CryptoNN: Training Neural Networks over Encrypted Data

Runhua Xu, James B.D. Joshi, and Chao Li

School of Computing and Information, University of Pittsburgh

runhua.xu@pitt.edu, jjoshi@pitt.edu, chl205@pitt.edu

Abstract—Emerging neural networks based machine learning techniques such as deep learning and its variants have shown tremendous potential in many application domains. However, they raise serious privacy concerns due to the risk of leakage of highly privacy-sensitive data when data collected from users is used to train neural network models to support predictive tasks. To tackle such serious privacy concerns, several privacy-preserving approaches have been proposed in the literature that use either secure multi-party computation (SMC) or homomorphic encryption (HE) as the underlying mechanisms. However, neither of these cryptographic approaches provides an efficient solution towards constructing a privacy-preserving machine learning model, as well as supporting both the training and inference phases.

To tackle the above issue, we propose a CryptoNN framework that supports training a neural network model over encrypted data by using the emerging functional encryption scheme instead of SMC or HE. We also construct a functional encryption scheme for basic arithmetic computation to support the requirement of the proposed CryptoNN framework. We present performance evaluation and security analysis of the underlying crypto scheme and show through our experiments that CryptoNN achieves accuracy that is similar to those of the baseline neural network models on the MNIST dataset.

Index Terms—neural networks; deep learning; machine learning; privacy-preserving; functional encryption; cryptography;

I. INTRODUCTION

Emerging deep learning (DL) architectures such as convolutional neural networks and recurrent neural networks show a huge promise for artificial intelligence (AI) based applications. Neural networks based machine learning techniques have succeeded in many fields including computer vision, speech/audio recognition, etc. [1]. Training of such a neural network model requires a huge amount of data to enable reliable predictive analytics. Commercial AI service providers such as Google, Microsoft, and IBM have devoted a lot of efforts to build DL models for various intelligence applications, where the models are trained based on data collected from their customers. Even though the AI-based applications bring convenience to daily life, it also raises serious privacy concerns due to the risk of leakage of highly privacy-sensitive data. Such privacy concerns are hindering the applicability of neural network models for real world applications [2]–[5].

To tackle the serious privacy concerns when applying neural network models in the cloud-based applications where users' privacy-sensitive data such as electronic health/medical records, location information, etc., are stored and processed, a well designed privacy-preserving neural networks framework is very essential so that the cloud and the data owners are not required to reveal their models and sensitive data, respectively. Existing privacy-preserving machine learning approaches are essentially of two types : (i) non-crypto based approaches such as SecureML [6], DeepSecure [5], and approaches proposed in [4], [7], [8]; and (ii) crypto based approaches such as CryptoNets [3], and approaches proposed in [9]–[15]. TABLE I illustrates the differences among the existing solutions. Non-crypto approaches have several limitations such as significant communication overhead in the secure protocol based solutions where a high degree of coordination and information exchange are required [5], [6], training overhead for the data owner [6], and utility loss [8]. Most of the existing crypto based approaches, such as [3], [9]–[15], employ homomorphic encryption (HE) to protect the data, where the computation over encrypted data is allowed. As the sensitive data encrypted by using HE can only be decrypted by the data owner, such proposed HE-based approaches, however, only support prediction over encrypted data using existing trained models, rather than training a model over encrypted data. This is because the computed results of HE is confidential to the server, and hence cannot be used for evaluation with the label during the back-propagation phase. That is, the machine learning model should be trained on the plaintext data, and then the trained model can be applied over encrypted data to do the prediction.

Consider a computer aided diagnostic application scenario: several distributed federal clinics want to train an AI model to help diagnose the images generated from magnetic resonance imaging (MRI), X-ray, etc. The limited IT infrastructure or AI experts in such clinics means that they cannot train the model by themselves, and hence have to rely on a third party service, such as a cloud based machine learning service. However, the regulations for protecting patients' healthcare records require a secure approach for training machine learning models on the privacy-sensitive data collected from different federal clinics. To the best of our knowledge, only the work proposed in [2] supports both training and predictive analysis over encrypted data by integrating several crypto schemes (i.e., Quadratic Residuosity cryptosystem, Paillier cryptosystem, and homomorphic encryption) with secure protocols designed for them. However, it only supports limited types of machine learning model as presented in TABLE I.

To tackle the challenge of training a model over encrypted data in a simpler manner, in this paper, we propose a novel

TABLE I
COMPARISON OF PRIVACY-PRESERVING APPROACHES IN MACHINE LEARNING MODELS

Proposed Work	Training	Prediction	Privacy ▷	ML Model	Approach
[4]	●	●	○	General	Delegation‡
[7]	●	○	○	Deep Learning	Distributed*
[8]	●	○	○	Deep Learning	Differential Privacy◇
SecureML [6]	●	●	◖	General	Secure Protocol (SMC)
DeepSecure [5]	●	●	◖	Deep Learning	Secure Protocol (Garbled Circuits)
CryptoNets [3], [9], [10], [11], [12], [13], [14], [15]	○	●	●	Covers All	Homomorphic Encryption (HE)
[2]	●	●	●	Limited ML†	HE + Secure Protocol
CryptoNN (our work)	●	●	●	Neural Networks	Functional Encryption

▷ This column indicates the privacy strength/guarantee such as mild approach ○ (e.g. differential privacy) and strong guarantee ● (e.g. crypto system).
† It only supports Hyperplane Decision, Nave Bayes, and Decision Trees models.
‡ The data owner trains the model by itself and outsources partial computation in a privacy-preserving setting.
* The model is trained in a distributed manner where each data owner trains a partial model on their private data.
◇ It applies differential privacy method on the training data.

framework called CryptoNN, where a neural network model is trained over encrypted data in a very simple way without the overhead of interactive communication protocols (i.e., without complex secure protocols), while also supporting predictive analytics in a privacy-preserving way. The underlying crypto scheme in our proposed CryptoNN is the emerging *functional encryption* [16]. Unlike traditional encryption schemes such as homomorphic encryption, where decryption reveals all or nothing, in a *functional encryption* scheme, the decryption keys can also reveal some partial information about the plaintext [17]. To be concrete, for a function $f(\cdot)$, an authority holding a master secret key can generate a key sk_f that enables the computation of the function $f_{\mathsf{sk}_f}(\cdot)$ on the encrypted data. For instance, using function related key sk_f the decryptor can compute $f_{\mathsf{sk}_f}(x)$ from the encrypted data, $\mathsf{enc}(x)$, without revealing the plaintext x. CryptoNN can compute several permitted functions over the privacy-sensitive data that is encrypted by a distributed set of data owners, and only acquires the computed results instead of the original plaintext data. This makes it possible to train a neural network model over encrypted data without the overhead of the communication protocols. The proposed CryptoNN framework relies on a secure matrix computation scheme which is based on existing functional encryption for the inner-product scheme proposed by Abdalla et al. in [18] and our newly constructed functional encryption for the basic arithmetic operations.

We summarize our key contributions as follows:

- We propose a functional encryption scheme for basic arithmetic operations to support basic computation over encrypted data.
- We also propose a secure matrix computation scheme, where the elements are protected by those functional encryption schemes.
- We propose a general CryptoNN framework built upon the secure computation components to support the training of neural network models over encrypted data. Essentially, CryptoNN inserts a *secure feed-forward* step and a *secure back-propagation/evaluation* step into the training phase of the normal neural network. Also, we present a crypto-based *convolutional neural network* model as an instantiation of the CryptoNN scheme.

- Finally, we analyze the security of the proposed work, and present experiments to evaluate the performance of the underlying secure matrix computation and the CryptoNN schemes. We show that the accuracy of CryptoNN is similar to that of the original NN model.

To the best of our knowledge, this is the first approach for training a model over encrypted data using the functional encryption scheme.

Organization. The paper is organized as follows. In Section II, we introduce the background and preliminaries. We propose our CryptoNN framework and its related secure components in Section III. The security analysis and experimental evaluation are presented in Section IV. We discuss the related work in Section V and conclude the paper in Section VI.

II. BACKGROUND AND PRELIMINARIES

A. Application Scenario

Most of the existing AI-based applications such as Google's Cloud AI and Microsoft's Azure Machine Learning are built using the cloud-client or server-client architecture, where the client holds the data and the cloud/server is responsible for training a machine learning model for a specific task such as classification, regression, etc., and/or providing a predictive analytics service for the client.

In this paper, we consider the same architecture, but with stronger privacy guarantees as all the data owned by a client is in an encrypted form. To be specific, in the training phase, what the client needs to do is encrypting its sensitive data as required and then sending it to the server. The server computes several permitted functions over encrypted data, acquires the computation result, and builds the final neural network model. Similarly, the data to be used for prediction is also encrypted in the prediction phase, and then the server outputs the prediction using the trained model.

Note that we put more efforts in the training phase in this paper, which is currently the biggest challenge in crypto-based machine learning models [2]. In the prediction phase, it is still possible to apply (i) existing homomorphic encryption (HE) based approaches (e.g., CryptoNets [3], and approaches in [9]–[15]), or (ii) our proposed functional encryption (FE) based approach. Neither our FE based approach nor the existing HE

TABLE II
GENERAL NOTATIONS AND SYMBOLS IN THIS PAPER

Symbols	Description
mpk	The public key generated by the authority.
msk	The master secret key generated by the authority.
$f(\cdot)$	The general function over arbitrary input.
sk_f	The secret key related to function $f(\cdot)$.
$[\eta]$	A set $\{1, 2, ..., \eta\}$.
$\mathbf{x}$	The bold lowercase letters denoting vectors.
$\mathbf{X}$	The bold uppercase letters denoting matrices.
$[[d]]$	The cihpertext of d.
∂	The partial derivative symbol.

based approaches allow the server to learn the prediction data. The only difference is that an HE based approach does not allow the server to learn the prediction result, while the FE based approach allows that. Such a setting provides flexible choices for the client with varying levels of privacy concerns.

B. Functional Encryption

Recently proposed approaches such as those in [19]–[24], have focused their attention on the theoretical feasibility or existence of functional encryption, and thus constructing schemes for general functionalities at the expense of efficiency. In particular, those schemes rely on strong primitives such as indistinguishable obfuscation or multilinear maps that are prohibitively inefficient [25]. As a result, these schemes are far from being practically deployable.

In our paper, we adopt the functional encryption approach proposed in [18] as the basis for the matrix computation over encrypted data. It is a simple functional encryption scheme only supporting *inner-product* computation, whose security is based on DDH assumption. Even though the functionality is limited to *inner-product* computation, it is the state-of-the-art practical construction so far. To support the element-wise computation (i.e., product, division, addition, and subtraction), we also propose our construction based on the same DDH security assumption in the next section.

Here, we present the specific construction of functional encryption for an inner product proposed in [18]. Suppose the inner-product functionality is defined as follows:

$$f(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{\eta} (x_i y_i),$$

where η is the length of the vectors $\mathbf{x}$ and $\mathbf{y}$. Several general symbols and notations are presented in TABLE II. The functional encryption scheme for the inner-product functionality $f(\mathbf{x}, \mathbf{y})$ is defined as `FEIP = (Setup, KeyDerive, Encrypt, Decrypt)`, as follows:

- `Setup`$(1^\lambda, 1^\eta)$: The algorithm first generates two samples as $(\mathbb{G}, p, g) \leftarrow_\$ \texttt{GroupGen}(1^\lambda)$, and $\mathbf{s} = (s_1, ..., s_\eta) \leftarrow_\$ \mathbb{Z}_p^\eta$ on the inputs of security parameters λ and η, and then sets $mpk = (g, h_i = g^{s_i})_{i \in [\eta]}$ and $msk = \mathbf{s}$. It returns the pair (mpk, msk).
- `KeyDerive`$(msk, \mathbf{y})$: The algorithm outputs the function secret key $sk_f = \langle \mathbf{y}, \mathbf{s} \rangle$ on the inputs of master secret key msk and vector $\mathbf{y}$.

- `Encrypt`$(mpk, \mathbf{x})$: The algorithm first chooses a random $r \leftarrow_\$ \mathbb{Z}_p$ and computes $ct_0 = g^r$. For each $i \in [\eta]$, it computes $ct_i = h_i^r \cdot g^{x_i}$. Then the algorithm outputs the ciphertext $ct = (ct_0, (ct_i)_{i \in [\eta]})$.
- `Decrypt`$(mpk, ct, sk_f, \mathbf{y})$: The algorithm takes the ciphertext ct, the public key mpk and functional key sk_f for the vector $\mathbf{y}$, and returns the discrete logarithm in basis g, i.e., $g^{\langle \mathbf{x}, \mathbf{y} \rangle} = \prod_{i \in [\eta]} ct_i^{y_i} / ct_0^{sk_f}$.

Note that even though it is impossible to recover the exponent of generator g^r in the group $\mathbb{G}$ according to the DDH assumption, it is still possible to recover the inner-product $\langle \mathbf{x}, \mathbf{y} \rangle$ in the discrete logarithm with the same basis, g, as $\langle \mathbf{x}, \mathbf{y} \rangle \ll r$. There exists efficient approaches to computing the discrete logarithm such as *baby-step giant-step* algorithm [26]. We refer the readers to [18] for the correctness proof of the decryption.

C. Neural Networks

A neural network usually refers to learning a model that is a hierarchical and non-linear in structure consisting of several layers, where each layer includes several neural units (i.e., artificial neurons). In a neural network, each layer receives the data generated by its previous layer and outputs the processed data for the next layer. In particular, the raw data is encoded properly and fed into the first layer of a neural network, also known as the input layer. Then, these features from the raw data are gradually mapped to higher-level abstractions via the iterative update (a.k.a, feed-forward and back-propagation) in the intermediate (hidden) layers of the neural network until the convergence condition is achieved (e.g., the specified number of iteration). These mapping abstractions known as learned neural network model then can be used to predict the label in the last layer (i.e., the output layer).

With the emergence of the deep neural network (a.k.a deep learning) techniques, several architectures have been proposed that have been successfully used in different application domains. For instance, the convolutional neural network (CNN, or ConvNet) [27] is a kind of deep learning model that is typically applied towards learning the features in the computer vision domain. A typical CNN architecture includes the following layer: the input layer, the convolutional layer, the pooling layer (also known as the down-sampling layer), activation layer (e.g., ReLU layer, sigmoid layer, etc.), fully connected layer, and the output layer.

In this paper, we present a framework that is able to train a neural network model over encrypted data. Here, we present several common processing operations that are computed in each layer or neural unit.

- *Dot-product function*: It is used in the general hidden layer, where the dot-product operation needs to be executed between the trained parameters of a specific hidden layer and the outputs of the previous layer.
- *Weighted-sum function*: It is used in the convolutional layer, where the filter or kernel (i.e., the layer's parameter) is convolved across the height and width of the input

volume by computing each weighted-sum of the input and the filter matrix.

- *Pooling function*: It is used in the pooling layer where the maximal/average value of some of the components of the feeding layer is computed.
- *Activation function*: It is used in the activation layer, where the computation is based on a specific activation function such as rectified linear (ReLu) function, sigmoid function, tanh function, etc.
- *Cost function*: It is used in the output layer, where the output of the neural network model is measured by a specified cost function with ground truth label to evaluate the performance of the model.

III. CRYPTONN FRAMEWORK

A. Overview

The proposed CryptoNN framework includes three entities: *authority*, *server*, and *client* as depicted in the Fig.1.

The *authority* is responsible for setting up the crypto parameters such as public key *mpk* and master secret key *msk*. Then it holds the *msk* and distributes the public keys to the servers and clients. Besides, it also generates the function derived key for the *server* to decrypt the function result.

The *client* first needs to pre-process the training data as required, and then encrypt all the pre-processed training data using the public key, *mpk*. The pre-processing operation can be quite simple. For instance, in the computer vision related neural network model, the pretreatment involves mapping each image into a vector. Suppose that the data is a color image with size 28×28. The client needs to map all $28 \times 28 \times 3$ pixel values into one vector and then encrypt the vector. Similar pretreatment and encryption are applied to the label. Once the encrypted training data is ready, the client sends it to the server for training a neural network model.

The *server* collects all the encrypted data from a distributed set of clients. For each training iteration, the *server* first needs to acquire the function derived key, sk_f, from the *authority* for a specific function (e.g., dot-product, element-wise subtract, etc.) representing the underlying computation between the input layer and the first hidden layer; then it needs to decrypt the function result and continue the process in the rest of the hidden layers. For instance, in the the feed-forward phase, a typical process for the first hidden layer is $a = g(\mathbf{WX} + \mathbf{b})$, where a is the output of the layer, g is the activation function, and $\mathbf{b}$ is the bias. $\mathbf{X}$ represents the features of samples from the input layer, and $\mathbf{W}$ is the parameters of each neural node in the first hidden layer.

In our CryptoNN framework, the input $enc(\mathbf{X})$ is protected by functional encryption scheme; thus, it is not feasible to compute $\mathbf{WX}$ directly. Thus, the process needs to be transferred to $a = g(sk_f(\mathbf{W}) \cdot enc(\mathbf{X}) + \mathbf{b}) = g(f(\mathbf{WX}) + \mathbf{b})$. Even though the *server* does not learn the value of $\mathbf{X}$, it can compute the result (i.e., $f(\mathbf{WX})$) using the function derived key, sk_f. As a result, the *server* can continue the feed-forward process in the neural networks until it reaches the output layer, where similar secure computation is applied to the encrypted

label. Note that the label $\mathbf{Y}$ also should be pre-processed before encryption to prevent a *direct inference attack* discussed later in Section III-B. For instance, to prevent inference, the label should be mapped to a random number first. More details will be discussed in Section IV-A.

Distributed data source. Our CryptoNN framework does not limit each neural network model to correspond to only one client. Actually, the model can be trained over multiple, distributed data sources, where the only requirement is that the training data should be encrypted using the same public key.

Comparison with existing solutions. As our CryptoNN framework is a crypto approach to deal with privacy issues in a machine learning model, we compare the existing homomorphic encryption (HE) based solutions here rather than secure multiparty computation based approaches, which will be discussed in the related work section later.

Intuitively, both the HE-based solutions and our CryptoNN support the computation over encrypted data, and hence, they do not reveal the data to the server. The main difference is that the result of the computation using HE is still in encrypted form, while the result of the computation using CryptoNN is plaintext. Such a difference allows CryptoNN to do secure computation in the training phases, while other HE-based models only support prediction over encrypted data. Besides, the intermediate data is not ciphertext in the hidden layers, which intuitively indicates that CryptoNN will be more efficient than the HE-based solutions.

In the rest of this section, we first present the proposed underlying crypto scheme, namely, *functional encryption*; then we present the construction of the secure matrix computation based on this functional encryption crypto-system. Finally, we present our CryptNN framework that is built using the proposed secure components, where a general neural network model can be transferred to CryptoNN to enable training over encrypted data.

B. Functional Encryption for Basic Operations

Even though the functional encryption for inner-product can be employed to compute the dot-product over encrypted matrix, there is still lack of functional encryption schemes to support element-wise computation. Here, we propose our construction of the functional encryption scheme supporting basic operations, i.e., addition, subtraction, multiplication, and division. Suppose the basic functionality is defined as follows:

$$f_{\Delta \in [+,-,*,/]}(x,y) = x\Delta y,$$

where Δ denotes the specific computation between x and y. The functional encryption scheme for the basic functionality $f_\Delta(x,y)$ is defined as FEBO = (Setup, KeyDerive, Encrypt, Decrypt). Each algorithm is defined as follows:

- Setup(1^λ): It takes the security parameter λ as the input, and outputs the key pair (*mpk, msk*). It first generates the group samples $(\mathbb{G}, p, g) \leftarrow_\$ \text{GroupGen}(1^\lambda)$
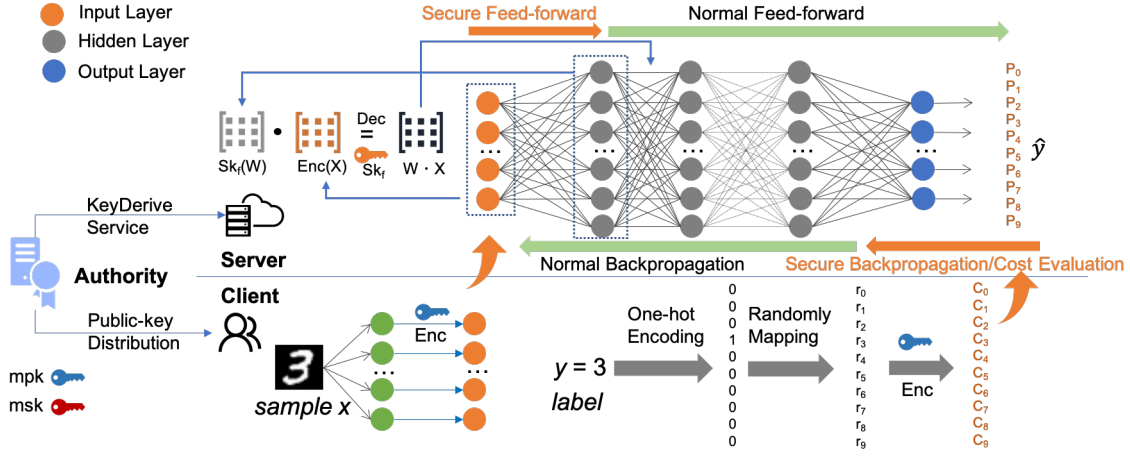
Fig. 1. An overview of the CryptoNN framework. Note that for the demonstration purpose, we just use one image sample to illustrate the secure process of the CryptoNN framework, as well as the one-hot encoding method for the label. Besides, as our CryptoNN framework only focuses on the data processing which takes place between the input layer and first hidden layer in the secure feed-forward step, and between the output layer and last hidden layer in the secure back-propagation/evaluation step, we omit the details of the neural network in the middle of hidden layers.

and randomly chooses the secret $s \leftarrow_{\$} \mathbb{Z}_p$. Then, it sets $msk = s$ and $mpk = (h, g)$, where $h = g^s$.

- KeyDerive $(mpk, msk, cmt, \Delta, y)$: The algorithm takes the master secert key, msk, the commitment, cmt, the input of function y as the inputs and outputs the secret key, sk_{f_Δ}. The algorithm has different key generation approaches according to different arithmetic computations. The function derived key sk_{f_Δ} is generated as follows:

$$sk_{f_\Delta} = \begin{cases} cmt^s \cdot g^{-y} & \text{if } \Delta = +, \\ cmt^s \cdot g^{y} & \text{if } \Delta = -, \\ (cmt^s)^{y} & \text{if } \Delta = *, \\ (cmt^s)^{y^{-1}} & \text{if } \Delta = /. \end{cases}$$

- Encrypt (mpk, x): The algorithm takes the pubic key, mpk, and the input of function f_Δ as the inputs, and outputs a commitment, cmt, and the ciphertext of x. To achieve that, it first randomly chooses a nonce, $r \leftarrow_{\$} \mathbb{Z}_p$, and generates the commitment $cmt = g^r$. The ciphertext is computed as $ct = h^r \cdot g^x$.

- Decrypt $(mpk, sk_{f_\Delta}, ct, \Delta, y)$: Based on the inputs - the public key mpk, the function derived key sk_{f_Δ} for function f_Δ with another input y, and the ciphertext ct - the algorithm recovers the discrete logarithm in basis g as follows:

$$g^{f_\Delta(x,y)} = \begin{cases} ct/sk & \text{if } \Delta = + \text{ or } -, \\ (ct)^{y}/sk & \text{if } \Delta = *, \\ (ct)^{y^{-1}}/sk & \text{if } \Delta = /. \end{cases}$$

As we explained in Section II-B, it is practical to recover the exponent, i.e., the result of $f_\Delta(x, y)$.

Correctness. For all $(mpk, msk) \leftarrow \text{Setup}(1^\lambda)$, all $x, y \in \mathbb{Z}_p, \Delta \in [+, -, *, /]$, for $sk_{f_\Delta} \leftarrow \text{KeyDerive}(msk, cmt, \Delta, y)$, and $cmt, ct \leftarrow \text{Encrypt}(mpk, x)$, we have Decrypt $(mpk, sk_{f_\Delta}, ct, \Delta, y)$ computed as follows:

$$\begin{cases} ct/sk = (h^r g^x)/((g^r)^s g^{(\mp y)}) = g^{x \pm y} & \text{if } \Delta = \pm, \\ (ct)^y/sk = (h^r g^x)^y/(g^r)^{sy} = g^{xy} & \text{if } \Delta = *, \\ (ct)^{y^{-1}}/sk = (h^r g^x)^{(y^{-1})}/(g^r)^{sy^{-1}} = g^{x/y} & \text{if } \Delta = /. \end{cases}$$

Remark. The construction of FEBO is derived from ElGamal encryption [28]. The security proof of FEBO scheme is presented in Section IV-A. The FEBO scheme can resist an attacker (i.e., unauthorized user without function key) trying to break the encrypted data, but cannot prevent the *direct inference attack*, where the attacker (i.e., authorized decryptor with function key) can infer the encrypted data $\text{Enc}(x)$ using the final function result $f(x\Delta y)$ and its own y in the four basic arithmetic computations. The issue will be addressed at the construction of framework level components.

C. Secure Matrix Computation

Matrix computation is the basic operation in the training and prediction phases of neural networks. To support matrix computation over encrypted data, we construct an encrypted matrix computation method, called *secure matrix computation* in the rest of the paper, by employing the functional encryption for inner-product proposed in [18] and our proposed functional encryption for the basic operations presented in Section III-B.

Suppose that we need to compute a function $f \in \mathcal{F}$, where $\mathcal{F}$ is the permitted set of functions over encrypted matrix $\mathbf{X}$ and $\mathbf{Y}$ that come from the *client* and the *server*, respectively. The permitted function set includes dot-product and element-wise arithmetic computation. The specific scheme is presented in Algorithm 1.

The scheme includes three parts: *pre-process-encryption* function, *pre-process-key-derivative* function, and *secure-computation* function. As a client, it just needs to encrypt the matrix using the *pre-process-encrypt* function and send out the ciphertext (lines 14-21). For the server, it first needs to decide the specific function $f \in \mathcal{F}$, and then request/prepare the function derived key $\mathbf{sk}_f$ from the *authority* (lines 25-30). When the encrypted data from the clients and the function derived key are ready, the *server* starts the processing over encrypted data using *secure-computation* function as described in lines 2-13.

Note that even though the secure dot-product computation can also be achieved using secure element-wise multiplication

---
Algorithm 1: secure matrix computation scheme

Input: X, Y, a permitted function $f \in \mathcal{F}$ over encrypted matrix.
Output: Computation result Z and pre-process result [[x]], [[X]], sk_f.

1 The authority initializes and distributes public key $mpk_{\text{FEIP}}, mpk_{\text{FEBO}}$
2 **function** *secure-computation([[x]], [[X]], f, $\mathcal{F}$, $sk_{f,y/Y}$, Y)*
3   initialize an empty matrix Z
4   **if** $f \in \mathcal{F}$ **and** *f is dot-product* **then**
5   **for** $i \leftarrow 0$ **to** size of $sk_{f,y}$ **do**
6              $\mathbf{y} := i$-th row of Y
7   **for** $j \leftarrow 0$ **to** size of [[x]] **do**
8                  $\mathbf{Z}[i][j] := \texttt{FEIP.Decrypt}\,(mpk_{\text{FEIP}},\mathbf{x}[j],sk_{f,\mathbf{y}}[i],\mathbf{y})$
9   **else**
10   **for** $i \leftarrow 0$ **to** row of [[X]] **do**
11   **for** $j \leftarrow 0$ **to** column of [[X]] **do**
12                  $\mathbf{Z}[i][j] := \texttt{FEBO.Decrypt}\,(mpk_{\text{FEBO}},sk_{f,\mathbf{Y}}[i][j],$
                 $[[\mathbf{X}]][i][j],\mathcal{F},\mathbf{Y}[i][j])$
13   **return** Z

14 **function** *pre-process-encryption(X)*
15   initialize an empty list [[x]] and an empty matrix [[X]]
16   **for** $i \leftarrow 0$ **to** column size of X **do**
17          $\mathbf{x} := i$-th column of X
18          $[[\mathbf{x}]][i] := \texttt{FEIP.Encrypt}\,(mpk_{\text{FEIP}},\mathbf{x})$
19   **for** $j \leftarrow 0$ **to** row size of X **do**
20              $[[\mathbf{X}]][j][i] := \texttt{FEBO.Encrypt}\,(mpk_{\text{FEBO}},\mathbf{X}[j][i])$
21   **return** [[x]], [[X]]

22 **function** *pre-process-key-derivative(Y, f, $\mathcal{F}$)*
23   initialize an empty list $sk_{f,\mathbf{y}}$ and matrix $sk_{f,\mathbf{Y}}$
24   **for** $i \leftarrow 0$ **to** row size of Y **do**
25   **if** $f \in \mathcal{F}$ **and** *f is dot-product* **then**
26              $\mathbf{y} := i$-th row of Y
27              $sk_{f,\mathbf{y}}[i] := sk_f$ from authority by $\mathbf{y}$
28   **else**
29   **for** $j \leftarrow 0$ **to** column size of Y **do**
30                  $sk_{f,\mathbf{Y}}[i][j] := sk_f$ from authority by $\mathbf{Y}[i][j]$
31   **return** $sk_{f,\mathbf{y}}$ or $sk_{f,\mathbf{Y}}$

Algorithm 2: The CryptoNN framework

Input: The encrypted training data [[x]] and label [[Y]], the layers information $\mathcal{L}$, the hyperparameters $\mathcal{H}$, the permitted function set $\mathcal{F}$ over encrypted data.
Output: the trained parameters $\mathcal{P}$.

1 **function** *CryptoNN([[x]], [[Y]], $\mathcal{L}$, $\mathcal{H}$, $\mathcal{F}$)*
2      $\mathcal{P} \leftarrow$ initialize parameters according to $\mathcal{L}$
3   **foreach** *iterator in total iterations* **do**
4          $sk_{\mathcal{F},\mathcal{P}^{[1]}} \leftarrow$ *pre-process-key-derive($\mathcal{P}^{[1]}$, $\mathcal{F}$)*
5          $\mathcal{R}_X \leftarrow$ *secure-computation([[x]], $\mathcal{F}$, $sk_{\mathcal{F},\mathcal{P}^{[1]}}$, $\mathcal{P}^{[1]}$)*
6          $\mathcal{A} \leftarrow$ normal feed-forward process using $\mathcal{R}_X$, $\mathcal{P}$
7          $sk_{\mathcal{F},\mathcal{A}^{[l]}} \leftarrow$ *pre-process-key-derive($\mathcal{A}^{[l]}$, $\mathcal{F}$)*
8          $\mathcal{R}_{\mathcal{A}^{[l]}} \leftarrow$ *secure-computation([[Y]], $\mathcal{F}$, $sk_{\mathcal{F},\mathcal{A}^{[l]}}$, $\mathcal{A}^{[l]}$)*
9   cost $\mathcal{C} \leftarrow$ cost evaluation using $\mathcal{R}_{\mathcal{A}^{[l]}}$, $\mathcal{A}$ on the cost function
10   grads $\mathcal{G} \leftarrow$ normal back-propagation using $\mathcal{R}_{\mathcal{A}^{[l]}}$, $\mathcal{A}$, $\mathcal{H}$
11          $\mathcal{P} \leftarrow$ update parameters using existing $\mathcal{P}$, and $\mathcal{G}$, $\mathcal{H}$
12   **return** $\mathcal{P}$

---

between the parameters and the encrypted data (i.e., lines 4-5 and lines 7-8) for each iteration. Such two rounds of secure computation only take place at the beginning of feed-forward processing and back-propagation processing, and hence, they are called ***secure feed-forward*** and ***secure back-propagation and evaluation***, respectively, in the rest of the paper (see Fig.1 as the example).

Scalability. CryptoNN is adaptive to any existing neural network model if the *secure feed-forward* and *secure back-propagation / evaluation* steps in such models can be narrowed down to process the basic matrix computation supported by the permitted function set $\mathcal{F}$. As we cannot enumerate all the existing neural network models to explain the general applicability of our proposed CryptoNN, here, we first use a simple neural network model for a binary classification task for illustration. Then, we present a concrete case: applying the proposed CryptoNN framework to the typical Convolutional Neural Network (CNN) in the next section.

Suppose the *secure feed-forward* step of the model computes $\mathbf{A} = \theta(\mathbf{Z}) = \theta(\mathbf{WX}+\mathbf{b})$, where $\theta(\cdot)$ is a sigmoid function $\theta(z) = \frac{1}{1+e^{-z}}$ and $\mathbf{A}$ is the output of the layer. The process in the *secure back-propagation/evaluation* step of the model includes the sigmoid function for the output layer, and the cost function is $E = \frac{1}{2}\sum_i(\hat{y}^{(i)} - y^{(i)})^2$, where $\hat{y}^{(i)}$ is the prediction and $y^{(i)}$ is the label. The *secure feed-forward* process can be split into two steps: (i) computing $\mathbf{WX}$ first, which is a dot-product supported by the secure matrix computation scheme; and then (ii) computing the remaining parts of the formula. For *secure evaluation*, it can compute $\hat{\mathbf{Y}}-\mathbf{Y}$ first, which is element-wise subtraction also supported by CryptoNN. To update the parameter for *secure back-propagation* we need to compute $\frac{\partial E}{\partial \mathbf{W}}$ according to gradient descent method. Based on the partial derivative rule, we know that

$$\frac{\partial E}{\partial \mathbf{W}} = \frac{\partial \mathbf{E}}{\partial \mathbf{A}} \cdot \frac{\partial \mathbf{A}}{\partial \mathbf{Z}} \cdot \frac{\partial \mathbf{Z}}{\partial \mathbf{W}},$$

such that,

$$\frac{\partial \mathbf{Z}}{\partial \mathbf{W}} = \mathbf{A}^{[l-1]}, \frac{\partial \mathbf{A}}{\partial \mathbf{Z}} = \mathbf{A}^{[l]}(1 - \mathbf{A}^{[l]}), \frac{\partial E}{\partial \mathbf{A}} = \mathbf{A}^{[l]} - \mathbf{Y},$$

method in our proposed scheme, we still separate it as an independent function here due to efficiency considerations. The related analysis is presented in Section IV. For simplicity, we just use the 2-dimensional array (i.e., matrix) to demonstrate the secure computations in the neural network model. Such a design is similar to the setting in the *NumPy*, which is the most widely used library for the scientific computing in the machine learning area. Note that the underlying decryption involves time-consuming operations in the secure computation scheme; however, our scheme also supports parallel computation. For instance, it just needs to make decryption code segment (line 8 and line 12) parallelized. Our experiments show that the parallelization will increase the efficiency significantly (see Section IV for more details).

D. CryptoNN Framework

The proposed CryptoNN framework is built on the secure matrix computation scheme and its underlying functional encryption schemes. As shown in Algorithm 2, we present the general CryptoNN. As the training data is encrypted, the CryptoNN framework inserts two rounds of secure computations presented in Section III-C to deal with the computation
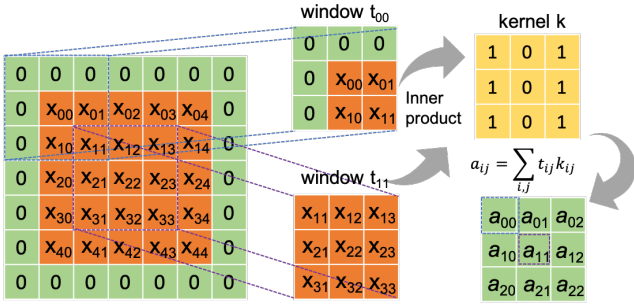
Fig. 2. An illustration of padding and convolution operations in the CNN model. Suppose that the original image size is $5 \times 5 \times 1$, and the padding is 1. The filter size is $3 \times 3 \times 1$. The stride is 2, hence, the output size is $3 \times 3 \times 1$.

where $\mathbf{A}^{[l-1]}$ is the output of the last hidden layer, and $\hat{\mathbf{Y}} = \mathbf{A}^{[l]}$ is the output (prediction) of the output layer. Obliviously, the computation related to label $\mathbf{Y}$ is the basic element-wise subtraction that is also supported by CryptoNN.

Prediction. We do not present the details of the prediction (or inference) phase of the CryptoNN framework here, as the prediction can be viewed as the sub-process of the training phases. To be specific, the prediction phase only covers the *secure feed-forward* and normal feed-forward, and gets the output of the neural network model. As the trained model of CryptoNN is also plaintext, it can also be integrated with existing homomorphic encryption-based solutions at the prediction phase. Hence, it provides a flexible choice of privacy setting, namely, keeping the predicted label confidential or not. If the user prefers the confidential predicted label, the HE-based prediction is employed; otherwise, the FE-based prediction can be applied.

E. A Concrete Case: CryptoCNN

Here, we use another concrete and relatively more complicated neural network model, namely, LeNet-5, a classic convolutional neural network [29] for multiple classification, to illustrate the scalability of CryptoNN.

Except for the input and output layer, the LeNet-5 model includes five hidden layers: convolutional layer (C1), the average pooling layer (S2), the convolutional layer (C3), the average pooling layer (S4) and fully connected layer (C5). As our CryptoNN only focuses on the *secure feed-forward* and *secure back-propagation / evaluation* steps in the model, we need to focus on the C1 and the output layers. To be concrete, we need to address the *padding* and *convolution* operations in the *secure feed-forward* step, and *softmax output* function, *softmax cross-entropy loss* function in the *secure back-propagation/evaluation* step.

1) secure feed-forward step: It takes place in the first convolutional layer. As depicted in Fig.2, we illustrate classic *padding* and *convolution* operations. Typically, the original image is surrounded by *zero-padding* data if the padding is required in the CNN model. Next, a sliding window with the same size of the filter (a.k.a. kernel) is set to capture parts of the image, and then next step is to do the convolution

---

Algorithm 3: secure convolution scheme

Input: training data $\mathbf{X}$, filter $\mathbf{K}$, a permitted function $f \in \mathcal{F}$ over encrypted matrix, public parameter mpk_{FEIP}
Output: the result $\mathbf{Z}$.

1   Server prepare the $\mathbf{sk}_{f,\mathbf{K}}$ for element-wise product for filter $\mathbf{K}$
2   **function** *secure-convolution(T, $sk_{f,K}$, K)*
3   initialize an empty matrix $\mathbf{Z}$
4   **foreach** *element [[t]] in T* **do**
5       $\mathbf{k} \leftarrow$ transfer $\mathbf{K}$ to vector
6   t := FEIP.Decrypt (mpk_{FEIP}, $\mathbf{sk}_{f,\mathbf{k}}$, [[t]], f, $\mathcal{F}$, $\mathbf{k}$)
7   fill t into $\mathbf{Z}$ at right position
8   **return** $\mathbf{Z}$

9   **function** *pre-process-encryption(X)*
10   initialize an empty window list $\mathbf{T}$
11     $\mathbf{X}' \leftarrow \mathbf{X}$ with padding
12   **repeat**
13       $\mathbf{t} \leftarrow$ abstract window matrix from $\mathbf{X}'$ and transfer to vector
14   append FEIP.Encrypt (mpk_{FEIP}, $\mathbf{t}$) into $\mathbf{T}$
15   **until** *slide window finished;*
16   **return** $\mathbf{T}$

17   **function** *pre-process-key-derivative(K, f, $\mathcal{F}$)*
18     $\mathbf{k} \leftarrow$ transfer $\mathbf{K}$ to vector
19     $\mathbf{sk}_{f,\mathbf{K}} :=$ request key from authority by $\mathbf{k}$
20   **return** $\mathbf{sk}_{f,\mathbf{K}}$

---

with the filter to fill the corresponding element in the output volume. In particular, the computation is the sum of element-wise product of each corresponding element in the two matrix, namely, $a_{ij} = \sum_{i,j}(t_{ij}k_{ij})$, where t_{ij} and k_{ij} are the elements from the window and filter volume, respectively. The sliding window will keep moving according to the stride length.

In the CryptoNN framework, the original image (i.e., training data) is pre-processed under the protection of functional encryption. Thus, the padded image will be a "mixed" matrix, as depicted in Fig.2, where the yellow element is the encrypted data, while the green element is the plaintext *zero-padding* data. When doing the convolution, there are two types of sliding windows: fully encrypted window and partially encrypted window, e.g., t_{11} and t_{00}, respectively, as shown in Fig.2. To tackle such an issue, we present the *secure convolution computation* scheme, as shown in Algorithm 3. In essence, it is similar to the secure matrix computation scheme, as the convolution operation of two matrices can be converted to the computation of dot-product of two vectors. As the architecture is fixed in the adopted CNN model, the *client* needs to learn the padding strategy and the filter size from the *server*, and then pre-process the training data according to the filter size (lines 9-16). The *server* first prepares the key $\mathbf{sk}_{f,\mathbf{K}}$ based on the filter $\mathbf{K}$ (lines 17-20), and then executes the secure convolution operation for the corresponding filter/kernel (lines 2-8). Note that we only present the *one-filter* case in the algorithm, it is obviously applicable to *multi-filter* case.

2) secure back-propagation/evaluation step: It takes place at the output layer. We assume that the output is the *softmax* function, namely, $p_i = \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}$, where p_i denotes the probability of data x belonging to the class i, and N denotes the total number of categories. Here, a_i is the element of the output vector $\mathbf{a}$ of the fully connected layer for data

x. Also, suppose that the label y is encoded as a vector $\mathbf{y}$ using the one-hot method. Thus, the *cross-entropy loss* is $L = -\sum_{i=1}^{N} y_i \log p_i$, where y_i is the component of label $\mathbf{y}$ and p_i is the component of output vector $\mathbf{p}$ of the softmax function.

In the CryptoNN framework, $\mathbf{y}$ is encrypted using functional encryption. However, obviously, the loss function $L = -\langle \mathbf{y}, \mathbf{p}' \rangle$ is a kind of inner-product computation that is supported by CryptoNN, where $\mathbf{p}'$ is the logarithmic vector of $\mathbf{p}$.

To do *secure back-propagation* for the parameter update, it needs to compute $\frac{\partial L}{\partial a_j} = \frac{\partial L}{\partial p_i} \frac{\partial p_i}{\partial a_j}$. Briefly, we can calculate the partial derivative of $\frac{\partial p_i}{\partial a_i}$ first and the result is

$$\frac{\partial p_i}{\partial a_j} = \begin{cases} p_i(1 - p_j) & \text{if } i = j, \\ -p_j p_i & \text{if } i \neq j. \end{cases}$$

Then, $\frac{\partial L}{\partial a_j} = p_i - y_i$, and the vectorized expression for all labels is $\frac{\partial L}{\partial \mathbf{A}} = \mathbf{P} - \mathbf{Y}$. The computation related to label $\mathbf{Y}$ is the basic element-wise subtraction that is also supported by CryptoNN.

IV. IMPLEMENTATION AND EVALUATION

A. Security Analysis

The Discrete Diffie-Hellman assumption. Suppose that the group triplet $(\mathbb{G}, p, g)$ is generated by a PPT algorithm with the security parameter 1^λ, where $\mathbb{G}$ is the group of a λ-bit prime order p, and g is the generator. The Discrete Diffie-Hellman assumption states that the tuples (g, g^a, g^b, g^{ab}) are computationally indistinguishable from (g, g^a, g^b, g^c), where $\{a, b, c\} \in \mathbb{Z}_p$ are chosen independently and uniformly at random.

Theorem 1 establishes the security of FEBO scheme.

Theorem 1. *If the Discrete Diffie-Hellman problem is hard (i.e., DDH assumption holds), the constructed FEBO scheme has selective security against chosen-plaintext attacks (IND-CPA).*

Proof. For the sake of contradiction, assume that there exists an adversary $\mathcal{A}$ that can break the IND-CPA security of FEBO scheme with non-negligible advantage.

Given the adversary $\mathcal{A}$ that runs in time t and has advantage ϵ, we can construct an adversary $\mathcal{B}$ for the DDH problem that runs in time $t + O(1)$ and also has the advantage ϵ. Then $\mathcal{B}(g, h_1 = g^a, h_2 = g^r, h_3 = g^c)$ is constructed as follows: (i) set the *mpk* as (g, h_1), $b \leftarrow_\$ \{0, 1\}$, and ciphertext as $(h_2, h_3 \cdot g^{x\Delta y})$; (ii) run $\mathcal{A}(mpk, c)$ to get the output b'; (iii) if $b = b'$, $\mathcal{B}$ guesses it as the valid DDH tuple; otherwise, $\mathcal{B}$ guesses it is the random DDH tuple. If $c = ra$, then the ciphertext is a legitimate encryption of $g^{x\Delta y}$. If c is uniformly distributed, independent of a and r, then the ciphertext is just the original output. This makes $\mathcal{B}$ a perfect simulator of a random oracle in this case. Thus, $\mathcal{B}$ also has the advantage ϵ to break the DDH assumption, which violates our security assumption. Thus, the adversary $\mathcal{A}$ does not have the non-negligible advantage to break the FEBO scheme. $\square$

As the FEIP scheme has been employed in the work proposed in [18], where the security proof has been presented, we do not provide related security analysis here.

Security analysis of CryptoNN. The underlying crypto scheme of CryptoNN is the functional encryption scheme, for which the security has been explained above. Here, we present the security analysis of CryptoNN at a high level. The *authority* is the third party trusted authority and is supposed to be not in collusion with anyone. From the perspective of the *server*, it can only acquire the mapping relationship between the training data and the label, where the original training/label data collected from multiple data sources is protected by the functional encryption. Even for the simple label that has high similarity, for instance, one label maps to a set of training data, the encrypted result is uniformly distributed in the ciphertext space at random for each same label. Thus, such mapping relationship does not reveal any private information.

The fact that the output of the *secure feed-forward* and *secure back-propagation* steps is the plaintext also indicates a trade-off problem. On the one hand, it will increase the efficiency of training a neural network model. On the other hand, the output of the first hidden layer may include the intermediate data that can be used to infer partial information. A pre-processing approach such as a randomly mapping as mentioned in Section III can avoid the direct inference for the label, while the inner-product result cannot be used to directly infer the factor. The content related to complex inference attack such as measuring neural network memorization and extracting secrets is beyond the scope of this paper. We refer the reader to [30] for more details and corresponding countermeasures.

B. Performance Evaluation

1) Prototype implementation and test platform: To verify the functionality and evaluate the performance, we implement a prototype of the CryptoNN framework. The underlying crypto system is implemented using the Charm library [31]. Charm is a Python-based crypto toolkit and its underlying numerical calculations rely on GMP library. The neural network model is implemented using Numpy , a scientific computation library.

The test platform is a personal computer with Intel Core i7 processor, 16GB memory and macOS. Note that the training process of the model only relies on the CPU. In all the experiments, the security parameter is set to 256-bit. The time measurement for program execution is based on the built-in *time* package of python.

2) Performance of secure matrix computation: We evaluate the performance of the secure matrix computation scheme over the simulated encrypted data. Due to consideration of page limitation, although the scheme supports several arithmetic computations, we only present the evaluation result of three typical arithmetic computations, namely, element-wise addition, element-wise product, and dot-product. The execution times of those operations are depicted in Fig.3, Fig.4 and Fig.5.
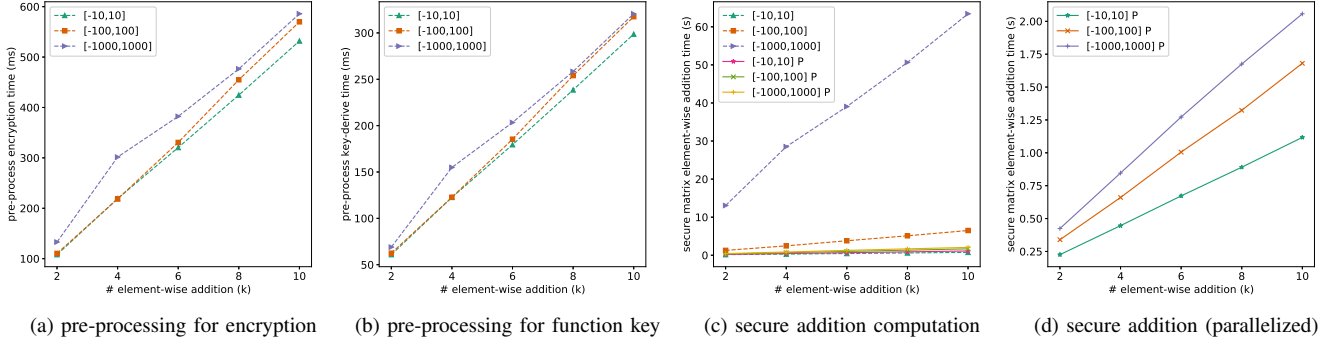
(a) pre-processing for encryption   (b) pre-processing for function key   (c) secure addition computation   (d) secure addition (parallelized)

Fig. 3. The time cost of element-wise addition in secure matrix computation scheme.

(a) pre-processing for encryption   (b) pre-processing for function key   (c) secure multiplication computation (d) secure multiplication (parallelized)
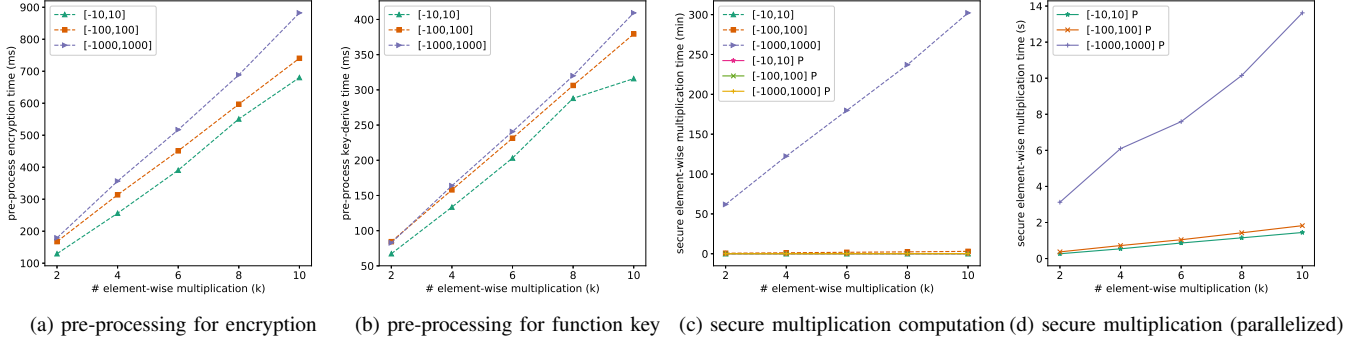
Fig. 4. The time cost of element-wise multiplication in secure matrix computation scheme.

(a) pre-processing for encryption   (b) pre-processing for function key   (c) secure dot-product computation   (d) secure dot-product (parallelized)
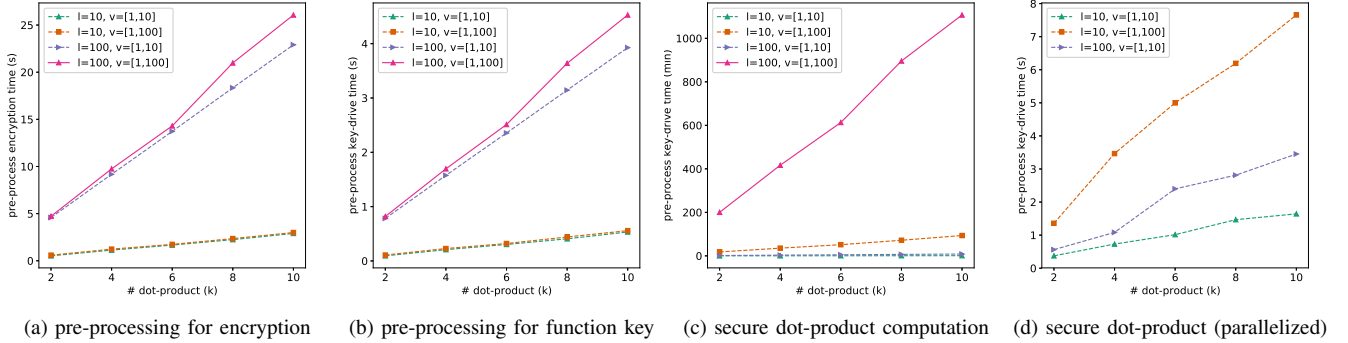
Fig. 5. The time cost of dot-product in secure matrix computation scheme.
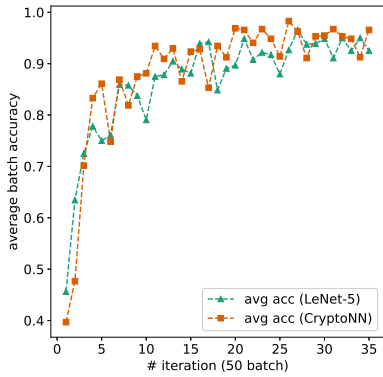
Fig. 6. Comparison of average batch accuracy between original LeNet-5 and CryptoCNN.

TABLE III
THE ACCURACY AND TRAINING TIME

model	epoch 1 (acc)	epoch 2 (acc)	training time
LeNet-5	93.04%	95.48%	4h
CryptoCNN	93.12%	95.49%	57h

measure the pre-processing times for encryption, generating function derived key, and the final secure computation, which are executed by the *client*, *authority*, and *server*, respectively. As depicted in Fig.3a and Fig.4a, the encryption time is nearly linear to the element size in the matrix, where the x-axis represents the element size (k), and the y-axis denotes the processing time in ms. Similarly, the execution time of pre-processing for the function derived key and element-wise computation also has a linear characteristic. As shown in Fig.3c, Fig.4c, and Fig.5c, the secure element-wise and dot-product computation are time-consuming. To tackle that issue,

Specifically, the plaintext matrix is generated randomly in a specific range, as described in the legend of each figure. We

we apply parallelization technique in the implementation. As shown in Fig.3d, Fig.4d, and Fig.5d, the parallelization technique makes secure computation more practical, where the execution time of secure dot-product can be reduced from near 90 mins to 8 seconds.

Communication overhead of key generation. Suppose that for training a two-class classification NN model with k units in the first hidden layer, the data set is $X_{m \times n}$, where m is sample size and n is feature size. For each training iteration, the *server* sends $k \times n \times |w|$ to the *authority* and acquires the key with size of $k \times |\text{sk}|$, where $|w|$ and $|\text{sk}|$ are the size of one weight parameter and derivative key, respectively.

3) Performance of CryptoNN: We implement an instance of the CryptoNN framework, namely, CryptoCNN model, as described in Section III-E. The model is built on the LeNet-5 architecture. We train the model on the classic MNIST dataset [32] including a training set of 60000 examples, and a test set of 10000 examples, and then compare the performance on the test set. Note that since the underlying functional encryption does not support floating point number computation and has lower efficiency on integers with longer lengths, we discard some precision on the parameter. For instance, for a floating point parameter, we only keep two-decimal places approximately and then transfer the floating point number to the integer for the underlying crypto related computation. Here, we only train two epochs using stochastic gradient descent method, where the batch size is set to 64. The comparison of the accuracy and total training time is shown in TABLE III. Also, the average batch accuracy of each model is depicted in Fig.6. We can conclude that CryptoCNN has similar accuracy compared to the original model, while the training time is much longer than the original model due to the time-consuming cryptographic computations.

V. RELATED WORK

Functional Encryption. The concept of functional encryption was proposed by Amit Sahai and Brent Waters in [33], and the formal definition was presented by Boneth et al. in [16]. As a new technique for public-key cryptography [17], functional encryption can be viewed as the abstraction of most of the existing encryption schemes such as attribute-based encryption [34], inner production function [35], and order-revealing encryption [36]. Generally, unlike traditional encryption schemes, where decryption reveals all or nothing, in a functional encryption scheme, the decryption keys may reveal only partial information about the plaintext, for instance, a computed result of a specific function.

Several constructions of functional encryption has been proposed in the literature to deal with different functions such as inner-product [25], comparing the order [16], and access control [37]. On the other hand, researchers also have focused on the functional encryption construction using different existing techniques such as multi-party computation [38] and multilinear maps [23], [24], or using functional encryption to construct other cryptographic schemes such as indistinguishablility obfuscation [22] and program obfuscation

[23]. However, most of these schemes are in theoretical stages and not practical enough for applications. Recently, more progressive approaches such as multi-input functional encryption [19], functional encryption using Intel SGX [39] and practical implementation [18], [25] have been proposed. To achieve secure function computation, the work proposed in [18] is one of the underlying crypto systems in our proposed CryptoNN framework.

Secure Machine Learning. The development of machine learning techniques, especially the emerging deep learning approaches [1], [27] have enabled applications to be more intelligent than before. The combination of machine learning and security study can be seen in two directions: (i) security issues in artificial intelligence, and (ii) artificial intelligence for increasing the capability of existing security mechanisms [40]. A series of secure or privacy-preserving machine learning models have been proposed in [2]–[15]. Note that we only cover the discussion about the privacy-preserving machine learning models, rather than other security issues in AI such as adversary machine learning.

Most of the existing privacy-preserving approaches adopt either secure multi-party computation protocol (e.g., [5], [6]) or homophobic encryption (such as [3], [9]–[15]). Neither of them supports training the machine learning model over encrypted data, except for the solution proposed in [2], where the solution only supports limited machine learning model and also relies on the homomorphic encryption and the associated secure protocol designed. To address the gap of training a machine learning model over encrypted data, our proposed CryptoNN focuses on training a secure deep learning model over the data encrypted using functional encryption scheme.

VI. CONCLUSION

Emerging neural networks based machine learning models such as deep learning and its variants have shown tremendous potential for many application domains that rely or use huge amounts of users' privacy sensitive data. To tackle the serious privacy concerns of the collected data, several privacy-preserving machine learning approaches that use either secure multi-party computation or homomorphic encryption as the underlying mechanisms have been proposed in the literature. In this paper, we have proposed a novel CryptoNN framework to support the training of neural networks over encrypted data using emerging functional encryption schemes. The security analysis and performance evaluation show that CryptoNN achieves the privacy goal, as well as model accuracy. One future direction is to explore the scalability of our CryptoNN framework for more complex datasets, and various other neural network models. Another future work is the use of different hardware support for CryptoNN.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[2] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data." in *NDSS*, 2015.

[3] 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.

[4] A. Mirhoseini, A.-R. Sadeghi, and F. Koushanfar, "Cryptoml: Secure outsourcing of big data machine learning applications," in *Hardware Oriented Security and Trust (HOST), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 149–154.

[5] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: Scalable provably-secure deep learning," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

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

[7] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. ACM, 2015, pp. 1310–1321.

[8] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 308–318.

[9] 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.

[10] F.-J. González-Serrano, A. Amor-Martín, and J. Casamayón-Antón, "Supervised machine learning using encrypted training data," *International Journal of Information Security*, vol. 17, no. 4, pp. 365–377, 2018.

[11] A. Wiesberg and F. Armknecht, "Unsupervised machine learning on encrypted data," *Cryptology ePrint archive*, vol. 2018, pp. Report–411, 2018.

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

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

[14] H. Park, P. Kim, H. Kim, K.-W. Park, and Y. Lee, "Efficient machine learning over encrypted data with non-interactive communication," *Computer Standards & Interfaces*, vol. 58, pp. 87–108, 2018.

[15] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 1209–1222.

[16] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: Definitions and challenges," in *Theory of Cryptography Conference*. Springer, 2011, pp. 253–273.

[17] ——, "Functional encryption: a new vision for public-key cryptography," *Communications of the ACM*, vol. 55, no. 11, pp. 56–64, 2012.

[18] M. Abdalla, F. Bourse, A. De Caro, and D. Pointcheval, "Simple functional encryption schemes for inner products," in *IACR International Workshop on Public Key Cryptography*. Springer, 2015, pp. 733–751.

[19] S. Goldwasser, S. D. Gordon, V. Goyal, A. Jain, J. Katz, F.-H. Liu, A. Sahai, E. Shi, and H.-S. Zhou, "Multi-input functional encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2014, pp. 578–602.

[20] D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, and J. Zimmerman, "Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 563–594.

[21] B. Waters, "A punctured programming approach to adaptively secure functional encryption," in *Annual Cryptology Conference*. Springer, 2015, pp. 678–697.

[22] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters, "Candidate indistinguishability obfuscation and functional encryption for all circuits," *SIAM Journal on Computing*, vol. 45, no. 3, pp. 882–929, 2016.

[23] B. Carmer, A. J. Malozemoff, and M. Raykova, "5gen-c: multi-input functional encryption and program obfuscation for arithmetic circuits," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 747–764.

[24] K. Lewi, A. J. Malozemoff, D. Apon, B. Carmer, A. Foltzer, D. Wagner, D. W. Archer, D. Boneh, J. Katz, and M. Raykova, "5gen: A framework for prototyping applications using multilinear maps and matrix branching programs," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 981–992.

[25] S. Kim, K. Lewi, A. Mandal, H. Montgomery, A. Roy, and D. J. Wu, "Function-hiding inner product encryption is practical," in *International Conference on Security and Cryptography for Networks*. Springer, 2018, pp. 544–562.

[26] D. Terr, "A modification of shanks baby-step giant-step algorithm," *Mathematics of Computation*, vol. 69, no. 230, pp. 767–773, 2000.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[28] 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.

[29] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[30] N. Carlini, C. Liu, J. Kos, Ú. Erlingsson, and D. Song, "The secret sharer: Measuring unintended neural network memorization & extracting secrets," *arXiv preprint arXiv:1802.08232*, 2018.

[31] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin, "Charm: a framework for rapidly prototyping cryptosystems," *Journal of Cryptographic Engineering*, vol. 3, no. 2, pp. 111–128, 2013. [Online]. Available: http://dx.doi.org/10.1007/s13389-013-0057-3

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

[33] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2005, pp. 457–473.

[34] R. Ostrovsky, A. Sahai, and B. Waters, "Attribute-based encryption with non-monotonic access structures," in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 195–203.

[35] J. H. Park, "Inner-product encryption under standard assumptions," *Designs, Codes and Cryptography*, vol. 58, no. 3, pp. 235–257, 2011.

[36] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. ACM, 2004, pp. 563–574.

[37] A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters, "Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010, pp. 62–91.

[38] S. Gorbunov, V. Vaikuntanathan, and H. Wee, "Functional encryption with bounded collusions via multi-party computation," in *Advances in Cryptology–CRYPTO 2012*. Springer, 2012, pp. 162–179.

[39] B. Fisch, D. Vinayagamurthy, D. Boneh, and S. Gorbunov, "Iron: functional encryption using intel sgx," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 765–782.

[40] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognition*, vol. 84, pp. 317–331, 2018.