Privacy-Preserving Inference for Public Neural Networks

Mihail-Iulian Plesa[1][0000–0001–5954–7199], Robert Poenaru[1][0000–0003–0692–4194], Sebastian Irimia[1], Simona David[1][0009–0001–3291–8241], and Andrei Farcasanu[1]

Orange Services, Strada Gara Herăstrău, București 077190, Romania
mihail.plesa@orange.com

Abstract. In this paper, we present a lightweight symmetric encryption scheme specifically designed to support homomorphic operations with respect to constant multiplication. The scheme is based on the principles of singularization, a moving target defense strategy designed to safeguard resource-constrained devices. It allows clients to outsource linear computations to remote servers while requiring only a simple addition operation for encryption, making it highly efficient and well-suited for devices with limited resources. Additionally, we propose a remote key generation protocol to address scenarios in which the client lacks the computational capacity to generate keys. Building on this encryption scheme, we develop a protocol for privacy-preserving neural network inference, particularly for cases where the model parameters are public but the network is accessed through a service. Experimental results demonstrate that our protocol achieves greater efficiency compared to existing approaches based on secure multi-party computation or homomorphic encryption.

Keywords: Privacy · Neural Network Inference · Moving Target Defense · Cryptography

1 Introduction

Machine learning, particularly neural networks, is now widely integrated into various software products. With large-scale deployments, especially in cloud-based models, safeguarding the confidentiality of user input has become critically important, driven by regulations such as the European General Data Protection Regulation (GDPR). While protecting trained models was equally important in the past, the emergence of openly available models has made this concern less relevant.

Although solutions based on Secure Multi-Party Computation (SMPC) such as CrypTen and Fully Homomorphic Encryption (FHE) like Concrete-ML offer strong security guarantees, their current running times are not optimal for practical applications.

Our approach is designed to address real-world requirements where model parameters are publicly available [18], but the client accesses the network itself as

a service. In this setup, the client uploads their data to the service to utilize the network. While running the network locally on the client side would ensure the confidentiality of the client's data, this approach is often impractical due to the significant computational resources required. Existing privacy-enhancing technologies introduce substantial overhead to the performance of privacy-preserving protocols [10, 13, 9, 11], as they are typically designed to protect both the confidentiality of the data and the model. However, in our scenario, protecting the confidentiality of the model is unnecessary, allowing for a more efficient solution.

2 Related work

The field of privacy-preserving computations has advanced significantly, particularly in machine learning. Homomorphic encryption enables computations on encrypted data, as demonstrated by CryptoNets [6].

SecureML [15] and SecureNN [7] leverage secure multi-party computation for privacy-preserving machine learning. Slalom [19] uses Trusted Execution Environments for secure DNN inference, while MiniONN [12] transforms neural networks into oblivious ones for private predictions. CodedPrivateML [17] employs techniques like Lagrange coding for distributed privacy-preserving learning.

CrypTen [8] facilitates secure multi-party computation but faces computational overhead challenges. Concrete ML [20] uses Fully Homomorphic Encryption for privacy-preserving models, with performance limitations for deep neural networks.

Contributions

Our contributions are as follows:

1. We propose a novel lightweight encryption scheme that supports homomorphic operations with respect to constant multiplication.
2. We present a protocol for remote key generation, specifically designed for scenarios where the device lacks sufficient computational resources to generate keys locally.
3. We develop a protocol for privacy-preserving inference in feed-forward neural networks and benchmark its performance against two prominent industry solutions: Concrete-ML by Zama and CrypTen by Meta [20, 8].

The paper is structured in the following way: in Section 3, we present some preliminaries; in Section 4, we describe the encryption scheme; in Section 5, we introduce the privacy-preserving protocol. Section 6 illustrates the experiments, while Section 7 is left for the conclusions.

3 Preliminaries

3.1 Notations

We denote vectors using bold lower letters, matrices, and sets with upper letters, and scalars using regular lower letters. For a vector $\mathbf{x}$, we denote by $\mathbf{x}_\mathrm{i}$ the i-th

component of $\mathbf{x}$. We represent the length of $\mathbf{x}$ as $|\mathbf{x}|$. We denote by $s \xleftarrow{\$} S$ a uniform random sampling of s from the set S. $\mathbf{x} \cdot \mathbf{y}^{\mathrm{T}}$ represents the inner product between the row vector $\mathbf{x}$ and the column vector $\mathbf{y}$. We represent by $w \times \mathbf{x}$ or $w\mathbf{x}$ the scalar multiplication between the vector $\mathbf{x}$ and the scalar w, i.e., given $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ and w, $\mathbf{x} \times w = (\mathbf{x}_1 w, \mathbf{x}_2 w, \ldots, \mathbf{x}_n w)$. We demote by $[m]$ the encryption of m. A function $\mathsf{negl}(n)$ is called negligible if for every positive polynomial $p(\lambda)$, there exists an integer N such that for all $n > N$ we have $\mathsf{negl}(n) < \frac{1}{p(n)}$.

3.2 Homomorphic encryption

A homomorphic encryption (HE) scheme enables computations to be performed directly on encrypted data. The outcomes of these computations are ciphertexts that correspond to the results of operations conducted on the plaintext data [1]. In this manner, the confidentiality of the data is ensured throughout the computation process. An additive homomorphic encryption (AHE) scheme, defined over modular integers, is naturally homomorphic with respect to constant multiplication. The structure of an AHE scheme is given in Figure 1. We suppose that the AHE scheme used in this paper is IND-CPA secure [2].

Some AHE schemes include the Paillier cryptosystem [16] or the ElGamal cryptosystem [4].

3.3 Neural network model

We model a feed-forward pre-trained neural network with ℓ layers as a function $f : \mathbb{R}^{n^{[0]}} \to \mathbb{R}^{n^{[\ell]}}$, represented by a sequential composition of linear and non-linear functions:

$$f(\mathbf{x}^0) = \sigma^{[\ell]} \circ f^{[\ell]} \circ \sigma^{[\ell-1]} \circ f^{[\ell-1]} \circ \cdots \circ \sigma^{[2]} \circ f^{[2]} \circ \sigma^{[1]} \circ f^{[1]}(\mathbf{x}^0) \ ,$$

where:

- $\mathbf{x}^{[0]} \in \mathbb{R}^{n^{[0]}}$ is the input vector.
- $f^{[\ell]} : \mathbb{R}^{n^{[\ell-1]}} \to \mathbb{R}^{n^{[\ell]}}$ for $1 \leq \ell \leq L$ are linear functions, represented as $f^{[i]}(\mathbf{x}^{[\ell-1]}) = W^{[\ell]}\mathbf{x}^{[\ell-1]} + \mathbf{b}^{[\ell]}$, where $W^{[\ell]} \in \mathbb{R}^{n^{[\ell]} \times n^{[\ell-1]}}$ and $\mathbf{b}^{[\ell]} \in \mathbb{R}^{n^{[\ell]}}$ are the weight matrix and bias vector for layer ℓ, respectively.
- $\sigma^{[\ell]} : \mathbb{R}^{n^{[\ell]}} \to \mathbb{R}^{n^{[\ell]}}$ is the non-linear activation function of the layer, applied element-wise.
- $n^{[\ell]}$ represents the number of input neurons of layer $\ell + 1$.

Figure 2 displays a neural network with L layers. Each layer is fully connected to the next layer. The input layer (orange) holds the input vector $\mathbf{x}^{[0]} \in \mathbb{R}^{n^{[0]}}$, the output layer (green) produces the final output vector $\mathbf{x}^{[L]} \in \mathbb{R}^{n^{[L]}}$, the hidden layers perform an affine transformation (red) followed by the non-linear activation function applied element-wise (blue). Note that $n^{[L]}$ is also the number of outputs.

An additive homomorphic encryption scheme is a tuple AHE = (KeyGen, Enc, Dec) composed from the following algorithms:

1. KeyGen $\left(1^{\lambda}\right)$: The key generation algorithm receives as input the security parameter and returns a public-private key pair $(\mathrm{pk}, \mathrm{sk})$.
2. Enc $(\mathrm{pk}, \mathrm{m})$: The encryption algorithm receives the plaintext message m and the public key pk as inputs and outputs the ciphertext $[\mathrm{m}]$.
3. Dec $(\mathrm{sk}, [\mathrm{m}])$: The decryption algorithm receives the ciphertext message $[\mathrm{m}]$ and the secret key sk as inputs and outputs the plaintext m.

The scheme supports the following operations:

1. EvalAdd $([\mathrm{m}_1], [\mathrm{m}_2])$:
   This operation receives as inputs two ciphertexts $[\mathrm{m}_1]$ and $[\mathrm{m}_2]$ and returns the ciphertext that encrypts the sum of the underlying plaintext, i.e., $[\mathrm{m}_1 + \mathrm{m}_2]$.
2. EvalMulConstant $([\mathrm{m}], \alpha)$:
   This operation receives as inputs a ciphertext $[\mathrm{m}]$ and a constant α and outputs a ciphertext that encrypts the product between the underlying plaintext and the constant, i.e., $[\alpha \mathrm{m}]$.
3. EncDotVec $([\mathbf{m}], \mathbf{a})$: This operation receives as inputs a vector of ciphertexts $[\mathbf{m}]$ and a vector of constants $\mathbf{a}$ and outputs a ciphertext that encrypts the dot product between the underlying plaintext vector and the vector of constants, i.e., $[\mathbf{m} \cdot \mathbf{a}^{\mathrm{T}}]$.
4. EncMatMult $([\mathbf{m}], \mathrm{A}, \mathbf{b})$: This operation receives as inputs a vector of ciphertexts $[\mathbf{m}]$, a constant matrix A and a vector of constants $\mathbf{b}$ and outputs a vector of ciphertexts that encrypts linear transformation induced by the constant matrix and the vector of constants over the underlying plaintext vector, i.e., $[\mathrm{A}\mathbf{m} + \mathbf{b}]$.

Fig. 1. Main structure of an AHE scheme

4 The encryption scheme

In this section, we introduce a symmetric encryption scheme based on the hardness of the decisional subset-sum problem. The scheme is specifically designed to enable lightweight encryption and decryption operations while supporting homomorphic properties for constant multiplication. This encryption scheme serves as the foundation of our privacy-preserving inference protocol, which facilitates the execution of a neural network without compromising the confidentiality of the client's input.

The proposed encryption scheme is based on the concept of singularization, a recently introduced moving target defense strategy designed to protect cryptographic implementations on resource-constrained devices [5, 14]. Singularization aims to make every input to a running system unique, enhancing security by introducing confusion and forcing attackers to continuously reassess the attack surface, all without altering the system's internal workings. Leveraging this strat-

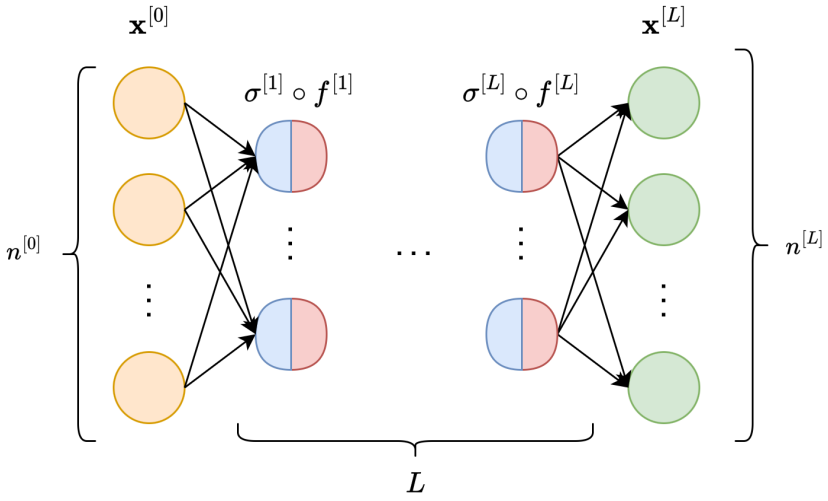

Fig. 2. A typical feed-forward neural network architecture.

egy, our encryption scheme introduces carefully designed noise to encrypt inputs, enabling a neural network to perform inference on the underlying data without modifying the neural network's implementation. In contrast, traditional homomorphic encryption schemes [16, 4] require operations in the ciphertext space that differ from those in the plaintext space. This limitation prevents the direct use of widely available and well-established neural network libraries for computations on encrypted data.

The proposed scheme uses only additions for encryption and decryption, delegating resource-intensive multiplications to an external party. To address memory demands during key generation, a remote key generation protocol is introduced, ensuring security without impacting encryption performance. Key features include:

1. The AHE scheme performs a constant number of encryptions during key generation, independent of data size.
2. Keys can be generated in advance, prior to the encryption phase.

We provide formal proof of the security of the encryption scheme and the remote key generation procedure.

The decisional subset sum problem. Given a multiset $\mathrm{R} = (\mathrm{r}_1, \mathrm{r}_2, \ldots, \mathrm{r}_\mathrm{n})$ with elements from a field $\mathbb{F}$ and an element $\mathrm{s} \in \mathbb{F}$, the decisional subset set problem (DSS for short) asks to determine whether there exists a subset $\mathrm{R}' \subseteq \mathrm{R}$ such that $\sum_{\mathrm{r_i} \in \mathrm{R}'} \mathrm{r_i} = \mathrm{s}$.

The DSS problem is conjectured to be NP-hard [3], i.e., given R, for all probabilistic polynomial-time algorithms $\mathcal{D}$ there is a negligible function negl such that:

$$\left| \mathsf{Pr}\left[\mathrm{R}' \subseteq \mathrm{R}, \mathrm{s} \leftarrow \sum_{\mathrm{r_i} \in \mathrm{R}'} \mathrm{r_i} : \mathcal{D}\left(\mathrm{R}, \mathrm{s}\right) = 1 \right] - \mathsf{Pr}\left[\mathrm{s} \xleftarrow{\$} \mathbb{F} : \mathcal{D}\left(\mathrm{R}, \mathrm{s}\right) = 1 \right] \right| \leq \mathsf{negl}\left(|\mathrm{R}|\right)$$

4.1 The cryptosystem

The symmetric cryptosystem $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ is formally defined in Figure 3. The correctness and homomorphic properties of the scheme are established in Theorems 1 and 2. Proofs are omitted as they follow directly from the definitions. The security is based on the DSS problem in Theorem 3. The proof is deferred to the full version of the paper. The cryptosystem is assumed to operate over a field $\mathbb{F}$.

Theorem 1 (Correctness). *Given* $[\mathrm{m}] \leftarrow \Pi.\mathsf{Enc}\left(\mathrm{s}, \mathrm{m}\right)$ *and* $\mathrm{m}' \leftarrow \Pi.\mathsf{Dec}\left(\mathrm{s}, [\mathrm{m}]\right)$ *we have:*

$$\mathrm{m}' = \mathrm{m}$$

Theorem 2 (Homomorphism). *Given* $[\mathrm{m}] \leftarrow \Pi.\mathsf{Enc}\left(\mathrm{s}, \mathrm{m}\right)$, $[\mathrm{mw}], \mathrm{s_e} \leftarrow \Pi.\mathsf{Eval}\left(\mathbf{p}, [\mathrm{m}], \mathrm{w}\right)$ *and* $\mathrm{m}' \leftarrow \Pi.\mathsf{Dec}\left(\mathrm{s_e}, [\mathrm{mw}]\right)$ *we have:*

$$\mathrm{m}' = \mathrm{mw}$$

Theorem 3 (Security). *The encryption scheme has indistinguishable encryptions in the presence of an eavesdropper, supposing that the DSS problem is hard with respect to the set of parameters* $\mathrm{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_\mathrm{n}\}$.

4.2 Remote key generation

The protocol enables a client to generate the secret key, s, without needing to process the parameter vector, $\mathbf{p}$, locally. The overall process is as follows: the random secret vector, $\mathbf{r}$, is encrypted using an additive homomorphic encryption scheme and sent to the server. Upon receiving the encrypted vector, the server computes the dot product between the encrypted vector and the public parameter vector of the scheme using $\Pi.\mathsf{EncDotVec}$. The server then returns the result to the client, which decrypts it to retrieve the secret key, s. Notably, the remote key generation protocol requires a constant number of AHE calls, regardless of the size of the vector being encrypted.

The remote key generation protocol is described in Figure 4. The proofs of correctness and security are deferred to the full version of the paper.

Theorem 4 (Correctness). *The output of the protocol is the dot product between the secret random vector and the public parameters vector, i.e.,* $\mathbf{r} \cdot \mathbf{p}$

The cryptosystem is the tuple of algorithm $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ composed from:

1. $\mathsf{Setup}\left(1^{\lambda}\right)$: The setup algorithm takes as input the security parameter λ and generates a vector $\mathbf{p}$ consisting of $n = \text{poly}(\lambda)$ field elements, chosen independently and uniformly at random. This vector $\mathbf{p}$ serves as the public parameters of the scheme.
2. $\mathsf{KeyGen}\,(\mathbf{p})$: The key generation algorithm takes the parameter vector $\mathbf{p}$ as input and produces the symmetric key s. The key is computed as the sum of a randomly selected subset of elements from $\mathbf{p}$:
$$s \leftarrow \mathbf{r} \cdot \mathbf{p}^{T}, \mathbf{r} \xleftarrow{\$} \{0,1\}^{n}$$
3. $\mathsf{Enc}\,(s, m)$: The encryption algorithm takes as inputs the secret key s and the plaintext message m. It outputs the ciphertext [m], computed as the sum of the plaintext and the secret key:
$$[m] \leftarrow m + s$$
4. $\mathsf{Dec}\,(s, [m])$: The decryption algorithm takes as inputs the secret key s and the ciphertext [m]. It recovers the plaintext m by subtracting the secret key from the ciphertext:
$$m \leftarrow [m] - s$$
5. $\mathsf{Eval}\,(\mathbf{p}, [m], w)$: The evaluation algorithm takes as inputs the parameter vector $\mathbf{p}$, the ciphertext [m], and a plaintext constant w. It outputs the ciphertext [mw] and the corresponding secret evaluation key:
$$[mw],\ s_e \leftarrow [m]w,\ s \times w$$

Fig. 3. The symmetric cryptosystem based on the subset sum problem

It is assumed that all parties are honest but curious, meaning they follow the protocol specifications but are interested in obtaining additional information beyond their designated outputs.

Theorem 5 (Security). *The remote key generation protocol described in Figure 4 achieves the ideal functionality* $\mathcal{F}_{\text{generate}}$ *in the presence of honest but curious adversaries.*

5 Privacy-preserving neural network evaluation

We present a protocol for evaluating a publicly pre-trained neural network using the proposed encryption scheme, involving a client and a server. The server hosts the neural network, and the client's input remains private.

The protocol divides neural network computations into linear and non-linear categories. The client and server engage offline in the remote key generation pro-

1. **Setup and AHE Key Generation.**
   (a) The client runs the key generation algorithm of an AHE scheme:
   $$\mathrm{pk}, \mathrm{sk} \leftarrow \mathrm{AHE.KeyGen}(1^{\lambda})$$
   (b) The server runs the setup algorithm of the symmetric encryption scheme and sends the length of the parameter vector to the client:
   $$\mathbf{p} \leftarrow \Pi.\mathsf{Setup}(1^{\lambda})$$
   $$\mathrm{S} \rightarrow \mathrm{C} : \mathrm{n} = |\mathrm{p}|$$
2. **Symmetric Key generation.**
   (a) The client generates the secret random vector:
   $$\mathbf{r} \xleftarrow{\$} \{0,1\}^{\mathrm{n}}$$
   (b) The client encrypts the secret random vector using the AHE scheme and sends the ciphertexts to the server:
   $$[\mathbf{r}]_{\mathrm{i}} \leftarrow \mathsf{AHE.Enc}(\mathsf{pk}, \mathbf{r}_{\mathrm{i}}), \forall 1 \leq \mathrm{i} \leq \mathrm{n}$$
   $$\mathrm{C} \rightarrow \mathrm{S} : [\mathbf{r}]$$
3. **Remote computation and result**
   (a) The server computes the dot product between the encrypted random vector and the public parameters of the symmetric scheme and returns the result to the client:
   $$[\mathbf{r} \cdot \mathbf{p}^{\mathrm{T}}] \leftarrow \mathsf{EncDotVec}([\mathbf{r}], \mathbf{p})$$
   $$\mathrm{S} \rightarrow \mathrm{C} : [\mathbf{r} \cdot \mathbf{p}^{\mathrm{T}}]$$
   (b) The client decrypts the result received from the server. The protocol outputs the secret key :
   $$\mathrm{s} \leftarrow \mathsf{AHE.Dec}(\mathsf{sk}, [\mathbf{r} \cdot \mathbf{p}^{\mathsf{T}}])$$
   $$\mathsf{Output}\ \mathrm{s}$$

Fig. 4. Remote key generation

tocol (Figure 4). For each layer $1 \leq \ell \leq \mathrm{L}$, $\mathrm{s}^{[\ell]}$ denotes the secret key encrypting the input $\mathrm{x}^{[\ell-1]}$, and $\mathrm{s}_{\mathrm{e}}^{[\ell_{\mathrm{i}}]}$ is the evaluation key for the i^{th} row of the weight matrix $\mathrm{W}^{[\ell]}$. The server performs the linear operation $\mathrm{W}^{[\ell]}\mathbf{x}^{[\ell-1]} + \mathbf{b}^{[\ell]}$ over encrypted data using $\Pi.\mathsf{EncMatMult}$, interpreting matrix-vector multiplication as dot products. This is supported by $\Pi.\mathsf{Eval}\left(\mathbf{p}, [\mathbf{x}^{[\ell-1]}], \mathbf{W}_{\mathrm{i}}^{[\ell]}\right)$, treating rows and vectors as elements of $\mathbb{R}^{\mathrm{n}^{[\ell-1]}}$. The protocol is detailed in Figure 5. The proofs of correctness and security are deferred to the full version of the paper.

Ideal functionality $\mathcal{F}_{\textbf{generate}}$

1. The client sends the secret random vector $\mathbf{r}$ to the TTP.
2. The TTP computes the dot product between $\mathbf{r}$ and the public parameters $\mathbf{p}$.
3. The TTP returns the result, i.e., $\mathbf{r} \cdot \mathbf{p}$ to the client.

For each layer, $1 \leq \ell \leq \mathrm{L}$ of the neural network, the client and the server execute:

1. The client encrypts the input of the current layer and sends it to the server:

$$[\mathbf{x}^{[\ell-1]}]_{\mathrm{i}} \leftarrow \Pi.\mathsf{Enc}\left(\mathrm{s}^{[\ell]}, \mathbf{x}_{\mathrm{i}}^{[\ell-1]}\right), \forall 1 \leq \mathrm{i} \leq \mathrm{n}^{[\ell-1]}$$

$$\mathrm{C} \rightarrow \mathrm{S} : [\mathbf{x}^{[\ell-1]}]$$

2. The server executes the linear operation of the layer over encrypted data and sends the result to the client:

$$[\mathrm{W}^{[\ell]}\mathbf{x}^{[\ell-1]} + \mathbf{b}^{[\ell]}] \leftarrow \Pi.\mathsf{EncMatMult}\left(\mathbf{x}^{[\ell-1]}, \mathrm{W}^{[\ell]}, \mathbf{b}^{[\ell]}\right)$$

$$\mathrm{S} \rightarrow \mathrm{C} : [\mathbf{y}^{[\ell]}] = [\mathrm{W}^{[\ell]}\mathbf{x}^{[\ell-1]} + \mathbf{b}^{[\ell]}]$$

3. The client decrypts the result, applies the activation function associated with the current layer, and repeats step 1:

$$\mathbf{y}_{\mathrm{i}}^{[\ell]} \leftarrow \Pi.\mathrm{Dec}\left(\mathrm{s}_{\mathrm{e}}^{[\mathrm{l_i}]}, [\mathbf{y}^{[\ell]}]_{\mathrm{i}}\right), \forall 1 \leq \mathrm{i} \leq \mathrm{n}^{[\ell]}$$

$$\mathbf{y}_{\mathrm{i}}^{[\ell]} \leftarrow \sigma^{[\ell]}\left(\mathbf{y}_{\mathrm{i}}^{[\ell]}\right)$$

$$\mathbf{x}^{[\ell]} \leftarrow \mathbf{y}^{[\ell]}$$

The client retrieves the result of the inference as $\mathbf{x}^{[\mathrm{L}]}$:

$$\text{Output } \mathbf{x}^{[\mathrm{L}]}$$

Fig. 5. Our protocol on privacy-preserving neural network evaluation

Theorem 6 (Correctness). *The result of the protocol described in Figure 5 is the same as the result produced by the network run directly over plaintext data:*

$$\mathbf{x}^{[\mathrm{L}]} = \sigma^{[\mathrm{L}]} \circ \mathrm{f}^{[\mathrm{L}]} \circ \cdots \circ \sigma^{[1]} \circ \mathrm{f}^{[1]}(\mathbf{x}^{[0]})$$

where $\mathrm{f}^{[\ell]}(\mathbf{x}^{[\ell-1]}) = \mathrm{W}^{[\ell]}\mathbf{x}^{[\ell-1]} + \mathbf{b}^{[\ell]}$, $1 \leq \ell \leq \mathrm{L}$.

Ideal functionality $\mathcal{F}_{\text{protocol}}$

1. The client transmits the input vector $\mathbf{x}^{[0]}$ to TTP.
2. The TTP computes the output of the neural network.
3. The TTP sends the result of the inference to the client.

Theorem 7 (Security). *The protocol described in Figure 5 achieves the ideal functionality $\mathcal{F}_{protocol}$ in the presence of honest but curious adversaries.*

6 Experiments

We compared our protocol with two leading industry solutions: Concrete-ML from Zama and CrypTen from Meta [20, 8]. The experiments were conducted using an NVIDIA A100-SXM4-80GB GPU and an Intel Xeon Gold 6240R CPU. The CPU was used for both the client and the server, while the GPU was only on the server side. We analyzed both the running time and memory usage for performing one image classification.

All solutions were tested using two models: VGG19 on images from the CIFAR10 dataset and a three-layer feed-forward neural network on images from the MNIST dataset. At the time of writing, Concrete-ML does not support GPU acceleration, so we were unable to test it with GPU support. Additionally, Concrete-ML did not complete the large VGG19 model within a reasonable time frame. Nevertheless, we highlight the fact that the solutions we chose to compare with are tailored for use cases in which the confidentiality of the model must also be preserved. This shows that our protocol is more suitable for practical scenarios in which the model is public but exposed through a service that the client can access.

The results are presented in Tables 1 and 2. Our solution is faster than both Concrete-ML and CrypTen. Regarding GPU acceleration, our protocol did not show a significant difference in running time, likely due to the large number of memory swaps between the CPU and GPU. However, this indicates that our protocol does not necessarily require a GPU to perform classifications in a reasonable amount of time, which can be advantageous in situations where a GPU is unavailable due to cost or other constraints.

Table 1. Benchmarks on the running time for one classification (seconds)

	GPU		CPU	
Approach	VGG19	FNN	VGG19	FNN
Our approach	0.32706	0.0066	0.35709	0.00135
CrypTen	12.092	0.5152	129.823	7.261
Concrete-ML	N/A	N/A	N/A	23.25063

Table 2. Benchmarks on the memory for one classification (GB)

	GPU		CPU	
Approach	VGG19	FNN	VGG19	FNN
Our approach	1.17	0.46	1.60	0.49
CrypTen	41.13	2.29	8.76	1.69
Concrete-ML	N/A	N/A	N/A	5

7 Conclusions

This paper introduces a lightweight symmetric encryption scheme, homomorphic for constant multiplication, designed for neural network evaluation. It ensures indistinguishable encryptions under the decisional subset sum problem and includes a remote key generation mechanism to address memory challenges.

We propose a protocol for privacy-preserving neural network inference with superior performance compared to Concrete-ML and CrypTen, supported by formal security proofs.

Future work includes enabling privacy-preserving large language models, improving efficiency in cryptographic methods like FHE, and secure multi-party computation for public model parameters.

References

1. Acar, A., Aksu, H., Uluagac, A.S., Conti, M.: A survey on homomorphic encryption schemes: Theory and implementation. ACM Computing Surveys (Csur) **51**(4), 1–35 (2018)
2. Bellare, M., Rogaway, P.: Introduction to modern cryptography. Lecture Notes (2001)
3. Bonnetain, X., Bricout, R., Schrottenloher, A., Shen, Y.: Improved classical and quantum algorithms for subset-sum. In: Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part II 26. pp. 633–666. Springer (2020)
4. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE transactions on information theory **31**(4), 469–472 (1985)
5. Gaber, C., Macariot-Rat, G., David, S., Wary, J.P., Cuaboz, A.: Position paper: Strengthening applets on legacy sim cards with singularization, a new moving target defense strategy. In: International Conference on Mobile, Secure, and Programmable Networking. pp. 71–74. Springer (2023)
6. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: International conference on machine learning. pp. 201–210. PMLR (2016)
7. Hesamifard, E., Takabi, H., Ghasemi, M.: Cryptodl: Deep neural networks over encrypted data. arXiv preprint arXiv:1711.05189 (2017)

8. Knott, B., Venkataraman, S., Hannun, A., Sengupta, S., Ibrahim, M., van der Maaten, L.: Crypten: Secure multi-party computation meets machine learning. Advances in Neural Information Processing Systems **34**, 4961–4973 (2021)
9. Kumar, G.S., Premalatha, K., Maheshwari, G.U., Kanna, P.R., Vijaya, G., Nivaashini, M.: Differential privacy scheme using laplace mechanism and statistical method computation in deep neural network for privacy preservation. Engineering Applications of Artificial Intelligence **128**, 107399 (2024)
10. Lee, H., Finke, D., Yang, H.: Privacy-preserving neural networks for smart manufacturing. Journal of Computing and Information Science in Engineering **24**(7), 071002 (2024)
11. Li, J., Yan, Y., Zhang, K., Li, C., Yuan, P.: Fpcnn: A fast privacy-preserving outsourced convolutional neural network with low-bandwidth. Knowledge-Based Systems **283**, 111181 (2024)
12. Liu, J., Juuti, M., Lu, Y., Asokan, N.: Oblivious neural network predictions via minionn transformations. In: Proceedings of the 2017 ACM SIGSAC conference on computer and communications security. pp. 619–631 (2017)
13. Liu, Q., Huang, Q., Chen, X., Wang, S., Wang, W., Han, S., Lee, P.P.: Ppstream: Toward high-performance privacy-preserving neural network inference via distributed stream processing. In: Proceedings of the 40th IEEE International Conference on Data Engineering (ICDE 2024) (2024)
14. Macario-Rat, G., Plesa, M.I.: Singularization: A new approach to designing block ciphers for resource-constrained devices. In: Meng, W., Yung, M., Shao, J. (eds.) Attacks and Defenses for the Internet-of-Things. pp. 155–167. Springer Nature Switzerland (2025)
15. Mohassel, P., Zhang, Y.: Secureml: A system for scalable privacy-preserving machine learning. In: 2017 IEEE symposium on security and privacy (SP). pp. 19–38. IEEE (2017)
16. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International conference on the theory and applications of cryptographic techniques. pp. 223–238. Springer (1999)
17. So, J., Güler, B., Avestimehr, A.S.: Codedprivateml: A fast and privacy-preserving framework for distributed machine learning. IEEE Journal on Selected Areas in Information Theory **2**(1), 441–451 (2021)
18. Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al.: Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971 (2023)
19. Tramer, F., Boneh, D.: Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. arXiv preprint arXiv:1806.03287 (2018)
20. Zama: Concrete ML: a privacy-preserving machine learning library using fully homomorphic encryption for data scientists (2022), https://github.com/zama-ai/concrete-ml