Learning in the Dark: Privacy-Preserving Machine Learning using Function Approximation

1st Tanveer Khan
Department of Computing Sciences
Tampere University
Tampere, Finland
tanveer.khan@tuni.fi

2nd Antonis Michalas
Department of Computing Sciences
Tampere University, Finland and
RISE Research Institutes of Sweden
antonios.michalas@tuni.fi

***Abstract*—Over the past few years, a tremendous growth of machine learning was brought about by a significant increase in adoption and implementation of cloud-based services. As a result, various solutions have been proposed in which the machine learning models run on a remote cloud provider and not locally on a user's machine. However, when such a model is deployed on an untrusted cloud provider, it is of vital importance that the users' privacy is preserved. To this end, we propose Learning in the Dark – a hybrid machine learning model in which the training phase occurs in plaintext data, but the classification of the users' inputs is performed directly on homomorphically encrypted ciphertexts. To make our construction compatible with homomorphic encryption, we approximate the ReLU and Sigmoid activation functions using low-degree Chebyshev polynomials. This allowed us to build Learning in the Dark – a privacy-preserving machine learning model that can classify encrypted images with high accuracy. Learning in the Dark preserves users' privacy since it is capable of performing high accuracy predictions by performing computations directly on encrypted data. In addition to that, the output of Learning in the Dark is generated in a blind and therefore privacy-preserving way by utilizing the properties of homomorphic encryption.**

***Index Terms*—Activation Function, Homomorphic Encryption, Neural Networks, Polynomial Approximation, Privacy,**

I. Introduction

Machine Learning (ML), specifically Deep Learning (DL), has garnered significant attention from researchers due to its solid performance in many tasks, such as speech recognition, spam detection, image classification, traffic analysis, face recognition, financial detection, and genomics prediction [1], [2], [3], [4], [5], [6]. To meet the growing demand for ML services, Cloud Service Providers (CSPs) such as Google Prediction API [7], Microsoft Azure ML [8], and Ersatz Lab [9] also offer Machine Learning as a Service (MLaaS), enabling users to train and test the ML models using the CSP infrastructure. Typically, these models involve a training phase where the model learns from a dataset and a testing phase where the model predicts outputs based on unseen inputs. Once the model is trained and deployed on the CSP, the users can use it for online prediction services. However, the adoption of MLaaS raises concerns about the privacy of data being outsourced, in sensitive domains such as finance and healthcare [10]. There is a risk of data misuse or theft when sending data to prediction models hosted by CSPs. To address these privacy concerns, researchers proposed various methods to protect user data in MLaaS settings [11], [12], [4], [13], [14].

This work aims to demonstrate the application of Neural Network (NN) on Encrypted Data (ED) using Homomorphic Encryption (HE). HE allows performing arithmetic operations (addition and multiplication) over ED without decryption, enabling the homomorphic evaluation of functions relying on these operations. More specifically, our focus is to evaluate the Convolution Neural Network (CNN) on ED, where most operations, except for Non-linear Activation Functions (NLAF), can be homomorphically evaluated.

Enabling the homomorphic evaluation of CNNs on ED has been an active area of research, with significant efforts dedicated to designing efficient support for NLAFs [15]. Various approaches have been proposed, including the utilization of power functions [4], look-up table [16], and polynomial approximations [17], [18], [19]. In this work, we employ low-degree Chebyshev polynomials to approximate NLAF.

A. Background on Polynomial Approximations

Approximating continuous functions is a problem that has drawn mathematicians' attention for a very long time. While there are several ways to approximate a continuous function, in this work we are only interested in polynomial approximations. More specifically, we are using Chebysev polynomials to approximate the Sigmoid and the ReLU functions. However, there are various works that use different approaches such as the x^2 function [4], the *Piecewise* approximation [18], lookup tables [16] etc. Unfortunately, all these methods face certain limitations. For example, the x^2 method can cause instability during the training phase and the creation of a piecewise linear approximation can sometimes be a complex optimization problem. With this in mind, we chose to work with Chebyshev polynomials. The general form of these polynomials is:

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \tag{1}$$

where $T_n(x)$ represents a polynomial of degree n. Chebyshev polynomials allow us to efficiently compute any continuous

This work was funded by the Technology Innovation Institute (TII) for the project ARROWSMITH and from Horizon Europe for HARPOCRATES (101069535).

function in a given interval, using only low-degree polynomials. This feature significantly boosts efficiency and lower the overall computational complexity.

B. Our Contribution

The main contributions of this paper are manifold.

- We show how to approximate NLAFs like ReLU and Sigmoid using Chebyshev polynomials. By substituting these NLAFs with the Chebyshev polynomials, we conduct a comprehensive analysis to compare the differences in terms of efficiency and accuracy.
- We design a PPML model in which the CNN is trained on plaintext data while the classification process operates on homomorphically ED.
- To illustrate the effectiveness of our model, we conducted extensive experiments and provided a comparative analysis with other state-of-the-art works in the field of PPML.
- We designed a protocol that demonstrate the practical application of our PPML model in a realistic scenario while ensuring its security under malicious threat model.

C. Organization

The rest of the paper is organized as follows: In Section II, we present important published works in the area of PPML. In Section III, we provide the necessary background information needed for our construction. Then, in Section IV, we show how to approximate the ReLU and Sigmoid AFs using low-degree Chebysev polynomials. The methodology of our work is illustrated in Section V, followed by extensive experimental results in Section VI. In Section VII, we design a protocol, that demonstrates the applicability of our work and finally, in Section VIII we conclude the paper.

II. Related Work

The first step in preserving the privacy of the ML model is achieved through Multiparty Computation (MPC). This approach allows parties jointly compute a function while keeping the original inputs private. Several methods based on MPC have been proposed for preserving the privacy of ML models, such as K-means clustering, linear regression, SVM classifier, Decision tree, etc. [20], [21], [22], [23], [24].

One approach called SecureML, designed by Mohassel *et al.,* [25], uses a two-server model in which data is distributed among two non-colluding servers. It is an efficient protocol for preserving the privacy of various ML models using MPC. These servers train various models on the joint data using secure MPC with support for approximating Activation Functions (AF) during training. Since SecureML requires changes in the training phase, the model does not apply to the problem of making the existing NN model oblivious. Another approach MiniONN [26], converts any NN into an oblivious NN using MPC providing privacy-preserving predictions. While MiniONN uses cryptographic primitives, such as garbled circuits and secret sharing, it still reveals information about the network (e.g. size of the filter) [27]. MOBIUS is another secure prediction protocol for binarized NN [28], allowing fast and scalable PPML model by delegating a protected model to a resource provider. The resource provider offers prediction to client without knowing anything about client's input.

Due to the high communication cost associated with MPC techniques mentioned above, alternative methods using HE have been explored. Wu *et al.* [29] proposed a privacy-preserving logistic regression model. As the logistic function is *not* linear, the authors use polynomial fitting to achieve a good approximation. However, it lowers the accuracy of model. Graepel *et al.* [30] used Somewhat Homomorphic Encryption (SHE) [31] to train two simple classifiers on ED, employing low-degree polynomials for efficient computations.

Ehsan *et al.* [32] proposed a technique based on Leveled HE (LHE) [33] to preserve the privacy of CNN while at the same time keep the accuracy as close as possible to the original model. They approximated the Sigmoid, ReLU and Tanh functions and achieved an accuracy of 99.52% on MINST dataset [34]. This is good, as the accuracy of the original model was measured at 99.56%. Unfortunately, their approach is computationally expensive, as the training and testing phases are both performed on ED.

In [35], the authors present Fast Homomorphic Evaluation of Deep Discretized NN (FHE-DiNN). Their design utilizes HE to evaluate an NN. The user encrypts the data using HE and transfers it to the cloud. The cloud blindly classifies the ED using HE and sends the ED back to the user. Upon reception, the user uses her secret key to decrypt it. In this scheme, the encryption parameters are dependent on the model structure. So, if the server updates its model, the client is forced to re-encrypt all of its data. While communication-wise HE schemes are very efficient, the computation cost at the server-side is very large.

A notable related work is CryptoNets [4] which applies an NN model to ED. While CryptoNets achieves remarkable accuracy, the construction is based on the use of square AF. Hence, approximating a non-linear function causes instability during the training phase when the interval[1] is large. In our work, we address this issue by using Chebyshev approximation, which accurately approximates AFs even in larger intervals. Additionally, we adapt an approach where the client's input is encrypted but the model remains in plaintext, aiming for better efficiency in the classification process.

III. Preliminaries

Notation: If x and y are two strings, by $x||y$ we denote the concatenation of x and y. A *probabilistic polynomial time* (PPT) adversary $\mathcal{ADV}$ is a randomized algorithm for which there exists a polynomial $p(\cdot)$ such that for all input x, the running time of $\mathcal{ADV}(x)$ is bounded by $p(|x|)$. A neuron is a mathematical function that takes one or more inputs, multiplies them by some values called "weights" and adds them together. This value is then passed to NLAF, to become neuron's output.

[1]By interval we mean the domain of definition of the AF.

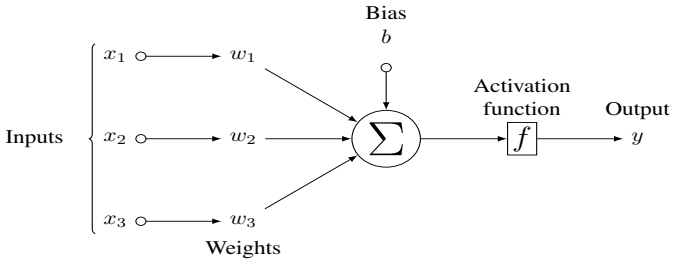


Fig. 1. Structure of a Neuron in a Neural Network

A. Convolutional Neural Network (CNN)

A typical NN is a combination of neurons arranged in layers. Each neuron receives input from other neurons with an associated weight w and a bias b, as shown in Figure 1. It then uses equation 2 to compute some function f on the weighted sum of its input. The output of this neuron is given as input to other neurons.

$$y = f\left(\sum_{i=1}^{3} x_i w_i + b\right) \tag{2}$$

In equation 2, x_i is the input, w_i is the weight, b is the bias term and f is the AF.

In our work, we focus on CNN, a deep NN algorithm primarily used for *image classification*. In CNN, each input passes through a series of layers during the training and testing phases[2]. These layers consist of convolutional layers (Conv), AFs, pooling layers, Fully Connected (FC) layers and a softmax layer [36]:

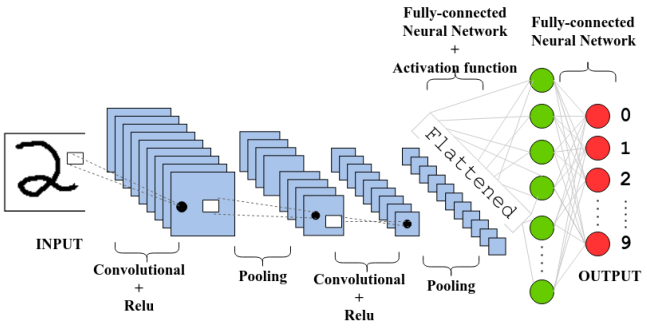


Fig. 2. Convolutional Neural Network

- Convolution Layer: Conv is the first layer in CNN and acts as a feature detector (used for feature mapping). To generate a feature map, convolution is performed by moving the filter over the input with a certain stride. On a single input, multiple convolutions can be performed using numerous filters to extract more than one feature from the input. Also, padding is performed to make size of convolved features same as that of the input.

[2]https://shorturl.at/nzHK1

- Activation Function: In NN, all operations are linear except the AF. These functions are used to introduce non-linearity in the network. The most commonly used AFs are Sigmoid, ReLU and Tanh, as shown in the Table I.
- Pooling: This layer is responsible for extracting the dominant features (maximum or average pixel values) to reduce the size of the input image. The popular pooling operations are *max-pooling* and *average-pooling*. In max-pooling, the maximum value, and in average pooling, the average values are extracted from the part of the image covered by the filter.
- Flattening: It convert data into a 1-dimensional array that is given as input to the next layer. There, the image matrix is converted into a vector and feed to a FC NN.
- Fully Connected: The FC layers are activated at the last phase of the process after Conv, pooling and AFs. This layer connects every neuron in one layer to every neuron in the next layer and performs a weighted sum of the inputs and add bias.
- Softmax: In a classification problem, softmax is the final output layer with discrete class labels. It assigns a probability to each class that adds up to 1. The node having the highest probability is determined to be the most likely class for the given input.

Our CNN (Figure 2) consists of two Conv with a ReLU AF, two pooling operations, two FC layers and a softmax layer.

TABLE I
DIFFERENT ACTIVATION FUNCTIONS.

Name	Function	Derivative	Figure
ReLU	$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0. \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 1 \geq 0. \end{cases}$	
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))^2$	
Tanh	$f(x) = \frac{e^{x}-e^{-x}}{e^{z}+e^{-z}}$	$f'(x) = 1 - f(x)^2$	

B. Homomorphic Encryption (HE)

HE is an encryption scheme that allows users to perform computations on ED. Given two ciphertexts c and c', a user can compute $f(c, c')$ where f is a function associated either with an addition or multiplication operation. A typical HE scheme consists of the following four algorithms:

- $\mathsf{HE.KeyGen}(1^\lambda) \rightarrow (\mathsf{pk}, \mathsf{sk})$: The key generation algorithm takes as input a security parameter λ and outputs a public/private key pair (pk, sk).
- $\mathsf{HE.Enc}(\mathsf{pk}, m) \rightarrow c$: This algorithm takes as input a pk and a message m and outputs a ciphertext c.

- HE.Eval(pk,$f, c, c^{'}$) $\rightarrow c_{eval}$: This algorithm takes as an input two ciphertexts c and $c^{'}$, a pk and a homomorphic function f and outputs an evaluated ciphertext c_{eval}.
- HE.Dec(sk,c) $\rightarrow m$: The decryption algorithm takes as input a private key sk and c_{eval} and outputs $f(m, m')$.

Currently, there are three different kinds of HE schemes; Partial HE (PHE), Fully HE (FHE) and somewhat HE (SHE). PHE allows users to perform an *unlimited number* of operations on the ciphertexts [37], [38]. However, they support only one type of operation (either addition or multiplication) and hence, are not suitable for our work. Furthermore, while FHE schemes offer the possibility to perform an unlimited number of both additions and multiplications [39], they are computationally expensive [40]. As a result, we choose to work with SHE that offers similar functionalities as FHE but in a more efficient manner [31], [41]. The key difference between FHE and SHE is that in SHE schemes users can only perform a limited number of operations.

IV. Chebyshev Polynomials

In this section, we show how low-degree Chebyshev polynomials can be utilized to approximate the AFs. As mentioned in [42], using these polynomials the AFs can be approximated at a given interval. The first few Chebyshev polynomials are given below while their generalization is given in equation 1.

$$T_0(x) = 1, T_1(x) = x, T_2(x) = 2x^2 - 1, T_3(x) = 4x^3 - 3x$$

Chebyshev approximation is also known as the minimax approximation. The minimax polynomial approach is used for function approximation by improving the accuracy and lowering the overall computational complexity [43]. Instead of minimizing the error at the point of expansion like Taylor's polynomial approximation, the minimax approach minimizes the error across a given input segment. The minimax approximation is used to find a mathematical function that minimizes the maximum error. As an example, for a function f defined over the interval $[a, b]$, the minimax approximation finds a polynomial $p(x)$ that minimizes $\underset{a \leq x \leq b}{max} |f(x) - p(x)|$.

The first order minimax polynomial is defined as:

$$p(x) = c_0 + c_1 x \approx f(x)$$

where c_0 and c_1 are the coefficients of the polynomial.

A. Chebyshev Approximation

To approximate a continuous function f, defined over $[a, b]$, we first need to express f as a series of Chebyshev polynomials at $[-1, 1]$. More precisely, f is expressed as: $f(x) = \sum_{k=0}^{n} c_k T_k(x), \quad x \in [-1, 1]$, where c_k is the Chebyshev coefficient and $T_k(x)$ can be calculated from equation 1. As a next step, we calculate the coefficients of the polynomial and finally, express the polynomial in the original interval $[a, b]$. This procedure is illustrated in algorithm 1.

Our results for approximating Sigmoid and ReLU, using algorithm 1, are illustrated in Table II. The approximation error for both AFs is calculated using equation $E(x) = f(x) - p(x)$.

Algorithm 1: Chebyshev Polynomial Approximation

Input: $c_k, f(x), T_k(x)$
Output: $p(x)$
1 Express f as: $f(x) = \sum_{k=0}^{n} c_k T_k(x), \quad x \in [-1, 1]$
2 Chebyshev coefficients $c_k = \frac{2}{\pi} \int_{-1}^{1} f(x) \frac{T_k(x)}{\sqrt{1-x^2}}$
3 Approximation domain: from [-1, 1] to [a, b]: $x = \frac{a+b-2z}{b-a}$, $z \in [a, b]$

TABLE II
Approximating Sigmoid and ReLU Activation Function

Approximation: Sigmoid				
x	Interval	Function(x)	Approximation	Difference
-4	[-5, 5]	0.017986	0.016360	-1.63e-03
-3	[-5, 5]	0.047426	0.049098	1.67e-03
-2	[-5, 5]	0.119203	0.118340	-8.63e-04
-1	[-5, 5]	0.268941	0.268522	-4.19e-04
1	[-5, 5]	0.731059	0.731478	4.19e-04
2	[-5, 5]	0.880797	0.881660	8.63e-04
3	[-5, 5]	0.952574	0.950902	-1.67e-03
4	[-5, 5]	0.982014	0.983640	1.63e-03
Approximation: ReLU				
-4	[-5, 5]	0.000000	-0.008871	-8.87e-03
-3	[-5, 5]	0.000000	0.014340	1.43e-02
-2	[-5, 5]	0.000000	-0.015085	-1.51e-02
-1	[-5, 5]	0.000000	-0.026883	-2.69e-02
1	[-5, 5]	1.000000	0.973117	-2.69e-02
2	[-5, 5]	2.000000	1.984915	-1.51e-02
3	[-5, 5]	3.000000	3.014340	1.43e-02
4	[-5, 5]	4.000000	3.991129	-8.87e-03

V. Methodology

We start this section by describing our system model. We assume a client-server model involving the following entities:

- *Users*: We consider users who own a list of images and wish to use a cloud-based ML service to classify them in a privacy-preserving way (i.e. without revealing anything about the content of the images to the cloud).
- *Cloud Service Provider (CSP)*: The CSP can receive a large number of *encrypted* images from users and classify them in a privacy-preserving way by giving them as input to a ML algorithm.

The topology of our work is illustrated in Figure 3.

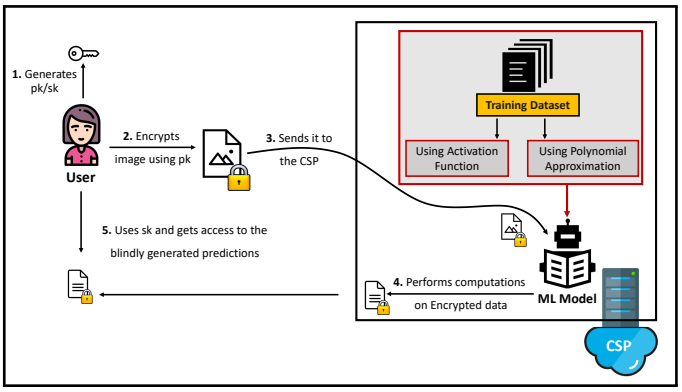


Fig. 3. Learning in the Dark High Level Overview

In our model, we consider a CNN capable of analyzing large volumes of data (images) in a variety of domains. The CNN is deployed in a privacy-preserving manner in the CSP. To

1) *Conv*: Input image 28 × 28, Window size 5 × 5, Stride(1,1), Number of input channels 1, number of output channels 5, Filters 5, Output 28 × 28 × 5.
2) *Activation Function*: ReLU .
3) *Pooling*: Mean, Window size 2 × 2 × 1, Stride(1,1), Output 14 × 14 × 5.
4) *Conv*: Window size 5 × 5, Stride(1,1), Filters 10, Output 14 × 14 × 10.
5) *Activation Function*: ReLU .
6) *Pooling*: Mean, Window size 2 × 2 × 1, Stride(1,1), Output 7 × 7 × 10.
7) *Fully Connected*: Fully connects the incoming 7 × 7 × 10 nodes to the outgoing 128 nodes.
8) *Activation*: ReLU
9) *Fully Connected*: Fully connects the incoming 128 nodes to the outgoing 10 nodes.
10) *Softmax*: Generate a probability for 10 nodes

Fig. 4. Convolutional Neural Network for Training Phase

preserve the privacy of users data, we use HE. Using an HE scheme allows us to perform computations on ED. However, HE schemes face certain limitations as they only support addition and multiplication operations. Most of the operations in a CNN are simple additions and multiplications and can thus be evaluated using HE. However, AFs are non linear and as a result, we cannot use HE to perform operations on them. To this end, we replace the AFs with polynomial approximations as already discussed in section IV. While higher degree polynomials would provide us better approximation, they also introduce higher computation and communication costs and hence, render our construction inefficient.

Flow: The CNN model is deployed in the CSP and is trained using plaintext data. The weights and biases for this model are measured and made available to the CSP. For the training phase, we use the CNN given in Figure 4. The user generates a public/private key pair for the HE scheme, encrypts an image and sends it to the CSP. Upon reception, the CSP runs the ML model and performs the classification in a privacy-preserving way.

A. Inference Phase

Although, the operations performed in the inference phase are nearly the same as in the training phase, there are few fundamental differences. For example, all operations in the inference phase are taking place on ED while in contrast to the training phase where plaintext data is used. Similarly, the softmax which is part of the training phase is no longer available in the inference phase as shown in Figure 5.

For the inference phase, we use the Fan-Vercauteren SHE scheme [31]. The reason for using this specific scheme is that it allows us to perform *both* addition and multiplication. It is important to note that this scheme has three important parameters that affect the security level, and its performance:

- Polynomial Modulus: This is an important parameter that affects the security level of the scheme. Polynomial modulus uses a power of two cyclotomic polynomial [44] and the recommended degrees for these polynomials are 1024, 2048, 4096, 8192 and beyond. On one side, a higher degree gives more security to the scheme while on the other side it degrades its performance.
- Coefficient Modulus: This parameter determines the Noise Budget (NB) in the encrypted ciphertext. The coefficient modulus is directly proportional to NB and inversely proportional to the security level of the scheme.
- Plaintext Modulus: The plaintext modulus affects NB in the freshly encrypted ciphertext. Additionally, it affects the NB consumption of homomorphic multiplications. For good performance, the recommendation is to keep the plaintext modulus as small as possible.

Each ciphertext in this encryption scheme has a specific quantity called NB – measured in bits. The NB is determined by the above parameters and consumed by the homomorphic operations. The consumption of the NB is based on the chosen encryption parameters. For addition operations, this budget consumption is almost negligible in comparison to multiplication operation. In sequential multiplication that occurs at the Conv and FC layer, the consumption of NB is very high. Hence, it is important to reduce the multiplicative depth of the circuit by considering appropriate encryption parameters. Once the NB drops to zero, then the decryption of ciphertext is not possible. Therefore it is necessary to choose the parameters to be large enough to avoid this, but not so large that it becomes ineffective and non functional.

While the HE scheme is based on polynomials, user's input is provided as a real number. Therefore, there is a clear mismatch between the two. Hence, it is important to use an encoding scheme that maps one to the other. To this end, the user encodes the input using the plaintext modulus and then encrypts it using the public key. The user generates encryption parameter and shares it with CSP. To perform computations on ED, CSP must have access to these parameters.

Using calculated weights and biases from the training phase and encryption parameters, CSP runs the inference phase on encrypted image. The inference network is same as training network except that AFs are replaced by polynomial approximation and are built using an HE function.

The AFs are substituted by polynomials. Since these polynomials only have addition and multiplication operations that are supported by HE. Consequently, we can perform encrypted computations on these functions. Similarly, pooling operation in the inference phase is straightforward – calculate the average value of four ciphertexts and multiply it with appropriate values. However, Conv is a bit expensive in terms of NB as it is a sequence of multiplication operations.

Softmax is not a part of the inference network and CSP use it to perform computation on ED and obtains an encrypted output. The CSP does not have access to the secret key and thus cannot access the result. Furthermore, as the softmax layer is removed from the inference network the CSP is not able to predict the final output of the layer.

At the end, the encrypted result of the output layer – an array of 10 values which are homomorphically encrypted – is sent back to the user. The user decrypts the results using the secret key and finds the output of the model which is the index corresponding to the highest among the 10 values.

1) *Conv*: Input image 28 × 28, Window size 5 × 5, Stride(1,1), Number of input channels 1, number of output channels 5, Filters 5, Output 28 × 28 × 5.
2) *Activation Function*: Approximated using polynomial approximation.
3) *Pooling*: Mean, Window size 2 × 2 × 1, Stride(1,1), Output 14 × 14 × 5.
4) *Conv*: Window size 5 × 5, Stride(1,1), Filters 10, Output 14 × 14 × 10.
5) *Activation Function*: Approximated using polynomial approximation.
6) *Pooling*: Mean, Window size 2 × 2 × 1, Stride(1,1), Output 7 × 7 × 10.
7) *Fully Connected*: Fully connects the incoming 7 × 7 × 10 nodes to the outgoing 128 nodes.
8) *Activation*: Approximated using polynomial approximation.
9) *Fully Connected*: Fully connects the incoming 128 nodes to the outgoing 10nodes.

Fig. 5. Convolutional Neural Network for Inference Phase

At this point, it is important to highlight that the user utilized the ML model offered by the CSP and received the results without getting any valuable information about the underlying model. Similarly, the CSP ran the model on the encrypted image but at the same time was unable to extract any valuable information either for the content of the image or the actual prediction that sent back to the user. Hence, our model is considered as a privacy-preserving one.

VI. Performance Evaluation

We present our experimental results. In the first part, we provide experimental results on function approximation using Chebyshev polynomials. Then, we evaluate the performance of the proposed ML model and compare it with CryptoNETs.

*a) **Experimental Setup**:* All experiments were conducted in Python 3 using Ubuntu 18.04 LTS 64 bit (Intel Core i7, 2.80 GHz, 32GB). For the training phase, we used Tensor flow to train our CNN model, while the actual experiments for that phase were conducted on Google Colab (with GPU enabled). Finally, for the inference phase we used Microsoft's Simple Encrypted Arithmetic Library (SEAL) [45].

*b) **Dataset**:* To evaluate our model, similar to other works in the area, we used the MNIST dataset [34] which consists of 60,000 images of handwritten digits. To train our CNN model we used 50,000 images while the rest 10,000 were used for testing. Each image is 28×28 pixel array and is represented by its gray level in the range of 0-255.

A. Activation Function Approximation

As we mentioned in the previous sections, in our approach we use Chebyshev polynomials to approximate the ReLU and Sigmoid AFs where inputs are images encrypted with an SHE scheme. The polynomial approximation of the ReLU AF is shown in Table III while for Sigmoid in Table IV. Since the choices of the degree and the interval affect the performance of the model, it is necessary to choose suitable parameters. For this purpose, we conducted a series of experiments using different degrees and intervals. As can be seen in Table III, the AFs are more accurately approximated when using polynomials of higher degree in small intervals. For example, the polynomial having degree 9 and interval $[-10, 10]$ more accurately approximate the ReLU function than the rest of the polynomials. The same applies to the Sigmoid AF, where a high degree 9 and small interval $[-10, 10]$ give a better approximation as can be seen in Table IV. However, the use of higher degree polynomials introduces a significant computation overhead, and small intervals limit the use of the approximation function. The results for approximating the Sigmoid AF using low-degree Chebyshev polynomials are presented in Table IV.

Furthermore, we performed a plethora of different experiments on the CNN model. We trained different networks by increasing the size of the Conv and the size of the filters. We noticed that changing the number of Conv and filters affects the overall accuracy of the network. As the size of the filter and layer increases, the accuracy of the network also increases. However, the efficiency of the network drops significantly. Hence, for the training phase, we considered the network given in Figure 4. First, we trained the CNN model using the ReLU AF. The measured accuracy for that part was 99.2%. Then the same network was trained using the polynomial approximation function where we got an accuracy of 98.5% – a result that is very close to the original AF.

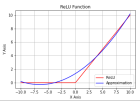
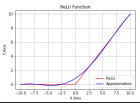
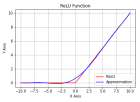
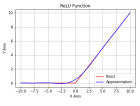
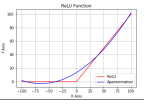
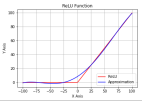
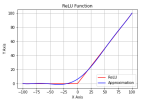
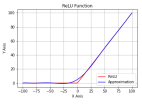
For comparison, we used the model proposed in CryptoNets [4] which is similar to the one proposed in our paper – a Conv, FC layers and an average polling layer as shown in Table V. Training the model with the ReLU AF, the accuracy of our model was 99.20% whereas CryptoNets achieved a 99%. Similarly, for the approximated function we obtained an accuracy of 98.5% while CryptoNets achieved 98.95%. For the same network, the accuracy of the model proposed in [46] was 99.02% using the ReLU AF and 99% using the approximated function.

B. Performing Computation on Encrypted Data

Now, we proceed by discussing how the use of HE can affect the performance of the NN model. In our work, we trained the CNN model on plaintext data while the classification was performed on the ciphertexts. As a result, we had to perform computations on two types of data – plaintext and ciphertext. For this purpose, we used the SEAL library that allowed us to perform computations on ciphertext. Although the use of SEAL is straightforward, we still had to define certain parameters (see Section V-A).

We performed a series of experiments using different encryption parameters. First, we looked at the polynomial modulus – the encryption parameter used in SEAL. During the experiments we observed that a smaller value of polynomial modulus leads to a more efficient result but at the same time the accuracy is decreased. In contrast, a higher value of the polynomial modulus gives more accurate results, however, degrades the performance. The second encryption parameter is the coefficient modulus that decides the NB in the freshly encrypted ciphertext. This parameter is automatically set by SEAL based on the value of security level and polynomial

TABLE III
POLYNOMIAL APPROXIMATION OF THE RELU FUNCTION ON TWO INTERVALS ([-10,10], [-100,100]) USING DIFFERENT DEGREES

Degree	Interval	Polynomial Approximation	ReLU Function
3	$[-10, 10]$	$(-4.44089209850063e-18) \times x^3 + (0.038268343236509) \times x^2 + (0.5) \times x + 1.35299025036549$	
5	$[-10, 10]$	$(2.368475785867e-19) \times x^5 - (0.000252624921308674) \times x^4 - (2.90138283768708e-17) \times x^3 + (0.0660873211772537) \times x^2 + (0.5) \times x + 0.862730150341736$	
7	$[-10, 10]$	$(-8.88178419700125e-21) \times x^7 + (3.66197231323541e-6) \times x^6 + (1.33226762955019e-18) \times x^5 - (0.00084792718318668 2) \times x^4 - (5.24025267623074e-17) \times x^3 + (0.0920352084972136) \times x^2 + (0.5) \times x + 0.637244473880199$	
9	$[-10, 10]$	$(1.15960574476048e-21) \times x^9 - (7.03111115816643e-8) \times x^8 - (2.41868747252738e-19) \times x^7 + (1.87324195121527e-5) \times x^6 + (1.66338054441439e-17) \times x^5 - (0.00189480875502865) \times x^4 - (4.21263024463769e-16) \times x^3 + (0.117284304779533) \times x^2 + (0.5) \times x + 0.506232562894004$	
3	$[-100, 100]$	$(-4.2632564145606e-20) \times x^3 + (0.0038268343236509) \times x^2 + (0.5) \times x + 13.5299025036549$	
5	$[-100, 100]$	$(2.27373675443232e-23) \times x^5 - (2.52624921308674e-7) \times x^4 - (2.70006239588838e-19) \times x^3 + (0.00660873211772537) \times x^2 + (0.5) \times x + 8.62730150341737$	
7	$[-100, 100]$	$(-6.82121026329696e-27) \times x^7 + (3.6619723132354e-11) \times x^6 + (1.03739239420975e-22) \times x^5 - (8.47927183186682e-7) \times x^4 - (4.2277292777726e-19) \times x^3 + (0.00920352084972135) \times x^2 + (0.5) \times x + 6.37244473880199$	
9	$[-100, 100]$	$(1.12777343019843e-29) \times x^9 - (7.03111115816644e-15) \times x^8 - (2.35559127759188e-25) \times x^7 + (1.87324195121527e-10) \times x^6 + (1.62231117428746e-21) \times x^5 - (1.89480875502865e-6) \times x^4 - (4.11404244005098e-18) \times x^3 + (0.0117284304779533) \times x^2 + (0.5) \times x + 5.06232562894004$	

modulus. Finally, increasing the value of the plaintext modulus, decreases the consumption of the NB.

C. Comparison with the Existence Model

Finally, we compared our results with state-of-the-art privacy-preserving NNs that utilize HE. The work proposed in CryptoNets [4] is similar to ours. In CryptoNets, the model is trained on plaintext data and then the trained model is used for the classification of encrypted instances. In order to have a fair comparison, it is important to incorporate the same network used in both works. To this end, we used the CryptoNets model. Instead of using the overall performance of the model we decided to equate each layer. As can be seen in Table V, our model outperforms CryptoNets at both the encryption and decryption times as well as in the activation layer.

VII. LEARNING IN THE DARK PROTOCOL

In the first part of this section, we formalize the communication between the user and the CSP by designing a detailed protocol. Then, we prove the security of our construction in the presence of a malicious adversary. For the rest of the section, we assume the existence of a cryptogrpahic hash function that is first and second pre-image resistant. Before we proceed to the formal description of our protocol, we present a high-level overview of our construction.

High-Level Overview: We assume that a user u wishes to classify an image in a privacy-preserving way. To this end, u first outputs an image and encrypts it using an HE scheme. As a next step, u sends the encrypted image to the CSP. Upon reception, the CSP commences the classification process directly on the encrypted image without the need to decrypt it. To achieve this, the CSP runs the evaluation algorithm of the HE scheme on the encrypted image and finally, outputs an encrypted vector. Each element of the vector represents the probability that the image belongs to a certain class. Finally, the CSP sends the encrypted vector back to u. Upon reception, u decrypts the vector and classifies her image to the class that has the highest probability.

A. Construction

As already stated in Section V, we assume a client-server model. Our protocol takes part in two different phases; a *Setup*

TABLE IV
POLYNOMIAL APPROXIMATION OF THE SIGMOID FUNCTION ON TWO INTERVALS ([-10,10], [-100,100]) USING DIFFERENT DEGREES

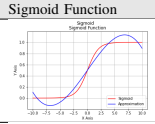
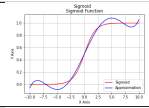
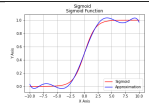
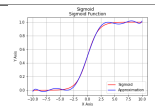
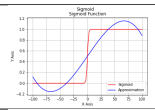
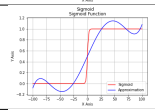
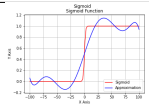
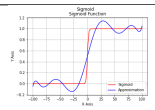
Degree	Interval	Polynomial Approximation	Sigmoid Function
3	$[-10, 10]$	$(-0.00100377373568484) \times x^3 + (1.45518367592346e-13) \times x^2 + (0.139786538317376) \times x + 0.499999999992724$	
5	$[-10, 10]$	$(2.0467424332792e-5) \times x^5 + (5.82078097744257e-15) \times x^4 - (0.00336794817488311) \times x^3 - (5.65619279205865e-13) \times x^2 + (0.187819515164365) \times x + 0.500000000006453$	
7	$[-10, 10]$	$(-4.34913635838155e-7) \times x^7 - (5.82079696725621e-16) \times x^6 + (9.18419138902492e-5) \times x^5 + (8.44014964905896e-14) \times x^4 - (0.00652613009889838) \times x^3 - (3.00134166245124e-12) \times x^2 + (0.216030242339756) \times x + 0.500000000015461$	
9	$[-10, 10]$	$(9.32721914680041e-9) \times x^9 + (1.39698499452418e-17) \times x^8 - (2.42773327147286e-6) \times x^7 - (2.60854055994519e-15) \times x^6 + (0.000229352354062705) \times x^5 + (1.47390762630842e-13) \times x^4 - (0.0097848700927233) \times x^3 - (2.49775180627496e-12) \times x^2 + (0.231624826001611) \times x + 0.500000000005353$	
3	$[-100, 100]$	$(-1.0823922002923939 45e-6) \times x^3 + 6.9623566103164815542e-21 \times x^2 + 0.014650756326574837423 \times x + 0.49999999999999996519$	
5	$[-100, 100]$	$(2.76073648126615e-10) \times x^5 + (5.82075253629132e-19) \times x^4 - (4.39372964314194e-6) \times x^3 - (5.65616515992073e-15) \times x^2 + 0.0221378748242352 \times x + 0.500000000006453$	
7	$[-100, 100]$	$(-8.156729162126 68e-14) \times x^7 - (5.82076636528339e-22) \times x^6 + (1.66796555589344e-9) \times x^5 + (8.44011305257745e-18) \times x^4 - (1.10438386427853e-5) \times x^3 - (3.00133479250796e-14) \times x^2 + 0.029595343798993 \times x + 0.500000000015461$	
9	$[-100, 100]$	$(2.59190909648308e-17) \times x^9 + (1.3969822658824e-25) \times x^8 - (6.55008389245384e-13) \times x^7 - (2.60853504193157e-21) \times x^6 + (5.85712544194627e-9) \times x^5 + (1.47390439498889e-17)) x^4 - (2.21440994162872e-5) \times x^3 - (2.49774773767457e-14) \times x^2 + 0.0370400456474675 \times x + 0.500000000005353$	

TABLE V
COMPARISON WITH THE PREVIOUS MODELS

Layer	**Description**	**Time**	
		CryptoNets	Learning in the Dark
Encryption	Encoding+Encryption	44.5	8.5242
1st Conv	Same except stride value	30	60.36
1st AF	NLAF	81	6.62
1st pooling layer	Mean pooling	127	0.188
2nd Conv	-	-	64.822
2nd AF	NLAF	10	0.199
2nd pooling layer	-	-	0.092
1st FC layer	Generates 10 output	1.6	12.1839
2nd FC layer	-	-	0.326
Decryption	Image decryption	3	0.0021

phase and a *Running* phase.

Setup Phase: In the Setup phase, the user u and the CSP establish a shared symmetric key K. This key will be used to secure the communication between the two entities. Apart from that u also generates a public/private key pair for a HE scheme. More specifically, u executes $\mathsf{HE.KeyGen}(1^\lambda) \rightarrow (\mathsf{pk}, \mathsf{sk})$, for some λ. We assume that upon its generation, pk is publicly known while sk remains secret.

Running Phase: After the successful execution of the *Setup* phase, u can start communicating with the CSP. To do so, u first encrypts an image img by running $\mathsf{HE.Enc}(\mathsf{pk}, img) \rightarrow c_{img}$. Moreover, u generates an unpredictable random number r_1 and sends to the CSP $m_1 = \langle r_1, c_{img}, \mathsf{HMAC}(\mathsf{K}, r_1 || c_{img}) \rangle$. Upon reception, the CSP checks the freshness of the message by looking at r_1, and verifies the HMAC using the shared key K. If any of the above verifications fail, the CSP will output $\perp$ and abort the protocol. Otherwise, the CSP proceeds with the execution of the ML model described in Section V. In particular, the CSP starts running $\mathsf{HE.Eval}$ and finally, outputs an encrypted vector c_{eval}. The encrypted vector is then sent back to u via $m_2 = \langle r_2, c_{eval}, \mathsf{HMAC}(\mathsf{K}, r_2 || c_{eval} | c_{img} |) \rangle$. Upon receiving m_2, u verifies both the freshness of the message and the HMAC. If the verification fails, u outputs $\perp$ and aborts the protocol. Otherwise, u decrypts c_v by running $\mathsf{HE.Dec}(\mathsf{sk}, c_{eval}) \rightarrow v$. Having the plaintext vector at her disposal, u can now classify her image in accordance with the probabilities included in the

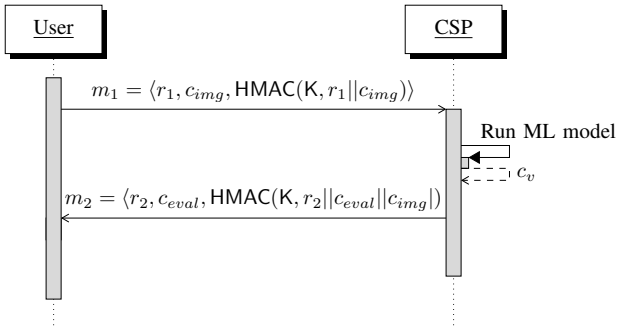


Fig. 6. Running Phase.

vector. Our construction is illustrated in Figure 6

B. Security Analysis

We prove security of our protocol in presence of malicious adversary $\mathcal{ADV}$. We start by defining the threat model:

Threat Model: Our threat model is similar to the one described in [47], which is based on the Dolev-Yao adversarial model [48]. Moreover, we extend the above threat model by defining a set of attacks available to $\mathcal{ADV}$.

Attack 1 (Image Substitution Attack (ISA))**.** *Let $\mathcal{ADV}$ be an adversary that overhears the communication between user and CSP. $\mathcal{ADV}$ successfully launches an ISA if she manages to replace encrypted image sent from user to CSP, in a way that is indistinguishable from CSP.*

Attack 2 (Vector Substitution Attack (VSA))**.** *Let $\mathcal{ADV}$ be an adversary that overhears the communication between the user and the CSP. $\mathcal{ADV}$ successfully launches a VSA, if she manages to replace the encrypted vector sent from the CSP to the user, in a way that is indistinguishable for the user.*

We now proceed with proving that our protocol is secure against the defined threat model.

Proposition 1 (Image Substitution Attack Soundness)**.** *Let $\mathcal{ADV}$ be a malicious adversary. Then $\mathcal{ADV}$ cannot successfully launch an ISA.*

Proof. For $\mathcal{ADV}$ to successfully launch an ISA, she needs to tamper with $m_1 = \langle r_1, c_{img}, \mathsf{HMAC}(\mathsf{K}, r_1||c_{img})$. To do so, $\mathcal{ADV}$ has two options:

1) Forge a new m_1 message.
2) Replay an old m_1 message.

We will show that in both cases, $\mathcal{ADV}$ can successfully launch her attack with negligible probability.

- Since we assume that the pk of the HE scheme is publicly known, $\mathcal{ADV}$ can generate a valid ciphertext c'_{img} that is indistinguishable from the original c_{img}. As a next step, $\mathcal{ADV}$ replaces the original c_{img} with the newly generated c'_{img} and forwards $m'_1 = \langle r_1, c'_{img}, \mathsf{HMAC}(\mathsf{K}, r_1||c_{img})$ to the CSP. Upon reception, the CSP will try to verify the HMAC. However, as $c'_{img} \neq c_{img}$ the verification will fail, and the CSP will abort the protocol. Hence, $\mathcal{ADV}$ also needs to forge a valid HMAC. However, as $\mathcal{ADV}$ does not possess the shared key K, this can only happen with negligible probability and thus, the attack fails.
- The only other alternative for $\mathcal{ADV}$, is to replay an older message. To do so, $\mathcal{ADV}$ replaces the m_1 message sent from the user to the CSP with an older m'_1 message from a previous session. Upon receiving m'_1, the CSP will verify the validity of the HMAC but it will fail to verify the freshness of the message. An alternative for $\mathcal{ADV}$, would be to generate a fresh random number and to replace the old one. However, since the random number is also included in the HMAC, $\mathcal{ADV}$ would also need to forge a valid HMAC. Given the fact that $\mathcal{ADV}$ does not possess the shared key K, this can only happen with negligible probability and thus, the attack fails.

□

Proposition 2 (Vector Substitution Attack Soundness)**.** *Let $\mathcal{ADV}$ be a malicious adversary. Then $\mathcal{ADV}$ cannot successfully launch a VSA.*

Proof. The proof is omitted as it is similar to the previous one. More specifically, the security properties of the HMAC, and the fact that $\mathcal{ADV}$ does not know the shared K are enough to ensure that $\mathcal{ADV}$ cannot successfully launch a VSA. □

Open Science and Reproducible Research: To support open science and reproducible research, and provide researchers with the opportunity to use, test, and hopefully extend our work, our source code has been made available online[3].

VIII. Conclusion

Undoubtedly, ML models and their underlying applications are driving the big-data economy. However, in practice, the systems using these models incorporate proxies. Many existing systems can introduce biases or rely on proxies like gender or race, leading to unfair outcomes. With this work, we aim to create a more equitable and unbiased approach to decision-making. Learning in the Dark allows us to apply ML models directly to encrypted data so the information remains secure. We accomplished this by estimating the behavior of activation functions, which are components of ML models. Our experiments and evaluations showed promising results, demonstrating that Learning in the Dark can effectively analyze encrypted data while maintaining high accuracy. We believe this research can inspire further advancements in privacy-preserving machine learning and contribute to systems that promote fairness, privacy, and transparency in our increasingly data-driven world.

References

[1] S. D. Rane, W. Sun, and A. Vetro, "Secure distortion computation among untrusting parties using homomorphic encryption," in *2009 16th IEEE international conference on image processing*.

[2] N. islam, W. Puech, K. Hayat, and R. Brouzet, "Application of homomorphism to secure image sharing," *Optics Communications*, vol. 284, no. 19, pp. 4412–4429, 2011.

[3]https://gitlab.com/nisec/blind_faith

[3] M. Kim and K. Lauter, "Private genome analysis through homomorphic encryption," in *BMC medical informatics and decision making*, vol. 15, no. S5. Springer, 2015, p. S3.

[4] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International Conference on Machine Learning*, 2016, pp. 201–210.

[5] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Manual for using homomorphic encryption for bioinformatics," *Proceedings of the IEEE*, vol. 105, no. 3, 2017.

[6] T. Shortell and A. Shokoufandeh, "Secure signal processing using fully homomorphic encryption," *IET Information Security*, 2019.

[7] G. cloud, "Google prediction api," https://cloud.google.com/prediction/.

[8] M. Azure, "Azure machine learning," https://azure.microsoft.com/en-us/services/machine-learning/#features.

[9] E. Labs, "Ersatz labs," http://www.ersatzlabs.com/.

[10] A. Michalas, N. Paladi, and C. Gehrmann, "Security aspects of e-health systems migration to the cloud," in *e-Health Networking, Applications and Services (Healthcom), 2014 IEEE 16th International Conference on*. IEEE, 2014, pp. 212–218.

[11] P. Thaine, "Perfectly privacy-preserving ai."

[12] S. Garge and D. Galindo, "Neural networks for encrypted data using homomorphic encryption," 2018.

[13] T. Khan, K. Nguyen, and A. Michalas, "Split ways: Privacy-preserving training of encrypted data using split learning," in *Proceedings of the Workshops of the EDBT/ICDT 2023 Joint Conference, Ioannina, Greece, March, 28, 2023*, ser. CEUR Workshop Proceedings, G. Fletcher and V. Kantere, Eds., vol. 3379. CEUR-WS.org, 2023. [Online]. Available: https://ceur-ws.org/Vol-3379/HeDAI_2023_paper402.pdf

[14] T. Khan, K. Nguyen, A. Michalas, and A. Bakas, "Love or hate? share or split? privacy-preserving training using split learning and homomorphic encryption," in *The 20th Annual International Conference on Privacy, Security & Trust (PST2023) 21-23 August, 2023, Copenhagen, Denmark*, 2023.

[15] T. Khan, A. Bakas, and A. Michalas, "Blind faith: Privacy-preserving machine learning using function approximation," in *2021 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2021, pp. 1–7.

[16] J. L. Crawford, C. Gentry, S. Halevi, D. Platt, and V. Shoup, "Doing real work with fhe: the case of logistic regression," in *Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2018, pp. 1–12.

[17] E. Hesamifard, H. Takabi, and M. Ghasemi, "Cryptodl: towards deep learning over encrypted data," in *Annual Computer Security Applications Conference, Los Angeles, California, USA*, 2016.

[18] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network." *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 35, 2017.

[19] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, "Faster cryptonets: Leveraging sparsity for real-world encrypted inference," *arXiv preprint arXiv:1811.09953*, 2018.

[20] P. Bunn and R. Ostrovsky, "Secure two-party k-means clustering," in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, pp. 486–497.

[21] A. P. Sanil, A. F. Karr, X. Lin, and J. P. Reiter, "Privacy preserving regression modelling via distributed computation," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004, pp. 677–682.

[22] Y. Lindell and B. Pinkas, "Privacy preserving data mining," in *Annual International Cryptology Conference*. Springer, 2000.

[23] J. Vaidya, H. Yu, and X. Jiang, "Privacy-preserving svm classification," *Knowledge and Information Systems*, 2008.

[24] A. B. Slavkovic, Y. Nardi, and M. M. Tibbits, "" secure" logistic regression of horizontally and vertically partitioned distributed databases," in *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)*. IEEE, 2007, pp. 723–728.

[25] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 19–38.

[26] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 619–631.

[27] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "{GAZELLE}: A low latency framework for secure neural network inference," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1651–1669.

[28] H. Kitai, J. P. Cruz, N. Yanai, N. Nishida, T. Oba, Y. Unagami, T. Teruya, N. Attrapadung, T. Matsuda, and G. Hanaoka, "Mobius: model-oblivious binarized neural networks," *IEEE Access*, vol. 7, 2019.

[29] S. Wu, T. Teruya, J. Kawamoto, J. Sakuma, and H. Kikuchi, "Privacy-preservation for stochastic gradient descent application to secure logistic regression," in *The 27th Annual Conference of the Japanese Society for Artificial Intelligence*, 2013.

[30] T. Graepel, K. Lauter, and M. Naehrig, "Ml confidential: Machine learning on encrypted data," in *International Conference on Information Security and Cryptology*. Springer, 2012, pp. 1–21.

[31] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption." *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, 2012.

[32] E. Hesamifard, H. Takabi, and M. Ghasemi, "Cryptodl: Deep neural networks over encrypted data," *arXiv preprint arXiv:1711.05189*, 2017.

[33] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.

[34] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[35] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," in *Annual International Cryptology Conference*. Springer, 2018.

[36] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*. IEEE, 2017, pp. 1–6.

[37] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1999, pp. 223–238.

[38] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[39] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009, pp. 169–178.

[40] W. Wang, Y. Hu, L. Chen, X. Huang, and B. Sunar, "Exploring the feasibility of fully homomorphic encryption," *IEEE Transactions on Computers*, vol. 64, no. 3, pp. 698–706, 2013.

[41] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–35, 2018.

[42] K. Atkinson and W. Han, "A functional analysis framework," *Theoretical Numerical Analysis, Texts in Applied Mathematics*, vol. 39, 2005.

[43] J. Schlessman, "Approximation of the sigmoid function and its derivative using a minimax approach," 2002.

[44] R. Thangadurai, "On the coefficients of cyclotomic polynomials," *Cyclotomic fields and related topics (Pune, 1999)*, 2000.

[45] "Microsoft SEAL (release 3.2)," https://github.com/Microsoft/SEAL, Feb. 2019, microsoft Research, Redmond, WA.

[46] E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright, "Privacy-preserving machine learning as a service," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 3, pp. 123–142, 2018.

[47] N. Paladi, C. Gehrmann, and A. Michalas, "Providing user security guarantees in public infrastructure clouds," *IEEE Transactions on Cloud Computing*, vol. 5, no. 3, pp. 405–419, July 2017.

[48] D. Dolev and A. C. Yao, "On the security of public key protocols," *Information Theory, IEEE Transactions on*, vol. 29, no. 2, 1983.