2023.

[106] Dongqi Cai, Tao Fan, Yan Kang, Lixin Fan, Mengwei Xu, Shangguang Wang, and Qiang Yang. Accelerating Vertical Federated Learning. *IEEE Transactions on Big Data*, 10(6):752–760, December 2024.

[107] Yihang Xu, Yuxing Mao, Jian Li, Xueshuo Chen, and Shunxin Wu. Edge server enhanced secure and privacy preserving federated learning. *Computer Networks*, 249:110465, July 2024.

[108] Chen Fang, Yuanbo Guo, Yongjin Hu, Bowen Ma, Li Feng, and Anqi Yin. Privacy-preserving and communication-efficient federated learning in internet of things. *Computers & Security*, 103:102199, 2021.

[109] Eric Appiah Mantey, Conghua Zhou, Joseph Henry Anajemba, John Kingsley Arthur, Yasir Hamid, Atif Chowhan, and Obinna Ogbonnia Otuu. Federated Learning Approach for

Secured Medical Recommendation in Internet of Medical Things Using Homomorphic Encryption. *IEEE Journal of Biomedical and Health Informatics*, 28(6):3329–3340, June 2024.

[110] Jianping Wu, Chunming Wu, Chaochao Chen, Jiahe Jin, and Chuan Zhou. EVFL: Towards Efficient Verifiable Federated Learning via Parameter Reuse and Adaptive Sparsification. *Mathematics*, 12(16):2479, January 2024.

[111] Yange Chen, Suyu He, Baocang Wang, Pu Duan, Benyu Zhang, Zhiyong Hong, and Yuan Ping. Cryptanalysis and Improvement of DeepPAR: Privacy-Preserving and Asynchronous Deep Learning for Industrial IoT. *IEEE Internet of Things Journal*, 9(21):21958–21970, November 2022.

[112] Mingwu Zhang, Shijin Chen, Jian Shen, and Willy Susilo. Privacyeafl: Privacy-enhanced aggregation for federated learning in mobile crowdsensing. *IEEE Transactions on Information Forensics and Security*, 18:5804–5816, 2023.

[113] Khoa Nguyen, Tanveer Khan, Hossein Abdinasibfar, and Antonis Michalas. A privacy-centric approach: Scalable and secure federated learning enabled by hybrid homomorphic encryption. arXiv preprint arXiv:2507.14853, 2025.

[114] Jincheol Ha, Seongkwang Kim, Byeonghak Lee, Jooyoung Lee, and Mincheol Son. Rubato: Noisy ciphers for approximate homomorphic encryption (full version). Cryptology ePrint Archive, Paper 2022/537, 2022.

[115] Eugene Frimpong, Khoa Nguyen, Mindaugas Budzys, Tanveer Khan, and Antonis Michalas. Guardml: Efficient privacy-preserving machine learning services through hybrid homomorphic encryption. In *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, SAC '24, page 953–962, New York, NY, USA, 2024. Association for Computing Machinery.

[116] Khoa Nguyen, Mindaugas Budzys, Eugene Frimpong, Tanveer Khan, and Antonis Michalas. A pervasive, efficient and private future: Realizing privacy-preserving machine learning through hybrid homomorphic encryption. arXiv preprint arXiv:2409.06422, September 2024.

[117] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards Federated Learning at Scale: System Design. arXiv preprint arXiv:1902.01046, March 2019.

[118] Alexander Ziller, Andrew Trask, Antonio Lopardo, Benjamin Szymkow, Bobby Wagner, and Emma et al. Bluemke. Pysyft: A library for easy federated learning. In Muhammad Habib ur Rehman and Mohamed Medhat Gaber, editors, *Federated Learning Systems: Towards Next-Generation AI*, pages 111–139. Springer International Publishing, Cham, 2021.

[119] Viraaji Mothukuri, Prachi Khare, Reza M. Parizi, Seyedamin Pouriyeh, Ali Dehghantanha, and Gautam Srivastava. Federated-Learning-Based Anomaly Detection for IoT Security Attacks. *IEEE Internet of Things Journal*, 9(4):2545–2554, February 2022.

[120] Patrick Foley, Micah J Sheller, Brandon Edwards, Sarthak Pati, Walter Riviera, Mansi Sharma, Prakash Narayana Moorthy, Shih-han Wang, Jason Martin, Parsa Mirhaji, Prashant Shah, and Spyridon Bakas. Openfl: the open federated learning library. *Physics in Medicine & Biology*, 67(21):214001, oct 2022.

[121] Junming Ma, Yancheng Zheng, Jun Feng, Derun Zhao, Haoqi Wu, Wenjing Fang, Jin Tan, Chaofan Yu, Benyu Zhang, and Lei Wang. SecretFlow-SPU: A Performant and User-Friendly Framework for Privacy-Preserving Machine Learning. In *2023 USENIX Annual Technical Conference (USENIX ATC '23)*, pages 17–33, Boston, MA, July 2023. USENIX Association.

[122] Yang Liu, Tao Fan, Tianjian Chen, Qian Xu, and Qiang Yang. FATE: An industrial grade platform for collaborative learning with data protection. *J. Mach. Learn. Res.*, 22(1):226:10320–226:10325, January 2021.

[123] Holger R. Roth, Yan Cheng, Yuhong Wen, Isaac Yang, Ziyue Xu, Yuan-Ting Hsieh, Kristopher Kersten, Ahmed Harouni, Can Zhao, Kevin Lu, Zhihong Zhang, Wenqi Li, Andriy Myronenko, Dong Yang, Sean Yang, Nicola Rieke, Abood Quraini, Chester Chen, Daguang Xu, Nic Ma, Prerna Dogra, Mona Flores, and Andrew Feng. NVIDIA FLARE: Federated Learning from Simulation to Real-World. *arXiv preprint arXiv:2210.13291*, 2022.

[124] Alessio Catalfamo, Lorenzo Carnevale, Marco Garofalo, and Massimo Villari. Flower Full-Compliant Implementation of Federated Learning with Homomorphic Encryption. In *2024 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–5, June 2024.

[125] Junming Ma, Yancheng Zheng, Jun Feng, Derun Zhao, Haoqi Wu, Wenjing Fang, Jin Tan, Chaofan Yu, Benyu Zhang, and Lei Wang. SecretFlow-SPU: A performant and User-Friendly framework for Privacy-Preserving machine learning. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 17–33, Boston, MA, July 2023. USENIX Association.

[126] Introduction to grpc. https://grpc.io/docs/what-is-grpc/introduction/, 2025. Accessed: 2025-08-31.

[127] Jiedong Lang, Zhehao Guo, and Shuyu Huang. A comprehensive study on quantization techniques for large language models. arXiv preprint arXiv:2411.02530, 2024.

[128] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *Advances in Cryptology – CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part III*, page 483–512, Berlin, Heidelberg, 2018. Springer-Verlag.

[129] Gaurav Sharma. Mnist cnn with 8k parameters. https://www.kaggle.com/code/gauravsharma99/mnist-8k-parameters, 2020. Accessed: 2025-06-11.

[130] Elaine Barker. Recommendation for Key Management Part 1: General. Technical Report NIST SP 800-57pt1r4, National Institute of Standards and Technology, January 2016.

[131] David L. Olson and Dursun Delen. *Advanced Data Mining Techniques*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[132] Juan Terven, Diana-Margarita Cordova-Esparza, Julio-Alejandro Romero-González, Alfonso Ramírez-Pedraza, and E. A. Chávez-Urbiola. A comprehensive survey of loss functions and metrics in deep learning. *Artificial Intelligence Review*, 58(7), April 2025.

[133] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.