2024 IEEE 3rd International Conference on AI in Cybersecurity (ICAIC) | 979-8-3503-8185-6/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICAIC60265.2024.10433836

Secure federated learning applied to medical imaging with fully homomorphic encryption

Xavier Lessage
Applied Research Centre
CETIC
Charleroi, Belgium
xavier.lessage@cetic.be

Leandro Collier
Applied Research Centre
CETIC
Charleroi, Belgium
leandro.collier@cetic.be

Charles-Henry Bertrand Van Ouytsel
Faculty of Engineering
UCLouvain
Louvain-la-Neuve, Belgium
charles-henry.bertrand@uclouvain.be

Axel Legay
Faculty of Engineering
UCLouvain
Louvain-la-Neuve, Belgium
axel.legay@uclouvain.be

Saïd Mahmoudi
Faculty of Engineering
UMONS
Mons, Belgium
said.mahmoudi@umons.ac.be

Philippe Massonet
Applied Research Centre
CETIC
Charleroi, Belgium
Philippe.massonet@cetic.be

***Abstract*—This study explores the convergence of Federated Learning (FL) and Fully Homomorphic Encryption (FHE) through an innovative approach applied to a confidential dataset composed of mammograms from Belgian medical records. Our goal is to clarify the feasibility and challenges associated with integrating FHE into the context of Federated Learning, with a particular focus on evaluating the memory constraints inherent in FHE when using sensitive medical data. The results highlight notable limitations in terms of memory usage, underscoring the need for ongoing research to optimize FHE in real-world applications. Despite these challenges, our research demonstrates that FHE maintains comparable performance in terms of Receiver Operating Characteristic (ROC) curves, affirming the robustness of our approach in secure machine learning applications, especially in sectors where data confidentiality, such as medical data management, is imperative. The conclusions not only shed light on the technical limitations of FHE but also emphasize its potential for practical applications. By combining Federated Learning with FHE, our model preserves data confidentiality while ensuring the security of exchanges between participants and the central server**

***Index Terms*—breast cancer, masses and microcalcifications detection, federated learning, homomorphic encryption, convolutional neural networks.**

I. Introduction

Deep neural networks offer a promising approach to the development of health assistance tools. Yet when it comes to training these models using real data from, for example, the medical or industrial sectors, there are several concerns:

- **Class imbalance**: often classes with anomalies (or special cases) are in the minority when compared with classes that possess no anomalies.
- **Data access**: the data acquisition process requires expert annotation and is subject to data protection legislation (GDPR) [37].
- **Risk of leaking private information**: the rise of artificial intelligence entails a major risk of a confidentiality breach and a loss of control over personal data.
- **Trust between participants**: trust and clearly defined roles for participants ensure data quality and reliability.

The development of healthcare assistance tools suffers from a lack of accessible, relevant data. Getting a set of reliable, anonymized data often requires a long time – sometimes months or years. Even when data is available, the data may be unbalanced and of limited use in developing effective tools.

Federated learning secured via fully homomorphic encryption offers a solution to these many constraints.

A. Federated learning

Federated learning is a solution for overcoming this limited availability of data. Federated learning encompasses tools that enable a global model to learn from several distributed sub-models. Participants agree to train similar models on their local data and to exchange these models with a central server that then aggregates the models before sending them back to participants. This allows each participant to maintain privacy for and control over their local data while ultimately benefiting from a 'shared' final model that is likely to perform better. Figure 1 illustrates a typical federated learning ecosystem.

In this federated learning ecosystem, a group of participants share the weights of their locally trained model. A central unit aggregates these weights. The architecture allows participants to share only the gradients and weights of their model. However, the architecture also raises several challenges in terms of preserving privacy and confidentiality. These challenges include the anonymization (or pseudonymization if necessary) of the locally used dataset and the confidentiality of the models and gradients. This confidentiality must be guaranteed to prevent any reverse engineering of the training dataset. As a final challenge, the model requires protection against degradation resulting from training on inadequate data. To resolve these challenges, Kaissis et al. [31] in their work proposed

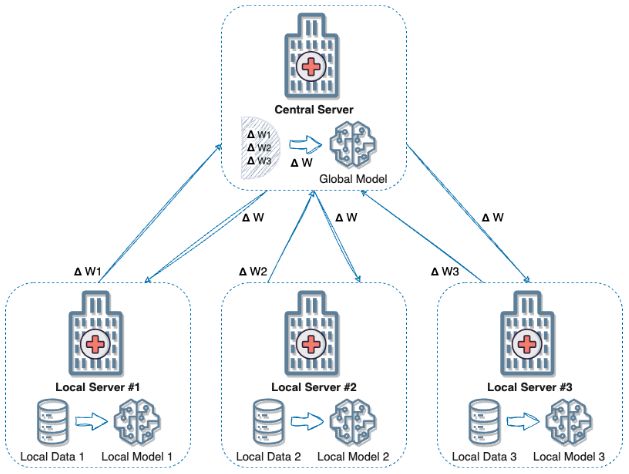


Fig. 1. Typical federated learning ecosystem.

a secure, privacy-preserving architecture that use federated learning to develop a global model from the aggregation of weights transmitted by local hospitals. They proposed multi-layer protection, from the anonymization of local datasets to the secure transmission of local models to the central server.

B. Inference attack and Model Poisoning Attacks

In the example under discussion in Figure 1, participant's data is not shared with the group but remains under the control of its original owners thanks to the federated architecture. That said, each participant's data is still vulnerable to various attacks. The distributed architecture introduces various issues that require solutions if the participants want to avoid any leakage of information or data poisoning. As illustrated in Figure 2 below, it is possible to access information from the model during the transmission between different participants and the global server; it is also possible to alter the contents of a transmission. Such inference or poisoning attacks threaten the shared model weights between participants and the global server.

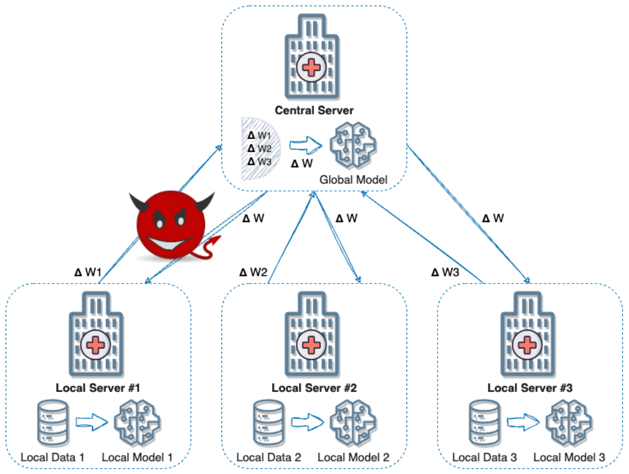


Fig. 2. Inference attack on federated learning architecture.

Inference attacks in federated learning consist of inferring or deducing information about the training data of individual participants from the global model. There are different types of inference attacks in federated learning[28]:

- **Membership inference attacks** aim to determine whether a particular data sample was used to train a federated learning model.
- **Attribute inference attacks** aim to infer the values of sensitive attributes of data samples from a federated learning model.
- **Model inversion attacks** aim to reconstruct confidential training data from a federated learning model.
- **Gradient leakage attacks** aim to infer sensitive information from a machine learning model by observing the gradients of the model's output with respect to its inputs to help reconstruct inputs.
- **Property inference attacks** aim to infer properties of the training data, e.g., data distribution or the presence of certain patterns in the data.
- **Data reconstruction attacks** aim to reconstruct the training data from the federated learning model, even if the attacker is without access to individual data samples.

For each of the above federated learning inference attacks, several available countermeasures exist. For membership inference attacks, useful mitigation techniques include: differential privacy, data perturbation techniques, federated averaging with random shuffling, client sampling and data hiding, federated transfer learning, model regularization techniques, and secure aggregation protocols. For attribute inference attacks, mitigation techniques comprise: differential privacy, local model aggregation with noise, secure multi-party Computation (MPC), federated transfer learning, data perturbation techniques, limiting model access, and adversarial training. Model inversion attacks mitigation techniques to consider include: differential privacy, model regularization, adversarial training, secure model aggregation protocols, restricted model access, and model distillation techniques. For gradient leakage attacks the following mitigation techniques can be effective: secure aggregation protocols, differential privacy, gradient compression techniques, client sampling and data shuffling, privacy-preserving optimization algorithms, gradient masking techniques, and regularization and noise injection. Property inference attacks can be mitigated via: differential privacy, secure aggregation protocols, data perturbation techniques, federated averaging with random shuffling, and adversarial training. It is possible to mitigate data reconstruction attacks with: differential privacy, data perturbation techniques, federated averaging with random shuffling, secure aggregation protocols, data anonymization and masking, restricted data access, and federated transfer learning.

To guard against these types of inference attacks, we examine the use of full homomorphic encryption as an additional countermeasure to those listed above. The principle of full homomorphic encryption is described in the section I-C.

Model learning is generally considered a high-level representation of the data [34]. As such, it is possible to illegitimately gain knowledge from a model via an inference attack and to modify the behavior of a model via model poisoning attacks during model learning. By carefully incorporating fake clients into the federated learning system, the attacker can unobtrusively alter the learning process of the global model, leading to a reduction in its accuracy against different test data. The attack method involves subtly steering the global model towards a chosen reference model characterized by reduced accuracy and using fake participants to construct and amplify false local model updates that are directed towards this reference model at each stage of federated learning [13].

C. Full homomorphic encryption (FHE)

Homomorphic encryption solves a central problem with traditional encryption algorithms. Traditional encryption algorithms require decryption before operations can be performed on the data, but decrypting the data makes it vulnerable to unauthorized access. The literature shows that not even quantum attacks are capable of decrypting homomorphically encrypted data.

A homomorphic property generally exists if the image of a composite of two elements is the composite of the images of those elements. A homomorphic encryption system has this property for the encoder, decoder, encryptor, and decryptor, so that the ciphertext is decrypted and decoded as if we had performed the operations directly on the plaintext.

In [3], an encryption algorithm E is considered mathematically homomorphic for a given operation $\triangle$ if:

$$E(m1) \triangle E(m2) = E(m1 \triangle m2) \quad \forall\ m1, m2 \in M \tag{1}$$

With the set of all possible messages M.

In short, homomorphic encryption (HE) is a cryptographic scheme that allows homomorphic operations on encrypted data without decryption. As shown in the figure below, there are different types of homomorphic encryption:

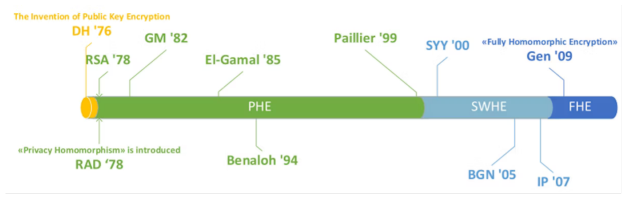

The Invention of Public Key Encryption
DH '76
GM '82
RSA '78
El-Gamal '85
Paillier '99
SYY '00
«Fully Homomorphic Encryption»
Gen '09
PHE
SWHE
FHE
«Privacy Homomorphism» is introduced
RAD '78
Benaloh '94
BGN '05
IP '07

Fig. 3. Timeline of HE schemes until Gentry's first FHE scheme [26]

- **Partially Homomorphic Encryption (PHE)** supports only one operation type. RSA encryption developed by Rivest et al. in 1978 is the first encryption of this kind.
- **Somewhat Practical Fully Homomorphic Encryption (SHWE)** allows some combinations of additions and multiplications before returning an invalid result.
- **Fully Homomorphic Encryption (FHE)** [21] allows repeated addition and multiplication without loss of fidelity.

This paper concentrates on the latter. There are many schemes for homomorphic encryption:

- Gentry scheme [21], 2009.
- DGHV scheme (Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan), 2010;
- BGV scheme (Brakerski-Gentry-Vaikuntanathan), 2011;
- **BFV scheme (Brakerski/Fan-Vercauteren), [19]2012**;
- GSW scheme (Craig Gentry, Amit Sahai, and Brent Waters), 2013;
- **CKKS scheme (Cheon-Kim-Kim-Song [15]), 2017**.

Our selected library only supports BFV[19] and CKKS [15]. BFV is performed strictly on integers and is more precise yet slower than CKKS. CKKS supports approximate arithmetic over real numbers. For weight encryption, we prefer the CKKS scheme comprised of four steps (Figure 4).

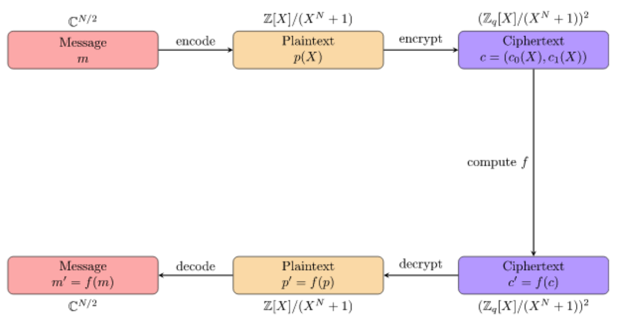


Fig. 4. Steps in the CKKS scheme [25]

1) Encoding: The first step is to encode the vector/tensor of real numbers $\mathbb{Z} \in \mathbb{C}^{\frac{N}{2}}$ into a plaintext polynomial $m(X) \in R = \mathbb{Z}[X]/(X^N+1)$. R being the set of cyclotomic polynomial rings[1] and N being a ring dimension that is a power of two. The use of polynomials is a good compromise between security and efficiency compared to vector computations. This step involves three parameters:

- **Scaling factor** (*global_scale*), which defines the encoding precision using floating-point numbers for the binary representation of the number. In equation 2, 1.100101 is the Significant (mantissa), and 3 is the scaling factor (exponent). A simple explanation of binary coding, specifically of the IEEE 754 standard representation for memory storage, is [20].

$$(12.625)_{10} = (1100.101)_2 \Leftrightarrow 1.100101 * 2^3 \tag{2}$$

- **Polynomial modulus degree** (*poly_modulus_degree*), which directly affects the number of coefficients in the

[1]The nth cyclotomic polynomial[32], for any positive integer n, is the unique irreducible polynomial with integer coefficients that divides $x^n - 1$ and does not divide $x^k - 1$ for any positive integer k coprime to n and less than n. Its roots are all the nth primitive roots of unity $e^{\frac{2i\pi k}{n}}$. This polynomial holds significant importance in the field of mathematics and finds applications in various areas including number theory [42], group theory, and cryptography. [45]

plaintext polynomials, the size of ciphertext elements, the computational performance of the scheme (higher is worse), and the level of security (higher is better).

- **Coefficient modulus sizes**, which is a list of binary sizes. Using this list, *SEAL* generates another list of prime numbers of those binary sizes called coefficient modulus q. In *TenSEAL*, as in *Microsoft SEAL*, each prime number in the coefficient modulus must be at most 60 bits and must be congruent to $1 \pmod{2 * poly\_modulus\_degree}$. q directly affects the size of ciphertext elements and the level of security. The list's length indicates the number of encrypted multiplications supported.

2) Encryption: In the second step, the data is encrypted using the public keys. The encryption is performed by adding noise to the data to ensure system security. Adding noise before encryption aims to make operations on encrypted data less predictable and prevent any correlation between these operations and the original data. The CKKS scheme will have output $m' = m + e$ of the decryption algorithm which is slightly different from the original message m, but these can be considered approximate values if the canonical embedding norm $\|e\|_{\infty}^{can}$ [2] is sufficiently small compared to $\|m\|_{\infty}^{can}$.

An intermediate rescaling step is performed during encryption to manage the magnitude (absolute value) of plaintexts. This intermediate step acts as a rounding step in standard approximate computations using floating-point numbers or scientific notation. This helps prevent noise created during encryption from accumulating during subsequent computations. The scaling algorithm divides a ciphertext by an integer to remove some inaccurate LSBs (least significant bit). The message's magnitude is maintained during homomorphic evaluation; thus the required size of the largest ciphertext modulus increases linearly with the depth of the circuit being evaluated.

3) Computation: The third step includes operations performed on the encrypted data (addition or multiplication).

4) Decryption: In the fourth step, the data is decrypted using the private keys and then decoded. The decryption removes the noise added in step 2 to recover approximate results. Noise suppression during decryption is an essential part of the homomorphic encryption process, with different approaches depending on the encryption scheme used.

D. Medical imaging

Our study uses medical data for several reasons:

- Medical data is highly sensitive data.
- Hospitals are a favorite target for hackers.
- Health-related artificial intelligence requires medical data from myriad sources to train well (multicentric study).
- Datasets are often unbalanced and more data is preferable, particularly for the minority class.

To bring us closer to the reality of the field, we had the opportunity to use a set of private data on breast cancer (prospective study) obtained from a Belgian hospital. More specifically, this consisted of mamographic abnormalities (malignant and benign tumours) confirmed by biopsy.

[2]The representation of these elements of the ring in the form of vectors is what we call embedding. The norm is then taken from the new representation. [29]

II. Related Works

Since federated learning can help support building accurate and robust models from medical data, the use of federated learning has been broadly studied in related literature. In this section, we focus on work in the detection of breast cancer, relevant federated learning architecture for medical imaging, and techniques that guarantee data privacy in this context.

a) Breast cancer detection: Different works propose to improve the detection of breast cancer using deep learning [17] [5] and a variety of imaging: mammography [39], histopathology [27], ultrasound [7], MRI [24], etc. Classifiers like support vector machine or K-nearest neighbors, based on a restricted set of features (e.g., 30 features of the Breast Cancer Wisconsin diagnostic dataset [8]), have been compared in literature [6], and a range of deep learning architecture has been proposed [44, 22].

b) Federated Learning for medical imaging: Federated learning is a promising way to improve access to the benefits of broad and impactful medical data and to unlock the potential of machine learning for digital health [38]. Two workflows are generally explored in literature: using either an *aggregation server* or a *peer to peer* approach [14]. Nguyen Tan et al. [43] proposed an FL framework for breast cancer classification based on mammography. They extracted the feature's image with ConvNet and used a linear classifier to make their prediction. Similarly, Jimenze-Sanchez et al. [30] used mammography and curriculum learning [9] in their FL architecture to improve the consistency of their local models. They demonstrated how their approach ensures better resilience against domain shift (i.e., different medical systems could induce a data heterogeneity which could hinder training). Finally, Li et al. [33] proposed a federated learning architecture based on the breakHis [41] dataset and homomorphic encryption.

c) Privacy concerns and solutions: While federated learning does not require directly sharing patient data among participants, research evidences global models trained via federated learning have sometimes memorized aspects of participant data [40]. The most common approach to guarantee a particular level of privacy is differential privacy [2] which has been demonstrated efficiently using histopathology images [4] with only a limited loss of accuracy. To avoid injecting noise into their model to fulfill differential privacy requirements, related federal learning architecture using secure multi-party computation [23] or homomorphic encryption [33] have also been proposed despite the higher computational cost.

III. Proposed Approach

Various methods exist to exploit homomorphic encryption. The first encrypts the data and trains the model on this encrypted data. However, this leads to significant memory consumption and incompatibility with certain layers or activation functions that do not support encrypted weights (e.g., division operations between encrypted results are not possible).

The second approach is to encrypt the model weights rather than the data. This prevents information being obtained from the model itself but does not protect the dataset.

We chose the second approach in our work with medical data. Our approach consists of federated learning with fully homomorphic encryption in which only the last layer of the model is encrypted (see section VI) while the principle remains the same for encrypting a complete model. We chose to use the MobileNetV2 architecture, a model that is both lightweight and high-performance, with fewer parameters than more complex architectures. Figure 5 shows its architecture and Section V offers more details.

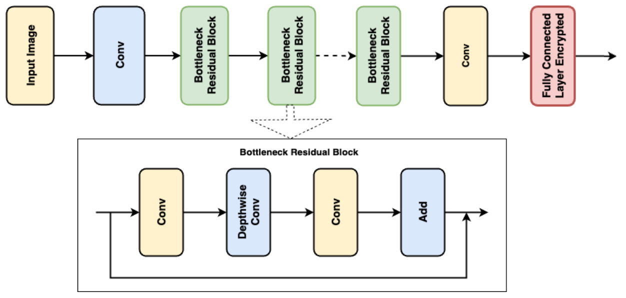

Input Image
Conv
Bottleneck Residual Block
Bottleneck Residual Block
Bottleneck Residual Block
Conv
Fully Connected Layer Encrypted
Bottleneck Residual Block
Conv
Depthwise Conv
Conv
Add

Fig. 5. MobileNetV2 architecture with FHE.

Before the federated learning phase begins, each participant must have the same set of keys for homomorphic encryption. Keys are transmitted physically to participants (by USB key) or via a different network to the one used for participant/server federation. After key generation, four steps repeat according to a number of server-specified *rounds*, described in Figure 6:

- **Initialization**: The server sends a copy of the central model to each participant selected for training. Since the connection protocol is gRPC (Google Remote Procedure Call), the data is serialized (converted into bytes) before being sent to the participants and then deserialized (converted back into real matrices) on the participant side.
- **Training (train and validation)**: If the weights received are not encrypted (*Plaintext*), training begins directly after updating the local model with the received weights. On the other hand, if the data is encrypted (*Cyphertext*), a decryption phase using the locally-held private key is carried out before the local model is updated.
- **Encryption**: Once the training phase is complete, each participant encrypts the model weights with the locally-held public key. As before, the data needs to be serialized and its values sent to the server.
- **Aggregation**: The server receives and then deserializes the weights from each participant. Next, the server performs the aggregation (in our case using the *FedAvg* strategy). The weights do not need to be decrypted to perform the aggregation; however, this makes server-side evaluation impossible. Thus there is no server-side evaluation phase when using homomorphic encryption.

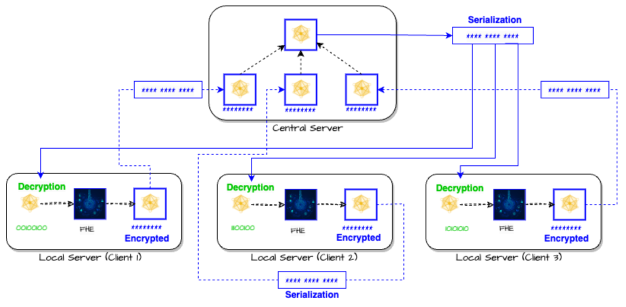


Fig. 6. Serialization and encryption/decryption phases in FL with FHE.

Homomorphic encryption creates new challenges.

Firstly, operations can only be carried out between values encrypted with the same private key or the results obtained after decryption are inconsistent. This necessitates sending the same key to each participant, thereby multiplying the number of vulnerable parties. If a malicious actor attacks a participant and obtains the private key (or if a participant is malicious), encrypted results will not be secure. Assuming that each participant is benevolent, all that needs to be done is for each participant to encrypt the common private key used in the FL process with their own private key. This limits the risk of an attack on the key, even though it will have to be decrypted before it can be used to encrypt/decrypt data, so there will still be a vulnerable phase on the server side.

Secondly, it is no longer possible to set up a global model evaluation phase on the server side because this would require the weights to be decrypted and therefore the server to have access to the private key, which would remove the security layer on the server side. For example, a malicious actor could carry out an inference attack after the server has decrypted the weights or simply retrieve the private key on the server side. The global model must be evaluated on the side of each participant once the participant copy has been received.

Lastly, homomorphic encryption is time and memory-intensive. This limits the complexity of the layers (number of parameters) to be encrypted.

IV. Technological Choices

A. Framework used

1) PyTorch: PyTorch is an open-source machine learning framework accelerating the path from research prototyping to production deployment [1]. Created by Facebook's research teams in 2016 [36], PyTorch performs tensor calculations needed for deep learning. Calculations are optimized and performed either by the processor (CPU) or, if possible, by a CUDA-supporting graphics processor (GPU).

2) Flower: Various frameworks are available for deploying the federated approach. For our experiments, we opted for Flower (1.5.0) [11]. Flower is an open-source framework dedicated to facilitating the deployment of machine learning algorithms in federated environments. Figure 7 shows the Flower architecture based on an aggregation server and several participants operating in different environments. Flower's most distinguishing feature compared with other federated learning frameworks is its support for a wide variety of machine learning libraries such as Pytorch and Tensorflow. This is thanks to Flower's agnostic implementation of the ML Framework. Flower is also flexible, allowing both single-host simulation and multi-host deployment of the federated procedure.

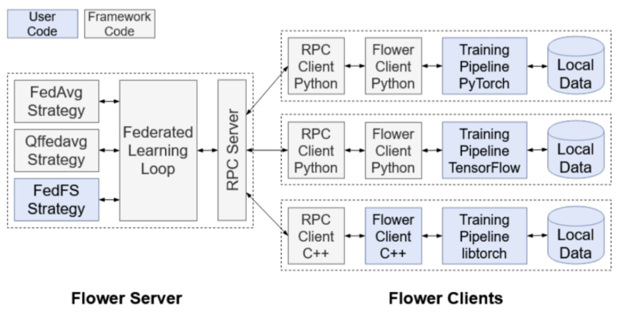


Fig. 7. Flower Framework architecture.

B. TenSEAL library

To implement homomorphic encryption, we used a library created by OpenMined called TenSEAL (version 0.3.14). TenSEAL is based on the SEAL API, which is the most widely used API for homomorphic encryption, and most homomorphic encryption libraries on Python are based on SEAL. The encrypted objects support addition, subtraction, and multiplication. It is therefore possible to average the encrypted values for aggregation using this library.

TenSEAL uses the BFV scheme for integers and the CKKS scheme for real numbers (section I-C), which is known to be a leveled homomorphic encryption scheme. This means that depending on our parameter selection, there is a limit to the number of multiplications we can perform on the encrypted data. This limit has either a direct impact on which machine learning model we can use or the selected model's depth. We don't need to use TenSEAL layers as an alternative to the operations performed on a standard model (such as the 2D convolution layer) because encryption only takes place after training and evaluation. We encrypt only the network weights and not the datasets. If we had wanted to encrypt the training and testing phases, we would have had to take into account that some machine learning models also use non-linear activation functions, which require approximation using polynomials in the case of CKKS. The TFHE scheme [16] solves this problem by allowing the evaluation of deeper models as well as non-linear activation functions.

The choice of parameters for the encryption key creation context is complex and greatly impacts performance (in terms of memory, time, and encryption and serialization accuracy). If the poly_modulus_degree is higher, the number of bits will be higher but the sum of coeff_mod_bit_sizes should be the same or smaller (example of TenSEAL context in Listing 1).

Listing 1. TenSEAL context example.

```python
def context(bits_scale=40, n=13, number=2, extrema=60,
galois=True):

    # Create TenSEAL context
    cont = ts.context(
        ts.SCHEME_TYPE.CKKS,

        # Integer setting
        # Number of values that can be encoded in
        a cipher element
        poly_modulus_degree=2 ** n,

        # List of bit size for each coefficient modulus.
        coeff_mod_bit_sizes=[extrema,*[bits_scale]
        *number, extrema]
    )

    # Set the scale
    # Manage the precision of values
    # in homomorphic calculations
    cont.global_scale = 2 ** bits_scale

    return cont
```

The choice of the global_scale (see section I-C1) is an important initial concern. The scale controls the precision of the fractional part, as it is the value by which *plaintext* is multiplied before being encoded into a polynomial with integer coefficients. This directly influences the memory and accuracy of the encryption and decryption.

Then, the multiplicative depth is controlled by the number of primes making up the coefficient module. All elements outside the extremities – so coeff_mod_bit_sizes$[1 : -1]$ – must be equal since TenSEAL supports the scaling of *ciphertexts*. It is advisable to choose a value close to $\log_2(2^{scale})$ for these values to minimize approximation errors. For example, for global_scale $= 2^{40}$, we would take coeff_mod_bit_sizes$[1 : -1] = 40$. In our case, the coefficient modulus contains primes of 60 bits for the extremes and $\log_2(2^{bits\_scale})$ bits for the others (to minimize approximation errors).

Finally, there are different types of security, such as classical security or quantum security, each with different levels of security (at 128 bits, 192 bits or 256 bits) and there are also different degrees of polynomial moduli accepted on SEAL, ranging from 1024 (2^{10}) to 32768 (2^{15}). The poly_modulus_degree 2^N (see section I-C1) must be able to perform all the homomorphic operations so the greater the number of homomorphic operations required, the higher N will be. For a given security level (e.g., 128 bits) and polynomial module degree (e.g., 8192), there is an upper bound on the number of bits in the coefficient module ($\sum$ coeff_mod_bit_sizes). If the upper bound is exceeded, it is necessary to use a higher polynomial module degree (e.g., 16384) to ensure that the required security level is still met. The upper limits for the other security levels supported by SEAL can be found in the official SEAL GitHub repository. [35]

We also deploy Galois keys to rotate the *ciphertext*. The current implementation of SEAL uses a prime number for noise control. For small N's (e.g., 1024, 2048), to ensure the system is secure, we cannot have many bits for the modulus, so SEAL does not generate Galois keys for these small N's.

V. Description of the experimentation

A. Model architecture

For our Pytorch model, we based ourselves on the mobilenetV2 architecture, which is pre-trained on imagenet (1000 classes). Here is a brief description of its main components:

1) **Features**:
   - **Conv2dNormActivation (layer 0)**:
     - 3x3 convolution with 32 filters, 2x2 stride.
     - Batch normalization.
     - ReLU6 activation.
   - **InvertedResidual Blocks (layers 1 to 17)**:
     - These blocks are the main feature of MobileNetV2.
     - Each block contains a sequence of convolutions, batch normalization, and ReLU6 activations.
     - The blocks use 1x1 convolutions, 3x3 convolutions, and depth-wise separable convolutions to reduce complexity and enhance efficiency.
     - Some blocks have shortcuts (skip connections) to facilitate gradient flow.
   - **Conv2dNormActivation (layer 18)**:
     - The last layer of features, a 1x1 convolution that increases the number of filters to 1280.
     - Batch normalization.
     - ReLU6 activation.
2) **Classifier**:
   - **Dropout (layer 0)**: A dropout layer with a rate of 0.2.
   - **Linear (layer 1)**: A fully connected layer that maps the output features to 1000 classes (for ImageNet).

However, for binary classification, we employed transfer learning by modifying the output layer (see in Listing 2), specifically the "Linear" layer (layer 1) of the classifier. We replaced it with:

- **"Linear" (layer 1)**: A fully connected layer that maps the 1280 output features to a reduced-dimensional space (320).
- **"Linear" (layer 2)**: The final fully connected output layer that maps the 320 output features to 2 classes.
- A **Softmax** function is added at the output.

Listing 2. Lightweight CNN used for FHE.

```
class MobileNet(nn.Module):
    def __init__(self, num_classes=2):
        super(MobileNet, self).__init__()
        self.model = models.mobilenet_v2(weights=
        'MobileNet_V2_Weights.DEFAULT')
        num_ftrs = self.model.classifier[-1].in_features
        self.model.classifier = nn.Sequential(
            self.model.classifier[0],
            nn.Linear(num_ftrs, int(num_ftrs / 4)),
            nn.Linear(int(num_ftrs / 4), num_classes),
            nn.Softmax(dim=1)
        )

    def forward(self, x):
        out = self.model.features(x)
        features = out.view(out.size(0), -1)
        out = self.model.classifier(features)
        return out
```

B. Private Dataset description

Thanks to an agreement with a Belgian hospital and UMONS University, our work makes use a private database of mammographic lesions divided between positive and negative classes in the context of breast cancer.

C. Federated learning configuration

Our FL configuration with the framework Flower ([11]):

- Experiments with different batch size;
- 50 epochs;
- Experiments with different numbers of rounds;
- We worked on a Mac M1;
- Training was done with MPS acceleration.
- The homomorphic encryption library doesn't support GPU acceleration on the Macbook Pro M1, so the encryption was done on the CPU.

VI. Experimental Results

As noted, several parameters in the encryption context influence performance. In a table, we summarise performance in terms of encryption time (not including serialization, deserialization, and decryption times). For example, the first row means that a single-valued tensor with a polynomial of order 2^{15} and modulus coefficients of size $[60, *[40] * 8, 60]$ was encrypted in less than a tenth of a second (Values greater than ten seconds are rounded to the nearest integer.)

For a given bits_scale, encryption is limited depending on the number of possible values. It is conceivable to increase the number of possible values that can be encrypted by reducing the number of intermediate values in the modulus coefficients or by reducing the order of the polynomial.

Increasing the size of the key (scale, number of polynomial coefficients, number of bits in the module coefficients) increases accuracy, but at the same time increases the memory and time required for encryption and serialization.

A. *Evaluation of the impact of a change in key size*

As explained earlier, there are many possible combinations at the key level. We can find a selection of the results in the table I. The first row corresponds to the heaviest key considered. Thus, by reducing the number of intermediate coefficients, we significantly decrease the encryption time.

Then, it is possible to increase the number of possible values to encrypt by decreasing Bits_scale. Thus, it becomes possible to encrypt almost four times more values by halving the Bits_scale.

Similarly, changing the values of extremas by dividing them by two allows encrypting nearly twice as many values. Finally, the choice of the polynomial degree is the last option to maximize the number of possible values to encrypt. We can observe that, with other fixed parameters, changing from 2^{15} to 2^{12} allows encrypting 8.5 times more values, making it possible to encrypt 683000 values when using the simplest possible key.

However, we cannot use homomorphic encryption for 683000 values because the most limiting aspect is memory at the data serialization level. Taking the parameters from the last row of the table to create the lightest encryption key possible, we can serialize only 39293 values, which is more than 17 times fewer values.

Unless otherwise stated, the results presented below were obtained with the first bolded row in the table. We will refer to it as the "heavy key." Additionally, we will have the opportunity to work with another lighter key, which we will refer to as such (second bolded row in the table). We couldn't choose the one allowing the encryption of the most values because it didn't allow representing all binary numbers from the last layer of the model.

Bits_scale	2^n	number	extremas	values	Time (s)
40	32768	8	60	12000	301
"	"	1	"	12000	20
30	"	8	"	17000	326
20	"	1	"	46000	231
"	"	"	30	80000	364
"	16384	1	"	172500	451
40	**8192**	**2**	**60**	135000	334
"	8192	"	"	340000	403
20	**4096**	**2**	**30**	400000	346
"	"	1	"	683000	428

TABLE I
SUMMARY OF THE PARAMETERS COMBINATION FOR THE KEY

B. *Time comparison*

It is necessary to serialize the encrypted values before we can send them to the server. The graph 8 compares the influence of encryption and serialization on the processing time as a function of the number of values contained in the tensor to be encrypted. We can encrypt a tensor of six thousand values in two seconds; however, the serialization takes longer, with six thousand values serialized in ten seconds. The serialization time is linearly related to the number of values in the tensor.

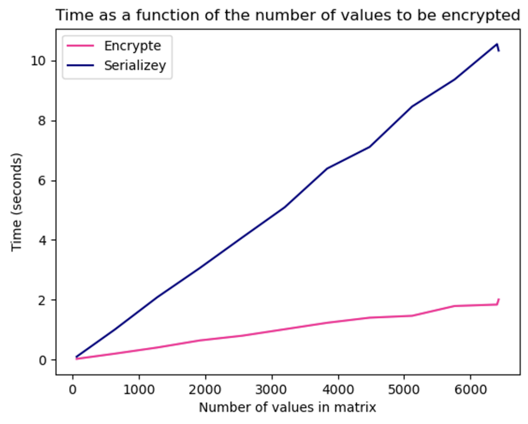


Fig. 8. Time comparison.

Figure 9 compares the duration of a federated learning simulation involving 20 participants and 10 epochs. Homomorphic encryption begins at the end of each participant's training session, regardless of the number of epochs. Conversely, as the number of rounds increases, the number of encryption/decryption phases also increases, resulting in a slight overall simulation duration increase. However, this increase is independent of the training time and would become negligible for a sufficient number of epochs. In our case of ten epochs, considering the entire homomorphic encryption process (encryption, serialization, and decryption), the difference is 28 minutes for twenty clients and fifteen cycles compared to federated learning without homomorphic encryption. If the number of epochs were increased, the training time would increase accordingly, but the total duration of the homomorphic encryption process would remain unchanged.

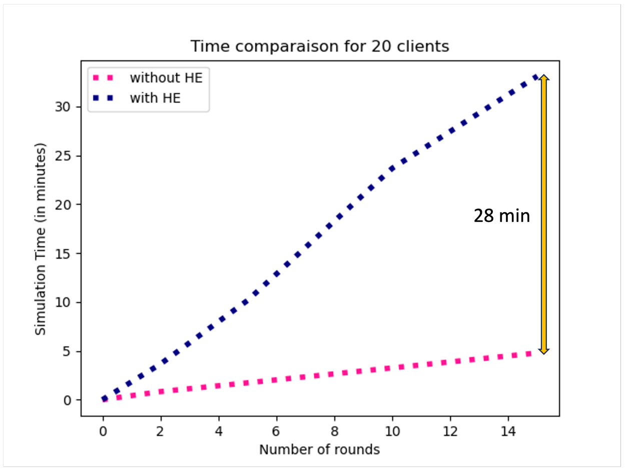


Fig. 9. Comparison of influence of encryption on total federated learning time

C. Memory comparison

The memory required for encryption increases linearly with the number of values used (reaching 4GB for a tensor of 6,000 encrypted values). The serialization phase also requires a significant amount of memory. The library has its own function for serializing tensors, but it has a limit in the function of the key size. With the key size specified (section VI-A), it was not possible in our work at this time to serialize tensors with more than 6424 values. Exceeding this number created an error. Beyond this limit inherent to the library, we are also constrained by the capabilities of our machine. In our scenario, we had 32 GB of RAM, but it was extended to 160 GB through the machine's automatic swapping system, allowing for the encryption of a larger set of values (see section VI-A). This limitation prevents homomorphic encryption on all layers of the model. In our model, only 2 layers of the network have too many parameters to be individually encrypted, but for a fixed client model, the accumulation of encrypted layers quickly reaches the memory limit. Furthermore, the server receives models from each client, so even if, in theory, it is possible to encrypt n-2 layers of a client's model, the server must also have sufficient memory capacity to receive it, which is not the case for us. Therefore, we decided to encrypt only the last layer of the model, so all the results presented below were obtained with a network in which only the last layer was encrypted. The next chapter discusses this decision.

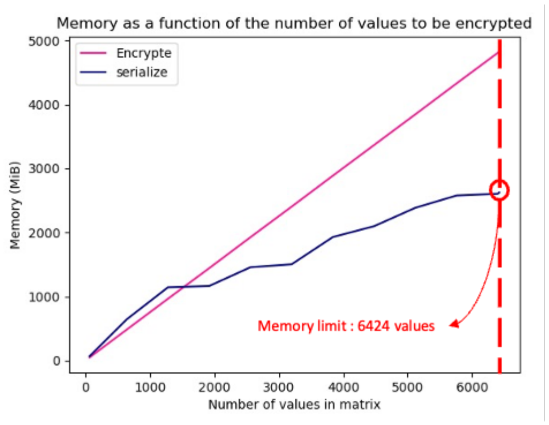


Fig. 10. Memory comparison.

D. Metrics comparison

The encryption adds a certain amount of noise to the data, so there will be a slight difference between the data before and after encryption (loss of precision). In terms of metrics, we can first compare accuracy with and without HE for different batch sizes (see Table II). The addition of noise with homomorphic encryption can be an advantage. In fact, in terms of accuracy, performance with homomorphic encryption doesn't necessarily worsen results. Thus we can see that for a batch size of 64, homomorphic encryption results in a 0.87% loss of accuracy, but on the contrary, for the batch of 32 images, homomorphic encryption significantly improves results, with an increase of almost 14%.

Accuracy	Without HE	With HE
Batch 16	70.6%	73.2%
Batch 32	73.8%	87.4%
Batch 64	83.4%	82.5%

TABLE II
CLASSIC FL VS FL WITH HE IN FUNCTION OF THE BATCH SIZE, FOR 3 ROUNDS AND 3 CLIENTS

An important metric for calculating the performance of a classification model in the medical world is the ROC curve (Receiver Operating Characteristic). Figure 11 compares ROC curves with and without homomorphic encryption for a batch of 64. In this particular case, we can see that the results are not as good when encrypted, with a loss of 7% at the micro average ROC curve. However, making the same comparison with the batch of 32, we have the opposite trend, with an 8% increase in the micro average ROC curve (see figure 12).

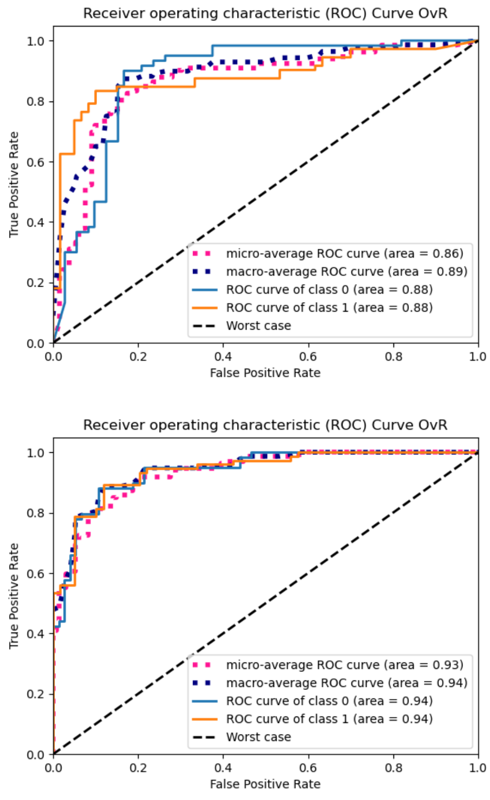
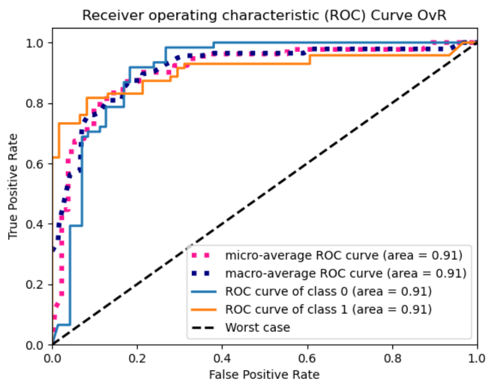


Fig. 11. ROC curve with (above) and without HE (below) for batch of 64

We might think that encrypting the last layer with a lighter key than the one used so far would degrade the results because the difference between the non-encrypted and decrypted data would be larger. However, this is not necessarily the case. In fact, if we repeat the process with a batch size of 32 but this time with a lighter encryption key, as shown in Figure 13, the results are not as good as with the heavier key (due to



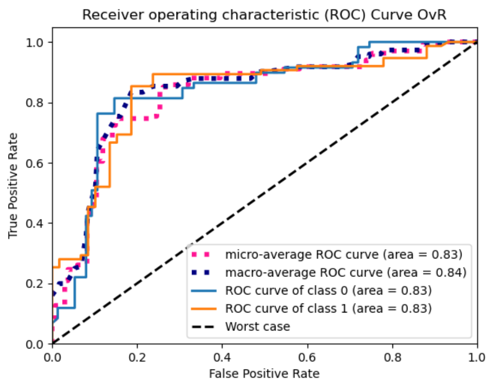


Fig. 12. ROC curve with (above) and without HE (below) for batch of 32

a loss of precision in the reconstruction of values) but they remain slightly better than the results without Homomorphic Encryption (HE). We assume that the loss of precision in the reconstruction of the decrypted value adds noise to the results, preventing the model from overfitting to the training data. Similarly, a smaller key will add more noise during decryption, so the decrypted values will be further away from the values that would have been obtained without HE.

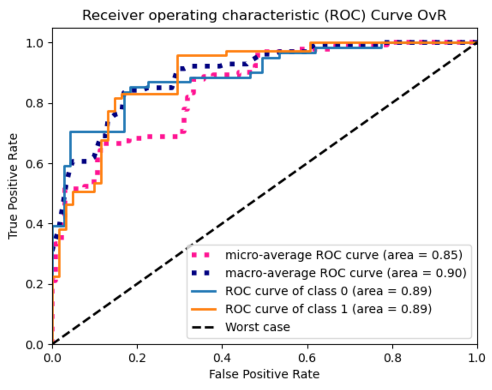


Fig. 13. ROC curve for the HE with a smaller key for batch of 32

Similarly, only the last layer has been encrypted, so one might think that encrypting a second layer would decrease the results. However, once again, this is not the case. Indeed, with a fixed key size (heavier key) and a batch fixed at 16, we tested encrypting four layers instead of just the last one. We observed that for a low number of epochs (11), we degrade the results compared to encrypting only one layer. We go from a ROC curve at 71% to a micro-average ROC curve at 67%. The difference is, however, smaller (and reversed) for a higher number of epochs (50), which takes us from 79% for one encrypted layer to 81% for four encrypted layers. We can, therefore, assume that the model adapts to the loss of precision caused by encryption over rounds and epochs.

VII. Conclusion and Future Work

In conclusion, our work has resulted in the implementation of a federated architecture integrating end-to-end fully homomorphic encryption (FHE) between participants and the central server. This encryption was specifically applied to the critical layers of the model without compromising the accuracy of the binary classification model (MobiteNetV2). Our experiments were conducted with anonymized medical images from a Belgian hospital, focusing on breast cancer research using positive and negative mammograms confirmed by biopsy. Our research highlighted a significant limitation in memory when applying fully homomorphic encryption (FHE) in the context of federated learning. The inability to encrypt the entire parameters of a deep learning model due to memory constraints was clearly identified. Despite this limitation, the study reveals that FHE maintains comparable performance in terms of ROC curves, emphasizing the robustness of this approach in high confidentiality contexts. These findings prompt a thorough consideration of the delicate balance between security and efficiency in federated learning. Our contribution lies not only in identifying the challenges but also in proposing pathways to more optimized solutions, thereby contributing to the advancement of secure machine learning in memory-constrained environments.

The most limiting point is linked to serialization (which is necessary when using the gRPC connection); we could look at alternatives to this, e.g., changing the connection framework.

Optimising a model that is already in use could open up other perspectives thanks to the knowledge distillation.

The key management of the architecture could be improved in several ways to increase the security of the model. As suggested in [10], a new private-public key pair could be generated at each round to reduce risks related to a compromised client. This proposal requires the implementation of secure communication channels, e.g., through TLS, between hospitals. On the client side, different measures [18] like key split could improve architectural resilience.

We work with the CKKS scheme but different schemes might improve performance. According to the literature [12], a CKKS scheme variant with bootstrapping offers more efficiency regarding CPU cost and number of levels consumed.

We could try alternatives to homomorphic encryption.

A good alternative for protecting communication between client and server (but which does not protect against attacks on the aggregator side) would be to use the Secure Multi-Party Computation (SMPC) cryptographic protocol, which enables collaborative computation between several parties without any of them being able to see each other's data (Shamir secret sharing). However, unlike homomorphic encryption, it does not protect against server-side attacks.

Another alternative is to use Trusted Execution Environments (TEEs), which are physically secure zones (a central processing unit) and therefore provide an isolated execution environment. This system can be used for secure remote computing and key encryption.

To prevent inference attacks, we can use differential privacy (DP), which randomly adds Gaussian noise to the model to prevent a malicious actor from discovering participants' private data through model retrieval.

We can use the checksum as an integrity verification mechanism to identify any malicious modifications to the model (malicious attacks) that may occur during the collaboration between clients and the central server.

Acknowledgement

This research was partially funded by the project AIDE and Infortech of UMONS. Special thanks to Linda Broughton of UCLouvain University for proofreading the English structure of this document and Dr Salvatore Murgo of the Belgian Helora hospital network for annotating mammographic abnormalities.

References

[1] PyTorch. URL: https://pytorch.org/ (visited on 09/17/2023).

[2] Martin Abadi et al. "Deep learning with differential privacy". In: (2016), pp. 308–318.

[3] Abbas Acar et al. "A Survey on Homomorphic Encryption". In: (2017). arXiv: 1704.03578 [cs.CR].

[4] Mohammed Adnan et al. "Federated learning and differential privacy for medical image analysis". In: Scientific reports 12.1 (2022), p. 1953.

[5] Guilherme Aresta et al. "Bach: Grand challenge on breast cancer histology images". In: Medical image analysis 56 (2019), pp. 122–139.

[6] Hiba Asri et al. "Using machine learning algorithms for breast cancer risk prediction and diagnosis". In: Procedia Computer Science 83 (2016), pp. 1064–1069.

[7] Anton S Becker et al. "Classification of breast cancer in ultrasound imaging using a generic deep learning analysis software: a pilot study". In: The British journal of radiology 91.1083 (2018), p. 20170576.

[8] Houssam Benbrahim, Hanaâ Hachimi, and Aouatif Amine. "Comparative study of machine learning algorithms using the breast cancer dataset". In: (2020). Advanced Intelligent Systems for Sustainable Development (AI2SD'2019) Volume 2-Advanced Intelligent Systems for Sustainable Development Applied to Agriculture and Health, pp. 83–91.

[9] Yoshua Bengio et al. "Curriculum learning". In: (2009), pp. 41–48.

[10] Charles-Henry Bertrand Van Ouytsel, Khanh Huu The Dam, and Axel Legay. "Symbolic analysis meets federated learning to enhance malware identifier". In: (2022), pp. 1–10.

[11] Beutel et al. "Flower: A friendly federated learning framework". In: (2021). On-device Intelligence Workshop at the Fourth Conference on Machine Learning and Systems (MLSys).

[12] Jean-Philippe Bossuat et al. "Efficient Bootstrapping for Approximate HE". In: (2020). https://eprint.iacr.org/2020/1203. URL: https://eprint.iacr.org/2020/1203.

[13] Xiaoyu Cao and Neil Zhenqiang Gong. In: (2022). MPAF: Model Poisoning Attacks to Federated Learning based on Fake Clients. arXiv: 2203.08669 [cs.CR].

[14] Ken Chang et al. "Distributed deep learning networks among institutions for medical imaging". In: 25.8 (2018). Journal of the American Medical Informatics Association, pp. 945–954.

[15] J.H. Cheon et al. "Homomorphic Encryption for Arithmetic of Approximate Numbers". In: Springer, Cham (Nov. 2017). URL: https://doi.org/10.1007/978-3-319-70694-8_15.

[16] Ilaria Chillotti et al. "TFHE: Fast Fully Homomorphic Encryption over the Torus". In: (2018). https://eprint.iacr.org/2018/421. URL: https://eprint.iacr.org/2018/421.

[17] Rayees Ahmad Dar, Muzafar Rasool, Assif Assad, et al. "Breast cancer detection using deep learning: Datasets, methods, and challenges ahead". In: Computers in biology and medicine (2022), p. 106073.

[18] Barker Elaine et al. "Recommendation for Key Management, Part 1: General". In: NIST Special Publication (2016), pp. 51–54.

[19] Junfeng Fan and Frederik Vercauteren. "Somewhat Practical Fully Homomorphic Encryption". In: (2012). https://eprint.iacr.org/2012/144. URL: https://eprint.iacr.org/2012/144.

[20] Floating point representation. URL: https://courses.engr.illinois.edu/cs357/sp2023/assets/lectures/complete-slides/4-Floating-Point.pdf (visited on 10/25/2023).

[21] Craig Gentry. "Fully Homomorphic Encryption Using Ideal Lattices". In: cs.cmu.edu (Jan. 2009). URL: https://www.cs.cmu.edu/~odonnell/hits09/gentry-homomorphic-encryption.pdf.

[22] Zhongyi Han et al. "Breast cancer multi-classification from histopathological images with structured deep learning model". In: Scientific reports 7.1 (2017), p. 4172.

[23] Seyedeh Maryam Hosseini et al. "Cluster Based Secure Multi-party Computation in Federated Learning for Histopathology Images". In: (2022), pp. 110–118.

[24] Qiyuan Hu, Heather M Whitney, and Maryellen L Giger. "A deep learning methodology for improved breast cancer diagnosis using multiparametric MRI". In: Scientific reports 10.1 (2020), p. 10536.

[25] Daniel Huynh. "CKKS explained: Part 1, Vanilla Encoding and Decoding". In: (Sept. 2020). URL: https://blog.openmined.org/ckks-explained-part-1-simple-encoding-and-decoding/.

[26] John Iwasz. "Microsoft SEAL et l'aube du chiffrement homomorphe". In: (Jan. 2023). URL: https://hackernoon.com/fr/Microsoft-Seal-et-1%27aube-du-chiffrement-homomorphe.

[27] Andrew Janowczyk and Anant Madabhushi. "Deep learning for digital pathology image analysis: A comprehensive tutorial with selected use cases". In: Journal of Pathology Informatics 7.1 (2016), p. 29. ISSN: 2153-3539. DOI: https://doi.org/10.4103/2153-3539.186902. URL: https://www.sciencedirect.com/science/article/pii/S2153353922005478.

[28] Marija Jegorova et al. "Survey: Leakage and Privacy at Inference Time". In: IEEE Transactions on Pattern Analysis 45 (2021), pp. 9090–9108. URL: https://api.semanticscholar.org/CorpusID:235731711.

[29] Robson R. de Araujo Jheyne N. Ortiz et al. "The Ring-LWE Problem in Lattice-Based Cryptography: The Case of Twisted Embeddings". In: entropy (Aug. 2021). URL: https://www.mdpi.com/1099-4300/23/9/1108.

[30] Amelia Jiménez-Sánchez et al. "Memory-aware curriculum federated learning for breast cancer classification". In: Computer Methods and Programs in Biomedicine 229 (2023), p. 107318.

[31] Georgios Kaissis et al. "Secure, privacy-preserving and federated machine learning in medical imaging". In: Nature Machine Intelligence 2 (June 2020). DOI: 10.1038/s42256-020-0186-1.

[32] Serge Lang. "Algebraic Number Theory". In: (1994). URL: https://eprint.iacr.org/2020/1118.

[33] Lingxiao Li, Niantao Xie, and Sha Yuan. "A Federated Learning Framework for Breast Cancer Histopathological Image Classification". In: Electronics 11.22 (2022), p. 3767.

[34] Lingjuan Lyu et al. "Privacy and Robustness in Federated Learning: Attacks and Defenses". In: (2022). arXiv: 2012.06337 [cs.CR].

[35] Microsoft SEAL. URL: https://github.com/microsoft/SEAL/blob/master/native/src/seal/util/hestdparms.h (visited on 09/28/2023).

[36] PyTorch: all about Facebook's Deep Learning framework. DataScientest. URL: https://datascientest.com/en/pytorch-all-about-this-framework#:~:text=Based%20on%20the%20former%20Torch,a%20simple%20and%20efficient%20way. (visited on 09/17/2023).

[37] "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of individuals with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)". In: (2016). URL: http://data.europa.eu/eli/reg/2016/679/oj (visited on 03/02/2020).

[38] Nicola Rieke et al. "The future of digital health with federated learning". In: NPJ digital medicine 3.1 (2020), p. 119.

[39] Li Shen et al. "Deep learning to improve breast cancer detection on screening mammography". In: Scientific reports 9.1 (2019), p. 12495.

[40] Reza Shokri et al. "Membership inference attacks against machine learning models". In: (2017), pp. 3–18.

[41] Fabio Alexandre Spanhol et al. "Breast cancer histopathological image classification using convolutional neural networks". In: (2016), pp. 2560–2567.

[42] Lawrence Sun. "Cyclotomic Polynomials in Olympiad Number Theory". In: (Feb. 2013).

[43] Y Nguyen Tan et al. "A Transfer Learning Approach to Breast Cancer Classification in a Federated Learning Framework". In: IEEE Access 11 (2023), pp. 27462–27476.

[44] Ankit Titoriya and Shelly Sachdeva. "Breast cancer histopathology image classification using AlexNet". In: (2019), pp. 708–712.

[45] Eric Weisstein. Cyclotomic Polynomial. MathWorld–A Wolfram Web Resource. URL: https://mathworld.wolfram.com/CyclotomicPolynomial.html.