FACULTY OF SCIENCE AND TECHNOLOGY

MASTER THESIS

Study programme / specialisation:
Data Science

The spring semester, 2022

[✓ Open] / Confidential

Author: Febrianti Wibawa

[illegible]
.............FEBRIANTI WIBAWA............
(signature author)

Course coordinator: Tom Ryen

Supervisor(s): Ferhat Özgur Catak

Thesis title: Privacy-Preserving Machine Learning for Health Institutes

Credits (ECTS): 30

Keywords: machine learning, federated learning, homomorphic encryption

Pages: ......59.........

+ appendix: ...4.........

Stavanger, June, 15 2022
date/year

University of Stavanger

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Privacy-Preserving Machine Learning for Health Institutes

Master's Thesis in Computer Science
by

Febrianti Wibawa

Supervisor

Ferhat Özgur Catak

June 2022

Abstract

Medical data is, due to its nature, often susceptible to data privacy and security concerns. The identity of a person can be derived from medical data. Federated learning, one type of machine learning technique, is popularly used to improve the privacy and security of medical data. In federated learning, the training data is distributed across multiple machines, and the learning process of deep learning (DL) models is performed collaboratively. However, the privacy of DL models is not protected. Privacy attacks on the DL models aim to obtain sensitive information. Therefore, the DL models should be protected from adversarial attacks, especially those which utilize medical data. One of the solutions to solve this problem is homomorphic encryption-based model protection. This paper proposes a privacy-preserving federated learning algorithm for medical data using homomorphic encryption. The proposed algorithm uses a Secure Multiparty Computation (SMPC) protocol to protect the deep learning model from adversaries. In this study, the proposed algorithm using a real-world medical dataset is evaluated in terms of the model performance.

Acknowledgement

I would thank my family and friends for all their support and understanding during my master's degree. It has been quite a journey, working on both my education and daily full-time job. Perseverance, I guess is the word which can describe the journey.

I would also like to thank my supervisor Ferhat Özgur Catak for his advice and support. He's the person who introduced me to Homomorphic Encryption, which I have never heard before but yet so 'magical'.

Contents

Chapter 1

Introduction

1.1 Background and motivation

The recent pandemic COVID-19 hit countries around the world mercilessly and without exception. Researchers from different parts of the world rushed against time to produce vaccines while country leaders tried to find a good balance between the economy and public health measures. Some countries enforced a zero COVID-19 policy, whilst others enforced a herd immunity policy whereby a majority of the population should contract the virus.

Health institutions around the world worked together to find solutions. Countries that contracted the virus first, such as China, shared their information on virus features, symptoms, and its effects on human organs, with other countries. This cooperation is crucial as one country alone cannot resolve nor contain the global pandemic. For this reason, information sharing should be done seamlessly across countries.

COVID-19 is contracted by breathing contaminated air and contacting with contaminated surfaces, allowing the virus to spread exponentially. The diagnosis and treatments have to be determined quickly in order to save lives. Time and resources are essential in solving this global pandemic.

With current technological advancements, health institutions are able to develop and train machine learning models using their local datasets. They build their own machine learning model and train it using local samples. The models support the medical staff in fighting against COVID-19, especially in diagnosing COVID-19.

One example of machine learning model usage is to detect COVID-19 contraction based on X-ray images of individuals. The COVID-19 detection model is an image classification model primarily developed on a neural network framework with X-ray images as input.

In order to have a machine learning model that can provide accurate predictions, datasets used in model training should be varied. This condition can only be met in larger health institutions. In this case, having locally trained machine learning models is not good enough. The solution is ineffective in solving a global pandemic situation such as COVID-19, where collaboration and information sharing are essential.

In order to solve a global problem, global cooperation is needed.

1.2 Problem Definition

Large and diverse samples are indeed required in any given machine learning model, and the COVID-19 detection model is no exception to this rule. Lack of data diversity can potentially result in model overfitting, but not all health institutions are able to obtain such samples. Therefore collaboration among health institutions is necessary.

One way of collaboration can be in the form of data sharing. However, the medical information of a patient is considered sensitive information since it contains private information such as fingerprints, from which one can derive a person's identity. Therefore it is required to preserve its privacy. A known procedure to protect data privacy is by applying encryption. Due to its nature, it is almost impossible to transfer the data across the border without in some way violating data protection regulations such as General Data Protection Regulation (GDPR) in the European Union (EU).

Another way to collaborate is by sharing the machine learning model itself. This way, the model is trained collaboratively without requiring data transfer. But this approach has its challenge. There are situations where the model's privacy should also be protected, such as when collaboration takes place in an untrusted public cloud domain. In this case, the model should be protected from any attacks or unauthorized model alterations which are likely to occur.

1.3 Objectives

This thesis aims to address the privacy-preserving machine learning problem for health institutions using the COVID-19 case, specifically COVID-19 detection.

We implemented encryption to model weights in a collaborative machine learning setting and observed the model performance in different hyperparameter settings.

1.4 Approach and Contributions

In this thesis, we implemented homomorphic encryption to protect the privacy of the collaborative machine learning (commonly known as Federated Learning) framework.

Our contribution is to give proof of concept implementation of homomorphic encryption application in federated learning framework using Pyfhel Python library.

We also presented and observed model performance in several hyperparameter settings to conclude its relation with model performance and execution time.

1.5 Outline

- Chapter 2 - Background: This chapter explains the introduction and explanation of each framework or concept implemented in this thesis and existing related works.
- Chapter 3 - Solution Approach: This chapter explains the proposed model and implementation approach.
- Chapter 4 - Experimental Evaluation: This chapter explains the experimentation setup and results.
- Chapter 5 - Environmental Accounts: This chapter describes an overview of the environmental impact of the proposed solution.
- Chapter 6 - Discussion: This chapter further discusses and summarizes the experimentation results.
- Chapter 7 - Conclusion and Future Directions: This chapter explains the conclusion drawn based on experimentation results and possible future works.

Chapter 2

Background

2.1 Homomorphic encryption

Nowadays, data encryption is a common practice not only used by governments and enterprises but also by individuals to protect data privacy. Data encryption is typically applied at rest when the data is stored and in transit when the data is transferred, and both encryption methods require decryption in computations.

Coming from word homomorphism, which is "a transformation of one set into another that preserves in the second set the relations between elements of the first." according Oxford dictionary, homomorphic encryption is an encryption method that allows arithmetical computations to be performed directly on the encrypted or ciphered text, without requiring any decryption. The computations' output is in encrypted form and is an (almost) identical result when decrypted. Homomorphic encryption allows data processing without disclosing the actual data. Homomorphism offers a function that maps between plaintext and ciphertext spaces and preserves the operation between these two.

If Enc denotes encryption, Dec denotes decryption, and $f()$ is a function applied on actual values (plaintexts) a and b, using encrpytion key pk , then homomorphic encryption property would be:

$$f(a, b) = Dec(Enc(pk, a), Enc(pk, b))$$

The additive and multiplicative operations in homomorphic encryption correspond to the binary circuits made from AND and XOR gates of a circuit. This is due to the fact that these two gates behave respectively as operations of binary multiplication and addition, and in addition to that, any functions between sets of integers can be computed by using binary representations of numbers and multiple circuits of the gates [1].

For example [2], operation AND can be represented as two bits multiplication:

$$AND(b_1, b_2) = b_1\dot{b}_2$$

and operation XOR as addition of two bits modulo 2:

$$XOR(b_1, b_2) = (b_1 + b_2) \mod 2$$

2.2 Types of Homomorphic encryption

There are several types of homomorphic encryption [3, 4];

1. Partially homomorphic encryption (PHE) is homomorphic encryption that supports only one homomorphic operation, either addition or multiplication, with an unlimited number of times.
2. Somewhat homomorphic encryption (SHE) schemes allow both addition and multiplication, for only a limited number of times or up to a certain level of complexity, a certain number of multiplication and addition operations, before the result becomes impossible to decrypt.
3. Leveled fully homomorphic encryption (LHE) supports the evaluation of arbitrary circuits composed of multiple types of gates of bounded (pre-determined) depth.
4. Fully homomorphic encryption (FHE) supports both addition and multiplication operations for unlimited times.

All existing homomorphic encryption schemes have common behavior: adding a small noise component during encryption. This behavior causes the noise, or error, to grow during homomorphic computation on the ciphertext to the point that it becomes so large that decryption fails. There are several ways, introduced in previous works to minimize the noise: bootstrapping approach and modulus switching. While bootstrapping is good in performance, it's not practical to implement due to its complexity. Modulus switching is an alternative approach to reducing noise generated. The main idea of modulus switching is to scale down the ciphertext by a modulus factor after each multiplication so as to be within an acceptable noise budget.

We describe more about Somewhat Homomorphic Encryption (SHE) in the next sections as we utilized SHE in our implementation.

2.2.1 Somewhat Homomorphic Encryption (SHE)

SHE is one type of homomorphic encryption which offers both addition and multiplication operations on ciphertexts. However, it is limited to functions that are not too complicated, functions with low-degree polynomials. That said, SHE is preferable nowadays since it is much faster than FHE, and in most cases, SHE is sufficient to accommodate homomorphic calculations.

The following explains briefly on the SHE functions, taken from existing article [2]. SHE procedure starts with determining homomorphic parameters. Several parameters are required such as λ,$N = \lambda$, $P = \lambda^2$, and $Q = \lambda^5$ .Generally SHE has several functions which are :

- KeyGen: This function generates key p, which is a random P-bit odd integer with security parameter λ as input.
- Encryption: Takes as input the key p and a message(ciphertext) which is encoded as bit $m \in 0, 1$. To encrypt the bit m:
  - Choose m' as a random N-bit number, such that $m' = m \mod 2$
  - Choose q to be a random Q-bit number

  output of the function is ciphertext $c = m' + pq$, with m' is the noise that mask the plaintext m.
- Decryption: Input of this function is ciphertext c and key p. The output of the function is decrypted ciphertext $m_f = (c \mod p) \mod 2$. Here, $c \mod p$ is in the range $(-p/2, p/2)$ with the condition that p divides $c - (c \mod p)$ with no remainder.
- Evaluation: Input of this function is f and a set of ciphertexts $Sc = \{c_1, ..., c_n\}$. Function f is represented as a circuit C made of XOR and AND gates, which are then replaced by addition and multiplication gates over integers. The function f is then transformed as a multivariate polynomial which corresponds to the gates. The output is $c = f'(c1, ..., cn)$, or the computation of the ciphertexts through the function.

The evaluation function shows homomorphic behaviour since there is a map (or function) between the plaintext and ciphertext spaces that preserves the operations in these two spaces.

Homormorphic circuit depth is limited in SHE since the noise grows after each operation and eventually the absolute value of the noise will be large until decryption error occurs.

2.2.1.1 Brakerski-Fan-Vercauteren (BFV) scheme

Brakerski-Fan-Vercauteren (BFV) [5] scheme is a well-known homomorphic encryption scheme. It encrypts polynomials instead of bits where the encrypted polynomials can be evaluated homomorphically.

BFV differentiates plaintext space and ciphertext space as two polynomial rings with the same polynomial modulus n, but different coefficient modulus (t, q), where t is the coefficient modulus of plaintext and q is the coefficient modulus of ciphertext. The polynomial ring notation can be seen as a set of polynomials with integer coefficients modulo both t or q and $(x^n + 1)$.

As SHE scheme, BFV has similar functions used in encryption process such as key generation, encryption, decryption, evaluation function. Details on BFV functions relevant for this thesis can be found in Appendix A.

2.3 Federated Learning

Federated Learning (FL) is a machine learning technique that enables multiple parties (clients) to build and train a common machine learning model without exchanging or sharing data. By its definition, federated learning is a subset of Multi-Party Computation (MPC) protocol.

Federated learning process is described as follows:

- Select clients which participate in learning process.
- Each client obtains current global model (weights).
- Each client trains and develops local version of global model.
- Local versions are aggregated and global model is updated.
- The updated global model is distributed back to participating clients.

Federated learning addresses data security and privacy issues since it doesn't require access to the dataset of each client, nor does it require the dataset to be distributed. The local dataset itself doesn't have to be identically distributed and can be heterogeneous. This behaviour makes Federated Learning more popular in healthcare applications. Federated learning enables health institutions to form and train a common model without transferring sensitive patient data out.

There are several topologies in federated learning [6]:

1. Centralized: in this setting, a central server is used to collect, aggregate, and distribute models from participating clients during learning process.

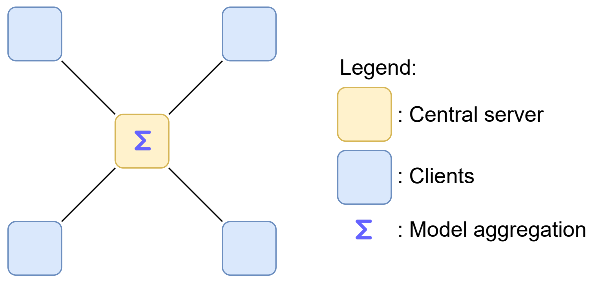

Figure 2.1: Centralized federated learning

2. Decentralized: in this setting, participating clients are connected to one or more peers. The clients coordinate among themselves to obtain a global common model in parallel.

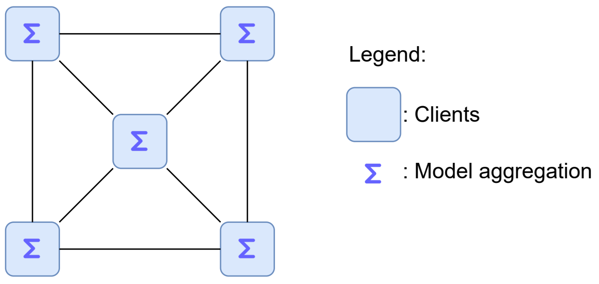

Figure 2.2: Decentralized federated learning

3. Hierarchical: federated networks consists of several sub-federations, which can be a mix of Peer to Peer and Aggregation Server federations.

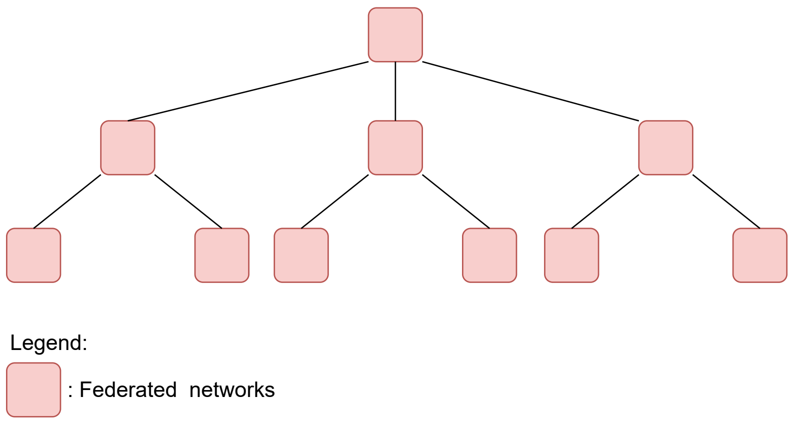

Figure 2.3: Hierarchical federated learning

4. Hybrid hierarchical: federated networks consists of a mix of hierarchical federated network and directly connected clients.

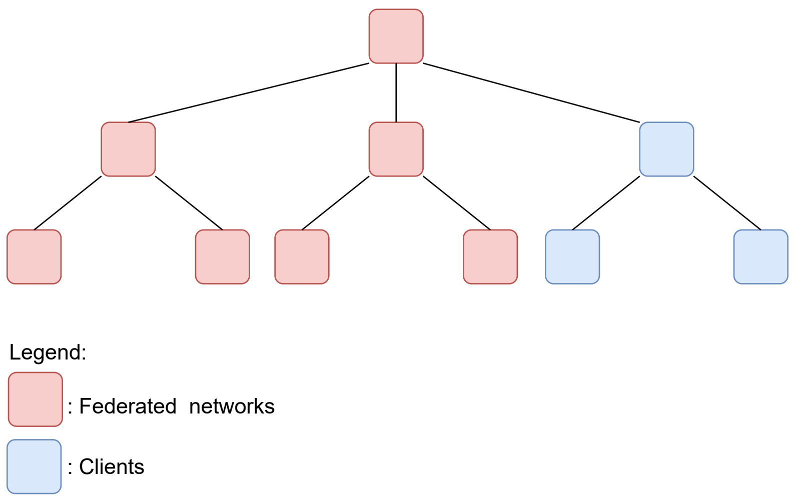

Figure 2.4: Hybrid Hierarchical federated learning

In federated learning models are aggregated by either central server or clients(training nodes). There are several known model aggregation approaches [7]:

1. Federated Averaging (FedAvg) In FedAvg approach, global model is constructed by averaging weights received from participating clients.
2. Federated Personalization (FedPer) In FedPer approach, local models are split into base and personalized layers. Only base layers are sent to server for aggregation, while personalized layers are not.
3. Federated Match Averaging (FedMA) In FedMA, common global model is constructed by matching and averaging hidden elements by layers.

2.4 Image classification with Neural Network

By the time this report is written, most COVID-19 detection machine learning models are using X-ray imaging and defined in neural network, specifically Convolutional Neural Network (CNN).

The main idea behind CNN is to reduce image features. The image pixels are read as an array of pixels ranging from 0 to 255, representing color intensity. This array represents the image features. The features are then reduced by retaining the most important features, focusing on the small parts of the image or pixels which are most distinct.

Below are neural network layers forming CNN architecture:

2.4.1 Convolution

The model architecture starts with convolutional layer which is the most essential component. This is where image pixels are reduced by keeping only important features of the image. This is done through convolution operations which divide the image pixels into small pieces (feature maps) and apply mathematical operation, in order to obtain the critical or essential elements or pixels within each piece.

2.4.2 Non Linearity

The output of the convolution operation, is then passed to an activation function to allow non-linearity. ReLu (Rectified Linear Unit) is most used activation function which replaces negative values from a feature map with zero.

2.4.3 Pooling

The pooling layer is meant to reduce the dimension of input image and therefore reduce complexity of computation and control overfitting.

2.4.4 Fully Connected Layers

Fully connected layers is the last part of CNN architecture where every neuron in one layer is connected to a neuron in another (next) layer. Each neuron computes an output value by applying a function to the input values, which is determined by vector of weights. In model training, these weights are continuously adjusted.

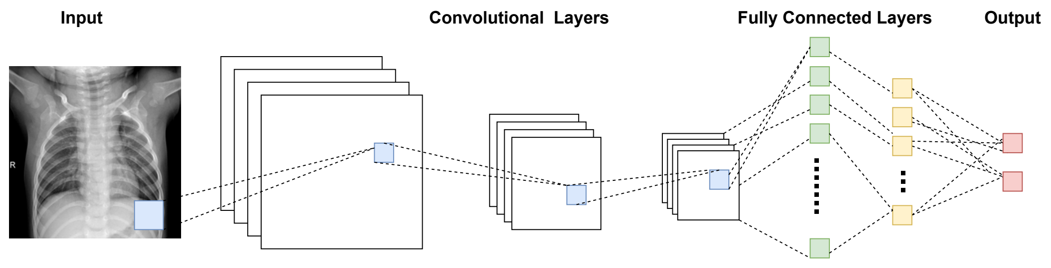

Figure 2.5: CNN Architecture

2.5 Serialization

Serialization is the process of converting an object into a stream of bytes to a format that can be stored. Its primary purpose is to save the state of an object in order to be able to recreate it when needed. In python, serialization can be done by using Pickle library and its file extension is .pickle.

2.6 Related Works

2.6.1 Federated Learning

Data-driven ML models provide unprecedented opportunities for healthcare with the use of sensitive health data. These models are trained locally to protect sensitive health data. However, it is difficult to build robust models without diverse and large datasets which themselves contain the full spectrum of health concerns. Prior proposed works to overcome these problems include federated learning techniques. For instance, the studies [6, 8, 9] reviewed the current applications and technical considerations of the federated learning technique to preserve the sensitive biomedical data. The impact of federated learning is examined by stakeholders such as patients, clinicians, healthcare facilities, and manufacturers. In another study, the authors in [10] utilized federated learning systems for brain tumor segmentation on the BraTS dataset, which consists of magnetic resonance imaging brain scans. The results show that privacy protection correlates to a decrease in performance. The same BraTS dataset is used in [11] to compare three collaborative training techniques, i.e., federated learning, incremental institutional learning (IIL), and cyclic institutional learning (CIIL). In IIL and CIIL, institutions train a shared model successively, whereas CIIL adds a cycling loop through organizations. The results indicate that federated learning achieves similar Dice scores to models trained by sharing data. It outperforms the IIL and CIIL methods since these methods suffer from catastrophic forgetting, where previous learned tasks are lost, as well as complexity.

2.6.2 Homomorphic encryption

Medical data is also protected by encryption techniques such as homomorphic encryption. In [12], authors propose an online secure multiparty computation by sharing patient information to hospitals using homomorphic encryption. Bocu et al. [13] proposed a homomorphic encryption model that is integrated into a personal health information system utilizing heart rate data. The results indicate that the described technique successfully addressed the requirements for secure data processing for the 500 patients with expected storage and network challenges. Another study by Wang et al. [14] proposed a data division scheme based on homomorphic encryption for wireless sensor networks. The results show that there is a trade-off between resources and data security.

In [15], the applicability of homomorphic encryption is shown by measuring the vitals of the patients with a lightweight encryption scheme. Sensor data such as respiration and heart rate are encrypted using homomorphic encryption before transmitting to the non-trusting third party, while encryption takes place only in the medical facility. The study in [16] developed an IoT based architecture with homomorphic encryption to combat data loss and spoofing attacks for chronic disease monitoring. Results suggest that homomorphic encryption provides cost-effective and straightforward protection of sensitive health information. Blockchain technologies are also utilized in cooperation with homomorphic encryption to secure medical data. Authors in [17] proposed a practical pandemic infection tracking using homomorphic encryption and blockchain technologies in intelligent transportation systems using automatic healthcare monitoring. In another study, Ali et al. [18] developed a search-able distributed medical database on a blockchain using homomorphic encryption.

An increased need to secure sensitive information necessarily leads to the use of various techniques together. In the scope of this thesis, a multiparty computation tool using federated learning with homomorphic encryption is developed and analyzed.

Paper	Differential Privacy	Federated learning	Cryptography	Secure multi-party
Xu, Jie and Glicksberg, Benjamin S and Su, Chang and Walker, Peter and Bian, Jiang and Wang, Fei [8]		X		
Rieke et. al [6]		X		
Antunes, Rodolfo Stoffel and da Costa, Cristiano André and Küderle, Arne and Yari, Imrana Abdullahi and Eskofier, Björn [9]		X		
Li, Wenqi and Milletarì, Fausto and Xu, Daguang and Rieke, Nicola and Hancox, Jonny and Zhu, Wentao and Baust, Maximilian and Cheng, Yan and Ourselin, Sébastien and Cardoso, M Jorge and others [10]	X			
Sheller, Micah J and Reina, G Anthony and Edwards, Brandon and Martin, Jason and Bakas, Spyridon [11]		X		
Kumar, A Vijaya and Sujith, Mogalapalli Sai and Sai, Kosuri Tarun and Rajesh, Galla and Yashwanth, Devulapalli Jagannadha Sriram [12]				X
Bocu, Razvan and Costache, Cosmin [13]			X	
Wang, Xiaoni and Zhang, Zhenjiang [14]			X	
Kara, Mostefa and Laouid, Abdelkader and Yagoub, Mohammed Amine and Euler, Reinhardt and Medileh, Saci and Hammoudeh, Mohammad and Eleyan, Amna and Bounceur, Ahcène [15]			X	
Talpur, Mir Sajjad Hussain and Bhuiyan, Md Zakirul Alam and Wang, Guojun [16]			X	
Tan, Haowen and Kim, Pankoo and Chung, Ilyong [17]			X	
Ali, Aitizaz and Pasha, Muhammad Fermi and Ali, Jehad and Fang, Ong Huey and Masud, Mehedi and Jurcut, Anca Delia and Alzain, Mohammed A [18]			X	

Table 2.1: Summary of related works

Chapter 3

Solution Approach

3.1 Proposed Model

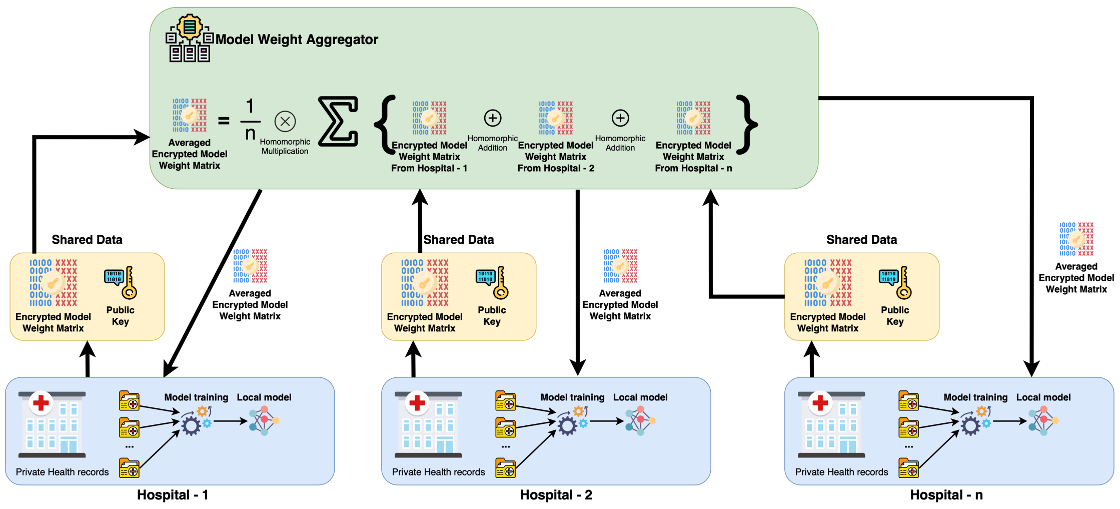
Model Weight Aggregator
Averaged Encrypted Model Weight Matrix
= 1/n
Homomorphic Multiplication
Encrypted Model Weight Matrix From Hospital - 1
Homomorphic Addition
Encrypted Model Weight Matrix From Hospital - 2
Homomorphic Addition
Encrypted Model Weight Matrix From Hospital - n
Shared Data
Encrypted Model Weight Matrix
Public Key
Averaged Encrypted Model Weight Matrix
Model training
Local model
Private Health records
Hospital - 1
Hospital - 2
Hospital - n

Figure 3.1: Overall Model

In summary, the proposed model applies a centralized federated learning framework with an encrypted aggregation process. Figure 3.1 above describes the overall proposed model implemented in this thesis. Each client holds a public and shared key pair which is used to encrypt and decrypt the model. Firstly, the global model is initialized and distributed to all participating clients. Clients then train their copy of global model with their own local dataset. To update the global model, the local model weights are encrypted using homomorphic encryption and exported, as a serialized files, back to a central server. The central server imports the serialized files received from participating clients, aggregates the encrypted weights and updates the global model. The updated global model is then re-distributed back to the clients in a serialized file which the client decrypts using the private key.

This thesis proposed a model which utilizes Secure Multi-party Computation (SMPC) protocol which distributes the machine learning process across multiple parties with a centralized federated learning framework, a subset of Multi-party Computation (MPC), without any data sharing or distribution.

3.1.1 Key Pair Generation

At the initial stage, parameters used to generate key pairs were determined. In this thesis, parameters observed are security level and polynomial modulus. 2.2.1.1The security level is security level equivalent in AES-bits, 128 or 192 (AES key size), while the polynomial modulus determines both security and performance of the encryption process. The larger security level increases security but also increases cost. The larger polynomial modulus increases the noise ceiling without decreasing the security level [19].

In this thesis, all clients have the same security parameter values. Public and secret key pairs are then generated by utilizing these parameters. As a result, all clients use the same key pairs.

3.1.2 Client Initialization

The process started with machine learning model initialization in all participating clients by initializing the image classification model instance in each client. At this point, all clients obtained the initial model architecture. The model is untrained and consists of layers upon layers defined in the CNN model.

3.1.3 Model Training

Sequence diagram 3.2 describes on detail implementation steps in training at clients:

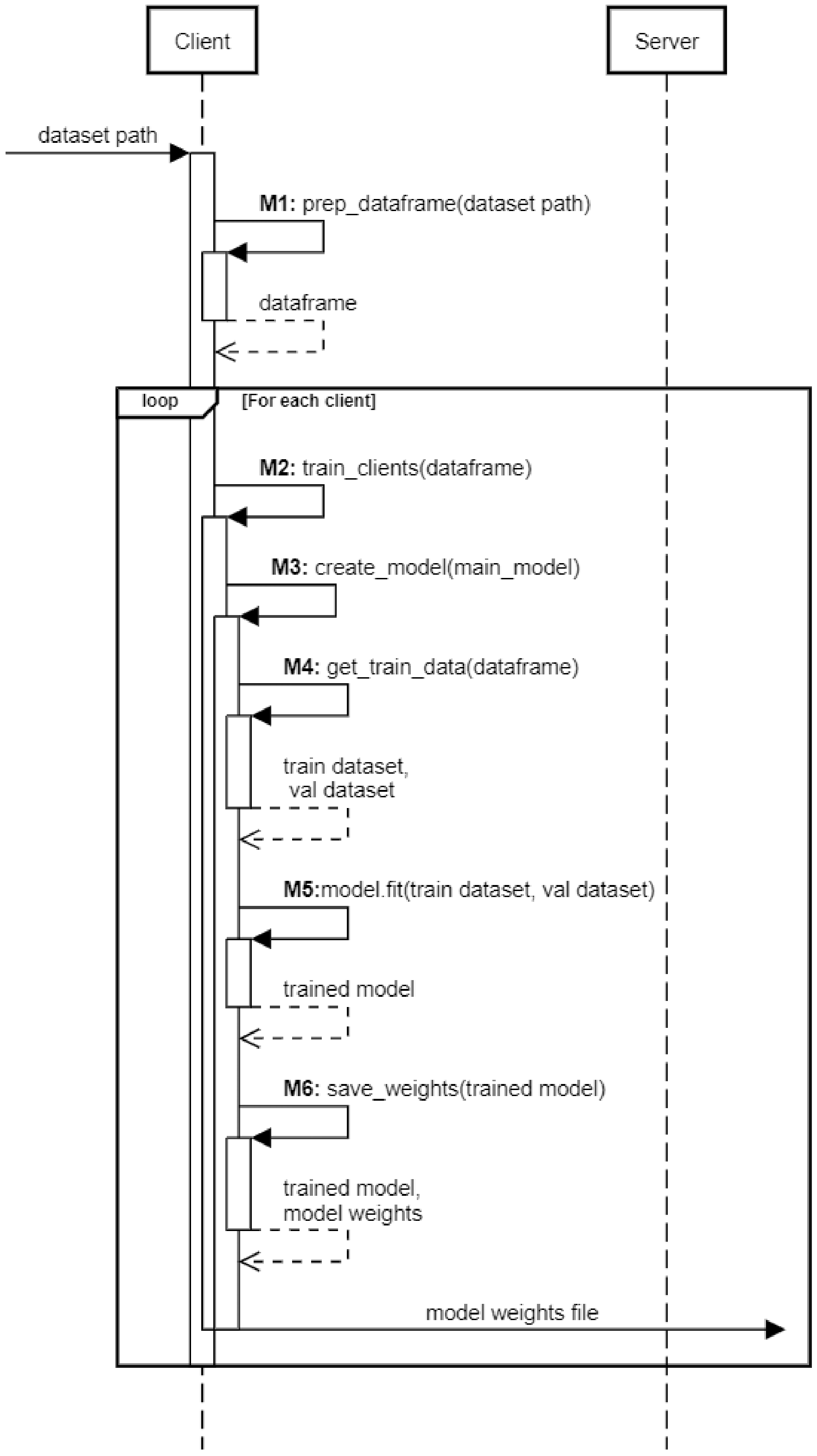
Client
Server
dataset path
M1: prep_dataframe(dataset path)
dataframe
loop
[For each client]
M2: train_clients(dataframe)
M3: create_model(main_model)
M4: get_train_data(dataframe)
train dataset,
val dataset
M5:model.fit(train dataset, val dataset)
trained model
M6: save_weights(trained model)
trained model,
model weights
model weights file

Figure 3.2: Sequence Diagram: Training at Clients

The implementation starts with training and validation preparation. X-ray image files used for training are placed in a specific file directory. The directory is declared and defined in the implementation, and the X-ray image files are then read and stored in a dataframe. This is performed by *prep_dataframe* (**M1**) function.

The main function of this process is *train_clients*(**M2**) which contains nested sub-functions. Function *get_train_data* (**M4**) is the first function to run. This function splits the dataframe imported evenly based on the number of participating clients. A dataframe subset is allocated to a different client. The dataframe subsets are then transformed into training and validation datasets. Data preprocessing is also performed in this function.

Below pseudocode1 describes the algorithm defined in *prep_dataframe* (**M1**) function.

Algorithm 1: Training Dataset Preparation

```
Input: Training_dataset_path, num_client
1  $training\_file\_path \leftarrow [listdir(Training\_dataset\_path)]$ ; // Get all files in path
2  $label \leftarrow [subfolder.name]$ ;            // Get subfolder name as class label
3  $Training\_dataframe \leftarrow [training\_file\_path, label]$
   $Training\_dataframe.sample()$ ;                  // Shuffle dataframe
4  foreach client do
5      $index \leftarrow 0$ ;                          // initialize index as 0
6      $subset \leftarrow int(len(df\_train.index)/num\_client)$ ;       // get number of
        training records for each client
7      $image\_gen \leftarrow ImageDataGenerator(rescale = 1./255, validation\_split = 0.1)$
       // setup image generator
8      $start\_index \leftarrow index * subset$ ;             // get start index
9      $end\_index \leftarrow start + ratio$ ;                // get end index
10     $Training\_dataframe[index] \leftarrow Training\_dataframe[start : end]$
       // slice dataframe for a client
11     $X_{train}[index] \leftarrow$
        $image\_gen.flow\_from\_dataframe(Training\_dataframe[index], shuffle =' True', subset =' training')$
       // generate training dataset
12     $X_{validation}[index] \leftarrow$
        $image\_gen.flow\_from\_dataframe(Training\_dataframe[index], shuffle =' True', subset =' validation')$
       // generate validation dataset
13     $index \leftarrow index + 1$ // increase index
14 end
15 return $X_{train}$,$X_{validation}$
```

The basic idea of the algorithm is quite standard, just like an array slicing algorithm. The process is done in iteration, from the first client to the nth client. *subset* is a variable that is obtained by dividing the number of dataset records by the number of clients. The first client gets the first subset of dataframe, starting from index zero to the next *subset* elements. The start index of the next client is then set as *subset* multiplied by the index of the current iteration. In this case, a dataframe subset will not be allocated twice by doing so.

A dataframe subset contains a list of image files to be used in model training. Each image file found in the subset is imported as an image pixel array and data preprocessing is performed to the array. At this point, the training and validation datasets are generated and ready for usage.

The implementation then continues with the training process. The model training starts with initializing and replicating the global model to all clients. All clients obtained the same model version. This is performed by function *create_model* (**M3**).

The training process uses the training dataset as input, while the training validation process uses the validation dataset as input. Each client trains and obtains a local version of the global model after the training (**M5**: *model_fit*). At the end of the training, only the best weights, weights that produced the highest training accuracy, are stored and exported to an external file(**M6**: *save_weights*). Model weights are exported merely to track weight changes and reduce repeating training steps when troubleshooting.

3.1.4 Weight Encryption

Somewhat Homomorphic Encryption (SHE) was implemented to encrypt model weights at clients. This is due to the fact that SHE allows simple addition and multiplication operations on encrypted data, which were used in calculating average weights. Weights are encrypted at clients by reading each weight element in CNN layers and applying the encryption method using the public key.

Sequence diagram 3.3 below describes implementation of weight encryption at clients.

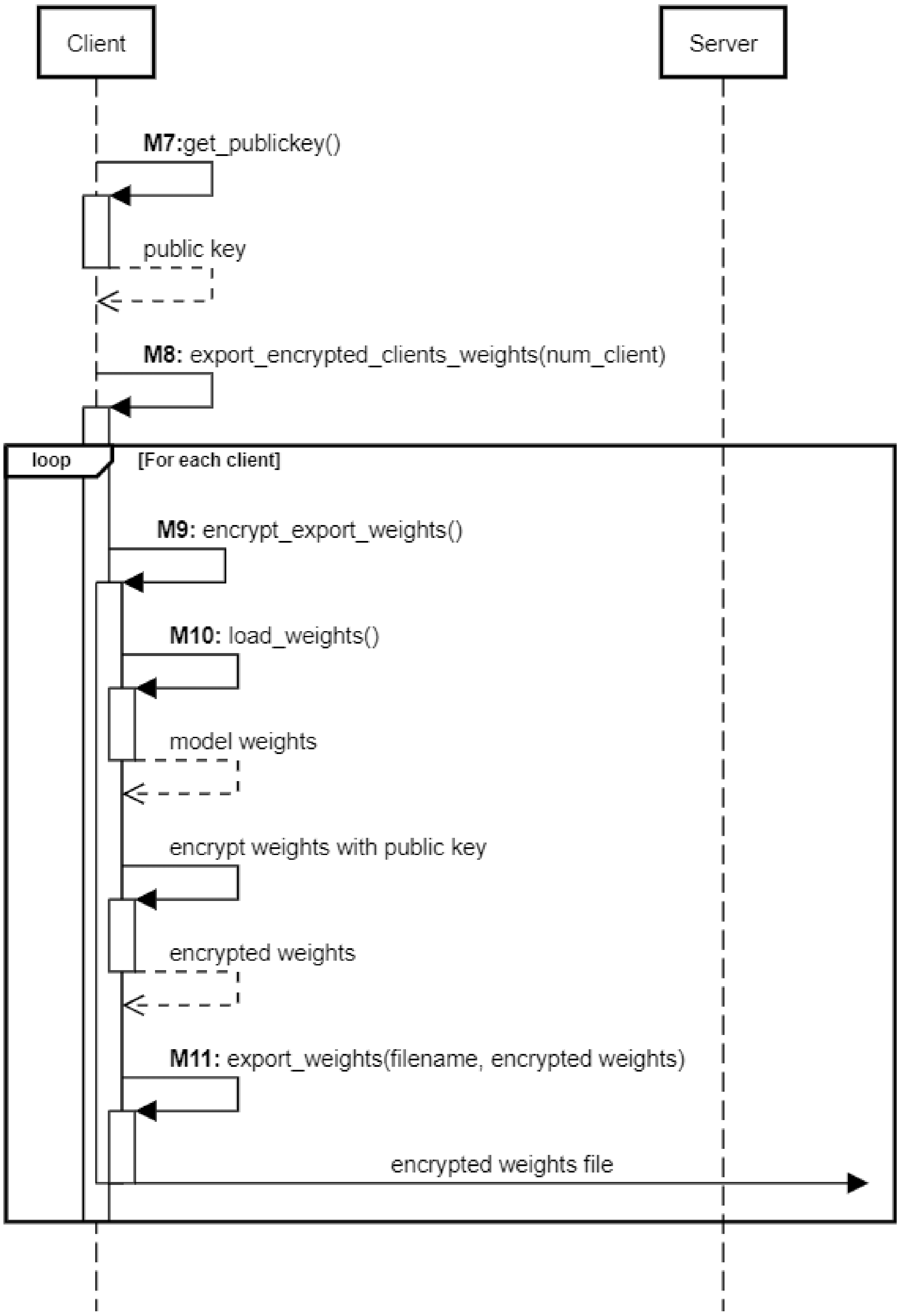
Client
Server
M7:get_publickey()
public key
M8: export_encrypted_clients_weights(num_client)
loop
[For each client]
M9: encrypt_export_weights()
M10: load_weights()
model weights
encrypt weights with public key
encrypted weights
M11: export_weights(filename, encrypted weights)
encrypted weights file

Figure 3.3: Sequence Diagram: Weight Encryption at Clients

The weight encryption process starts with obtaining the public key used for encryption (**M7**:*get_publickey*). In the main method (**M8**:*export_encrypted_clients_weights*), the external file which contains model weights is imported back into the model(**M10**:*load_weights*) and then encrypted using the public key. The weights are then serialized and exported as a serialized file (**M11**:*export_weights*). The serialization is required since encrypted weights are objects with a specific type, ciphertext, and the object type must be retained upon import.

The algorithm 2 below summarizes the process of training (3.1.3) and model weights encryption at clients (3.1.4) discussed previously.

Algorithm 2: Model training in each client

```
Input: The dataset at client $c$: $\mathcal{D}_c = \{(\mathbf{x}, y) | \mathbf{x} \in \mathbb{R}^m, y \in \mathbb{R}\}_{i=0}^m, publickey : Key_{pub}$
1 $X_{train}, X_{test}, \mathbf{y}_{train}, \mathbf{y}_{test} \leftarrow train\_test\_split(\mathcal{D})$ ;
2 $h \leftarrow global\_model$ ;
3 $h.fit(X_{train}, \mathbf{y}_{train})$ ;
4 $W \leftarrow \emptyset$ ;      // Create an empty matrix for the encrypted layer weights
5 foreach $layer \in h$ do
6     $[\![W]\!] \leftarrow encrypt_fraction(layer.weights, key_{pub})$ ;      // Encrypt the layer
      weights ($layer.weights \in \mathbb{R}^m$) with public key
7 end
8 return $[\![W]\!]$ ;                              // The encrypted weight matrix
```

The pseudocode describes the process as follows, let variable h be the initial global CNN model and variable Key_{pub} be the generated public key for encryption. The training dataset is split evenly based on the number of clients. It is then used as input in training the global CNN model h, and a local version of global model was obtained. Weights of the model then are encrypted using the public key by applying encryption ($encrypt_fraction$) method to each model weight in each CNN layer. The ($encrypt_fraction$) method is used to encrypt float (decimals) numbers, since data type of model weights is float.

The export and import encrypted weights to serialized files are performed to simulate the model weights transfer between clients and server.

3.1.5 Model Aggregation

Centralized federated learning is implemented in this thesis. In this case, model weights are transferred to a centralized server. The server aggregates the model weights and then updates the global model before finally sending the updated global model back to clients. The model aggregation in the central server is performed with Federated Averaging (FedAvg) algorithm. With FedAvg, the global model is updated by computing the average model weights received from clients.

Algorithm 3: Model aggregation at the server

```
Input: public key: $Key_{pub}$, the number of clients: $c$, client model weights:
       $H = \{[\![W]\!]_i\}_{i=0}^{c} [\![W]\!]_{aggr} \leftarrow \emptyset$
1 foreach $h \in H$ do
2     foreach $[\![row]\!] \in h$ do
3         $[\![W]\!]_{aggr} \leftarrow [\![W]\!]_{aggr} \oplus [\![row]\!]$ ;        // Homomorphic addition
4     end
5 end
6 foreach $[\![row]\!] \in [\![W]\!]_{aggr}$ do
7     $[\![row]\!] \leftarrow [\![row]\!] \otimes c^{-1}$ ;        // Homomorphic multiplication
8 end
9 return $[\![W]\!]_{aggr}$ ;        // Return the aggregated weight matrix in the
   encrypted domain
```

Algorithm 3 above describes the model weights aggregation function. The function reads the imported weight elements iteratively by the number of clients and layers. It then calculates the average of model weights by calculating the sum of weights and multiplying by 1/(number of clients) instead of dividing by the number of clients. The multiplication is performed in this implementation as multiplication, and not division, because it is supported by homomorphic encryption. The addition and multiplication are done homomorphically, and no decryption is required.

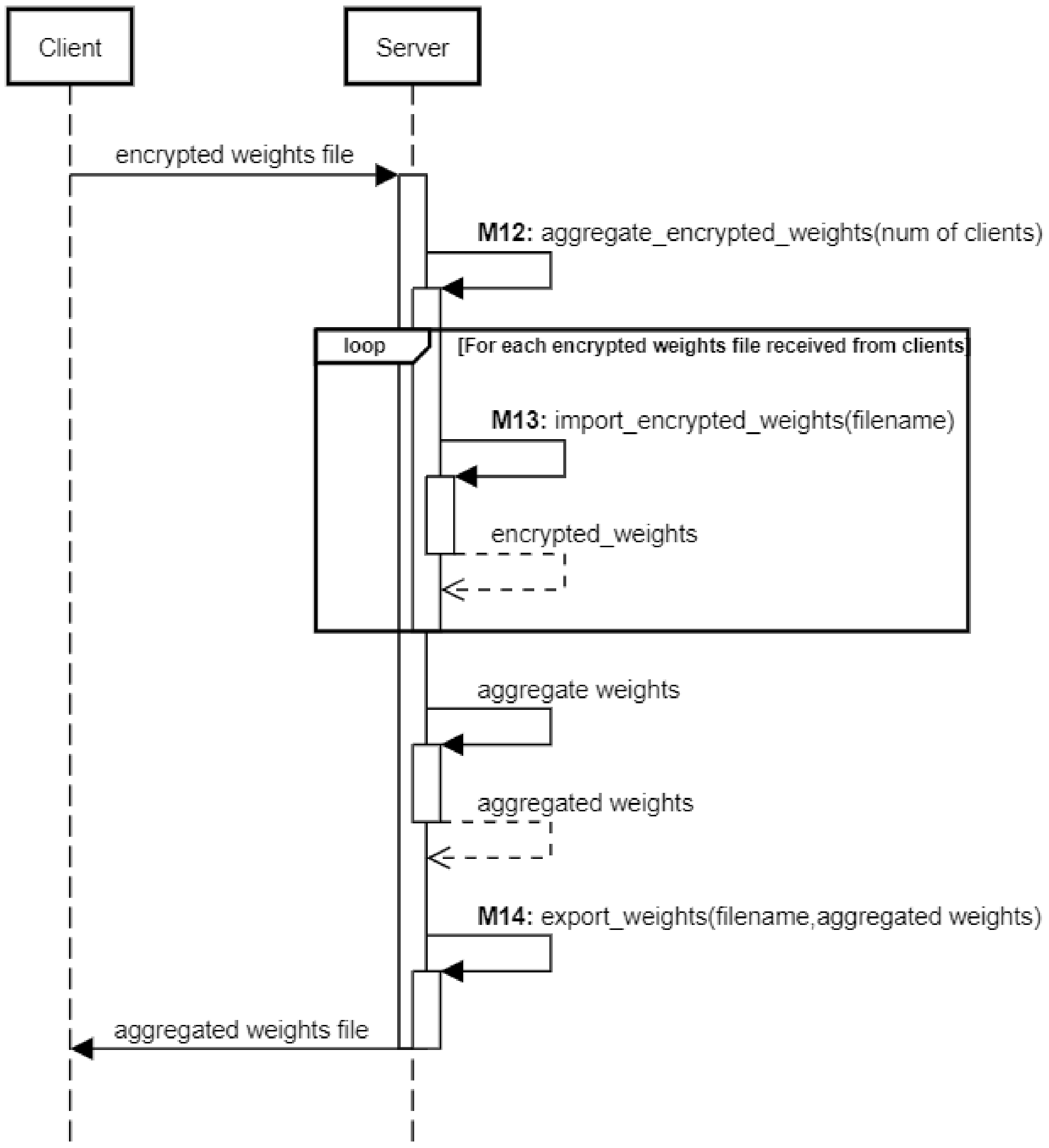
Client
Server
encrypted weights file
M12: aggregate_encrypted_weights(num of clients)
loop
[For each encrypted weights file received from clients]
M13: import_encrypted_weights(filename)
encrypted_weights
aggregate weights
aggregated weights
M14: export_weights(filename,aggregated weights)
aggregated weights file

Figure 3.4: Sequence diagram: Weights Aggregation at Server

The sequence diagram 3.4 describes the implementation of weights aggregation at central server.

The weight aggregation process starts with clients sending encrypted weights serialized files to the central server. The central server aggregates the weights received from all clients by calling the main method *aggregate_encrypted_weights* (**M12**). This function aggregates model weights according to algorithm 3. There are sub-functions called in this method. Sub-function *import_encrypted_weights* (**M13**) imports the encrypted weights serialized files. The weights are then aggregated by the main function and finally exported again as serialized file and sent back to clients by calling sub-function *export_weights*(**M14**).

3.1.6 Weight Decryption

Similar to weight encryption, weight decryption at clients is done by applying the decryption method on weights elements in each layer but using the secret key.

Upon receiving aggregated weights file from the server, each client imports and decrypts the aggregated weights received from the server before finally applying the global model. In this stage, all clients are aligned with the global model.

Algorithm 4 below describes the decryption process at clients.

Algorithm 4: Client decryption

```
Input: private key: $Key_{priv}$, encrypted aggregated weights: $[\![W]\!]_{aggr}$
1 $h \leftarrow global\_model$ ;
2 foreach $layer \in h$ do
3     $[\![row]\!] \leftarrow [\![W]\!]_{aggr}(layer)$ ;          // Get the corresponding row for layer
4     $layer \leftarrow decrypt_fraction([\![row]\!], key_{priv})$ ;    // Decrypt the row and update
        the layer weights
5 end
6 $h.save\_model(global\_model)$ ;              // Save the aggregated model as
  global_model at client
```

Decryption process is performed by applying decryption ($decrypt_fraction$) method to each model weight, after the encrypted weights are imported from serialized file. The ($decrypt_fraction$) method is used to encrypt float (decimals) numbers, since data type of model weights are float.

Sequence diagram 3.5 below describes implementation done to decrypt weights at clients.

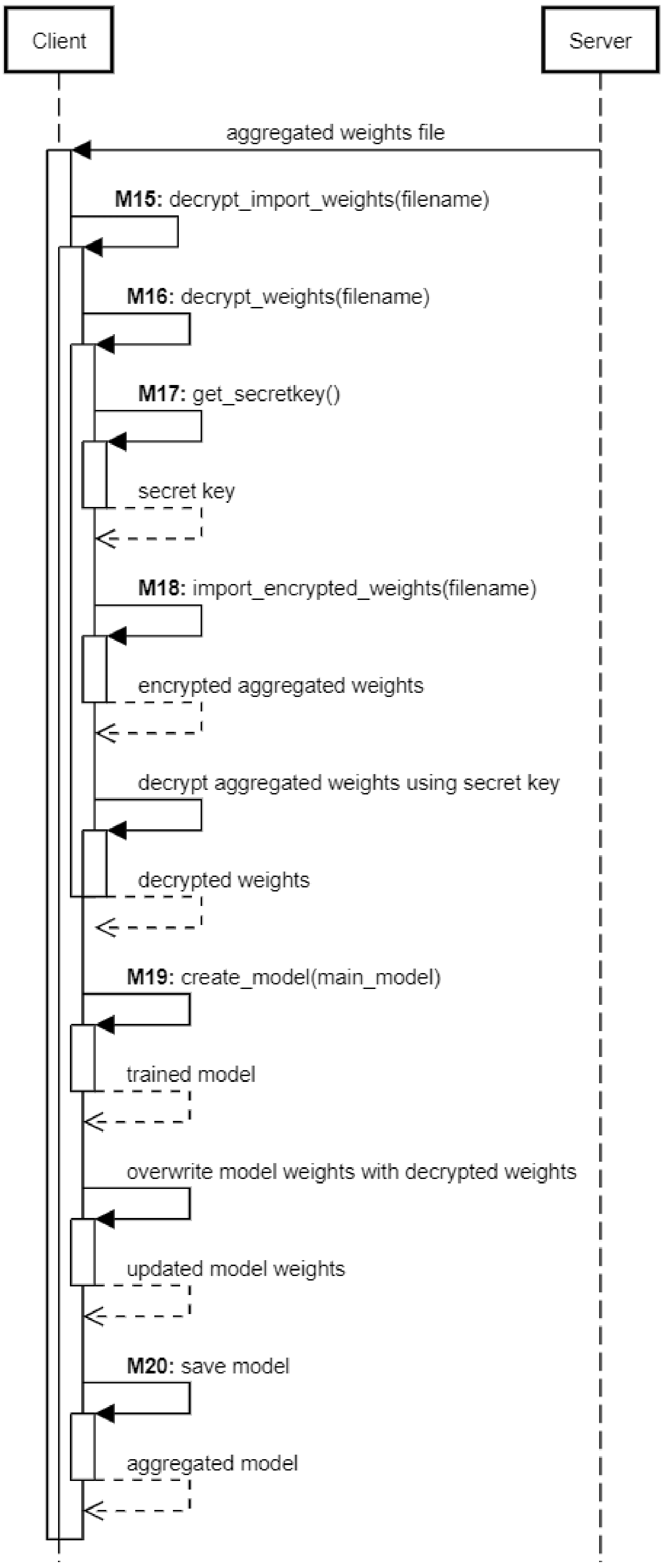
Client
Server
aggregated weights file
M15: decrypt_import_weights(filename)
M16: decrypt_weights(filename)
M17: get_secretkey()
secret key
M18: import_encrypted_weights(filename)
encrypted aggregated weights
decrypt aggregated weights using secret key
decrypted weights
M19: create_model(main_model)
trained model
overwrite model weights with decrypted weights
updated model weights
M20: save model
aggregated model

Figure 3.5: Import and Decryption at Clients

The main method *decrypt_import_weights*(**M15**), consists of several sub-functions. Sub-function *get_secretkey*(**M17**) obtains the secret key. The aggregated weights are imported by calling sub-function *import_ecnrypted_weights*(**M18**), which then decrypted by method *decrypt_weights*(**M16**). At the end, Sub-function *save_model* (**M20**) imported the decrypted weights to the local model.

In this thesis, aggregated model weights were decrypted for a client, at which prediction took place and evaluation metrics were generated.

3.1.7 Evaluation Metrics Generation

Evaluation metrics is generated and executed after the model prediction is run for a client.

Chapter 4

Experimental Evaluation

4.1 Experimental Setup and Data Set

4.1.1 Dataset

In this thesis, the COVID-19 Radiography dataset from previous works [20, 21] was used as training, validation and test dataset. The image file size is 299 x 299 pixels when then resized to 255 X 255 pixels.The dataset contains X-ray lung images with four different classifications: COVID, Lung opacity, Normal, and Viral Pneumonia. However, we solely utilized two of them, which are COVID and Normal X-ray images, focusing on the COVID-19 detection machine learning process.

Below figures show samples of the dataset.

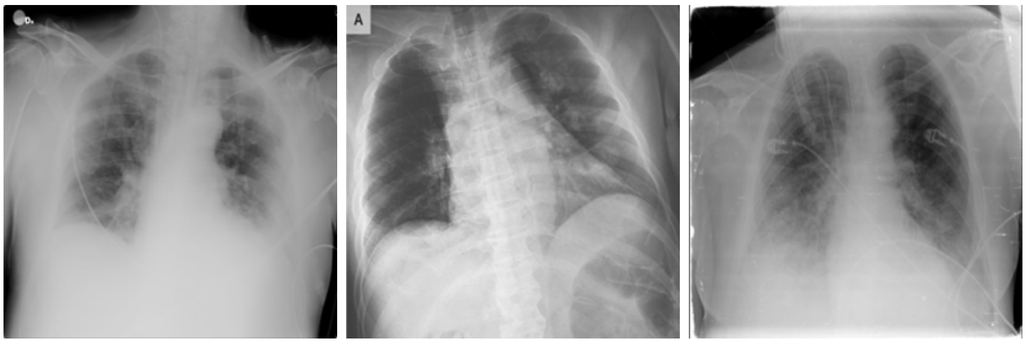

Figure 4.1: COVID-19 positive X-ray images

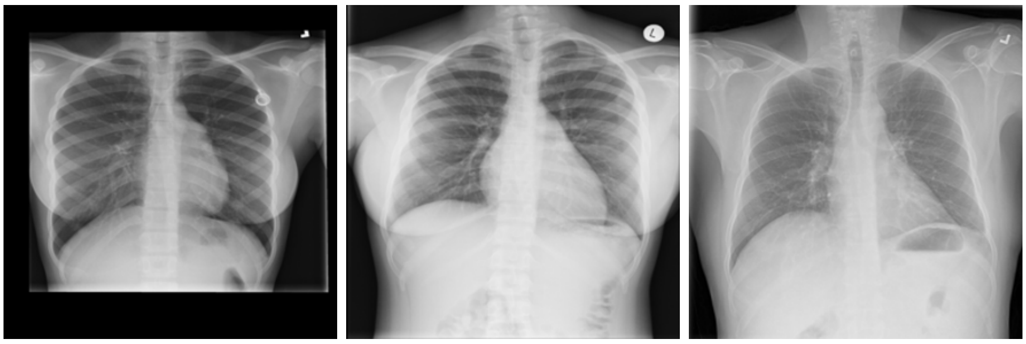

Figure 4.2: Normal X-ray images

The X-ray lung images were placed into a folder directory which was divided into the training dataset folder and test dataset folder. Below figure 4.3 shows dataset folder structure.

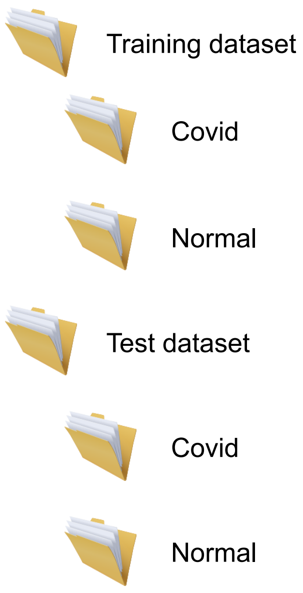

Figure 4.3: Dataset folder structure

Such folder structure was required since we used the standard image preprocessing method in Keras *ImageDataGenerator,flow_from_dataframe*). We obtained the first 1000 records from the original dataset for each classification, with 80% of the sample used for the training dataset (800 records) and the remaining 20% as the test dataset (200 records). The training dataset was further split with 10% of the dataset as the training-validation dataset.

Table 4.1 shows the dataset segregation.

Dataset	**Rows**	**Label**
Training, Validation	800	Negative
	800	Positive
Test	200	Negative
	200	Positive

Table 4.1: Dataset description

4.1.2 Preprocessing

Data preprocessing performed in this implementation consisted of data augmentation and rescaling. Data augmentation was applied to the training dataset to provide additional data variety during training. We applied rescaling to both the training and test dataset by multiplying each pixel value of each X-ray image in the dataset by 1/255, to transform the pixel value range from [0,255] to [0,1]. The purpose of rescaling was to treat all images equally. Scaling every image to the same range [0,1] makes the image contribute evenly to total loss during model training. High pixel range images will have larger votes on determining the weights without scaling.

4.1.3 Experimental Setup

In this thesis, we experimented with various numbers of clients (2, 3, 5, and 7 clients). We run the federated learning process in iteration based on the number of clients and observed the running time and evaluation metrics to analyze model prediction performance in the encrypted federated learning framework.

4.1.4 Microsoft SEAL

Microsoft Simple Encrypted Arithmetic Library (SEAL) is a homomorphic encryption library developed by Microsoft, which was released in 2015. SEAL library implements both Brakerski/Fan-Vercauteren (BFV) [22, 23] and Cheon-Kim-Kim-Song (CKKS) [24] homomorphic encryption schemes and provides standard SHE functions starting from encoding, key generation, encryption, decryption, additive, multiplicative, and relinearization functions. SEAL was developed in C++ and can be deployed on Windows, Linux, macOS, and Android [2].

4.1.5 Pyhthon Libraries

The implementation was developed with Python 3.8.8 and its libraries, specifically Keras, TensorFlow, NumPy, Pyfhel version 2.3.1, time, and sklearn libraries.

We used Keras and TensorFlow libraries for machine learning processes such as in preprocessing, training, importing and exporting the model. Keras and Tensorflow are open-source libraries run in Python. Keras provides image data preprocessing functions such are *ImageDataGenerator* and *flow_from_dataframe* , which was called during the data preprocessing step.

We used Numpy library to perform array operations on model weights and Pickle library to export and import weights as serialized files. We used sklearn, metrics module, to generate evaluation metrics on the prediction result. We used time library to measure the execution time. Most importantly, we used Pyfhel to perform homomorphic encryption. Pyfhel [25] is a Python wrapper for SEAL, which provides the same functionalities as SEAL in Pyhton and requires simpler programming.

4.1.6 Model Infrastructure

Using Keras, we defined CNN infrastructure as shown in below Table 4.2:

Layer (type)	Output	Shape	No. of Parameters
conv2d	(Conv2D)	(None, 254, 254, 32)	896
max_pooling2d	(MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_1	(Conv2D)	(None, 125, 125, 32)	9248
max_pooling2d_1	(MaxPooling 2D)	(None, 62, 62, 32)	0
conv2d_2	(Conv2D)	(None, 60, 60, 32)	9248
max_pooling2d_2	(MaxPooling 2D)	(None, 30, 30, 32)	0
conv2d_3	(Conv2D)	(None, 28, 28, 64)	18496
max_pooling2d_3	(MaxPooling 2D)	(None, 14, 14, 64)	0
conv2d_4	(Conv2D)	(None, 12, 12, 64)	36928
max_pooling2d_4	(MaxPooling 2D)	(None, 6, 6, 64)	0
conv2d_5	(Conv2D)	(None, 4, 4, 128)	73856
max_pooling2d_5	(MaxPooling 2D)	(None, 2, 2, 128)	0
flatten	(Flatten)	(None, 512)	0
dense	(Dense)	(None, 128)	65664
dense_1	(Dense)	(None, 64)	8256
dense_2	(Dense)	(None, 2)	130
Total params: 222,722			
Trainable params: 222,722			
Non-trainable params: 0			

Table 4.2: CNN image classification model summary in Keras

4.1.7 Hyperparameters Settings

In the initialization step of Pyfhel object, we generated a homomorphic encryption context by calling the context generation method, *contextGen*, using several parameters as input. These parameters are required for all homomorphic operations(encryption, decryption, scheme, decoding, mathematical operations).

The first parameter defines which scheme to use in homomorphic encryption. In this case, the BFV(SHE) scheme was selected. We then experimented with two other parameters, security level(sec) and polynomial modulus(m) parameters, and observed the impact of tuning these parameters [1].

At the time of writing, there are two possible values in Pyfhel 2.3.1 library for security level(sec) parameter, which are 128 and 192. Combination of sec and polynomial modulus (m) parameters determine length of coefficient modulus, which is modulus in the ciphertext space (q), as described in below Table 4.3 [19].

Bit-length of default q		
m	128-bit security	192-bit security
1024	27	19
2048	54	37
4096	109	75
8192	218	152
16384	438	300
32768	881	600

Table 4.3: Default pairs (n, q) for 128-bit and 192-bit security levels.

Due to hardware limitations, we could only experiment with m values of 1024 and 2048.

4.2 Evaluation Metrics

Accuracy, Precision, Recall, and F1 score (evaluation metrics) were used to evaluate the model's performance in COVID-19 prediction. We utilized the evaluation metrics to observe the relation between the number of clients participating in encrypted federated learning and its global model prediction performance.

4.2.1 Accuracy

Accuracy measures the percentage of correct predictions by the total number of samples.

$$Accuracy = \frac{Number of correct predictions}{Number of samples}$$

[1]In BFV paper [5], SEAL document [19] and Pyfhel version 3.1.4, polynomial modulus is referred as n

4.2.2 Precision and Recall

Precision indicates the proportion of positive identification that was actually correct or belonged to the target class.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

Recall indicates the proportion of actual positive identification was identified correctly.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

4.2.3 F1 Score

F1 Score combines precision and recalls into a single number using the following formula.

$$F1 = 2 * \frac{(Recall * Precision)}{Recall + Precision}$$

4.2.4 Hardware Specification

We performed the implementation on a laptop with specifications: 11^{th} Gen Intel(R) Core i5-1145G7 @2.60GHz 2.61GHz and 16 GB RAM.

4.3 Experimental Results

4.3.1 Evaluation Metrics

The following are the model prediction performance results, measured with evaluation metrics.

4.3.1.1 Experimentation of plain model

We first experimented COVID-19 detection model with neither encryption nor federated learning implemented. We observed only one client for this case, which would be a typical standalone machine learning performed by a client.

Precision	Recall	F1 Score	Accuracy
0.868924	0.840000	0.836801	0.840000

Table 4.4: Prediction result without Federated Learning

Table 4.4 above shows the evaluation metrics of prediction result of plain model.

4.3.1.2 Experimentation results of plain model with federated learning

We then applied a federated learning framework in iteration with 2,3,5, and 7 number of clients, without applying homomorphic encryption. We observed model prediction performance based on evaluation matrices as showed in the below Table 4.5.

Number of clients	2	3	5	7
Precision	0.872128	0.865112	0.859288	0.850277
Recall	0.845000	0.837500	0.835000	0.827500
F1 Score	0.842123	0.834369	0.832164	0.824649
Accuracy	0.845000	0.837500	0.835000	0.827500

Table 4.5: Performance measurements: Federated learning without homomorphic encryption

4.3.1.3 Experimentation results of encrypted model with federated learning

We continued our experimentation by implementing homomorphic encryption in a federated learning framework.

Experimentation was performed in iteration with 2,3,5, and 7 number of clients and also by adjusting hyperparameters in Pyfhel's context generation method according to section 4.1.7. Hyperparameters values are tuned as below:

- Security level (sec) between values 128 and 192.
- Polynomial modulus (m) between values 1024 and 2048.

Table 4.6, Table 4.7, Table 4.8, and Table 4.9 below are the evaluation metrics generated based on different hyperparameters settings.

Number of clients	2	3	5	7
Precision	0.853925	0.250000	0.250000	0.250000
Recall	0.830000	0.500000	0.500000	0.500000
F1 Score	0.827078	0.333333	0.333333	0.333333
Accuracy	0.830000	0.500000	0.500000	0.500000

Table 4.6: Performance measurements: Federated learning with encryption sec=128, m=1024

4.3.1.3.1 Experimentation results of *sec*=128 and *m*=1024

Number of clients	2	3	5	7
Precision	0.250000	0.250000	0.250000	0.250000
Recall	0.500000	0.500000	0.500000	0.500000
F1 Score	0.333333	0.333333	0.333333	0.333333
Accuracy	0.500000	0.500000	0.500000	0.500000

Table 4.7: Performance measurements: Federated learning with encryption sec=192, m=1024

4.3.1.3.2 Experimentation results of *sec*=192, *m*=1024

Number of clients	2	3	5	7
Precision	0.867337	0.857293	0.853925	0.869584
Recall	0.837500	0.840000	0.830000	0.852500
F1 Score	0.834132	0.838040	0.827078	0.850776
Accuracy	0.837500	0.840000	0.830000	0.852500

Table 4.8: Performance measurements: Federated learning with encryption sec=128, m=2048

4.3.1.3.3 Experimentation results of *sec*=128 and *m*=2048

Number of clients	2	3	5	7
Precision	0.866735	0.868924	0.855624	0.86800
Recall	0.840000	0.840000	0.832500	0.84500
F1 Score	0.837030	0.836801	0.829732	0.84254
Accuracy	0.840000	0.840000	0.832500	0.84500

Table 4.9: Performance measurements: Federated learning with encryption sec=192,m=2048

4.3.1.3.4 Experimentation results of *sec*=192 and *m*=2048 The results are also shown in histograms to provide visual comparison of evaluation metrics which were obtained with different hyperparameter adjustments.

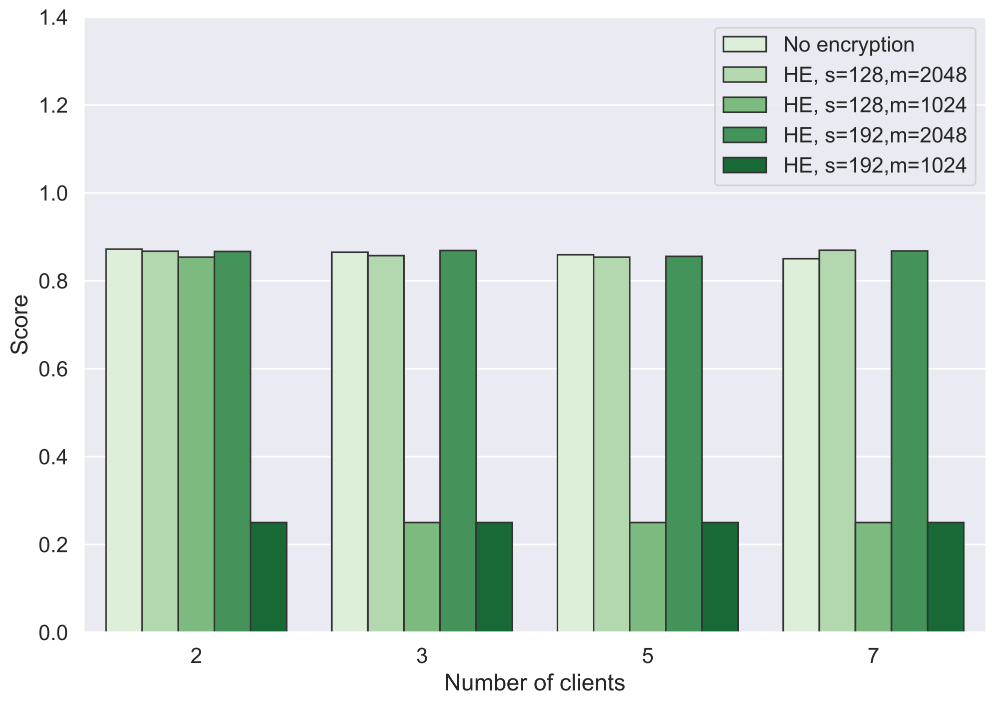

Figure 4.4: Precision Scores

Figure 4.4 shows that good Precision scores were recorded when m=2048, while bad scores were recorded when m=1024, with the exception of federated learning with two participating clients where the Precision score doesn't suffer from a smaller m(1024) value. The histogram also shows that there's no significant difference on the Precision scores between sec=128 and sec=192 for the same m value.

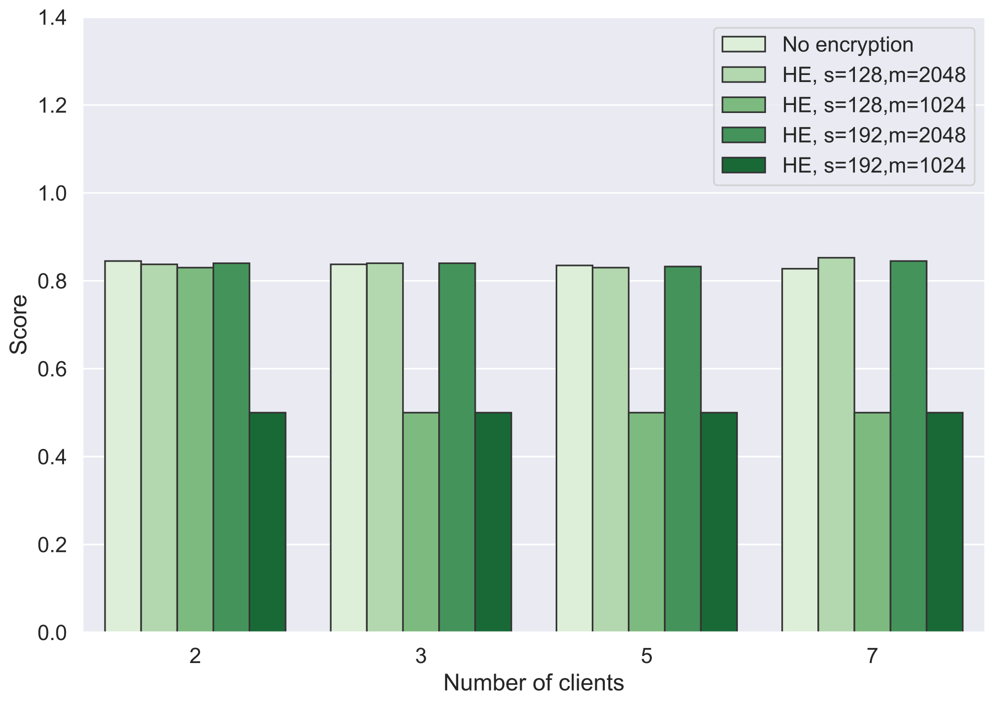

Figure 4.5: Recall Scores

Figure 4.5 shows that good Recall scores were recorded when m=2048, while bad scores were recorded when m=1024, with the exception of federated learning with two participating clients where the Recall score doesn't suffer from a smaller m(1024) value. The histogram also shows that there's no significant difference on the Recall scores between sec=128 and sec=192 for the same m value.

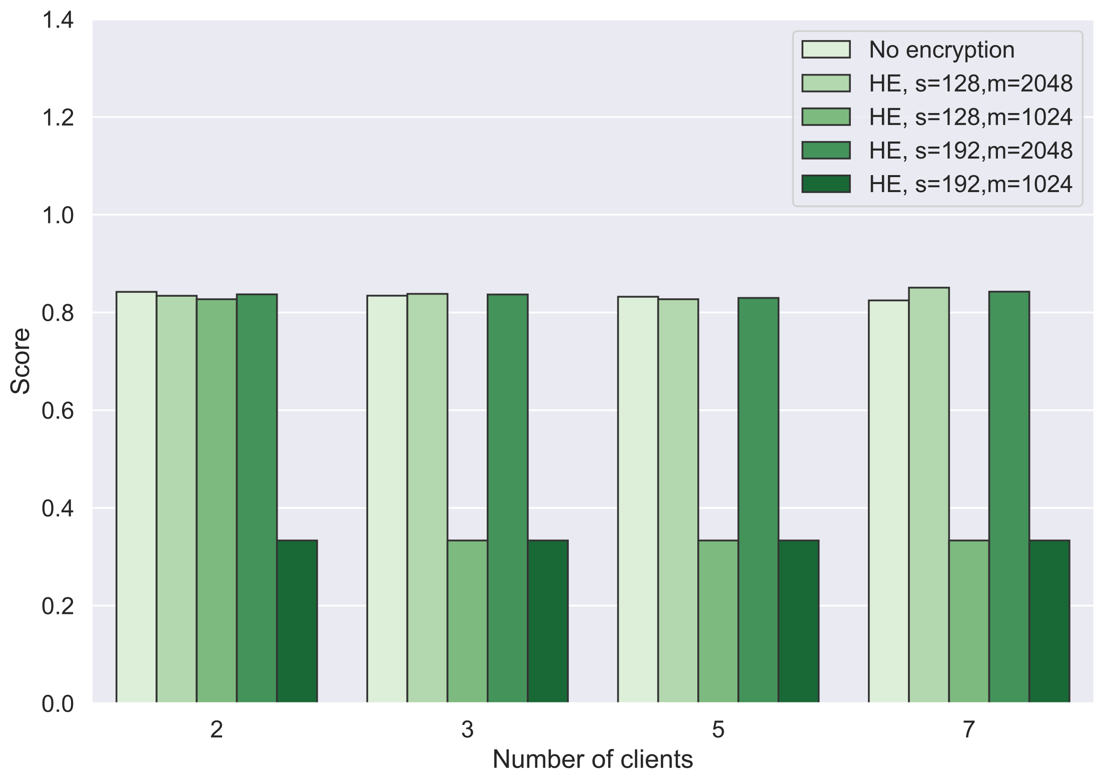

Figure 4.6: F1 Scores

Figure 4.6 shows that good F1 Scores were recorded when m=2048, while bad scores were recorded when m=1024, with the exception of federated learning with two participating clients where the F1 Score doesn't suffer from a smaller m(1024) value. The histogram also shows that there's no significant difference on the F1 scores between sec=128 and sec=192 for the same m value.

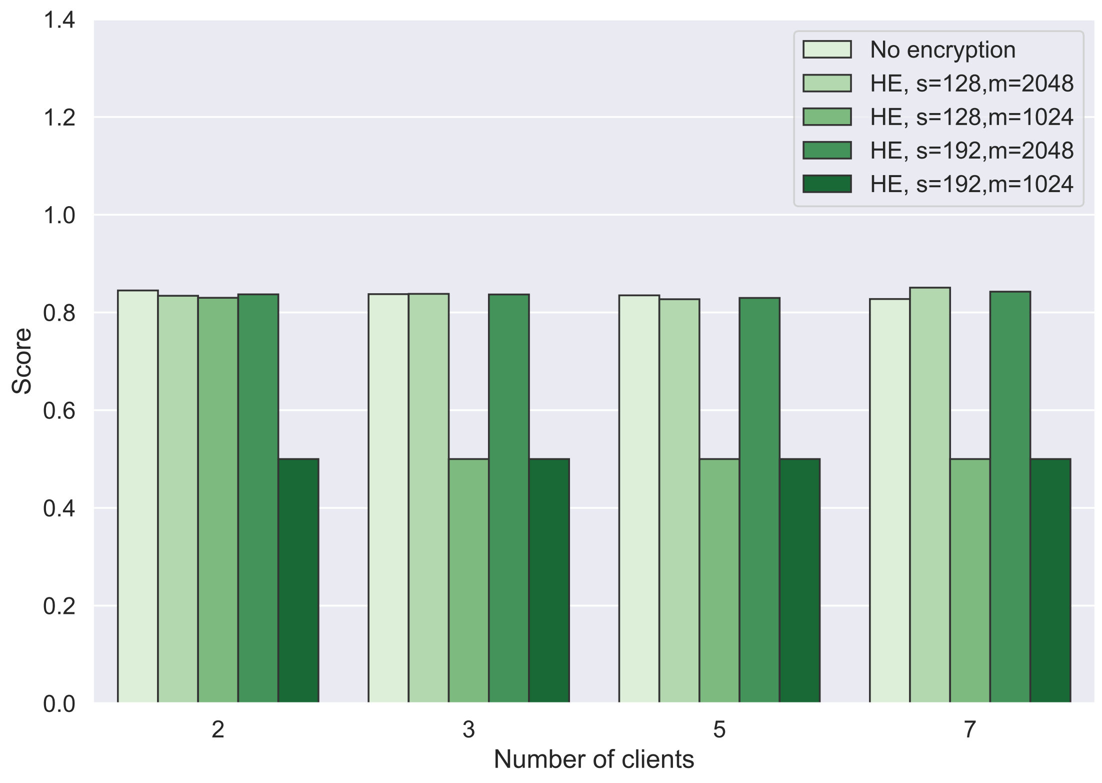

Figure 4.7: Accuracy Scores

Figure 4.7 shows that good Accuracy scores were recorded when m=2048, while bad scores were recorded when m=1024 , with the exception of federated learning with two participating clients where the Accuracy score doesn't suffer from a smaller m(1024) value. The histogram also shows that there's no significant difference on the Accuracy scores between sec=128 and sec=192 for the same m value.

4.3.2 Execution Time

Apart from measuring the model prediction performance using evaluation metrics, we also measured the total running time of each experimentation. The total running time consists of time running in:

- Training the COVID-19 prediction model at clients
- Encrypt weights and export encrypted weights as pickle files for all clients
- Import pickle files
- Aggregate weights
- Export aggregated model weights as a pickle file

- Import aggregated model weights for one client
- Update model and run prediction for one client
- Generate model evaluation metrics

Table 4.10 and Table 4.11 below show processing time in seconds, with various number of clients based on parameter m equals to 1024 and m equals to 2048 respectively.

Number of clients	Without Encryption	Encryption(sec=128)	Encryption (sec=192)
2	594.165448	3155.804616	4177.638341
3	647.963712	4826.452512	5010.716965
5	720.175786	4958.373842	5167.173689
7	948.704833	5557.469271	6077.708571

Table 4.10: Execution time in seconds with m=1024

Number of clients	Without Encryption	Encryption(sec=128)	Encryption (sec=192)
2	594.165448	4333.672333	4765.874634
3	647.963712	5124.841524	7504.239611
5	720.175786	6777.249099	10518.012003
7	948.704833	9223.281346	13277.904182

Table 4.11: Execution time in seconds with m=2048

The execution time is also shown as histogram in below Figure 4.8.

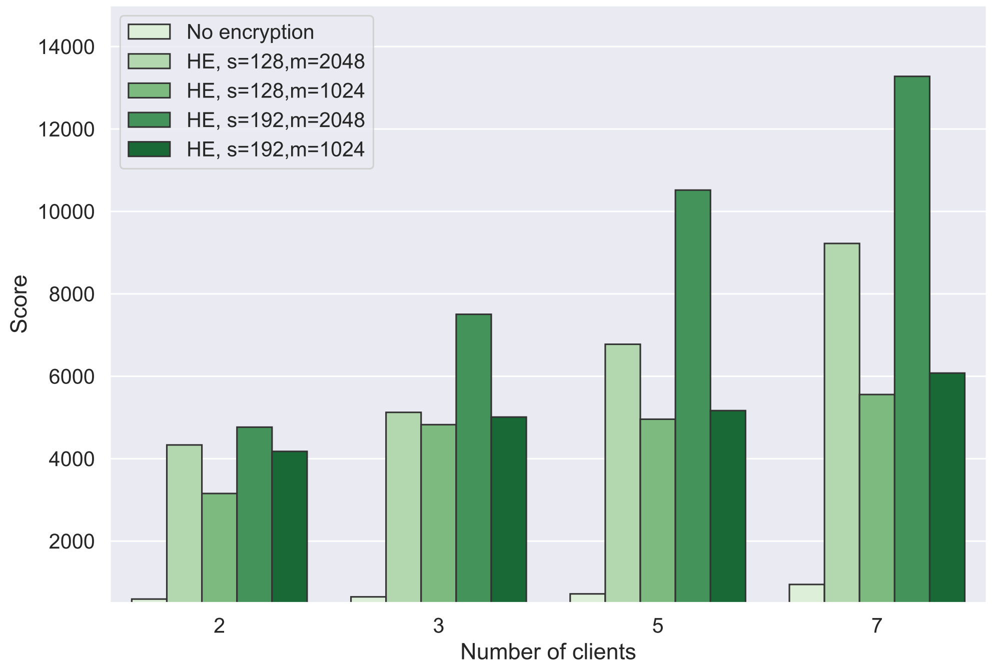

Figure 4.8: Execution time in seconds

From Figure 4.8, we see that there's significant difference in execution time between model with hyperparameter sec=128 and sec=192, where it took much longer execution time when hyperparameter sec=192. We also see that hyperparameter m=2048 also caused longer execution time.

Lastly, we also observed that the size of encrypted weights pickle file is around 7GB for polynomial modulus $(m) = 2048$, 3.5 GB for m=1024. While the pickle file size for unencrypted weights is 871 KB.

Chapter 5

Environmental Accounts

From the experimentation results, implementation of homomorphic encryption together with federated learning increases execution time and harddisk space, which in the end also increases power consumption.

Power consumption may be justifiable due to the increasing usage of renewable energy in generating power. The trade-off between environmental impact and the proposed solution is perhaps not unreasonable, considering the solution offers privacy-preserved federated learning, which is essential to federated learning process across borders.

Chapter 6

Discussion

Figure 4.8 provides new insights into the relationships between the different numbers of clients and execution time. There is a significant difference in execution time between plain (unencrypted) and encrypted data processes. These exponential differences are due to the complexity of homomorphic encryption in processing model weights. The execution time also noticeably increases with increasing parameter (m) value from 1024 to 2048.

For the prediction result, the performances of both encrypted and unencrypted models are very similar as indicated in Table 4.5, Table 4.8, and Table 4.9, when polynomial modulus (m) was set to 2048. On the contrary, poor model prediction performance is recorded when polynomial modulus (m) was set to 1024, as show in Table 4.6 and Table 4.7.

Adjusting the security level parameter (sec) from 128 to 192 did not make much difference in model prediction performance, though it did increase execution time. By maintaining the same value in the security level parameter (sec), the model provided a better prediction result when the polynomial modulus (m) parameter was set to 2048. Reducing the parameter value to 1024 helped to reduce the execution time but did not maintain or improve the model prediction performance.

Similar model prediction performances were achieved in each model by increasing the number of clients. For some cases, results with plain data performed slightly better than the applied encryption results. For instance, the accuracy results of five clients indicated that plain versions achieved better results for each evaluation metric namely, Accuracy, F1, Precision, and Recall.

Chapter 7

Conclusion and Future Directions

7.1 Conclusion

Privacy-preserving has become an essential practice of healthcare institutions as both the European Union and the United States mandate it. Federated learning and homomorphic encryption will play a critical role in maintaining data security and model training. The proposed model benefits from both techniques as it achieves competitive performance although there can be a significant trade-off in respect of the execution time when varying the number of clients. In some cases where privacy-preserving is very crucial, such as in healthcare as in this instance, this trade-off can be acceptable.

The evaluation metrics, i.e., Accuracy, F1, Precision, and Recall score, reach over 80% using both encrypted and plain data for each federated learning case which means that homomorphic encryption, SHE, does not deteriorate model prediction performance.

The slightly lower performance of encrypted model is due to the fact that somewhat homomorphic encryption does not convert the ciphertext precisely back to its original plaintext value but rather an approximate value with very small noise.

There is a trade-off between execution time, hard disk space, power consumption, and preserving privacy. Although homomorphic implementation increases execution time, hard disk space requirement, and possibly communication traffic and power consumption, the model privacy is preserved. This protection enables flexibility in central server hosting in a public cloud (untrusted) environment so that collaborative learning can be done seamlessly across borders.

Any privacy attacks will cause immense damage to the security and privacy of patient information. It will hinder the advancement in healthcare using data-driven models.

Therefore, it is indispensable to take crucial steps to strengthen the safety of the information and the way data is processed. This study demonstrated that federated learning with homomorphic encryption could be successfully applied to enhance data-driven models by eliminating and minimizing the share of sensitive data. This thesis could be helpful for scientists and researchers working on sensitive healthcare data in multi-party computation settings.

7.2 Future Direction

This thesis proves the concept of a privacy-preserving of federated learning, where we utilized one public key in homomorphic encryption for all clients. In this case, there's room for improvement where multikey homomorphic encryption is considered. Multikey homomorphic encryption allows computation on encrypted data with multiple unrelated keys. It is beneficial since, in real-life scenarios, clients do not necessarily trust each other to share the same encryption key. Thus it improves the security of the proposed model.

Appendix A

Supplementary information

A.1 Link to Codes Repository

https://github.com/FebriantiW/Homomorphic-Encryption-and-Federated-Learning-based-Privacy-Preserving-CNN-Training-

A.2 Brakerski-Fan-Vercauteren(BFV) functions

In the early stage, plaintexts are encoded as polynomials. The encoding works by converting the plaintexts to binary $m = a_n - 1 ... a_2 a_1 a_0$ then composing polynomials $M = a_n - 1x + ... + a_2 x^2 + a_1 x + a_0$. The unused bits are then removed to simplify the polynomials.

The plaintext and ciphertext space are denoted by $\mathcal{P} = \mathcal{R}_t = \mathbb{Z}_t[x]/(x^n + 1)$ and $\mathcal{C} = \mathcal{R}_q \times \mathcal{R}_q$ where $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$, $n \in \mathcal{Z}$ is the ring dimension, $t \in \mathcal{Z}$ plaintext coefficient modulus with $1 < t < q$, and $q \in \mathcal{Z}$ is ciphertext coefficient. n is commonly set as power of 2 integer. The polynomial ring notation can be seen as set of polynomials with integer coefficients modulo both t or q and $(x^n + 1)$.

Below are BFV functions that are relevant for this thesis, taken from [26]:

- Parameter Generator Apart from mentioned parameter (t, q, n), BFV uses several random distributions as follows:
  1. $\mathcal{R}_2$: key distribution to sample polynomials with integer coefficients $-1, 0, 1$.

2. $\mathcal{X}$: a discrete Gaussian distribution with parameters μ, σ over $\mathcal{R}$ bounded by some integer β which are set as $(0, \frac{8}{\sqrt{2\pi}} \approx 3.2, \lfloor 6 \cdot \sigma \rceil = 19)$, used for error sampling.
3. $\mathcal{R}_q$: is a uniform random distribution over $\mathcal{R}_q$

- Key generation This function takes encryption parameters which are generated by Parameter generator function, as input and returns a set of keys, secret key (**Sk**), public key (**Pk**), and evaluation key (**Ek**). **Sk** and **Pk** is used for encryption and decryption, while **Ek** is used for homomorphic operation on ciphertexts.
  - Secret key (**Sk**) is generated as a random polynomial from $\mathcal{R}_2$ parameter mentioned earlier in (1)
  - Public key (**Pk**) : is a pair of polynomials $(\mathsf{PK}_1 = -(a \cdot \mathsf{SK} + e) \mod q, \mathsf{PK}_2 = a)$ where a is a random polynomial in $\mathcal{R}_q$ and e is a random error polynomial sampled from $\mathcal{X}$.
- Encryption The encryption function takes public key **Pk** and plaintext **P** in plaintext space $\mathcal{P}$ as input and return ciphertext **C**. Encryption works by selecting three small random polynomials u from $\mathcal{R}_2$ and e_1, e_2 from $\mathcal{X}$. These random polynomials then used to encrypt the plain text to be ciphertext $\mathsf{C} = (\mathsf{C}_1, \mathsf{C}_2)$ in ciphertext space $\mathcal{C}$ as follows:

$$\begin{aligned} \mathsf{C}_1 &= (\mathsf{Pk}_1 \cdot u + e_1 + \Delta \mathsf{M}) \mod q \qquad (A.1) \\ \mathsf{C}_2 &= (\mathsf{Pk}_2 \cdot u + e_2) \mod q \end{aligned}$$

The parameter Δ is defined as $\Delta = \lfloor \frac{q}{t} \rfloor$, which is used to scale the message.

- Decryption The decryption function takes **Pk** and ciphertext **C** as input and return plaintext **M**.

$$\mathsf{M} = \left[\left\lfloor \frac{t(\mathsf{C}_1 + \mathsf{C}_2 \cdot \mathsf{Sk}) \mod q}{q} \right\rceil \right] \mod t \qquad (A.2)$$

- Homomorphic evaluation
  - Additive
    The procedure to perform additive operation is quite straight forward.

$$\mathsf{EvalAdd}(\mathsf{C}^{(1)}, \mathsf{C}^{(2)}) = ([\mathsf{C}_1^{(1)} + \mathsf{C}_1^{(2)}] \mod q, [\mathsf{C}_2^{(1)} + \mathsf{C}_2^{(2)}] \mod q) \qquad (A.3)$$

Substituting the previous ciphertext expression, the additive can be expressed as:

$$\mathsf{C}^{(1)} = ([\mathsf{Pk}_1 \cdot u^{(1)} + e_1^{(1)} + \Delta\mathsf{M}^{(1)}]_q, [\mathsf{Pk}_2 \cdot u^{(1)} + e_2^{(1)}] \mod q) \tag{A.4}$$
$$\mathsf{C}^{(2)} = ([\mathsf{Pk}_1 \cdot u^{(2)} + e_1^{(2)} + \Delta\mathsf{M}^{(2)}]_q, [\mathsf{Pk}_2 \cdot u^{(2)} + e_2^{(2)}] \mod q)$$

$$\begin{aligned}\mathsf{C}^{(3)} &= (\mathsf{C}_1^{(3)}, \mathsf{C}_2^{(3)}) && \text{(A.5)}\\ &= ([\mathsf{Pk}_1 \cdot (u^{(1)} + u^{(2)}) + (e_1^{(1)} + e_1^{(2)}) + \Delta(\mathsf{M}^{(1)} + \mathsf{M}^{(2)})] \mod q,\\ &\quad [\mathsf{Pk}_2 \cdot (u^{(1)} + u^{(2)}) + (e_2^{(1)} + e_2^{(2)})] \mod q)\\ &= ([\mathsf{Pk}_1 \cdot u^{(3)} + e_1^{(3)} + \Delta(\mathsf{M}^{(1)} + \mathsf{M}^{(2)})] \mod q, [\mathsf{Pk}_2 \cdot u^{(3)} + e_2^{(3)}] \mod q) && \text{(A.6)}\end{aligned}$$

– Multiplicative

Let the ciphertext is expressed in (**Sk**).

$$\mathsf{C}^{(1)}(\mathsf{Sk}) = \Delta\mathsf{M}^{(1)} + v_1 + q \cdot r_1 \tag{A.7}$$
$$\mathsf{C}^{(2)}(\mathsf{Sk}) = \Delta\mathsf{M}^{(2)} + v_2 + q \cdot r_2$$

Multiplying the ciphertexts would be:

$$\begin{aligned}(C^{(1)} \cdot \mathsf{C}^{(2)})(\mathsf{SK}) =&(\Delta\mathsf{M}^{(1)} + v_1 + q \cdot r_1) \cdot (\Delta\mathsf{M}^{(2)} + v_2 + q \cdot r_2) && \text{(A.8)}\\ &= \Delta^2\mathsf{M}^{(1)} \cdot \mathsf{M}^{(2)} + \Delta(\mathsf{M}^{(1)} \cdot v_2 + \mathsf{M}^{(2)} \cdot v_1)+\\ &\quad q(v_1 \cdot r_2 + v_2 \cdot r_1) + q \cdot \Delta(\mathsf{M}^{(1)} \cdot r_2 + \mathsf{M}^{(2)} \cdot r_1)+ && \text{(A.9)}\\ &\quad v_1 \cdot v_2 + q^2 \cdot r_1 \cdot \cdot r_2\end{aligned}$$

In order to get an encryption of $[m_1, m_2]_t$ and avoid large noise , above expression A.8 must be divided by q/t, or equivalently multiply by t/q. $v_1 \cdot v_2$ can also be substituted with $[v_1 \dot{v}_2]_\delta + \Delta\dot{r}_v$ and above expression can be rewritten as:

$$\begin{aligned}\frac{t}{q}(\mathsf{C}^{(1)} \cdot \mathsf{C}^{(2)})(\mathsf{SK}) =&\Delta[\mathsf{M}^{(1)} \cdot \mathsf{M}^{(2)}]_t + (\mathsf{M}^{(1)} \cdot v_2 + \mathsf{M}^{(2)} \cdot v_1)+\\ &t(v_1 \cdot r_2 + v_2 \cdot r_1) + r_v + (q - [q]_t) \cdot (r_M + \mathsf{M}^{(1)} \cdot r_2 + \mathsf{M}^{(2)} \cdot r_1)+\\ &q \cdot t \cdot r_1 \cdot r_2 + \frac{t}{q}[v_1 \cdot v_2]_\Delta -\\ &\frac{[q]_t}{q}(\Delta\mathsf{M}^{(1)} \cdot \mathsf{M}^{(2)} + \mathsf{M}^{(1)} \cdot v_2 + \mathsf{M}^{(2)} \cdot v_1 + r_v)\end{aligned} \tag{A.10}$$

The final expression after reducing equation A.10 modulo q is:

$$\begin{aligned}\frac{t}{q}(\mathsf{C}^{(1)} \cdot \mathsf{C}^{(2)})(\mathsf{SK}) =& \Delta[\mathsf{M}^{(1)} \cdot \mathsf{M}^{(2)}] \mod t + (\mathsf{M}^{(1)} \cdot v_2 + \mathsf{M}^{(2)} \cdot v_1) + \\ & t(v_1 \cdot r_2 + v_2 \cdot r_1) + r_v - \\ & [q]_t(r_M + \mathsf{M}^{(1)} \cdot r_2 + \mathsf{M}^{(2)} \cdot r_1) + r_e\end{aligned} \tag{A.11}$$

In short, tensor product of ciphertexts, scaled by t/q as follows:

$$\begin{aligned}\mathsf{EvalMult}(\mathsf{C}^{(1)}, \mathsf{C}^{(2)}) = \Bigg(& \left[\left\lfloor \frac{t(\mathsf{C}_1^{(1)} \cdot \mathsf{C}_1^{(2)})}{q} \right\rceil\right] \mod q, \left[\left\lfloor \frac{t(\mathsf{C}_1^{(1)} \cdot \mathsf{C}_2^{(2)} + \mathsf{C}_2^{(1)} \cdot \mathsf{C}_1^{(2)})}{q} \right\rceil\right] \mod q, \\ & \left[\left\lfloor \frac{t(\mathsf{C}_2^{(1)} \cdot \mathsf{C}_2^{(2)})}{q} \right\rceil\right] \mod q\Bigg)\end{aligned} \tag{A.12}$$

It is shown then that ciphertext elements increases from 2 to 3 ring elements in multiplicative operation.

List of Figures

List of Tables

Bibliography

[1] Oskar Goldhahn. A look into homomorphic cryptography and the bv homomorphic encryption scheme. Master's thesis, NTNU, 2020. URL https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2980245/no.ntnu%3Ainspera%3A56982622%3A22574430.pdf?sequence=1.

[2] A. Carey. On the explanation and implementation of three open-source fully homomorphic encryption libraries. Master's thesis, University of Arkansas, 2020. URL https://scholarworks.uark.edu/csceuht/77.

[3] Monique Ogburn, Claude Turner, and Pushkar Dahal. Homomorphic encryption. *Procedia Computer Science*, 20:502–509, 12 2013. doi: 10.1016/j.procs.2013.09.310.

[4] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøsteen, Angela Jäschke, Christian A. Reuter, and Martin Strand. A guide to fully homomorphic encryption. Cryptology ePrint Archive, Paper 2015/1192, 2015. URL https://eprint.iacr.org/2015/1192. https://eprint.iacr.org/2015/1192.

[5] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2012:144, 2012.

[6] Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletari, Holger R Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N Galtier, Bennett A Landman, Klaus Maier-Hein, et al. The future of digital health with federated learning. *NPJ digital medicine*, 3(1):1–7, 2020.

[7] Sannara Ek, François Portet, Philippe Lalanda, and German Vega. Evaluation of federated learning aggregation algorithms application to human activity recognition. *UbiComp/ISWC*, 20:638–643, 2020. ISSN 10.1145/3410530.3414321. URL https://hal.archives-ouvertes.fr/hal-02941944/document.

[8] Jie Xu, Benjamin S Glicksberg, Chang Su, Peter Walker, Jiang Bian, and Fei Wang. Federated learning for healthcare informatics. *Journal of Healthcare Informatics Research*, 5(1):1–19, 2021.

[9] Rodolfo Stoffel Antunes, Cristiano André da Costa, Arne Küderle, Imrana Abdullahi Yari, and Björn Eskofier. Federated learning for healthcare: Systematic review and architecture proposal. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2022.

[10] Wenqi Li, Fausto Milletarì, Daguang Xu, Nicola Rieke, Jonny Hancox, Wentao Zhu, Maximilian Baust, Yan Cheng, Sébastien Ourselin, M Jorge Cardoso, et al. Privacy-preserving federated brain tumour segmentation. In *International workshop on machine learning in medical imaging*, pages 133–141. Springer, 2019.

[11] Micah J Sheller, G Anthony Reina, Brandon Edwards, Jason Martin, and Spyridon Bakas. Multi-institutional deep learning modeling without sharing patient data: A feasibility study on brain tumor segmentation. In *International MICCAI Brainlesion Workshop*, pages 92–104. Springer, 2018.

[12] A Vijaya Kumar, Mogalapalli Sai Sujith, Kosuri Tarun Sai, Galla Rajesh, and Devulapalli Jagannadha Sriram Yashwanth. Secure multiparty computation enabled e-healthcare system with homomorphic encryption. In *IOP Conference Series: Materials Science and Engineering*, volume 981, page 022079. IOP Publishing, 2020.

[13] Razvan Bocu and Cosmin Costache. A homomorphic encryption-based system for securely managing personal health metrics data. *IBM Journal of Research and Development*, 62(1):1–1, 2018.

[14] Xiaoni Wang and Zhenjiang Zhang. Data division scheme based on homomorphic encryption in wsns for health care. *Journal of medical systems*, 39(12):1–7, 2015.

[15] Mostefa Kara, Abdelkader Laouid, Mohammed Amine Yagoub, Reinhardt Euler, Saci Medileh, Mohammad Hammoudeh, Amna Eleyan, and Ahcène Bounceur. A fully homomorphic encryption based on magic number fragmentation and el-gamal encryption: Smart healthcare use case. *Expert Systems*, page e12767, 2021.

[16] Mir Sajjad Hussain Talpur, Md Zakirul Alam Bhuiyan, and Guojun Wang. Shared–node iot network architecture with ubiquitous homomorphic encryption for healthcare monitoring. *International Journal of Embedded Systems*, 7(1):43–54, 2015.

[17] Haowen Tan, Pankoo Kim, and Ilyong Chung. Practical homomorphic authentication in cloud-assisted vanets with blockchain-based healthcare monitoring for pandemic control. *Electronics*, 9(10):1683, 2020.

[18] Aitizaz Ali, Muhammad Fermi Pasha, Jehad Ali, Ong Huey Fang, Mehedi Masud, Anca Delia Jurcut, and Mohammed A Alzain. Deep learning based homomorphic secure search-able encryption for keyword search in blockchain healthcare system: A novel approach to cryptography. *Sensors*, 22(2):528, 2022.

[19] Kim Laine. Simple encrypted arithmetic library 2.3.1. https://www.microsoft.com/en-us/research/uploads/prod/2017/11/sealmanual-2-3-1.pdf.

[20] Muhammad E. H. Chowdhury, Tawsifur Rahman, Amith Khandakar, Rashid Mazhar, Muhammad Abdul Kadir, Zaid Bin Mahbub, Khandakar Reajul Islam, Muhammad Salman Khan, Atif Iqbal, Nasser Al Emadi, Mamun Bin Ibne Reaz, and Mohammad Tariqul Islam. Can ai help in screening viral and covid-19 pneumonia? *IEEE Access*, 8:132665–132676, 2020. doi: 10.1109/ACCESS.2020.3010287.

[21] Tawsifur Rahman, Amith Khandakar, Yazan Qiblawey, Anas Tahir, Serkan Kiranyaz, Saad Bin Abul Kashem, Mohammad Tariqul Islam, Somaya Al Maadeed, Susu M. Zughaier, Muhammad Salman Khan, and Muhammad E.H. Chowdhury. Exploring the effect of image enhancement techniques on covid-19 detection using chest x-ray images. *Computers in Biology and Medicine*, 132:104319, 2021. ISSN 0010-4825. doi: https://doi.org/10.1016/j.compbiomed.2021.104319. URL https://www.sciencedirect.com/science/article/pii/S001048252100113X.

[22] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 868–886, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-32009-5.

[23] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. https://ia.cr/2012/144.

[24] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437, Cham, 2017. Springer International Publishing. ISBN 978-3-319-70694-8.

[25] Alberto Ibarrondo and Alexander Viand. Pyfhel: Python for homomorphic encryption libraries. In ACM, editor, *WAHC 2021, 9th Workshop on Encrypted Computing &amp; Applied Homomorphic Cryptography, Associated with the ACM CCS 2021 conference, 15 November 2021, Seoul, South Korea*, Seoul, 2021.

[26] BFVinferati. Introduction to the bfv encryption scheme. https://inferati.com/blog/fhe-schemes-bfv.