HAL open science

Privacy-preserving machine learning for large-scale collaborative healthcare data analysis

Riccardo Taiello

▶ To cite this version:

Riccardo Taiello. Privacy-preserving machine learning for large-scale collaborative healthcare data analysis. Medical Imaging. Université Côte d'Azur, 2024. English. ⟨NNT : 2024COAZ4031⟩. ⟨tel-04792608⟩

HAL Id: tel-04792608

https://theses.hal.science/tel-04792608v1

Submitted on 20 Nov 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Autorisation HAL

HAL Authorization

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

Thèse de Doctorat

Apprentissage Automatique Sécurisé Pour L'Analyse Collaborative Des Données De Santé À Grande Échelle

Riccardo Taiello

Centre Inria d'Université Côte d'Azur, Équipe Epione

Thèse dirigée par Marco Lorenzi/co-dirigée par Melek Önen

et co-encadrée par : Olivier Humbert

Soutenue le 24 Septembre 2024

Présentée en vue de l'obtention du grade de Docteur en Informatique d'Université Côte d'Azur.

Devant le jury composé de :

Sébastien Gambs	Université du Québec à Montréal	Rapporteur
Pascal Lafourcade	Université Clermont Auvergne	Rapporteur
Holger Roth	NVIDIA	Examinateur
Marco Lorenzi	Centre INRIA d'Université Côte d'Azur	Directeur de thèse
Melek Önen	EURECOM, Sophia Antipolis	Co-directrice de thèse
Olivier Humbert	Université Côte d'Azur	Co-encadrant

Apprentissage Automatique Sécurisé Pour L'Analyse Collaborative Des Données De Santé À Grande Échelle

Privacy-Preserving Machine Learning for Large-Scale Collaborative Healthcare Data Analysis

Riccardo Taiello

Jury :

Rapporteurs:
Sébastien Gambs, Professeur d'université, Université du Québec à Montréal
Pascal Lafourcade, Professeur d'université, Université Clermont Auvergne

Examinateur:
Holger Roth, Chef de projet, NVIDIA

Directeur de thèse:
Marco Lorenzi, Chargé de recherche, Centre INRIA d'Université Côte d'Azur

Co-Directrice de thèse:
Melek Önen, Professeure associée, EURECOM, Sophia Antipolis

Co-encadrant de thèse:
Olivier Humbert, Professeur d'université, Université Côte d'Azur

Résumé

Cette thèse de doctorat explore l'intégration de la préservation de la confidentialité, de l'imagerie médicale et de l'apprentissage fédéré (FL) à l'aide de méthodes cryptographiques avancées. Dans le cadre de l'analyse d'images médicales, nous développons un cadre de recalage d'images préservant la confidentialité (PPIR). Ce cadre aborde le défi du recalage des images de manière confidentielle, sans révéler leur contenu. En étendant les paradigmes de recalage classiques, nous incorporons des outils cryptographiques tels que le calcul multipartite sécurisé et le chiffrement homomorphe pour effectuer ces opérations en toute sécurité. Ces outils sont essentiels car ils empêchent les fuites de données pendant le traitement. Étant donné les défis associés à la performance et à l'évolutivité des méthodes cryptographiques dans les données de haute dimension, nous optimisons nos opérations de recalage d'images en utilisant des approximations de gradient. Notre attention se porte sur des méthodes de recalage de plus en plus complexes, telles que les approches rigides, affines et non linéaires utilisant des splines cubiques ou des difféomorphismes, paramétrées par des champs de vitesses variables dans le temps. Nous démontrons comment ces méthodes de recalage sophistiquées peuvent intégrer des mécanismes de préservation de la confidentialité de manière efficace dans diverses tâches. Parallèlement, la thèse aborde le défi des retardataires dans l'apprentissage fédéré, en mettant l'accent sur le rôle de l'agrégation sécurisée (SA) dans l'entraînement collaboratif des modèles. Nous introduisons "Eagle", un schéma SA synchrone conçu pour optimiser la participation des dispositifs arrivant tardivement, améliorant ainsi considérablement les efficacités computationnelle et de communication. Nous présentons également "Owl", adapté aux environnements FL asynchrones tamponnés, surpassant constamment les solutions antérieures. En outre, dans le domaine de la Buffered AsyncSA, nous proposons deux nouvelles approches : "Buffalo" et "Buffalo+". "Buffalo" fait progresser les techniques de SA pour la Buffered AsyncSA, tandis que "Buffalo+" contrecarre les attaques sophistiquées que les méthodes traditionnelles ne parviennent pas à détecter. Cette solution exploite les propriétés des fonctions de hachage incrémentielles et explore la parcimonie dans la quantification des gradients locaux des modèles clients. "Buffalo" et "Buffalo+" sont validés théoriquement et expérimentalement, démontrant leur efficacité dans une nouvelle tâche de FL inter-dispositifs pour les dispositifs médicaux. Enfin, cette thèse a accordé une attention particulière à la traduction des outils de préservation de la confidentialité dans des applications réelles, notamment grâce au cadre open-source FL Fed-BioMed. Les contributions concernent l'introduction de l'une des premières implémentations pratiques de SA spécifiquement conçues pour le FL inter-silos entre hôpitaux, mettant en évidence plusieurs cas d'utilisation pratiques.

Mots-clés: Sécurité et confidentialité, Technologies de renforcement de la confidentialité, Apprentissage fédéré.

Abstract

This PhD thesis explores the integration of privacy preservation, medical imaging, and Federated Learning (FL) using advanced cryptographic methods. Within the context of medical image analysis, we develop a privacy-preserving image registration (PPIR) framework. This framework addresses the challenge of registering images confidentially, without revealing their contents. By extending classical registration paradigms, we incorporate cryptographic tools like secure multi-party computation and homomorphic encryption to perform these operations securely. These tools are vital as they prevent data leakage during processing. Given the challenges associated with the performance and scalability of cryptographic methods in high-dimensional data, we optimize our image registration operations using gradient approximations. Our focus extends to increasingly complex registration methods, such as rigid, affine, and non-linear approaches using cubic splines or diffeomorphisms, parameterized by time-varying velocity fields. We demonstrate how these sophisticated registration methods can integrate privacy-preserving mechanisms effectively across various tasks. Concurrently, the thesis addresses the challenge of stragglers in FL, emphasizing the role of Secure Aggregation (SA) in collaborative model training. We introduce "Eagle", a synchronous SA scheme designed to optimize participation by late-arriving devices, significantly enhancing computational and communication efficiencies. We also present "Owl", tailored for buffered asynchronous FL settings, consistently outperforming earlier solutions. Furthermore, in the realm of Buffered AsyncSA, we propose two novel approaches: "Buffalo" and "Buffalo+". "Buffalo" advances SA techniques for Buffered AsyncSA, while "Buffalo+" counters sophisticated attacks that traditional methods fail to detect, such as model replacement. This solution leverages the properties of incremental hash functions and explores the sparsity in the quantization of local gradients from client models. Both Buffalo and Buffalo+ are validated theoretically and experimentally, demonstrating their effectiveness in a new cross-device FL task for medical devices. Finally, this thesis has devoted particular attention to the translation of privacy-preserving tools in real-world applications, notably through the FL open-source framework Fed-BioMed. Contributions concern the introduction of one of the first practical SA implementations specifically designed for cross-silo FL among hospitals, showcasing several practical use cases.

Keywords: Security and privacy, Privacy enhancing technologies, Federated Learning.

Acknowledgement

I am grateful to Marco and Melek for giving me the opportunity to undertake this PhD project. Their trust from the beginning, followed by their support and patience, guided me through this journey, enriching me both professionally and personally. I extend my thanks to Olivier, whose support and technical knowledge were crucial in providing context and understanding of topics often distant from my expertise.

As an external collaborator, I would like to primarily thank Clementine. Her patience and rigor have been a steadfast support, especially in the final year of my PhD. The challenges we faced were always less daunting with her assistance. I also thank the entire SED team: Sergen, Yannick, Marc, and Francesco. Through the Fed-BioMed project, we worked together and I learned a great deal about development and teamwork. My gratitude extends to all the members of my Epione group, past and present, with whom I shared many important moments.

Finally, I wish to thank everyone who has supported me throughout this journey, which extends far beyond just the PhD. From my early education through to the completion of my doctorate, I am especially thankful to Professors M. Varnier and G. Barletta. Their guidance, dedication, and enthusiasm sparked my curiosity and helped shape my approach, even during times when I was prone to distraction and lacked consistency.

During my first university journey, I am deeply grateful to my Aunt Gina for her constant support and belief in me, and to the precious friendship and brotherhood I found in Luca (First) and Alessandro. To Andrea, whom I met during my master's, for making me realize that passion can overcome any limitations.

In this journey, two people have been fundamental: Mr. Giulio and Lucia. I met them thanks to this path, and they have supported and helped me through the hardest moments, making me a better person.

Lastly, to my parents, my mom and dad, Cinzia and Michele, to whom I owe everything.

Financial Support

This work has been supported by the French government, through the 3IA Côte d'Azur Investments in the Future project managed by the National Research Agency (ANR) with the reference number ANR-19-P3IA-0002by the TRAIN project ANR-22-FAI1-0003-02, and by the ANR JCJC project Fed-BioMed 19-CE45-0006-01. The authors are grateful to the OPAL infrastructure from Université Côte d'Azur for providing resources and support.

Contents

1 Introduction **1**

1.1 Requirements . . . 3

1.2 Privacy-Enhancing Technologies . . . 4

1.3 Challenges in Privacy-Preserving Medical Applications . . . 7

1.4 Contributions . . . 8

2 Privacy Preserving Image Registration **11**

2.1 Introduction . . . 12

2.2 Background . . . 13

2.2.1 Analysis of classical IR loss functions under a privacy-preserving perspective . . . 15

2.2.2 Optimization of SSD loss . . . 15

2.2.3 Mutual Information . . . 16

2.2.4 Cross Correlation with Advanced Normalization Tools . . . 18

2.2.5 Building blocks for Secure Computation . . . 19

2.2.6 Secure Multi-Party Computation . . . 19

2.2.7 Homomorphic Encryption . . . 19

2.3 Methods: from IR to PPIR . . . 20

2.3.1 PPIR based on SSD . . . 20

2.3.2 PPIR based on MI . . . 21

2.3.3 PPIR based on ANTS CC loss . . . 21

2.3.4 Protocols enhancement for SSD loss . . . 22

2.4 Experiments & Results . . . 23

2.4.1 Experimental data . . . 24

2.4.2 Experimental Details . . . 24

2.4.3 Results . . . 26

2.5 Discussion . . . 31

2.6 Conclusion and future works . . . 32

3 Secure Aggregation for Real-World Applications of Federated Learning in Healthcare **33**

3.1 Related Works . . . 36

3.2 Methods . . . 37

3.3 Evaluation . . . 39

3.3.1 Experimental evaluation: . . . 40
3.4 Conclusion and Future Works . . . 42

4 Let Them Drop: Scalable and Efficient Federated Learning Solutions Agnostic to Client Stragglers 43
4.1 Introduction . . . 44
4.2 Building Blocks . . . 46
4.2.1 Joye-Libert Secure Aggregation Scheme . . . 46
4.2.2 Threshold Joye-Libert SA Scheme . . . 47
4.3 Eagle in SyncFL . . . 48
4.4 Owl in AsyncFL . . . 53
4.5 Related Work . . . 55
4.6 Complexity Analysis . . . 57
4.6.1 **Eagle** . . . 57
4.6.2 **Eagle** with decryptors . . . 58
4.6.3 **Owl** . . . 59
4.7 Experimental Study . . . 60
4.7.1 Experimental Setting . . . 60
4.7.2 Eagle . . . 61
4.7.3 **Eagle** vs. **SecAgg** and **FTSA** . . . 61
4.7.4 **Eagle** vs. **Flamingo** . . . 62
4.7.5 **Owl** . . . 63
4.8 Conclusion . . . 65

5 Buffalo: A Practical Secure Aggregation Protocol for Asynchronous Federated Learning 67
5.1 Introduction . . . 68
5.2 Related work . . . 70
5.3 Background . . . 72
5.3.1 Synchronous Federated Learning . . . 72
5.3.2 Buffered Asynchronous Federated Learning . . . 73
5.3.3 SA Threat Model and Security. . . . 75
5.4 Building Blocks . . . 76
5.4.1 Joye-Libert SA . . . 76
5.4.2 LWE-based SA . . . 77
5.4.3 Aggregation Verifiability . . . 77
5.5 Our protocols . . . 78
5.5.1 Buffalo . . . 78
5.5.2 Buffalo+ . . . 82
5.6 Complexity Analysis . . . 85
5.7 Experimental Results . . . 86
5.7.1 Overall performance of BAsyncFL schemes . . . 90

5.8 Conclusion 92

6 Conclusion **93**
6.1 Summary of the Main Contributions 93
6.1.1 Privacy-Preserving Image Registration 93
6.1.2 Enhancing Privacy in Federated Learning: Secure Aggregation for Real-World Healthcare Applications 94
6.1.3 Let Them Drop: Scalable and Efficient Federated Learning Solutions Agnostic to Stragglers 94
6.1.4 Buffalo: A Practical Secure Aggregation Protocol for Asynchronous Federated Learning 95
6.2 Perspectives and Future Applications 95
6.2.1 PPIR extensions 95
6.2.2 Privacy-Preserving Neural Networks for Medical Imaging Applications 97
6.2.3 Leveraging Sparsity in Distributed Learning with Secure Aggregation 97
6.2.4 Post-Quantum Secure Aggregation 98

A Appendix Chapter 2 **99**
A.1 Rigid Point Cloud Registration 100

B Appendix Chapter 3 **105**

C Appendix Chapter 4 **107**
C.1 Hybrid Security Proofs 107
C.1.1 Honest-but-Curious Model 107
C.1.2 Active Model 110

D Appendix Chapter 5 **115**
D.1 Notations 115
D.2 Cryptographic Building Blocks 115
D.2.1 Elliptic Curves 115
D.2.2 Shamir Secret Sharing 116
D.2.3 Threshold ElGamal Encryption 116
D.2.4 Other Cryptographic Primitives 117
D.3 Complexity LightSecAgg in BAsyncFL 118
D.4 Implementation Details of BAsyncFL 118
D.5 Hybrid Security Proofs 119

Bibliography **123**

1 Introduction

Contents

1.1 Requirements 3
1.2 Privacy-Enhancing Technologies 4
1.3 Challenges in Privacy-Preserving Medical Applications 7
1.4 Contributions 8

The integration of digital technologies in healthcare offers the potential to enhance diagnosis, treatment, and patient outcomes, offering unprecedented opportunities to optimize patient care, streamline administrative processes, and advance medical research. These applications include electronic health records (EHRs), telemedicine platforms, mobile health apps, and data analytics systems. The use of artificial intelligence (AI) is expected in particular to significantly improve healthcare at both patients and through improved workflow automation. The potential of AI is expected to disrupt the way we conceive healthcare nowadays, improving every aspect from management to patient care.

The performance and quality of AI systems heavily depend on the availability of large, high-quality and representative datasets used for model training. Nevertheless, when it comes to healthcare applications, medical data format, availability, and quality do not often match the standards needed by AI.

In particular, the sensitive nature of medical data requires robust privacy protection measures when using multiple data sources for training AI models. Medical data contains personal and confidential information, which requires stringent protections to ensure patient privacy. Furthermore, a single hospital often lacks a sufficient volume of data to train comprehensive AI models, effectively. Consequently, collaborative efforts that aggregate data from multiple institutions become essential for developing accurate and reliable models. Complying with data protection regulations encourages the adoption of collaborative learning (CL) strategies, which minimize the exchange of sensitive information while facilitating the creation of robust AI models.

In real-world healthcare applications of AI, due to concerns over data privacy and security, the parties involved, often, may not be allowed to share information in clear form. As reported in Figure 1.1(1), in a setting involving two parties, one may possess sensitive

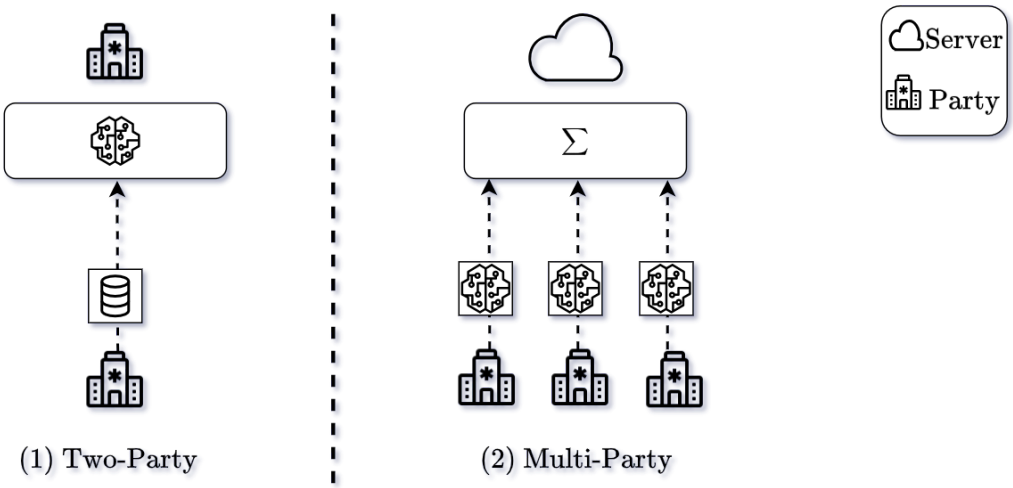


Figure 1.1.: Example of collaborative learning (CL) with two-party and with multiple-party settings.

data while the other could provide a model for training. This situation creates a need for mechanisms enabling collaboration without compromising the confidentiality of the shared information, either under the form of data or model's parameters. A typical application is represented by the collaboration between a hospital having extensive patient records, and a tech company requiring to train a model on this data. Both entities need to collaborate to improve patient outcomes but are worried about of sharing sensitive information directly. A more complex scenario requires multiple parties (Figure 1.1(2)), each holding different sensitive data. In such cases, Federated Learning (FL) [McMahan, 2017a] has emerged as a promising collaborative approach. FL is a decentralized machine learning approach where multiple institutions collaborate to train a shared model without exchanging raw data. Each institution performs rounds of partial model training on the respective local data and only shares model updates which are then aggregated to form a global model. Two primary types of FL have been identified in the literature: cross-silo and cross-device [Kairouz, 2019]. Cross-silo FL involves collaboration between a small number of institutions or organizations, such as hospitals or research centers. Each silo typically has significant computational resources and large datasets. Cross-device FL, on the other hand, consists of a collaboration of a large number of devices, such as smartphones or IoT devices, each with limited computational power and small amounts of data. This approach is useful for applications where data is generated at the edge, such as mobile health apps collecting user data.

Notably, collaborative training alone does not inherently provide strong privacy guarantees [Shokri, 2017]. To further enhance privacy in such environments, Privacy-Enhancing Technologies (PETs) can be integrated into the AI model development process. PETs offer tools and frameworks that ensure data privacy and security by providing explicit guarantees on the protection of the data used to perform mathematical operations, for

example through encryption. Despite the significant potential of PETs to enhance privacy, there is currently a lack of comprehensive frameworks allowing to seamlessly integrate PETs in compliance with the restrictive requirements of real-world healthcare application scenarios. Filling this gap requires to extend the current technology to address unmet needs for real-world deployment, for example guaranteeing the trade-off between privacy protection and system scalability.

To address thes challenges, in this thesis we study the use of advanced PETs in collaborative healthcare applications. We first show that their integration is not straightforward and consequently list the main system requirements for successfull practical deployment. We further review existing PETs to later identify the main challenges raised by their integration into FL systems.

1.1 Requirements

Health information is highly sensitive, and its unauthorized disclosure can result in severe consequences for individuals and organizations. To mitigate these risks, governments have enacted stringent data protection laws, such as the European Union's General Data Protection Regulation (GDPR)[1] and the United States' Health Insurance Portability and Accountability Act (HIPAA)[2].

The GDPR, effective since 2018, is a data protection regulation applicable to all EU member states. It aims to unify data privacy laws across Europe and empower individuals with greater control over their personal information. One of its key requirements is implementing appropriate technical and organizational measures to ensure data security and privacy. In a similar vein, HIPAA, enacted in 1996, establishes national standards for the protection of certain health information in the United States.

To meet the requirements of these regulations and ensure the effective implementation of PETs in AI applications, the following key requirements must be addressed:

- **Privacy (R1):** To design a privacy-preserving application using the previously mentioned PET techniques, it is essential to identify and address key privacy requirements. The provided input must remain confidential and accessible only to its actual owner, ensuring input data privacy. In data privacy, particularly in a two-party setting, the primary focus is protecting data from the other party. While, in a multi-party setting, the individual models must be protected to ensure that local data and models remain private.

[1] https://gdpr-info.eu/
[2] https://www.hhs.gov/hipaa/index.html

- Scalability (R2): Scalability is crucial for the successful adoption of emerging technologies, particularly in real-time clinical settings where performance metrics such as computation and communication overhead vary with factors such as the number of parties involved and the size of the model used.

- Availability (R3): Availability is fundamental, as healthcare providers must have uninterrupted access to critical systems and data. High availability ensures these applications are accessible whenever needed, reducing downtime and minimizing disruptions in patient care. This implies implementing robust infrastructure, and fault-tolerant protocols to maintain continuous operation even during hardware failures or general issues.

In the next section, we introduce the technologies that we use in this thesis, to address the aforementioned requirements.

1.2 Privacy-Enhancing Technologies

Privacy-enhancing technologies (PETs) protect individuals' privacy while allowing data processing and analysis. Below, are the key PETs discussed in this thesis:

Homomorphic Encryption

Homomorphic encryption [Cheon, 2017] enables computations on encrypted data without decrypting it, preserving privacy. This is useful for processing sensitive information, such as medical data, by third-party services without exposing the raw data. It consists of the following algorithms: Key Generation, Encryption, Decryption, and Evaluation. For this explanation, we assume asymmetric encryption, where the key generation algorithm randomly generates a pair of public and private keys. However, symmetric encryption, which uses a single key for both encryption and decryption, is also possible.

$$\text{KeyGen}(1^\lambda) \rightarrow (pk, sk)$$

where λ is the security parameter, pk is the public key, and sk is the private key. The encryption algorithm takes the public key pk, a plaintext message m, and outputs a ciphertext c.

$$\text{Enc}(pk, m) \rightarrow c$$

The decryption algorithm takes a secret key sk and a ciphertext c and outputs the plaintext message m.

$$\text{Dec}(sk, c) \rightarrow m$$

The evaluation algorithm takes a public key pk, a function f, and a set of ciphertexts $c_1, \ldots, c_n$, and outputs a new ciphertext c_f which is the encryption of f applied to the plaintexts.

$$\text{Eval}(pk, f, c_1, \ldots, c_n) \rightarrow c_f$$

For an FHE scheme to be correct, it must satisfy:

$$\text{Dec}(sk, \text{Eval}(pk, f, \text{Enc}(pk, m_1), \ldots, \text{Enc}(pk, m_n))) = f(m_1, \ldots, m_n)$$

for any function f and any plaintexts $m_1, \ldots, m_n$ and any pair of (sk, pk).

Multi-Party Computation

Multi-Party Computation (MPC) [Yao, 1982] allows multiple parties to compute a function over their inputs while keeping those inputs private. This is valuable for collaborative research on sensitive data, such as health information from different institutions.

Let $P_1, P_2, \ldots, P_n$ be n parties, and let $x_1, x_2, \ldots, x_n$ be their respective private inputs. The goal is to compute a function $f(x_1, x_2, \ldots, x_n)$ such that no party learns anything about the inputs of the other parties beyond what can be inferred from the output.

In additive secret sharing [Shamir, 1979], each input x is split into n shares such that the sum of the shares equals the original input. This technique is commonly used in MPC protocols. Share Generation, randomly split x in shares:

$$\text{Share}(x) \rightarrow (x_1, x_2, \ldots, x_n)$$

such that $x = \sum_{i=1}^{n} x_i$. And the reconstruction takes n shares:

$$\text{Reconstruct}(x_1, x_2, \ldots, x_n) \rightarrow x$$

obtaining the original secret x.

Secure Aggregation

Recent studies [Nasr, 2019; Shokri, 2017] have shown that even sharing local FL model parameters may expose some information about the clients' training data through various attacks, such as membership inference or model inversion. A popular solution to tackle such attacks is Secure Aggregation (SA) [Mansouri, 2023], which ensures that the global model's parameters are computed through the aggregation of the individual ones without disclosing them individually. Each client first protects its local parameters and sends them to the server, which then computes the aggregated parameters and shares them back with all the clients.

Formally, are defined three algorithms: Setup, Protect, and Aggregate.

Let $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ be a set of n parties. The Setup algorithm generates secret keys for each party and a server key:

$$\text{Setup}(1^\lambda) \rightarrow (\{sk_i\}_{\forall i \in [1,n]}, sk_s)$$

where λ is the security parameter, sk_i are private keys for parties P_i, and sk_s is the server's key used for aggregation.

Each party P_i, with private input $x_i \in \mathbb{Z}$, protects its input using its secret key sk_i:

$$\text{Protect}(sk_i, x_i) \rightarrow y_i$$

Each party sends y_i to the server. The server aggregates these protected values $\{y_i\}_{\forall i \in [1,n]}$ to compute the aggregate result without decrypting individual inputs:

$$x = \text{Agg}(sk_s, \{y_i\}_{\forall i \in [1,n]})$$

Thus, Secure Aggregation ensures that no individual party learns anything about the inputs of other parties beyond the final aggregate result, thereby safeguarding privacy in collaborative learning scenarios.

Other PETs

In addition to the PETs discussed in this thesis, it is important to briefly mention Differential Privacy (DP) mechanism [Dwork, 2006] and Trusted Execution Environments (TEEs) [Sabt, 2015]. A DP mechanism ensures that the removal or addition of a single database item does not significantly affect the outcome of any analysis, thereby protecting individual data privacy. It typically involves adding random noise to the results of queries on the dataset to mask the presence or absence of any single individual's data. Trusted Execution Environments, on the other hand, provide secure areas within a processor where sensitive data can be processed in isolation from the rest of the system, protecting it from potential breaches.

1.3 Challenges in Privacy-Preserving Medical Applications

The primary aim of this thesis is to explore and implement appropriate Privacy Enhancing Technologies for collaborative medical applications, addressing the previously defined requirements. We emphasize three main challenges that we tackle:

- **Challenge 1: Balancing Privacy, Performance, and Accuracy (R1)**
  Applying PETs to medical applications implies a trade-off between privacy, performance, and accuracy. While these PETs enable patient data protection, they can introduce latency and reduce the speed of data processing. For example, processing over encrypted data may require more computational resources, leading to slower response times and increased communication overhead. This can be problematic, especially in medical applications where real-time or fast responses are needed. We expect accuracy to be reasonably preserved when adapting privacy-preserving solutions. This requirement arises because privacy-preserving solutions often do not efficiently support the original operations, and approximations are required. Hence, they might introduce approximation errors. Balancing these trade-offs is critical to ensure that privacy-preserving medical applications remain both effective and efficient.

- **Challenge 2: Implementing and demonstrating PETs for specific Real-life medical applications. (R2)**
  Implementing PETs in collaborative medical applications presents significant challenges, particularly when ensuring efficient operation over encrypted data. Moreover, the performance overhead associated with these technologies can be substantial, potentially impacting the efficiency of medical applications. Additionally, collaborative applications that require data sharing between multiple institutions or systems must navigate complex encryption and decryption processes, which can further complicate their implementation.

- **Challenge 3: Stragglers in Cross-Device Settings (R3)**
  In cross-device FL, ensuring availability and efficiency can be challenging. Stragglers, or devices that are slower or less reliable than others, can significantly impact the performance of the distributed process and in particular of SA. Addressing this issue requires robust strategies for handling device variability and fault-tolerant protocols. Ensuring high availability in these settings also involves implementing secure communication channels and maintaining integrity across devices.

	Chapter 2	Chapter 3	Chapter 4
C1: Balancing PETs accuracy and performance	✓	✓	
C2: PETs for specific medical task	✓		
C3: Stragglers in cross-device settings		✓	✓

Table 1.1.: Overview of the main contributions tackling challenges.

1.4 Contributions

Table 1.1 depicts our contributions, we address the aforementioned requirements and challenges mentioned.

Chapter 2 deals with C1 and C2 in a two-party setting, addressing a well-known problem in medical applications: image registration. The goal of image registration is to align features between scans to facilitate the representation of medical images in a common spatial reference frame. Image registration has evolved significantly over the years. Initially, the optimization task has been solved through traditional optimization routines such as gradient descent or iterative closest point, to identify optimal parameterized transformations (e.g. linear of splines) optimizing the spatial matching. Despite their effectiveness, these algorithms can be computationally intensive and may struggle with large datasets and complex transformations. With advances in machine learning, neural networks have increasingly been applied to solve the image registration task. They can learn complex patterns and transformations from training data, offering a more flexible and scalable approach compared to traditional methods. These networks can be trained to predict the optimal transformation parameters directly from image pairs, significantly reducing the computational burden during the registration process. However, in the medical context, the scarcity and poor quality of data samples pose challenges in training neural networks to accurately learn the correct data distribution and generalize to new images, especially in non-linear cases. Consequently, standard registration algorithms today still remain a valid and robust option.

The ensemble of these methods is not privacy-compliant because they necessitate direct access to medical data. Current PETs, such as MPC and HE, face significant challenges when dealing with these complex, non-linear processes. A major issue is the scalability of these cryptographic methods. Medical image registration involves computationally intensive tasks, including matrix operations, gradient computations, and iterative refinement. Executing these tasks on encrypted data significantly increases overhead, leading to substantial resource consumption. Hence, in Chapter 2, we study this problem and propose the first 2-party privacy-preserving image registration.

Chapter 3 works with a multi-party setting and presents the practical development of SA in a real-world healthcare FL framework, Fed-BioMed, considering the unique

requirements and constraints of this environment. Despite extensive research in FL, many proposed SA schemes do not fully address the specific needs of healthcare settings, particularly regarding scalability and availability. Most existing SA protocols are designed for scenarios with a large number of clients. These protocols can become computationally expensive when applied to healthcare environments with a limited number of centers, typically requiring multiple communication rounds between clients and the server to ensure privacy. In healthcare cross-silo settings, where network reliability and bandwidth are not limited, the classic scenario accounting for client failure adds unnecessary overhead to the FL framework. Since medical imaging and other healthcare tasks are already resource-intensive to process, the added cryptographic operations can further strain computational resources. This overhead can make real-time or near-real-time processing impractical, which is often required in clinical settings. We implement a suitable SA protocol within the Fed-BioMed framework, showcasing its applications with four known collaborative medical environments.

Finally, the third and fourth contributions of this thesis (Chapters 4 and 5) address C3 and work with a multi-party setting. Specifically, we study how stragglers negatively impact SA in cross-device FL. For Synchronous FL (SyncFL), we analyze how client selection can cause additional overhead due to inherent dropout to cope with stragglers (Chapter 4). We introduce a new SA method agnostic to stragglers in SyncFL, namely *Eagle*, which improves upon previous state-of-the-art methods and stands as one of the first agnostic solutions to stragglers. We also provide a new protocol for Buffered Asynchronous FL. Most SA solutions rely on SyncFL and cannot be adapted to asynchronous environments where clients are not synchronized. We illustrate this with a realistic use case example. In Chapter 5, we further enhance Buffered Asynchronous SA. We introduce one of the first practical Buffered Asynchronous SA schemes, namely *Buffalo*, and extend it by offering aggregation verification. This guarantees that the server correctly aggregates the protected input, ensuring the client that their contribution to the aggregate has been considered. We present the first verifiable asynchronous SA, *Buffalo+*, with low computational overhead, thanks to the properties of incremental hashing. We evaluate the performance of these SA schemes in a cross-device medical use case.

The contributions of this manuscript led to the following publications and submissions in conferences and peer-reviewed journals.

- *Privacy Preserving Image Registration* [Taiello, 2022], Riccardo Taiello, Melek Önen, Olivier Humbert and Marco Lorenzi. The 25th International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI 2022).

- *Privacy Preserving Image Registration* [Taiello, 2024a], Riccardo Taiello, Melek Önen, Francesco Capano, Olivier Humbert and Marco Lorenzi. Medical Image Analysis Journal (MedIA), Volume 94, May 2024, 103129.

- *Enhancing Privacy in Federated Learning: Secure Aggregation for Real-World Healthcare Applications*. Riccardo Taiello, Sergen Cansiz, Marc Vesin, Francesco Cremonesi, Lucia Innocenti, Melek Önen and Marco Lorenzi. The 5th Workshop on Distributed, Collaborative, and Federated Learning, in conjunction with MICCAI 2024.

- *Let Them Drop: Scalable and Efficient Federated Learning Solutions Agnostic to Client Stragglers* [Taiello, 2024b], Riccardo Taiello, Melek Önen, Clémentine Gritti and Marco Lorenzi. The 19th International Conference on Availability, Reliability and Security (ARES 2024).

- *Buffalo: A Practical Secure Aggregation Protocol for Asynchronous Federated Learning*, Riccardo Taiello, Melek Önen, Clémentine Gritti and Marco Lorenzi. Under Review.

2 Privacy Preserving Image Registration

Contents

2.1 Introduction **12**

2.2 Background **13**

2.2.1 Analysis of classical IR loss functions under a privacy-preserving perspective 15

2.2.2 Optimization of SSD loss 15

2.2.3 Mutual Information 16

2.2.4 Cross Correlation with Advanced Normalization Tools 18

2.2.5 Building blocks for Secure Computation 19

2.2.6 Secure Multi-Party Computation 19

2.2.7 Homomorphic Encryption 19

2.3 Methods: from IR to PPIR **20**

2.3.1 PPIR based on SSD 20

2.3.2 PPIR based on MI 21

2.3.3 PPIR based on ANTS CC loss 21

2.3.4 Protocols enhancement for SSD loss 22

2.4 Experiments & Results **23**

2.4.1 Experimental data 24

2.4.2 Experimental Details 24

2.4.3 Results 26

2.5 Discussion **31**

2.6 Conclusion and future works **32**

In this chapter, we present a novel framework for image registration under privacy-preserving conditions. Image registration is a crucial task in medical imaging, facilitating the representation of medical images in a common spatial reference frame. Traditional image registration methods typically assume that image content is accessible in clear form for spatial transformation estimation. However, due to the sensitive nature of medical images, practical applications often require privacy constraints that prevent open sharing of image content.

To address this, we formulate the problem of image registration in a privacy-preserving regime, assuming images are confidential and cannot be disclosed openly. Our privacy-preserving image registration (PPIR) framework extends classical registration paradigms by incorporating advanced cryptographic tools, such as secure multi-party computation and homomorphic encryption. These tools enable operations without revealing underlying data.

To tackle the performance and scalability challenges of cryptographic tools in high-dimensional spaces, we introduce several optimization techniques. These include gradient approximations and the use of homomorphic encryption through packing, which allows efficient encryption and multiplication of large matrices. Our focus spans registration methods of varying complexity, including rigid, affine, and non-linear registration based on cubic splines or diffeomorphisms parameterized by time-varying velocity fields.

We demonstrate how the registration problem can be adapted to privacy-preserving operations and showcase the effectiveness of PPIR across a range of registration tasks. This chapter has been accepted to MICCAI 2022 [Taiello, 2022] and published in Medical Image Analysis [Taiello, 2024a].

2.1 Introduction

Image Registration is a crucial task in medical imaging applications, allowing to spatially align imaging features between two or multiple scans. Registration methods are today a central component of state-of-the-art methods for atlas-based segmentation [Shattuck, 2009; Cardoso, 2013], morphological and functional analysis [Dale, 1999; Ashburner, 2000], multi-modal data integration [Heinrich, 2011], and longitudinal analysis [Reuter, 2010; Ashburner, 2013]. Typical registration paradigms are based on a given transformation model (e.g. affine or non-linear), a cost function and an associated optimization routine. A large number of image registration approaches have been proposed in the literature over the last decades, covering a variety of assumptions on the spatial transformations, cost functions, image dimensionality and optimization strategy [Schnabel, 2016]. Image registration is the workhorse of many real-life medical imaging software and applications, including public web-based services for automated segmentation and labelling of medical images. Using these services generally requires uploading and exchanging medical images over the Internet, to subsequently perform image registration with respect to one or multiple (potentially proprietary) atlases. Besides these classical medical imaging use-cases, emerging paradigms for collaborative data analysis, such as Federated Learning (FL) [McMahan, 2017b], have been proposed to enable analysis of medical images in multicentric scenarios for performing group analysis [Gazula, 2021] and distributed machine learning [Kaissis, 2021; Zerka, 2020]. However,

in these settings, typical medical imaging tasks such as spatial alignment and downstream operations are generally not possible without disclosing the image information.

Due to the evolving juridical landscape on data protection, medical image analysis tools need to be adapted to guarantee compliance with regulations currently existing in many countries, such as the European General Data Protection Regulation (GDPR) [1], or the US Health Insurance Portability and Accountability Act (HIPAA)[2]. Medical imaging information falls within the realm of personal health data [Lotan, 2020] and its sensitive nature should ultimately require the analysis under privacy preserving constraints, for instance by preventing to share the image content in clear form.

Advanced cryptographic tools hold great potential in sensitive data analysis problems (e.g., [Lauter, 2021]). Examples of such approaches are Secure-Multi-Party-Computation (MPC) [Yao, 1982] and Homomorphic Encryption (HE) [Rivest, 1978b]. While MPC allows multiple parties to jointly compute a common function over their private inputs and discover no more than the output of this function, HE enables computation on encrypted data without disclosing either the input data or the result of the computation.

This work presents *privacy-preserving image registration* (PPIR), a new methodological framework allowing image registration under privacy constraints. To this end, we reformulate the image registration problem to integrate cryptographic tools, namely MPC or FHE, thus preserving the privacy of the image data. Due to the well-known scalability issues of such cryptographic techniques, we investigate strategies for the practical use of PPIR. In our experiments, we evaluate the effectiveness of PPIR on a variety of registration tasks and medical imaging modalities. Our results demonstrate the feasibility of PPIR and pave the way for the application of secured image registration in sensitive medical imaging applications.

2.2 Background

Given images $I, J : \mathbb{R}^d \mapsto \mathbb{R}$, image registration (IR) aims at estimating the parameters $\boldsymbol{\theta}$ of a spatial transformation $\boldsymbol{W}_{\boldsymbol{\theta}} \in \mathbb{R}^d \mapsto \mathbb{R}^d$, either linear or non-linear, maximizing the spatial overlap between J and the transformed image $I(\boldsymbol{W}_{\boldsymbol{\theta}})$, by minimizing a registration loss function f:

$$\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}}\ f\left(I(\boldsymbol{W}_{\boldsymbol{\theta}}(\boldsymbol{x})), J(\boldsymbol{x})\right). \tag{2.1}$$

The loss f can be any similarity measure, e.g., the Sum of Squared Differences (SSD), the negative Mutual Information (MI), or normalized cross correlation (CC). Equation

[1] https://gdpr-info.eu/
[2] https://www.hhs.gov/hipaa/index.html

Algorithm 1 IR via Gauss-Newton optimization

Input:
- ▷ Moving image I
- ▷ Template image J
- ▷ Distance function f
- $\boldsymbol{W_\theta}$
- ▷ convergence threshold ϵ

Output:
- ▷ Transformed image $I(\boldsymbol{W_\theta})$ after convergence is reached

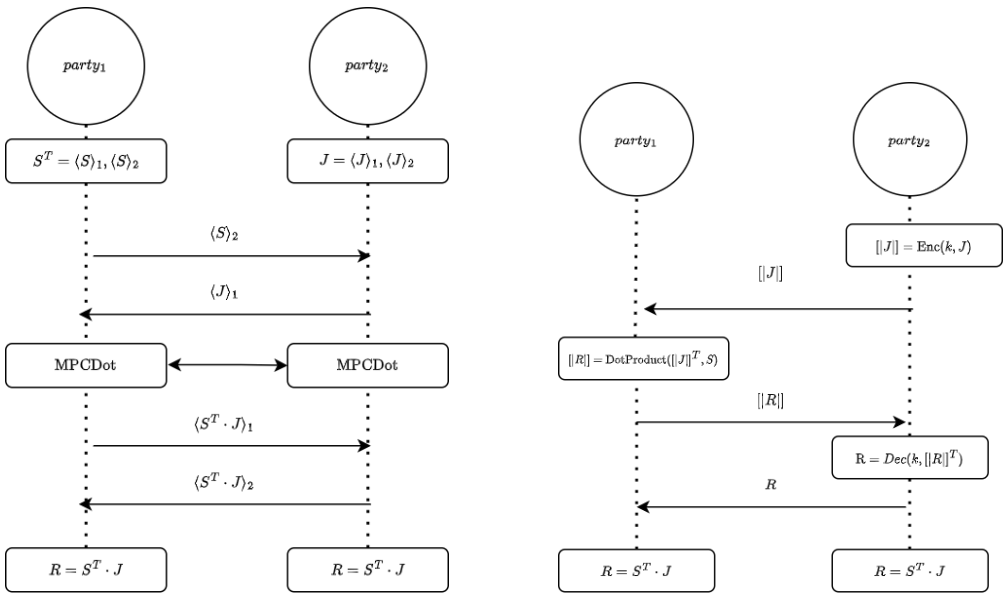
```
function IMAGEREGISTRATION(I, J, W_θ):
    θ ⟵ InitializeParameters()
    repeat
        e ⟵ f[I(W_θ), J]
        G ⟵ ∂f/∂θ
        H ⟵ ∂²f/∂θ²
        Δθ ⟵ H⁻¹ · G
        θ ⟵ θ + Δθ
    until ‖Δθ‖ ≤ ϵ

    return I(W_θ), e
end function
```

(2.1) can be typically optimized through gradient-based methods, where the parameters $\boldsymbol{\theta}$ are iteratively updated until convergence. In particular, when using a Gauss-Newton optimization scheme (Algorithm 1), the update of the spatial transformation can be computed through Equation (2.2):

$$\boldsymbol{\Delta\theta} = H^{-1} \cdot G, \tag{2.2}$$

where $G = \frac{\partial f}{\partial \boldsymbol{\theta}}$ is the Jacobian and $H = \frac{\partial^2 f}{\partial \boldsymbol{\theta}^2}$ the Hessian of f. Besides the Gauss-Newton schemes proposed in the field of IR [Pennec, 1999; Modersitzki, 2009], gradient-based techniques are classically adopted to solve the IR task, for example in diffeomorphic image registration problems [Ashburner, 2007; Avants, 2011].

In all these cases we consider a scenario with two parties, $party_1$, and $party_2$, whereby $party_1$ owns image I and $party_2$ owns image J. The parties wish to collaboratively optimize the image registration problem without disclosing their respective images to each other. We assume that only $party_1$ has access to the transformation parameters $\boldsymbol{\theta}$ and that it is also responsible for computing the update at each optimization step. In what follows, we introduce the basic notation to develop PPIR based on different registration frameworks. We focus on registration methods of increasing complexity, including *(i)* rigid, *(ii)* affine, and *(iii)* non-linear registration based on cubic splines or diffeomorphisms parametrized by time-varying velocity fields (large deformation diffeomorphic metric mapping, LDDMM) [Beg, 2005]. In all these settings, we demonstrate how the registration problem can be naturally adapted for accounting to privacy-preserving operations, and illustrate the effectiveness of PPIR on a variety of registration tasks.



(a) PPIR(MPC) **(b)** PPIR(FHE)-v1

Figure 2.1.: Optimization of SSD loss: proposed framework to compute matrix-vector multiplication $S^T \cdot J$ based on PPIR(MPC) and PPIR(FHE)-v1.

2.2.1 Analysis of classical IR loss functions under a privacy-preserving perspective

In this section we review typical loss functions used in image registration, and analyze the related requirements for privacy-preserving optimization.

2.2.2 Optimization of SSD loss

A typical loss function to be optimized during the registration process is the sum of squared intensity differences (SSD) evaluated on the set of image coordinates:

$$\mathrm{SSD}(I, J, \boldsymbol{\theta}) = \mathrm{argmin}_{\boldsymbol{\theta}} \sum_{\boldsymbol{x}} [I(\boldsymbol{W}_{\boldsymbol{\theta}}(\boldsymbol{x})) - J(\boldsymbol{x})]^2 \tag{2.3}$$

with Jacobian:

$$G = \sum_{\boldsymbol{x}} S(\boldsymbol{x}) \cdot (I(\boldsymbol{W}_{\boldsymbol{\theta}}(\boldsymbol{x})) - J(\boldsymbol{x})), \tag{2.4}$$

where the quantity

$$S(\boldsymbol{x}) = \nabla I(\boldsymbol{x}) \frac{\partial \boldsymbol{W}_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial \boldsymbol{\theta}} \tag{2.5}$$

quantifies image and transformation gradients, and

$$H = \sum_{\boldsymbol{x}} \left(\nabla I(\boldsymbol{x}) \frac{\partial \boldsymbol{W}_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial \boldsymbol{\theta}} \right)^T \left(\nabla I(\boldsymbol{x}) \frac{\partial \boldsymbol{W}_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial \boldsymbol{\theta}} \right) \tag{2.6}$$

is the second order term obtained from Equation (2.3) through linearization [Pennec, 1999; Baker, 2004]. The solution to this problem requires the joint availability of both images I and J, as well as of the gradients of I and of $\boldsymbol{W}_{\boldsymbol{\theta}}$. In a privacy-preserving setting, this information cannot be disclosed, and the computation of Equation (2.2) is therefore impossible. We note that to calculate the update of the registration $\boldsymbol{\Delta\theta}$ of Equation (2.2), the only operation that requires the joint availability of information from both parties is the term $R = \sum_{\boldsymbol{x}} S(\boldsymbol{x}) \cdot J(\boldsymbol{x})$, which can be computed a matrix-vector multiplication of vectorized quantities $R = S^T \cdot J$.

2.2.3 Mutual Information

Mutual Information quantifies the joint information content between the intensity distributions of the two images. This is calculated from the joint probability distribution function (PDF):

$$MI(I, J, \boldsymbol{\theta}) = \text{argmin}_{\boldsymbol{\theta}} - \sum_{r,t} p(r, t; \boldsymbol{\theta}) \log \left(\frac{p(r, t; \boldsymbol{\theta})}{p(r; \boldsymbol{\theta}) p(t)} \right) \tag{2.7}$$

where, given N_r and N_t the maximum intensity for respectively I and J, we define $r \in [0; N_r - 1] \subseteq \mathbb{N}$ and $t \in [0; N_t - 1] \subseteq \mathbb{N}$ as the range of discretized intensity values of I and J, respectively. A Parzen window [Parzen, 1961] is used to generate continuous estimates of the underlying intensity distributions, thereby reducing the effects of quantization from interpolation, and discretization from binning the data. Let $\psi_I^3 : \mathbb{R} \mapsto [0, 1]$ be a cubic spline Parzen window, and let $\psi_J^0 : \mathbb{R} \mapsto [0, 1]$ be a zero-order spline Parzen window. The smoothed joint histogram of I and J [Viola, 1997; Mattes, 2003] is given by:

$$p(r, t; \boldsymbol{\theta}) = \\ \frac{1}{N_{\boldsymbol{x}}} \sum_{\boldsymbol{x}} \psi_I^3 \left(r - \frac{I(\boldsymbol{W}_{\boldsymbol{\theta}}(\boldsymbol{x})) - I(\boldsymbol{W}_{\boldsymbol{\theta}}(\boldsymbol{x}))^\circ}{\boldsymbol{\Delta} b_r} \right) \cdot \psi_J^0 \left(t - \frac{J(\boldsymbol{x}) - J^\circ}{\boldsymbol{\Delta} b_t} \right)$$

In the above formula, the intensity values of I and J are normalized by their respective minimum (denoted by $I(\boldsymbol{W}_{\boldsymbol{\theta}})^\circ$ and J°), and by the bin size (respectively $\boldsymbol{\Delta} b_r$ and $\boldsymbol{\Delta} b_t$), to fit into the specified number of bins (b_r or b_t) of the intensity distribution. The final value for $p(r, t; \boldsymbol{\theta})$ is computed by normalizing by $N_{\boldsymbol{x}}$, the number of sampled voxels. Marginal probabilities are simply obtained by summing along one axis of the PDF, that is, $p(r) = \sum_t p(r, t; \boldsymbol{\theta})$ and $p(t) = \sum_r p(r, t; \boldsymbol{\theta})$. Let the matrices $A_I^3 \in \mathbb{R}^{N_{\boldsymbol{x}} \times N_r}$ and $B_J^0 \in \mathbb{R}^{N_{\boldsymbol{x}} \times N_t}$ be defined as:

$$A_I^3(\boldsymbol{x}, r; \boldsymbol{\theta}) = \psi_I^3 \left(r - \frac{I(\boldsymbol{W}_{\boldsymbol{\theta}}(\boldsymbol{x})) - I(\boldsymbol{W}_{\boldsymbol{\theta}})^\circ}{\boldsymbol{\Delta} b_r} \right)$$

and

$$B_J^0(\boldsymbol{x}, t) = \psi_J^0\left(t - \frac{J(\boldsymbol{x}) - J^\circ}{\boldsymbol{\Delta} b_t}\right),$$

the discretized joint PDF can be rewritten in a matrix form via the multiplication:

$$P = \frac{1}{N_{\boldsymbol{x}}} \cdot (A_I^3)^T \cdot B_J^0. \tag{2.8}$$

The first derivative of the joint PDF is calculated as follows [Dowson, 2006]:

$$\frac{\partial p(r, t; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -\frac{1}{N_{\boldsymbol{x}}} \cdot \sum_{\boldsymbol{x}} B_J^0(\boldsymbol{x}, t) \cdot \frac{\partial \psi_I^3(\epsilon)}{\partial \epsilon} \cdot \frac{\partial \epsilon}{\partial I(\boldsymbol{W}_{\boldsymbol{\theta}}(\boldsymbol{x}))} \cdot \frac{\partial I(\boldsymbol{W}_{\boldsymbol{\theta}}(\boldsymbol{x}))}{\partial \boldsymbol{\theta}}$$

where $\epsilon = \epsilon(\boldsymbol{x}, r; \boldsymbol{\theta}) = r - \frac{I(\boldsymbol{W}_{\boldsymbol{\theta}}(\boldsymbol{x})) - I(\boldsymbol{W}_{\boldsymbol{\theta}})^\circ}{\boldsymbol{\Delta} b_r}$ is the input of the cubic spline. We also introduce the tensor $C_I^3 \in \mathbb{R}^{N_{\boldsymbol{x}} \times N_r \times |\boldsymbol{\theta}|}$ defined as $C_I^3(\boldsymbol{x}, r; \boldsymbol{\theta}) = \frac{\partial \psi_I^3(\epsilon)}{\partial \epsilon} \cdot \frac{\partial \epsilon}{\partial I(\boldsymbol{W}_{\boldsymbol{\theta}}(\boldsymbol{x}))} \cdot \frac{\partial I(\boldsymbol{W}_{\boldsymbol{\theta}}(\boldsymbol{x}))}{\partial \boldsymbol{\theta}}$, to write the discretized first derivative as:

$$P' = -\frac{1}{N_{\boldsymbol{x}}} \cdot (B_J^0)^T \cdot C_I^3, \tag{2.9}$$

where $P' \in \mathbb{R}^{N_t \times N_r \times |\boldsymbol{\theta}|}$.

The Jacobian of the MI is obtained from the chain rule and takes the form:

$$G = P' \log\left(\frac{P}{P_I}\right),$$

while the linearized Hessian [Dowson, 2007] can be written as:

$$H = P'^T P' \left(\frac{1}{P} - \frac{1}{P_I}\right),$$

where $P_I = \sum_t p(r, t, \boldsymbol{\theta})$ is a vector which defines the discretized marginal PDF of the moving image.

The derivatives can be easily calculated from the properties of B-splines since we have $\frac{\partial \psi_I^3}{\partial \epsilon} = \psi_I^2(\epsilon + \frac{1}{2}) - \psi_I^2(\epsilon - \frac{1}{2})$. In a privacy-preserving scenario, to calculate the update of the registration $\boldsymbol{\Delta\theta}$ of Equation (2.7), two operations require the joint availability of information from both parties, which are the matrix P of Equation (2.8) and the matrix P' of Equation (2.9).

2.2.4 Cross Correlation with Advanced Normalization Tools

In the Advanced Normalization Tools (ANTs) introduced by [Avants, 2008], the normalized cross-correlation (CC) loss was specified in the context of diffeomorphic image registration. Let ϕ define a diffeomorphism over the domain $\Omega = \mathbb{R}^d$, parameterized by a time-varying velocity field $\boldsymbol{v}(\boldsymbol{x}, t)$.

In the ANTs setting, inverse consistency is obtained by optimizing the CC loss with respect to both forward and backward (inverse) transformations, here denoted by $\phi_1(\boldsymbol{x}, t)$ and $\phi_2(\boldsymbol{x}, t)$, and parameterized by velocity fields $\boldsymbol{v}_1(\boldsymbol{x}, t)$ and $\boldsymbol{v}_2(\boldsymbol{x}, t)$ respectively. In particular, both images I and J are simultaneously warped towards a "half-way" space, to obtain $I_1 = I(\phi_1(\boldsymbol{x}, 0.5))$ and $J_2 = J(\phi_2(\boldsymbol{x}, 0.5))$. The CC loss is thus defined as:

$$CC(\boldsymbol{x}) = \frac{\sum_{\boldsymbol{x}_i} (\bar{I}(\boldsymbol{x}_i), \bar{J}(\boldsymbol{x}_i))^2}{\sum_{\boldsymbol{x}_i} (\bar{I}(\boldsymbol{x}_i))^2 \sum_{\boldsymbol{x}_i} (\bar{J}(\boldsymbol{x}_i))^2} = \frac{D^2}{EF}$$

where $\bar{I}(\boldsymbol{x}_i) = (I_1(\boldsymbol{x}_i) - \mu_{I_1}(\boldsymbol{x}))$ and $\bar{J}(\boldsymbol{x}_i) = (J_2(\boldsymbol{x}_i) - \mu_{J_2}(\boldsymbol{x}))$ quantify the images appearance at location $\boldsymbol{x}_i$, with respect to the average intensity μ_{I_1} and μ_{J_2} measured in a local window of size M. Coherently with the LDDMM formulation, the variational optimization problem is defined as:

$$\begin{aligned} E_{CC} = \inf_{\phi_1} \inf_{\phi_2} \int_{t=0}^{0.5} \|\boldsymbol{v}_1(\boldsymbol{x}, t)\|_L^2 + \|\boldsymbol{v}_2(\boldsymbol{x}, t)\|_L^2 \, \mathrm{d}t \\ + \int_\Omega CC(\boldsymbol{x}) \, \mathrm{d}\Omega. \end{aligned}$$

where L is a linear operator prescribing a norm on the velocity fields acting as a regularizer. The equation for the derivative of the forward update is given by:

$$\begin{aligned} \nabla_{\phi_1(\boldsymbol{x}, 0.5)} CC(\boldsymbol{x}) = 2L\boldsymbol{v}_1(\boldsymbol{x}, 0.5) + \frac{2D}{EF} \\ \times \left(\bar{J}(\boldsymbol{x}) - \frac{D}{E} \bar{I}(\boldsymbol{x}) \right) |D\phi_1| \nabla \bar{I}(\boldsymbol{x}), \end{aligned} \tag{2.10}$$

while the derivative of the backward update is analogously given by:

$$\begin{aligned} \nabla_{\phi_2(\boldsymbol{x}, 0.5)} CC(\boldsymbol{x}) = 2L\boldsymbol{v}_2(\boldsymbol{x}, 0.5) + \frac{2D}{EF} \\ \times \left(\bar{I}(\boldsymbol{x}) - \frac{D}{F} \bar{J}(\boldsymbol{x}) \right) |D\phi_2| \nabla \bar{J}(\boldsymbol{x}) \end{aligned} \tag{2.11}$$

In privacy-preserving scenario, the sensitive terms carrying private image information are $\frac{2D}{EF}$, $(\bar{J}(\boldsymbol{x}) - \frac{D}{E}\bar{I}(\boldsymbol{x}))$ and $(\bar{I}(\boldsymbol{x}) - \frac{D}{F}\bar{J}(\boldsymbol{x}))$, which must therefore be computed in a privacy-preserving regime.

2.2.5 Building blocks for Secure Computation

After introducing in Section 1 the IR optimization problem and the related functionals addressed in this work, in this section, we review the standard privacy-preserving techniques that will be employed to develop PPIR.

2.2.6 Secure Multi-Party Computation

Introduced by [Yao, 1982], MPC is a cryptographic tool that allows multiple parties to jointly compute a common function over their private inputs (secrets) and discover no more than the output of this function. Among existing MPC protocols, additive secret sharing consists of first splitting every secret s into additive shares $\langle s \rangle_i$, such that $\sum_{i=1}^{n} \langle s \rangle_i = s$, where n is the number of collaborating parties. Each party i receives one share $\langle s \rangle_i$ and executes an arithmetic circuit in order to obtain the final output of the function. In this paper, we adopt the two-party computation protocol defined in SPDZ [Damgård, 2012], whereby the actual function is mapped into an arithmetic circuit and all computations are performed within a finite ring with modulus Q. Additions consist of locally adding shares of secrets, while multiplications require interaction between parties. Following [Damgård, 2012], SPDZ defines: MPCMUL to compute element-wise multiplication, MPCDOT to compute matrix-vector multiplication and MPCMATMUL to compute matrix-matrix multiplication. These operations are performed within an honest but curious protocol.

2.2.7 Homomorphic Encryption

Initially introduced by Rivest et al. in [Rivest, 1978b], Homomorphic Encryption (HE) enables the execution of operations over encrypted data without disclosing either the input data or the result of the computation. Hence, $party_1$ encrypts the input with its public key and sends the encrypted input to $party_2$. In turn, $party_2$ evaluates a circuit over this encrypted input and sends the result, which still remains encrypted, back to $party_1$ which can finally decrypt them. Among various HE schemes, CKKS [Cheon, 2017] supports the execution of all operations on encrypted real values and is considered a levelled homomorphic encryption (LHE) scheme. The supported operations are: SUM ($+$), Element-wise multiplication ($*$) and DOTPRODUCT. With CKKS, an input vector is mapped to a polynomial and further encrypted with a public key in order to obtain a pair of polynomials $c = (c_0, c_1)$. The original function is further mapped into a set of operations that are supported by CKKS, which are executed over c. The performance and security of CKKS depend on multiple parameters including the degree of the polynomial N, which is usually sufficiently large (e.g. $N = 4096$, or $N = 8192$).

2.3 Methods: from IR to PPIR

In this section, we describe the privacy preserving variants of the three IR methods described in Section 2.2.1. We propose two versions of PPIR for SSD according to the underlying cryptographic tool, namely PPIR(MPC), integrating MPC, and PPIR(FHE), integrating FHE. In the case of MI and NCC, we focus on the design of the MPC-variant, due to the non-negligible computational overhead of FHE in these applications. Finally, we also study rigid point cloud registration and describe its privacy-preserving variant in Appendix A.1.

2.3.1 PPIR based on SSD

As mentioned in Section 2.2.2, when optimizing the SSD cost, the only sensitive operation that must be jointly executed by the parties is the matrix-vector multiplication: $R = S^T \cdot J$, where S^T is only known to $party_1$ and J to $party_2$. Figure 2.1 illustrates how cryptographic tools are employed to ensure privacy during registration.

With MPC (Figure 2.1a), $party_1$ secretly shares the matrix S^T to obtain $(\langle S \rangle_1, \langle S \rangle_2)$, while $party_2$ secretly shares the image J to obtain $(\langle J \rangle_1, \langle J \rangle_2)$. Each party also receives its corresponding share, so that $party_1$ holds $(\langle S \rangle_1, \langle J \rangle_1)$ and $party_2$ holds $(\langle S \rangle_2, \langle J \rangle_2)$. The parties execute a circuit with MPCMUL operations to calculate the 2-party dot product between S^T and J. The parties further synchronize to allow $party_1$ to obtain the product and finally to calculate $\boldsymbol{\Delta\theta}$ (Equations (2.4) and (2.6)).

When using FHE (Figure 2.1b), $party_2$ first uses a FHE key k to encrypt J and obtains $[\![J]\!] \leftarrow \textsc{Enc}(k, J)$. This encrypted image is sent to $party_1$, who computes the encrypted matrix-vector multiplication $[\![R]\!]$. In this framework, only vector J is encrypted, and therefore $party_1$ executes scalar multiplications and additions in the encrypted domain only (which are less costly than multiplications over two encrypted inputs). The encrypted result $[\![R]\!]$ is sent back to $party_2$, which can obtain the result by decryption: $R = \textsc{Dec}(k, [\![R]\!])$. Finally, $party_1$ receives R in clear form and can therefore compute $\boldsymbol{\Delta\theta}$.

Thanks to the privacy and security guarantees of these cryptographic tools, during the entire registration procedure, the content of the image data S and J is never disclosed to the opposite party.

2.3.2 PPIR based on MI

In the MPC variant to calculate the joint PDF P in a privacy-preserving manner, the quantity A_I^3 is only known to $party_1$, while the quantity B_J^0 to $party_2$ (Section 2.2.1.2). With reference to Supplementary Figure A.1a, $party_1$ secretly shares matrix $(A_I^3)^T = (\langle A_I^3 \rangle_1, \langle A_I^3 \rangle_2)$, while $party_2$ does the same with $B_J^0 = (\langle B_J^0 \rangle_1, \langle B_J^0 \rangle_2)$. Each party also receives its corresponding share: $party_1$ now holds $(\langle A_I^3 \rangle_1, \langle B_J^0 \rangle_1)$, and $party_2$ holds $(\langle B_J^0 \rangle_2, \langle A_I^3 \rangle_2)$. The parties execute a circuit with MPCMATMUL operation to calculate the 2-party matrix multiplication between $(A_I^3)^T$ and B_J^0. The next operation carried out in the privacy-preserving setting is the computation of the first derivative of Equation (2.9). Supplementary Figure A.1b illustrates the MPC variant, where $party_1$ only knows C_I^3 and $party_2$ only knows B_J^0. Initially, $party_1$ secretly shares the matrix $C_I^3 = (\langle C_I^3 \rangle_1, \langle C_I^3 \rangle_2)$, while $party_2$ does the same with $(B_J^0)^T = (\langle B_J^0 \rangle_1, \langle B_J^0 \rangle_2)$. Each party also receives its corresponding share, namely: $party_1$ holds $(\langle C_I^3 \rangle_1, \langle B_J^0 \rangle_1)$ and $party_2$ holds $(\langle C_I^3 \rangle_2, \langle B_J^0 \rangle_2)$. The parties also execute a circuit with the MPCMATMUL between $(B_J^0)^T$ and C_I^3.

2.3.3 PPIR based on ANTS CC loss

According to Section 2.2.1.3, $party_1$ has access to $\bar{I}$ and E, whereas $party_2$ has access to $\bar{J}$ and F. The computation begins with the optimization of the CC term, specifically the quantity $\frac{2D}{EF}$. In the initial phase, as illustrated in Supplementary Figure A.2a, $party_1$ and $party_2$ secretly share $\bar{I} = (\langle \bar{I} \rangle_1, \langle \bar{I} \rangle_2)$, and $\bar{J} = (\langle \bar{J} \rangle_1, \langle \bar{J} \rangle_2)$, respectively. The subsequent multiplication of these shared values results in the computation of the shares of $D = (\langle D \rangle_1, \langle D \rangle_2)$, which is never reconstructed. In the next step of the protocol, $party_1$ secretly shares $\frac{1}{E}$, and $party_2$ secretly shares $\frac{1}{F}$. Through a multiplication of the shares of D, $\frac{1}{E}$, and $\frac{1}{F}$, both parties collectively obtain the final value of $\frac{2D}{EF}$.

The other two terms that need to be jointly computed are the third term of Equation (2.10) and (2.11), namely $(\bar{J} - \frac{D}{E}\bar{I})$ and $(\bar{I} - \frac{D}{F}\bar{J})$, reported in Supplementary Figure A.2b. In the case of $(\bar{J} - \frac{D}{E}\bar{I})$, $party_1$ secretly shares $\bar{I}$ and $\frac{1}{E}$, while $party_2$ secretly shares $\bar{J}$. Since both parties already have access to the share of D from the previous protocol, they proceed to multiply the secret shares of D, $\frac{1}{E}$, and $\bar{I}$. Subsequently, they subtract the result from $\bar{J}$ to obtain the final value of $(\bar{I} - \frac{D}{F}\bar{J})$. The computation of $(\bar{J} - \frac{D}{E}\bar{I})$ is analogous to the one of $(\bar{I} - \frac{D}{F}\bar{J})$, where $party_2$ secretly shares $\frac{1}{E}$, and both parties participate to the multiplication of D, $\frac{1}{E}$, and $\bar{J}$. The resulting value is finally subtracted from $\bar{I}$, yielding the final result $(\bar{J} - \frac{D}{E}\bar{I})$.

2.3.4 Protocols enhancement for SSD loss

Effectively optimizing Equation (2.1) with MPC or FHE is particularly challenging, due to the computational bottleneck of these techniques when applied to large-dimensional objects [Haralampieva, 2020; Benaissa, 2021], notably affecting the computation time and the occupation of communication bandwidth between parties. Because cryptographic tools introduce a non-negligible overhead in terms of performance and scalability, in this section we introduce specific techniques to optimize the underlying image registration operations.

Gradient sampling

Since the registration gradient is generally driven mainly by a fraction of the image content, such as the image boundaries in the case of SSD cost, a reasonable approximation of Equations (2.4) and (2.6) can be obtained by evaluating the cost only on relevant image locations. This idea has been introduced in medical image registration [Viola, 1997; Mattes, 2003; Sabuncu, 2004], and here is adopted to optimize Equation (2.3) by reducing the dimensionality of the arrays on which encryption is performed. We test two different techniques: *(i)*: Uniformly Random Selection (URS), proposed by [Viola, 1997; Mattes, 2003], in which a random subset of dimension $l \leq d$ of spatial coordinates is sampled at every iteration with uniform probabilities, $p(\boldsymbol{x}) = \frac{1}{d}$; and *(ii)*: Gradient Magnitude Sampling (GMS) [Sabuncu, 2004], which consists of sampling a subset of coordinates with probability proportional to the norm of the image gradient, $p(\boldsymbol{x}) \sim \|\nabla I(\boldsymbol{x})\|$. We note that gradient sampling is not necessary for computing the MI since in Equation (2.8) the computation is already performed on a subsample of the image voxels.

Matrix partitioning in FHE

We now describe additional improvements dedicated to PPIR(FHE) and propose two versions of this solution: *(i)* PPIR(FHE)-v1 implements an optimization of the matrix-vector multiplication by partitioning the vector image into a vector of submatrices, whereas *(ii)* PPIR(FHE)-v2 enhances the workload of the parties by fairly distributing the computation among them.

PPIR(FHE)-v1

We introduce here a novel optimization dedicated to PPIR with FHE, in particular when the CKKS algorithm is adopted. CKKS allows multiple inputs to be packed into a single

ciphertext to decrease the number of homomorphic operations. To optimize matrix-vector multiplication, we propose to partition the image vector J into k sub-arrays of dimension l, and the matrix S^T into k sub-matrices of dimension $|\boldsymbol{\theta}| \times l$. Once all sub-arrays J_i are encrypted, we propose to iteratively apply DOTPRODUCT as proposed by Benaissa et al. [Benaissa, 2021], between each sub-matrix and corresponding sub-array; these intermediate results are then summed to obtain the final result, namely: $[\![R]\!] = \sum_{i=0}^{K} \text{DOTPRODUCT}\left([\![J_i^T]\!], S_i\right) = S^T \cdot [\![J]\!]$.

PPIR(FHE)-v2

In addition to packing multiple inputs into a single ciphertext, the application of FHE to PPIR can be optimized by more equally distributing the workload among the two parties. We note that in PPIR(FHE)-v1, $party_1$ is in charge of computing the matrix-vector multiplication entirely while $party_2$ only encrypts the input and decrypts the result. Following Supplementary Figure A.4, PPIR(FHE)-v2 starts by splitting the matrix S of $party_1$ into two sub-matrices S_1 and S_2 using the operation $split_{h,K}$. This operation partitions the matrix into K equally-sized sub-matrices. Next, the operation $flatten$ is applied to S_1 and S_2, obtaining vectors S_1' and S_2' respectively. Then, $party_1$ encrypts S_2' and sends it to $party_2$, which subsequently applies to its vector J the operation $split_{v,K}$, obtaining J_1 and J_2. This operation splits J into K equally-sized partitions, and it executes the operation $replicate_d$ to J_1 and J_2, obtaining J_1' and J_2' respectively. $party_2$ encrypts J_1' and sends it to $party_1$. Both parties then iteratively perform the element-wise multiplication $*$ and sum up the results of the different partitions using the primitive $Sum_{i,k}$. The protocol doesn't rely anymore on DOTPRODUCT and leads to a significant gain in computational load.

2.4 Experiments & Results

Dataset	Dimension	Modality	Registration Type	Loss function	PETs
2D Point Cloud	193 points	Mono	Rigid	SSD	MPC and FHE-v1
2D Whole body PET	1260 × 1090 voxels	Mono	Affine/Cubic splines	SSD	MPC+URS/GMS, FHE-v1 and v2 + URS/GMS
2D Brain MRI	121 × 121 voxels	Mono	Cubic splines	SSD	MPC, FHE-v1 and v2
3D Brain MRI and PET	180 × 256 × 256 and 160 × 160 × 96 voxels	Multi	Affine	MI	MPC
3D Abdomen MR and CT	192 × 160 × 192 voxels	Multi	Diffeomorphic (ANTs)	CC	MPC

Table 2.1.: Overview of the datasets used in the study. PETs: Privacy Enhancing Technologies.

We demonstrate and assess the different versions of PPIR illustrated in Section 2.2.1 on a variety of image registration problem, namely: *(i)* SSD for rigid transformation of point cloud data, *(ii)* SSD with linear and non-linear alignment of whole body positron emission tomography (PET) data; *(iii)* SSD and MI for mono- and multimodal linear alignment of MRI and PET brain scans; *(iv)* diffeomorphic non-linear registration with CC

of multimodal abdomen data from CT and MRI scans. Experiments are carried out on 2D (mainly for the SSD case) and 3D imaging data. In Table 2.1 is reported an overview of the datasets used specifying their dimensions, modality, registration type, loss functions, and the Privacy Enhancing Technologies (PETs) employed.

2.4.1 Experimental data

Point Cloud Data. We showcase rigid registration on 2D point cloud data representing the corpus callosum, as presented in Vachet et al. [Vachet, 2012], with a set size $n = 193$. The registration loss here considered is SSD between point coordinates (additional details are provided in Appendix A.1).

Whole body PET data. The dataset considered for linear and non-linear registration with SSD consists of 18-Fluoro-Deoxy-Glucose (^{18}FDG) whole body PET scans. The images are a frontal view of the maximum intensity projection reconstruction, obtained by 2D projection of the voxels with the highest intensity across views (1260×1090 pixels). **Brain MRI and PET data.** This dataset regroups brain MRI and PET images obtained from the Alzheimer's Disease Neuroimaging Initiative [Mueller, 2005]. MRI data were processed via a standard processing pipeline to estimate gray matter density maps [Ashburner, 2000]. Non-linear registration was carried out on the extracted mid-coronal slice, of dimension 121×121 pixels. For 3D multimodal linear registration with MI, we use both MRI images and PET images, with respective dimension of $180 \times 256 \times 256$ and $160 \times 160 \times 96$ voxels.

Abdomen MR and CT data. The multimodal dataset Abdomen-MR-CT [Hering, 2022] was used for experiments with ANTs registration based on CC. The data was compiled from public studies of the cancer imaging archive (TCIA) [Clark, 2013] that contains 8 paired scans of MRI and CT from the same patients. The data have an isotropic resolution of $2mm$ and a voxel dimension of $192 \times 160 \times 192$. They also provide 3D segmentation masks for the liver, spleen, and left and right kidney. All scans were pre-aligned by groupwise affine registration.

2.4.2 Experimental Details

In order to avoid local minima and to decrease computation time, we use a hierarchical multiresolution optimization scheme. The scheme involves M resolution steps, denoted as $r_1 \dots r_M$. At each resolution step r_m, the input data is downsampled by a scaling factor m, where $m \in [1 \dots M]$. The quality of PPIR is assessed by comparing the registration results with those obtained with standard registration on clear images (CLEAR). The metrics considered are the difference in image intensity at optimum (for SSD), the total

number of iterations required to converge, and the displacement root mean square difference (RMSE) between CLEAR and PPIR. We also evaluate the performance of PPIR in terms of average computation (running time) and communication (bandwidth) across iterations. In the multimodal Abdomen data, the quality of the registration result was assessed by the overlap across the labeled anatomical regions, quantified by the DICE score.

Point Cloud Data. The registration protocol here adopted is detailed in Appendix A.1. For MPC we set as the prime modulus $Q = 2^{32}$. For PPIR(FHE), we define the polynomial degree modulus as $N = 4096$.

Whole body PET data. Whole-body PET image alignment was first performed by optimizing the transformation $\boldsymbol{W_\theta}$ in Equation (2.1) with respect to affine registration parameters. The multiresolution steps used are r_1, r_5, r_{10}, r_{20}. A second whole-body PET image alignment experiment was performed by non-linear registration, without gradient approximation based on a cubic spline model (one control point every four pixels along both dimensions), with multiresolution steps r_1,r_2,r_5,r_{10},r_{20} and r_{30}. Concerning the PPIR framework, transformations were optimized for both MPC and FHE by using gradient approximation (Section 2.3) using the same sampling seed for each test. For MPC we set as the prime modulus $Q = 2^{32}$. For PPIR(FHE), we define the polynomial degree modulus as $N = 4096$, and set the resizing parameter D to optimize the trade-off between run-time and bandwidth. Since D needs to be a divisor of the image size image data we set $D = 128$.

Brain MRI and PET data. The registration of brain gray matter density images was performed by non-linear registration based on SSD, without gradient approximation, based on a cubic spline model (one control point every five pixels along both dimensions), with multiresolution steps r_1 and r_2. For PPIR(MPC) we use the same configuration defined in the previous section, while for PPIR(FHE) we use the same N and we set $D = 121$.

We tested PPIR with MI for multi-modal 3D affine image registration between PET and MRI brain scans where, in addition to varying the multi-resolution steps, a Gaussian blurring filter is applied to the images with a kernel that narrows as multi-resolution proceeds. The kernel size at different resolutions, denoted with $\sigma_1 \dots \sigma_M$, is used to control the amount of blurring applied to the image at each step of the multi-resolution process. The multiresolution steps applied are r_5 and r_{10}, with 10% of the image's pixels utilized as the number of subsample pixels ($N_{\boldsymbol{x}}$). Gaussian image blurring is applied with a degree of $\sigma_5 = 1$ and $\sigma_{10} = 3$. For MPC we set as the prime modulus $Q = 2^{64}$.

Abdomen MRI and CT data. We tested PPIR with CC for multi-modal 3D ANTs image registration between MRI and CT abdomen scans. The multiresolution steps applied are r_3, r_2 and r_1 and the CC window size $M = 5 \times 5 \times 5$.

Implementation Details

The PPIR framework for the SSD is implemented using two state-of-the-art libraries: `PySyft` [Ryffel, 2018], which provides SPDZ two-party computation, and `TenSeal` [Benaissa, 2021], which implements the CKKS protocol[3].

PPIR based on MI and CC is implemented by extending the `Dipy` framework of Garyfallidis et al. [Garyfallidis, 2014] [4]. Finally, PPIR for point cloud data is released in a separated repository[5].

All the experiments are executed on a machine with an Intel(R) Core(TM) i7-7800X CPU @ (3.50GHz x 12) using 132GB of RAM. For each registration configuration, the optimization is repeated 10 times to account for the random generation of MPC shares and FHE encryption keys.

2.4.3 Results

Point Cloud Data. In Supplementary Table A.1 we present the registration metrics for PPIR(MPC) and PPIR(FHE)-v1. The registration shows that PPIR(MPC) achieves the best results compared to PPIR(FHE), which exhibits not only a longer computation time but also requires higher bandwidth, thanks to its non-iterative algorithm. However, to carry out MatMul, a sufficiently large N (4096) is required, and in this scenario, it leads to a significant loss of chipertext slots compared to the dimension of the point set $n = 193$. Finally, the qualitative results reported in Figure A.7 show negligible differences between point cloud transformed with Clear, PPIR(MPC) and PPIR(FHE)-v1.

Whole body PET data: affine registration (SSD). Table 2.2 compares Clear, PPIR(MPC), PPIR(FHE)-v1 and v2, showcasing metrics resulting from the affine transformation of whole-body PET images. Notably, registration through PPIR(MPC) yields negligible differences compared to Clear in terms of the number of iterations, intensity, and displacement. In contrast, registering with PPIR(FHE) is not feasible when considering entire images due to computational complexity. Nevertheless, Supplementary Figure A.5 shows that neither MPC nor FHE decreases the overall quality of the affine registered

[3] https://github.com/rtaiello/pp_image_registration
[4] https://github.com/rtaiello/pp_dipy/tree/main
[5] https://github.com/rtaiello/ppir_pc

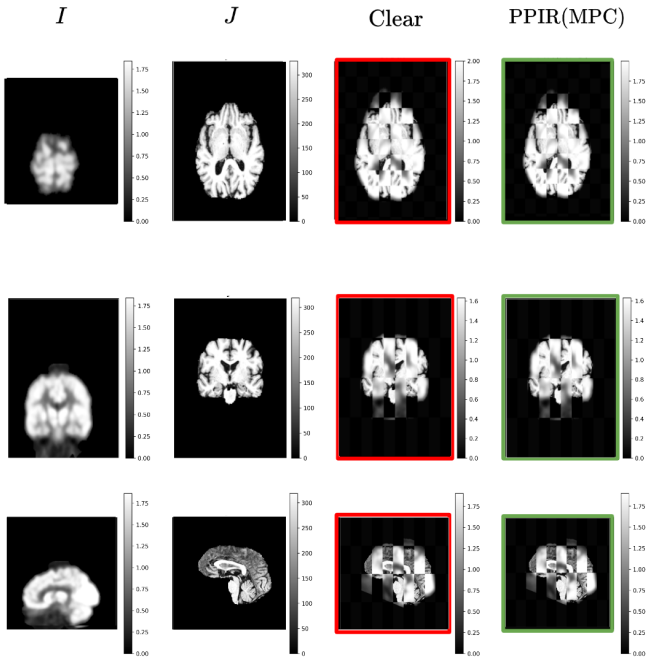


Figure 2.2.: Qualitative results for affine registration with MI over 3D medical images using ADNI dataset [Mueller, 2005]. The images are presented in a 3×4 grid, with the first row representing the axial axis, the second row the coronal axis, and the third row the sagittal axis. In the first column of each row, the moving image obtained using PET modality is shown, while in the second column, the fixed image obtained using MRI modality is displayed. The third column shows the checkerboard alignment result using CLEAR, while the fourth column shows the result using PPIR(MPC). The different protocols are highlighted by red and green frames, respectively.

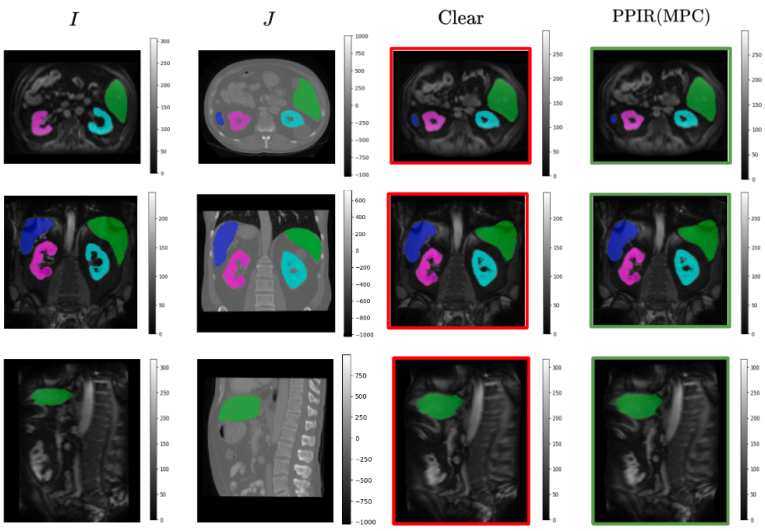


Figure 2.3.: Qualitative results for diffeomorphic registration with CC between 3D medical images from the AbdomenMRCT dataset [Hering, 2022]. The images are presented in a 3×4 grid, with the first row representing the axial axis, the second row the coronal axis, and the third row the sagittal axis. First and second column show respectively MRI and CT images. The third column shows the MRI transformed using CLEAR, while the fourth column shows the MRI transformed using PPIR(MPC). The transformed images are highlighted by red and green frames, respectively.

SSD Affine Registration Metrics			
Solution	**Loss**	**Num. Iteration**	**Metric**
CLEAR	4.34 ± 0.0	118 ± 0.0	-
PPIR(MPC)	4.34 ± 0.0	114.8 ± 4.0	0.13 ± 0.04
CLEAR + URS	4.38 ± 0.0	61 ± 0.0	-
PPIR(MPC) + URS	4.34 ± 0.0	60.4 ± 6.85	1.75 ± 0.19
PPIR(FHE)-v1 ($D = 128$) + URS	4.34 ± 0.10	61.80 ± 4.82	13.47 ± 2.87
PPIR(FHE)-v2 ($D = 128$) + URS	4.34 ± 0.10	61.60 ± 7.21	13.93 ± 4.28
CLEAR + GMS	4.34 ± 0.0	63 ± 0.0	-
PPIR(MPC) + GMS	4.34 ± 0.0	59.80 ± 6.20	0.93 ± 0.42
PPIR(FHE)-v1 ($D = 128$) + GMS	4.34 ± 0.05	60.40 ± 5.12	0.59 ± 0.35
PPIR(FHE)-v2 ($D = 128$) + GMS	4.34 ± 0.05	57.03 ± 4.07	0.50 ± 0.36

SSD Affine Efficiency Metrics				
Solution	**Time (s)**		**Comm. (MB)**	
	$party_1$	$party_2$	$party_1$	$party_2$
PPIR(MPC)	0.13	0.13	1.54	1.54
PPIR(MPC) + URS	0.02	0.02	0.20	0.20
PPIR(FHE)-v1 ($D = 128$) + URS	2.55	0.02	0.06	0.01
PPIR(MPC) + GMS	0.02	0.02	0.20	0.20
PPIR(FHE)-v1 ($D = 128$) + GMS	2.51	0.02	0.06	0.01
PPIR(FHE)-v2 ($D = 128$) + GMS	0.02	0.02	0.73	0.93

Table 2.2.: Affine SSD registration test, comparison between Clear, PPIR(MPC), PPIR(FHE)-v1 and PPIR(FHE)-v2. Registration metrics are reported as mean and standard deviation. Efficiency metrics in terms of average across iterations. RMSE: root mean square error.

images. A comprehensive assessment of the registration results is available in the Appendix. Table 2.2 (Efficiency metrics) shows that PPIR(MPC) performed on full images requires higher computation time and required communication bandwidth compared to PPIR(MPC)+URS/GMS. These numbers improve sensibly when using URS or GMS (by factors $10\times$ and $20\times$ for time and bandwidth, respectively). Concerning PPIR(FHE)-v1, we note the uneven computation time and bandwidth usage between clients, due to the asymmetry of the encryption operations and communication protocol (Figure 2.1). PPIR(FHE)-v2, which shares the computational workload between the two parties and avoids DOTPRODUCT, allows obtaining an important speed-up over PPIR(FHE)-v1 ($100\times$ faster). Notably, this gain is obtained without affecting the quality of the registration metrics, and improves the execution time of PPIR(MPC). On the other side, although PPIR(FHE)-v1 is able to improve communication with respect to PPIR(MPC), it still suffers from the highest communication among the three proposed solutions. This is due to the fact PPIR(FHE) protocols can find different applications depending on the requirements in terms of computational power or bandwidth. **Brain MRI data and whole body PET data: non-linear registration (SSD).** Table 2.3, comparing CLEAR and PPIR(MPC), PPIR(FHE)-v1 and v2, showcases the metrics resulting from spline-based non-linear registration between grey matter density images without the application of gradient approximation. Additionally, the table includes results for the registration between whole-body PET images when the gradient approximation is applied.

SSD Cubic Splines Registration Metrics			
Solution	**Loss**	**Num. Iteration**	**Metric**
CLEAR	0.65 ± 0.0	413 ± 0.0	-
PPIR-MPC	0.65 ± 0.0	345.70 ± 91.22	7.31 ± 1.86
PPIR(FHE)-v1($D = 121$)	0.64 ± 0.0	224.7 ± 79.15	9.50 ± 4.34
PPIR(FHE)-v2($D = 256$)	0.64 ± 0.0	379.2 ± 75.82	11.02 ± 4.93
Clear + URS	0.02 ± 0.0	101 ± 0.0	-
PPIR(MPC) + URS	0.02 ± 0.00	79.3 ± 1.88	5.59 ± 0.39
PPIR(FHE)-v1($D = 128$) + URS	0.02 ± 0.00	105.40 ± 1.71	7.63 ± 0.01
PPIR(FHE)-v2($D = 128$) + URS	0.02 ± 0.00	105.20 ± 2.54	8.74 ± 1.90
Clear + GMS	0.02 ± 0.0	103.00 ± 0.0	-
PPIR(MPC) + GMS	0.02 ± 0.04	80.20 ± 1.62	6.17 ± 0.37
PPIR(FHE)-v1($D = 128$) + GMS	0.02 ± 0.00	105.70 ± 2.40	5.60 ± 2.22
PPIR(FHE)-v2($D = 128$) + GMS	0.02 ± 0.00	106.32 ± 1.30	9.11 ± 2.34

SSD Cubic Splines Efficiency Metrics				
Solution	**Time (s)**		**Comm. (MB)**	
	$party_1$	$party_2$	$party_1$	$party_2$
PPIR-MPC	0.63	0.63	21.47	28.98
PPIR(FHE)-v1($D = 121$)	3.41	0.00	0.06	0.01
PPIR(FHE)-v2($D = 256$)	0.98	0.43	40.45	3.56
PPIR(MPC) + URS	0.41	0.41	8.00	8.00
PPIR(FHE)-v1($D = 128$) + URS	12.23	0.0	0.06	0.01
PPIR(FHE)-v2($D = 128$) + URS	0.62	0.26	24.74	3.37
PPIR(MPC) + GMS	0.41	0.41	8.00	8.00
PPIR(FHE)-v1($D = 128$) + GMS	11.95	0.0	0.06	0.01
PPIR(FHE)-v2($D = 128$) + GMS	0.62	0.26	24.91	3.35

Table 2.3.: Non-Linear SSD registration test comparison between Clear, PPIR(MPC), PPIR(FHE)-v1, and PPIR(FHE)-v2. The registration metrics are reported as mean and standard deviation. Efficiency metrics are in terms of average across iterations. RMSE: root mean square error.

Brain MRI data without gradient approximation. Regarding the registration accuracy, we draw conclusions similar to those of the affine case, where PPIR(MPC) leads to minimum differences with respect to CLEAR, while PPIR(FHE)-v1 seems slightly superior. PPIR(MPC) is associated with a lower execution time and a higher computational bandwidth, due to the larger number of parameters of the cubic splines, which affects the size of the matrix S. Although PPIR(FHE)-v1 has a slower execution time, the demanded bandwidth is inferior to the one of PPIR(MPC), since the encrypted image is transmitted only once. PPIR(FHE)-v2, as in the affine case, outperforms PPIR(FHE)-v1 (still about $100\times$ faster) leading to comparable values between registration metrics and is still inferior to PPIR(MPC). Here, the limitations of PPIR(FHE)-v1 on the bandwidth size are even more evident than in the affine case, since the bandwidth increases according to the number of parameters. This result gives a non-negligible burden to the $party_1$, due to the multiple sending of the flattened and encrypted submatrices of updated parameters. Furthermore, in this case, PPIR(FHE)-v1 performs slightly worse than PPIR(MPC) in terms of execution time.

Whole Body PET Data With Gradient Approximation. Incorporating gradient approximation for handling whole-body PET data leads to similar conclusions as for the experiments on brain data. Qualitative results, reported in Supplementary Figure A.6, show negligible

differences between images transformed with CLEAR+GMS, PPIR(MPC)+GMS, and PPIR(FHE)-v1+GMS.

MI Affine Registration Metrics			
Solution	**Loss**	**Num. Iteration**	**Metric**
Clear	0.22 ± 0.00	213 ± 0.0	-
PPIR(MPC)	0.22 ± 0.00	264 ± 0.0	0.41 ± 0.04

MI Affine Efficiency Metrics				
Solution	**Time (s)**		**Comm. (MB)**	
	$party_1$	$party_2$	$party_1$	$party_2$
PPIR(MPC)	1.02	1.02	15.00	15.00

Table 2.4.: Affine MI registration test, comparison between Clear and PPIR(MPC). Registration metrics (Loss = Mutual Information and metric = displacement RMSE (mm))are reported as mean and standard deviation. Efficiency metrics are reported as average across iterations.

Brain MRI and PET data: affine registration (MI). Table 2.4 provides information on the computation cost of the protocol and the registration metrics for both the joint PDF and the First Derivative of the Joint PDF, using both CLEAR and PPIR(MPC). The results demonstrate a reasonable execution time (completed in under 5 minutes for the entire process) and noteworthy data transfer, totaling less than $4GB$. Qualitative results for the image registration are shown in Figure 2.2, indicating that there is no difference between the moving transformed using CLEAR and PPIR(MPC).

CC ANTs Registration Metrics				
Solution	**Initial DICE score**	**Final DICE score**	**Num. Iteration**	**Displacement RMSE (mm)**
Forward				
Clear	0.54 ± 0.13	0.68 ± 0.19	26 ± 0.0	-
PPIR(MPC)		0.67 ± 0.19	26 ± 0.0	0.22 ± 0.04
Backward				
Clear	0.54 ± 0.13	0.69 ± 0.19	26 ± 0.0	-
PPIR(MPC)		0.68 ± 0.19	26 ± 0.0	0.22 ± 0.04

CC ANTs Efficiency Metrics				
Solution	**Time (s)**		**Comm. (MB)**	
	$party_1$	$party_2$	$party_1$	$party_2$
PPIR(MPC)	24.00	24.00	152.25	152.25

Table 2.5.: ANTs registration with CC, comparison between Clear and PPIR(MPC). Registration metrics are reported as mean and standard deviation. Efficiency metrics are reported as average across iterations. RMSE: root mean square error.

Abdomen MRI and CT data: diffeomorphic non-linear registration (CC). Table 2.5 presents the metrics for ANTs registration using both CLEAR and PPIR(MPC) methods. The initial DICE scores for both forward and backward transformations are consistent between CLEAR and PPIR(MPC), with slight variations in the final DICE scores. The number of iterations and displacement RMSE values also exhibit similar trends. In terms of communication and computation times, PPIR(MPC) demonstrates comparable performance with CLEAR, showcasing its feasibility for secure registration. The computation

times and communication bandwith between the two parties are well within reasonable limits. These qualitative result in Figure 2.3 indicates that the proposed PPIR(MPC) solution maintains the quality of registration metrics while ensuring secure and private communication between parties.

2.5 Discussion

This current work builds upon Taiello et al. [Taiello, 2022] by extending its application to include MI with linear registration (3D images), CC using the ANTs framework (3D images), enhancing the FHE for the SSD loss function, namely PPIR(FHE)-v2, and also integrating rigid point cloud registration. We recognize specific limitations associated with the use of FHE in our PPIR framework, which limited the effective use of this techniques besides the optimization of the SSD loss. FHE's computational cost during homomorphic operations poses challenges, limiting the scalability and real-time applicability of PPIR(FHE). This is particularly evident when dealing with large datasets, such as 3D images, or when employing advanced image registration cost functions that demand significant computational resources. To address this gap, researchers are actively exploring the optimization of FHE through the integration of hardware accelerators [Boemer, 2021].

For the SSD loss function, we provide comparison experiments with both URS and GMS [Viola, 1997; Mattes, 2003; Sabuncu, 2004] for sake of completeness and compatibility with subsampling approaches in IR. We recognized that URS doesn't bring substantial improvements with respect to GRS, and this latter method should be preferred in the considered application or testing scenario.

While PPIR focuses on the privacy-preserving formulation of classical image registration methods based on gradient-based optimization, throughout the past years the research community has been steering the attention towards deep learning (DL)-based image registration [Simonovsky, 2016; Krebs, 2017; Yang, 2017; Balakrishnan, 2019]. Among the medical imaging application of privacy-preserving methodologies, Kaissis et al. [Kaissis, 2021] discussed privacy-preserving FL with Secure Aggregation [Bonawitz, 2017b] and Differential Privacy [Abadi, 2016] for 2D medical image classification tasks. However, as highlighted by [Kaissis, 2021], deploying DL models for privacy-preserving inference nowadays is predominantly achievable through Multi-Party Computation (MPC). This process necessitates multiple servers and incurs significant overhead, primarily attributed to the size of the DL model, especially when handling 3D image registration tasks within a DL-based framework [Balakrishnan, 2019]. To the best of our knowledge both DL and non-DL registration methods available in the literature do not satisfy the PPIR require-

ments investigated in our work, as they always require the disclosure of the target and moving images in clear.

2.6 Conclusion and future works

This study introduces the novel paradigm of Privacy Preserving Image Registration, designed for allowing image registration in privacy-preserving scenarios where images are confidential and cannot be shared in clear. Leveraging both secure multi-party computation (MPC) and Fully Homomorphic Encryption (FHE), we propose in PPIR effective strategies integrating cryptographic techniques into a variety of state-of-the-art registration frameworks, encompassing different parameterization and loss functions. We evaluate the framework's performance across various registration benchmarks, conducting quantitative and qualitative assessment for all the considered image registration problems. Our future direction involve extending PPIR to encompass additional cost functions commonly used in image registration, aiming to enhance the framework's versatility and applicability.

3 Secure Aggregation for Real-World Applications of Federated Learning in Healthcare

Contents

3.1 Related Works **36**

3.2 Methods **37**

3.3 Evaluation **39**

3.3.1 Experimental evaluation: 40

3.4 Conclusion and Future Works **42**

In this chapter, we address the challenges of deploying federated learning (FL) in real-world scenarios, particularly within the healthcare sector, focusing on communication and security. A key aspect of FL is the federated aggregation procedure, where secure aggregation (SA) schemes are studied to provide privacy guarantees over the model parameters transmitted by clients. However, the practical application of SA in existing FL frameworks is limited due to computational and communication bottlenecks.

To bridge this gap, we explore the implementation of SA within the open-source Fed-BioMed framework. We implement and compare two SA protocols, Joye-Libert (JL) and Low Overhead Masking (LOM), providing extensive benchmarks across various healthcare data analysis problems. Our theoretical and experimental evaluations on four datasets demonstrate that SA protocols effectively protect privacy while maintaining task accuracy. The computational overhead during training is less than 1% on a CPU and less than 50% on a GPU for large models, with protection phases taking less than 10 seconds. Integrating SA into Fed-BioMed impacts task accuracy by no more than 2% compared to non-SA scenarios.

Overall, this study demonstrates the feasibility of SA in real-world healthcare applications and contributes to narrowing the gap towards adopting privacy-preserving technologies in sensitive applications.

This chapter has been accepted at the 5th Workshop on Distributed, Collaborative, and Federated Learning, in conjunction with MICCAI 2024.

Federated Learning (FL) is a distributed machine learning paradigm that enables multiple clients to collaboratively train a global model without sharing their local datasets. While researchers have largely focused in developing FL theories and methods in a variety of applications, the deployment of FL in real-world scenarios is still challenging, particularly in terms of communication protocols, security, and customization bottlenecks.

A critical requirement for real-world applications of FL concerns the protection of the model's parameters shared by the clients during model aggregation. To this end, privacy-preserving methodologies such as Secure Aggregation (SA) [Mansouri, 2023] are currently under study, to guarantee that aggregated data shared among participants does not reveal individual contributions.

Compared to other privacy-enhancing technologies like Differential Privacy [Dwork, 2006], which requires only minor adjustments to the federated aggregation process through the injection of noise to the model's parameters, implementing SA in production is more complex as it requires changes to the standard operational flow of the FL framework by incorporating new communication phases. As a result, the adoption of SA in currently available FL software frameworks is lagging behind. Existing SA solutions primarily target settings with a large number of clients, where hardware limitations can lead to protocol execution failures. Some preliminary solutions have been proposed in the framework FLOWER [Beutel, 2020], which however introduce a non-negligible overhead. The approach provided by NVFLARE is simpler but suffers from a weak security model [Roth, 2022]. Finally SA in OPENFL [Reina, 2021] requires dedicated hardware solutions. Overall, the applications of these SA protocols in the cross-silo healthcare setting is suboptimal, due to the limited number of clients, and their general availability as compared to the cross-device setting.

To address these limitations, in this work we explore the implementation of SA schemes optimally customized for cross-silo healthcare applications. In particular, we study the two suitable categories of SA based on masking and additively homomorphic encryption [Mansouri, 2023]. We identify respectively LOW OVERHEAD MASKING [Kursawe, 2011] and JOYE-LIBERT [Joye, 2013] as the most relevant solutions for our application. These protocols are designed to protect individual updates from being exposed during the aggregation process.

This work is based on theoretical and experimental evaluation of these SA protocols within the Fed-BioMed framework [Cremonesi, 2023]. In particular, we conducted a comprehensive comparison on four distinct medical datasets including medical images and tabular data: Fed-IXI [Ogier du Terrail, 2022], Fed-Heart [Ogier du Terrail, 2022], REPLACE-BG [Aleppo, 2017], and FedProstate [Innocenti, 2023]. We measured the computational resources required for training, encryption, and overall execution time. When training was performed on a CPU, we achieved a total computation overhead of

less than 1%, while on a GPU, for larger machine learning models ($> 5M$ parameters), the overhead was less than 50%, with a protection phase that took less than 10 seconds. Furthermore, we analyzed the impact of SA on task accuracy, demonstrating that incorporating SA into Fed-BioMed affects accuracy by no more than 2% compared to non-SA scenarios. Overall this study demonstrates the feasibility of SA in real-world healthcare applications and contributes in reducing the gap towards the adoption of privacy-preserving technologies in sensitive applications.

Federated Learning.

As introduced by McMahan et al. [McMahan, 2017a], FL consists of a distributed machine learning paradigm where a group of clients, denoted as $\mathcal{U}$, collaboratively trains a global model with parameters $\vec{\theta} \in \mathbb{R}^d$, under the guidance of a FL server. One of the first and popular methods used to train a FL model is the FedAvg scheme [McMahan, 2017a]. With FedAvg, at each FL round denoted by τ, each client $u \in \mathcal{U}$ trains the model $\vec{\theta}_{u,\tau}$ on the private local data $\mathcal{D}_u$, for example through Stochastic Gradient Descent (SGD) [Bottou, 2004]. Upon completion of the local training, each client forwards its updated model $\vec{\theta}_{u,\tau}$ to the server and the local dataset size $w_u = |\mathcal{D}_u|$. When the server receives the updated models from all participating clients, it proceeds to the weighted aggregation step:

$$\vec{\theta}_{\tau+1} \leftarrow \frac{\sum_{\forall u \in \mathcal{U}} w_u \vec{\theta}_{u,\tau}}{\sum_{\forall u \in \mathcal{U}} w_u}.$$

This iterative process continues until the global model $\vec{\theta}$ reaches some desired level of accuracy. The presence of a large number of FL clients significantly impacts the communication overhead. To mitigate this, instead of involving all clients in the training, at each FL round, the server selects a subset of clients (*client selection* [McMahan, 2017a]), denoted as $\mathcal{U}^{(\tau)} \subseteq \mathcal{U}$, with $|\mathcal{U}^{(\tau)}| = n$, and collects their parameters only for aggregation.

Secure Aggregation.

SA [Mansouri, 2023] typically involves multiple *users* and a single *aggregator*. Each user possesses a private input, and the role of the aggregator is to calculate the sum of these inputs. A property of SA is that the aggregator learns nothing more than the aggregated sum, thereby preserving the privacy of individual user inputs.

SA has found significant applications in Federated Learning (FL), where it is used to securely aggregate the updated model parameters received from FL clients (aligned with the *user's* concept in SA) during each FL round, by instantiating an FL server (*aggregator*

in the context of SA). The adoption of SA is motivated by the potential threats posed by adversaries having access to the client's updated model $\vec{\theta}_{u,\tau}$ which may infer information about its private dataset $\mathcal{D}_u$ [Nasr, 2019; Shokri, 2017]. Hence, the local models should remain confidential even against the FL server. SA in FL was first developed by Bonawitz et al. [Bonawitz, 2017a]. The protocol considered in that study faced two different challenges:

• *Threat models* defining the potential risks and behaviors that the security protocol is designed to protect against. The primary threat scenarios in SA include the honest-but-curious model where parties (server and clients) follow the protocol without tampering with the data but may attempt to infer additional information.

• *Client dropouts*, caused by factors such as connectivity issues or voluntary withdrawal, are common in real-world federated learning environments. Dropouts can significantly impact the computation and number of communication rounds of the SA protocol, as they often require the participation of all selected clients within a training round. With communication rounds, we refer to the number of interactions required between the clients and the server to complete a particular phase of the protocol.

3.1 Related Works

In real-world deployments, only a few FL frameworks implement some form of SA: OpenFL[Reina, 2021], NVFlare[Roth, 2022], and Flower [Beutel, 2020].

Flower implements SecAgg+ [Bell, 2020], a masking-based protocol that ensures security in the honest-but-curious model. This protocol requires four communication rounds and uses Shamir's Secret Sharing to recover missing masks in case of client dropout, ensuring the server can complete the aggregation. Compared to the SA schemes here introduced in Fed-BioMed, Flower's approach is more costly in terms of communications, albeit accommodating for client dropout.

NVFlare introduces an SA method that leverages the CKKS asymmetric homomorphic encryption scheme [Cheon, 2017]. This threat model is considered weaker than typical state-of-the-art protocols because it requires clients to share a common secret key and assumes clients are honest. Clients protect their inputs using a public key, while the server, operating under the honest-but-curious model, aggregates these inputs and returns the aggregate to each client for decryption using the same secret key. This approach requires one communication round and allows client dropout.

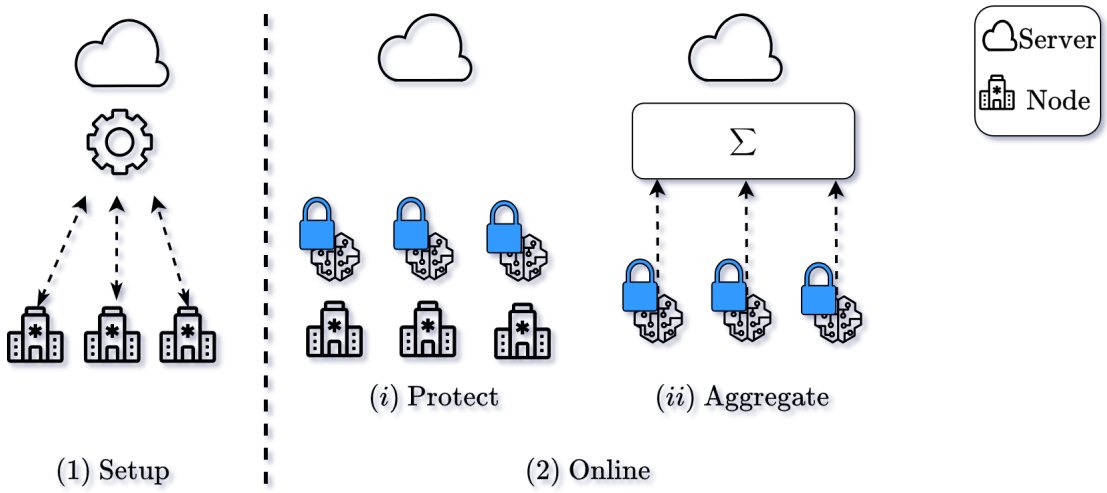


Figure 3.1.: Overview of Secure Aggregation phases.

OpenFL's use of Trusted Execution Environments (TEEs) represents a further step in sandboxing and securing local computations, but requires specific hardware which may not be available in typical FL studies involving hospitals.

3.2 Methods

In this section, we detail the implementation in Fed-BioMed of the two SA protocols, Joye-Libert (JL) and Low Overhead Masking (LOM). From this point on, we adopt the terminology of Fed-BioMed, where a client is referred to as a node.

A general overview of SA is depicted in Figure 3.1, and a more detailed scheme is provided in Supplementary Figure B.2. An SA protocol comprises two phases: **setup** and **online**. The setup phase, illustrated in Figure 3.1.1, is executed among all participating nodes in $\mathcal{U}$ before the FL training. This step ensures that all parties have the appropriate cryptographic material necessary to run the specific SA protocol.

The online phase, Figure 3.1.2 is repeated during each FL round τ and consists of two steps: (i) *protect* and (ii) *aggregate*. In the *protection* step, each node protects its private local model using specific SA primitives and then sends the protected model to the server. In the *aggregation* step, the server receives the protected local models, computes the aggregate, and then decrypts it. To ensure the correct functioning of the cryptographic primitives, the locally-trained model vector of each node must be quantized beforehand.

Prerequisites.

To perform FedAvg with SA, we first convert the node's local parameters $\vec{\theta}_{u,\tau} \in \mathbb{R}^d$ into integers $\mathbb{Z}_{2^L}^d$, where L represents the maximum number of bits of the plaintext. This conversion is achieved by applying uniform quantization, defined as: $Q(\vec{\theta}_{u,\tau}) = \left\lfloor \frac{2^L \cdot (\vec{\theta}_{u,\tau} - \theta_{\min})}{(\theta_{\max} - \theta_{\min})} \right\rceil$. Here, $\lfloor \cdot \rceil$ denotes the standard rounding function. To ensure that real values are within a desired range, we apply a clipping function, $\text{clip}(x, \theta_{\min}, \theta_{\max}) = \min(\max(x, \theta_{\min}), \theta_{\max})$, where $\theta_{\min}$ and $\theta_{\max}$ are the lower and upper bounds, respectively.

To apply weighted averaging over the integers, we assume that $w_u \in \mathbb{Z}_{2^{W_u}}$, where W_u is the number of bits to represent the node's dataset size, and we define $W = \max(\{W_u\}_{\forall u \in \mathcal{U}})$.

The weighted local model is computed as $\vec{x}_{u,\tau} = Q(\vec{\theta}_{u,\tau}) \cdot w_u$, resulting in $\vec{x}_{u,\tau} \in \mathbb{Z}_{2^{L+W}}^d$. To avoid overflow, we define $M = L + W + \log_2(n)$ as the maximum number of bits for sum computation. The aggregate $\vec{x}_\tau = (\sum_{u \in \mathcal{U}^{(\tau)}} \vec{x}_{u,\tau}) \in \mathbb{Z}_{2^M}^d$ is then divided by $s = \sum_{u \in \mathcal{U}^{(\tau)}} w_u$ and dequantized using the following formula: $\vec{\theta}_{\tau+1} = Q^{-1}(\vec{x}_\tau) = \vec{x}_\tau \cdot \frac{(\theta_{\max} - \theta_{\min})}{2^L} + \theta_{\min}$.

In this context, we assume that quantization has been performed and omit the details of the dequantization process in the protocol explanation.

Joye-Libert

In Supplementary Figure B.2a, we illustrate the Joye-Libert (JL) implementation in Fed-BioMed. During the **setup** phase, the participating nodes $\mathcal{U}$ generate their private keys sk_u, and the server creates its server key sk_0 — which is the sum of the node keys — using Shamir Secret Sharing (SS) [Shamir, 1979].

During the **online** phase, the protection and aggregation are applied as described in JL (Section 4 [Joye, 2013]). In the protection step, each node uses a private secret key sk_u at FL round τ with a one-time mask derived from sk_u and τ, to obtain a protected local model through modular exponentiation over a large modulus N. Using the server key sk_0, the server can recover the aggregate of the nodes' private local models in clear.

Our JL implementation works with vectors; the protection and aggregation algorithms are applied element-wise. We use the element's index i to generate a unique FL round (need to guarantee a one-time mask) for each element in the vector. For instance, to protect $\vec{x}_{u,\tau}$, we execute protect and aggregate over the FL round $\tau || i$ and input $\vec{x}_{u,\tau}[i]$, where $\vec{x}_{u,\tau}[i]$ represents the i-th element of the vector $\vec{x}_{u,\tau}$.

The computation and communication of the protected local model is optimized by using vector encoding [Mansouri, 2022].

Software details.

SS is integrated into Fed-BioMed using MP-SPDZ library [Keller, 2020]. The modulus N is provided by the server, and the modular operations are performed using the GMPY2[1] Python library.

Low Overhead Masking

The second implementation, Low Overhead Masking (LOM) [Kursawe, 2011], which supports client selection, is depicted in Figure B.2b. During the **setup** phase, all participating nodes $\mathcal{U}$ establish a pairwise secret $s_{u,v}$, such that $s_{u,v} = s_{v,u}$, with all nodes through the Diffie-Hellman Key Agreement (KA) [Diffie, 2022], which will be used in the protect step.

In the **online** phase, during the *protection* step, a selected node $u \in \mathcal{U}^{(\tau)}$ runs the protect algorithm (Section 3.4 [Kursawe, 2011]). This algorithm protects the local model with a one-time mask derived through a Pseudo-Random Function (PRF) which uses the pairwise secret with the selected nodes $\mathcal{U}^{(\tau)}$ and the current FL round τ, and sends the protected local model to the server. The server then sums the protected local models and collects the final aggregate $\vec{x}_\tau$.

Software details.

Diffie-Hellman KA and PRF are implemented in the CRYPTOGRAPHY[2] Python library, with ECDH and the ChaCha20[Bernstein, 2008] stream cipher, respectively. Distribution of the DH public key is assumed outside of Fed-BioMed, offline, or trough a Public Key Infrastructure.

3.3 Evaluation

In this Section we provide our theoretical and experimental evaluation of the two implemented SA protocols.

[1] GMPY2: https://gmpy2.readthedocs.io/
[2] CRYPTOGRAPHY: https://github.com/pyca/cryptography

Complexity Analysis: JL's node computation is $O(d)$, independent of the number of selected nodes, but requires modular exponentiation, and node communication for vector encoding is $O(d \cdot 2 \cdot M)$ [Mansouri, 2022]. The server's computation is $O(n+d)$, involving n multiplications and d exponentiation [Joye, 2013].

LOM's node computation is $O(nd)$, dependent on the number of selected nodes, using faster modular addition and PRF evaluation. The server's computation involves nd modular additions, and node communication is $O(d \cdot M)$ [Kursawe, 2011].

3.3.1 Experimental evaluation:

The experimental evaluation consists of tracking the computation time between JL and the LOM. We carried out the experiments by considering varying FL hyper-parameters represented by the number of total nodes n_{tot}, the number of selected nodes n, the number of FL rounds T, the number of local SGD steps e, the batch size b and the learning rate η. For SA, the hyper-parameterswe explored were the number of bits input L, the number of bits weight W. Moreover, we fixed the aggregation number of bits $M = 32$ and the clipping range min and max. Finally, we report the hardware used to train ML model. We report all this information for each experiments in Table 3.1.

We use four medical datasets to evaluate the task accuracy of our SA implementations over the aggregated global model at each FL round, using a dedicated tasks-specific test set, and tracking the required computational resources for the nodes.

The four datasets are:

• Fed-Heart [Ogier du Terrail, 2022], providing patients' demography and clinical history from four hospitals. The task is to predict the clinical status of a patient (binary classification from tabular data). For FL training and testing we follow [Ogier du Terrail, 2022], and the target evaluation metric is the balanced accuracy.

• Fed-IXI [Ogier du Terrail, 2022], is composed by T1 and T2 brain magnetic resonance images (MRIs) from three hospitals. The task is supervised brain segmentation, and ground truth segmentations are provided. For FL training and testing we follow [Ogier du Terrail, 2022], and the target evaluation metric is the dice score.

• REPLACE-BG dataset [Aleppo, 2017] was obtained from a cohort of 202 participants. The task is prediction of blood glucose levels for the subsequent hour based on data from the last three hours, including glucose levels, insulin boluses, and CHO content.

• FedProstate dataset [Innocenti, 2023] provides T2 MRIs of the whole prostate from three publicly available datasets, and the task is supervised prostate segmentation. We defined the splitting criteria into different clients, the pre-processing methods, and the FL training and testing parameters coherently with [Innocenti, 2023].

Supplementary Table B.1 reports the dataset details, the FL and SA hyper-parameters, and the hardware specific for model training across experiments. The code is publicly available[3].

SA	Time Training (s)	Time Encrypt (s)	Time Total (s)
FedIXI ($d = 246K; n_{\text{tot}} = n = 3$)			
JL	68.10 ± 2.17	52.21 ± 0.85	121.48 ± 2.50
LOM	46.51 ± 1.39	0.62 ± 0.14	48.22 ± 1.03
FedHeart ($d = 258; n_{\text{tot}} = n = 4$)			
JL	0.24 ± 0.08	0.08 ± 0.01	0.68 ± 0.09
LOM	0.20 ± 0.09	> 0.01	0.59 ± 0.08
REPLACE-BG ($d = 256K; n_{\text{tot}} = 180; n = 18$)			
JL	N/A	N/A	N/A
LOM	53.72 ± 8.61	0.39 ± 0.06	57.42 ± 6.95
FedProstate ($d = 7.4M; n_{\text{tot}} = n = 4$)			
JL	N/A	> 300	> 300
LOM	7.65 ± 1.6	9.22 ± 0.38	23.86 ± 2.1

Table 3.1.

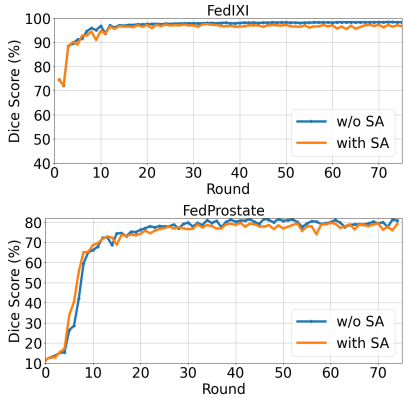


Figure 3.2.

Table 3.1 Comparison of node average computation time across different SA protocols using four medical datasets. Each dataset is characterized by the total number of nodes (n_{tot}), the number of selected nodes (n), and the size of the local model (d). Fig. 3.2 Compare the task accuracy of the global model at each FedAvg aggregation with and without applying SA for FedIXI and FedProstate. The SA is characterized by L bits for representing the local model, W bits for representing the maximum dataset size, and the specified maximum and minimum clipping range.

In Table 3.1, we report the required node's computational resources comparing the two SA solutions. We present the average training time, encryption time, and total time. LOM consistently outperforms JL due to its faster underlying primitive (modular addition vs. modular exponentiation). Specifically, in all experiments where training runs on a CPU, LOM accounts for less than 1% of the total time. When a GPU (e.g., in FedProstate) is available, the overall encryption time is around 40% of the total time, considering a large input parameter dimension of $d = 7.4M$.

In Figure 3.2, we display the task accuracy comparison with and without SA for FedIXI and FedProstate. Additionally, Appendix Figure B.1 presents the results for FedHeart and REPLACE-BG. These figures demonstrate that incorporating SA in Fed-BioMed affects the accuracy by no more than 2% compared to the case without SA.

[3] Anonymous GitHub code

3.4 Conclusion and Future Works

We have demonstrated that SA can be effectively implemented within the Fed-BioMed framework to enhance privacy in federated learning. Our evaluations using four medical datasets show that both Joye-Libert and Low Overhead Masking protocols protect privacy while maintaining task accuracy. The computational overhead is minimal, making SA a viable option for real-world deployments. As part of future work, we plan to replace MP-SPDZ with a direct implementation of additive secret sharing within Fed-BioMed. We also aim to replace JL with a quantum-resistant SA [Brakerski, 2011] using the SHELL C++ library [4].

[4] SHELL library: https://github.com/google/shell-encryption/

4 Let Them Drop: Scalable and Efficient Federated Learning Solutions Agnostic to Client Stragglers

Contents

4.1 Introduction **44**
4.2 Building Blocks **46**
4.2.1 Joye-Libert Secure Aggregation Scheme 46
4.2.2 Threshold Joye-Libert SA Scheme 47
4.3 Eagle in SyncFL **48**
4.4 Owl in AsyncFL **53**
4.5 Related Work **55**
4.6 Complexity Analysis **57**
4.6.1 **Eagle** 57
4.6.2 **Eagle** with decryptors 58
4.6.3 **Owl** 59
4.7 Experimental Study **60**
4.7.1 Experimental Setting 60
4.7.2 Eagle 61
4.7.3 **Eagle** vs. **SecAgg** and **FTSA** 61
4.7.4 **Eagle** vs. **Flamingo** 62
4.7.5 **Owl** 63
4.8 Conclusion **65**

In this chapter, we explore innovative solutions for Secure Aggregation (SA) in Federated Learning (FL) systems, which are crucial for collaboratively training global machine learning models while preserving the privacy of individual clients' local datasets. Existing SA protocols in FL literature often operate synchronously, leading to significant runtime slowdowns due to the presence of stragglers, or late-arriving clients. To mitigate this issue, one common approach treats stragglers as client failures and utilizes SA solutions robust against dropouts. However, this method negatively impacts protocol performance, as the cost is highly dependent on the dropout ratio, which increases significantly when

stragglers are considered. Another strategy explored in the literature is introducing asynchronicity into the FL system, but the existing asynchronous SA solutions suffer from high overhead. In this work, we propose to manage stragglers as client failures while designing SA solutions that do not rely on the dropout ratio, ensuring that an inevitable increase in this metric does not affect performance. We first introduce **Eagle**, a synchronous SA scheme that depends solely on the inputs of online users, rather than client failures. This approach offers better computation and communication costs compared to existing solutions in realistic scenarios with a high number of stragglers. We then propose **Owl**, the first SA solution suitable for asynchronous settings, which again considers only the contributions of online clients.

We implement both solutions and demonstrate that: *(i)* in synchronous FL with realistic dropout rates, **Eagle** outperforms the best SA solution, **Flamingo**, by a factor of four; *(ii)* in the asynchronous setting, **Owl** exhibits superior performance compared to the state-of-the-art solution, **LightSecAgg**.

This chapter has been accepted at ARES 2024 [Taiello, 2024b].

4.1 Introduction

Federated Learning (FL) [McMahan, 2017a] is a popular framework enabling multiple clients to collaborate in training a common machine learning model without sharing their local data. In centralized FL, the primary server initializes the parameters of a global model and sends them to the clients for optimization with respect to the local data. The locally trained parameters are transmitted to the server and aggregated (e.g. through weighted averaging) to produce a new global model for the next FL round.

Recent studies [Nasr, 2019; Shokri, 2017] show that even sharing local model parameters may expose some information about the clients' training data, through various attacks such as membership inference or model inversion. A popular solution to tackle such attacks is Secure Aggregation (SA), which ensures that the global model's parameters are computed through the aggregation of the individual ones without disclosing them individually. Informally, each client first protects its local parameters and sends them to the server, and the server computes the aggregated parameters and shares them back to all the clients. The underlying privacy protection technique usually consists of either secure masking, additively homomorphic encryption, or differential privacy mechanisms [Mansouri, 2023].

As pointed out in [Mansouri, 2023], initial SA solutions were strongly relying on the online presence of all FL clients and even a single client failure, referred to as *client*

dropout, was resulting in the complete failure of the aggregation protocol. To cope with this problem of robustness, many works [Bonawitz, 2017a; Mansouri, 2022; So, 2021; Kadhe, 2020; Bell, 2020; Ma, 2023] propose to initially secret share clients' keying material with the others so that whenever a client failure occurs, the remaining online clients can collaborate to reconstruct the dropped client's material and complete the aggregation operation correctly.

While these solutions have indeed been proven robust against client dropouts, their security is only valid when clients are synchronized and share their parameters on an FL-round basis. Unfortunately, a synchronous FL (SyncFL) setting encounters challenges in heterogeneous environments whereby slow, late-arriving clients, known as *stragglers* [Xie, 2019; Nguyen, 2022; Fraboni, 2023], can be detrimental to the overall system performance.

Very few solutions under SyncFL settings, namely [Bonawitz, 2019b; Yang, 2018], address this challenge and employ a technique known as *over-selection*. In this approach, a larger pool of clients is initially engaged so that potential stragglers are inherently avoided. If this approach is adopted in the context of SA in SyncFL, then the dropout rate needs to be set as the sum of the potential ratio of stragglers and the actual client failures. In practical FL deployments, a dropout rate, including the ratio of stragglers, is expected to be around 30% [Bonawitz, 2019b; Nguyen, 2022]. Unfortunately, this non-negligible ratio will result in a significant increase in SA parameters and consequently in a significant overhead both at the server [Bell, 2020; Bonawitz, 2017a] and at the client [Ma, 2023; Mansouri, 2022].

Asynchronous FL (AsyncFL) [Xie, 2019; Fraboni, 2023; Nguyen, 2022] modifies SyncFL by taking into account clients' model updates as soon as they arrive to the server. This allows to leverage the impact of stragglers, that do not block the system. Nevertheless, as previously mentioned in [So, 2022], existing SA solutions become insecure in such AsyncFL settings.

Contributions

	Client Comp.	Client Comm.	Online Rounds	FL Type
SecAgg [Bonawitz, 2017a]	$O(n^2 + nd)$	$O(n + d)$	4	SyncFL
FTSA [Mansouri, 2022]	$O(n^2 + n\log(n)d)$	$O(n + d)$	3	SyncFL
Eagle	$\boldsymbol{O(n\log(n) + d)}$	$O(n + d)$	3	SyncFL
LightSecAgg [So, 2022]	$O(n^2\frac{d}{(1-\delta)n-t} + d)$	$O(n\frac{d}{(1-\delta)n-t} + d)$	3	AsyncFL
Owl	$\boldsymbol{O(n^2 + d)}$	$\boldsymbol{O(n + d)}$	3	AsyncFL

Table 4.1.: Complexity analysis for one round (n: number of clients; δ: fraction of dropped clients; t: threshold value; d: input dimension).

In this paper, we cope with the problem of stragglers and propose two new SA protocols that address the aforementioned challenges inherent to realistic FL systems. More specifically:

- In the context of SyncFL, we significantly reduce the computation and communication overheads of FL clients and server through a new protocol named **Eagle**. More specifically, **Eagle** ignores dropped clients (including stragglers) and hence supports realistic dropout rates (from 10% to 30%). The performance improvement comes from a variant of the Threshold Joye-Libert scheme (**TJL**) proposed in [Mansouri, 2022].

- We develop a second protocol, named **Owl**, tailored to the AsyncFL setting. Similar to **Eagle**, **Owl** does not need to be aware of dropped clients and stragglers to complete the aggregation. We show that **Owl** is more efficient than the unique existing work [So, 2022]. Moreover, **Owl** is particularly suitable for deep learning models with large numbers of parameters.

- We conduct an extensive performance study and compare these two newly proposed schemes with relevant state-of-the-art solutions. Table 4.1 shows the asymptotic improvements of our two solutions compared to existing SA schemes. Especially, **Eagle** theoretically outperforms **SecAgg** and **FTSA**, whereas **Owl** asymptotically and experimentally shows better performance than **LightSecAgg** .

Notations

We provide the notations used throughout our paper in Table D.1.

4.2 Building Blocks

4.2.1 Joye-Libert Secure Aggregation Scheme

The Joye-Libert scheme (**JL**) [Joye, 2013], involving a Trusted Dealer (TD), n clients and one aggregator, is defined as follows:

- $(sk_0, \{sk_u\}_{u\in[1,n]}, N, H) \leftarrow$ **JL.Setup**(λ): Given security parameter λ, this algorithm generates two large and equal-size prime numbers p and q and sets $N = pq$. It randomly generates n secret keys $sk_u \xleftarrow{R} \mathbb{Z}_{N^2}$ and sets the aggregator key $sk_0 = -\sum_{u=1}^{n} sk_u$. Then, it defines a cryptographic hash function $H : \mathbb{Z} \rightarrow \mathbb{Z}^*_{N^2}$. It outputs the $n+1$ keys and the public parameters (N, H).

- $y_{u,\tau} \leftarrow$ **JL.Protect**$(pp, sk_u, \tau, x_{u,\tau})$: This algorithm encrypts the private input $x_{u,\tau} \in \mathbb{Z}_N$ for time period τ using secret key $sk_u \in \mathbb{Z}_{N^2}$. It outputs the cipher

$$y_{u,\tau} = (1 + x_{u,\tau}N) \cdot H(\tau)^{sk_u} \mod N^2$$

.

- $x_\tau \leftarrow$ **JL.Agg**$(pp, sk_0, \tau, \{y_{u,\tau}\}_{u\in[1,n]})$: This algorithm aggregates the n ciphers received at time period τ to obtain $y_\tau = \prod_1^n y_{u,\tau}$ and decrypts the result

$$x_\tau = \sum_1^n x_{u,\tau} = \frac{H(\tau)^{sk_0} \cdot y_\tau - 1}{N} \mod N$$

The **JL** scheme ensures Aggregator Obliviousness under the Decision Composite Residuosity (DCR) assumption [Paillier, 1999] in the random oracle model and assuming that each client encrypts only one value per time period [Joye, 2013].

4.2.2 Threshold Joye-Libert SA Scheme

In this section, we elaborate on a new variant of the Threshold **JL** scheme (**TJL**) [Mansouri, 2022]. In the original **TJL**, clients assist the aggregator in recovering the inputs of failed clients, which consist of the protected zero value encrypted under the failed client's individual key. This process allows for the computation of the final aggregate value. The **TJL** proposed here is a slightly modified version that utilizes the same primitive but instead of reconstructing the encrypted zero value for dropped clients, it reconstructs the aggregated zero-value for online clients. This approach helps us remove the need for defining an aggregation key in advance. Instead, an on-the-fly, per-round aggregation key is built based on the actual online clients at the specific round.

TJL cannot directly use the standard Shamir Secret Sharing scheme (**SS**) [Shamir, 1979] because sk_u is defined in $\mathbb{Z}^*_{\phi(N^2)}$ and $\phi(N^2)$ is not known to the clients (see Section 3.1 [Mansouri, 2022]). Hence, the solution uses the Integer version of **SS** (**ISS**) [Rabin, 1998], which is defined over integers rather than in a field (see Section 3.2 [Mansouri, 2022]). Informally, **ISS.Share** is run to split the secret key sk_u into n shares while **ISS.Recon** is called to recover the key given at least t shares.

The **TJL** scheme consists of the following PPT algorithms:

- $(sk_0, \{sk_u\}_{u\in[1,n]}, N, H) \leftarrow$ **TJL.Setup**(λ, σ): Given a security parameter λ, this algorithm essentially calls the original **JL.Setup**(λ) and outputs the aggregator

key, one secret key per client, and the public parameters. Additionally, it sets the security parameter of the **ISS** scheme to σ.

- $\{(v, [\Delta sk_u]_v)\}_{\forall v \in \mathcal{U}} \leftarrow$ **TJL.SKShare**$(sk_u, t, \mathcal{U})$: Upon input of client u's secret key sk_u, this algorithm calls **ISS.Share** where the interval of the secret is $\mathbb{Z}_{N^2}$.

- $y_{u,\tau} \leftarrow$ **TJL.Protect**$(pp, sk_u, \tau, x_{u,\tau})$: This algorithm primarily calls **JL.Protect**$(pp, sk_u, \tau, x_{u,\tau})$ and outputs the ciphertext $y_{u,\tau}$.

- $[y'_\tau]_u \leftarrow$ **TJL.ShareProtect**$(pp, \{[\Delta sk_v]_u\}_{v \in \mathcal{U}_{on}}, \tau)$: This algorithm protects a zero-value using client u's shares of all online clients secret keys (i.e. $v \in \mathcal{U}_{on}$). It calls **JL.Protect**$(pp, -\sum_{v \in \mathcal{U}_{on}} [\Delta sk_v]_u, \tau, 0)$ and outputs $[y'_\tau]_u = H(\tau)^{-\sum_{v \in \mathcal{U}_{on}} [\Delta sk_v]_u} \mod N^2$.

- $y'_\tau \leftarrow$ **TJL.ShareCombine**$(\{(u, [y'_\tau]_u, n)\}_{u \in \mathcal{U}_{shares}}, \mathcal{U}_{shares}, t)$: This algorithm combines t-out-of-n protected shares of the protected zero-value for time period τ and clients in $\mathcal{U}_{shares} \subseteq \mathcal{U}_{on}$ such that $|\mathcal{U}_{shares}| \geq t$ and $\Delta = n!$. It executes the Lagrange interpolation on the exponent to get $y'_\tau = \prod_{u \in \mathcal{U}_{shares}} ([y'_\tau]_u)^{\mu_u} = H(\tau)^{-\Delta^2 \sum_{v \in \mathcal{U}_{on}} sk_v}$ where the μ_u coefficients are defined in **ISS.Recon**.

- $x_\tau \leftarrow$ **TJL.Agg**$(pp, sk_0, \tau, \{y_{u,\tau}\}_{u \in \mathcal{U}_{on}}, y'_\tau)$: Given the public parameters pp, the aggregation key sk_0 (set to 0), the individual ciphertexts of online clients (i.e. $u \in \mathcal{U}_{on}$), and the ciphertexts of the zero-value corresponding to the clients $\mathcal{U}_{on}$, this algorithm aggregates the ciphertexts for time period τ. It first multiplies the inputs for all clients in $\mathcal{U}_{on}$, raises them to the power of Δ^2, and multiplies the result with the ciphertext of the zero-value to get $y_\tau = (\prod_{u \in \mathcal{U}_{on}} y_{u,\tau})^{\Delta^2} \cdot y'_\tau \cdot H(\tau)^{sk_0} \mod N^2$. To decrypt the final result, the algorithm calculates $x_\tau = \frac{y_\tau - 1}{N\Delta^2} \mod N$.

  The **TJL** scheme provides AO under the DCR assumption in the random oracle model if the number of corrupted clients is strictly less than t [Mansouri, 2022].

4.3 Eagle in SyncFL

We present **Eagle**, a fault-tolerant SA solution in the context of a synchronous setting. The design principles are the following: (1) The first aim is to not depend on dropped clients/stragglers anymore and to consider online clients' inputs only. We hence eliminate the need for blinding masks as in existing SA solutions [Bonawitz, 2017a; Bell, 2020] and significantly reduce the communication cost at the client side. (2) Instead of reconstructing the actual model parameters (which are usually assumed to be numerous), only one key is periodically reconstructed. Each client protects its private input using

Parties: Server and selected clients in $\mathcal{U}^{(\tau)}$, such that $|\mathcal{U}^{(\tau)}| = n$
Public Parameters: Generate the public parameters $pp^{KA} \leftarrow$ **KA.Param**(λ), $(\perp, \perp, N_0, H_0, \sigma) \leftarrow$ **TJL.Setup**(λ) and $(\perp, \perp, N_1, H_1) \leftarrow$ **JL.Setup**(λ) s.t. $N_0 \geq 2 \cdot N_1 + \log_2(n)$ and set $pp = (pp^{KA}, N_0, N_1, H_0, H_1, \tau_0, \sigma, t, n, d, R)$
Prerequisites: Each client $u \in \mathcal{U}$ generates a key pair $(c_u^{PK}, c_u^{SK}) \leftarrow$ **KA.gen**(pp^{KA}) and registers c_u^{PK} to Server or to a PKI

Setup - Key Setup:

Client $u \in \mathcal{U}^{(\tau)}$: *// Generate* **TJL** *key and secret share*

1. $\forall v \in \mathcal{U}^{(\tau)} \setminus \{u\}$, $c_{u,v} \leftarrow$ **KA.agree**$(pp^{KA}, c_u^{SK}, c_v^{PK})$. *// Establish pairwise channel keys with each client*
2. $sk_u \xleftarrow{R} \mathbb{Z}_{N_0^2}$. *// Generate* ***TJL*** *secret key*
3. $\{(v, [sk_u]_v)\}_{v \in \mathcal{U}^{(\tau)}} \leftarrow$ **TJL.SKShare**$(sk_u, t, \mathcal{U}^{(\tau)})$. *// Generate t-out-of-n shares*
4. $\forall v \in \mathcal{U}^{(\tau)} \setminus \{u\}$, $\epsilon_{u,v} \leftarrow$ **AE.enc**$(c_{u,v}, u \mid\mid v \mid\mid [sk_u]_v)$. *// Encrypt each share with the corresponding public key*
5. Send $\{(u, v, \epsilon_{u,v})\}_{v \in \mathcal{U}^{(\tau)}}$ to *Server*.

Server: *// Collect encrypted shares of* ***TJL*** *keys and forward them to destined clients*

1. Collect $\{(u, v, \epsilon_{u,v})\}_{v \in \mathcal{U}^{(\tau)}}$.
2. $\forall v \in \mathcal{U}^{(\tau)} \setminus \{u\}$, send $\{(u, v, \epsilon_{u,v})\}_{u \in \mathcal{U}^{(\tau)}}$.

Client $u \in \mathcal{U}^{(\tau)}$: *// Decrypt the received shares*

1. $\forall v \in \mathcal{U}^{(\tau)} \setminus \{u\}$, $[sk_v]_u \leftarrow$ **AE.dec**$(c_{u,v}, v \mid\mid u \mid\mid \epsilon_{v,u})$. *// Decrypt each share with the corresponding public key*

Online - Protection (step τ):

Client $u \in \mathcal{U}^{(\tau)}$: *// Protect the private input using* ***JL*** *key, the per-round* ***JL*** *secret key using* ***TJL*** *key, and send them to the server*

1. $sk_{u,\tau} \xleftarrow{R} \mathbb{Z}_{N_1^2}$. *// Generate the per-round* ***JL*** *secret key*
2. $\vec{y}_{u,\tau} \leftarrow$ **JL.Protect**$(pp, sk_{u,\tau}, \tau_0, \vec{x}_{u,\tau})$. *// Protect private input $\vec{x}_{u,\tau} \in \mathbb{Z}_R^d$ using* ***JL***
3. $\langle sk_{u,\tau} \rangle \leftarrow$ **TJL.Protect**$(pp, sk_u, \tau, sk_{u,\tau})$. *// Protect the per-round* ***JL*** *secret key $sk_{u,\tau} \in \mathbb{Z}_{N_1^2}$ using* ***TJL***
4. Send $\vec{y}_{u,\tau}$ and $\langle sk_{u,\tau} \rangle$ to *Server*.

Server: *// If the number of online clients ($\mathcal{U}_{on}^{(\tau)} \subseteq \mathcal{U}^{(\tau)}$) is less than t, abort; otherwise, collect the protected secret keys, the protected inputs, and broadcast $\mathcal{U}_{on}^{(\tau)}$ to all clients*

1. Collect $\{\langle sk_{u,\tau} \rangle\}_{\forall u \in \mathcal{U}_{on}^{(\tau)}}$ and $\{\vec{y}_{u,\tau}\}_{\forall u \in \mathcal{U}_{on}^{(\tau)}}$.
2. If $|\mathcal{U}_{on}^{(\tau)}| < t$, abort; otherwise broadcast $\mathcal{U}_{on}^{(\tau)}$.

Online - Consistency Check (step τ): *// See Figure 4 (Consistency Check) of [Bonawitz, 2017a]* **Online - Reconstruction (step τ):**

Client $u \in \mathcal{U}_{on}^{(\tau)}$: // Compute the share of the per-round ***JL*** *server key, and send it to the server*

1. $[\langle sk'_{0,\tau} \rangle]_u \leftarrow$ **TJL.ShareProtect**$(pp, \{[sk_v]_u\}_{v \in \mathcal{U}_{on}^{(\tau)}}, \tau)$.
2. Send $[\langle sk'_{0,\tau} \rangle]_u$ to *Server*.

Server: *// If the number of honest clients ($\mathcal{U}_{shares}^{(\tau)} \subseteq \mathcal{U}_{on}^{(\tau)}$) is less than t abort; otherwise collect t shares, reconstruct the per-round* ***JL*** *server key and complete the aggregation*

1. Collect $\{[\langle sk'_{0,\tau} \rangle]_u\}_{u \in \mathcal{U}_{shares}^{(\tau)}}$.
2. If $|\mathcal{U}_{shares}^{(\tau)}| < t$, abort; otherwise, proceed.
3. $sk'_{0,\tau} \leftarrow$ **TJL.ShareCombine**$(\{[\langle sk'_{0,\tau} \rangle]_u\}_{u \in \mathcal{U}_{shares}^{(\tau)}}, t)$. *// Reconstruct the zero-scalar value of the online clients*
4. $sk_{0,\tau} \leftarrow$ **TJL.Agg**$(pp, 0, \tau, \{sk_{u,\tau}\}_{u \in \mathcal{U}_{on}^{(\tau)}}, sk'_{0,\tau})$. *// Reconstruct the per-round* ***JL*** *server key*
5. $\vec{x}_\tau \leftarrow$ **JL.Agg**$(pp, -sk_{0,\tau}, \tau_0, \{\vec{y}_{u,\tau}\}_{u \in \mathcal{U}_{on}^{(\tau)}})$. *// Complete the aggregation*

Figure 4.1.: **Eagle** in SyncFL.

a freshly generated per-round **JL** key and this key is then protected with the **TJL** key. Thanks to this approach, **Eagle** exhibits a quasi-linear computation cost at the client. The solution is defined in two phases: the setup phase during which clients first register to the server and receive their keying material, and the online phase during which aggregation occurs.

Description

The protocol is depicted in Figure 4.1. It starts with the setup phase, where each FL client first generates a pair of secret and public keys and transmits its public key to the FL server who then broadcasts them to all FL clients together with the public parameters pp. Delegating the public parameter generation to a TD is common in other existing works[1]. At the *Key Setup* step, each client independently computes t out of n shares of its secret key sk_u using **TJL.SKShare**. Subsequently, similar to **FTSA** [Mansouri, 2022], these shares are one-by-one sent to the appropriate clients, via the server, through authenticated encrypted (**AE**) channels. The online phase, is broken down into three steps for each round:

(1) At the *protection step*, online clients generate one per-round **JL** secret key $sk_{u,\tau}$ that is then used to protect their private input vectors $\vec{x}_{u,\tau}$ using **JL.Protect** at round τ, such that the protection with **JL** uses a fixed $\tau = \tau_0$. This key is further protected using **TJL.Protect** and all this information is sent to the FL server. The server gathers both the protected inputs (vectors) and the protected per-round clients' keys (scalars).

(2) The *consistency check step* is the same as for **SecAgg** [Bonawitz, 2017a].

(3) At the *reconstruction step*, the clients receive the list of the online clients $\mathcal{U}_{on}^{(\tau)}$. Their goal is to help compute/reconstruct the per-round aggregation key for the server in order to have access to the actual sum of private inputs in plaintext. This aggregation key basically consists of the sum of the per-round keys of online clients that will be reconstructed with the collaboration of at least t online clients using **TJL.ShareCombine**. Then, the actual model parameters $\vec{x}_\tau$ can be computed using **JL.Agg**.

Security Analysis (Sketch)

We briefly analyse the security of **Eagle** by following the approach given in [Mansouri, 2022], hybrid security proof is provided in Appendix C.1.

[1]Note that the generation of public parameters pp depends on the existence of a TD. Nevertheless, there exist methods in the decentralized setting [Nishide, 2011; Veugen, 2019; Chen, 2021].

Parties: Server and all clients in $\mathcal{U}$, such that $|\mathcal{U}| = n_{tot}$, and clients in the buffer $\mathcal{U}_{on}$, such that $|\mathcal{U}_{on}| = n$
Public Parameters: Generate the public parameters $pp^{KA} \leftarrow$ **KA.Param**(λ) and $(\perp, \perp, N, H, \perp) \leftarrow$ **JL.Setup**(λ) and set $pp = (pp^{KA}, N, H, \tau_0, t, n_{tot}, d, \mathbb{F}_p, R)$
Prerequisites: Each client $u \in \mathcal{U}$ generates a key pair $(c_u^{PK}, c_u^{SK}) \leftarrow$ **KA.gen**(pp^{KA}) and registers c_u^{PK} to Server or to a PKI

Setup - Key Setup:

Client $u \in \mathcal{U}$:

1. $\forall v \in \mathcal{U} \setminus \{u\}$, $c_{u,v} \leftarrow$ **KA.agree**$(pp^{KA}, c_u^{SK}, c_v^{PK})$. *// Establish pairwise channel keys with each client*

Online - Protection:

Client u: *// Protect private input using **JL** key, secret share per-client-round **JL** secret key*

1. $sk_{u,\tau_u} \xleftarrow{R} \mathbb{Z}_{N^2}$. *// Generate **JL** secret key for round τ_u*
2. $\vec{y}_{u,\tau_u} \leftarrow$ **JL.Protect**$(pp, sk_{u,\tau_u}, \tau_0, \vec{x}_{u,\tau_u})$. *// Protect private input $\vec{x}_{u,\tau_u}$ using **JL** key*
3. $\{(v, [sk_{u,\tau_u}]_v)\}_{\forall v \in \mathcal{U}} \leftarrow$ **SS.Share**$(sk_{u,\tau_u}, t, \mathcal{U})$. *// Generate shares of the per-client **JL** secret key for round τ_u*
4. $v \in \mathcal{U} \setminus \{u\}$, $\epsilon_{u,v} \leftarrow$ **AE.enc**$(c_{u,v}, u \;||\; v \;||\; [sk_{u,\tau_u}]_v)$. *// Encrypt each share with the corresponding public key*
5. Send $\vec{y}_{u,\tau_u}$ and $\{(u, v, \epsilon_{u,v})\}_{u \in \mathcal{U}}$ to *Server*.

Server: *// If the number of clients is less than t, abort; otherwise, collect protected inputs, encrypted shares of secret keys, forward encrypted shares to destined clients and broadcast $\mathcal{U}_{on}$*

1. Collect $\{\vec{y}_{u,\tau_u}\}_{u \in \mathcal{U}_{on}}$ and $\{(u, v, \epsilon_{u,v})\}_{u \in \mathcal{U}_{on}}$.
2. If $|\mathcal{U}_{on}| < t$, abort; otherwise, broadcast $\mathcal{U}_{on}$ and $\forall v \in \mathcal{U}_{on}$ send $\{(u, v, \epsilon_{u,v})\}_{u \in \mathcal{U}_{on}}$.

Online - Consistency Check: *// See Figure 4 (Consistency Check) of [Bonawitz, 2017a]*

Online - Reconstruction:

Client u: *// Compute the share of the **JL** aggregation key*

1. $\forall v \in \mathcal{U}_{on} \setminus \{u\}$, $[sk_{v,\tau_v}]_u \leftarrow$ **AE.dec**$(c_{u,v}, v \;||\; u \;||\; \epsilon_{v,u})$. *// Decrypt each share*
2. $[sk_0]_u \leftarrow \sum_{\forall v \in \mathcal{U}_{on}} [sk_{v,\tau_v}]_u$. *// Compute share of per-round aggregation key*
3. Send $[sk_0]_u$ to *Server*.

Server: *// If the number of honest clients is less than t, abort; otherwise, collect t shares, reconstruct the per-client-round **JL** server key and complete the aggregation*

1. Collect $\{[sk_0]_u\}_{u \in \mathcal{U}_{shares}}$.
2. If $|\mathcal{U}_{shares}| < t$, abort; otherwise, proceed.
3. $sk_0 \leftarrow$ **SS.Recon**$(\{[sk_0]_v\}_{v \in \mathcal{U}_{shares}}, t)$. *// Reconstruct **JL** aggregation key*
4. $\vec{x} \leftarrow$ **JL.Agg**$(pp, -sk_0, \tau_0, \{\vec{y}_{u,\tau_u}\}_{u \in \mathcal{U}_{on}})$. *// Compute aggregate value*

Figure 4.2.: **Owl** in BAsyncFL.

- In the **honest-but-curious** model, we assume that the server correctly follows the protocol but can collude with (or corrupts) up to $n-t$ clients. Let $\mathcal{U}_{corr}$ be the set of corrupted clients and $\mathcal{C} = \mathcal{U}_{corr} \cup \mathcal{S}$ where $\mathcal{S}$ represents the server. The view of $\mathcal{C}$ is computationally indistinguishable from a simulated view if the number of corrupted clients is less than the threshold t (i.e., $|\mathcal{U}_{corr}| < t$). Based on that, the minimum number of honest clients t should be strictly larger than half of the number of clients in the protocol (i.e., $t > \frac{n}{2}$). Hence the protocol can recover from up to $\frac{n}{2} - 1$ client failures. The security of the **TJL** ensures that parties in $\mathcal{C}$ cannot distinguish the protected temporary **JL** key of an honest client $\langle sk_{u,\tau} \rangle$ from random values. It also ensures that if parties in $\mathcal{C}$ have access to at most $t-1$ shares of sk_u (i.e., $|\mathcal{U}_{corr}| < t$), then, they cannot distinguish the shares held by the honest clients from random values. Therefore, the view of parties in $\mathcal{C}$ at the end of each FL round τ is computationally indistinguishable from a simulated view. Thus, the server learns nothing more than the sum of the online clients' inputs if $|\mathcal{U}_{on}^{(\tau)}| \geq |\mathcal{U}_{shares}^{(\tau)}| \geq t$ and hence *AO* is ensured.

- In the **active** model, $\mathcal{S}$ can additionally manipulate its inputs to the protocol. The only messages $\mathcal{S}$ distributes, other than the clients' public keys, are the protected shares that are forwarded from and to the clients. $\mathcal{S}$ cannot modify the values of these encrypted shares thanks to the underlying authenticated encryption scheme **AE**. Therefore, $\mathcal{S}$'s power in the protocol is limited to not forwarding some of the shares. This may make clients reach some false conclusions about the set of online clients $\mathcal{U}_{on}^{(\tau)}$. It is important to note that $\mathcal{S}$ can present different views to different clients regarding their online/dropped status. This capability enables $\mathcal{S}$ to easily acquire the individual temporary **JL** key $sk_{u,\tau}$ of a client u. More precisely, $\mathcal{S}$ can convince a subset of honest clients that the set of online clients is $\mathcal{U}_{on}^{(\tau)}$ while indicating to another subset that the online clients' set is $\mathcal{U}_{on}^{(\tau)'} = \mathcal{U}_{on}^{(\tau)} \setminus \{u\}$ (i.e., u is dropped). If this occurs, $\mathcal{S}$ can aggregate the protected inputs from $\mathcal{U}_{on}^{(\tau)}$ to derive the per-round aggregation key sk_τ, and also aggregate inputs from $\mathcal{U}_{on}^{(\tau)'}$ to derive sk'_τ, and then calculate $sk_{u,\tau} = sk_\tau - sk'_\tau$. Considering the scenario where $\mathcal{S}$ may collude with $n-t$ corrupted clients, it can obtain $n-t$ shares of $\langle sk_\tau \rangle$ and $\langle sk'_\tau \rangle$ and hence $n-t$ shares of $sk_{u,\tau}$. Furthermore, $\mathcal{S}$ has the ability to convince $\frac{t}{2}$ honest clients that client u is online, and the other $\frac{t}{2}$ honest clients that u is dropped, thereby collecting shares of $\vec{y}_\tau$ and $\vec{y}'_\tau$ respectively. Hence, in total, the server can learn a maximum number of $n - t + \frac{t}{2}$ shares of $\langle sk_\tau \rangle$ and $\langle sk'_\tau \rangle$. Therefore, to prevent such attacks and still ensure *AO*, we additionally require that $n - t + \frac{t}{2} < t \implies t > \frac{2n}{3}$. Hence the protocol can recover from up to $\frac{n}{3} - 1$ client failures in the active model.

To conclude, if the server operates under an **honest-but-curious** model, selecting $t > \frac{n}{2}$ ensures security. However, if the server **actively** manipulates protocol messages, the

threshold should be set to $t > \frac{2n}{3}$. Additional details about the threat model and the rationale behind these thresholds can be found in [Bonawitz, 2017a] (Sections 6.1 and 6.2). Note that we achieve the same threshold values as those in **SecAgg** [Bonawitz, 2017a] and **FTSA** [Mansouri, 2022].

- A new type of attack, called model inconsistency attack, launched by an active aggregator is defined and studied in [Pasquini, 2022]. This attack consists of a malicious server sending carefully crafted models to specific clients instead of the actual global model parameters, with the aim of extracting private clients' parameters. **Eagle** can easily prevent such attacks by adopting the same approach proposed in [Pasquini, 2022]. In more details, using the hash of the global model to set the server's value τ_0, which needs to be the same for all clients, allows to overcome the aforementioned attack.

4.4 Owl in AsyncFL

In the asynchronous setting, we propose **Owl**, defined with a setup phase and an online phase. As opposed to the case in SyncFL, in the context of AsyncFL, clients cannot be expected to be synchronized with respect to the same round τ. Consequently, by design, the **TJL** scheme cannot be used directly. To counter this problem, one common round τ_0 is defined per client u and each client uses **JL.Protect** with its own per-round key sk_{u,τ_u}. On the other hand, the FL server also defines its own round τ_0 (which, in fact, never changes) and is still able to aggregate all values thanks to the use of **JL.Agg** with τ_0 to get the per-round aggregate key and to further obtain the aggregate model.

Description

The protocol is depicted in Figure 4.2. It starts with the setup phase, similar to **Eagle**, where each FL client generates a pair of secret and public keys and transmits its public key to the FL server who then broadcasts it to all FL clients, together with the public parameters pp. The online phase, consists of three steps:

(1) During the *protection step*, each online client u generates one **JL** secret key at round τ_u, denoted as sk_{u,τ_u}. The client then computes n shares of this secret key such that any t shares can reconstruct it, using the **SS** scheme. Subsequently, the private input $\vec{x}_{u,\tau_u}$ (vector) is protected using **JL.Protect** which takes as inputs sk_{u,τ_u} and a fixed τ_0 defined by the server. The server collects both the protected inputs and the encrypted shares of the protection key from the online clients.

(2) The *consistency check step* is the same as for **SecAgg** [Bonawitz, 2017a].

(3) At the *reconstruction step*, each client u receives the encrypted shares of each other client's protection key and computes the share of the server's **JL** aggregation key $\sum_{\forall v \in \mathcal{U}_{on}} [sk_{v,\tau_v}]_u$. This global share is forwarded to the server. The server should receive at least t shares to reconstruct the aggregation key sk_0. Finally, the server aggregates the inputs of the online clients using **JL.Agg**.

Security Analysis (Sketch)

We briefly analyse the security of **Owl** by following the approach given in [Mansouri, 2022], hybrid security proof is provided in Appendix C.1.

- In the **honest-but-curious** model, the **JL** scheme ensures that the server together with clients in $\mathcal{C}$ cannot distinguish protected inputs $\vec{y}_{u,\tau_u}$ from random values. Furthermore, the **SS** scheme ensures that if parties in $\mathcal{C}$ have access to less than $t-1$ shares of the client's secret key sk_{u,τ_u} (i.e. $|\mathcal{U}_{corr}| < t$), then they cannot distinguish the shares held by the honest clients from random values. Therefore, the view of clients in $\mathcal{C}$ is computationally indistinguishable from a simulated view. Thus, the server learns nothing more than the sum of the online clients' inputs if $|\mathcal{U}_{on}^{(\tau)}| \geq |\mathcal{U}_{shares}^{(\tau)}| \geq t$.

- When $\mathcal{S}$ is an **active** adversary, it can try to convince a subset of honest clients that the set of online clients is $\mathcal{U}_{on}$ while indicating to another subset that the set of online clients is $\mathcal{U}'_{on} = \mathcal{U}_{on} \setminus \{u\}$ for a client u. If this occurs, $\mathcal{S}$ can reconstruct sk_0 and sk'_0 and then, it can compute $sk_{u,\tau_u} = sk_0 - sk'_0$. Because we assume that there are $n-t$ corrupted clients, $\mathcal{S}$ can obtain $n-t$ shares of sk_0 and sk'_0 respectively. Furthermore, $\mathcal{S}$ has the ability to convince $\frac{t}{2}$ honest clients that the client u is online, and the other $\frac{t}{2}$ honest clients that u is dropped, thereby collecting shares of sk_0 and sk'_0 respectively. Therefore, to ensure *AO*, we require that $n - t + \frac{t}{2} < t \implies t > \frac{2n}{3}$.

To conclude, if $\mathcal{S}$ is **honest-but-curious**, selecting $t > \frac{n}{2}$ ensures security. However, if $\mathcal{S}$ **actively** manipulates protocol messages, the threshold should be $t > \frac{2n}{3}$. We get the same threshold values obtained in [So, 2022].

- As mentioned in [Pasquini, 2022], defenses against model inconsistency attacks in AsyncFL settings are a difficult task and no one has yet proposed a potential solution. Consequently, those attacks are out of scope for **Owl**.

4.5 Related Work

Secure Aggregation for SyncFL

Bonawitz et al. [Bonawitz, 2017a] propose **SecAgg**, a fault-tolerant SA approach that employs secure masking. Each pair of clients creates a shared mask through a key agreement scheme and uses this mask to protect clients' inputs. Additionally, before this protection, clients also combine their input data with another blinding mask. The purpose of this blinding mask is to prevent any active, malicious server from discovering one individual input at the reconstruction step. To address the potential issue of client dropout, the protocol implements secret sharing: the clients secretly share their respective shared masks and blinding masks. Then, the server computes the sum of the masked inputs and further recovers the shared masks of the failed clients and the blinding masks of the online clients, thereby completing the aggregation. Compared to **SecAgg**, **Eagle** does not use any blinding mask and consequently is more efficient in terms of computation and communication costs.

Mansouri et al. [Mansouri, 2022] develop a fault-tolerant SA solution called **FTSA**, which uses the **TJL** scheme to protect client inputs and reconstruct the aggregate in case of client failures, reducing the online communication rounds from 4 to 3 rounds compared to **SecAgg**. To address the issue of potential client dropouts, the clients secretly share their respective secret keys using the **ISS** scheme. Similar to **SecAgg**, **FTSA** also uses blinding masks for clients' inputs which once again increases the communication cost compared to our protocol. Furthermore, **FTSA** cannot directly support *client selection* as the non-selected clients would be considered as failed clients and this would significantly increase the computation and communication overhead. Our solution instead only depends on the number of online client.

Ma et al. [Ma, 2023] introduce **Flamingo**, a fault-tolerant SA protocol employing secure masking. The authors construct connected graphs of clients (as opposed to fully connected clients) such that the shared mask is created among the connected clients. **Flamingo** introduces the concept of *decryptors* to help the server reconstruct the aggregate, as opposed to distribute the reconstruction to all clients. Its functioning is as follows: each pair of connected clients generates a shared seed through key agreement and creates a shared pairwise mask using a Pseudo-Random Function (PRF). Moreover, a blinding mask is created. To address the potential issue of client dropout, the protocol implements Threshold ElGamal asymmetric encryption (TEG). In **Flamingo**, clients encrypt the pairwise mask using TEG, while the blinding mask is secretly shared with the decryptors. If a client fails, the server asks the decryptors to reconstruct the pairwise mask using the partial decryption of TEG. Otherwise, they reconstruct the blinded mask

for the online clients. Consequently, **Flamingo** incurs three client-server trips. Similar to previously explained protocols, the computational complexity increases with the use of the blinding mask. Additionally, its complexity scales with the number of dropped clients, impacting the number of connected clients in the sparse graph. Furthermore, dropped clients have a negative impact during the reconstruction led by the decryptors, as the more they drop, the more pairwise masks the decryptors and the server have to reconstruct.

Bell et al. [Bell, 2020] enhance the scalability of **SecAgg** [Bonawitz, 2017a]. Their method does not require the clients to secretly share secret keys with every other client to ensure resilience against dropouts. Instead, the authors build connected graphs of clients (as opposed to fully connected clients) in which SA is exclusively carried out among the connected clients. We did not provide a detailed experimental comparison, as [Ma, 2023] already demonstrated that [Bell, 2020] requires 6 online communication rounds compared to 3 for **Flamingo** and **Eagle**.

Several protocols provide slight improvements either on the computation cost [Kadhe, 2020] or on the communication cost [So, 2021]. We do not extensively compare **Eagle** with these solutions, mainly because they do not offer the same privacy guarantees.

A recent work [Li, 2023] proposes a SA method, called **Lerna**, for a large number of clients and which shows a similar DCR construction to **Eagle**. However, their work focuses only on a large number of clients, without considering dropped clients/stragglers and AsyncFL settings. Therefore, we choose to not include **Lerna** in our comparisons.

Secure Aggregation for AsyncFL

So et al. [So, 2022] propose a SA method, called **LightSecAgg**, that is designed to be compatible with AsyncFL and is the closest solution to **Owl**. In **LightSecAgg**, each client independently generates a random mask which is further secretly shared (using Lagrange code computing [Yu, 2018]) among other clients. At the protection step, each client adds the mask to protect its local model and sends the masked model to the server. At the reconstruction step, the server can reconstruct and cancel out the aggregated masks of the online clients through one-shot decoding. This decoding is performed using the aggregated shared masks received in a second round of communication. **Owl** offers better scalability mainly because, instead of secretly sharing the random mask, it only requires the secret sharing of a single value, the client's **JL** secret key. This optimization significantly reduces both the runtime and communication costs associated with the protocol.

Other protocols, such as **FedBuff** [Nguyen, 2022], rely on the use of a Trusted Execution Environment (TEE). While such solutions may be more efficient, such a memory-constrained technology cannot be assumed available in all FL settings.

4.6 Complexity Analysis

4.6.1 Eagle

We evaluate the computation and communication costs of **Eagle** and we compare them with **SecAgg** [Bonawitz, 2017a] and **FTSA** [Mansouri, 2022]. Table 4.1 on page 45 summarizes this study. Note that our analysis considers the size of the secret shares since this metric has a non-negligible impact on the *reconstruction step*.

- *Client computation:* Firstly, at the *protection step*, the client protects its d-size input $\vec{x}_{u,\tau}$ which results in a cost worth $O(d)$. Then, at the *reconstruction step*, the client executes **TJL.SharеProtect** which consists of: *(i)* computing the sum of the secret shares of online clients which requires a computational cost of $O(n)$ as opposed to $O(n^2)$ for **FTSA** and **SecAgg**, mainly because both the latter compute the secret shares of the blinding mask; *(ii)* protecting the zero-*scalar* value using this sum of secret shares through **ISS**, and hence incurring an overhead of $O(n\log(n))$, as opposed to $O(n\log(n)d)$ for **FTSA**. Thus, the overall computation cost for this step is $O(n\log(n)+d)$, which is a quasi-linear complexity.

- *Client communication:* There are two communication rounds. Firstly, at the *protection step*, the client sends its protected input $\vec{y}_{u,\tau}$ to the server, which has a size of $O(d)$. Then, at the *reconstruction step*, the client receives the information on other online clients which is of size $O(n)$, and sends one zero-*scalar* value protected with the combination of their shares, namely $sk'_{0,\tau}$, to the server, which has a constant size $O(1)$. Hence, the communication cost at the client is $O(n+d)$ which is asymptotically the same as **FTSA** and **SecAgg**. Nevertheless, in **FTSA**, during the reconstruction step, a zero-*vector* value is sent which has the same size as of the protected input, that is $O(d)$. When client dropouts occur, our solution would outperform **FTSA** mainly because **Eagle** does not depend on the number of client failures. This is also experimentally studied and evaluated in Section 4.7.

- *Server computation:* The server performs two main operations: *(i)* at the *reconstruction step*, it reconstructs the protected zero-*scalar* value $sk'_{0,\tau}$ from the t shares of the online clients, requiring a computation cost of $O(n^2+n)$ (from Lagrange coefficients); *(ii)* the server aggregates the protected values and unmasks the result, which requires a computation cost of $O(nd)$. The overall cost is the same as in

FTSA, worth $O(n^2 + nd)$. As at the clients, the computation cost at the server is not impacted by dropouts. On the other side, in **FTSA** and **SecAgg**, the server needs to reconstruct the blinding masks of dropped clients. In particular, in **FTSA**, the computation of the protected zero-*vector* value of the dropped clients incurs a cost of $O(n^2 + nd)$, and in **SecAgg**, the masks of the dropped clients are reconstructed with a complexity $O(n^2 d)$. This is also experimentally studied and evaluated in Section 4.7.

- *Server communication:* Similar to **SecAgg** and **FTSA**, since the message exchanges in the protocol only occur between the server and the clients, the server's communication cost is equal to n times each client's communication cost.

	Flamingo [Ma, 2023]	**Eagle**
Client Comp.	Regular client: $O(k^2 + ad)$ Decryptor: $O(\delta an + (1-\delta)n)$	Regular client: $O(d)$ Decryptor: $O((1-\delta)n + k \log k)$
Client Comm.	Regular client: $O(a + k + d)$ Decryptor: $O(\delta an + (1-\delta)n)$	Regular client: $O(d)$ Decryptor: $O(n)$

Table 4.2.: Complexity analysis for one SyncFL round with decryptors (n: number of clients, d: input dimension; k: number of decryptors; δ: dropout rate; a: upper bound on the number of neighbors per client).

4.6.2 **Eagle** with decryptors

We conduct a comparative complexity analysis of **Eagle** when decryptors are involved, and compare the online phase against **Flamingo** [Ma, 2023]. We distinguish regular clients (who contribute to the global model) from decryptors (who help the server obtain this global model). Let the dimension of the model be d, the dropout rate be δ, the number of neighbors for a given client be a, the number of decryptors be k and the number of regular clients be n. Note that the number of neighbors a is determined by the dropout rate δ, as shown in [Ma, 2023] (see Appendix C). The results are summarized in Table 4.2.

- *Client Computation:* For regular clients in **Eagle**, the primary computational cost is attributed to the *protection step*, worth $O(d)$. In the case of **Flamingo**, the protection cost is $O(ad)$. This cost involves the secret sharing of the blinding mask with k decryptors, incurring an expense of $O(k^2)$, and the TEG encryption of the pairwise masks worth $O(a)$.

  For decryptors in **Eagle**, the computational requirements resemble those without decryptors, with the different being the size of the **TJL** secret key share, which is $k \log k$. Consequently, the cost associated with protecting the zero-scalar value becomes $O(k \log k)$, resulting in a total cost of $O(n + k \log k)$. In the context of

Flamingo, the overhead incurred by decryptors is proportional to the fraction of dropped clients δ, since decryptors partially decrypt δan TEG ciphertexts and decrypt $(1-\delta)n$ secret shares.

- *Client Communication:* For regular clients in **Eagle**, the communication cost worth $O(d)$ depends on the protected input size d. In the case of **Flamingo**, in addition to the protected input, the communication also involves sending a TEG ciphertexts and k encrypted secret shares of the blinding mask.

  For decryptors in **Eagle**, the only required information is the set of online clients, which has to be reconstructed, incurring a cost of $O((1-\delta)n)$. In the context of **Flamingo**, the decryptors receive from the server δan TEG ciphertexts and $(1-\delta)n$ encrypted blinding masks, costing $O(\delta an + (1-\delta)n)$.

- *Server Computation:* During the *reconstruction step* in **Eagle**, the server reconstructs the protected zero-scalar value using t shares from the decryptors, requiring a computational cost of $O(k^2 + k)$ (due to Lagrange coefficients' computation). Subsequently, the server aggregates the protected values and unmasks the result, which requires a computational cost of $O(nd)$.

  In **Flamingo**, the *reconstruction step* requires the cost of the Lagrange coefficients' computation, worth $O(k^2)$, in addition to the reconstruction of $(1-\delta)n$ blinding masks, incurring a cost of $O((1-\delta)nk)$. The reconstruction of the pairwise masks implies a cost of $O(\delta ank)$, and then the aggregation and unmasking cost $O(k^2 + \delta ank + (1-\delta)nk + nd)$.

- *Server Communication:* In both protocols, the communication cost is equal to n times the client's communication cost.

4.6.3 Owl

We evaluate our protocol **Owl** tailored for AsyncFL and compare its costs with **LightSecAgg** [So, 2022]. Table 4.1 on page 45 summarizes our study and shows that our solution outperforms **LightSecAgg**. We set the complexity of polynomial evaluation and interpolation as $O(n^2)$ for all solutions. Note that this complexity can be reduced to $O(n \log n)$ as pointed out in [So, 2022] and acknowledged in [Shamir, 1979].

- *Client computation:* At the *protection step*, the client generates t out of n shares of the secret key sk_{u,τ_u}, which requires a computation cost of $O(n^2)$. Also, the client protects its message $\vec{x}_{u,\tau_u}$ using the secret key sk_{u,τ_u}, which requires a computation

cost of $O(d)$. Finally, at *reconstruction step*, the client computes the sum of the secret key shares of other online clients, which requires a computation cost of $O(n)$. This cost is better than in **LightSecAgg** , which is $O(n^2 \frac{d}{(1-\delta)n-t} + d)$, mainly due to their underlying encoding [Yu, 2018].

- *Client communication:* At the *protection step*, the client sends $O(n)$ shares of its secret key sk_{u,τ_u} and receives $O(n)$ shares in return. The client further sends the encrypted input $\vec{y}_{u,\tau_u}$ to the server, which is of size $O(d)$. Finally, at the *reconstruction step*, the client sends its share of the server's secret key $[sk_0]_u$ which has a size of $O(1)$. The total cost, worth $O(n+d)$, is better than $O(n \frac{d}{(1-\delta)n-t} + d)$ from **LightSecAgg**.

- *Server computation:* At the *reconstruction step*, the server constructs the server key sk_0 from its t shares, which requires a computation cost of $O(n^2)$. Additionally, the server aggregates the ciphertexts received from each client and unmasks the result, which requires a computation cost of $O(nd)$.

- *Server communication:* Similar to **Eagle**, since the message exchanges in the protocol only occur between the server and the clients, the server's communication cost is equal to n times each client's communication cost.

4.7 Experimental Study

We also conduct an experimental study of the performance of **Eagle** and **Owl**, with respect to the number of selected clients and buffer size respectively, the size of the machine learning model, while considering realistic dropout rates and over selection. We also study use cases of training a machine learning model for MNIST, CIFAR-10 and Shakespeare datasets.

4.7.1 Experimental Setting

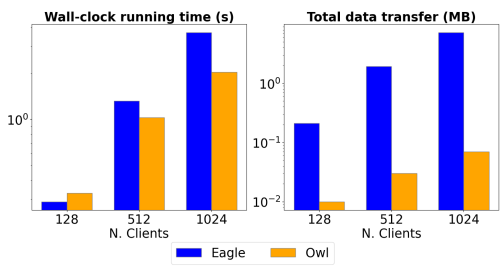


Figure 4.3.: Computation and communication costs (sent/received) per client during the setup phase of **Eagle** and **Owl**.

All our implementations use the Python programming language[2]. Experiments were carried out on a single-threaded process, using a machine equipped with an Intel(R) Core(TM) i7-7800X CPU @ 3.50GHz processor and 126 GB of RAM. For the sake of fair comparison, our solutions along with **SecAgg**, **FTSA**, **Flamingo**, and **LightSecAgg** are implemented using the same building blocks and libraries mentioned in [Mansouri, 2022] (see Appendix C).

We consider different settings that simulate realistic environments. The number of selected SyncFL clients and the size of the AsyncFL buffer is set to $n = \{512, 1024\}$, and the model size is $d = \{10^5, 10^6\}$. Similar to previous works, the client dropout and over-selection rates are set to $\delta = \{0.0, 0.1, 0.3\}$, and the chosen threshold to $t = \frac{2n}{3}$. We assume that client dropouts happen before the clients send their protected inputs. This is essentially the "worst dropout case", since the server must perform an expensive recovery computation to correctly compute the aggregate. Regarding the packing technique, we adopt the vector encoding approach proposed by [Mansouri, 2022].

We measure the execution time (i.e. computation cost) and the bandwidth (i.e. communication cost) at both the client and the server sides. The values shown for each experiment are the result of an average of measurements from 5 independent executions. We report performance results of the the setup phase in Figure 4.3. Below, we detail the performance results of the online phase below.

4.7.2 Eagle

We evaluate the performance of **Eagle** by first comparing it with **SecAgg** and **FTSA** since these three solutions have similar settings. Since **Flamingo** considers the involvement of some *decryptors* who help reconstruct clients' material (see Section 4.6.2), we conduct additional experiments that simulate these decryptors for **Eagle** and enable a fair comparison with **Flamingo**.

4.7.3 Eagle vs. SecAgg and FTSA

Table 4.3 depicts the running time of one FL client as well as the total data transfer for various realistic settings. As expected, we observe that the best running time is obtained with **Eagle**, which is independent of the dropout rates. While there is a significant difference with **SecAgg** due to its strong dependence on the number of clients n, it is lighter in the case of **FTSA** when there is no dropout, since **Eagle** does not require any sharing of blinding masks. Even if only one client drops, the running time at the client

[2]The code of the paper can be found at: GitHub repository.

N. Clients	Dim.	Drop.	Client						Server		
			Wall-clock running time (s)			Total data transfer (sent/received) (MB)			Wall-clock running time (s)		
			SecAgg	**FTSA**	**Eagle**	**SecAgg**	**FTSA**	**Eagle**	**SecAgg**	**FTSA**	**Eagle**
512	10^5	0.0	35.32	7.81	**7.15**	**0.49**	0.71	0.64	496.12	52.08	**18.08**
		0.1	35.39	48.77	**7.14**	**0.49**	1.35	0.64	2167.81	6516.23	**17.61**
		0.3	35.48	48.79	**7.13**	**0.49**	1.34	0.64	5527.75	5020.67	**16.81**
	10^6	0.0	336.23	72.99	**70.86**	**3.30**	6.47	6.40	2376.32	390.31	**120.21**
		0.1	336.91	404.82*	**71.11**	**3.30**	12.87*	6.40	19994.62	> 7d.	**116.59**
		0.3	340.87	403.86*	**70.71**	**3.30**	12.87*	6.40	52499.26	> 7d.	**107.28**
1024	10^5	0.0	73.02	8.90	**7.55**	0.67	0.79	**0.66**	2904.75	187.34	**45.94**
		0.1	72.67	94.21	**7.36**	0.67	1.45	**0.66**	9449.02	29534.22	**45.03**
		0.3	72.61	93.79	**7.40**	0.68	1.43	**0.66**	22853.70	23470.55	**43.08**
	10^6	0.0	667.20*	75.46	**72.75**	**3.60***	6.70	6.57	> 7d.	861.19	**193.16**
		0.1	655.43*	739.85*	**72.81**	**3.60***	13.20*	6.57	> 7d.	> 7d.	**185.30**
		0.3	658.34*	743.77*	**72.49**	**3.60***	13.20*	6.57	> 7d.	> 7d.	**164.72**

Table 4.3.: Computation and communication costs per client and computation costs for the server, for one SyncFL round. Comparison with SecAgg and FTSA. Values highlighted with "*" are the result of an estimation since the underlying experiments took more than seven days and hence were aborted.

in **FTSA** increases significantly because the reconstruction of dropped clients' inputs is performed over all model parameters and not over a key such as in **Eagle** (see Section 4.6). When $n = 1024$, $d = 10^6$ and $\delta = 0.1$, **Eagle** is approximately $10\times$ faster than the two other solutions. Regarding the communication cost, when dealing with large client inputs, **SecAgg** exhibits the best communication cost, as also identified in [Mansouri, 2022]. This is primarily because **FTSA** and **Eagle** use vector encoding, whereas **SecAgg** implements secure masking. On the other hand, our protocol always shows better results compared with **FTSA**. This is achieved thanks to the **TJL** protection of a scalar instead of a vector. In conclusion, on the client's side, we believe that **Eagle** shows its best performance when dropouts would most probably happen.

We have also evaluated the computation cost of the FL server for one SyncFL round, and depict the results in Table 4.3. We do not show the communication cost as this would correspond to n times the communication cost of one FL client. We observe that the running time of the FL server in **SecAgg** and **FTSA** increases with the dropout rate, while this trend is reversed when it comes to **Eagle**. The reason behind this performance comes from the fact that the number of online clients decreases, and consequently the aggregation time, when the dropout rate increases.

4.7.4 Eagle vs. Flamingo

To compare the performance of **Eagle** with **Flamingo**, we have emulated **Flamingo**'s environment and re-implemented **Eagle** accordingly. As detailed in Section 4.5, **Flamingo** employs a pairwise masking scheme built upon the creation of a random sparse graph and introduces k decryptors, which correspond to special clients helping the server reconstruct the aggregate. We incorporate the concept of decryptors in **Eagle** by involving them during the reconstruction phase. Accordingly, the threshold value t is set to $\frac{2}{3}k$. We have

N. Clients	Dim.	Drop.	Regular client / Decryptor				Server	
			Wall-clock running time (s)		Total data transfer (sent/received) (MB)		Wall-clock running time (s)	
			Flamingo	**Eagle**	**Flamingo**	**Eagle**	**Flamingo**	**Eagle**
512	10^5	0.0	**0.30** / 0.06	7.13 / **0.02**	**0.31** / 0.04	0.64 / **0.01**	239.91	**11.47**
		0.1	**6.44** / 3.67	7.14 / **0.02**	**0.31** / 0.38	0.64 / **0.01**	742.09	**11.00**
		0.3	15.53 / 20.33	**7.09** / **0.02**	**0.31** / 1.95	0.64 / **0.01**	2936.63	**10.15**
	10^6	0.0	**2.50** / 0.11	70.57 / **0.02**	**3.12** / 0.04	0.64 / **0.01**	2049.78	**113.83**
		0.1	**64.78** / 3.84	70.65 / **0.02**	**3.12** / 0.38	0.64 / **0.01**	4799.36	**109.76**
		0.3	156.50 / 20.69	**70.93** / **0.02**	**3.12** / 1.95	0.64 / **0.01**	18720.37	**101.48**
1024	10^5	0.0	**0.07** / 0.13	7.33 / **0.02**	**0.33** / 0.08	0.64 / **0.01**	66.12	**16.38**
		0.1	12.13 / 14.90	**7.30** / **0.02**	**0.33** / 1.44	0.64 / **0.01**	1807.76	**15.48**
		0.3	28.54 / 79.57	**7.41** / **0.02**	**0.33** / 7.49	0.64 / **0.01**	10381.25	**13.54**
	10^6	0.0	**4.38** / 0.35	72.46 / **0.02**	**3.25** / 0.08	0.64 / **0.01**	683.48	**163.41**
		0.1	129.91 / 14.94	**72.44** / **0.02**	**3.25** / 0.08	0.64 / **0.01**	12703.08	**154.31**
		0.3	300.20 / 78.74	**72.26** / **0.02**	**3.25** / 7.46	0.64 / **0.01**	75420.51	**135.11**

Table 4.4.: Computation and communication costs per regular client and decryptor, and computation costs for the server, for one SyncFL round. Comparison with **Flamingo** with a number of decryptors set to 60.

conducted similar experiments as above, with the addition of the value $k = 60$ as in [Ma, 2023].

Table 4.4 shows our experimental results for regular clients, decryptors and server. As expected, **Flamingo** is the preferred choice when the dropout rate is null. However, as the dropout rate increases ($\delta \geq 0.1$), **Flamingo** becomes more costly in terms of regular client computation. This is mainly due to the increasing number of the clients' neighbors, as detailed in Section 4.6.2. On the other hand, **Flamingo** is always better in term of communication since its plaintext space is smaller than **Eagle**. Regarding decryptors, **Eagle** is consistently better for both computation and communication, primarily due to the costly reconstruction operations for the pairwise masks in **Flamingo**. To summarize, **Eagle** shows its best performance with $n = 1024$, $d = 10^5$ and a dropout rate exceeding $\delta = 0.1$ when compared with **Flamingo**. Specifically, **Eagle** is $\times 4$ better for computation and $\times 3$ better for communication in the aforementioned scenario.

4.7.5 Owl

We also run the previously described experiments for **Owl** and **LightSecAgg** [So, 2022] in the asynchronous setting. In Table 4.5, we report the wall-clock time and size of the data transferred within a single FL round, for one FL client and one FL server respectively. Firstly, we observe that **Owl** always exhibits better computation time, as already identified in the asymptotic analysis (see Section 4.6.3). In **LightSecAgg**, the computation time increases with the dropout rate, mainly because the reconstruction step is conducted by fewer online clients. Furthermore, the computation and communication costs of **LightSecAgg**, with a model size larger than 10^5 and a buffer size larger than 512, could not be measured as the execution took more than seven days. To conclude, **Owl** exhibits the best performance in AsyncFL.

N. Clients	Dim.	Drop.	Client Wall-clock running time LightSecAgg	Client Wall-clock running time Owl	Client Total data transfer (sent/received) (MB) LightSecAgg	Client Total data transfer (sent/received) (MB) Owl	Server Wall-clock running time LightSecAgg	Server Wall-clock running time Owl
512	10^5	0.0	> 7d	**7.72s**	–	**0.96**	> 7d	**12.85s**
		0.1	> 7d	**7.66s**	–	**0.96**	> 7d	**12.13s**
		0.3	> 7d	**7.78s**	–	**0.96**	> 7d	**10.77s**
	10^6	0.0	> 7d	**71.28s**	–	**6.72**	> 7d	**116.60s**
		0.1	> 7d	**72.00s**	–	**6.72**	> 7d	**111.80s**
		0.3	> 7d	**71.60s**	–	**6.72**	> 7d	**102.58s**
1024	10^5	0.0	> 7d	**12.88s**	–	**1.31**	> 7d	**22.29s**
		0.1	> 7d	**9.48s**	–	**1.30**	> 7d	**20.31s**
		0.3	> 7d	**9.46s**	–	**1.30**	> 7d	**16.65s**
	10^6	0.0	> 7d	**74.76s**	–	**7.21**	> 7d	**171.74s**
		0.1	> 7d	**75.12s**	–	**7.21**	> 7d	**159.11s**
		0.3	> 7d	**74.50s**	–	**7.21**	> 7d	**137.53s**

Table 4.5.: Computation and communication costs per client and computation costs for the server, for one AsyncFL round. Comparison with **LightSecAgg**. "> 7d" denotes experiments with an overall execution taking more than seven days.

Realistic Use Cases

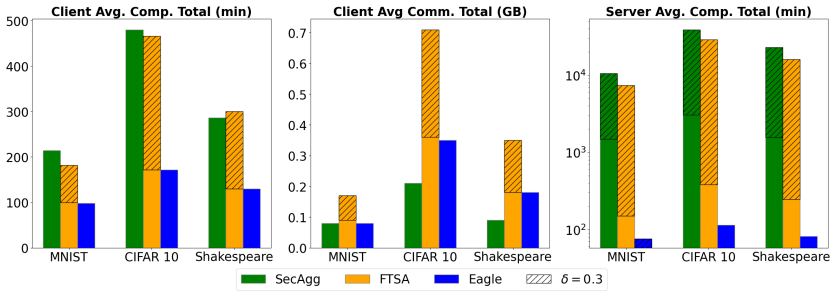


Figure 4.4.: FL training simulation of **Eagle**, **SecAgg** and **FTSA** (δ is the dropout rate).

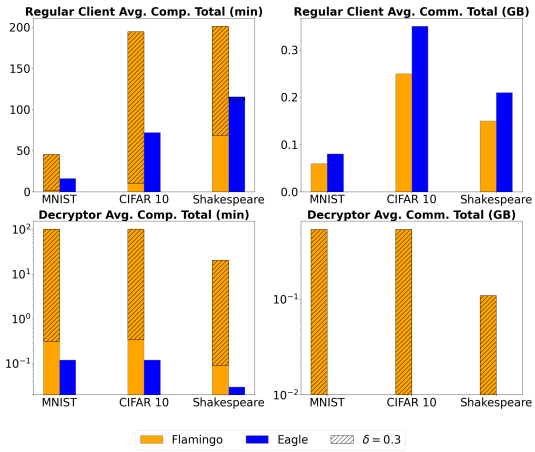


Figure 4.5.: FL training simulation of **Eagle** and **Flamingo** with 60 decryptors (δ is the dropout rate).

In Figures 4.4 and 4.5, we report the results of experiments conducted on three FL tasks, namely MNIST [LeCun, 1998] ($d = 61k$ parameters, 0.99 accuracy), CIFAR-10 [Krizhevsky, 2014] ($d = 270k$ parameters, 0.83 accuracy), and Shakespeare [Caldas, 2018] ($d = 819k$ parameters, 0.56 accuracy). We consider a first scenario without client failure and another scenario with client failure with a dropout rate set to 30%. We first

compare **Eagle** against **SecAgg** and **FTSA**. When decryptors are available, we compare **Eagle** against **Flamingo**. Neural network training is performed using Python in PyTorch framework [Paszke, 2019] without GPU acceleration. For each online communication round, we consider a timeout of 10 seconds as in [Ma, 2023]. We set $n = 400$ and use SGD as the training algorithm with learning rate η, number T of FL rounds, batch size B, number E of epochs, and number S of samples. More precisely: (i) for MNIST: $T = 300$, $\eta = 0.1$, $B = 32$, $E = 5$ and $S = 150$; (ii) for CIFAR-10: $T = 300$, $\eta = 0.1$, $B = 8$, $E = 4$ and $S = 125$; (iii) for Shakespeare: $T = 60$, $\eta = 0.3$, $B = 8$, $E = 1$ and $S = 2000$. Model parameter updates are converted to 8-bit fixed point values by applying 8-bit probabilistic quantization with 7 fractional bits [Konečnỳ, 2016]. The results show that in all three datasets, with a dropout rate $\delta = 0.3$, **Eagle** always outperforms previous works in terms of total computation.

4.8 Conclusion

We have studied the problem of stragglers in FL, which have a non-negligible impact on the performance and robustness of SA protocols. To cope with this problem, we have considered stragglers as client dropouts and developed two new SA protocols, namely **Eagle** and **Owl**. **Eagle** in SyncFL does not depend on dropouts anymore, and hence is more efficient than existing works, especially when the number stragglers is non-negligible. **Owl** in AsyncFL does not suffer from stragglers inherently, and is thus more efficient than the only existing solution in asynchronous settings.

As part of future work, we aim to optimize the cost of **Owl** and to consider stronger threat models whereby both FL clients and the server can be malicious and modify the actual aggregate model. In such a setting, honest FL clients should be able to verify the correctness of the computation of the aggregate value.

Acknowledgments

We thank the ARES reviewers for their comments. We also thank Mohamed Mansouri for his help with **FTSA**, Yiping Ma for his assistance with **Flamingo**, and Lucia Innocenti for the helpful discussions about client selection and dropouts. This work has been supported by the French government, through the 3IA Côte d'Azur Investments in the Future project managed by the National Research Agency (ANR) with the reference number ANR-19-P3IA-0002, by the TRAIN project ANR-22-FAI1-0003-02, and by the ANR JCJC project Fed-BioMed 19-CE45-0006-01.

5 Buffalo: A Practical Secure Aggregation Protocol for Asynchronous Federated Learning

Contents

5.1 Introduction **68**
5.2 Related work **70**
5.3 Background **72**
5.3.1 Synchronous Federated Learning 72
5.3.2 Buffered Asynchronous Federated Learning 73
5.3.3 SA Threat Model and Security. 75
5.4 Building Blocks **76**
5.4.1 Joye-Libert SA 76
5.4.2 LWE-based SA 77
5.4.3 Aggregation Verifiability 77
5.5 Our protocols **78**
5.5.1 Buffalo 78
5.5.2 Buffalo+ 82
5.6 Complexity Analysis **85**
5.7 Experimental Results **86**
5.7.1 Overall performance of BAsyncFL schemes 90
5.8 Conclusion **92**

In this chapter, we introduce a pioneering approach to Secure Aggregation (SA) in the context of Buffered Asynchronous Federated Learning (BAsyncFL), a paradigm that has revolutionized the collaborative training of Machine Learning (ML) models while preserving data privacy. Traditional synchronous FL methods face challenges due to stragglers, or slow clients, which delay the training process. BAsyncFL addresses these inefficiencies by allowing clients to update the global model as they complete their local computations, thereby eliminating synchronization constraints. However, existing SA techniques, which enable servers to aggregate client updates without learning individual updates and thus prevent inference attacks, are not easily applicable to asynchronous settings due to their need for synchronized rounds.

To bridge this gap, we present **Buffalo**, the first practical SA protocol specifically designed for BAsyncFL. **Buffalo** utilizes lattice-based encryption to manage the scalability challenges associated with large ML models and introduces a novel role, the *decryptor*, to assist the server in the aggregation phase. To further enhance client trust and ensure the integrity of the aggregation process, we propose **Buffalo+**, an extension of **Buffalo** that incorporates verifiable SA. This extension allows clients to verify that their updates have been accurately included in the global model.

Our comprehensive evaluation, encompassing both theoretical analysis and experimental validation on real-world datasets, demonstrates the efficiency and practicality of our protocols. Both **Buffalo** and **Buffalo+** significantly enhance client computation efficiency and ensure robust, scalable SA in BAsyncFL environments.

This chapter is currently under review.

5.1 Introduction

Cross-device Federated Learning (FL) [McMahan, 2017a] has rapidly emerged as a dominant paradigm for collaboratively training Machine Learning (ML) models while maintaining the privacy of individual data. In the traditional Synchronous FL (SyncFL) framework, a central server initializes the global model and sends these parameters to a pre-selected subset of clients. These selected clients optimize the model parameters using their local data and transmit their updates to the server for aggregation, which typically consists of weighted averaging. The same process is repeated through several rounds until the model reaches a certain level of accuracy.

However, sharing local model parameters introduces vulnerabilities that can inadvertently expose sensitive client data to risks such as membership inference or model inversion attacks [Shokri, 2017; Nasr, 2019]. To mitigate these risks, Secure Aggregation (SA) techniques have been proposed [Mansouri, 2023; Bell, 2020; Bell, 2023; Ma, 2023; Mansouri, 2022], which protect client updates prior to transmission.

Despite its advantages, FL faces challenges due to the heterogeneity of the clients involved, differing significantly in storage, communication, and computation capabilities. Notably, the presence of *stragglers*, i.e. slower participating clients, can severely delay the training rounds, especially in traditional FL frameworks which assume that FL clients are strongly synchronized with the server. Such a delay can adversely affect the overall performance of the FL system [Bonawitz, 2019a; Nguyen, 2022].

One strategy to manage *stragglers* in a SyncFL environment is the over-selection of clients [Bonawitz, 2019a]. This approach involves selecting a larger subset of clients than necessary in anticipation that some will not complete their tasks promptly. However, this method can lead to inefficiencies, as slow clients that have locally trained the model might not be included in the updated global model, leading to wasted computational resources and potential loss of diverse data contributions. Additionally, it significantly affects SA since non-selected clients are treated as dropped, increasing protocol complexities.

Asynchronous Federated Learning (AsyncFL) frameworks, as described by [Xie, 2019], address inefficiencies by processing client updates as they arrive, thus eliminating the need for synchronized rounds. However, in pure asynchronous FL methods, each client update results in an immediate server model update. This approach poses challenges for privacy, as traditional SA techniques become inadequate. SA relies on aggregating multiple updates to protect individual contributions, which is not possible in a pure asynchronous setting where updates are processed individually.

To address these issues, a compatible privacy-preserving asynchronous framework has been proposed: Buffered AsyncFL (BAsyncFL) [Nguyen, 2022], where a server collects local inputs in a buffer and updates the global model periodically, i.e., every time the buffer is full. Nonetheless, existing SA solutions typically assume a synchronous environment, necessitating prior knowledge by clients of their participation, complicating their direct application to BAsyncFL.

Our contributions in this paper include:

• We introduce **Buffalo**, the first practical SA protocol designed specifically for BAsyncFL. **Buffalo** employs lattice-based encryption, coupled with homomorphic encryption on keys rather than on models, to address scalability issues related to the dimensions of machine learning models. We also introduce a new role, the *decryptor*, to assist the server during the aggregation phase. Clients generate on-the-fly keys for their updates, and *decryptors* help the server reconstruct these keys, making the process asynchronous and avoiding synchronization issues. This ensures efficient and secure aggregation without requiring synchronized rounds.

• To further incentivize client participation and ensure that clients' local computations are considered for aggregation, we propose **Buffalo+**, an asynchronous verifiable SA protocol. **Buffalo+** allows clients to verify that the server has correctly included their updates in the aggregation process, ensuring trust in the distributed training.

• Evaluating our protocols and comparing them with existing solutions adapted to BAsyncFL in real asynchronous simulations, using benchmark datasets and a novel medical dataset [Aleppo, 2017]. Our evaluations include both theoretical and experi-

mental validations of our protocols across three real datasets. Our results showcase the practicality and effectiveness of our solutions, particularly in medical applications.

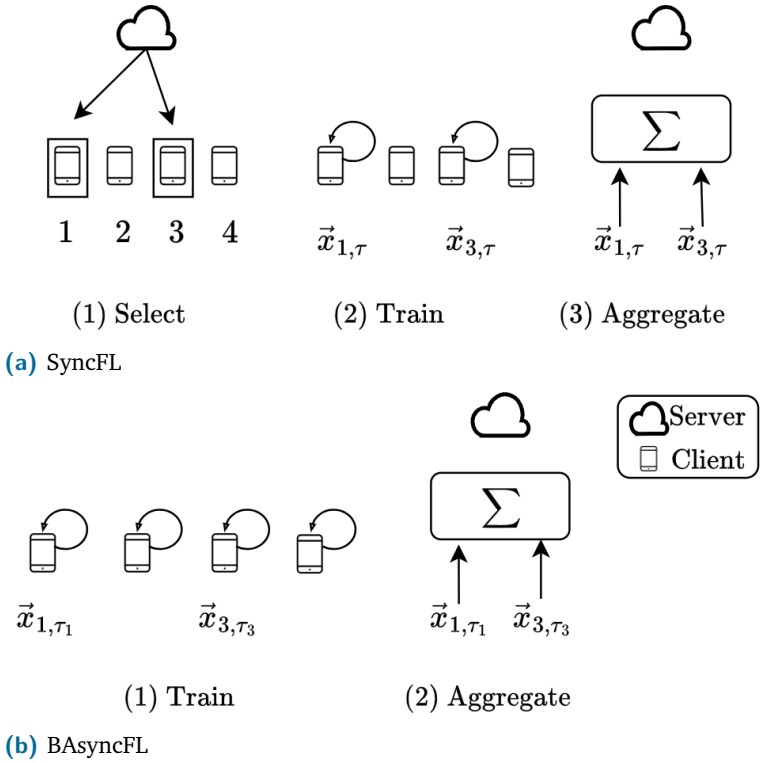


Figure 5.1.: Illustrations of SyncFL (a) and BAsyncFL (b). SyncFL: (1) The server randomly selects two clients, here clients 1 and 3; (2) The selected clients locally train and produce updated local models $\vec{x}_{1,\tau}$ and $\vec{x}_{3,\tau}$ resp.; (3) The server aggregates these local models. BAsyncFL: (1) All available clients start local training asynchronously such that clients 1 and 3 are the fastest and thus send their updated local models $\vec{x}_{1,\tau_1}$ and $\vec{x}_{3,\tau_3}$ resp., to fill the buffer on the server; (2) The server aggregates the received local models.

5.2 Related work

SA for FL- SA for FL faces challenges with client dropouts and failures. Bonawitz et al. [Bonawitz, 2017a] introduced SecAgg, the first single-server SA protocol, using Shamir's secret sharing and masking techniques. Subsequent enhancements include SecAgg+ [Bell, 2020] and ACORN [Bell, 2023] utilizing sparse graphs and lattice-based masking. Stevens et al. [Stevens, 2022] proposed DPSecAgg, a scheme that replaces standard masking with lattice-based masking and employs a packed version of secret sharing over the lattice-based mask. Similarly, So et al. [So, 2022] developed LightSecAgg, which utilizes packed Shamir secret sharing techniques over inputs.

Recent works have also aimed to reduce the number of communication rounds required in the protocol and eliminate the need for a trusted third party in actively secure models. Protocols such as Flamingo [Ma, 2023] and LERNA [Li, 2023] introduce special

roles, called *decryptor* and *member committees* respectively, to aid the server during the aggregation process.

There are also solutions with two or more non-colluding servers to achieve distributed trust and privacy preservation [Addanki, 2022; Corrigan-Gibbs, 2017; Rathee, 2023]. A promising approach named ELSA [Rathee, 2023] leverages two servers to aggregate vectors. Their scheme can withstand model poisoning attacks by detecting and filtering out boosted gradients. In our work, we consider only solutions with a single server.

Buffered Asynchronous SA- Existing solutions, such as SecAgg, SecAgg+, ACORN, Flamingo and LERNA [Bonawitz, 2017a; Bell, 2020; Bell, 2023; Ma, 2023; Li, 2023], share a common characteristic: they operate within a synchronized FL framework. In this setting, each selected client submits its updated model or data in alignment with a synchronized FL round. This synchronization typically relies on information shared among the selected clients or utilizes round-specific information.

In contrast, LightSecAgg demonstrates that it can adapt to BAsyncFL settings. However, this solution requires involvement from the entire FL client population, leading to high computational complexity and communication overhead. DPSecAgg [Stevens, 2022], while not originally designed for BAsyncFL, shows adaptability to such settings, we provide more details in Section 5.3.2.

Verifiable SA- SA protocols prevent a curious FL server from learning about the clients' inputs but do not protect against a malicious server that might modify the aggregated model and the users to verify the correct aggregate. The initial FL solution with verifiable SA, introduced by Xu et al. [Xu, 2019], allows clients to verify the correctness of aggregation through a proof generated by the server. However, the communication overhead of this method, proportional to the model size, is impractical for large-scale FL systems.

An alternative by Guo et al. [Guo, 2020] involves hashing clients' local model updates for verification but was later found to be insecure, and fixed in [Guo, 2022]. Subsequently, Buyukates et al. [Buyukates, 2022] introduced LightVeriFL, a more scalable approach that utilizes elliptic curve-based secret sharing techniques, effectively reducing both computation and communication costs, particularly in scenarios involving client dropout.

5.3 Background

In this section, we review both SyncFL and AsyncFL framwworks, along with SA techniques. Notations can be found in Appendix D.1.

Setup: Server initializes the global model $\vec{x}_0$, the empty buffer set $\mathcal{U}_{\text{BUFF}} = \emptyset$ and a set of available clients $\mathcal{U}' = \mathcal{U}$

ServerAggregation($\vec{x}_\tau, \mathcal{U}_{\text{BUFF}}, \mathcal{U}'$)

Server repeats steps 1-4 until convergence criteria is reached

1. Run **ClientUpdate**($\vec{x}_\tau$) on $\mathcal{U}'$ asynchronously
2. If Client u's input has been submitted:
   Receive input $\vec{x}_{u,\tau_u} \leftarrow$ from Client u
   $\mathcal{U}_{\text{BUFF}} \leftarrow \mathcal{U}_{\text{BUFF}} \cup \{u\}$
   $i \leftarrow i + 1$
3. If $i == n$:
   $\vec{x}_\tau \leftarrow \sum_{u \in \mathcal{U}_{\text{BUFF}}} \vec{x}_{u,\tau_u}$
   Reset buffer: $\mathcal{U}_{\text{BUFF}} \leftarrow \emptyset$, $i \leftarrow 0$, $\tau \leftarrow \tau + 1$
4. Set available clients $\mathcal{U}' = \mathcal{U}_{\text{BUFF}}$

ClientUpdate($\vec{x}$)

Client $u \in \mathcal{U}'$ proceeds as follows

1. Receive global model $\vec{x}$ from Server
2. $\vec{y}_{u,0} \leftarrow \vec{x}$
3. Perform local SGD updates: $\vec{y}_{u,q} =$ **LocalSGD**($\vec{y}_{u,0}, q, \eta$)
4. Compute update difference: $\vec{x}_{u,\tau_u} \leftarrow \vec{y}_{u,0} - \vec{y}_{u,q}$
5. Return $\vec{x}_{u,\tau_u}$ to Server

Figure 5.2.: FedBuff Algorithm

5.3.1 Synchronous Federated Learning

As introduced by McMahan et al. [McMahan, 2017a], FL consists of a distributed ML framework where a set $\mathcal{U}$ of clients ($|\mathcal{U}| = n_{tot}$) collaboratively trains a global model $\vec{x} \in \mathbb{R}^d$ under the guidance of a FL server, as reported in Figure 5.1a. One of the first and popular methods used to train a FL model is FedAvg [McMahan, 2017a]. With FedAvg, at each FL round τ, the server selects a subset $\mathcal{U}^{(\tau)} \subseteq \mathcal{U}$ of clients ($|\mathcal{U}^{(\tau)}| = n \leq n_{tot}$) through *client selection* [McMahan, 2017a], Fig. 5.1a (1). Each client $u \in \mathcal{U}^{(\tau)}$ trains the model $\vec{x}_{u,\tau}$ on its private local data $\mathcal{D}_u$, for example through Stochastic Gradient Descent (SGD) [Ruder, 2016], Fig. 5.1a (2), and forwards this updated model $\vec{x}_{u,\tau}$ to the server. When the server receives the updated models from all clients in $\mathcal{U}^{(\tau)}$, it proceeds to the aggregation step, Fig. 5.1a (3), by computing the average of these models and updating the round counter as follows: $\vec{x}_\tau \leftarrow \frac{1}{n} \sum_{u \in \mathcal{U}^{(\tau)}} \vec{x}_{u,\tau}$ and $\tau \leftarrow \tau + 1$. This iteration proceeds until the global model $\vec{x}$ exhibits some desired level of accuracy. This approach to FL requires a Synchronous FL (SyncFL) setting whereby FL clients should be synchronized and participate on a round-by-round basis. Usually, in such a setting, $n_{tot} \in [10^6, 10^{10}]$ and $n \in [50, 5000]$ (see [Kairouz, 2019]).

5.3.2 Buffered Asynchronous Federated Learning

SyncFL settings usually are slowed down by *stragglers*, i.e. slow clients [Nguyen, 2022]: a FL round is completed only when all selected clients send their updated model (see Figure5.1a). Hence, the impact of *stragglers* becomes significant especially when the set of clients is system heterogeneous. To mitigate such a problem, some systems, like those described by Bonawitz et al. [Bonawitz, 2019a], employ client over-selection, where the size of the subset of selected clients is usually increased by 30% in order to reach the actual sufficient number of model updates to run FedAvg. This means that to execute FedAvg with 1000 client inputs, 1300 clients are selected, and the FL round concludes whenever the server receives 1000 updates from the most rapid clients. Hence SyncFL becomes straggler-resistant with a non-negligible cost of over-selection.

As an alternative, Buffered Asynchronous FL (BAsyncFL) is proposed to remove the need for synchronization and hence avoid the effect of late arrivals (see Figure 5.1b). FedBuff [Nguyen, 2022] is introduced as a buffered asynchronous framework whereby the FL server collects local models received from clients in a buffer and updates the global model whenever this buffer is full. As reported in Figure 5.2, each client $u \in \mathcal{U}'$ runs its local round τ_u such that the local update is denoted as $\vec{x}_{u,\tau_u}$. Note that the training round is specific to each client: for another client $v \neq u$, $\tau_v \neq \tau_u$. When the first n clients fill the buffer $\mathcal{U}_{\text{BUFF}}$, the server computes the aggregation and resets the buffer. This process is repeated until a convergence criterion is reached. In this setting, *stragglers*' inputs are still taken into account as they will eventually fill a future buffer.

Secure Aggregation

Many studies, such as [Nasr, 2019; Shokri, 2017], have shown that, although FL clients train the model locally and keep their datasets $\mathcal{D}_u$ private in their premises, model updates that are shared with the FL server do leak information about the local datasets. Hence, local model updates should also remain confidential even against the FL server. As already shown in [Bonawitz, 2017a; Mansouri, 2022; Bell, 2020; Ma, 2023; Li, 2023], the main solution to prevent such a leakage is the use of Secure Aggregation (SA) which enables the FL server, named the *aggregator*, to compute the average of model updates, i.e. the global model parameters, without having access to the clients' individual updates. In Figures 5.1a and 5.1b, we illustrate the processes of SA in SyncFL and AsyncFL environments, respectively.

Towards SA for BAsyncFL

In this paper, we investigate the design of SA protocols for the BAsync FL setting. An initial study from [So, 2022] highlights the incompatibility of current SA protocols with the BAsyncFL setting (see Appendix F.2 of [So, 2022] for more details). The authors identify that this incompatibility stems from the fact that clients know in advance which other clients are participating to the FL round, and protect their input accordingly. In Figure 5.1a - step (1), two clients are selected by the server and thus protect their local inputs based on this selection. In general, in SyncFL settings, the n_{tot} clients in $\mathcal{U}$ must learn who are the n clients selected for a given round τ. Moreover, in order to overcome *stragglers*, over-selection is performed, meaning that $n + 0.3 \cdot n$ clients have been actually selected (when fixing over-selection at 30% as suggested above). Once this knowledge has been acquired, each selected client submits their updated input using some information specifically shared with other selected clients. Such a constraint comes against the idea behind buffered asynchronicity where clients train and submit their inputs at their own pace, without following what other clients are doing. In Figure 5.1b - step (1), two clients have finished their training and simply submit their local inputs to the server without noticing and requiring other clients. Therefore, to successfully develop SA protocols for BAsync FL settings, we must overcome the above constraint by avoiding clients requiring to know which other clients participate to the current round, as proposed in [So, 2022].

Following the last remark, we identify two potential SA protocols that can be easily transformed to be compatible with BASyncFL: LightSecAgg [So, 2022] and DPSecAgg [Stevens, 2022]. Indeed, in both LightSecAgg and DPSecAgg, clients do not need to know who is participating to the actual round τ. Nevertheless, those solutions have been first designed for SyncFL, and thus assume that all n_{tot} clients are online during the aggregation phase. Indeed, clients secret share their inputs with all clients to enable successful aggregation. This is due to the fact that clients are missing the information related to client participation for a specific round. We recall that $n_{tot} >> n$, hence making such a design really inefficient. One way to overcome such a limitation is to let the server provide this missing information to the clients, resulting into an extra communication round. Another way to make LightSecAgg and DPSecAgg compatible in BAsyncFL settings is to introduce several special clients that must remain online, called *decryptors*, whose task is to help the server reconstruct the aggregation key and hence the aggregate result.

Based on this last suggestion, we propose a new, more efficient SA protocol, called **Buffalo**, suitable with the BASync setting. In **Buffalo**, the client's input is first encrypted using a lattice-based encryption scheme. The key used for the first encryption is further protected using a second encryption scheme. Thanks to this additional encryption step, a

scalar value, rather than a substantial vector value, needs to be secret shared and further reconstructed as the first aggregation key. The server then obtains the second, lattice-based, aggregation key from the first aggregation key to finally recover the aggregate result.

Aggregation Verifiability

Furthermore, inspired by [Buyukates, 2022], we propose a second protocol, called **Buffalo+**, that extends **Buffalo** by allowing clients to verify whether the actual aggregate result is correct and takes their inputs into account. The solution follows the same building block as in LightVeriFL [Buyukates, 2022], namely homomorphic hash functions, digital signatures and commitments. Nevertheless, our solution reaches significant performance improvements based on the observation that most of the client's model parameters from one round to another one remain unchanged. Consequently, in **Buffalo+**, clients do not need to recompute the hash value of their entire updated input, but only the hash value based on the modified parts of this new input compared to the previous one. This way, the computation cost at the client is decreased significantly compared to LightVeriFL.

5.3.3 SA Threat Model and Security.

Similar to the related work [Ma, 2023], we assume a static, malicious adversary that corrupts the server and up to a fraction γ of the total number n_{tot} of clients in the system.

In **Buffalo** and **Buffalo+**, there are two types of clients in the system: regular clients that provide their input to the server, and special clients, called *decryptors*, whose job is to help the server recover the final result.

Contrary to existing fault-tolerant solutions which must consider dropouts both at client and decryptor sides when elaborating their security proofs, our protocols only rely on the availability of the *decryptors*. We distinguish between the fraction γ_n of corrupted regular clients and the fraction γ_k of corrupted *decryptors*. In particular, we examine the following types of decryptor failures: *(i)* honest *decryptors* that disconnect or are too slow to respond as a result of unstable network conditions, power loss, etc; *(ii)* arbitrary actions by an adversary that controls the server and a bounded fraction γ of clients. We upper bound the number of *decryptors* that drop out in any given aggregation round by the fraction δ_d. More details about the above examination are given in [Ma, 2023] (Appendix A). Moreover, we explore attacks against the integrity of the aggregation computation by embedding verifiability mechanisms in **Buffalo+**. Given a threshold t

and the number k of *decryptors* in the system, we require in both **Buffalo** and **Buffalo+** that the number of honest and alive *decryptors* has to be $t > \frac{2k}{3}$ [Bonawitz, 2017a]. We consider denial of services attacks as out of scope.

5.4 Building Blocks

In this section, we introduce the main cryptographic primitives utilized as foundational elements in our two protocols. The remaining cryptographic primitives are presented in Appendix D.2. Notations can be found in Appendix D.1.

Secure Aggregation

We are interested in SA schemes that are homomorphic with relation to the secret key sk_u and input x_u: $\sum_{u\in\mathcal{U}} \textbf{Protect}(sk_u, x_u) = \textbf{Protect}(-\sum_{u\in\mathcal{U}} sk_u, \sum_{u\in\mathcal{U}} x_u)$.

Namely, we describe the Joye-Libert (JL) scheme and a lattice-based scheme relying on the Learning With Errors (LWE) problem. For sake of simplicity, we denote the latter as the LWE scheme. Both schemes are used in **Buffalo** and **Buffalo+**.

5.4.1 Joye-Libert SA

Let us consider n clients and one aggregator. The JL scheme [Joye, 2013] is defined with three algorithms.

• $(sk_0, \{sk_u\}_{u\in[1,n]}, pp) \leftarrow$ **JL.Setup**(λ): Given the security parameter λ, this algorithm generates two large, equal-size prime numbers p and q and sets the modulus $N = pq$. It randomly generates n client secret keys $sk_u \in \mathbb{Z}_{N^2}$ and computes the aggregator secret key $sk_0 = -\sum_{u=1}^{n} sk_u$. Then, it defines a cryptographic hash function $F : \mathbb{Z} \to \mathbb{Z}^*_{N^2}$. It outputs the $n+1$ secret keys and the public parameters $pp = (N, F)$.

• $y_{u,\tau} \leftarrow$ **JL.Protect**$(pp, sk_u, \tau, x_{u,\tau})$: This algorithm encrypts private input $x_{u,\tau} \in \mathbb{Z}_N$ for time period τ using secret key $sk_u \in \mathbb{Z}_{N^2}$, resulting in ciphertext $y_{u,\tau} = (1 + x_{u,\tau}N) \cdot F(\tau)^{sk_u} \mod N^2$.

• $x_\tau \leftarrow$ **JL.Agg**$(pp, sk_0, \tau, \{y_{u,\tau}\}_{u\in[1,n]})$: This algorithm aggregates the n protected inputs from clients received at time period τ to obtain $y_\tau = \prod_{u=1}^{n} y_{u,\tau}$. It then decrypts y_τ to recover the plaintext aggregate $x_\tau = \sum_{u=1}^{n} x_{u,\tau} = (F(\tau)^{-sk_0} \cdot y_\tau - 1)/N \mod N$.

The JL scheme ensures *Aggregator Obliviousness* under the Decision Composite Residuosity (DCR) assumption [Paillier, 1999], in the random oracle model and assuming that each client u encrypts only one input $x_{u,\tau}$ per time period τ [Joye, 2013].

5.4.2 LWE-based SA

Several SA solutions [Bell, 2023; Li, 2023] have their security relying on the LWE assumption. Such SA schemes are parameterized by a ring R of degree m over $\mathbb{Z}$, an integer modulus $q > 0$ defining a quotient ring $R_q = R/qR$, and two distributions χ_s, χ_e over R. Let d be the length of the client's vector input $\vec{x}_u$. Let us consider n clients and one aggregator. The LWE scheme is defined with three algorithms.

• $pp \leftarrow$ **LWE.Setup**(λ): Given security parameter λ, this algorithm generates public matrix $A \in \mathbb{Z}_p^{m \times d}$ which is defined as the public parameters $pp = A$.

• $\vec{y}_u \leftarrow$ **LWE.Protect**$(pp, \vec{s}_u, \vec{x}_u)$: To encrypt a vector $\vec{x}_u \in \mathbb{Z}^d$, the algorithm first samples two vectors, namely the client's secret key $\vec{s}_u \leftarrow \chi_s \subseteq \mathbb{Z}_q^m$ and the error vector $\vec{e} \leftarrow \chi_e \subseteq \mathbb{Z}_q^m$. The ciphertext $\vec{y}_u \in \mathbb{Z}_D^m$ is computed as follows: $\vec{y}_u = A\vec{s}_u + D \cdot \vec{e} + \vec{x}_u \mod q$, where D is the chipertext modulus.

• $\vec{x} \leftarrow$ **LWE.Agg**$(pp, \vec{s}_0, \{\vec{y}_u\}_{u \in [1,n]})$: Let $\vec{s}_0 = \sum_{u \in [1,n]} \vec{s}_u$ be the aggregation key. This algorithm computes the aggregate $\vec{x}$ from the n chipertexts using $\vec{s}_0$ as follows: $\vec{x} = (\sum_{u \in [1,n]} \vec{y}_u - A\vec{s}_0) \mod D$.

The LWE scheme guarantees *Aggregator Obliviousness* under the Hint-LWE problem [Lee, 2018], assuming that each client u encrypts only one input $\vec{x}_u$ per round using the same secret key $\vec{s}_u$ [Bell, 2023].

5.4.3 Aggregation Verifiability

To ensure the integrity of the aggregation operation, a combination of hash functions, signatures, commitments and secret sharing techniques are employed in LightVeriFL [Buyukates, 2022]. In this subsection, we describe the hash process presented in [Buyukates, 2022], and show how we can benefit from its homomorphic properties to improve the aggregation check step. We choose to use the Threshold ElGamal (TEG) encryption scheme (Appendix D.2.3) instead of the Shamir Secret Sharing (SS) scheme (Appendix D.2.2) to reach constant computational costs. We detail other cryptographic primitives in Appendix D.2.4.

Homomorphic Hashing

Let us consider the cyclic group $\mathbb{G}$ of prime order p with generator g. Given d distinct elements $g_1, \ldots, g_d \in \mathbb{G}$, the hash of a vector $\vec{x}_u \in \mathbb{Z}_p^d$ is defined as follows [Bellare, 1994]:

$$h_u = H(\vec{x}_u) = \prod_{i=1}^{d} g_i^{\vec{x}_u[i]} \tag{5.1}$$

where $\vec{x}_u[i]$ represents the ith element of the vector $\vec{x}_u$. This hash function H is additively homomorphic, meaning that for any two vectors $\vec{x}_u, \vec{x}_v$, we have $H(\vec{x}_u + \vec{x}_v) = H(\vec{x}_u) \cdot H(\vec{x}_v)$. Additionally, the hash function H is incrementally computable. Specifically, let two vectors $\vec{x}_u$ and $\vec{x}_v$ differ in only one element: $\vec{x}_u[j] \neq \vec{x}_v[j]$ and $\forall i \neq j, \vec{x}_u[i] = \vec{x}_v[i]$. Thanks to the homomorphic property of H, given $h_u = H(\vec{x}_u)$, one can easily compute h_v using h_u. Indeed, $h_v = H(\vec{x}_v) = h_u \cdot g_j^{-\vec{x}_u[j]+\vec{x}_v[j]}$. Hence, instead of computing h_v from scratch using Eq. 5.1 (as it is done in LightVeriFL [Buyukates, 2022]), one can save exponentiation calculations by using the aforementioned technique. We thus define the function H_{INC} as follows:

$$H_{\text{INC}}(\vec{x}_v, \vec{x}_u, h_u) = \begin{cases} h_u \cdot g_j^{-\vec{x}_u[j]+\vec{x}_v[j]} & \text{if } \vec{x}_u[j] \neq \vec{x}_v[j] \\ \text{do nothing} & \text{otherwise} \end{cases} \tag{5.2}$$

To improve that efficiency of the hashing process, H and H_{INC} can be implemented using Elliptic Curve (EC) points [Buyukates, 2022].

5.5 Our protocols

5.5.1 Buffalo

We present **Buffalo**, a SA protocol in the context of a BAsyncFL setting. The design principles of **Buffalo** are the following:

(1) As opposed to the synchronized setting, clients who contribute to a given buffer are not known in advance. Hence, the aggregation key cannot be pre-computed or pre-defined. Therefore, **Buffalo** defines an 'on-the-fly' generation of the aggregation key once the buffer is full, along with the buffer set $\mathcal{U}_{\text{BUFF}}$. This aggregation key is computed through the help of *decryptors*. They are special clients who receive other clients' contribution to the aggregation key, and collaboratively construct the final version of that key. It is worth noting that, as opposed to the state-of-the-art solutions who make use of *decryptors* to reduce/optimize some computational costs [Ma, 2023; Li, 2023], the purpose here is to deal with potential clients turning offline after sending their

Parties: Server and clients in $\mathcal{U}$, such that $|\mathcal{U}| = n_{tot}$, where *decryptors* belong to $\mathcal{K} \subseteq \mathcal{U}$ such that $|\mathcal{K}| = k$

Public Parameters: input domain $\mathbb{Z}_L^d$; buffer set $\mathcal{U}_{\text{BUFF}}$ whose size is $|\mathcal{U}_{\text{BUFF}}| = n$; security parameter λ for cryptographic primitives; secret sharing threshold t

Prerequisites: For $\mathcal{K}$ and $r \in \{2, 3\}$, we denote $\mathcal{K}_r$ the set of *decryptors* that execute **Round** r, and we denote by $\mathcal{K}'_r$, the set of *decryptors* that completed without dropping out. It holds that $\mathcal{K}'_r \subseteq \mathcal{K}_r \subseteq \mathcal{K}_{r-1}$ for all $r \in \{2, 3\}$.

Setup

Client $u \in \mathcal{U}$:

Keys registration

1. $(c_u^{PK}, c_u^{SK}) \leftarrow$ **KA.Gen**(pp^{KA})
2. $\{d_u^{SK}, d_u^{PK}\}_{u \in \mathcal{U}} \leftarrow$ **Sig.Setup**(λ)
3. Register c_u^{PK} and d_u^{PK} to PKI

Decryptor channel key setup

4. $\forall i \in \mathcal{K}$, $c_{u,i} \leftarrow$ **KA.Agree**(c_u^{SK}, c_i^{PK})

Trusted Dealer (TD):

5. $(\perp, \perp, pp^{\text{JL}}) \leftarrow$ **JL.Setup**(λ)
6. $pp^{\text{LWE}} \leftarrow$ **LWE.Setup**(λ)
7. $(pk, \{[sk]_i\}_{i \in \mathcal{K}}) \leftarrow$ **TEG.Setup**$(t, \mathcal{K}, \lambda)$
8. Register pp^{JL}, pp^{LWE} and pk to PKI
9. Send $[sk]_i$ to Decryptor $i \in \mathcal{K}$

Decryptor $i \in \mathcal{K}$:

10. Receive $[sk]_i$ from TD

Online - Round 1

Client $u \in \mathcal{U}$:

Input protection and authentication

1. $\vec{s}_{u,\tau_u} \xleftarrow{R} \chi_s$ // *Generate LWE secret key for round* τ_u
2. $\vec{y}_{u,\tau_u} \leftarrow$ **LWE.Protect**$(pp^{\text{LWE}}, \vec{s}_{u,\tau_u}, \vec{x}_{u,\tau_u})$ // *Protect input using* **LWE**
3. $h_{u,\tau_u} \xleftarrow{R} H_{\text{INC}}(\vec{x}_{u,\tau_u}, \vec{x}_{u,\tau_u-1}, h_{u,\tau_u-1})$ // *Compute input hash*
4. $(z_{u,\tau_u}, z'_{u,\tau_u}, r_{u,\tau_u}) \xleftarrow{R} E(\mathbb{F}_p)$ // *Generate hash mask, witness mask and witness*
5. $\tilde{h}_{u,\tau_u} = h_{u,\tau_u} + z_{u,\tau_u}$ // *Mask local input hash*
6. $c_{u,\tau_u} \leftarrow$ **COM.Commit**$(h_{u,\tau_u}, r_{u,\tau_u})$ // *Commit to input hash*
7. $\sigma_{u,\tau_u} \leftarrow$ **Sig.Sign**(d_u^{SK}, c_{u,τ_u}) // *Sign commitment*

Key protection and authentication

8. $sk_{u,\tau_u} \xleftarrow{R} \mathbb{Z}_{N^2}$ // *Generate JL secret key for round* τ_u
9. $\langle \vec{s}_{u,\tau_u} \rangle \leftarrow$ **JL.Protect**$(pp^{\text{JL}}, sk_{u,\tau_u}, \tau_0, \vec{s}_{u,\tau_u})$ // *Protect LWE key using JL*
10. $\{(i, [sk_{u,\tau_u}]_i)\}_{i \in \mathcal{K}} \leftarrow$ **SS.Share**$(sk_{u,\tau_u}, t, \mathcal{K})$ // *Secret share JL secret key*
11. $\forall i \in \mathcal{K}$, $\epsilon_{u,i} \leftarrow$ **AE.Enc**$(c_{u,i}, u \mid\mid i \mid\mid [sk_{u,\tau_u}]_i)$ // *Encrypt JL key shares*
12. $\tilde{r}_{u,\tau_u} = r_{u,\tau_u} + z'_{u,\tau_u}$ // *Mask witness*
13. $\langle (z_{u,\tau_u}, z'_{u,\tau_u}) \rangle \leftarrow$ **TEG.Protect**$(pk, (z_{u,\tau_u}, z'_{u,\tau_u}))$ // *Protect hash and witness masks using TEG*
14. $\sigma'_{u,\tau_u} \leftarrow$ **Sig.Sign**$(d_u^{SK}, \langle (z_{u,\tau_u}, z'_{u,\tau_u}) \rangle)$ // *Sign encrypted masks*
15. Send $\vec{y}_{u,\tau_u}$, $\langle \vec{s}_{u,\tau_u} \rangle$, $\{\epsilon_{u,i}\}_{i \in \mathcal{K}}$, c_{u,τ_u}, σ_{u,τ_u}, σ'_{u,τ_u}, $\tilde{h}_{u,\tau_u}$, $\langle (z_{u,\tau_u}, z'_{u,\tau_u}) \rangle$, $\tilde{r}_{u,\tau_u}$ to *Server*

Server:

16. Collect $\{\vec{y}_{u,\tau_u}, \langle \vec{s}_{u,\tau_u} \rangle, \{\epsilon_{u,i}\}_{i \in \mathcal{K}}, c_{u,\tau_u}, \sigma_{u,\tau_u}, \sigma'_{u,\tau_u}, \tilde{h}_{u,\tau_u}, \langle (z_{u,\tau_u}, z'_{u,\tau_u}) \rangle, \tilde{r}_{u,\tau_u}\}_{u \in \mathcal{U}_{\text{BUFF}}}$
17. If $|\mathcal{U}_{\text{BUFF}}| < t$, abort; otherwise, broadcast $\{\epsilon_{u,i}\}_{u \in \mathcal{U}_{\text{BUFF}}}$, $\mathcal{U}_{\text{BUFF}}$, $\{\sigma'_{u,\tau_u}, \langle (z_{u,\tau_u}, z'_{u,\tau_u}) \rangle\}_{u \in \mathcal{U}_{\text{BUFF}}}$ to $i \in \mathcal{K}$

Online - Round 2

Decryptor $i \in \mathcal{K}_2$:

Consistency check

1. Receive $\{\epsilon_{u,i}, \langle (z_{u,\tau_u}, z'_{u,\tau_u}) \rangle, \sigma'_{u,\tau_u}\}_{u \in \mathcal{U}_{\text{BUFF}}}$ and $\mathcal{U}_{\text{BUFF}}$
2. Assert that $|\mathcal{U}_{\text{BUFF}}| = n$; if not, abort
3. $\sigma''_i \leftarrow$ **Sig.Sign**$(d_i^{SK}, \mathcal{U}_{\text{BUFF}})$
4. $\forall u \in \mathcal{U}_{\text{BUFF}}$, $1 \leftarrow$ **Sig.Ver**$(d_u^{PK}, \langle (z_{u,\tau_u}, z'_{u,\tau_u}) \rangle, \sigma'_{u,\tau_u})$; if not, abort
5. Send σ''_i to Server

Server:

6. Assert that $|\mathcal{K}'_2| \geq t$; if not, abort
7. Collect $\{\sigma''_i\}_{i \in \mathcal{K}'_2}$ and broadcast to *decryptors* in $\mathcal{K}'_2$

Online - Round 3

Decryptor $i \in \mathcal{K}_3$:

Aggregation key share construction

1. Receive $\{\sigma''_j\}_{j \in \mathcal{K}'_2}$
2. $\forall j \in \mathcal{K}'_2, 1 \leftarrow$ **Sig.Ver**$(d_j^{PK}, \mathcal{U}_{\text{BUFF}}, \sigma''_j)$; if not, abort
3. $\forall u \in \mathcal{U}_{\text{BUFF}}, [sk_{u,\tau_u}]_i \leftarrow$ **AE.Dec**$(c_{i,u}, u \mid\mid i \mid\mid \epsilon_{u,i})$ // *Decrypt JL secret key*
4. $[sk_0]_i = \sum_{u \in \mathcal{U}_{\text{BUFF}}} [sk_{u,\tau_u}]_i$ // *Compute share of JL aggregation key*
5. $\langle (z_0, z'_0) \rangle = \prod_{u \in \mathcal{U}_{\text{BUFF}}} \langle (z_{u,\tau_u}, z'_{u,\tau_u}) \rangle$ // *Compute aggregated encrypted hash and witness masks*
6. $[(z_0, z'_0)]_i =$ **TEG.PartialDecrypt**$([sk]_i, \langle (z_0, z'_0) \rangle)$ // *Compute partial TEG decryption*
7. Send $[sk_0]_i$ and $[(z_0, z'_0)]_i$ to Server

Server:

Aggregation

8. Assert that $|\mathcal{K}'_3| \geq t$; if not, abort
9. Collect $\{[sk_0]_i, [z_0]_i\}_{i \in \mathcal{K}'_3}$
10. $sk_0 \leftarrow$ **SS.Recon**$(\{[sk_0]_i\}_{i \in \mathcal{K}'_3}, t)$ // *Reconstruct JL aggregation key*
11. $\vec{s}_0 \leftarrow$ **JL.Agg**$(pp^{\text{JL}}, sk_0, \tau_0, \{\langle \vec{s}_{u,\tau_u} \rangle\}_{u \in \mathcal{U}_{\text{BUFF}}})$ // *Compute LWE aggregation key*
12. $\vec{x} \leftarrow$ **LWE.Agg**$(pp^{\text{LWE}}, \vec{s}_0, \{\vec{y}_{u,\tau_u}\}_{u \in \mathcal{U}_{\text{BUFF}}})$ // *Compute aggregated input*
13. $(z_0, z'_0) \leftarrow$ **TEG.Decrypt**$(\{[(z_0, z'_0)]_i\}_{i \in \mathcal{K}'_3})$ // *Compute full TEG decryption*
14. $h_0 = \prod_{u \in \mathcal{U}_{\text{BUFF}}} \tilde{h}_{u,\tau_u} - z_0$ // *Unmask aggregated input hash*
15. $r_0 = \prod_{u \in \mathcal{U}_{\text{BUFF}}} \tilde{r}_{u,\tau_u} - z'_0$ // *Unmask aggregated witness*
16. Broadcast $\vec{x}$, $\{c_{u,\tau_u}, \sigma_{u,\tau_u}\}_{u \in \mathcal{U}_{\text{BUFF}}}$, h_0, r_0 to $u \in \mathcal{U}_{\text{BUFF}}$

Verification

Client $u \in \mathcal{U}_{\text{BUFF}}$:

1. Receive $\{c_{v,\tau_v}\}_{v \in \mathcal{U}_{\text{BUFF}}}$, $\vec{x}$, h_0 and r_0
2. $\forall v \in \mathcal{U}_{\text{BUFF}}, 1 \leftarrow$ **Sig.Ver**$(d_v^{PK}, c_{v,\tau_v}, \sigma_{v,\tau_v})$; if not, abort
3. $c_0 = \prod_{v \in \mathcal{U}_{\text{BUFF}}} c_{v,\tau_v}$ // *Compute aggregated commitment*
4. Check **COM.Commit**$(h_0, r_0) = c_0$; if not, abort
5. $h_{agg} = H(\vec{x})$ // *Compute global model hash*
6. Check if $h_{agg} = h_0$; if not, abort

Figure 5.3.: **Buffalo** steps – blue parts guarantee SA verification (**Buffalo+** steps)

contributions to the buffer. Hence, in our case, *decryptors* are not mandatory if all clients are online during the construction of the aggregation key.

(2) In addition to being the first SA solution for the asynchronous setting, **Buffalo** exhibits good performance results at the client thanks to the use of a lattice-based batched encryption scheme to encrypt model parameters, as proposed in [Bell, 2023]. The number of LWE ring coefficients usually is in $[2^{11}, 2^{12}]$ [Bell, 2023] and each of them needs to be protected and further aggregated. In order to avoid clients sharing as many partial keys as the number of ring coefficients, **Buffalo** introduces an additional layer of protection. Informally, each client's input is first encrypted using a LWE key and the LWE key is further encrypted using a JL key. This key is then secretly shared with all the *decryptors*, only. Thanks to this approach, **Buffalo** achieves better client computation by sharing a single scalar value instead of a vector value. Such a performance improvement is very important in the context of an asynchronous setting wherein saving resources becomes fundamental and taking into account inputs from slow clients pivotal.

Description

Buffalo, reported in Figure 5.3, is defined over two phases: the Setup phase during which clients first register to the server and generate their keying material, and the Online phase during which aggregation occurs asynchronously (i.e. whenever the buffer is full).

In the Setup phase, similar to other solutions [Bonawitz, 2017a; Ma, 2023], each client creates a pair of secret and public keys and a pair of signing and verification keys, and sends the public and verification keys to the Public Key Infrastructure (PKI) to register them. Furthermore, each pair of client and decryptor u and i establish a pairwise key $c_{u,i}$. A Trusted Dealer (TD)[1] generates the public parameters, in particular the JL modulus N and the public LWE matrix A, and register them to the PKI.

The Online phase, also reported in Figure 5.3, is split into three rounds:

(**Round 1**) Each client u in the set $\mathcal{U}$ generates a secret LWE key $\vec{s}_{u,\tau_u}$ and a secret JL key sk_{u,τ_u} at their local round τ_u. The JL key sk_{u,τ_u} protects the LWE key $\vec{s}_{u,\tau_u}$ using a fixed 'round' $\tau = \tau_0$. Each private input vector $\vec{x}_{u,\tau_u}$ is protected with the LWE key $\vec{s}_{u,\tau_u}$. The server collects the n first submitted protected inputs and encrypted secret LWE keys. The clients owning those elements are finally included in a buffer set $\mathcal{U}_{\text{BUFF}} \subseteq \mathcal{U}$. Each client u then secretly shares their key sk_{u,τ_u} such that t out of these n shares can reconstruct it

[1]Alternative methods exist in decentralized settings to avoid the participation of a TD [Chen, 2021].

using the SS scheme. They send the shares of sk_{u,τ_u}, encrypted using the AE scheme, to the server. The latter broadcasts those elements to all *decryptors* in the set $\mathcal{K}$.

(**Round 2**) The *decryptors* follow the same process as in [Bonawitz, 2017a; Ma, 2023] over the set $\mathcal{U}_{\text{BUFF}}$. The latter is signed by each decryptor and the resulting signature is forwarded to the server, which passes it on to all *decryptors* in $\mathcal{K}$. This step ensures consistency over the set $\mathcal{U}_{\text{BUFF}}$ across the entire system.

(**Round 3**) Each decryptor in $\mathcal{K}_2 \subseteq \mathcal{K}$ receives the encrypted shares of the secret JL keys of clients in $\mathcal{U}_{\text{BUFF}}$. It computes the aggregated value $[sk_0]_i$, that is its share of the server's JL aggregation key sk_0. The decryptor then forwards it to the server. The latter must receive at least t of these aggregated shares in order to successfully reconstruct sk_0. Once this key is retrieved, the server can have access to the LWE aggregation key $\vec{s}_0$ through the aggregation of the clients' encrypted LWE keys. The server finally recovers the aggregated model $\vec{x}$ using the key $\vec{s}_0$.

Security Analysis

We briefly analyse the security of **Buffalo**. The full, hybrid-based, proof is given in Appendix D.5. Let the set of corrupted clients be denoted as $\mathcal{C} \subset \mathcal{U}$, where $|\mathcal{C}| = \gamma$, and the set of corrupted *decryptors* be denoted as $\mathcal{C}_k \subset \mathcal{K}$, where $|\mathcal{C}_k| = \gamma_k$.

The security of the LWE and JL schemes guarantees that parties in $\mathcal{C}$ cannot distinguish the protected input $\vec{x}_{u,\tau_u}$ (protected with a LWE key) and the LWE key $\vec{s}_{u,\tau_u}$ (protected with a JL key) of an honest client u from random values.

The security of the SS scheme guarantees that *decryptors* in $\mathcal{C}_k$ cannot distinguish the protected JL key sk_{u,τ_u} of an honest client u from a random value. More precisely, the security of the SS scheme ensures that if at most $t-1$ *decryptors* in $\mathcal{C}_k$ have access to shares of $[sk_0]_u$ (i.e. each decryptor has at most one share and $|\mathcal{C}_k| < t$), then they cannot reconstruct the key.

Moreover, when the server is a *malicious* adversary, it can try to convince some honest *decryptors* that the set of clients in the buffer is $\mathcal{U}_{\text{BUFF}}$ while indicating to other honest *decryptors* that the set of clients in the buffer is $\mathcal{U}^*_{\text{BUFF}} = \mathcal{U}_{\text{BUFF}} \setminus \{u\}$ for a client u. If this occurs, the server can reconstruct sk_0 for $\mathcal{U}_{\text{BUFF}}$ and sk^*_0 for $\mathcal{U}^*_{\text{BUFF}}$. Then, it can compute $sk_{u,\tau_u} = sk_0 - sk^*_0$. This is prevented during **Round 2**, through a consistency check step over the set $\mathcal{U}_{\text{BUFF}}$ [Bonawitz, 2017a; Ma, 2023]. Since we assume that there are $k-t$ corrupted *decryptors*, the server can obtain $k-t$ shares of sk_0 and sk^*_0, respectively. Furthermore, the server has the ability to convince $\frac{t}{2}$ honest *decryptors* that the client u is in the buffer and the other $\frac{t}{2}$ honest *decryptors* that u is not, thereby collecting shares of

sk_0 and sk'_0 accordingly. Therefore, to ensure *Aggregator Obliviousness* regarding the JL scheme, we require that $k - t + \frac{t}{2} < t \implies t > \frac{2k}{3}$.

5.5.2 Buffalo+

We introduce **Buffalo+**, a verifiable extension of **Buffalo**. Specifically, **Buffalo+** is an asynchronous adaptation of the protocol detailed in [Buyukates, 2022]. We depict the parts specific to **Buffalo+** in Figure 5.3 in blue. Moreover, for the sake of space, operations denoted as executed on an input pair are actually operations executed twice, once for each input taken individually (e.g. line 6 in **Online - Round 3**: **TEG.PartialDecrypt** on the pair of encrypted hash and witness masks). The design principles are the following:

(1) **Buffalo+** ensures that every client $u \in \mathcal{U}_{\text{BUFF}}$ can verify the aggregation of the global model obtained by the server, and in particular, that their input $\vec{x}_{u,\tau_u}$ has been included in the process. Each client generates a homomorphic hash over the local update $\vec{x}_{u,\tau_u}$ and commits to it. Then, both the hash value and the commitment witness are masked with some randomness. Instead of secretly sharing those random masks as in [Buyukates, 2022], we choose to use the Threshold ElGamal (TEG) scheme (Appendix D.2.3) to encrypt them because TEG provides constant encryption time (see Section 5.6 for the complexity analysis). The resulting TEG ciphertexts are sent to the *decryptors* who are in charge of applying threshold decryption to obtain the masks. This design choice allows constant communication and computation costs compared to using the classical SS scheme as in [Buyukates, 2022]. Specifically, **Buffalo+** reduces the regular client computation by a factor of k^2 and communication by a factor of k.

(2) We leverage the incremental nature of the homomorphic hash function H_{INC} to reduce computational overhead. As demonstrated in [Buyukates, 2022], computing a homomorphic hash function over a locally updated model of dimensions d remains a significant bottleneck. While most steps in the protocol can be parallelized during the training of local models [Mansouri, 2022; So, 2022], the critical hashing step (line 3 in **Online - Round 1**) must await the end of the training of the local model $\vec{x}_{u,\tau_u}$ to be executed [Buyukates, 2022].

Drawing from extensive research in distributed optimization [Ström, 2015; Fei, 2021; Sun, 2020], which shows that not all local ML parameters (here, gradients) change in each client round, we exploit parameter sparsification and quantization to avoid the full computation of Eq. 5.1. Utilizing this assumption of sparsity, each client computes the input hash for a given round τ_u and stores the local model $\vec{x}_{u,\tau_u}$. In the next client round $\tau_u + 1$, the client assesses the changes between $\vec{x}_{u,\tau_u}$ and $\vec{x}_{u,\tau_u+1}$, and employs the incremental hashing property to compute partial hashes only for the changed parameters using Eq. 5.2. Let ρ denote the fraction of parameters that changed between two client

rounds τ_u and $\tau_u + 1$. The aforementioned method requires only $O(d)$ comparisons and $O(\lfloor \rho \cdot d \rfloor)$ EC exponentiations, thus proving to be more efficient than directly hashing the input $\vec{x}_{u,\tau_u+1}$, despite incurring the cost of storing the previous local model input $\vec{x}_{u,\tau_u}$ in practice.

Description

To provide verifiable aggregation, **Buffalo+** follows the previously described protocol **Buffalo** with additional steps highlighted in blue in Figure 5.3. Specifically, there is a third phase, called *Verification*, that allows clients to check that the server has correctly aggregated their inputs submitted to the buffer. Additionally, in the Setup phase, the TD[2] generates the TEG public key and the decryptors' secret key shares. It registers the former to the PKI and sends the latter to the decryptors.

We report the differences between **Buffalo** and **Buffalo+** in the Online phase as follows:

(**Round 1**) The clients generate random EC points r_{u,τ_u} and $(z_{u,\tau_u}, z'_{u,\tau_u})$. They compute the local model hash h_{u,τ_u} (using Eq. 5.2 except at their first own round $\tau_u = 1$), and commit to the latter using the witness r_{u,τ_u}. They also mask h_{u,τ_u} and r_{u,τ_u} using the values z_{u,τ_u} and z'_{u,τ_u}, respectively. The clients then protect the hash and witness masks $(z_{u,\tau_u}, z'_{u,\tau_u})$ using the TEG scheme, and sign the resulting encryptions along with the commitment c_{u,τ_u}. They send both masked elements, protected masks, signatures and commitments to the server.

(**Round 2**) During the consistency check, each decryptor further verifies that the signatures over the encrypted masks.

(**Round 3**) At least t remaining online *decryptors* in $\mathcal{K}_3 \subseteq \mathcal{K}$ help the server construct the aggregated masking pair (z_0, z'_0), which enables the construction of the aggregated hash h_0 and witness r_0. As in **Buffalo**, the JL aggregation key sk_0 is constructed, followed by the LWE aggregation key $\vec{s}_0$. The latter allows the server to recover the aggregated model $\vec{x}$. All these elements are forwarded to the clients for aggregation verification.

In the Verification phase, all the clients in $\mathcal{U}_{\text{BUFF}}$ can verify if the server has correctly aggregated their inputs. More precisely, each client receives and aggregates the commitments of all the clients in $\mathcal{U}_{\text{BUFF}}$, and verifies the aggregated commitment c_0 given the aggregated witness r_0 and the aggregated hash value h_0. Finally, the client computes the hash h_{agg} of the received global model and verifies if it is equal to h_0.

[2]A possible solution with a decentralized TEG setup is proposed in [Ma, 2023].

Security Analysis

As in **Buffalo**, we require that $t > \frac{2k}{3}$. Here, we briefly analyze the security regarding aggregation verifiability. The full, hybrid-based, proof is given in Appendix D.5.

The TEG scheme ensures that the server, along with clients in the set $\mathcal{C}$, cannot distinguish the protected masking pair $\langle(z_{u,\tau_u}, z'_{u,\tau_u})\rangle$ from random values. During the verification phase of **Buffalo+**, clients in the buffer set $\mathcal{U}_{\text{BUFF}}$ receive two to-be-verified elements from the server. The first element is the aggregate hash reconstruction h_0 of the other clients in the buffer, using the individual (signed) commitments c_0. If the server sends an incorrect aggregate hash h_0 at this step, clients in $\mathcal{U}_{\text{BUFF}}$ can detect the error during the aggregate commitment check step (line 4 in Verification), thanks to the computationally binding property of the underlying commitment scheme. The second element is the aggregate model $\vec{x}$. The clients will not accept an incorrect global model if the hash h_{agg} of this value does not match the constructed hash h_0 sent by the server. This is ensured by the collision resistance of the homomorphic hash function H we use. More details on the proof are given in Appendix D.5.

Further Extensions

Decentralized Setup- To enhance readability, we initially present the two protocols with the participation of a TD for the key generation of the JL and TEG schemes. It is worth to note that their Setup phase can be run among the decryptors in a decentralized manner, i.e. without the intervention of a TD. Informally, the common modulus N of the JL scheme can be generated using the decentralised solution proposed in [Chen, 2021] and the TEG Setup phase can follow the decentralized process presented in [Ma, 2023]. The use of these two building blocks are also a justification of this decentralized setup (at the current stage, we are unaware of a threshold LWE encryption scheme that does not require a trusted third party for the setup phase and it requires only 2 communication rounds for *decryptors* as TEG in [Ma, 2023] in the online phases of the protocol.

Dynamic Decryptors- To balance the computation costs over decryptors, it could be beneficial to offer dynamicity over the set $\mathcal{K}$, i.e. the set of decryptors changes after some amount of aggregation operations. Unfortunately, current **Buffalo** and **Buffalo+** require the exchange of pairwise keys between each client and decryptor pair. Consequently, each time the set $\mathcal{K}$ changes, new pairwise keys need to be established between each new decryptor and other clients.

Another way of enabling dynamic decryptors would be to use the additively homomorphic TEG scheme (based on elliptic curves) which allows all clients to encrypt with the same

Table 5.1.: Complexity analysis for one BAsyncFL round (n: buffer size; k: number of decryptors; m: number of **LWE** key coefficients, t: threshold value; d: input dimension; δ_k: fraction of dropped decryptors).

	AsyncDPSecAgg [Stevens, 2022]	**Buffalo**
Client Comp.	Client: $O(k^2 \frac{m}{\lfloor \delta_k k - t \rfloor} + d)$ Decryptor: $O(n \frac{m}{\lfloor \delta_k k - t \rfloor})$	Client: $\boldsymbol{O(k^2 + m + d)}$ Decryptor: $\boldsymbol{O(n)}$
Client Comm.	Client: $O(k \frac{m}{\lfloor \delta_k k - t \rfloor} + d)$ Decryptor: $O(n \frac{m}{\lfloor \delta_k k - t \rfloor})$	Client: $\boldsymbol{O(k + m + d)}$ Decryptor: $\boldsymbol{O(n)}$
Server Comp.	$O(k^2 \frac{m}{\lfloor \delta_k k - t \rfloor} + nm + nd)$	$\boldsymbol{O(k^2 + nm + nd)}$

public key, no matter who the decryptors are. Unfortunately, using such a TEG scheme to encrypt messages would erase the advantage of the use of the LWE scheme with effective batching techniques. Moreover, if the TEG scheme is combined with either LWE or JL schemes (i.e. used to encrypt either LWE or JL keys), then the decryption of these keys would consist of computing the discrete log of a point in a large field, which will negatively affect the overall performance of the system. Consequently, it is not straightforward to enable dynamic decryptors in **Buffalo** and **Buffalo+** while maintaining acceptable overheads.

Model Inconsistency- A recent study in [Pasquini, 2022] highlights an attack against SA in FL, called *model inconsistency*. Informally, the server can bypass SA steps by sending different models to different clients. This attack has been shown only in SyncFL contexts and can be overcome by embedding the training round in the encryption of the local models. Note that in a BAsyncFL setting, the verification steps of **Buffalo+** straightforwardly overcome such an attack.

5.6 Complexity Analysis

We analyze the complexity of Buffalo and compare it with Async-DPSecAgg [Stevens, 2022], we report the comparison results in Table 5.1. In Table D.2, we additionally provide the complexity analysis for AsyncLightSecAgg [So, 2022]. However, we refrain from direct comparison due to its higher complexity in practical applications, as explained in Appendix D.2.

• *Client Computation:* The encryption of the private input $\vec{x}_{u,\tau_u}$ in **Buffalo** is equivalent to the one in AsyncDPSecAgg [Stevens, 2022]. However, the encryption of the LWE secret key in AsyncDPSecAgg is calculated as $O(k^2 \frac{m}{\lfloor \delta_k k - t \rfloor})$, due to the packed variant of the SS scheme. In contrast, in **Buffalo**, the cost of protecting this key through the JL scheme is $O(m)$. Furthermore, the cost for sharing the JL key using the SS scheme is $O(k^2)$ since such a process deals with a scalar value. Indeed, our protocol does not secretly shares a vector but a scalar value, and hence, improves the overall performance, as empirically

shown in Section 5.7. Regarding decryptors, the cost only corresponds to summing $O(n)$ shares in **Buffalo**, whereas it reaches $O(n\frac{m}{\lfloor \delta_k k-t \rfloor})$ in AsyncDPSecAgg.

• *Client Communication:* In both **Buffalo** and AsyncDPSecAgg, each client u sends a protected input $\langle \vec{x}_{u,\tau_u} \rangle$ of dimension $O(d)$. However, there are notable differences between the two solutions beyond this point. In AsyncDPSecAgg, clients transmit $O(k\frac{m}{\lfloor \delta_k k-t \rfloor})$ secret shares, and each decryptor receives $O(n\frac{m}{\lfloor \delta_k k-t \rfloor})$ of them. In contrast, our protocol involves sending the LWE secret key, of size m, to the server. This is accompanied by the protected JL secret key. Each regular client thus ends up sending k shares of the JL secret key, while each decryptor receives n of these shares. In practice, our experimental results have demonstrated that the bandwidth required for communication in AsyncDPSecAgg is greater compared to **Buffalo** at decryptors, since the number of shares is proportional to the size m of the LWE secret key.

• *Server Computation:* At **Round 3**, the server constructs the JL aggregation key sk_0 from t shares, requiring a computation cost of $O(k^2)$, in contrast with $O(k^2\frac{m}{\lfloor \delta_k k-t \rfloor})$ for AsyncDPSecAgg due to the reconstruction of $\vec{s_0}$. Additionally, for both protocols, the server aggregates the protected LWE secret keys and the protected private inputs received from clients and unmasks the aggregated results, which requires computation costs of $O(n \cdot m)$ and $O(n \cdot d)$, respectively.

• *Server Communication:* The message exchanges in both protocols only occur between the server and clients. Hence, the server communication cost is equal to n times each client communication cost.

Buffalo+

We analyze the complexity of **Buffalo+** and compare it with LightVeriFL [Buyukates, 2022]. In **Buffalo+**, regular clients' extra computational cost involves calculating the homomorphic hash using H_{INC}, worth $O(\lfloor \rho \cdot d \rfloor)$. Additionally, the cost of encrypting the masks using the TEG scheme is $O(1)$. In contrast, in LightVeriFL [Buyukates, 2022], the computational costs include calculating the homomorphic hash (using Equation 5.1) and secretly sharing the masks which together are worth $O(k^2 + d)$.

5.7 Experimental Results

In this section, we experimentally evaluate the performance of **Buffalo** and **Buffalo+**. In order to conduct a comparative study, in addition to these two solutions, we have also implemented AsyncDPSecAgg [Stevens, 2022] (which is more efficient than LightSecAgg

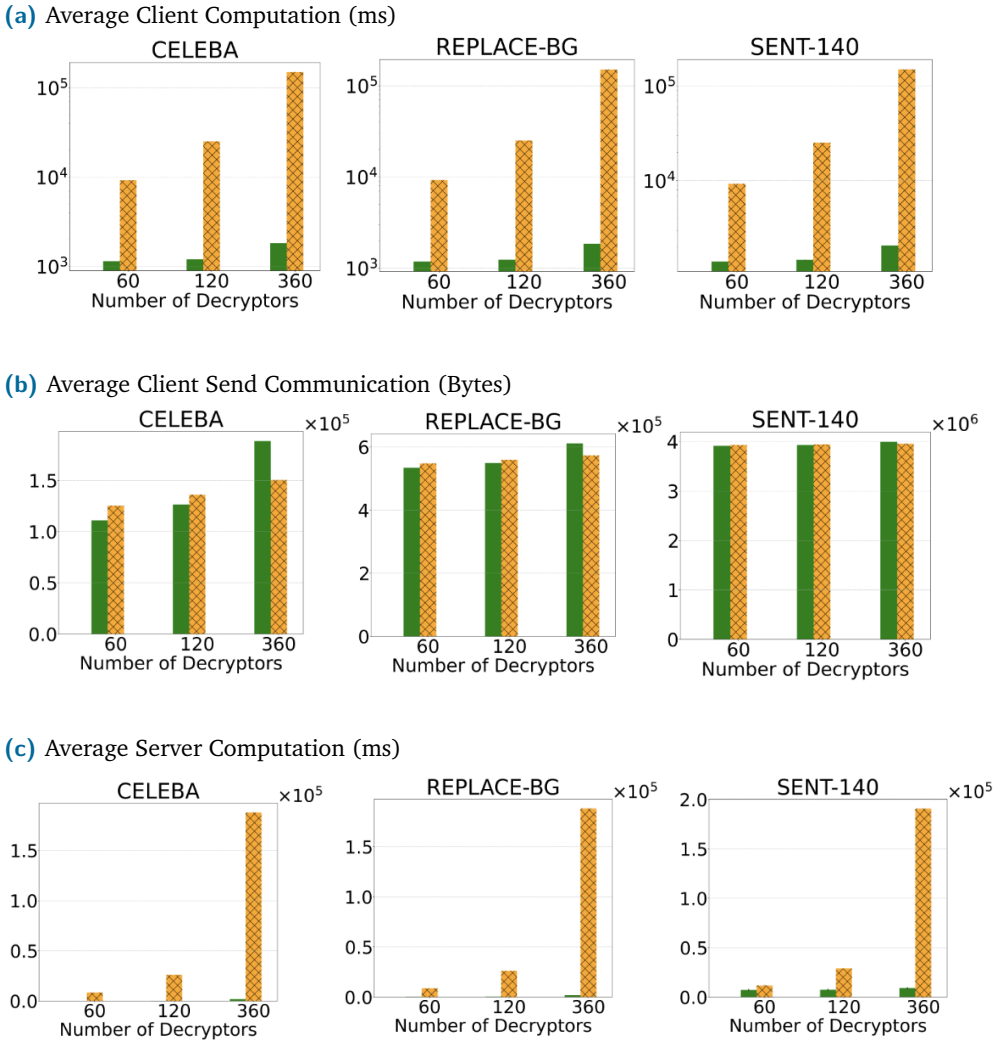


Figure 5.4.: Performance evaluation of **Buffalo** (green) and AsyncDPSecAgg (crossed-orange). Buffer size is fixed to $n = 512$ while varying the number k of decryptors.

[So, 2022; Ngong, 2023]) to compare with **Buffalo**, and **Buffalo** with LightVeriFL [Buyukates, 2022] to compare with **Buffalo+**.

Experimental Setting

Our implementations use Python with the **Olympia** framework [Ngong, 2023] and Pybind11 to wrap the C++ LWE SHELL library[3]. The code can be found on our anonymous GitHub page[4]. Experiments were conducted on a single-threaded processor, using a machine equipped with an Intel(R) Core(TM) i7-7800X CPU @ 3.50GHz and 126 GB of RAM. For the sake of a fair comparison, **Buffalo**, **Buffalo+**, AsyncDPSecAgg and

[3] https://github.com/google/shell-encryption/tree/master
[4] AnonymousGitHub code

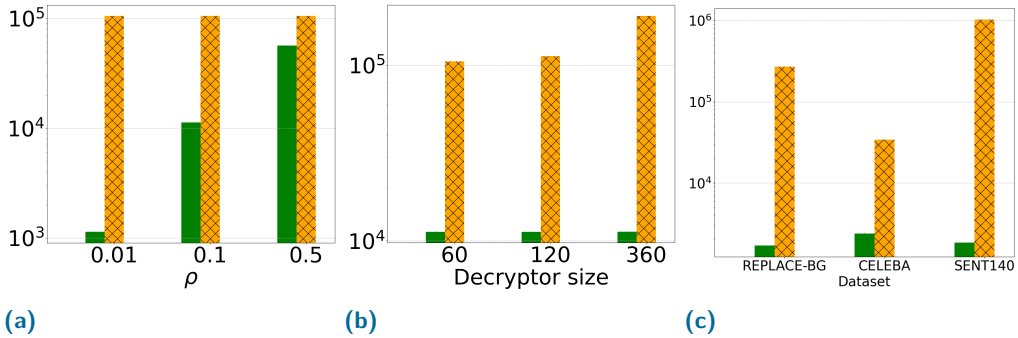


Figure 5.5.: Performance evaluation of **Buffalo+** (green) and LightVeriFL (crossed-orange). Input dimension is fixed to $d = 10^5$ and buffer size to $n = 2^6$. In (a), the number of decryptors is equal to $k = 60$. In (b), the fraction of parameters that changed between two client rounds is equal to $\rho = 0.01$.

LightVeriFL are implemented using the same building blocks and libraries mentioned in [Ngong, 2023; Buyukates, 2022]. We only evaluate the online phase of our protocols **Buffalo** and **Buffalo+**.

To have an idea of the cost of implementing a decentralized setup for **Buffalo**, we refer the reader to Table 3 in [Chen, 2021]. For instance, given 100 decryptors, the time needed is less than 8.3 minutes. Similarly for **Buffalo+**, an implementation of a decentralized TEG setup is evaluated in [Ma, 2023]) (Section 8). The verification step implementation is the same as the one evaluated in [Buyukates, 2022].

FL Parameters- To accurately evaluate the performance of the four schemes, we consider several scenarios that simulate realistic environments with the following varying parameters: buffer size n in {64, 128, 256, 512}; model dimension d in $\{30K, 260K, 1.2M\}$; number k of decryptors in {60, 120, 360} with threshold t set as $\frac{2k}{3}$. The performance of the four solutions is evaluated by measuring the execution time (i.e. computation cost) and the bandwidth (i.e. communication cost) at both the client and server sides. The values shown for each scenario are the result of the average of measurements from five independent executions.

SA Parameters- Following [Bell, 2023], the LWE error distribution χ_e with a discrete Gaussian distribution has a standard deviation equal to 3.2. The ring degree is set to $m = 2^{11}$ and the prime modulus to $q = 1 \mod 2m$, with $\lambda > 128$ bits of security according to [Albrecht, 2015]. The size of the JL modulus N is set to $\log_2(N) = 2048$. For both LWE and JL schemes, we apply packing techniques from [Bell, 2023; Mansouri, 2022] to pack multiple plaintexts in one single ciphertext.

BASyncLF Environment- We emulate realistic BASyncFL scenarios using the FLSim framework[5] with Pytorch. The FLSim framework allows us to emulate asynchronicity. We

[5] https://github.com/facebookresearch/FLSim/tree/main

use the same staleness distribution (i.e., the delay in clients' update that can occur in asynchronous systems) as FedBuff [Nguyen, 2022], namely a half-normal distribution with standard deviation equal to 1.25.

Test Datasets- We consider three FL use cases with two datasets from the LEAF benchmark [Caldas, 2018], namely CELEBA and SENT140, and one new medical dataset called REPLACE-BG [Aleppo, 2017].

CELEBA- The CELEBA is a binary image classification dataset that contains celebrity pictures. We use the Convolutional Neural Network (CNN) proposed in [Xie, 2019].

REPLACE-BG- The REPLACE-BG dataset was obtained from a cohort of 202 adult participants. It consists of three primary features:

• interstitial glucose levels measured in milligrams per deciliter (mg/dL) using Dexcom G4 Platinum sensors;

• insulin boluses administered in units (U);

• carbohydrate (CHO) content measured in milligrams (mg).

We have implemented the pre-processing steps for this dataset following the procedure described in [Jaloli, 2023]. Namely, the data were prepared as inputs to a CNN-Long Short-Term Memory (CNN-LSTM) architecture, a model commonly used for sequential data prediction tasks. The CNN-LSTM network is trained to predict blood glucose levels for the subsequent hour based on data from the last three hours, including glucose levels, insulin boluses, and CHO content [Cui, 2021].

SENT140- The SENT140 datasets is a text classification dataset for binary sentiment analysis. We use an LSTM model with 1.2M parameters.

Performance

We first evaluate the performance of **Buffalo** and compare it with AsyncDPSecAgg [Stevens, 2022]. We consider the three aforementioned use cases and fix the buffer size to $n = 512$. We first measure the computation and communication costs at the client with respect to the number k of decryptors. Figure 5.4 shows the experimental results. We observe that, thanks to the use of the JL scheme for the aggregation of the LWE key, **Buffalo** outperforms AsyncDPSecAgg in terms of computation both at the client and server sides. We recall that the LWE key is secretly shared with decryptors in AsyncDPSecAgg while

this is the JL key in **Buffalo**. The communication cost remains similar for both solutions since this cost mainly depends on the transmission of protected inputs of dimension d.

We then evaluate the performance of **Buffalo+** and compare it with LightVeriFL [Buyukates, 2022]. In Figure 5.5, we show the performance of the use of homomorphic hashng techniques in both **Buffalo+** and LightVeriFL, in terms of execution time in milliseconds (ms). We set the input dimension to $d = 10^5$ and the buffer size to $n = 256$. In Figure 5.5a, the number of decryptors is equal to to $k = 60$ while the input sparsity parameter ρ for the model varies. We observe the benefits of **Buffalo+** over LightVeriFL. Notably, when $\rho = 0.5$, the computation is almost $\times 2$ faster for the former. Furthermore, in Figure 5.5b, we fix $\rho = 0.01$ and vary the number of decryptors. This experimental result demonstrates the previous theoretical analysis discussed in Section 5.6 on the advantage of constant encryption time using the TEG scheme instead of the SS method as in LightVeriFL, which implies a quadratic result w.r.t. k.

5.7.1 Overall performance of BAsyncFL schemes

For each FL task (REPLACE-BG, CELEBA and SENT140), we consider Client Updates (CU), determined by the product of the buffer size and the total number of rounds necessary to achieve the Target Metric (TM), as reported in Tables 5.2a and 5.2b. We set the number of decryptors at $k = 60$, with a dropout rate of $\delta_k = 0.01$. Further implementation details are provided in Appendix D.4.

In Table 5.2a, we depict the total execution time in hours (h) and the bandwidth consumption in GigaBytes (GB) of **Buffalo**, **Buffalo+** and AsyncDPSecAgg, and compare them also with the case where aggregation is performed in cleartext. We observe that in terms of computation and communication costs, **Buffalo** always outperforms AsyncDPSecAgg by at least a factor of x3.

In Table 5.2b, we evaluate the the two verifiable SA protocols, namely **Buffalo+** and LightVeriFL (using **Buffalo** for SA). Compared to the previous table, we also include the per-buffer average number of updated model parameters, which corresponds to $\lfloor \rho \cdot d \rfloor$ and the input storage cost in MegaBytes (MB). **Buffalo+** consistently shows savings, especially when applied to the SENT140 task and its underlying ML model. This is primarily due to the underlying embedding layers: embedding layers are designed to process high-dimensional, typically sparse, data [Fei, 2021]. Finally, considering model sparsity for computational savings also requires storing the private inputs (in MB) from the previous client round. We therefore include this information as well in the fourth column.

(a) Comparisons for BAsyncFL SA schemes.

Protocol	Time (h)	Bandwidth (GB)
REPLACE-BG ($d = 260K$; CU=$24K$; TM: RMSE $11.5\% \downarrow$; $n = 16$; $n_{tot} = 180$)		
Clear	28.41	6.42
AsyncDPSecAgg	100.46	16.37
Buffalo	31.23	14.52
Buffalo+	73.43	14.63
CELEBA ($d = 31K$; CU=$48K$; TM: Accuracy $90.0\% \uparrow$; $n = 128$; $n_{tot} = 2336$)		
Clear	6.75	1.51
AsyncDPSecAgg	142.10	13.20
Buffalo	12.54	9.83
Buffalo+	54.61	10.04
SENT140 ($d = 1.2M$; CU=$99K$; TM: Accuracy $80.0\% \uparrow$, $n = 256$; $n_{tot} = 3482$)		
Clear	6.55	92.26
AsyncDPSecAgg	$> 7d.$	395.93
Buffalo	20.26	389.61
Buffalo+	77.98	390.01

(b) Comparisons for BAsyncFL VeriSA schemes.

Protocol	Time (h)	Bandwidth (GB)	Avg ($= \lfloor \rho \cdot d \rfloor$)	Storage (MB)
REPLACE-BG ($d = 260K$; CU=$24K$; TM: RMSE $11.5\% \downarrow$; $n = 16$; $n_{tot} = 180$)				
LightVeriFL(B)	$> 7d.$	14.73	$260K$	0
Buffalo+	73.43	14.63	1653	0.26
CELEBA ($d = 31K$; CU=$48K$; TM: Accuracy $90.0\% \uparrow$; $n = 128$; $n_{tot} = 2336$)				
LightVeriFL(B)	$> 7d.$	10.24	$31K$	0
Buffalo+	54.61	10.04	2310	0.03
SENT140 ($d = 1.2M$; CU=$99K$; TM: Accuracy $80.0\% \uparrow$, $n = 256$; $n_{tot} = 3482$)				
LightVeriFL(B)	$> 7d.$	390.61	$1M$	0
Buffalo+	77.98	390.01	1786	1.20

Table 5.2.: Comparison of execution time and bandwidth consumption, under various FL tasks. Client Updates (CU) product of buffer size n and rounds necessary to achieve the Target Metric (TM), we denote with $\uparrow$ when higher is better, and $\downarrow$ when lower

We show that, computation-wise, **Buffalo+** outperforms LightVeriFL, primarily due to the efficient use of homomorphic hash function (using Eq. 5.2 rather than Eq. 5.1). Moreover, **Buffalo+** exhibits better communication costs than LightVeriFL.

5.8 Conclusion

We introduced **Buffalo**, a SA protocol for BAsyncFL. To achieve asynchronicity, **Buffalo** uses lattice-based techniques for input protection and the participation of decryptors to help the server construct on-the-fly secret aggregation keys. **Buffalo+** extends **Buffalo** by adding verifiable SA to ensure aggregate integrity. Our evaluation, through theoretical analysis and real-world datasets, shows the efficiency and practicality of both protocols. Future work will extend our threat model to include malicious clients attempting to poison the global model, considering client input validation

Conclusion

6

Contents

6.1 Summary of the Main Contributions **93**
6.1.1 Privacy-Preserving Image Registration 93
6.1.2 Enhancing Privacy in Federated Learning: Secure Aggregation for Real-World Healthcare Applications 94
6.1.3 Let Them Drop: Scalable and Efficient Federated Learning Solutions Agnostic to Stragglers 94
6.1.4 Buffalo: A Practical Secure Aggregation Protocol for Asynchronous Federated Learning 95
6.2 Perspectives and Future Applications **95**
6.2.1 PPIR extensions 95
6.2.2 Privacy-Preserving Neural Networks for Medical Imaging Applications 97
6.2.3 Leveraging Sparsity in Distributed Learning with Secure Aggregation 97
6.2.4 Post-Quantum Secure Aggregation 98

This Chapter gives conclusive remarks and identifies future perspectives.

6.1 Summary of the Main Contributions

6.1.1 Privacy-Preserving Image Registration

In Chapter 2, we focus on the two-party setting in the dedicated medical image task. We have introduced a novel Privacy-Preserving Image Registration (PPIR) framework that addresses the need for confidentiality in medical imaging. We formulated the image registration problem under privacy-preserving conditions, recognizing the necessity to keep medical images confidential and not disclosed openly.

To address the data privacy problem, our approach incorporates advanced cryptographic tools, specifically secure multi-party computation (MPC) and fully homomorphic encryption (FHE), into traditional image registration paradigms. These tools allow operations

on encrypted data without revealing the underlying information. To tackle performance and scalability challenges, we introduced several optimization techniques, to meet Challenge 2 (See Chapter 1), including gradient approximations and homomorphic encryption through packing, enhancing the efficiency of cryptographic operations in high-dimensional spaces.

The PPIR framework supports a variety of registration methods, such as rigid, affine, and non-linear registration, providing a comprehensive solution adaptable to different medical imaging tasks. We demonstrated the effectiveness of the PPIR framework through extensive evaluation across various registration benchmarks, showcasing its ability to maintain the accuracy of traditional methods while ensuring privacy.

6.1.2 Enhancing Privacy in Federated Learning: Secure Aggregation for Real-World Healthcare Applications

In Chapter 3, we focus on the multi-party setting, and we implement secure aggregation (SA) schemes within the open-source Fed-BioMed framework. Despite the computational and communication bottlenecks that SA has in existing FL frameworks, we explored and implemented two SA protocols, Joye-Libert (JL) and Low Overhead Masking (LOM).

Our extensive benchmarks and evaluations on four healthcare datasets demonstrated that these SA protocols effectively protect privacy while maintaining task accuracy. The computational overhead introduced by SA is minimal, with training overheads being less than 1% on a CPU and less than 50% on a GPU for large models, and protection phases taking less than 10 seconds. Moreover, incorporating SA into Fed-BioMed resulted in a negligible impact on task accuracy, with deviations of no more than 2% compared to non-SA scenarios.

6.1.3 Let Them Drop: Scalable and Efficient Federated Learning Solutions Agnostic to Stragglers

Chapter 4 addresses the challenges posed by stragglers, in FL systems (Challenge 3 in Chapter 1) by proposing two new SA protocols, **Eagle** and **Owl**, which significantly improve upon existing solutions. In the context of synchronous FL (SyncFL), we introduced **Eagle**, a protocol that reduces computation and communication overheads for both FL clients and the server. **Eagle** effectively ignores dropped clients, including stragglers, and supports realistic dropout rates (10% to 30%). This improvement is achieved through a variant of the Threshold Joye-Libert scheme (TJL), resulting in enhanced performance compared to current methods. For asynchronous FL (AsyncFL) settings, we developed

Owl, a protocol that, like **Eagle**, does not require awareness of dropped clients and stragglers to complete the aggregation process. We conducted an extensive performance study, comparing **Eagle** and **Owl** with relevant state-of-the-art solutions both theoretically and experimentally.

6.1.4 Buffalo: A Practical Secure Aggregation Protocol for Asynchronous Federated Learning

In Chapter 5 we address the challenge in the realistic Buffered Asynchronous Federated Learning (BAsyncFL) setting. Our contributions include the introduction of two innovative SA protocols, **Buffalo** and **Buffalo+**, designed to improve scalability, security, and client participation. We introduced **Buffalo**, the first practical SA protocol tailored for BAsyncFL. **Buffalo** utilizes lattice-based encryption and homomorphic encryption on keys rather than models, addressing scalability issues related to the high dimensionality of machine learning models. A novel role, the *decryptor*, assists the server during the aggregation phase by helping to reconstruct keys generated on-the-fly by clients. This approach allows for asynchronous aggregation, avoiding synchronization issues and ensuring efficient and secure aggregation without requiring synchronized rounds. To further incentivize client participation and ensure that their local computations are considered in the aggregation process, we proposed **Buffalo+**, an asynchronous verifiable SA protocol. **Buffalo+** enables clients to verify that the server has correctly included their updates in the aggregation process, thereby ensuring trust and transparency in distributed training. We evaluated our protocols against existing solutions adapted to BAsyncFL through real asynchronous simulations using benchmark datasets and a novel medical dataset. Our evaluations, including both theoretical and experimental validations across three real datasets, demonstrate the practicality and effectiveness of our solutions, particularly in medical applications.

6.2 Perspectives and Future Applications

6.2.1 PPIR extensions

In Chapter 2, we consider classic gradient-based optimization to solve different cost functions, such as SSD and MI. In particular, we show how the communication overhead of PPIR compared to the classic image registration algorithm suffers with respect to the total number of gradient descent interactions needed, since for each iteration an exchange between the two parties has to be done. Some works have shown how to replace gradient-based optimization with neural networks [Andrychowicz, 2016] through

meta-learning [Hospedales, 2021]. In these works the parameters of one neural network (the optimizee) are learned using a different neural network (the optimizer), trained to solve the optimization task. Thus, there are two distinct neural networks: the optimizee performs a specific task such as regression or image classification, and the optimizer updates the weights of the optimizee. This approach has been shown to enhance the convergence speed of the cost function, decreasing the required number of optimization steps. More recently, a meta-learning approach has also been proposed in the context of image registration [Falta, 2022]. The paper introduces a novel recurrent framework that combines an iterative dynamic cost sampling step with a trainable optimizer designed to emulate Adam optimization, significantly reducing the number of iterations needed. We note that the underlying formulation proposed in this work does not directly decrease the PPIR computation cost, as their meta-learning optimizer uses the SSD displacement error over high-dimensional features [Heinrich, 2012]. This error, which must be computed in a privacy-preserving manner, is larger compared to the original input dimensions. Hence, finding a proper meta-learning optimizer, compatible with PPIR two-party setting optimization is a future challenge.

In the multi-party setting, a future perspective involves the process of registering multiple images to a common template [Ashburner, 2007], which comprises several critical steps to ensure accuracy and convergence. Initially, one of the images can be selected as the template, or an average of all the images can be used to create an initial estimate. Each image is subsequently registered to this template using an image registration algorithm. Once all images are aligned to the current template, a new template is computed by averaging the registered images. This process of registration and template updating is repeated iteratively until the difference between successive templates falls below a predefined threshold, indicating convergence. This approach can be studied and adapted to preserve privacy, for example, by enabling the use of SA, it becomes possible to protect individual images while revealing only the global template. Studying the impact of SA within the template creation and how it affects the final registration is a possible challenge.

An additional extension of PPIR might involve its impact on other domains, such as biometric verification algorithms. Biometric verification, such as facial recognition, verifies individual identities by comparing a live image to a stored template in a database. Classic biometric verification typically involves capturing a live image of the individual, extracting unique features (such as facial landmarks or fingerprint minutiae), and comparing these features to pre-registered templates to confirm a match. Existing privacy-preserving solutions protect the template data [Sandhya, 2017] during verification, enabling computation and comparison over encrypted data [Ibarrondo, 2023]. However, the pre-registration step of the database templates relies on a trusted party that uses centralized alignment algorithms, before storing the protected templates. Implementing PPIR within this context presents a future challenge, as it involves adapting

privacy-preserving techniques to improve the system's security and privacy, thereby removing the need for a trusted party to pre-register the template images.

6.2.2 Privacy-Preserving Neural Networks for Medical Imaging Applications

Neural networks (NN) are now commonly used in image registration [Balakrishnan, 2019] and other medical imaging tasks, such as image segmentation [Ronneberger, 2015]. Some medical imaging tasks have even been proposed with FL [Terrail, 2022], however, it does not guarantee privacy [Shokri, 2017] and incurs significant overhead, often failing to achieve good and reliable results in complex tasks due to data scarcity and heterogeneity of local datasets. Recently, Privacy-preserving NN with Multi-Party Computation (MPC) has gained significant attention. The concept involves jointly training or evaluating a neural network with several parties under different privacy-preserving scenarios: (i) one party may own both features and labels, increasing the total number of samples, (ii) some parties may own the features while others own the labels, and (iii) one party may own all the features while another party owns the neural network parameters. To reduce the introduced overhead, several software advancements have been made in recent years for privacy-preserving NN with MPC. For instance, to decrease computation overhead, two recent works, CryptGPU [Tan, 2021] and Piranha [Watson, 2022], have proposed methods to convert neural networks for MPC training and inference by exploiting GPU hardware acceleration. These methods perform linear operations and approximations entirely on GPUs, significantly decreasing computational overhead. Both studies demonstrated applicability in image classification tasks using datasets like CIFAR-10 and MNIST, and considered a broad range of neural networks, such as VGG and AlexNet. Applying privacy-preserving MPC with GPU frameworks can facilitate a training approach similar to a centralized one, potentially bridging the existing gap with FL and providing a future alternative. When dealing with these frameworks, it is essential to analyze different loss functions, such as DiceLoss. Studying and implementing the missing components of these MPC with GPU frameworks involves developing native solutions tailored for specific medical imaging tasks and addressing trade-offs for privacy protection and system scalability requirements.

6.2.3 Leveraging Sparsity in Distributed Learning with Secure Aggregation

In FL, each round involves a client sending its local model to the server. These transmitted models are usually large, causing the communication speed to become the bottleneck of the training process. This issue can be mitigated by either designing algorithms that

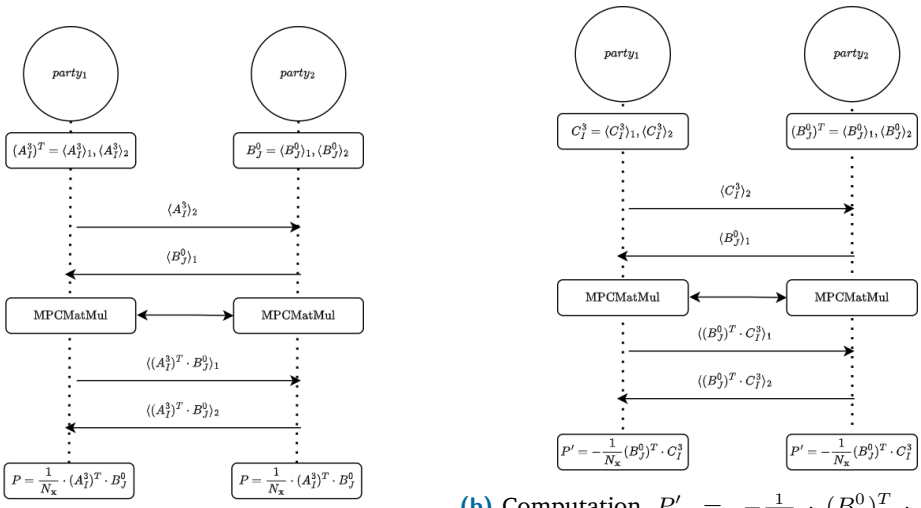
exchange fewer messages or reducing the size of each message. Sparsification[Konečnỳ, 2016] provides a way to reduce the size of exchanged models, it is a lossy compression technique typically used on parameters (gradients) in FL. By representing the gradients as a subset of their values selected using specific criteria, clients can send only the selected indices instead of the full model. In Chapter 5, we explored how to leverage the sparsity of a client's parameters (gradients) to exploit incremental hashing and improve the communication, within verifiable SA. However, only a few studies have investigated how this sparsity can be exploited in the context of distributed learning combined with SA [Beguier, 2020]. This solution requires a common indices agreement among all the clients, hence does not allow each client to select indices independently. Finally studying how SA quantization and convergence [Ström, 2015] can reduce communication and enable the application of existing sparsity techniques [Konečnỳ, 2016] in a privacy-preserving manner could significantly reduce SA communication overhead.

6.2.4 Post-Quantum Secure Aggregation

Peter Shor, recently named the winner of the IEEE Claude Shannon Award for 2025, is best known for his groundbreaking work on quantum computation. His Shor's algorithm, developed in 1994, demonstrated the potential of quantum computers to break conventional RSA codes by efficiently factoring large integers. If executed on a sufficiently powerful quantum computer, Shor's algorithm [Shor, 1999] could compromise public-key cryptographic schemes such as RSA, Finite Field Diffie-Hellman, and Elliptic Curve Diffie-Hellman [Rivest, 1978a; Diffie, 2022]. Post-quantum cryptographic algorithms, designed to be secure against quantum attacks, include lattice-based cryptography, hash-based cryptography, and multivariate polynomial cryptography. Lattice-based cryptography has shown promising features, such as its homomorphic properties, enabling computations on encrypted data without decryption. However, it faces significant challenges in practical deployment. One major issue is its computational efficiency. The algorithms often require high computational power [Nejatollahi, 2017] and memory usage, making them less practical for devices with limited resources, such as IoT devices or mobile phones. Another concern is the complexity of parameter selection and incorrect parameters can either weaken the security or make the system inefficient. Furthermore, the standardization of these parameters is still an ongoing process, and achieving consensus in the cryptographic community is challenging. Therefore, a possible future direction is to extend the SA protocols presented in Chapters 3, 4, and 5 with post-quantum privacy-preserving solutions, without compromising performances.

Appendix Chapter 2

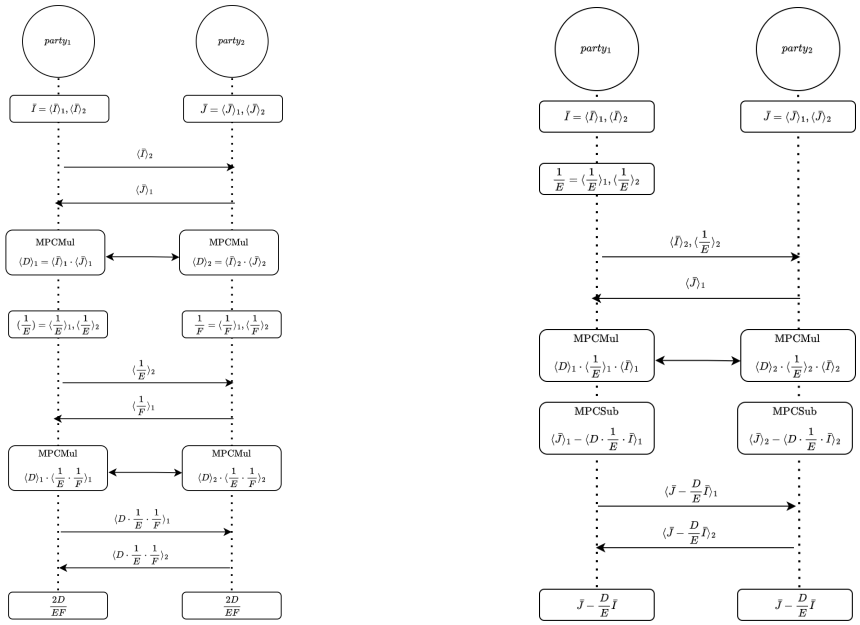
A



(a) Computation $P = \frac{1}{N_{\mathbf{x}}} \cdot (A_I^3)^T \cdot B_J^0$ with PPIR(MPC)

(b) Computation $P' = -\frac{1}{N_{\mathbf{x}}} \cdot (B_J^0)^T \cdot C_I^3$ with PPIR(MPC)

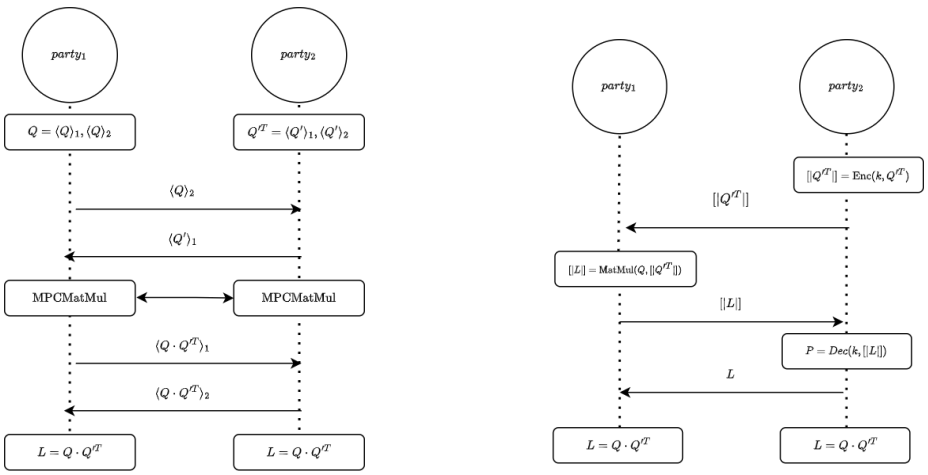
Figure A.1.: Optimization of MI loss: proposed framework to calculate matrix multiplication $P = \frac{1}{N_{\mathbf{x}}} \cdot (A_I^3)^T \cdot B_J^0$ and $P' = -\frac{1}{N_{\mathbf{x}}} \cdot (B_J^0)^T \cdot C_I^3$ based on PPIR(MPC).



(a) PPIR(MPC) - $\frac{2D}{EF}$

(b) PPIR(MPC) - $(\bar{J} - \frac{D}{E}\bar{I})$

Figure A.2.: Optimization of ANTS NCC loss: proposed framework to calculate $\frac{2D}{EF}$ and $(\bar{J} - \frac{D}{E}\bar{I})$ based on PPIR(MPC).



(a) PPIR(MPC) **(b)** PPIR(FHE)

Figure A.3.: Optimization of rigid point cloud: proposed framework to compute matrix multiplication $L = Q \cdot Q'^T$ based on PPIR(MPC) and PPIR(FHE).

Rigid Point Clouds Registration					
Solution	Displacement RMSE (mm)	Time $party_1$ (s)	Time $party_2$ (s)	Comm. $party_1$ (MB)	Comm. $party_2$ (MB)
PPIR(MPC)	1.11 ± 0.31	0.02	0.02	0.03	0.03
PPIR(FHE)	1.26 ± 0.37	1.10	0.18	1.42	35.93

Table A.1.: Rigid Point Clouds registration test, comparisson between PPIR(MPC) and PPIR(FHE). Registration metrics are reported as mean and standard deviation. Efficiency metrics in terms of average across iterations. RMSE: root mean square error.

A.1 Rigid Point Cloud Registration

Let $\{z_i\}$, $\{z'_i\}$ two finite n size point sets where $z_i, z'_i \in \mathbb{R}^d$, to continue ... A non-iterative least-squares approach to match two sets of points, was proposed by Arun et al. [Arun, 1987]. The method uses singular value decomposition (SVD) and is trying to minimize the following cost function:

$$\Sigma^2 = \sum_{i=1}^{n} ||z'_i - (Rz_i + \boldsymbol{t})||^2, \tag{A.1}$$

where R is the rotation matrix and $\boldsymbol{t}$ is the translation vector. Let define $\bar{z}$ and $\bar{z'}$ to represent the centroids of $\{z_i\}$ and $\{z'_i\}$ respectively. Let $\hat{R}$ being the estimated rotation matrix, and $\hat{\boldsymbol{t}}$ being the estimated translation. The method lies in the algorithm for finding $\hat{R}$ detailed below:

1. Calculate the following quantities:

$$q_i = p_i - \bar{z}$$
$$q'_i = p'_i - \bar{z'}$$

for all $0 \leq i \leq n$, which are distances from each point to its centroid.

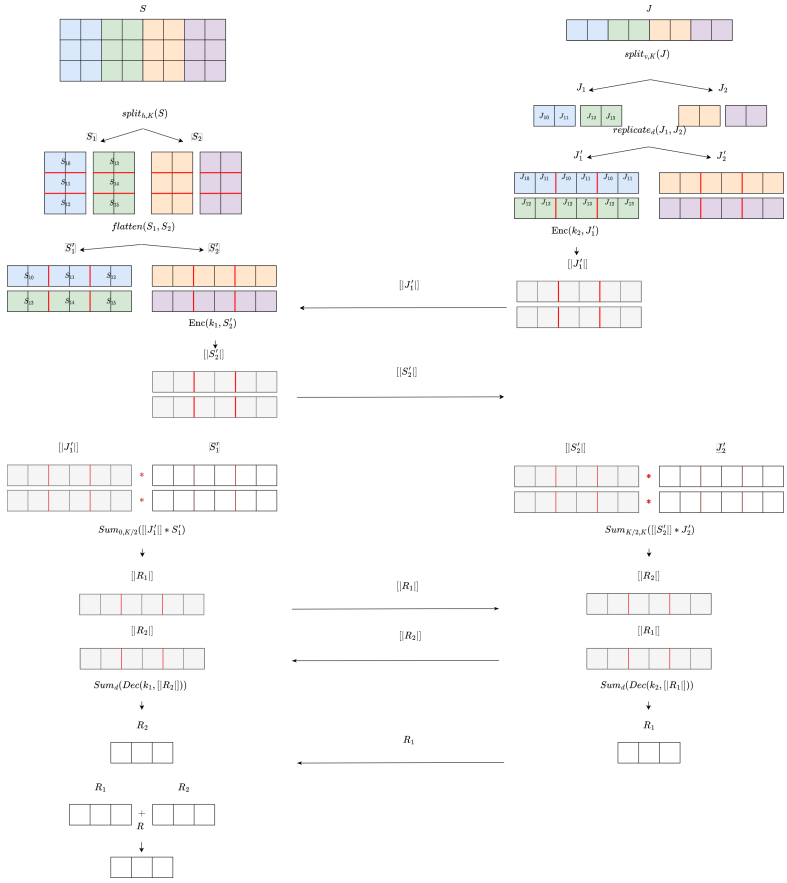


Figure A.4.: Proposed framework to compute matrix-vector multiplication $S^T \cdot J$ based on PPIR(FHE)-v2.

2. Calculate $L \in \mathbb{R}^{d \times d}$, $L = \sum_{i=1}^{n} q_i q_i'^T$, in vectorized form:

$$L = Q \cdot Q'^T, \tag{A.2}$$

   where $Q, Q' \in \mathbb{R}^{d \times n}$.

3. Find SVD of L, namely $L = U \Lambda V^T$;

4. Calculate $X = V U^T$;

5. Check the determinant of X. If it equals to $+1$, then $\hat{R} = X$ and $\hat{\boldsymbol{t}} = q' - \hat{R}q$. If the determinant equals to -1, the algorithm has failed.

We note that the only operation that requires the joint availability of information from both parties is Equation A.2 which can be computed with a matrix multiplication with PPIR(MPC) and PPIR(FHE) as reported in Figure A.7.

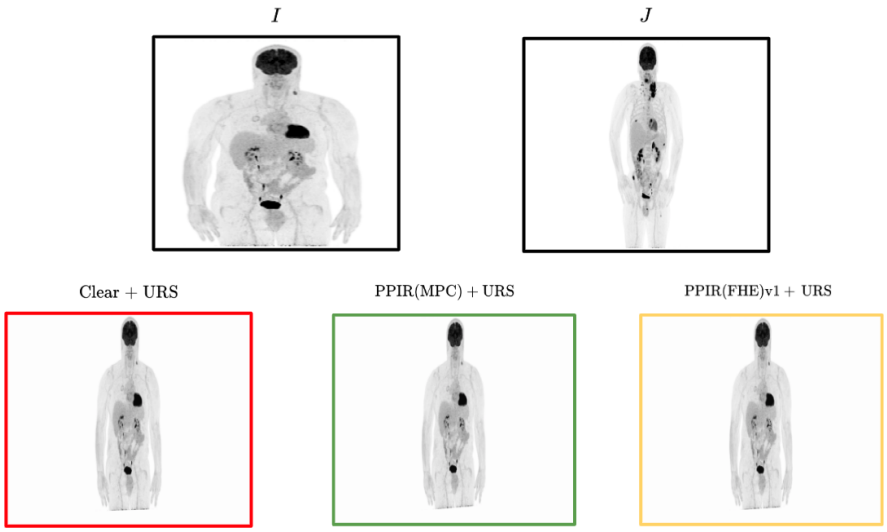


Figure A.5.: Qualitative results for affine registration with SSD between 2D medical images. The red frame is the transformed moving image using CLEAR+URS registration. Green and Yellow frames are the transformed images using respectively PPIR(MPC)+URS and PPIR(FHE)v1+URS.

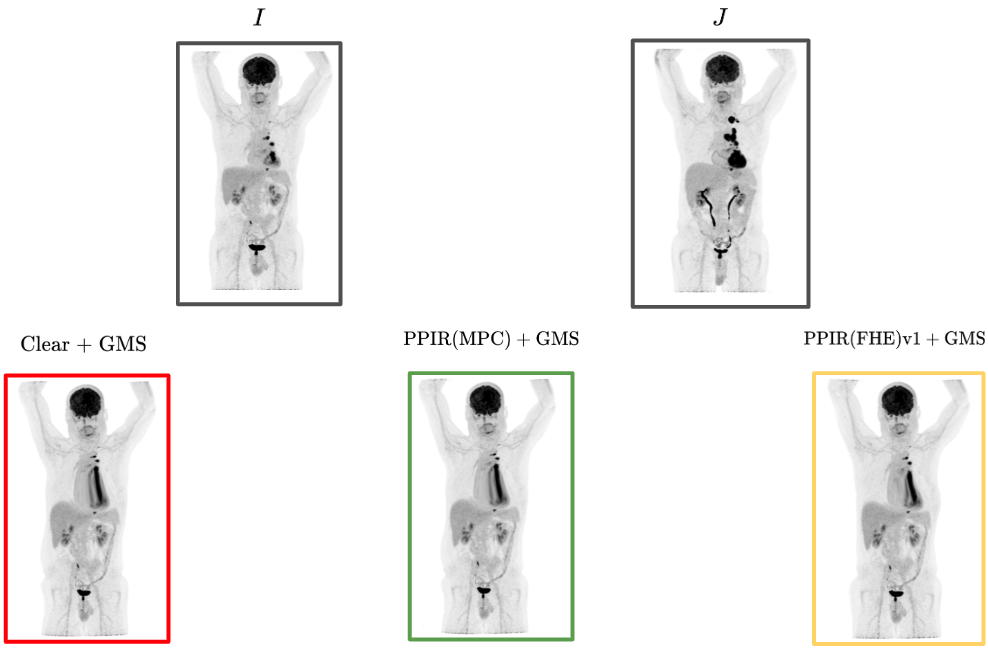


Figure A.6.: Qualitative results for Cubic splines registration with SSD between 2D medical images. The red frame is the transformed moving image using CLEAR+GMS registration. Green and Yellow frames are the transformed images using respectively PPIR(MPC)+GMS and PPIR(FHE)v1+GMS.

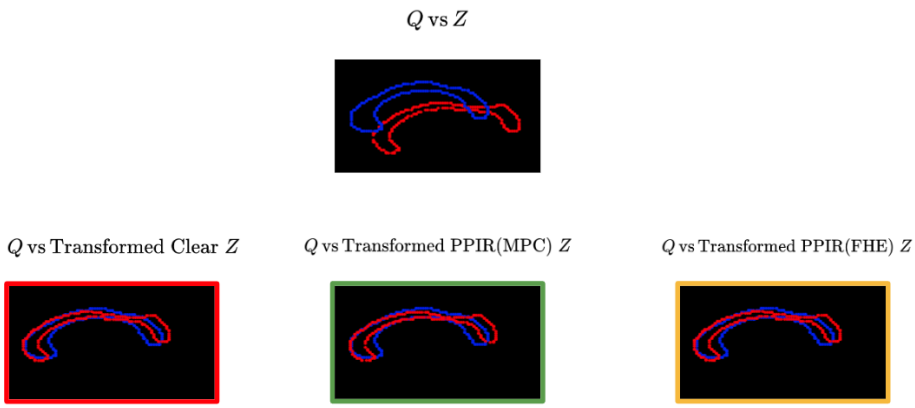


Figure A.7.: Qualitative results for rigid point cloud registration between 2D corpus callosum point sets. The red frame is the transformed moving image using CLEAR registration. Green and Yellow frames are the transformed images using respectively PPIR(MPC) and PPIR(FHE)v1.

Appendix Chapter 3

B

Dataset	FL Hyper-params						SA Hyper-params			Hardware spec.
	n_{tot}	n	T	e	b	η	L	W	max/min	
FedIIXI	3	3	75	10	2	1×10^{-3}	22	8	+20/-20	CPU
FedHeart	4	4	75	10	8	5×10^{-4}	15	17	+3/-3	CPU
REPLACE-BG	180	18	400	10	64	1×10^{-3}	13	15	+3/-3	CPU
FedProstate	4	4	75	6	8	1×10^{-3}	22	8	+2/-2	GPU

Table B.1.: FL hyper-params: number of total nodes n_{tot}, the number of selected nodes n, the number of FL rounds T, the number of local SGD steps e, the batch size b, and the learning rate η. SA hyper-params: number of bits input L, number of bits weight W and clipping range max/min.

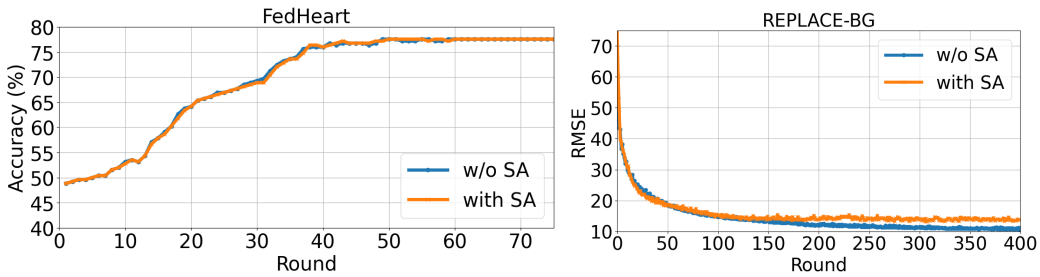


Figure B.1.: Compare the task accuracy of the global model at each FedAvg aggregation with and without applying SA for FedHeart and REPLACE-BG. The SA is characterized by L bits for representing input parameters, W bits for representing maximum dataset size, and the specified maximum and minimum clipping range.

Prerequisites Security paramter λ.
Parties: Server, nodes $\mathcal{U}$ and selected nodes $\mathcal{U}^{(\tau)}$, s.t $|\mathcal{U}| = n_{tot}$ and $|\mathcal{U}^{(\tau)}| = n$.

Public Parameters:

- $(\perp, pp^{JL}) \leftarrow$ **JL.Setup**(λ)

Setup - Key Setup:

Node u:

1. $sk_u \xleftarrow{R} \mathbb{Z}_{N^2}$.
2. $\{(v, [sk_u]_v)\}_{\forall v \in \mathcal{U}} \leftarrow$ **SS.Share**$(sk_u, t, \mathcal{U})$
3. Send $\forall v \in \mathcal{U} \setminus \{u\}, [sk_u]_v$
4. Receive $\{[sk_v]_u\}_{\forall v \in \mathcal{U} \setminus \{u\}}$
5. $[sk_0]_u \leftarrow \sum_{\forall v \in \mathcal{U}} [sk_v]_u$
6. Send $[sk_0]_u$ to Server

Server:

1. Collect $\{[sk_0]_u\}_{\forall u \in \mathcal{U}}$.
2. If $|\mathcal{U}| < t$, abort; otherwise, proceed.
3. $sk_0 \leftarrow$ **SS.Recon**$(\{[sk_0]_v\}_{\forall v \in \mathcal{U}}, t)$

Online - Protection (τ):

Node $u \in \mathcal{U}$:

1. $\vec{y}_{u,\tau} \leftarrow$ **JL.Protect**$(pp^{JL}, sk_u, \tau, \vec{x}_{u,\tau})$
2. Send $\vec{y}_{u,\tau}$ to *Server*.

Online - Aggregation (τ):

Server:

1. Collect $\{\vec{y}_{u,\tau}\}_{\forall u \in \mathcal{U}}$.
2. $\vec{x}_\tau \leftarrow$ **JL.Agg**$(pp, -sk_0, \tau, \{\vec{y}_{u,\tau}\}_{\forall u \in \mathcal{U}})$

(a) JL

Public Parameters:

- $(\perp, pp^{LOM}) \leftarrow$ **LOM.Setup**(λ)
- $(\perp, pp^{KA}) \leftarrow$ **KA.Param**(λ)

Setup - Key Setup:

Node $u \in \mathcal{U}$:

1. $\forall v \in \mathcal{U} \setminus \{u\}$,
   $s_{u,v} \leftarrow$ **KA.agree**$(pp^{KA}, c_u^{SK}, c_v^{PK})$

Online - Protection (τ, $\mathcal{U}^{(\tau)}$):

Node $u \in \mathcal{U}^{(\tau)}$:

1. $\vec{y}_{u,\tau} \leftarrow$
   LOM.Protect$(pp^{LOM}, \{s_{u,v}\}_{\forall v \in \mathcal{U}^{(\tau)} \setminus u}, \tau, \vec{x}_{u,\tau})$
2. Send $\vec{y}_{u,\tau}$ to *Server*.

Online - Aggregation (τ):

Server:

1. Collect $\{\vec{y}_{u,\tau}\}_{\forall u \in \mathcal{U}^{(\tau)}}$.
2. $\vec{x}_\tau \leftarrow$ **LOM.Agg**$(pp^{LOM}, \{\vec{y}_{u,\tau}\}_{\forall u \in \mathcal{U}^{(\tau)}})$

(b) LOM

Figure B.2.: SA protocols implemented in Fed-BioMed

Appendix Chapter 4

C

C.1 Hybrid Security Proofs

We present the security proofs of our two protocols using the hybrid argument technique. We do not consider the security proofs with an honest server since those proofs are relatively straightforward. We let the reader refer to [Bonawitz, 2017a] for more details.

We prove our protocols in the random oracle model for both adversarial models (i.e. honest-but-curious and active) since the security of the **JL** scheme and its **TJL** variant relies on the hash function being seen as a random oracle.

C.1.1 Honest-but-Curious Model

We first prove our protocol **Eagle** secure in the honest-but-curious model and then our protocol **Owl**.

Eagle

We prove the security of **Eagle** against honest-but-curious server and clients through a sequence of hybrids. We define a simulated execution, represented by a simulator $SIM_{\mathcal{C}}^{\mathcal{U},t,\lambda}(\vec{x}_{\tau,\mathcal{C}}, \mathcal{U}, \mathcal{U}_{on}^{(\tau)}, \mathcal{U}_{shares}^{(\tau)})$ (denoted as *SIM* for short), through several modifications to the real execution of the protocol, represented by $REAL_{\mathcal{C}}^{\mathcal{U},t,\lambda}(\vec{x}_{\tau,\mathcal{U}}, \mathcal{U}, \mathcal{U}_{on}^{(\tau)}, \mathcal{U}_{shares}^{(\tau)})$ (denoted as *REAL* for short), such that two subsequent hybrids are computationally indistinguishable.

***Hybrid*$_0$.** Let $\mathcal{C} \subseteq \mathcal{U}$ be any subset of adversarial parties (server and clients) such that there are strictly less than t adversarial clients. This hybrid is distributed exactly as *REAL*, that is the joint view of the parties in $\mathcal{C}$ in a real execution.

***Hybrid*$_1$.** In this hybrid, the behavior of the honest parties in $\mathcal{U} \setminus \mathcal{C}$ is changed. Instead of using the key $c_{u,v} \leftarrow$ **KA.agree**$(pp^{KA}, c_u^{SK}, c_v^{PK})$ to encrypt messages, a random key $c_{u,v}$ is uniformly chosen by *SIM*. The security of the Diffie-Hellman key agreement

KA under the Decisional Diffie-Hellman assumption ensures that this hybrid is indistinguishable from *Hybrid*$_0$.

Hybrid$_2$**.** In this hybrid, all ciphertexts encrypted by honest parties in $\mathcal{U} \setminus \mathcal{C}$ are changed. Instead of encryptions of shares $[sk_u]_v$, encryptions of 0 padded to the appropriate length are sent to other honest parties. Note that the honest clients in $\mathcal{U} \setminus \mathcal{C}$ still answer with the correct shares $[sk_u]_v$ in the *reconstruction step*. Only the contents of the ciphertexts have changed, hence the IND-CPA security of **AE** applies and guarantees that this hybrid is indistinguishable from *Hybrid*$_1$.

Hybrid$_3$**.** In this hybrid, all shares of sk_u generated by users in $\mathcal{U} \setminus \mathcal{C}$ are substituted with shares of 0 using a different sharing of 0 for every client in that set. The security of **TJL** has its secret sharing properties ensuring that the distribution of any $|\mathcal{C}|$ shares of 0 is identical to the distribution of $|\mathcal{C}|$ shares of sk_u. The server never receives sufficient shares to reconstruct sk_u since honest parties will not send their shares of sk_u. Therefore, this hybrid is indistinguishable from *Hybrid*$_2$.

Hybrid$_4$**.** In this hybrid, instead of using $\vec{y}_{u,\tau} \leftarrow$ **JL.Protect**$(pp, sk_{u,\tau}, \tau_0, \vec{x}_{u,\tau})$ for parties in $\mathcal{U} \setminus \mathcal{C}$, we use $\vec{y}_{u,\tau} \leftarrow$ **JL.Protect**$(pp, sk_{u,\tau}, \tau_0, \vec{w}_{u,\tau})$, where $\sum_{u \in \mathcal{U} \setminus \mathcal{C}} \vec{x}_{u,\tau} = \sum_{u \in \mathcal{U} \setminus \mathcal{C}} \vec{w}_{u,\tau}$ for uniformly random $\vec{w}_{u,\tau}$. The security of **JL** under the Decision Composite Residuosity assumption ensures that encryptions of private inputs $\vec{x}_{u,\tau}$ are indistinguishable from encryptions of $\vec{w}_{u,\tau}$. Hence, this hybrid is indistinguishable from *Hybrid*$_3$.

Hybrid$_5$**.** In this hybrid, for all parties in $\mathcal{U} \setminus \mathcal{C}$, instead of encryptions of the **JL** secret key $sk_{u,\tau}$, encryptions of 0 padded to the appropriate length are generated. The security of **TJL** under the Decision Composite Residuosity assumption ensures that this hybrid is indistinguishable from *Hybrid*$_4$.

Hybrid$_6$**.** Let us define $\mathcal{U}^* = \mathcal{U} \setminus \mathcal{C}$ when $|\mathcal{U}_{on}^{(\tau)}| < t$ and $\mathcal{U}^* = \mathcal{U} \setminus \mathcal{U}_{on}^{(\tau)} \setminus \mathcal{C}$ otherwise. In this hybrid, for all parties in $\mathcal{U}^*$, instead of computing $[\langle sk'_{0,\tau} \rangle]_u \leftarrow$ **TJL.ShareProtect**$(pp, \{[sk_v]_u\}_{v \in \mathcal{U}_{on}^{(\tau)}}, \tau)$, we set it to be a uniformly random variable of the appropriate size. Note that the key sk_u is chosen uniformly at random in the real execution and that in a previous hybrid, the shares $[sk_u]_v$ of sk_u given to the adversary were substituted with shares of 0. Hence, the only change in this hybrid is the substitution of the output of the algorithm **TJL.ShareProtect**. From the security of **TJL** under the Decision Composite Residuosity assumption, this hybrid is indistinguishable from *Hybrid*$_5$.

We finally define the PPT simulator *SIM* that samples from the distribution described in *Hybrid*$_6$. This proves that *SIM*'s output is computationally indistinguishable from *REAL*'s output.

Owl

We prove the security of **Owl** against honest-but-curious server and clients through a sequence of hybrids. We define a simulated execution, represented by a simulator $SIM_{\mathcal{C}}^{\mathcal{U},t,\lambda}(\vec{x}_{\mathcal{C}}, \mathcal{U}, \mathcal{U}_{on}, \mathcal{U}_{shares})$ (denoted as *SIM* for short), through several modifications to the real execution of the protocol, represented by $REAL_{\mathcal{C}}^{\mathcal{U},t,\lambda}(\vec{x}_{\mathcal{U}}, \mathcal{U}, \mathcal{U}_{on}, \mathcal{U}_{shares})$ (denoted as *REAL* for short), such that two subsequent hybrids are computationally indistinguishable.

***Hybrid*$_0$.** Let $\mathcal{C} \subseteq \mathcal{U}$ be any subset of adversarial parties (server and clients) such that there are strictly less than t adversarial clients. This hybrid is distributed exactly as *REAL*, that is the joint view of the parties in $\mathcal{C}$ in a real execution.

***Hybrid*$_1$.** In this hybrid, the behavior of the honest parties in $\mathcal{U} \setminus \mathcal{C}$ is changed. Instead of using the key $c_{u,v} \leftarrow$ **KA.agree**$(pp^{KA}, c_u^{SK}, c_v^{PK})$ to encrypt messages, a random key $c_{u,v}$ is uniformly randomly chosen by *SIM*. The security of the Diffie-Hellman key agreement **KA** under the Decisional Diffie-Hellman assumption ensures that this hybrid is indistinguishable from *Hybrid*$_0$.

***Hybrid*$_2$.** In this hybrid, all ciphertexts encrypted by honest parties in $\mathcal{U} \setminus \mathcal{C}$ are changed. Instead of encryptions of shares $[sk_{u,\tau_u}]_v$, encryptions of 0 padded to the appropriate length are sent to other honest parties. Note that the honest clients in $\mathcal{U} \setminus \mathcal{C}$ still answer with the correct shares $[sk_{u,\tau_u}]_v$ in the *reconstruction step*. Only the contents of the ciphertexts have changed, hence the IND-CPA security of **AE** applies and guarantees that this hybrid is indistinguishable from *Hybrid*$_1$.

***Hybrid*$_3$.** In this hybrid, all shares of sk_{u,τ_u} generated by users in $\mathcal{U} \setminus \mathcal{C}$ are substituted with shares of 0 using a different sharing of 0 for every client in that set. The properties of **SS** guarantees that the distribution of any $|\mathcal{C}|$ shares of 0 is identical to the distribution of $|\mathcal{C}|$ shares of sk_u. The server never receives sufficient shares to reconstruct sk_{u,τ_u} since honest parties will not send their shares of sk_{u,τ_u}. Therefore, this hybrid is indistinguishable from *Hybrid*$_2$.

***Hybrid*$_4$.** In this hybrid, instead of using $\vec{y}_{u,\tau_u} \leftarrow$ **JL.Protect**$(pp, sk_{u,\tau_u}, \tau_0, \vec{x}_{u,\tau_u})$ for parties in $\mathcal{U} \setminus \mathcal{C}$, we use $\vec{y}_{u,\tau_u} \leftarrow$ **JL.Protect**$(pp, sk_{u,\tau_u}, \tau_0, \vec{w}_{u,\tau_u})$, where $\sum_{u \in \mathcal{U} \setminus \mathcal{C}} \vec{x}_{u,\tau_u} = \sum_{u \in \mathcal{U} \setminus \mathcal{C}} \vec{w}_{u,\tau_u}$ for uniformly random $\vec{w}_{u,\tau_u}$. The security of **JL** under the Decision Composite Residuosity assumption ensures that encryptions of private inputs $\vec{x}_{u,\tau_u}$ are indistinguishable from encryptions of $\vec{w}_{u,\tau_u}$. Hence, this hybrid is indistinguishable from *Hybrid*$_3$.

We finally define the PPT simulator *SIM* that samples from the distribution described in $Hybrid_4$. This proves that *SIM*'s output is computationally indistinguishable from *REAL*'s output.

C.1.2 Active Model

Here, we consider active adversaries, that are clients or server deviating from the protocol by sending incorrect and/or arbitrarily chosen messages to honest users, aborting, omitting messages and sharing their protocol view with each other (including the server), as defined in [Bonawitz, 2017a].

We assume that authenticated encrypted channels exist to ensure that received messages come from clients and not the server. We thus prevent the server from launching Sybil attacks. We also ask the server to forward clients public keys in an honest way during the *registration step*, to assist clients to establish private and authenticated communication channels with each other.

Similarly to [Bonawitz, 2017a], we include a consistency check phase to prevent the server to give different information about which client is online during the online phase. This allows us to avoid the server to learn different sets of shares from different users, allowing the unauthorised reconstruction of secrets.

Finally, the security proof requires us to be in the random oracle model. Such a model helps the simulator reprogram the random oracle to make dummy information indistinguishable from values of honest clients.

We only present the security proof in the active model for **Eagle** since the differences between the latter and the protocol **Owl** are small. Informally, the security of the protocol **Owl** partly relies on the security of **JL** and of **SS**, rather than of **TJL** with **ISS**. The remaining for proving our AsyncFL protocol **Owl** secure follows the same pattern as for **Eagle** (i.e. encryption scheme, signature scheme).

Eagle - We prove the security of **Eagle** against active server and clients through a sequence of hybrids. Given n, t, λ and a subset $\mathcal{C}$ of adversarial parties, we define $M_{\mathcal{C}}$ as a probabilistic polynomial-time algorithm that denotes the "next-message" function of adversarial parties [Bonawitz, 2017a]. This function allows parties in $\mathcal{C}$ to dynamically select their inputs at any round of the protocol execution as well as the list of online users.

We define a simulated execution, represented by a simulator $SIM_{\mathcal{C}}^{\mathcal{U},t,\lambda,ID^{\xi}}(M_{\mathcal{C}})$ (denoted as *SIM* for short), through several modifications to the real execution of the protocol, rep-

resented by $REAL_{\mathcal{C}}^{\mathcal{U},t,\lambda}(M_{\mathcal{C}}, \vec{x}_{\mathcal{U}\setminus\mathcal{C}})$ (denoted as *REAL* for short), such that two subsequent hybrids are computationally indistinguishable. *REAL* exhibits the combined views of the adversarial parties in the protocol execution such that their messages and honest clients' aborts are chosen using $M_{\mathcal{C}}$.

$M_{\mathcal{C}}$ enables to dynamically set the subset of honest clients for which the server learns their local model aggregation. Therefore, this aggregation cannot be provided for a fixed subset of clients as input to *SIM*. Instead, *SIM* will make a single query to an ideal functionality *ID* (seen as a random oracle) that allows it to learn the aggregation for a dynamically chosen subset $\mathcal{L}$ of honest clients, such that $|\mathcal{L}| \geq \xi$ for a lower bound ξ of the number of honest clients. Note that all parties and $M_{\mathcal{C}}$ have access to the random oracle.

***Hybrid*$_0$.** Let $\mathcal{C} \subseteq \mathcal{U}$ be any subset of adversarial parties (server and clients) such that there are strictly less than t adversarial clients. This hybrid is distributed exactly as the joint view of $M_{\mathcal{C}}$ in *REAL*, that is the joint view of the parties in $\mathcal{C}$ in a real execution.

***Hybrid*$_1$.** In this hybrid, a simulator emulates the real execution. This simulator knows all the inputs $\vec{x}_{u,\tau}$ of the honest parties and runs a full execution of the protocol with $M_{\mathcal{C}}$, including a simulation of the random oracle "on the fly", the secure communication channel establishment and the setup phase. Thus, the adversarial view is the same as in *Hybrid*$_0$.

***Hybrid*$_2$.** In this hybrid, given any pair of honest users u and v, the messages between u and v are encrypted before being given to $M_{\mathcal{C}}$ and decrypted after being given to $M_{\mathcal{C}}$, using a uniformly random key rather than the one obtained from **KA.agree**$(pp^{KA}, c_u^{SK}, c_v^{PK})$. The security of the Diffie-Hellman key agreement **KA** under the Decisional Diffie-Hellman assumption ensures that this hybrid is indistinguishable from *Hybrid*$_1$.

***Hybrid*$_3$.** In this hybrid, *SIM* aborts if $M_{\mathcal{C}}$ manages to deliver a message to an honest client u on behalf of another honest client v during the key setup phase, such that the message is different from the message that *SIM* has given to $M_{\mathcal{C}}$ in that phase and that the decryption of this message does not fail (using the proper key). Note that the encryption key that u and v used in *Hybrid*$_1$ was randomly chosen. Hence, based on such a message, the integrity of the ciphertext could be threatened. Since the underlying encryption scheme is INT-CTXT secure, then this hybrid is indistinguishable from the previous one.

***Hybrid*$_4$.** In this hybrid, the simulator changes all encrypted shares sent between pairs of honest users with encryptions of 0. Note that *SIM* still returns the "real" shares in the *reconstruction step* as it did before. Since the encryption keys were chosen uniformly at random, the IND-CPA security of the underlying encryption scheme guarantees that *Hybrid*$_4$ is indistinguishable from *Hybrid*$_3$.

***Hybrid*$_5$.** In this hybrid, *SIM* aborts if $M_{\mathcal{C}}$ gives a signature on a set which correctly verifies based on an honest client's public key during the *consistency check step*, but the honest client has never created the signature on that set. The security of the underlying signature scheme guarantees that forgeries happen with negligible probability. Hence, *Hybrid*$_5$ is indistinguishable from *Hybrid*$_4$.

***Hybrid*$_6$.** Let $\mathcal{Q} \subseteq \mathcal{U}$ be the single set where an honest party received $\mathcal{Q}$ during the *consistency check step* and then received at least t valid signatures on it during the *reconstruction step*.

SIM aborts if $M_{\mathcal{C}}$ asks for key shares for some honest client u either before the adversary has received the responses from the honest parties in the *reconstruction step*, or after such responses have been received for $u \notin \mathcal{Q}$. In both cases, the key sk_u is information theoretically hidden from $M_{\mathcal{C}}$, and the simulator aborts if $M_{\mathcal{C}}$ can guess one of those sk_u, happening with negligible probability. Indeed, the values sk_u are chosen from the large domain $\mathbb{Z}_{N_0^2}$. Thus, the adversarial view is the same as in *Hybrid*$_5$.

***Hybrid*$_7$.** In this hybrid, the simulator aborts if $M_{\mathcal{C}}$ asks for encryptions of the per-round **JL** secret key for some honest client u either before the adversary has received the responses from the honest parties in the *reconstruction step*, or after such responses have been received for $u \notin \mathcal{Q}$. In both cases, the key $sk_{u,\tau}$ is information theoretically hidden from $M_{\mathcal{C}}$, and the simulator aborts if $M_{\mathcal{C}}$ can guess one of those $sk_{u,\tau}$, happening with negligible probability. Indeed, the values $sk_{u,\tau}$ are chosen from the large domain $\mathbb{Z}_{N_1^2}$. Thus, the adversarial view is the same as in *Hybrid*$_6$.

***Hybrid*$_8$.** In this hybrid, the values $\vec{y}_{u,\tau}$ computed by *SIM* on behalf of honest users and sent to $M_{\mathcal{C}}$ during the protection phase are changed with uniformly sampled values such that those values are independent from the rest of the view. Hence, this hybrid is indistinguishable from the previous one.

***Hybrid*$_9$.** For all $u \in \mathcal{Q} \setminus \mathcal{C}$, we choose values $\vec{w}_{u,\tau}$ such that $\sum_{u \in \mathcal{Q} \setminus \mathcal{C}} \vec{w}_{u,\tau} = \sum_{u \in \mathcal{Q} \setminus \mathcal{C}} \vec{x}_{u,\tau}$. Since $sk_{u,\tau}$ is never queried for $u \in \mathcal{Q} \setminus \mathcal{C}$ by $M_{\mathcal{C}}$, then in the view of $M_{\mathcal{C}}$, the above values are identically distributed as in the previous hybrid.

Hybrid$_{10}$**.** In this hybrid, *SIM* does not receive the inputs of honest parties. Instead, during the *reconstruction step*, the simulator submits a query to *ID* for the set $\mathcal{Q} \setminus \mathcal{C}$ and uses the output to sample the required elements $\vec{w}_{u,\tau}$. By construction, we have $|\mathcal{Q}| \geq t$ and $|\mathcal{Q} \setminus \mathcal{C}| \geq t - n_{\mathcal{C}}$ where $n_{\mathcal{C}} = \mathcal{U} \cap \mathcal{C}$. Hence, *ID* will not abort. This change does not modify the view of the adversary, making *Hybrid*$_9$ and *Hybrid*$_{10}$ indistinguishable. Moreover, this hybrid does not make use of the inputs of honest parties, concluding the proof.

Appendix Chapter 5

D

D.1 Notations

Symbol	Description (Chapter 4)	Description (Chapter 5)
n_{tot}	total number of clients	total number of clients
n	number of selected clients / buffer size	buffer size
δ	fraction of dropped clients	fraction of dropped decryptors
t	number of honest online clients	number of online honest decryptors
d	size of input	size of input
Δ	value set as equal to $n!$	-
$\mathcal{U}$	set of clients	set of clients s.t. $\lvert\mathcal{U}\rvert = n_{tot}$
$\mathcal{U}^{(\tau)}$	set of selected clients	set of clients in the buffer s.t. $\lvert\mathcal{U}_{\text{BUFF}}\rvert = n$
$\mathcal{U}_{on}$	set of online clients	-
$\mathcal{U}_{shares}$	set of honest online clients	set of decryptors s.t. $\lvert\mathcal{K}\rvert = k$
x_u	client's (scalar) input	scalar input of client u
$\vec{x}_u$	client's (vector) input	vector input of client u
y_u	protected client's (scalar) input	protected scalar input of client u
$\vec{y}_u$	protected client's (vector) input	protected vector input of client u
y'	protected (scalar) zero value	-
$\vec{y}'$	protected (vector) of zero values	-
x	aggregate (scalar)	scalar aggregate
$\vec{x}$	aggregate (vector)	vector aggregate
τ	current FL round	-
τ_0	value set as equal to 0	-
τ_u	current FL round of client u	current FL round of client u
$[s]$	share produced by secret sharing scheme	share of a secret s
N	modulus	JL modulus
R	plaintext size	-
m	-	size of LWE secret key
p, q	-	prime numbers
λ	security parameter	security parameter
$\xleftarrow{R}$	chosen uniformly at random	chosen uniformly at random

Table D.1.: Notations

D.2 Cryptographic Building Blocks

D.2.1 Elliptic Curves

An Elliptic Curve (EC) $E(\mathbb{F}_p)$, over the field $\mathbb{F}_p$ with prime number p, consists of points $P = (x, y)$, where $x, y \in \mathbb{F}_p$ satisfy $y^2 = x^3 + ax + b$, together with the point at infinity O [Lopez, 2000]. Let $p > 3$ be an odd prime, and $a, b \in \mathbb{F}_p$ satisfy $4a^3 + 27b^2 \neq 0$. Two

operations are defined on EC, namely point addition and scalar multiplication. Given an integer k, the scalar multiplication kP corresponds to adding the point P to itself k times and is analogous to the exponentiation operation in multiplicative groups.

D.2.2 Shamir Secret Sharing

A t-out-of-n Shamir Secret Sharing scheme [Shamir, 1979], denoted as SS, defined in a field $\mathbb{F}_p$, where p is a prime number, consists of two PPT algorithms:

- $\{(u, [s]_u)\}_{u \in \mathcal{U}} \leftarrow$**SS.Share**$(s, t, \mathcal{U})$: This algorithm splits a secret $s \in \mathbb{F}_p$ into n shares $[s]_u \in \mathbb{F}_p$, each of them for one client $u \in \mathcal{U}$. Note that each u is an element of $\mathbb{F}_p$, representing each client uniquely. Let t be the reconstruction threshold and n be the number of the clients in the set $\mathcal{U}$. The algorithm first generates a polynomial $p(x)$ of uniformly random coefficients and degree $t-1$ such as $p(0) = s$. It then computes $p(u) = [s]_u$ for all $u \in \mathcal{U}$.

- $s \leftarrow$**SS.Recon**$(\{(u, [s]_u)\}_{u \in \mathcal{U}'}, t)$: This algorithm reconstructs the secret $s \in \mathbb{F}_p$ using at least t shares. It is required that $\mathcal{U}' \subseteq \mathcal{U}$ and $|\mathcal{U}'| \geq t$. The algorithm uses the Lagrange interpolation to compute the value $p(0) = s$ as follows (all operation are in the field $\mathbb{F}_p$):

$$s = \sum_{u \in \mathcal{U}'} \lambda_u [s]_u \qquad \text{where} \qquad \lambda_u = \prod_{u \in \mathcal{U}' \setminus \{u\}} \frac{v}{v-u}$$

Packed SS Variant

A packed variant of the SS scheme can be constructed as follows [Franklin, 1992]. Let us consider the case of generating n shares from a secret vector $\vec{s} \in \mathbb{F}_p^l$ where $l < n$. Let t be the reconstruction threshold and $n - l - t$ be the dropout tolerance. We construct a polynomial of degree $l + t - 1$ with the l secret vector inputs and t random masks in $\mathbb{F}_p$ as coefficients. The polynomial evaluations on n distinct non-zero points in $\mathbb{F}_p$ yield the n shares of the SS scheme.

D.2.3 Threshold ElGamal Encryption

The following TEG scheme is additively homomorphic with respect to messages (represented as EC points) [Lopez, 2000]. The TEG scheme uses the Shamir SS scheme [Shamir, 1979], where the algorithm **SS.Share** shares a secret s into n shares while the

algorithm **SS.Recon** allows recovering this secret s by collecting t out of the n shares (see Appendix D.2.2). The TEG scheme is defined by four algorithms.

• $(pk, \{[sk]_u\}_{u \in \mathcal{U}}) \leftarrow$ **TEG.Setup**$(t, \mathcal{U}, \lambda)$: Let t be the threshold for successful secret reconstruction and $\mathcal{U}$ be the set of participants. The algorithm selects $sk \in \mathbb{Z}_p$ where p is a prime number, and computes $pk = g^{sk}$ where g is a generator of the cyclic group built from some elliptic curve. It then shares sk into $[sk]_u$ $\forall u \in \mathcal{U}$ using the algorithm **SS.Share**.

• $\langle m \rangle \leftarrow$ **TEG.Encrypt**(pk, m): To encrypt the message m, the algorithm computes the ciphertext $\langle m \rangle = (c_0, c_1) = (g^r, m \cdot pk^r)$ where $r \in \mathbb{Z}_p$ is the encryption randomness.

• $[m]_u \leftarrow$ **TEG.PartialDecrypt**$([sk]_u, \langle m \rangle)$: The partial decryption $[m]_u$ of the message m is computed as follows: $[m]_u = (c_0^{[sk]_u}, c_1)$ where $\langle m \rangle = (c_0, c_1)$.

• $m \leftarrow$ **TEG.Decrypt**$(\{(u, [m]_u)\}_{u \in \mathcal{U}'})$: The algorithm computes the Lagrange interpolation on the exponent as follows: $c_0^{sk} = \prod_{u \in \mathcal{U}'} (c_0^{[sk]_u})^{\lambda_u}$ where the coefficients λ_u are defined in the algorithm **SS.Recon**, and $\mathcal{U}' \subseteq \mathcal{U}$ such that $|\mathcal{U}'| \geq t$. The algorithm finally computes the original message m as follows: $m = (c_0^{sk})^{-1} \cdot c_1$.

D.2.4 Other Cryptographic Primitives

In our protocols, we employ the following cryptographic primitives for secure communication among clients and server.

Through a secure Key Agreement protocol, denoted as KA, any client can input their private key and the public key of another client to produce a shared secret key. In practice, it can be instantiated with a Diffie–Hellman KA protocol followed by a key derivation function [Bonawitz, 2017a].

Authenticated Encryption, denoted as AE, combines confidentiality and integrity guarantees for messages exchanged between two parties. It consists of a key generation algorithm that outputs a private key, an encryption algorithm **AE.Enc** that takes as input a key and a message, and outputs a ciphertext, and a decryption algorithm **AE.Dec** that takes as input a ciphertext and a key, and outputs the original plaintext.

We also consider a Signature scheme **Sig** that is existentially unforgeable under chosen message attacks. A signing algorithm **Sig.Sign** takes as input a secret key and a message and outputs a signature, and a verification algorithm **Sig.Ver** takes as input a public key, a signature and a message, and outputs '1' if and only if the signature is valid for the

	AsyncLightSecAgg [So, 2022]
Client Comp.	Client: $O(k^2 \frac{d}{\lfloor \delta_k k - t \rfloor} + d)$
	Decryptor: $O(n \frac{d}{\lfloor \delta_k k - t \rfloor})$
Client Comm.	Client: $O(k \frac{d}{\lfloor \delta_k k - t \rfloor} + d)$
	Decryptor: $O(n \frac{d}{\lfloor \delta_k k - t \rfloor})$
Server Comp.	$O(k^2 \frac{d}{\lfloor \delta_k k - t \rfloor} + nd)$

Table D.2.: Complexity analysis for one BAsyncFL round (n: buffer size; k: number of decryptors;t: threshold value; d: input dimension; δ_k: fraction of dropped decryptors).

given message. In practice, it can be instantiated with the Elliptic-Curve Digital Signature Algorithm (ECDSA) followed by a key derivation function.

Finally, a commitment scheme that is perfectly hiding and computationally binding under the discrete logarithm assumption is needed for aggregation verification. A committing algorithm **COM.- Commit** takes as input a message and a witness and outputs a commitment, and an opening algorithm **COM.Open** takes as input a commitment, a message and a witness, and outputs '1' if and only if the commitment is valid for the given message. In practice, it can be instantiated with the Pedersen scheme with vector commitments [Pedersen, 1992].

D.3 Complexity LightSecAgg in BAsyncFL

In Table D.2, we present the complexity analysis for AsyncLightSecAgg. A comparison with Table 5.1 reveals that AsyncLightSecAgg encounters challenges from the substantial input dimension d, which commonly surpasses 10^4 in practical scenarios [Kairouz, 2019]. Consequently, when compared with both **Buffalo** and AsyncDPSecAgg, it consistently exhibits higher computation and communication overheads. They both rely on lattice-based encryption, where the dimensionality of the **LWE** (m) is considerably smaller compared to d.

D.4 Implementation Details of BAsyncFL

Stochastic Gradient Descent (SGD) was employed as the client training algorithm with a learning rate (η), batch size (b), and a specific number (e) of local SGD steps.

More precisely: *(i)* For REPLACE-BG: $\eta = 0.1$, $b = 64$, $e = 20$; *(ii)* For CELEBA: $\eta = 0.01$, $b = 8$, $e = 10$; *(iii)* For SENT140: $\eta = 0.1$, $b = 32$, $e = 10$.

Model parameter updates are converted to 8-bit fixed-point values by multiplying by a factor of 10.

D.5 Hybrid Security Proofs

Buffalo

To prove security of **Buffalo**, we follow the standard simulation-based paradigm. In particular, we show that the view of any attacker against the protocol can be simulated using only the input of the corrupted parties and the protocol output. Intuitively, this means that corrupted parties learn nothing more than their inputs and the intended protocol leakage (i.e. the aggregated input in our case). We prove our protocol in the random oracle model for the active model (the security in the honest-but-curious setting follows directly) since the security of the JL scheme relies on the hash function being seen as a random oracle. We assume that the adversary controls a set $\mathcal{C}$ of corrupted clients in $\mathcal{U}$. As before, the set of the decryptors is $\mathcal{K}$. The honest decryptors form a set $\mathcal{K} \setminus (\mathcal{K} \cap \mathcal{C})$ such that $|\mathcal{K} \setminus (\mathcal{K} \cap \mathcal{C})| > \frac{2k}{3}$.

Given n, k, t, λ and the set $\mathcal{C}$ of adversarial parties, we define $\mathcal{A}$ as a probabilistic polynomial-time algorithm that denotes the "next-message" function of adversarial parties [Bonawitz, 2017a]. This function allows parties in $\mathcal{C}$ to dynamically select their inputs at any round of the protocol execution as well as the list of clients.

We define a simulated execution, represented by a simulator $\mathcal{S}_{\mathcal{C}}^{\mathcal{U},\mathcal{K},t,\lambda,ID^{\xi}}(\mathcal{A})$ (denoted as $\mathcal{S}$ for short), through several modifications to the real execution of the protocol, represented by $REAL_{\mathcal{C}}^{\mathcal{U},\mathcal{K},t,\lambda}(\mathcal{A}, \vec{x}_{\mathcal{U}\setminus\mathcal{C}})$ (denoted as *REAL* for short), such that two subsequent hybrids are computationally indistinguishable. *REAL* exhibits the combined views of the adversarial parties in the protocol execution such that their messages and honest clients' aborts are chosen using $\mathcal{A}$.

$\mathcal{A}$ enables to dynamically set the subset of honest clients for which the server learns their local model aggregation. Therefore, this aggregation cannot be provided for a fixed subset of clients as input to $\mathcal{S}$. Instead, $\mathcal{S}$ will make a single query to an ideal functionality *ID* (seen as a random oracle) that allows it to learn the aggregation for a dynamically chosen subset $\mathcal{L}$ of honest clients, such that $|\mathcal{L}| \geq \xi$ for a lower bound ξ of the number of honest clients. Note that all parties and $\mathcal{A}$ have access to the random oracle.

***Hybrid*$_0$.** Let $\mathcal{C} \subseteq \mathcal{U}$ be any subset of adversarial parties (server and clients) such that there are strictly less than t adversarial decryptors. This hybrid is distributed exactly

as the joint view of $\mathcal{A}$ in *REAL*, that is the joint view of the parties in $\mathcal{C}$ in a real execution.

***Hybrid*$_1$.** In this hybrid, the simulator emulates the real execution. $\mathcal{S}$ knows all the inputs $\{\vec{x}_{u,\tau_u}\}_{u\in\mathcal{U}\setminus\mathcal{C}}$ of the honest parties and runs a full execution of the protocol with $\mathcal{A}$, including a simulation of the random oracle "on the fly", the secure communication channel establishment and the setup phase. Thus, the adversarial view is the same as in *Hybrid*$_0$.

***Hybrid*$_2$.** In this hybrid, given any pair of honest client/decryptor $u \in \mathcal{U}$ and $i \in \mathcal{K}$, the messages between u and i are encrypted before and decrypted after being given to $\mathcal{A}$, using an uniformly random key rather than the one obtained from **KA.agree**$(pp^{KA}, c_u^{SK}, c_i^{PK}) =$ **KA.agree**$(pp^{KA}, c_i^{SK}, c_u^{PK})$. The security of the Diffie-Hellman key agreement **KA** under the Decisional Diffie-Hellman assumption ensures that this hybrid is indistinguishable from *Hybrid*$_1$.

***Hybrid*$_3$.** In this hybrid, $\mathcal{S}$ aborts if $\mathcal{A}$ manages to deliver a message to an honest client u on behalf of another honest decryptor i during **Round 1** (line 11), such that the message is different from the message that $\mathcal{S}$ has given to $\mathcal{A}$ in that phase and that the decryption of this message does not fail (using the proper key). Note that the encryption key that u and i used in *Hybrid*$_2$ was randomly chosen. Hence, based on such a message, the integrity of the ciphertext could be threatened. Since the underlying encryption scheme is INT-CTXT secure, then this hybrid is indistinguishable from the previous one.

***Hybrid*$_4$.** In this hybrid, the simulator changes all encrypted shares sent between pairs of honest clients/decryptors with encryptions of 0. Note that $\mathcal{S}$ still returns the "real" shares in the **Round 3** as it did before. Since the encryption keys were chosen uniformly at random, the IND-CPA security of the underlying encryption scheme guarantees that *Hybrid*$_4$ is indistinguishable from *Hybrid*$_3$.

***Hybrid*$_5$.** In this hybrid, $\mathcal{S}$ aborts if $\mathcal{A}$ gives a signature on a set $\mathcal{U}_{\text{BUFF}}$ which correctly verifies based on an honest decryptor's public key during **Round 3** (line 2), but the honest decryptor has never created the signature on that set. The security of the underlying signature scheme guarantees that forgeries happen with negligible probability. Hence, *Hybrid*$_5$ is indistinguishable from *Hybrid*$_4$.

***Hybrid*$_6$.** Let $\mathcal{Q} \subseteq \mathcal{K}$ be the single set where an honest decryptor received $\mathcal{U}_{\text{BUFF}}$ during **Round 2** (line 1) and then received at least t valid signatures on it during **Round 3** (line 2).

$\mathcal{S}$ aborts if $\mathcal{A}$ asks for key shares for some honest client u either before the adversary has received the responses from the honest parties in **Round 3**, or after such responses have been received for $u \notin \mathcal{U}_{\text{BUFF}}$. In both cases, the JL key sk_{u,τ_u} is information theoretically hidden from $\mathcal{A}$, and the simulator aborts if $\mathcal{A}$ can guess one of those sk_{u,τ_u}, happening with negligible probability. Indeed, the values sk_{u,τ_u} are chosen from the large domain $\mathbb{Z}_{N^2}$. Thus, the adversarial view is the same as in *Hybrid*$_5$.

***Hybrid*$_7$.** In this hybrid, the simulator aborts if $\mathcal{A}$ asks for encryptions of the per-round LWE secret key for some honest client either before the adversary has received the responses from the honest parties in the **Round 3**, or after such responses have been received for $u \notin \mathcal{U}_{\text{BUFF}}$. In both cases, the key $\vec{s}_{u,\tau_u}$ is information-theoretically hidden from $\mathcal{A}$, and the simulator aborts if $\mathcal{A}$ can guess one of those $\vec{s}_{u,\tau_u}$, happening with negligible probability. Indeed, the values $\vec{s}_{u,\tau_u}$ are chosen from the large domain $\mathbb{Z}_q^m$. Thus, the adversarial view is the same as in *Hybrid*$_6$

***Hybrid*$_8$.** In this hybrid, the values $\vec{y}_{u,\tau_u}$ computed by $\mathcal{S}$ on behalf of honest clients and sent to $\mathcal{A}$ during the protection phase are changed with uniformly sampled values such that those values are independent from the rest of the view. Hence, this hybrid is indistinguishable from the previous one.

***Hybrid*$_9$.** For all $u \in \mathcal{U}_{\text{BUFF}} \setminus \mathcal{C}$, we choose values $\vec{w}_{u,\tau_u}$ such that $\sum_{u \in \mathcal{U}_{\text{BUFF}} \setminus \mathcal{C}} \vec{w}_{u,\tau_u} = \sum_{u \in \mathcal{U}_{\text{BUFF}} \setminus \mathcal{C}} \vec{x}_{u,\tau_u}$. Since $\vec{s}_{u,\tau_u}$ and sk_{u,τ_u} are never queried for $u \in \mathcal{U}_{\text{BUFF}} \setminus \mathcal{C}$ by $\mathcal{A}$, then in the view of $\mathcal{A}$, the above values are identically distributed as in the previous hybrid.

***Hybrid*$_{10}$.** In this hybrid, $\mathcal{S}$ does not receive the inputs of honest parties. Instead, during **Round 3**, the simulator submits a query to *ID* for the set $\mathcal{U}_{\text{BUFF}} \setminus \mathcal{C}$ and uses the output to sample the required elements $\vec{w}_{u,\tau_u}$. By construction, we have $|\mathcal{K}| \geq t$ and $|\mathcal{K} \setminus (\mathcal{K} \cap \mathcal{C})| \geq t - k_{\mathcal{C}}$ where $k_{\mathcal{C}} = |\mathcal{K} \cap \mathcal{C}|$. Hence, *ID* will not abort. This change does not modify the view of the adversary, making *Hybrid*$_9$ and *Hybrid*$_{10}$ indistinguishable. Moreover, this hybrid does not make use of the inputs of honest parties, concluding the proof.

Buffalo+

We refer to the other works [Kempen, 2023] (Appendix E), [Ma, 2023] (Appendix D.5) and [Guo, 2022] (Section 3) for an overview of the hybrid security proof of **Buffalo+**. Similar to [Kempen, 2023], **Buffalo+**'s proof follows **Buffalo**'s hybrid proof as described above, in addition to extra hybrid steps that we shortly describe below. Following [Kempen, 2023], we can prove that **Buffalo+** is secure thanks to the unforgeability

property of the digital signature scheme, the binding property of the commitment scheme and the hiding property of the masking technique. Following [Ma, 2023], we can prove that **Buffalo+** is secure thanks to the CPA security of the TEG scheme. Following [Guo, 2022], we can prove that **Buffalo+** is secure thanks to the collision resistance of the underlying hash function.

Bibliography

[Abadi, 2016] Martin Abadi, Andy Chu, Ian Goodfellow, et al. "Deep learning with differential privacy". In: *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 2016, pp. 308–318 (cit. on p. 31).

[Addanki, 2022] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. "Prio+: Privacy preserving aggregate statistics via boolean shares". In: *International Conference on Security and Cryptography for Networks*. Springer. 2022, pp. 516–539 (cit. on p. 71).

[Albrecht, 2015] Martin R Albrecht, Rachel Player, and Sam Scott. "On the concrete hardness of learning with errors". In: *Journal of Mathematical Cryptology* 9.3 (2015), pp. 169–203 (cit. on p. 88).

[Aleppo, 2017] Grazia Aleppo, Katrina J Ruedy, Tonya D Riddlesworth, et al. "REPLACE-BG: a randomized trial comparing continuous glucose monitoring with and without routine blood glucose monitoring in adults with well-controlled type 1 diabetes". In: *Diabetes care* 40.4 (2017), pp. 538–545 (cit. on pp. 34, 40, 69, 89).

[Andrychowicz, 2016] Marcin Andrychowicz, Misha Denil, Sergio Gomez, et al. "Learning to learn by gradient descent by gradient descent". In: *Advances in neural information processing systems* 29 (2016) (cit. on p. 95).

[Arun, 1987] K Somani Arun, Thomas S Huang, and Steven D Blostein. "Least-squares fitting of two 3-D point sets". In: *IEEE Transactions on pattern analysis and machine intelligence* 5 (1987), pp. 698–700 (cit. on p. 100).

[Ashburner, 2000] John Ashburner and Karl J Friston. "Voxel-based morphometry—the methods". In: *Neuroimage* 11.6 (2000), pp. 805–821 (cit. on pp. 12, 24).

[Ashburner, 2007] John Ashburner. "A fast diffeomorphic image registration algorithm". In: *Neuroimage* 38.1 (2007), pp. 95–113 (cit. on pp. 14, 96).

[Ashburner, 2013] John Ashburner and Gerard R Ridgway. "Symmetric diffeomorphic modeling of longitudinal structural MRI". In: *Frontiers in neuroscience* 6 (2013), p. 197 (cit. on p. 12).

[Avants, 2008] Brian B Avants, Charles L Epstein, Murray Grossman, and James C Gee. "Symmetric diffeomorphic image registration with cross-correlation: evaluating automated labeling of elderly and neurodegenerative brain". In: *Medical image analysis* 12.1 (2008), pp. 26–41 (cit. on p. 18).

[Avants, 2011] Brian B Avants, Nicholas J Tustison, Gang Song, et al. "A reproducible evaluation of ANTs similarity metric performance in brain image registration". In: *Neuroimage* 54.3 (2011), pp. 2033–2044 (cit. on p. 14).

[Baker, 2004] Simon Baker and Iain Matthews. "Lucas-Kanade 20 years on: A unifying framework". In: *International journal of computer vision* 56.3 (2004), pp. 221–255 (cit. on p. 16).

[Balakrishnan, 2019] Guha Balakrishnan, Amy Zhao, Mert R Sabuncu, John Guttag, and Adrian V Dalca. "VoxelMorph: a learning framework for deformable medical image registration". In: *IEEE transactions on medical imaging* 38.8 (2019), pp. 1788–1800 (cit. on pp. 31, 97).

[Beg, 2005] M Faisal Beg, Michael I Miller, Alain Trouvé, and Laurent Younes. "Computing large deformation metric mappings via geodesic flows of diffeomorphisms". In: *International journal of computer vision* 61 (2005), pp. 139–157 (cit. on p. 14).

[Beguier, 2020] Constance Beguier, Mathieu Andreux, and Eric W Tramel. "Efficient sparse secure aggregation for federated learning". In: *arXiv preprint arXiv:2007.14861* (2020) (cit. on p. 98).

[Bell, 2020] James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. "Secure Single-Server Aggregation with (Poly)Logarithmic Overhead". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security – CCS '20*. 2020, pp. 1253–1269 (cit. on pp. 36, 45, 48, 56, 68, 70, 71, 73).

[Bell, 2023] James Bell, Adrià Gascón, Tancrède Lepoint, et al. "{ACORN}: input validation for secure aggregation". In: *32nd USENIX Security Symposium (USENIX Security 23)*. 2023, pp. 4805–4822 (cit. on pp. 68, 70, 71, 77, 80, 88).

[Bellare, 1994] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. "Incremental cryptography: The case of hashing and signing". In: *Advances in Cryptology—CRYPTO'94: 14th Annual International Cryptology Conference Santa Barbara, California, USA August 21–25, 1994 Proceedings 14*. Springer. 1994, pp. 216–233 (cit. on p. 78).

[Benaissa, 2021] Ayoub Benaissa, Bilal Retiat, Bogdan Cebere, and Alaa Eddine Belfedhal. "TenSEAL: A library for encrypted tensor operations using homomorphic encryption". In: *arXiv preprint arXiv:2104.03152* (2021) (cit. on pp. 22, 23, 26).

[Bernstein, 2008] Daniel J Bernstein et al. "ChaCha, a variant of Salsa20". In: *Workshop record of SASC*. Vol. 8. 1. Citeseer. 2008, pp. 3–5 (cit. on p. 39).

[Beutel, 2020] Daniel J Beutel, Taner Topal, Akhil Mathur, et al. "Flower: A friendly federated learning research framework". In: *arXiv preprint arXiv:2007.14390* (2020) (cit. on pp. 34, 36).

[Boemer, 2021] Fabian Boemer, Sejun Kim, Gelila Seifu, Fillipe DM de Souza, and Vinodh Gopal. "Intel hexl: Accelerating homomorphic encryption with intel avx512-ifma52". In: *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 2021, pp. 57–62 (cit. on p. 31).

[Bonawitz, 2017a] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, et al. "Practical Secure Aggregation for Privacy-Preserving Machine Learning". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security – CCS '17*. 2017, pp. 1175–1191 (cit. on pp. 36, 45, 48–51, 53–57, 70, 71, 73, 76, 80, 81, 107, 110, 117, 119).

[Bonawitz, 2017b] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, et al. "Practical secure aggregation for privacy-preserving machine learning". In: *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 1175–1191 (cit. on p. 31).

[Bonawitz, 2019a] Kallista A. Bonawitz, Hubert Eichner, Wolfgang Grieskamp, et al. "Towards Federated Learning at Scale: System Design". In: *CoRR* abs/1902.01046 (2019) (cit. on pp. 68, 69, 73).

[Bonawitz, 2019b] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, et al. "Towards federated learning at scale: System design". In: *Proceedings of machine learning and systems* 1 (2019), pp. 374–388 (cit. on p. 45).

[Bottou, 2004] Léon Bottou. "Stochastic Learning". In: *Advanced Lectures on Machine Learning*. Ed. by Olivier Bousquet and Ulrike von Luxburg. Lecture Notes in Artificial Intelligence – LNAI 3176. Berlin: Springer Verlag, 2004, pp. 146–168 (cit. on p. 35).

[Brakerski, 2011] Zvika Brakerski and Vinod Vaikuntanathan. "Fully homomorphic encryption from ring-LWE and security for key dependent messages". In: *Annual cryptology conference*. Springer. 2011, pp. 505–524 (cit. on p. 42).

[Buyukates, 2022] Baturalp Buyukates, Jinhyun So, Hessam Mahdavifar, and Salman Avestimehr. "LightVeriFL: Lightweight and Verifiable Secure Federated Learning". In: *Workshop on Federated Learning: Recent Advances and New Challenges (in Conjunction with NeurIPS 2022)*. 2022 (cit. on pp. 71, 75, 77, 78, 82, 86–88, 90).

[Caldas, 2018] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, et al. "Leaf: A benchmark for federated settings". In: *arXiv preprint arXiv:1812.01097* (2018) (cit. on pp. 64, 89).

[Cardoso, 2013] M Jorge Cardoso, Kelvin Leung, Marc Modat, et al. "STEPS: Similarity and Truth Estimation for Propagated Segmentations and its application to hippocampal segmentation and brain parcelation". In: *Medical image analysis* 17.6 (2013), pp. 671–684 (cit. on p. 12).

[Chen, 2021] Megan Chen, Carmit Hazay, Yuval Ishai, et al. "Diogenes: lightweight scalable RSA modulus generation with a dishonest majority". In: *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021, pp. 590–607 (cit. on pp. 50, 80, 84, 88).

[Cheon, 2017] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. "Homomorphic encryption for arithmetic of approximate numbers". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2017, pp. 409–437 (cit. on pp. 4, 19, 36).

[Clark, 2013] Kenneth Clark, Bruce Vendt, Kirk Smith, et al. "The Cancer Imaging Archive (TCIA): maintaining and operating a public information repository". In: *Journal of digital imaging* 26 (2013), pp. 1045–1057 (cit. on p. 24).

[Corrigan-Gibbs, 2017] Henry Corrigan-Gibbs and Dan Boneh. "Prio: Private, robust, and scalable computation of aggregate statistics". In: *14th USENIX symposium on networked systems design and implementation (NSDI 17)*. 2017, pp. 259–282 (cit. on p. 71).

[Cremonesi, 2023] Francesco Cremonesi, Marc Vesin, Sergen Cansiz, et al. "Fed-BioMed: open, transparent and trusted federated learning for real-world healthcare applications". In: *arXiv preprint arXiv:2304.12012* (2023) (cit. on p. 34).

[Cui, 2021] Ran Cui, Chirath Hettiarachchi, Christopher J Nolan, Elena Daskalaki, and Hanna Suominen. "Personalised short-term glucose prediction via recurrent self-attention network". In: *2021 IEEE 34th International Symposium on Computer-Based Medical Systems (CBMS)*. IEEE. 2021, pp. 154–159 (cit. on p. 89).

[Dale, 1999] Anders M Dale, Bruce Fischl, and Martin I Sereno. "Cortical surface-based analysis: I. Segmentation and surface reconstruction". In: *Neuroimage* 9.2 (1999), pp. 179–194 (cit. on p. 12).

[Damgård, 2012] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. "Multiparty computation from somewhat homomorphic encryption". In: *Annual Cryptology Conference*. Springer. 2012, pp. 643–662 (cit. on p. 19).

[Diffie, 2022] Whitfield Diffie and Martin E Hellman. "New directions in cryptography". In: *Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman*. 2022, pp. 365–390 (cit. on pp. 39, 98).

[Dowson, 2006] Nicholas Dowson and Richard Bowden. "A unifying framework for mutual information methods for use in non-linear optimisation". In: *European Conference on Computer Vision*. Springer. 2006, pp. 365–378 (cit. on p. 17).

[Dowson, 2007] Nicholas Dowson and Richard Bowden. "Mutual information for lucas-kanade tracking (milk): An inverse compositional formulation". In: *IEEE transactions on pattern analysis and machine intelligence* 30.1 (2007), pp. 180–185 (cit. on p. 17).

[Dwork, 2006] Cynthia Dwork. "Differential privacy". In: *International colloquium on automata, languages, and programming*. Springer. 2006, pp. 1–12 (cit. on pp. 6, 34).

[Falta, 2022] Fenja Falta, Lasse Hansen, and Mattias P Heinrich. “Learning iterative optimisation for deformable image registration of lung CT with recurrent convolutional networks”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2022, pp. 301–309 (cit. on p. 96).

[Fei, 2021] Jiawei Fei, Chen-Yu Ho, Atal N Sahu, Marco Canini, and Amedeo Sapio. “Efficient sparse collective communication and its application to accelerate distributed deep learning”. In: *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 2021, pp. 676–691 (cit. on pp. 82, 90).

[Fraboni, 2023] Yann Fraboni, Richard Vidal, Laetitia Kameni, and Marco Lorenzi. “A General Theory for Federated Optimization with Asynchronous and Heterogeneous Clients Updates”. In: *Journal of Machine Learning Research* 24.110 (2023), pp. 1–43 (cit. on p. 45).

[Franklin, 1992] Matthew Franklin and Moti Yung. “Communication complexity of secure computation”. In: *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*. 1992, pp. 699–710 (cit. on p. 116).

[Garyfallidis, 2014] Eleftherios Garyfallidis, Matthew Brett, Bagrat Amirbekian, et al. “Dipy, a library for the analysis of diffusion MRI data”. In: *Frontiers in neuroinformatics* 8 (2014), p. 8 (cit. on p. 26).

[Gazula, 2021] Harshvardhan Gazula, Bharath Holla, Zuo Zhang, et al. “Decentralized multisite VBM analysis during adolescence shows structural changes linked to age, body mass index, and smoking: a COINSTAC analysis”. In: *Neuroinformatics* 19.4 (2021), pp. 553–566 (cit. on p. 12).

[Guo, 2020] Xiaojie Guo, Zheli Liu, Jin Li, et al. “V eri fl: Communication-efficient and fast verifiable aggregation for federated learning”. In: *IEEE Transactions on Information Forensics and Security* 16 (2020), pp. 1736–1751 (cit. on p. 71).

[Guo, 2022] Xiaojie Guo. “Fixing Issues and Achieving Maliciously Secure Verifiable Aggregation in“VeriFL: Communication-Efficient and Fast Verifiable Aggregation for Federated Learning””. In: *Cryptology ePrint Archive* (2022) (cit. on pp. 71, 121, 122).

[Haralampieva, 2020] Veneta Haralampieva, Daniel Rueckert, and Jonathan Passerat-Palmbach. “A systematic comparison of encrypted machine learning solutions for image classification”. In: *Proceedings of the 2020 workshop on privacy-preserving machine learning in practice*. 2020, pp. 55–59 (cit. on p. 22).

[Heinrich, 2011] Mattias P Heinrich, Mark Jenkinson, Manav Bhushan, et al. “Non-local shape descriptor: A new similarity metric for deformable multi-modal registration”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2011, pp. 541–548 (cit. on p. 12).

[Heinrich, 2012] Mattias P Heinrich, Mark Jenkinson, Manav Bhushan, et al. “MIND: Modality independent neighbourhood descriptor for multi-modal deformable registration”. In: *Medical image analysis* 16.7 (2012), pp. 1423–1435 (cit. on p. 96).

[Hering, 2022] Alessa Hering, Lasse Hansen, Tony CW Mok, et al. "Learn2Reg: comprehensive multi-task medical image registration challenge, dataset and evaluation in the era of deep learning". In: *IEEE Transactions on Medical Imaging* 42.3 (2022), pp. 697–712 (cit. on pp. 24, 27).

[Hospedales, 2021] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. "Meta-learning in neural networks: A survey". In: *IEEE transactions on pattern analysis and machine intelligence* 44.9 (2021), pp. 5149–5169 (cit. on p. 96).

[Ibarrondo, 2023] Alberto Ibarrondo, Hervé Chabanne, and Melek Önen. "Funshade: Function Secret Sharing for Two-Party Secure Thresholded Distance Evaluation". In: *PETS 2023, 23rd Privacy Enhancing Technologies Symposium*. Vol. 2023. 4. 2023, pp. 21–34 (cit. on p. 96).

[Innocenti, 2023] Lucia Innocenti, Michela Antonelli, Francesco Cremonesi, et al. "Benchmarking Collaborative Learning Methods Cost-Effectiveness for Prostate Segmentation". In: *arXiv preprint arXiv:2309.17097* (2023) (cit. on pp. 34, 41).

[Jaloli, 2023] Mehrad Jaloli and Marzia Cescon. "Long-term prediction of blood glucose levels in type 1 diabetes using a cnn-lstm-based deep neural network". In: *Journal of diabetes science and technology* 17.6 (2023), pp. 1590–1601 (cit. on p. 89).

[Joye, 2013] Marc Joye and Benoît Libert. "A Scalable Scheme for Privacy-Preserving Aggregation of Time-Series Data". In: *Proceedings of the International Conference on Financial Cryptography and Data Security – FC '13*. 2013, pp. 111–125 (cit. on pp. 34, 38, 40, 46, 47, 76, 77).

[Kadhe, 2020] Swanand Kadhe, Nived Rajaraman, O Ozan Koyluoglu, and Kannan Ramchandran. "Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning". In: *arXiv preprint arXiv:2009.11248* (2020) (cit. on pp. 45, 56).

[Kairouz, 2019] Peter Kairouz, H. Brendan McMahan, Brendan Avent, et al. "Advances and Open Problems in Federated Learning". In: *CoRR* abs/1912.04977 (2019) (cit. on pp. 2, 72, 118).

[Kaissis, 2021] Georgios Kaissis, Alexander Ziller, Jonathan Passerat-Palmbach, et al. "End-to-end privacy preserving deep learning on multi-institutional medical imaging". In: *Nature Machine Intelligence* 3.6 (2021), pp. 473–484 (cit. on pp. 12, 31).

[Keller, 2020] Marcel Keller. "MP-SPDZ: A versatile framework for multi-party computation". In: *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*. 2020, pp. 1575–1590 (cit. on p. 39).

[Kempen, 2023] Elina van Kempen, Qifei Li, Giorgia Azzurra Marson, and Claudio Soriente. "LISA: LIghtweight single-server Secure Aggregation with a public source of randomness". In: *arXiv preprint arXiv:2308.02208* (2023) (cit. on p. 121).

[Konečnỳ, 2016] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, et al. "Federated learning: Strategies for improving communication efficiency". In: *arXiv preprint arXiv:1610.05492* (2016) (cit. on pp. 65, 98).

[Krebs, 2017] Julian Krebs, Tommaso Mansi, Hervé Delingette, et al. "Robust non-rigid registration through agent-based action learning". In: *Medical Image Computing and Computer Assisted Intervention- MICCAI 2017: 20th International Conference, Quebec City, QC, Canada, September 11-13, 2017, Proceedings, Part I 20*. Springer. 2017, pp. 344–352 (cit. on p. 31).

[Krizhevsky, 2014] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. "The CIFAR-10 dataset". In: *online: http://www. cs. toronto. edu/kriz/cifar. html* 55.5 (2014) (cit. on p. 64).

[Kursawe, 2011] Klaus Kursawe, George Danezis, and Markulf Kohlweiss. "Privacy-friendly aggregation for the smart-grid". In: *Privacy Enhancing Technologies: 11th International Symposium, PETS 2011, Waterloo, ON, Canada, July 27-29, 2011. Proceedings 11*. Springer. 2011, pp. 175–191 (cit. on pp. 34, 39, 40).

[Lauter, 2021] K. Lauter. *Private AI: Machine Learning on Encrypted Data*. Tech. rep. eprint report `https://eprint.iacr.org/2021/324.pdf`. 2021 (cit. on p. 13).

[LeCun, 1998] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on p. 64).

[Lee, 2018] Joohee Lee, Dongwoo Kim, Duhyeong Kim, et al. "Instant privacy-preserving biometric authentication for hamming distance". In: *Cryptology ePrint Archive* (2018) (cit. on p. 77).

[Li, 2023] Hanjun Li, Huijia Lin, Antigoni Polychroniadou, and Stefano Tessaro. "LERNA: Secure Single-Server Aggregation via Key-Homomorphic Masking". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2023, pp. 302–334 (cit. on pp. 56, 70, 71, 73, 77, 78).

[Lopez, 2000] Julio Lopez and Ricardo Dahab. "An overview of elliptic curve cryptography". In: *Institute of Computing, Sate University of Campinas, Sao Paulo, Brazil, Tech. Rep* (2000) (cit. on pp. 115, 116).

[Lotan, 2020] Eyal Lotan, Charlotte Tschider, Daniel K Sodickson, et al. "Medical imaging and privacy in the era of artificial intelligence: myth, fallacy, and the future". In: *Journal of the American College of Radiology* 17.9 (2020), pp. 1159–1162 (cit. on p. 13).

[Ma, 2023] Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin. "Flamingo: Multi-round single-server secure aggregation with applications to private federated learning". In: *Cryptology ePrint Archive, Paper 2023/486* (2023) (cit. on pp. 45, 55, 56, 58, 63, 65, 68, 70, 71, 73, 75, 78, 80, 81, 83, 84, 88, 121, 122).

[Mansouri, 2022] Mohamad Mansouri, Melek Önen, and Wafa Ben Jaballah. "Learning from Failures: Secure and Fault-Tolerant Aggregation for Federated Learning". In: *Proceedings of the 38th Annual Computer Security Applications Conference – ACSAC '22*. 2022, pp. 146–158 (cit. on pp. 39, 40, 45–48, 50, 53–55, 57, 61, 62, 68, 73, 82, 88).

[Mansouri, 2023] Mohamad Mansouri, Melek Onen, Wafa Ben Jaballah, and Mauro Conti. “Sok: Secure aggregation based on cryptographic schemes for federated learning”. In: *Proceedings on Privacy Enhancing Technologies – PoPETS ’23* (2023), pp. 140–157 (cit. on pp. 5, 34, 35, 44, 68).

[Mattes, 2003] David Mattes, David R Haynor, Hubert Vesselle, Thomas K Lewellen, and William Eubank. “PET-CT image registration in the chest using free-form deformations”. In: *IEEE transactions on medical imaging* 22.1 (2003), pp. 120–128 (cit. on pp. 16, 22, 31).

[McMahan, 2017a] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. “Communication-Efficient Learning of Deep Networks from Decentralized Data”. In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics – AISTATS ’17*. Vol. 54. 2017, pp. 1273–1282 (cit. on pp. 2, 35, 44, 68, 72).

[McMahan, 2017b] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. “Communication-efficient learning of deep networks from decentralized data”. In: *Artificial intelligence and statistics*. PMLR. 2017, pp. 1273–1282 (cit. on p. 12).

[Modersitzki, 2009] J. Modersitzki. *FAIR: Flexible Algorithms for Image Registration*. Philadelphia: SIAM, 2009 (cit. on p. 14).

[Mueller, 2005] Susanne G Mueller, Michael W Weiner, Leon J Thal, et al. “The Alzheimer’s disease neuroimaging initiative”. In: *Neuroimaging Clinics* 15.4 (2005), pp. 869–877 (cit. on pp. 24, 27).

[Nasr, 2019] Milad Nasr, Reza Shokri, and Amir Houmansadr. “Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning”. In: *Proceedings of the 2019 IEEE Symposium on Security and Privacy – SP ’19*. 2019, pp. 739–753 (cit. on pp. 5, 36, 44, 68, 73).

[Nejatollahi, 2017] Hamid Nejatollahi, Nikil Dutt, and Rosario Cammarota. “Special session: Trends, challenges and needs for lattice-based cryptography implementations”. In: *2017 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE. 2017, pp. 1–3 (cit. on p. 98).

[Ngong, 2023] Ivoline C Ngong, Nicholas Gibson, and Joseph P Near. “OLYMPIA: A Simulation Framework for Evaluating the Concrete Scalability of Secure Aggregation Protocols”. In: *arXiv preprint arXiv:2302.10084* (2023) (cit. on pp. 87, 88).

[Nguyen, 2022] John Nguyen, Kshitiz Malik, Hongyuan Zhan, et al. “Federated learning with buffered asynchronous aggregation”. In: *Proceedings of the International Conference on Artificial Intelligence and Statistics – AISTATS ’22*. 2022, pp. 3581–3607 (cit. on pp. 45, 57, 68, 69, 73, 89).

[Nishide, 2011] Takashi Nishide and Kouichi Sakurai. “Distributed Paillier Cryptosystem without Trusted Dealer”. In: *Proceedings of the International Workshop on Information Security Applications – WISA ’11*. 2011, pp. 44–60 (cit. on p. 50).

[Ogier du Terrail, 2022] Jean Ogier du Terrail, Samy-Safwan Ayed, Edwige Cyffers, et al. "Flamby: Datasets and benchmarks for cross-silo federated learning in realistic healthcare settings". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 5315–5334 (cit. on pp. 34, 40).

[Paillier, 1999] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques – EUROCRYPT '99*. 1999, pp. 223–238 (cit. on pp. 47, 77).

[Parzen, 1961] Emanuel Parzen. "Mathematical considerations in the estimation of spectra". In: *Technometrics* 3.2 (1961), pp. 167–190 (cit. on p. 16).

[Pasquini, 2022] Dario Pasquini, Danilo Francati, and Giuseppe Ateniese. "Eluding secure aggregation in federated learning via model inconsistency". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2022, pp. 2429–2443 (cit. on pp. 53, 54, 85).

[Paszke, 2019] Adam Paszke, Sam Gross, Francisco Massa, et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32 (2019) (cit. on p. 65).

[Pedersen, 1992] Torben Pryds Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *Advances in Cryptology — CRYPTO '91*. Springer Berlin Heidelberg, 1992 (cit. on p. 118).

[Pennec, 1999] Xavier Pennec, Pascal Cachier, and Nicholas Ayache. "Understanding the "Demon's algorithm": 3D non-rigid registration by gradient descent". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 1999, pp. 597–605 (cit. on pp. 14, 16).

[Rabin, 1998] Tal Rabin. "A Simplified Approach to Threshold and Proactive RSA". In: *Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology – CRYPTO '98*. 1998, pp. 89–104 (cit. on p. 47).

[Rathee, 2023] Mayank Rathee, Conghao Shen, Sameer Wagh, and Raluca Ada Popa. "Elsa: Secure aggregation for federated learning with malicious actors". In: *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2023, pp. 1961–1979 (cit. on p. 71).

[Reina, 2021] G Anthony Reina, Alexey Gruzdev, Patrick Foley, et al. "OpenFL: An open-source framework for Federated Learning". In: *arXiv preprint arXiv:2105.06413* (2021) (cit. on pp. 34, 36).

[Reuter, 2010] Martin Reuter, H Diana Rosas, and Bruce Fischl. "Highly accurate inverse consistent registration: a robust approach". In: *Neuroimage* 53.4 (2010), pp. 1181–1196 (cit. on p. 12).

[Rivest, 1978a] R. L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Commun. ACM* 21 (1978) (cit. on p. 98).

[Rivest, 1978b] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. "On data banks and privacy homomorphisms". In: *Foundations of secure computation* 4.11 (1978), pp. 169–180 (cit. on pp. 13, 19).

[Ronneberger, 2015] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer. 2015, pp. 234–241 (cit. on p. 97).

[Roth, 2022] Holger R Roth, Yan Cheng, Yuhong Wen, et al. "Nvidia flare: Federated learning from simulation to real-world". In: *arXiv preprint arXiv:2210.13291* (2022) (cit. on pp. 34, 36).

[Ruder, 2016] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016) (cit. on p. 72).

[Ryffel, 2018] Theo Ryffel, Andrew Trask, Morten Dahl, et al. "A generic framework for privacy preserving deep learning". In: *arXiv preprint arXiv:1811.04017* (2018) (cit. on p. 26).

[Sabt, 2015] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. "Trusted execution environment: What it is, and what it is not". In: *2015 IEEE Trustcom/BigDataSE/Ispa*. Vol. 1. IEEE. 2015, pp. 57–64 (cit. on p. 6).

[Sabuncu, 2004] Mert R Sabuncu and Peter J Ramadge. "Gradient based nonuniform subsampling for information-theoretic alignment methods". In: *The 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. Vol. 1. IEEE. 2004, pp. 1683–1686 (cit. on pp. 22, 31).

[Sandhya, 2017] Mulagala Sandhya and Munaga VNK Prasad. "Biometric template protection: A systematic literature review of approaches and modalities". In: *Biometric Security and Privacy: Opportunities & Challenges in The Big Data Era* (2017), pp. 323–370 (cit. on p. 96).

[Schnabel, 2016] Julia A Schnabel, Mattias P Heinrich, Bartłomiej W Papież, and J Michael Brady. "Advances and challenges in deformable image registration: from image fusion to complex motion modelling". In: *Medical Image Analysis* 33 (2016), pp. 145–148 (cit. on p. 12).

[Shamir, 1979] Adi Shamir. "How to Share a Secret". In: *Communications of the ACM* (1979), pp. 612–613 (cit. on pp. 5, 38, 47, 59, 116).

[Shattuck, 2009] David W Shattuck, Gautam Prasad, Mubeena Mirza, Katherine L Narr, and Arthur W Toga. "Online resource for validation of brain segmentation methods". In: *NeuroImage* 45.2 (2009), pp. 431–439 (cit. on p. 12).

[Shokri, 2017] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. "Membership Inference Attacks Against Machine Learning Models". In: *Proceedings of the 2017 IEEE Symposium on Security and Privacy – SP '17*. 2017, pp. 3–18 (cit. on pp. 2, 5, 36, 44, 68, 73, 97).

[Shor, 1999] Peter W Shor. “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer”. In: *SIAM review* 41.2 (1999), pp. 303–332 (cit. on p. 98).

[Simonovsky, 2016] Martin Simonovsky, Benjamín Gutiérrez-Becker, Diana Mateus, Nassir Navab, and Nikos Komodakis. “A deep metric for multimodal registration”. In: *Medical Image Computing and Computer-Assisted Intervention-MICCAI 2016: 19th International Conference, Athens, Greece, October 17-21, 2016, Proceedings, Part III 19*. Springer. 2016, pp. 10–18 (cit. on p. 31).

[So, 2021] Jinhyun So, Başak Güler, and A. Salman Avestimehr. “Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning”. In: *IEEE Journal on Selected Areas in Information Theory* 2.1 (2021), pp. 479–489 (cit. on pp. 45, 56).

[So, 2022] Jinhyun So, Chaoyang He, Chien-Sheng Yang, et al. “Lightsecagg: a lightweight and versatile design for secure aggregation in federated learning”. In: *Proceedings of Machine Learning and Systems* 4 (2022), pp. 694–720 (cit. on pp. 45, 46, 54, 56, 59, 63, 70, 74, 82, 85, 87, 118).

[Stevens, 2022] Timothy Stevens, Christian Skalka, Christelle Vincent, et al. “Efficient differentially private secure aggregation for federated learning via hardness of learning with errors”. In: *31st USENIX Security Symposium (USENIX Security 22)*. 2022, pp. 1379–1395 (cit. on pp. 70, 71, 74, 85, 86, 89).

[Ström, 2015] Nikko Ström. “Scalable distributed DNN training using commodity GPU cloud computing”. In: (2015) (cit. on pp. 82, 98).

[Sun, 2020] Jun Sun, Tianyi Chen, Georgios B Giannakis, Qinmin Yang, and Zaiyue Yang. “Lazily aggregated quantized gradient innovation for communication-efficient federated learning”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.4 (2020), pp. 2031–2044 (cit. on p. 82).

[Taiello, 2022] Riccardo Taiello, Melek Önen, Olivier Humbert, and Marco Lorenzi. “Privacy Preserving Image Registration”. In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2022*. Ed. by Linwei Wang, Qi Dou, P. Thomas Fletcher, Stefanie Speidel, and Shuo Li. Cham: Springer Nature Switzerland, 2022, pp. 130–140 (cit. on pp. 10, 12, 31).

[Taiello, 2024a] Riccardo Taiello, Melek Önen, Francesco Capano, Olivier Humbert, and Marco Lorenzi. “Privacy preserving image registration”. In: *Medical Image Analysis* 94 (2024), p. 103129 (cit. on pp. 10, 12).

[Taiello, 2024b] Riccardo Taiello, Melek Önen, Clémentine Gritti, and Marco Lorenzi. “Let Them Drop: Scalable and Efficient Federated Learning Solutions Agnostic to Client Stragglers”. In: *Cryptology ePrint Archive* (2024) (cit. on pp. 10, 44).

[Tan, 2021] Sijun Tan, Brian Knott, Yuan Tian, and David J Wu. “CryptGPU: Fast privacy-preserving machine learning on the GPU”. In: *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021, pp. 1021–1038 (cit. on p. 97).

[Terrail, 2022] Jean Ogier du Terrail, Samy-Safwan Ayed, Edwige Cyffers, et al. “Flamby: Datasets and benchmarks for cross-silo federated learning in realistic healthcare settings”. In: *arXiv preprint arXiv:2210.04620* (2022) (cit. on p. 97).

[Vachet, 2012] Clement Vachet, Benjamin Yvernault, Kshamta Bhatt, et al. “Automatic corpus callosum segmentation using a deformable active Fourier contour model”. In: *Medical Imaging 2012: Biomedical Applications in Molecular, Structural, and Functional Imaging*. Vol. 8317. SPIE. 2012, pp. 79–85 (cit. on p. 24).

[Veugen, 2019] Thijs Veugen, Thomas Attema, and Gabriele Spini. “An implementation of the Paillier crypto system with threshold decryption without a trusted dealer”. In: *Cryptology ePrint Archive, Paper 2019/1136* (2019) (cit. on p. 50).

[Viola, 1997] Paul Viola and William M Wells III. “Alignment by maximization of mutual information”. In: *International journal of computer vision* 24.2 (1997), pp. 137–154 (cit. on pp. 16, 22, 31).

[Watson, 2022] Jean-Luc Watson, Sameer Wagh, and Raluca Ada Popa. “Piranha: A {GPU} platform for secure computation”. In: *31st USENIX Security Symposium (USENIX Security 22)*. 2022, pp. 827–844 (cit. on p. 97).

[Xie, 2019] Cong Xie, Sanmi Koyejo, and Indranil Gupta. “Asynchronous federated optimization”. In: *arXiv preprint arXiv:1903.03934* (2019) (cit. on pp. 45, 69, 89).

[Xu, 2019] Guowen Xu, Hongwei Li, Sen Liu, Kan Yang, and Xiaodong Lin. “Verifynet: Secure and verifiable federated learning”. In: *IEEE Transactions on Information Forensics and Security* 15 (2019), pp. 911–926 (cit. on p. 71).

[Yang, 2017] Xiao Yang, Roland Kwitt, Martin Styner, and Marc Niethammer. “Quicksilver: Fast predictive image registration–a deep learning approach”. In: *NeuroImage* 158 (2017), pp. 378–396 (cit. on p. 31).

[Yang, 2018] Timothy Yang, Galen Andrew, Hubert Eichner, et al. “Applied federated learning: Improving google keyboard query suggestions”. In: *arXiv preprint arXiv:1812.02903* (2018) (cit. on p. 45).

[Yao, 1982] Andrew C Yao. “Protocols for secure computations”. In: *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE. 1982, pp. 160–164 (cit. on pp. 5, 13, 19).

[Yu, 2018] Qian Yu, Netanel Raviv, Jinhyun So, and Amir Salman Avestimehr. “Lagrange Coded Computing: Optimal Design for Resiliency, Security and Privacy”. In: *CoRR* abs/1806.00939 (2018). arXiv: 1806.00939 (cit. on pp. 56, 60).

[Zerka, 2020] Fadila Zerka, Visara Urovi, Akshayaa Vaidyanathan, et al. “Blockchain for privacy preserving and trustworthy distributed machine learning in multicentric medical imaging (C-DistriM)”. In: *Ieee Access* 8 (2020), pp. 183939–183951 (cit. on p. 12).