CRISP: An Efficient Cryptographic Framework for ML Inference Against Malicious Clients

Xiaoyu Fang* Shihui Zheng*† Lize Gu*
cugiolk@bupt.edu.cn shihuizh@bupt.edu.cn glzisc@bupt.edu.cn
*Beijing University of Posts and Telecommunications

***Abstract*—Machine learning inference protocols based on semi-honest security models are vulnerable to attacks from malicious clients in real-world applications. These attacks can lead to the leakage of machine learning model parameters. Previous works introduced additional MACs computations to ensure correct client behavior. However, this resulted in higher runtime and communication costs during online inference. In this work, we present CRISP, an efficient two-party cryptographic framework designed to defend against malicious clients. Specifically: 1)We design protocols for non-linear layers based on a new cryptographic primitive (Function Secret Sharing). The core of our approach focuses on optimizing the reconstruction process of MACs. 2)We propose a complex domain verification mechanism for linear layers. This mechanism eliminates the additional MACs computations by making better use of the complex space in homomorphic encryption CKKS. Furthermore, in previous work (SIMC, USENIX Security'22), we identified compatibility issues in practical applications. The MAC reconstruction process in the nonlinear layers may leak intermediate inputs and outputs of the model when certain garbled circuit optimizations are applied. In contrast, CRISP effectively avoids this problem. In secure inference benchmarks considered in SIMC, CRISP reduces the total communication cost of ML inference by up to 94% and cuts inference latency by up to 43%.**

I. Introduction

Online ML (machine learning) inference uses pre-trained models to help users solve downstream tasks such as prediction and classification. It has been widely adopted as a SaaS (Software as a Service) in various fields including medical diagnosis [1], image recognition [2], and financial analysis [3]. However, in scenarios with high data privacy requirements (such as healthcare and government affairs), ML inference must be secure and must not leak sensitive data from any participating party [4]. We refer to ML inference that meets this condition as secure inference.

In secure inference, server P_0 possesses a machine learning model $M(\mathbf{W}, \mathbf{x})$, where $\mathbf{W}$ represents the weights that are private and sensitive to P_0. The input data x is held by client P_1. The objective is to enable P_1 to learn the model output $\mathbf{y} = M(\mathbf{W}, \mathbf{x})$ without revealing the private information $\mathbf{x}$, while gaining no additional information.

Over the past few years, numerous studies have proposed methods based on secure two-party computation cryptographic primitives (i.e. homomorphic encryption [5], Yao's garbled circuits [6], function secret sharing [7], etc.) to achieve this goal. Due to the computational complexity of secure two-party computation, the vast majority of these works have been founded on weaker security assumptions (i.e., both parties are semi-honest [8], [9]), focusing primarily on addressing efficiency issues.

However, Lehmkuhl et al. show in MUSE [10] that in practical deployments, it is reasonable to assume that servers hosting ML models are semi-honest due to reputational constraints. Conversely, client entities come from diverse sources, and malicious attackers among them may violate protocol specifications. MUSE shows that secure inference protocols based on the semi-honest model can be easily broken in practice. An attacker can fully recover the model parameters using far fewer queries than state-of-the-art black-box model extraction attacks. This finding highlights a critical issue: it is essential to enforce correct behavior from the client side during secure inference.

A. Related Work

MUSE represents the first work designed for secure inference in a client-malicious setting. Its mechanism is built upon the Delphi [11], incorporating additional verification to ensure the correctness of client non-linear layer inputs (equivalent to linear layer outputs) and outputs (serving as linear layer inputs). Although MUSE's performance stands out among traditional protocols that counter malicious adversaries, its computational and communication costs remain approximately 15 times higher than Delphi due to the complexity of non-linear components.

To address efficiency concerns, Chandran et al. has proposed more efficient 2-PC protocols—SIMC [12] — specifically targeting malicious clients. SIMC analyzes the primary overhead in MUSE, identifying it within the complex non-linear layer computations: while using garbled circuits (GC) to complete non-linear function (primarily ReLU) calculations, participants must also learn corresponding message authentication codes (MACs) through specific multiplication protocols. This approach requires communication of at least $2c\lambda + 190\kappa\lambda + 232\kappa^2$ under security parameter λ, AND gates

†Corresponding author

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.230011
www.ndss-symposium.org

number c and field space κ. In response, SIMC introduces a novel protocol based on customized GC, utilizing garbled circuit labels as one-time pads to efficiently construct non-linear layer outputs with verification tags. The results show that SIMC achieves at least a fourfold improvement in computational efficiency and reduces communication by more than 28 times.

However, SIMC 2.0 argues that SIMC's performance still has room for improvement before practical application: first, both SIMC and MUSE employ homomorphic encryption (HE)-based designs for linear layers, incurring expensive computational costs in large-scale matrix-vector and convolution multiplications (operations that dominate computation in modern neural networks). Additionally, the GC portion in SIMC's non-linear layer encompasses excessive functionality, requiring at least $2d\lambda + 4\kappa\lambda + 6\kappa^2$ in communication overhead, where d denotes the number of AND gates. Following these reasoning, SIMC 2.0 proposes a new coding method for homomorphic linear computation in a SIMD manner and a block-combined diagonal encoding method, reducing the complexity of rotation operations required for matrix-vector computation. For non-linear layers, SIMC 2.0 computes only the non-linear parts of ReLU into GC rather than encapsulating entire ReLU functions, effectively reducing the number of AND gates. Compared with SIMC, SIMC 2.0 demonstrates a 17.4-fold increase in linear layer computation speed and a 1.3-fold reduction in non-linear layer communication costs across different data dimensions.

The work of SIMC 2.0 illustrates key optimization strategies in secure inference. On one hand, numerous research efforts in linear layers have focused on developing novel encoding methods to reduce the number of rotations in large-batch matrix-vector multiplications and convolution operations (rotations being considered the most expensive operations in multiplication) to improve computational efficiency. On the other hand, since evaluating non-linear layers (e.g., ReLU and MaxPool) using garbled circuits (GC) is several orders of magnitude more expensive than linear layer protocols in terms of communication and computation [13], many advanced works have proposed potentially superior alternatives. Projects like Poseidon [14] and Bumblebee [15] employ specific approximation techniques, processing non-linear functions through piecewise approximation, thereby replacing nonlinear functions with linear functions. SiRNN [16] and Iron [17] construct specialized processing methods for different non-linear functions, such as utilizing tree-reduction protocols to implement ReLU, Softmax, and other functions, thereby avoiding the overhead of garbled circuits. CryptFlow2 [18] designs new millionaire protocols to efficiently implement various non-linear functions, improving computational costs by 8 to 18 times and reducing communication costs by at least 7 times compared to garbled circuits.

The work mentioned above performs excellently in semi-honest adversary models, but it faces challenges when applied to scenarios with malicious clients. For example, in CryptFlow2's millionaire protocol, almost every operation and sub-protocol requires additional verification tag calculations to ensure the correctness of client behavior. This significantly increases the computational cost and communication volume of the entire protocol. Therefore, when dealing with such scenarios, a more effective solution might be to choose methods or cryptographic primitives that can naturally extend to address malicious participants.

We recognize that the Function Secret Sharing (FSS) method proposed by Elette Boyle et al. [7] offers unique advantages in this area. First, as a primitive for secure two-party computation, FSS inherently provides higher efficiency and lower communication costs compared to garbled circuits. For example, in FastSecNet [19], Meng Hao et al. used FSS to evaluate non-linear functions (ReLU, softmax) and achieved 64 times better communication efficiency and 11 times better computational efficiency compared to Delphi's garbled circuit-based methods for non-linear layers. Second, FSS can naturally extend to scenarios involving malicious clients. [20] introduced a lightweight verification mechanism that uses two steps to defend against malicious behavior. We optimize the approach in [20], which achieves the same security goal with one step in verification phase.

B. Our Contributions

In this work, we introduce CRISP, a novel secure inference model designed to defend against malicious clients. CRISP follows the SIMC infrastructure, employing MACs to ensure correct client behavior. However, compared to SIMC, CRISP achieves superior results in both linear and non-linear layers. Specifically:

- **Compatibility Issues Analysis of SIMC:** We investigated the compatibility of SIMC's nonlinear protocol with garbled circuit optimizations in practical settings, identifying potential risks to correctness or security. For the security issue, we propose an attack strategy and experimentally confirmed that it may lead to increased the risk of model leakage.
- **FSS-Based Protocol for Non-Linear Layers:** In the malicious client setting, we developed a secure computation method for non-linear functions based on FSS, with specific optimizations for ReLU. Our method completes the computation of a nonlinear function and two MACs in just one round of communication during the online phase. In terms of communication cost, each instance of a nonlinear layer only requires $2n$ bits, where n is the bit-length of the input. Compared to SIMC, CRISP also delivers computational efficiency improvements while maintaining security guarantees against malicious clients.
- **Linear Layer Computation Optimization via Complex Domain Verification:** Diverging from existing linear layer optimization approaches, we focus on the functionality of cryptographic primitives: utilizing the complex number encoding capabilities of the CKKS homomorphic encryption scheme to simultaneously process numerical operations and verification labels within a single computational operation. By activating the typically under-

utilized complex domain space, our method eliminates the need for additional verification label computations, significantly reducing both computational and communication costs in linear layers. Our approach reduces computational latency by 24%-67% compared to the SIMC scheme.

The remainder of this paper is organized as follows. In Section II, we introduce notations, the threat model, and related technologies. In Section III-A, we analyze security vulnerabilities in SIMC's non-linear layers and propose attack strategies. We provide detailed information about CRISP in Sections III-B, IV and V, followed by experimental evaluations in Section VI. Section VII concludes the paper.

II. Preliminaries

A. Threat Model

In a two-party ML inference scenario, we have a server P_0 and a client P_1. The server P_0 possesses an ML model M with private and sensitive weights $\mathbf{W}$. It is semi-honest, meaning it adheres to the ML inference procedure but may attempt to deduce the client's data by analyzing data flows during execution. The client P_1, on the other hand, holds a private input $\mathbf{s}$. It is considered malicious and may deviate from the protocol arbitrarily. Both parties are assumed to have knowledge of the model architecture NN used in M.

The objective is to develop a secure inference framework that allows P_1 to obtain the inference result of $\mathbf{s}$ without learning any information about $\mathbf{W}$, while ensuring P_0 gains no knowledge of the input $\mathbf{s}$.

B. Notations

λ is the computational security parameter. $\mathbb{Z}$, $\mathbb{R}$, and $\mathbb{C}$ denote the ring of integers, real, and complex numbers, respectively. Let $\mathbb{Z}_N$ be input and output domains of size $N = 2n$, where n is the bit length. We use $\mathcal{R} := \mathbb{Z}[X]/(X^N + 1)$ with N being power-of-two to denote the ring of polynomials modulo $X^N + 1$. Boldface lowercase letters represent vectors, such as $\mathbf{s}$. Boldface uppercase letters represent matrices, such as $\mathbf{W}$. $[n]$ denotes the set $\{1, 2, , n\}$.

For simplicity, we assume that the machine learning model NN has ℓ layers, and each layer consists of alternating linear layers(with weights $\mathbf{W}_1, \mathbf{W}_2, ..., \mathbf{W}_\ell$) and nonlinear layers(with nonlinear function $f_1, f_2, ..., f_{\ell-1}$). Given an input vector $\mathbf{v}_0$, the model sequentially computes $\mathbf{s}_i = \mathbf{W}_i \cdot \mathbf{v}_{i-1}$ and $\mathbf{v}_i = f_i(\mathbf{s}_i)$, where $i \in [\ell - 1]$. Finally, we have $\mathbf{s}_\ell = \mathbf{W}_\ell \cdot \mathbf{v}_{\ell-1} = NN(\mathbf{v}_0)$

C. Fully Homomorphic Encryption

FHE [21] is a public key encryption scheme that supports the evaluation of arbitrary functions expressed as polynomials on encrypted data. In this paper, we focus on the CKKS scheme [22], a widely utilized approach in PPML research that enables approximate arithmetic operations. This scheme features encryption and decryption algorithms (Enc/Dec) operating within the ring $\mathcal{R}$ which serves as the plaintext space. i.e., for $m_1, m_2 \in \mathcal{R}$. Its homomorphic properties satisfy:

$$\texttt{Dec}(\texttt{Enc}(m_1) \boxplus \texttt{Enc}(m_2)) \approx m_1 + m_2$$

$$\texttt{Dec}(\texttt{Enc}(m_1) \boxdot \texttt{Enc}(m_2)) \approx m_1 \cdot m_2$$

$$\texttt{Dec}(\texttt{Enc}(m_1) \boxplus m_2) \approx m_1 + m_2$$

$$\texttt{Dec}(\texttt{Enc}(m_1) \boxdot m_2) \approx m_1 \cdot m_2$$

Here, $\boxplus$ and $\boxdot$ denote ciphertext/plaintext addition and multiplication.

In addition, the CKKS scheme comes with a method (Ecd/Dcd) to encode complex messages into plaintext space $\mathcal{R}$:

$$\texttt{Dcd}(\texttt{Ecd}(\mathbf{z}_1) + \texttt{Ecd}(\mathbf{z}_2)) \approx \mathbf{z}_1 \oplus \mathbf{z}_2$$

$$\texttt{Dcd}(\texttt{Ecd}a(\mathbf{z}_1) \cdot \texttt{Ecd}(\mathbf{z}_2)) \approx \mathbf{z}_1 \odot \mathbf{z}_2$$

where $\oplus$ and $\odot$ represent the component addition and Hadamard product of vectors.

D. Arithmetic Secret Sharing

A 2-of-2 additive secret sharing scheme for $x \in \mathbb{Z}_N$ represents x as a pair $(\langle x \rangle_1, \langle x \rangle_2) = (x - r, r) \in \mathbb{Z}_N^2$, where r is a randomly chosen value from $\mathbb{Z}_N$. The reconstruction of x is achieved by summing the two shares: $x = \langle x \rangle_1 + \langle x \rangle_2 \mod N$. This scheme is perfectly hiding, meaning that knowing either share $\langle x \rangle_1$ or $\langle x \rangle_2$ reveals no information about x.

E. Function Secret Sharing

Function Secret Sharing (FSS) is a cryptographic technique that enables the decomposition of a function f into two additive shares, f_0 and f_1, such that:

- Each share f_b (where $b \in \{0, 1\}$) reveals no information about f.
- For any public input x, the sum of the shares satisfies $f_0(x) + f_1(x) = f(x)$.

Formally, a 2-party FSS scheme comprises the following algorithms:

- **Gen**$(1^\lambda, f)$: A probabilistic polynomial-time (PPT) key generation algorithm. It takes as input the security parameter λ and a function f, and outputs a pair of keys (k_0, k_1), where each key implicitly defines $f_b : \mathbb{Z}_N \to \mathbb{Z}_N$.
- **Eval**(b, k_b, x): A polynomial-time evaluation algorithm. It takes as input a party index $b \in \{0, 1\}$, a key k_b, and a public input $x \in \mathbb{Z}_N$, and computes $y_b = f_b(x)$, such that $f(x) = f_0(x) + f_1(x)$.

F. Other Cryptographic Primitives

In this section, we provide a brief overview of Oblivious Transfer (OT) and Garbled Circuits (GC) to facilitate the understanding of subsequent comparisons between different schemes.

Oblivious Transfer: The 1-out-of-2 Oblivious Transfer [23] is denoted as OT_n, where the inputs of the sender (P_0) are two strings $s_0, s_1 \in \{0, 1\}^n$, and the input of the receiver (P_1) is a choice bit $b \in \{0, 1\}$. At the end of the OT-execution, P_1 obtains s_b while P_0 receives nothing, and P_0 does not know which string has been chosen.

Garbled Circuits: A garbling scheme for boolean circuits [24] consists of a pair of algorithms (**Garble**, **GCEval**) defined as:

- **Garble**$(1^\lambda, C) \rightarrow (GC, \{\{\text{lab}_{i,j}^{in}\}_{i\in[n]}, \{\text{lab}_j^{out}\}\}_{j\in\{0,1\}})$. **Garble** on input the security parameter λ and a boolean circuit $C : \{0,1\}^n \rightarrow \{0,1\}$ outputs a garbled circuit GC, a collection of input labels $\{\text{lab}_{i,j}^{in}\}_{i\in[n],j\in\{0,1\}}$ and a collection of output labels $\{\text{lab}_j^{out}\}_{j\in\{0,1\}}$ where each label is of λ-bits. For any $x \in \{0,1\}^n$, the labels $\{\text{lab}_{i,x[i]}^{in}\}_{i\in[n]}$ are referred to as the garbled input for x and the label $\text{lab}_{C(x)}^{out}$ is referred to as the garbled output for $C(x)$.
- **GCEval**$(GC, \{\text{lab}_i\}_{i\in[n]}) \rightarrow \text{lab}'$. **GCEval** on input a garbled circuit GC and a set of labels $\{\text{lab}_i\}_{i\in[n]}$ outputs a label lab'.

III. Nonlinear Layer Optimization

In this section, we analyze the nonlinear protocol in SIMC and propose a attack strategies from the client's perspective to exploit its compatibility issue. Subsequently, we introduce a novel FSS-based optimization method for securing nonlinear layers against malicious clients. Our focus is on the secure computation of the ReLU activation function, which is one of the most widely used functions in the nonlinear layers of modern deep neural networks (DNNs).

A. Attack Strategy for SIMC

To defend against malicious clients, SIMC introduces additional Message Authentication Codes (MACs) within a general PPML framework to verify the correctness of client inputs at each layer. The nonlinear layer protocol in SIMC is implemented using garbled circuits and consists of three phases: the garbling phase, the authentication phase, and the local computation phase. (Details are in the Appendix A) The core idea behind computing the nonlinear output and its associated authenticated shares is to use the output labels from the garbling phase as encryption and decryption keys. Additionally, the point-and-permute technique is employed to obfuscate the correspondence between ciphertexts and actual values, preventing the client from linking any ciphertext to its true output.

The use of Garbled Circuits (GC) requires various optimizations to reduce the complex computation processes and high communication overheads, like [25]–[27], and SIMC follows a similar approach. However, while the extension of the point-and-permute technique in the SIMC protocol improves performance in authentication phase, its compatibility with other optimization methods may be contentious [1].

For example, in [25], the GPGRR2 method is proposed, which eliminates the need for external indices and reduces the size of ciphertext evaluations. [26] introduces an information-theoretic secure secret-sharing obfuscation scheme that assigns permutation bits only to the "A line" to indicate which part of the "B line" should be used, while no permutation or color bits are assigned to the "B line." This approach bypasses the lower bounds of garbled circuits. Additionally, [28], [29] construct reusable garbled circuits, allowing the evaluator to assess multiple inputs within a single circuit using the same labels.

When SIMC is combined with the aforementioned garbled circuit optimizations, issues related to correctness or security may arise. First, incompatibility with the point-and-permute technique can hinder P_1 from selecting the correct ciphertext during the authentication phase. For example, the method in [25], [26] removes the selection bits required for encrypting the randomness used in SIMC's authentication phase.

Second, in the case of reusable garbled circuits, forcibly adding permutation bits can reveal the correlation between permutation bits and actual values across multiple inputs, potentially exposing both the inputs and outputs of the nonlinear layer in SIMC [2]. This poses a serious security risk, which we illustrate through a proposed attack strategy.

Our attack strategy relies on two key observations regarding privacy-preserving machine learning:

1) In a neural network that alternates linear and nonlinear layers, the input to the linear layer of the $(i+1)$-th layer depends on the output of the nonlinear layer of the i-th layer.
2) In SIMC and most PPML frameworks, the client and server collaborate to compute the output of each layer in the neural network. As a result, the client gains knowledge of the model architecture. For example, in SIMC, the client can easily distinguish between linear layer computations and garbled circuit computations, allowing it to clearly identify which layer of the neural network is currently being processed.

Therefore, the client gains access to the inputs of each nonlinear layer from the second layer to the ℓ-th layer in the neural network. Although the client's input is verified in the SIMC setting—rendering the attack in MUSE infeasible—the exposure of the model structure allows the client to understand the composition of each layer in the neural network. Combined with knowledge of the nonlinear layer inputs and outputs, this makes a model extraction attack feasible.

Previously, most model extraction attacks [30]–[34] were conducted in black-box environments, which fall into two main categories. The first type involves attackers who know the model structure and understand the specific types of activation functions. The second type involves attackers who have no knowledge about the internal structure of the model and only know the input x and the final output $y = NN(x)$. This differs from our work in a significant way - the vulnerabilities in the SIMC protocol expose the nonlinear layer inputs and outputs, making the attack environment no longer a black box.

Taking fully connected operations as an example (convolutional operations can also be viewed as combinations of multiplication and addition), as our earlier observations

[1]Our analysis highlights potential compatibility issues when specific optimizations are introduced

[2]In the garbled circuit phase of the nonlinear protocol in SIMC, the output corresponding to input s includes both s and $f(s)$.

indicated, clients can learn the model structure through linear and non-linear protocols. In this case, attackers can obtain multiple sets of inputs and outputs for any linear layer (by repeatedly running the protocol), which transforms the model extraction attack into essentially a linear regression problem (as shown in Equation (1)).

$$\mathbf{W}\cdot\begin{bmatrix}\mathbf{x}_1^T\\ \mathbf{x}_2^T\\ \vdots\\ \mathbf{x}_n^T\end{bmatrix}=\begin{bmatrix}\mathbf{y}_1^T\\ \mathbf{y}_2^T\\ \vdots\\ \mathbf{y}_n^T\end{bmatrix}-\begin{bmatrix}b^T\\ b^T\\ \vdots\\ b^T\end{bmatrix} \tag{1}$$

If the data meets the necessary conditions, meaning $\mathbf{x}^T\mathbf{x}$ is invertible, we can directly calculate the weight matrix through the equation:

$$\hat{\mathbf{W}}=(\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y}$$

After we calculate the value of $\mathbf{W}$, we can obtain the bias term b by computing the average:

$$\hat{b}=\frac{1}{n}\sum_{i=1}^{n}(\mathbf{y}_i-\hat{\mathbf{W}}\mathbf{x}_i)$$

If $\mathbf{x}^T\mathbf{x}$ is not invertible, the attacker can use regularization methods such as Ridge Regression. This approach adds a regularization term to make the matrix invertible:

$$\hat{W}=(\mathbf{x}^T\mathbf{x}+\lambda I)^{-1}\mathbf{x}^T\mathbf{y}$$

By using this method, the attacker can obtain an approximation of the model's weights and bias. We conducted experiments on the fully connected layer of the 7-layer CNN model from [35]. As the number of input samples increased, the mean squared error (MSE) between the recovered weights\biases and its true values remained below 0.0001.

Additionally, for simple single-layer structures, the attacker can use the method provided in [36] to achieve effective functional approximation.

In summary, the nonlinear layer protocol in SIMC may lead to model information leakage in practical applications. This risk arises from the extended use of the point-and-permute technique, while the security proof of SIMC only addresses the basic protocol and does not account for real-world deployment scenarios. We argue that the garbled circuit approach itself may be the bottleneck in the performanc. Therefore, in the next section, we explore an alternative solution: leveraging Function Secret Sharing (FSS) to achieve both malicious-client resistance and efficient inference.

B. Our Protocol for the Non-Linear Layer

In this section, we build upon the work of [19] to present an efficient FSS-based implementation of ReLU. Additionally, we propose an optimized implementation tailored for settings with malicious clients. Compared to the protocol used in SIMC, this method offers enhanced efficiency while preventing the exposure of intermediate parameters in nonlinear layers, thereby mitigating the risk of model extraction attacks.

The ReLU function, one of the most widely used activation functions in machine learning, can be fundamentally understood as a comparison between the input and zero. State-of-the-art works [11], [18], [37] implement this functionality using garbled circuits (GC) and oblivious transfer (OT) techniques. However, these approaches incur high communication costs and require multiple rounds of interaction. Specifically, the OT-based solution [18] requires $(\kappa n + 18n)$ bits of communication over $\log(n+2)$ rounds, while GC-based solutions [11], [37] consume κn bits over two communication rounds. In contrast, our FSS-based solution [19] achieves efficient ReLU computation with only $2n$ bits of communication in a single round.

We begin by revisiting the FSS-based comparison protocol [20], which serves as a fundamental operation for nonlinear functions. The general comparison function is denoted as $f^{<}_{\alpha,\beta}(x)$, which outputs β if $x<\alpha$, and 0 otherwise. The FSS scheme for comparison consists of a pair of algorithms: $\texttt{Gen}^{<}_{\alpha,\beta}$ and $\texttt{Eval}^{<}_{\alpha,\beta}$. After obtaining their respective keys through $\texttt{Gen}^{<}_{\alpha,\beta}$, parties evaluate the common input using $\texttt{Eval}^{<}_{\alpha,\beta}$ to obtain the final output, which serves as their share of the function $f^{<}_{\alpha,\beta}(x)$ computation result.

FSS employs a fixed-point ring to represent the plaintext space, where $\mathrm{ReLU}(x)$ equals x if $x<N/2$, and 0 otherwise. To accommodate secret-shared inputs, ReLU requires an offset r (sampled from $\mathbb{Z}_N$ uniformly at random) to ensure privacy of x. [3] We define the offset function for ReLU as $\mathrm{ReLU}_r=\mathrm{ReLU}(x-r)$. We summarize the ReLU construction method in [19], to efficiently compute ReLU_r using FSS, we approximate its computation over $\mathbb{Z}_N$ as:

$$\mathrm{ReLU}_r(x)=\begin{cases}x-r, & x>r\\ 0, & else\end{cases} \tag{2}$$

For sufficiently large domains, Equation (2) holds with high probability.

The complete FSS-based ReLU protocol is presented in Algorithms 1 and 2. The protocol incorporates the splines function from [7] to output polynomial coefficients $\boldsymbol{\beta}$: When input $x>r$, $\boldsymbol{\beta}=(\beta_0,\beta_1)$; otherwise $\boldsymbol{\beta}=(0,0)$.

The aforementioned FSS protocol is designed for semi-honest parties, and adapting it to a malicious client scenario presents two main challenges. The first lies in ensuring the correct execution of correlated randomness setup (e.g. r and $\boldsymbol{\beta}$). Fortunately, trusted correlated randomness setup can be achieved through several approaches: assuming a semi-trusted dealer (e.g., [38]), introducing a trusted third party, or employing two-party maliciously secure preprocessing protocols. For two-party maliciously secure preprocessing protocols, this can be efficiently accomplished by incorporating state-of-the-art general-purpose 2PC protocols (e.g., [39]–[41]).

The second critical aspect is how to ensure the correct behavior of the client. To solve this, We first extend the original protocol by incorporating a verification mechanism for

[3]This is the prerequisite for ensuring that FSS works in secure two-party computation. The core concept can be found in [20].

Algorithm 1 $\texttt{Gen}^{\text{ReLU}}_{\alpha,\beta}$ for ReLU Function

```
1: Let $\boldsymbol{\beta} = (\beta_0, \beta_1) = (1, -r)$ and $\alpha = r$
2: $(k'_0, k'_1) \leftarrow \text{Gen}^{<}_{\alpha,-\boldsymbol{\beta}}$
3: Sample $\langle r\rangle_0, \langle r\rangle_1 \leftarrow \mathbb{Z}_N$, s.t., $\langle r\rangle_0 + \langle r\rangle_1 = r \mod N$
4: Sample $\langle\beta_0\rangle_0, \langle\beta_0\rangle_1, \langle\beta_1\rangle_0, \langle\beta_1\rangle_1 \leftarrow \mathbb{Z}_{2^n}$, s.t., $\langle\beta_0\rangle_0 + \langle\beta_0\rangle_1 = \beta_0 \mod N$ and $\langle\beta_1\rangle_0 + \langle\beta_1\rangle_1 = \beta_1 \mod 2^n$
5: Let $k_b = k'_b \parallel \langle r\rangle_b \parallel \langle\boldsymbol{\beta}\rangle_b$ for $b \in \{0,1\}$
6: **return** $(k_0, k_1)$
```

malicious security [20]. Specifically, for nonlinear function f, $f_\mu = \mu f$ is additionally constructed in each calculation, where μ is the verification key known only to the server. Verification is achieved by confirming whether μf equals f_μ, thereby ensuring the client has executed the correct operations during the current computational round. In addition, we optimized the mechanism of extending the verification gate. Using the Algorithm 1 as an example, by modifying $\boldsymbol{\beta}$ to $\boldsymbol{\beta}' = (\mu, -\mu r)$, we can control the evaluation function's output to be either 0 or μx, thereby quickly implementing the verification gate of ReLU. Since the input to f_μ is identical to the input for f, the evaluation of f_μ requires no additional communication costs.

Algorithm 2 $\texttt{Eval}^{\text{ReLU}}_{\alpha,\beta}$ for ReLU Function

```
1: Parse $k_p = k'_b \parallel \langle r\rangle_b \parallel \langle\boldsymbol{\beta}\rangle_b$
2: Send $\langle x\rangle_b + \langle r\rangle_b$ to the other party, receive $\langle x\rangle_{1-b} + \langle r\rangle_{1-b}$, and reconstruct $x + r \mod N$
3: Set $(\langle y_0\rangle_b, \langle y_1\rangle_b) \leftarrow \texttt{Eval}^{<}_{\alpha,-\boldsymbol{\beta}}(b, k'_b, x + r) + \boldsymbol{\beta}_b$
4: Compute $\langle y\rangle_b = \langle y_0\rangle_b(x + r) + \langle y_1\rangle_b \mod N$
5: **return** $\langle y\rangle_b$
```

The correct execution of FSS ensures the security of our nonlinear layer protocol. Building upon this, we focus on efficiently constructing MAC values under malicious parties. Our protocol consists of two main phases: function secret sharing phase and local computation phase. Below, we provide a high-level overview of our protocol. Recall that P_0 inputs $(\langle x\rangle_0, \mu)$ and P1 inputs $\langle x\rangle_1$.

- **Function Secret Sharing Phase:** The objective of this phase is to enable P_0 and P_1 to independently compute the output of the FSS gate and obtain their respective shares. During the offline phase, P_0 and P_1 can acquire FSS keys k_b, k'_b and $\langle r\rangle_b$, $\langle\mu\rangle_b$, $\langle\mu r\rangle_b$ (The specific functionality of the offline phase will be elaborated in Section V). And then, they subsequently invoke $\texttt{Eval}^{\text{ReLU}}_{\alpha,\beta}$ in this phase to obtain $\langle f(\mathbf{x})\rangle_b$ and $\langle f_\mu(\mathbf{x})\rangle_b$.
- **Local Computation Phase:** At this stage, both parties only need to use the $\langle\mu\rangle_b$, $\langle\mu r\rangle_b$ obtained in the offline phase and the $x+r$ obtained in the function secret sharing phase to reconstruct their shares of μx.x

We formally describe the Non-linear layer protocol $\pi^f_{\text{Non-lin}}$ in Figure 2. By relegating the majority of operations to the offline phase, our protocol can complete non-linear function computation and MACs reconstruction with just one round of interaction. Compared to SIMC, CRISP has a much lower online communication cost in the non-linear layer. Its communication is only $2n$, which grows linearly with the input size n. In contrast, SIMC's communication depends on the security parameter λ, the statistical security parameter κ, and the number of AND gates d. These factors lead to linear or even quadratic growth in total cost. Under common parameter settings, such as $\kappa = 44$, $\lambda = 128$, and $n = 32$, our protocol reduces the communication cost by more than two orders of magnitude compared to SIMC.

Remark 3.1 (Multi-Input Nonlinear Functions). Our technique can be readily extended to non-elementary functions that take multiple inputs and produce one or more outputs, such as Maxpool. Specifically, The underlying algorithm of Maxpool is to compute the maximum value over d elements $x_0, x_1, ..., x_{d-1}$. we can employ a tree reduction architecture that recursively divides the input in half and compares elements from each half. For each comparison between two secret-shared elements $\langle x_i\rangle$ and $\langle x_j\rangle$, we can simply extend using ReLU: $\max(\langle x_i\rangle, \langle x_j\rangle) = \text{ReLU}(\langle x_i\rangle - \langle x_j\rangle) + \langle x_j\rangle$. Building maxpool$_\mu$ follows the same principle. We provide a performance comparison experiment of Maxpool in Appendix B.

Theorem 1: the protocol $\pi^f_{\text{Non-lin}}$ securely realizes the functionality against a malicious client (P_1) and a semi-honest server (P_0).

Proof: We first prove correctness of the protocol followed by security.

Correctness. Based on the correctness of ($\texttt{Gen}^{<}_{\alpha,\beta}$, $\texttt{Eval}^{<}_{\alpha,\beta}$), each participating party can obtain the correct key and compute their corresponding share of the function output. This enables the parties to acquire the correct values of $f(\mathbf{x})$ and $f_\mu(\mathbf{x})$. Furthermore, both parties' shares of μx sum up to the correct value, such that:

$$\begin{aligned}
&\langle\mu x\rangle_0 + \langle\mu x\rangle_1 \\
&= \langle\mu\rangle_0(x + r) - \langle\mu r\rangle_0 + \langle\mu\rangle_1(x + r) - \langle\mu r\rangle_1 \\
&= \mu(x + r) - \mu r \\
&= \mu x
\end{aligned} \tag{3}$$

Security: Now, we prove security against any malicious adversary $\mathcal{A}$ controlling P_1

Claim 1: $\pi^f_{\text{Non-lin}}$ is secure against any malicious adversary $\mathcal{A}$ corrupting the client P_1.

Proof: We prove *Claim 1* through a sequence of hybrid experiments. We demonstrate that P_1's view in the real protocol execution is computationally indistinguishable from its view in an ideal world with a simulator.

Hybrid 0 (Real World) executes the protocol as specified in Figure 2.

Hybrid 1 replaces the FSS key generation with ideal functionality while maintaining and replaces P_0's $\langle\mu\rangle_b$, $\langle\mu r\rangle_b$. The value $\langle f(\mathbf{x})\rangle_0$, $\langle f_\mu(\mathbf{x})\rangle_0$ and $\langle\mu\mathbf{x}\rangle_0$ is now computed using these simulated randomness[4].

[4] $f_\mu(\mathbf{x})$ and $\mu\mathbf{x}$ will be uniformly verified in the final stage of secure inference

Preamble: The function f is such that $f : \mathbb{Z}_N \to \mathbb{Z}_N$. Consider f based on FSS Nonlin^f that takes additive shares of $x \in \mathbb{Z}_N$, i.e., $\langle a\rangle_0, \langle a\rangle_1 \in \mathbb{Z}_N$, as input and outputs $f(a)$.
Input: P_0 inputs $\langle \mathbf{x}\rangle_0 \in \mathbb{Z}_N^d$ and $\mu \in \mathbb{Z}_N$. P_1 inputs $\langle \mathbf{x}\rangle_1 \in \mathbb{Z}_N^d$ and $\langle \mu\rangle_1$.
Output: P_b learns $\langle \mu\mathbf{x}\rangle_b$, $\langle f(\mathbf{x})\rangle_b$, $\langle f_\mu(\mathbf{x})\rangle_b$ for $b \in \{0, 1\}$.
Protocol:

1. **Function Secret Sharing Phase:**
   - P_b compute $\langle f(\mathbf{x})\rangle_p \leftarrow \texttt{Eval}_{\alpha,\beta}$ by using k_p and $\langle \mathbf{x}\rangle_b$
   - P_b computes $\langle f_\mu(\mathbf{x})\rangle_p \leftarrow \texttt{Eval}^{\text{mac}}_{\alpha,\beta}$ by using k'_p and $\langle \mathbf{x}\rangle_b$
2. **Local Computation Phase:**
   - P_b computes $\langle \mu x\rangle_b = \langle \mu\rangle_b(x + r) - \langle \mu r\rangle_b$

Fig. 1: Protocol $\pi^f_{\text{Non-lin}}$

Hybrid 2 (Remove the Influence of Input) builds upon Hybrid 1 by further replacing replaces $\langle x+r\rangle_0$ with simulator-generated random value.

Hybrid 3 (Ideal World) replaces $\langle \mu\mathbf{x}\rangle_0$ and the real FSS evaluation with a simulator-generated random value and an ideal functionality, respectively. That is, all shares are produced by the simulator rather than computed through the protocol

The indistinguishability between adjacent hybrids follows from standard security arguments. The transition from Hybrid 0 to Hybrid 1 relies on the security of FSS and random value: Adversary $\mathcal{A}$ cannot distinguish whether the value evaluated by the key and the message associated with the random value is real or comes from the simulator. Hybrid 2 eliminates the influence of input $\mathbf{x}$. Due to the uniform randomness of the share of $\mathbf{x}$ and r, it remains indistinguishable from Hybrid 1. Finally, Hybrid 2 to Hybrid 3 preserves indistinguishability through the randomness of the same distribution, since $\langle \mu\mathbf{x}\rangle_0$ and the evaluation results of FSS are uniformly random in Hybrid 2.

This sequence of hybrids, justified by security of FSS and the randomness of the same distribution, establishes that Protocol $\pi^f_{\text{Non-lin}}$ achieves computational security against malicious P_1.

Security of $\pi^f_{\text{Non-lin}}$ against any semi-honest adversary $\mathcal{A}$ controlling the server P_0 follows the similar proof process as above.

IV. Linear Layer Optimization

We first describe the secure execution of linear layers in previous work. SIMC and SIMC2.0 designed two protocols (**InitLin** and **Lin**) to securely execute linear layer operations, maintaining additional MAC values in each layer to ensure the correctness of client inputs. This setup approximately doubles the computational overhead of linear layers compared to linear layer protocols based on semi-honest participants: each linear layer requires homomorphic computation of both $\mathbf{W} \cdot \mathbf{m}$ and $\mu\mathbf{W} \cdot \mathbf{m}$ (including matrix-vector multiplication in fully connected layers and convolution operations in convolutional layers), where $\mathbf{W}$ is the plaintext weight matrix held by the server and $\mathbf{m}$ is the client's input. However, in real-world scenarios involving large-scale computations, this increased computational cost creates a gap between theoretical performance and practical usability. To address this issue, SIMC 2.0 optimized GAZELLE's matrix-vector multiplication algorithm and applied it to linear computations, thereby reducing the computational cost of linear layers.

Our approach differs from SIMC2.0. We harness CKKS's ability to encrypt complex numbers to reduce multiplication computations. CKKS encoding allows data to be mapped from the complex domain to the integer polynomial domain ($\mathbb{C}^{N/2} \to \mathbb{Z}[X]/(X^N + 1)$) before encryption.

In real applications, operations usually happen in the real or integer domain, and the complex domain space remains underused. We activate this complex domain by extending the input to $\mathbb{C}^{N/2}$ and placing MAC value in the complex domain. This approach allows us to compute both values and verification tags at the same time in a single operation. It greatly reduces the number of multiplication operations needed for secure inference when dealing with malicious clients.

As shown in Figure 2, our linear layer protocol consists of **InitLIN** and **LIN**. **InitLin** is invoked exactly once and takes as input a matrix $\mathbf{W}$ and a MAC key μ from P_0 and $\mathbf{x}$ from P_1. It outputs authenticated shares of $\mathbf{m} = \mathbf{W}\mathbf{x}$ to both parties. Our approach is characterized by producing output shares whose plaintext reside in the complex domain.

For **LIN**, the computation process follows the original structure of SIMC, requiring only minor modifications to the parties's input and computation:

- We embed the secret shared portions used for verification into the complex domain. We then transform them to the integer domain through the CKKS encoding method.
- In operations involving addition (where the server possesses addition constants b, such as bias terms in fully connected layers), the constant b must be transformed into its complex form $b + bj$ and subsequently encoded prior to executing addition operations. For comprehensive details of the addition operations, please refer to the Appendix C.

Notably, the latter operation can be executed during the of-

Realization of InitLin in $\pi_{\mathbf{Lin}}$:

Input: P_0 holds $\mathbf{W} \in \mathbb{Z}_N^{n' \times d}$, $\mu \in \mathbb{Z}_N$ and parameter **InitLin**. P_1 holds $\mathbf{x} \in \mathbb{Z}_N^d$.

Output: P_b learns $\langle \mathbf{Wx} \rangle_b$, $\langle \mu \mathbf{Wx} \rangle_b$ for $b \in \{0, 1\}$.

Protocol:

- P_0 and P_1 (one time) engage in a two-party computation protocol secure against a semi-honest P_0 and malicious P_1 to sample $(\mathrm{pk}, \mathrm{sk})$ for AHE such that P_1 learns $(\mathrm{pk}, \mathrm{sk})$ and P_0 learns pk. Both parties store these for use in this and all subsequent calls to $\boldsymbol{\pi}_{\mathrm{Lin}}$ as well.
- P_b obtain share of $\langle \mu \mathbf{x} \rangle_b$ by methods such as multiplication triplets or garbled circuits, where $b \in \{0, 1\}$.
- P_1 sends the encryption $\mathbf{c}_1 \leftarrow \mathtt{Enc}_{\mathrm{pk}}(\mathbf{x} + \langle \mu \mathbf{x} \rangle_1 \mathrm{j})$ to P_0 along with a zero-knowledge (ZK) proof of plaintext knowledge of this ciphertext[a].
- P_0 set $\mathbf{x}' = \mathbf{c}_1 + \mathtt{Ecd}(\langle \mu \mathbf{x} \rangle_0 \mathrm{j})$
- P_0 samples $\langle \mathbf{m} \rangle_0 \in \mathbb{Z}_N^m$, and sets $\mathbf{m} = \mathbf{Wx}'$.
- P_0 sends the ciphertexts $\mathbf{c}_2 \leftarrow \mathtt{Enc}_{\mathrm{pk}}(\mathbf{m} - \langle \mathbf{m} \rangle_0)$ to P_1.
- P_1 sets $\langle \mathbf{m} \rangle_1 = \mathtt{Dec}_{\mathrm{sk}}(\mathbf{c}_2)$.
- P_b outputs $\langle \mathbf{m} \rangle_b$ for $b \in \{0, 1\}$.

Realization of Lin in $\pi_{\mathbf{Lin}}$:

Input: P_0 holds $\langle \mathbf{m} \rangle_0 \in \mathbb{C}_N^n$, $\mathbf{W} \in \mathbb{Z}_N^{n' \times n}$, $\mu \in \mathbb{Z}_N$ and parameter **Lin**. P_1 holds $\langle \mathbf{m} \rangle_1 \in \mathbb{C}_N^n$.

Output: P_b learns $\langle \mathbf{Wm} \rangle_b$ and $\langle \mathbf{t} \rangle_b$ for $b \in \{0, 1\}$.

Protocol:

- P_1 sends the encryptions $\mathbf{c}_1 \leftarrow \mathtt{Enc}_{\mathrm{pk}}(\mathrm{Re}(\langle \mathbf{m} \rangle_1))$ $\mathbf{c}_2 \leftarrow \mathtt{Enc}_{\mathrm{pk}}(\mathrm{Im}(\langle \mathbf{m} \rangle_1))$ and $\mathbf{c}_3 \leftarrow \mathtt{Enc}(\mathrm{pk}, \langle \mathbf{m} \rangle_1)$ to P_0 and a ZK proof of plaintext knowledge for this three.
- P_0 sets $\mathbf{t} \leftarrow \mu^3(\mathbf{c}_1 + \mathtt{Re}(\mathtt{Dcd}(\langle \mathbf{m} \rangle_0))) - \mu^2(\mathbf{C}_2 + \mathtt{Im}(\mathtt{Dcd}(\langle \mathbf{m} \rangle_0)))$
- P_0 samples $\langle \mathbf{Wm} \rangle_0 \in_R \mathbb{Z}_N^m$ and $\langle \mathbf{t} \rangle_0 \in_R \mathbb{Z}_N^n$.
- P_0 homomorphically evaluates the ciphertexts $\mathbf{c}_4 \in \mathtt{Enc}_{\mathrm{pk}}(\mathbf{W}(\mathbf{C}_3 + \langle \mathbf{m} \rangle_0) - \langle \mathbf{Wm} \rangle_0)$, $\mathbf{c}_5 \in \mathtt{Enc}_{\mathrm{pk}}(\mathbf{t} - \langle \mathbf{t} \rangle_0)$, and sends to P_1.
- P_1 sets $\langle \mathbf{Wm} \rangle_1 = \mathtt{Dec}_{\mathrm{sk}}(\mathbf{c}_4)$ and $\langle \mathbf{t} \rangle_1 = \mathtt{Dec}_{\mathrm{sk}}(\mathbf{c}_5)$.
- P_b outputs $\langle \mathbf{Wm} \rangle_b$ and $\langle \mathbf{t} \rangle_b$ for $b \in \{0, 1\}$.

---

[a] ZK proof of knowledge for the statement that c is a valid sample from $\mathtt{Enc}_{pk}(m)$ for an m known to the prover. We refer the reader to [10], [42] for more details.

Fig. 2: Protocol $\boldsymbol{\pi}_{\mathrm{Lin}}$

fline phase, thus incurring no additional computational burden on the server during the online phase.

Theorem 2: the protocol $\boldsymbol{\pi}_{\mathrm{lin}}$ securely realizes the functionality against a malicious client (P_1) and a semi-honest server (P_0).

Proof: **Correctness.** From Figure 2's description, we can clearly observe the correctness of the protocol. The secret sharing scheme ensures the accuracy of the output pairs. For **InitLIN**, $\langle \mathbf{M} \rangle_0 + \langle \mathbf{M} \rangle_1 = \mathbf{Wx}$. For **LIN**, $\langle \mathbf{WM} \rangle + 0 + \langle \mathbf{WM} \rangle_1 = \mathbf{WM}$.

Security. Our protocol's security stems from two key properties in the semi-honest model. First, for $\mu\mathbf{x}$, cryptographic primitives ensure share randomness, preventing P_0 from inferring $\mathbf{x}$ from $\langle \mu \mathbf{x} \rangle_b$ while P_1 obtains no information about μ. Second, all communication viewed by semi-honest P_0 consists entirely of ciphertexts protected by semantic security. Even under malicious adversarial control of P_1, while decryption of P_0's sent data becomes possible, the uniform randomness of the secret sharing ensures that decrypted values reveal nothing about P_0's inputs due to information-theoretic security properties. The indistinguishability between real and simulated views for both parties follows from the perfect secrecy of the secret sharing scheme and the semantic security of the underlying encryption system, thus completing our security proof under both semi-honest assumptions and with malicious extension considerations.

In homomorphic encryption applications, ciphertext space consumption is a critical factor affecting system throughput and deployment feasibility. The BFV method used in SIMC relies on integer modulus encoding, where MAC value μx calculations must be performed independently from x calculations (such as multiplication operations). This requires each linear layer to process two sets of ciphertexts. This operation causes each logical data element to redundantly occupy double the ciphertext space, and after homomorphic multiplication, each ciphertext expands into a triple ciphertext, further increasing storage overhead. Our proposed CKKS-based method leverages the structural properties of the complex domain to achieve compact representation while maintaining data consistency. CKKS ciphertexts are similarly composed of two or three polynomial rings, with the same basic units as BFV. However, because no additional ciphertext space is needed to construct MAC values, the overall number of ciphertexts is reduced, significantly lowering space redundancy.

For example, with $N = 2^{14} = 16384$ and a modulus length of 218 bits, each ciphertext polynomial occupies approximately 445 KB of storage space. When comparing based on processing $\frac{N}{2} = 8192$ slots per operation, in the CKKS scheme, an entire complex vector requires only 1 ciphertext (expanding to a triple ciphertext after multiplication, occupying about 1.33 MB). In contrast, BFV needs to encrypt two sets of integer data as two ciphertexts (expanding to two triple ciphertexts after multiplication, totaling about 2.67 MB). Therefore, with the same throughput and batch processing capacity, the CKKS scheme reduces ciphertext quantity and total storage consumption by approximately 50% compared to BFV. This demonstrates better space compression and resource utilization, making it particularly suitable for approximate homomorphic application scenarios requiring large-scale dense computations.

Furthermore, our method can naturally support encoding techniques similar to CHeetah [43] (our approach requires optimization over a ring) to improve the efficiency of multiplication operations .

V. Secure Inference

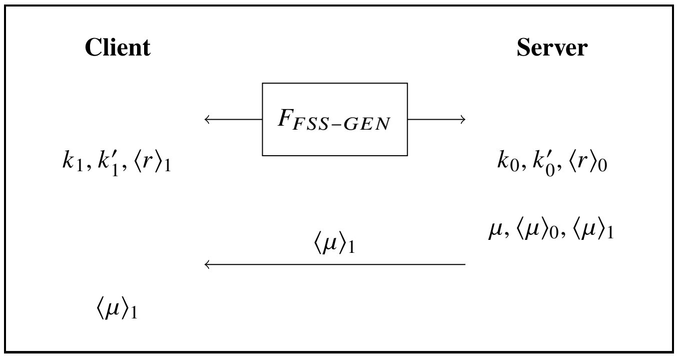
Neural network inference algorithms are generally structured with two distinct layer types: linear and non-linear layers. The linear layers comprise operations such as matrix-vector multiplication and convolution, whereas non-linear layers incorporate functions such as ReLU, Maxpool. Section V-A presents our comprehensive protocol for secure inference under malicious client assumptions. By combining the protocols detailed in Sections III-B and IV with a consistency verification phase, this protocol facilitates secure inference for neural networks incorporating any combination of linear and non-linear layers. We present the security and correctness proofs in Section V-B.

A. Neural Network Inference Protocol

In this section, we describe the details of our secure inference protocol (denoted as π_{Inf}). Similar to [10], [12], we consider a neural network NN with ℓ linear layers (with weights $\mathbf{W}_1, \mathbf{W}_2, ..., \mathbf{W}_\ell$) and $\ell - 1$ non-linear layers evaluating non-linear functions (with nonlinear function $f_1, f_2, ..., f_{\ell-1}$). The network exhibits an alternating structure of linear and non-linear layers, with the first layer being linear. Let $\mathbf{x}$ denote the input to the neural network, and $NN(\mathbf{x})$ represent the inference output. Let $\mathbf{s}_i$ denote the (intermediate) inference vector after evaluating the i-th linear layer, and $\mathbf{v}_i$ denote the (intermediate) inference vector after evaluating the i-th non-linear layer. Given that $\mathbf{s}_1 = \mathbf{W}_1\mathbf{x}$, for $i \in [\ell - 1]$, we have $\mathbf{v}_i = f_i(\mathbf{s}_i)$ and $\mathbf{s}_{i+1} = \mathbf{M}_{i+1}\mathbf{v}_i$. Finally, we have $\mathbf{s}_\ell = \mathbf{W}_\ell \cdot \mathbf{v}_{\ell-1} = NN(\mathbf{x})$.

Within the secure inference framework, the server (P_0) possesses the weights of all linear layers ($\mathbf{W}_1, \mathbf{W}_2, ..., \mathbf{W}_\ell$), while the client ($P_1$) holds the input $\mathbf{x}$. The protocol aims to enable the client to learn $NN(\mathbf{x})$. Figure 4 presents our formal description of protocol π_{Inf} for this setting, which ensures security in the presence of semi-honest servers and malicious clients.

We first describe the offline phase of the protocol. The offline phase primarily consists of two steps. The first relies on function F_{FSS} to generate the correlated randomness for function secret sharing, completing the key distribution. F_{FSS} includes two key generation functions: $\texttt{Gen}_{\alpha,\beta}$ and $\texttt{Gen}^{\text{mac}}_{\alpha,\beta}$, which are used to generate keys for evaluating the nonlinear function f and verifying function f_μ, respectively. In the second step, the server sends secret shared portions of μ to the client. These portions are used during the local computation phase of the nonlinear protocol. We provide specific details of the offline phase in Figure 3.



Client
Server
$F_{FSS-GEN}$
$k_1, k_1', \langle r\rangle_1$
$k_0, k_0', \langle r\rangle_0$
$\mu, \langle\mu\rangle_0, \langle\mu\rangle_1$
$\langle\mu\rangle_1$
$\langle\mu\rangle_1$

Fig. 3: The offline phase in CRISP.

Below, we provide an overview of secure inference. Our protocol can be broadly divided into three phases: linear layer evaluation, non-linear layer evaluation and consistency check phase. In the evaluation phase, we alternately execute computations for linear and non-linear layers while generating corresponding authentication tags. Subsequently, the server performs a consistency check phase to verify the correctness of all client inputs. Upon successful verification, the client obtains the final output.

- **Linear Layer Evaluation:** For the initial linear layer, parties P_0 and P_1 execute protocol π_{Lin} using the parameter **InitLin** to construct inputs with embedded share of MAC value and compute the corresponding linear layer outputs. For each subsequent linear layer i, the parties execute protocol π_{Lin} using the parameter **Lin**. Through this process, both parties obtain $(\langle\mathbf{s_i}\rangle_b, \langle\mathbf{t_i}\rangle_b)$ for b $\in \{0, 1\}$, where $\mathbf{s}$ has a plaintext space $\mathbb{C}$ with MAC value embedded in the complex domain, and $\mathbf{t}$ serves as the consistency tag between the linear layer's inputs and outputs.
- **Non-linear Layer Evaluation:** P_0 and P_1 can obtain the real part of $\mathbf{s}$ by invoking CKKS's decoding function DeCod on the input. Subsequently, both parties execute protocol $\pi^f_{\text{Non-lin}}$ to learn the outputs $(\langle\mathbf{u}\rangle_b, \langle\mathbf{v}^{Re}\rangle_b, \langle\mathbf{v}^{Im}\rangle_b)$ of the i-th non-linear layer. Here, $\mathbf{u}$ represents the reconstructed value $\mu\mathbf{s}^{Re}$, $\mathbf{v}^{Re}$ is the output $f(x)$ of the non-linear function, and $\mathbf{v}^{Im}$ is the MAC value for $\mathbf{v}^{Re}$ (specifically $\mu f(\mathbf{x})$). Furthermore, to accommodate linear layer inputs, the participating parties employ the Cod method to re-encode the constructed complex input $\mathbf{v}$.
- **Consistency Check:** During this phase, the server conducts consistency verification on all client inputs through two primary mechanisms:
  - For i in $\{2, ..., \ell - 1\}$, the correctness of linear layer inputs is validated by verifying that the MAC value t_i equals 0.
  - For i in $\{1, ..., \ell-1\}$, the correctness of non-linear layer inputs is confirmed by verifying that the difference between $\mathbf{s}_i$ and the reconstructed $\mathbf{u}_i$ equals 0 ($\mathbf{s}_i - \mathbf{u}_i =$

Preamble: A neural network NN with ℓ linear and $\ell-1$ non-linear layers. Let $f_1,\ldots,f_{\ell-1}$ be the elementary functions that need to be computed in $\ell-1$ non-linear layers.
Input: P_0 holds $\{\mathbf{W}_j \in \mathbb{Z}_N^{n_j \times n_{j-1}}\}_{j\in[\ell]}$, i.e., weights for the ℓ linear layers. P_1 holds the input $\mathbf{x} \in \mathbb{Z}_N^{n_0}$ for NN.
Output: P_1 learns $\mathrm{NN}(\mathbf{x})$.
Protocol:

1. P_0 samples MAC key μ uniformly from $\mathbb{Z}_N$ to be used throughout the protocol.
2. **First Linear Layer:** P_0 and P_1 invoke π_{Lin} with inputs $(\textbf{InitLin}, \mathbf{W}_1, \mu)$ and $(\textbf{InitLin}, \mathbf{x})$, respectively. For $b \in \{0,1\}$, P_b learns $\langle \mathbf{s}_1 \rangle_b$.
3. For each $j \in [\ell-1]$,
   - **Non-linear Layer** f_j**:** P_0 and P_1 invoke $\pi^{f_j}_{\text{Non-lin}}$ with inputs $(\langle \mathbf{s}^{Re}_j\rangle_0, \mu, k_0, k_0^{mac})$ and $(\langle \mathbf{s}^{Re}_j\rangle_1, k_1, k_1^{mac})$, respectively. For $b \in \{0,1\}$, P_b learns $(\langle \mathbf{u}_j\rangle_b, \langle \mathbf{v}^{Re}_j\rangle_b, \langle \mathbf{v}^{Im}_j\rangle_b)$, and set $\langle \mathbf{v}_j\rangle_b \leftarrow \texttt{Cod}(\langle \mathbf{v}^{Re}_j\rangle_b + \langle \mathbf{v}^{Im}_j\rangle_b \mathrm{j})$.
   - **Linear Layer** $j+1$**:** P_0 and P_1 invoke $\mathcal{F}_{\text{Lin}}$ with inputs $(\textbf{Lin}, \langle \mathbf{v}_j\rangle_0, \mathbf{W}_{j+1}, \alpha)$ and $(\textbf{Lin}, \langle \mathbf{v}_j\rangle_1, \langle \mathbf{v}^{Re}_j\rangle_1, \langle \mathbf{v}^{Im}_j\rangle_1)$, respectively. For $b \in \{0,1\}$, P_b learns $(\langle \mathbf{s}_{j+1}\rangle_b, \langle \mathbf{t}_{j+1}\rangle_b)$.
4. **Consistency Check:**
   - For $j \in [\ell-1]$, P_0 samples $\mathbf{r}_j \in \mathbb{Z}_N^{n_j}$ and $\mathbf{r}'_{j+1} \in \mathbb{Z}_N^{n_{j+1}}$ and sends $(\mathbf{r}_j, \mathbf{r}'_{j+1})$ to P_1.
   - P_1 computes $\langle q\rangle_1 = \sum_{j\in[\ell-1]} \Big((\langle \mathbf{s}^{Im}_j\rangle_1 - \langle \mathbf{u}_j\rangle_1) * \mathbf{r}_j + (\langle \mathbf{t}_{j+1}\rangle_1 * \mathbf{r}'_{j+1}) \Big)$ and sends it to P_0.
   - P_0 computes $\langle q\rangle_0 = \sum_{j\in[\ell-1]} \Big((\langle \mathbf{s}^{Im}_j\rangle_0 - \langle \mathbf{u}_j\rangle_0) * \mathbf{r}_j + (\langle \mathbf{t}_{j+1}\rangle_0 * \mathbf{r}'_{j+1}) \Big)$.
   - P_0 aborts if $\langle q\rangle_0 + \langle q\rangle_1 \mod p \neq 0$. Else, sends $\langle s_\ell\rangle_0$ to P_1.
5. **Output Phase:** P_1 outputs $\langle \mathbf{s}_\ell\rangle_0 + \langle \mathbf{s}_\ell\rangle_1 \mod p$ if P_0 didn't abort in the previous step.

Fig. 4: Protocol π_{Inf}

0).

Finally, P_0 combines all these verifications into a single check using randomly selected scalars. Upon verification failure, P_0 terminates the protocol; otherwise, P_1 obtains the final share and can proceed with output reconstruction.

Remark 5.1 The method proposed in [20] ensures the malicious security of the FSS protocol by combining two consistency checks: one to prevent selective failure attack, and another to detect tampering with intermediate shares. Selective failure attack aims to determine whether all wire values are zero by repeatedly modifying the α value. However, in machine learning scenarios, it is highly unlikely for all parameters in a linear layer to be zero. Moreover, in CRISP, any incorrect input to the nonlinear layer propagates through subsequent linear layers, causing the computed verification tags to deviate from their expected values. As a result, the protocol will abort during the final consistency check. Based on these observations, we omit verification step (a) from [20], thereby slightly improving efficiency without introducing additional security risks.

B. Correctness and Security

Theorem 3: Protocol π_{Inf} achieves secure computation of machine learning models under a threat model consisting of a semi-honest server P_0 and a malicious client P_1.

Proof: **Correctness**. By correctness of π_{Lin} on **InitLin**, we have $\mathbf{s}_1$, whose corresponding value in the message space is $\mathbf{W}_1\mathbf{x} + \mu\mathbf{W}_1\mathbf{x}\mathrm{j}$. By correctness of π_{Lin} on **Lin**, for each $i \in \{2,\ldots,\ell\}$ it holds that $\mathbf{s}_i$ and $\mathbf{t}_i$, whose corresponding values in the message space are $\mathbf{m}_i\mathbf{v}_{i-1} + \mu\mathbf{m}_i\mathbf{v}_{i-1}\mathrm{j}$ and $\mu^3\texttt{Re}(\mathbf{v}_{i-1}) - \mu^2\texttt{Im}(\mathbf{v}_{i-1})$. By correctness of $\pi^{f}_{\text{Non-lin}}$, for each $i \in [\ell-1]$ it follows that $\mathbf{u}_i = \mu\mathbf{s}_i$, $\mathbf{v}^{Re}_i = f_i(\mathbf{s}_i)$ and $\mathbf{v}^{Im}_i = \mu f_i(\mathbf{s}_i)$. On substituting, it is easy to see that $q = 0$ since for each $i \in [\ell-1]$, $\mathbf{t}_i = 0$, $\mathbf{s}^{Im}_i = \mathbf{u}_i$. Finally, we conclude correctness by noting that $s_\ell = NN(x)$.

Security. We prove that our protocol is secure against a semihonest server P_0 and a malicious client P_1 using simulation based security. For the scenario with a semi-honest adversary corrupting P_0: During the evaluation phase, P_0 only learns a share of the outputs from π_{Lin} and $\pi^{f}_{\text{Non-lin}}$, which is indistinguishable from random values chosen from the field by the simulator. In the consistency check phase, P_0 computes $\langle q\rangle_0$ and receives $\langle q\rangle_1$. We configure the simulator to know that $q = 0$ and receive P_0's input. Consequently, the simulator can compute $\langle q\rangle_0$ and set $\langle q\rangle_1 = -\langle q\rangle_0$. Due to the constraint of $q = 0$ and the randomness of $\mathbf{r}$ and $\mathbf{r}'$, P_0 cannot distinguish between the simulator's output and the output in the real world. Now, we prove security against a malicious client in the following claim.

claim 2: Protocol π_{Inf} is secure against a malicious adversary $\mathcal{A}$ controlling client P_1.

Proof: The adversary $\mathcal{A}$ may deviate arbitrarily from the protocol specifications. In protocol π_{Inf}, this manifests as:

1) In the **Initlin** phase of π_{Lin}, $\mathcal{A}$ can provide input $\mathbf{x}' \neq \mathbf{x}$(client's original input) and obtain $\langle \mathbf{s}\rangle_1$.
2) For each $i \in [\ell-1]$, $\mathcal{A}$ can invoke $\pi^{f}_{\text{Non-lin}}$ on input $\langle \mathbf{s}'^{Re}_i\rangle_1 = \langle \mathbf{s}^{Re}_i\rangle_1 + \Delta^1_i$ and receive $\langle \mathbf{u}_i\rangle_1$, $\langle \mathbf{v}^{Re}_i\rangle_1$, $\langle \mathbf{v}^{Im}_i\rangle_1$.
3) In the **Lin** phase of π_{Lin}, $\mathcal{A}$ can provide inputs $\langle \mathbf{v}'^{Re}_i\rangle_1 = \langle \mathbf{v}^{Re}_i\rangle_1 + \Delta^2_i$, $\langle \mathbf{v}'^{Im}_i\rangle_1 = \langle \mathbf{v}^{Im}_i\rangle_1 + \Delta^3_i$, $\langle \mathbf{v}'_i\rangle_1 = \langle \mathbf{v}_i\rangle_1 +$

$\texttt{Ecd}(\Delta_i^4 + \Delta_i^{4'}\text{j})$ and learn $\langle \mathbf{s}_{i+1}\rangle_1$, $\langle \mathbf{t}_{i+1}\rangle_1$.

4) During consistency check phase, $\mathcal{A}$ can modify $\langle q\rangle_1$ to $\langle q\rangle_1 + \Delta^5$ before sending it to P_0

In the above description, if all Δ values are 0, then $\mathcal{A}$ can be considered to have honestly followed the protocol specifications.

Therefore, we can summarize two distinct scenarios: In the case of protocol compliance, the simulator and real views are demonstrably identical. In the case of protocol deviation (characterized by the existence of a non-zero Δ), the simulator terminates with probability 1. This is primarily due to the authentication value properties that guarantee detection of illegal inputs. Similarly, in the real world, P_0 terminates the protocol with overwhelming probability under such circumstances.

claim 3: In real execution, if at least one of the Δ is nonzero, the probability of P_0 continuing the protocol is negligible.

Proof: Based on the analysis presented in *Claim 1*, we have that:

$$\mathbf{s}_1^{Im} = \mu \mathbf{W}_1 \mathbf{x}' \tag{4}$$

$$\mathbf{u}_1 = \mu(\mathbf{W}_1 \mathbf{x}' + \Delta_1^1) \tag{5}$$

For $i \in \{2, ..., \ell - 1\}$

$$\mathbf{s}_i^{Im} = \mathbf{W}_i(\mathbf{v}_{i-1}^{Im} + \Delta_{i-1}^{4'}) \tag{6}$$

$$\mathbf{u}_i = \mu(\mathbf{W}_i(\mathbf{v}_{i-1}^{Re} + \Delta_i^4) + \Delta_i^1) \tag{7}$$

For $i \in [\ell - 1]$

$$\mathbf{v}_i^{Im} = \mu \mathbf{v}_i^{Re} \tag{8}$$

$$\mathbf{t}_{i+1} = (\mu^3(\mathbf{v}_i^{Re} + \Delta_i^2) - \mu^2(\mathbf{v}_i^{Im} + \Delta_i^3)) \tag{9}$$

$$q = \Delta^5 + \sum_{i=1}^{\ell-1}((\mathbf{s}_i^{Im} - \mathbf{u}_i) * \mathbf{r}_i + \mathbf{t}_{i+1} * \mathbf{r}'_i) \tag{10}$$

By substituting equations (4) through (9) into equation (10), we obtain that:

$$\begin{aligned} q =& \Delta^5 + \sum_{i=2}^{\ell-a}((\mathbf{W}_i \Delta_{i-1}^{4'}) * \mathbf{r}_i) \\ &- \mu((\sum_{i=1}^{\ell-1} \Delta_i^1 + \sum_{i=2}^{\ell-1} \mathbf{W}_i \Delta_{i-1}^4) * \mathbf{r}_i) \\ &+ \mu^3(\sum_{i=1}^{\ell-1}(\Delta_i^2) * \mathbf{r}'_{i+1}) - \mu^2(\sum_{i=1}^{\ell-1} \Delta_i^3 * \mathbf{r}'_{i+1}) \end{aligned} \tag{11}$$

The RHS of Equation (11) is a degree-3 polynomial in μ, denoted by $Q(\mu)$. When $\mathcal{A}$ introduces any error, at least one Δ must be non-zero, resulting in $Q(\mu)$ being a non-zero polynomial. Unlike in finite fields, the probability analysis in $\mathbb{Z}_N$ is more intricate due to the presence of zero divisors and the ring's algebraic structure.

First, let us examine the case where $\mathbf{r}$ and $\mathbf{r}'$ are chosen to make all coefficients zero in $Q(\mu)$. When there exists a non-zero Δ, with $gcd(\Delta, 2^n) = 2^k$, the probability of choosing $\mathbf{r}$ and $\mathbf{r}'$ that result in all coefficients being zero is at least $1/2^{n-k}$. This is fundamentally different from the finite field case, as the ring structure allows for more solutions due to zero divisors.

Next, consider the scenario where $Q(\mu)$ is a non-zero polynomial, and we analyze the probability of choosing an μ that satisfies $Q(\mu) = 0$. Given that $Q(\mu)$ is a degree-3 polynomial with leading coefficient a_3, where $gcd(a_3, 2^n) = 2^m$, the equation $Q(\mu) = 0$ may have up to 2^m solutions in $\mathbb{Z}_N$. Consequently, the probability of selecting a μ that makes $Q(\mu) = 0$ is at least $1/2^{n-m}$.

Furthermore, the ring structure of $\mathbb{Z}_N$ introduces additional complexities. The polynomial $Q(\mu)$ might degenerate to a lower degree due to specific combinations of $\mathbf{r}$ and $\mathbf{r}'$, and the presence of zero divisors could lead to unexpected solutions. These factors contribute to a more complex probability landscape than in the finite field setting.

Therefore, the overall success probability of the protocol is bounded by $1 - max(1/2^{n-k}, 1/2^{n-m})$. However, this bound is likely optimistic as it does not account for the potential interaction between different failure cases and additional failure modes introduced by the ring's algebraic properties. The actual success probability is expected to be lower due to the non-independence of these events and the additional complexities introduced by the ring structure of $\mathbb{Z}_N$.

Fortunately, Cramer et al. [44] present an efficient solution to this challenge through a lifting technique: given a security parameter s, the computation is elevated to $\mathbb{Z}_{N'}$, where $N' = 2^{n+s}$. In this framework, data elements x and key μ are elements of $\mathbb{Z}_{N'}$, and the MAC is computed as $m = \mu x \bmod N'$. The verification procedure is restricted to the n least significant bits. As proven in [44], this methodology effectively reduces the adversary's probability of successful cheating to 2^{-s}.

VI. Experimental Evaluation

In this section, we conduct experiments to demonstrate the performance of CRISP.

A. Implementation Details

The experiment is conducted on an Apple M1 Pro processor (10-core CPU@3.2GHz, 16-core GPU) and 16 GB of RAM. All cryptographic computations are performed in a 32-bit integer ring. The specific experimental setup is as follows:

- Similar to the SIMC experimental setup, since the individual components of the connection protocol incur no cost, we implemented each component separately and evaluated their end-to-end execution time. We implemented the CRISP scheme in C++, utilizing the OpenSSL library for secure random number generation, and incorporating the EMP (Efficient Multi-Party Computation) library to support secure two-party computation protocols in a local area network (LAN) environment, ensuring that data exchanges and computations during the inference process remain confidential between the participating parties. Additionally, we employed the homomorphic encryption library provided by SEAL (Simple Encrypted

Arithmetic Library) to implement the CKKS scheme for the linear layers.

B. Performance of Non-Linear Layers

We compared the computational and communication overheads of SIMC and CRISP in implementing the nonlinear ReLU activation function. Given the characteristics of the activation layer, where ReLU operates primarily on each element of the matrix, a significant number of activation function evaluations are required when entering the activation layer. As a result, we focus on highlighting the differences between the two approaches in terms of the cost associated with executing varying numbers of ReLU functions. In this experiment, we set the security parameter $\lambda = 128$ bits and the plaintext space to $\mathbb{Z}_{2^{32}}$. In addition, we use four threads to parallelize the computation in both of the above schemes.

Computation Overhead: Figure 5 shows the running time of SIMC and CRISP for different numbers of ReLU functions. Our method improves the running time by 10%-90% compared to SIMC. At lower ReLU densities, the CRISP approach exhibits marked superiority. Nevertheless, this comparative advantage progressively decreases with the increasing number of ReLU. Based on our analysis, this phenomenon is primarily attributed to the distinct fixed overhead structures of these two protocols. The SIMC protocol, which relies on garbled circuits, necessitates complex circuit generation and label transmission operations, introducing substantial protocol initialization overhead. In contrast, the CRISP protocol leverages function secret sharing mechanisms, thereby simplifying the protocol structure, reducing communication volume, and lowering fixed overheads that are independent of computation scale. Consequently, when the number of ReLU operations is relatively small, these fixed overheads constitute a significant proportion of the total execution time, allowing CRISP to demonstrate a marked advantage. However, as the number of ReLU operations increases to 2^{14} and beyond, communication and computational costs that scale proportionally with computation size gradually become the dominant factors, causing the asymptotic performance of both protocols to converge and narrowing the performance gap. Furthermore, in the SIMC non-linear layer, participants exclusively assume the role of either sender or receiver, whereas CRISP requires bidirectional information exchange between both parties to reconstruct x+r during each ReLU execution, resulting in cumulative communication latency when processing large quantities of ReLU operations.

Communication Overhead: Figure 6 illustrates the communication overhead comparison between SIMC and our proposed method when computing varying numbers of ReLU functions.In the nonlinear intermediate layer, CRISP only requires a single round of communication to reconstruct $x+r$, followed by local computations to obtain the shares of μx, $f(x)$ and $\mu f(x)$. In contrast, SIMC necessitates a substantial number of garbled circuit labels for one-time pad calculations. This design results in a communication overhead reduction of approximately 96%-99% for CRISP compared to SIMC.

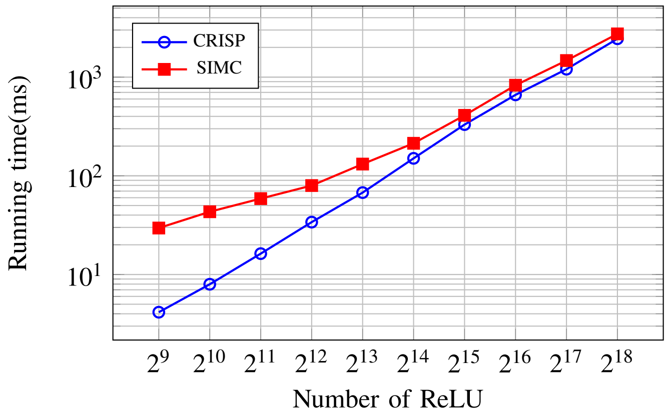


Fig. 5: Non-linear layer computational overhead

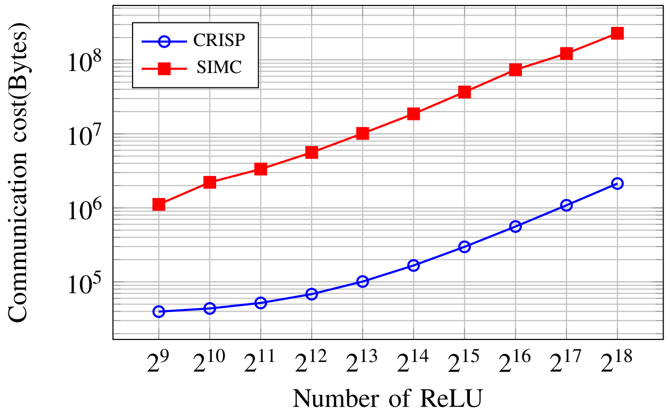


Fig. 6: Non-linear layer communication overhead

TABLE I: Compare of Fully Connected Layer

	Polynomial Modulus	Comm. (MB)		Lantency (s)	
		SIMC	CRISP	SIMC	CRISP
Benchmark A	8192 16384	53 432	54 433	0.65 4.66	0.49 1.52
Benchmark B	8192 16384	53 429	54 429	0.64 4.29	0.45 1.42

Furthermore, as the number of ReLU operations increases, the savings in communication overhead grow accordingly. For instance, when evaluating 2^{18} ReLU gates, CRISP requires only about 2 KB of communication, whereas the SIMC approach demands at least 218 MB. This advantage is non-trivial, as mainstream DNN models typically involve tens of thousands of ReLU operations.

C. Performance of Secure Inference

Following the idea of SIMC, we evaluate the performance of secure inference on two benchmarks: benchmark A, a 2-layer convolutional neural network trained on MNIST, and benchmark B, a 7-layer CNN architecture for CIFAR-10 classification. For details on networks, see [10], [35].

Considering the impact of polynomial modulus on homomorphic encryption performance in linear layers, we first

TABLE II: Compare of Offline Phase

	Comm. (MB)		Lantency (s)	
	SIMC	CRISP	SIMC	CRISP
Benchmark A	266	5	4.26	1.45
Benchmark B	4063	78	32.81	16.37

TABLE III: Compare of Online Phase

		Comm. (MB)		Lantency (s)	
		SIMC	CRISP	SIMC	CRISP
Benchmark A	Linear Layer	48	49	6.73	4.57
	Non-linear Layer	126	0.11	1.53	0.09
	Total	174	49.11	8.26	4.66
Benchmark B	Linear Layer	135	135	32.37	20.86
	Non-linear Layer	2109	1.36	2.14	1.58
	Total	2244	136.36	34.51	22.44

conducted comparative experiments exclusively on fully connected layers. As illustrated in Table I, regarding computational latency, CRISP achieved performance improvements ranging from 24.6% to 67.3% compared to SIMC, with the most substantial efficiency gain of 67.3% observed when processing larger polynomial modulus scales (16384). This demonstrates that CRISP's approach of placing verification labels in the complex domain for synchronous computation effectively enhances computational performance. In terms of communication overhead, CRISP and SIMC performed similarly. Indeed, both SIMC and CRISP protocols transmitted five ciphertexts during the linear layer phase, which aligns with intuitive expectations for both protocols. The experimental results indicate that CRISP exhibits increasingly pronounced communication performance advantages as the polynomial modulus scale expands, making it particularly suitable for application scenarios requiring enhanced security levels or larger data volumes.

As shown in Table II and Table III, we conducted a comprehensive performance evaluation of CRISP and SIMC under two different benchmark scales.

In offline phase, CRISP eliminates the need to generate a large number of circuits and tags, resulting in a twofold improvement in computational efficiency and a 50-fold reduction in communication overhead.

In online phase, CRISP achieves a 32-36% latency reduction in linear layers while maintaining communication volume comparable to SIMC. However, in processing non-linear layers, CRISP demonstrates excellent performance, reducing communication overhead by nearly two orders of magnitude while significantly decreasing computational latency. This architectural optimization translates into substantial overall advantages: in Benchmark A, CRISP reduces total communication cost from 174MB to 49.11MB and total latency from 8.26 seconds to 4.66 seconds. In the more complex Benchmark B, these advantages further expand, with total communication

TABLE IV: Compare of Model Accuracy

Dataset	Model	Accuracy	
		Plaintext	CRISP
MNIST	Benchmark A	99.31%	99.23%
CIFAR-10	Benchmark B	81.61%	81.56%

volume decreasing from 2244MB to 136.36MB and total latency shortening from 34.51 seconds to 22.44 seconds. These data highlight CRISP's exceptional performance in handling large-scale secure inference tasks, making it an ideal choice for scenarios with high security requirements.

D. Model Accuracy

A key requirement for private inference is that the introduced cryptographic protocols should not compromise model performance. To validate this, we use the EzPC framework to test inference accuracy on two benchmarks for both plaintext and CRISP, as shown in Table II. The results indicate that, compared to the plaintext model, CRISP causes a slight decrease in accuracy. This performance degradation can likely be attributed to three factors: 1. precision loss due to fixed-point arithmetic. 2. errors introduced during encryption and decryption by the approximate homomorphic encryption algorithm CKKS, and 3. Equation (2) in Section III-B are not hold (small probability).

VII. Conclusion

In this paper, we propose CRISP, a system designed for secure machine learning inference against malicious clients. CRISP effectively avoids the issue in non-linear layers of SIMC through an FSS-based approach, significantly reducing communication overhead while maintaining computational efficiency. Additionally, we introduce a novel acceleration method for linear layer computation that minimizes expensive redundant multiplication operations through the construction of a complex domain verification scheme. In future work, we will focus on designing more effective optimization strategies to further reduce the computational costs of CRISP and explore its extension to more complex machine learning models, making secure ML inference suitable for a broader range of practical applications.

References

[1] M. Bakator and D. Radosav, "Deep learning and medical diagnosis: A review of literature," *Multimodal Technologies and Interaction*, vol. 2, no. 3, p. 47, 2018.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[3] N. Nazareth and Y. V. R. Reddy, "Financial applications of machine learning: A literature review," *Expert Systems with Applications*, vol. 219, p. 119640, 2023.

[4] K. Nyarko, P. Taiwo, C. Duru, and E. Masa-Ibi, "Ai/ml systems engineering workbench framework," in *2023 57th Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2023, pp. 1–5.

[5] G. Xu, G. Li, S. Guo, T. Zhang, and H. Li, "Secure decentralized image classification with multiparty homomorphic encryption," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 33, no. 7, pp. 3185–3198, 2023.

[6] M. Bellare, V. T. Hoang, and P. Rogaway, "Foundations of garbled circuits," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 784–796.

[7] E. Boyle, N. Gilboa, and Y. Ishai, "Secure computation with preprocessing via function secret sharing," in *Theory of Cryptography: 17th International Conference, TCC 2019, Nuremberg, Germany, December 1–5, 2019, Proceedings, Part I 17*. Springer, 2019, pp. 341–371.

[8] Q. Lou and L. Jiang, "Hemet: A homomorphic-encryption-friendly privacy-preserving mobile neural network architecture," in *International conference on machine learning*. PMLR, 2021, pp. 7102–7110.

[9] S. U. Hussain, M. Javaheripi, M. Samragh, and F. Koushanfar, "Coinn: Crypto/ml codesign for oblivious inference via neural networks," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 3266–3281.

[10] R. Lehmkuhl, P. Mishra, A. Srinivasan, and R. A. Popa, "Muse: Secure inference resilient to malicious clients," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2201–2218.

[11] W. Z. Srinivasan, P. Akshayaram, and P. R. Ada, "Delphi: A cryptographic inference service for neural networks," in *Proc. 29th USENIX secur. symp*, vol. 3, 2019.

[12] N. Chandran, D. Gupta, S. L. B. Obbattu, and A. Shah, "{SIMC}:{ML} inference secure against malicious clients at {Semi-Honest} cost," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 1361–1378.

[13] Z. Ghodsi, A. K. Veldanda, B. Reagen, and S. Garg, "Cryptonas: Private inference on a relu budget," *Advances in Neural Information Processing Systems*, vol. 33, pp. 16 961–16 971, 2020.

[14] S. Sav, A. Pyrgelis, J. R. Troncoso-Pastoriza, D. Froelicher, J.-P. Bossuat, J. S. Sousa, and J.-P. Hubaux, "Poseidon: Privacy-preserving federated neural network learning," *arXiv preprint arXiv:2009.00349*, 2020.

[15] W.-j. Lu, Z. Huang, Z. Gu, J. Li, J. Liu, C. Hong, K. Ren, T. Wei, and W. Chen, "Bumblebee: Secure two-party inference framework for large transformers," *Cryptology ePrint Archive*, 2023.

[16] D. Rathee, M. Rathee, R. K. K. Goli, D. Gupta, R. Sharma, N. Chandran, and A. Rastogi, "Sirnn: A math library for secure rnn inference," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1003–1020.

[17] M. Hao, H. Li, H. Chen, P. Xing, G. Xu, and T. Zhang, "Iron: Private inference on transformers," *Advances in neural information processing systems*, vol. 35, pp. 15 718–15 731, 2022.

[18] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow2: Practical 2-party secure inference," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 325–342.

[19] M. Hao, H. Li, H. Chen, P. Xing, and T. Zhang, "Fastsecnet: An efficient cryptographic framework for private neural network inference," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 2569–2582, 2023.

[20] E. Boyle, N. Chandran, N. Gilboa, D. Gupta, Y. Ishai, N. Kumar, and M. Rathee, "Function secret sharing for mixed-mode and fixed-point secure computation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2021, pp. 871–900.

[21] C. Gentry, *A fully homomorphic encryption scheme*. Stanford university, 2009.

[22] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. Springer, 2017, pp. 409–437.

[23] M. Keller, E. Orsini, and P. Scholl, "Actively secure ot extension with optimal overhead," in *Annual Cryptology Conference*. Springer, 2015, pp. 724–741.

[24] A. C.-C. Yao, "How to generate and exchange secrets," in *27th annual symposium on foundations of computer science (Sfcs 1986)*. IEEE, 1986, pp. 162–167.

[25] Y. Wang and Q. M. Malluhi, "Reducing garbled circuit size while preserving circuit gate privacy," in *International Conference on Cryptology in Africa*. Springer, 2024, pp. 149–173.

[26] V. Kolesnikov, "Gate evaluation secret sharing and secure one-round two-party computation," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2005, pp. 136–155.

[27] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free xor gates and applications," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2008, pp. 486–498.

[28] S. Goldwasser, Y. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich, "Reusable garbled circuits and succinct functional encryption," in *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, 2013, pp. 555–564.

[29] S. Agrawal, "Stronger security for reusable garbled circuits, general definitions and attacks," in *Annual international cryptology conference*. Springer, 2017, pp. 3–35.

[30] M. Juuti, S. Szyller, S. Marchal, and N. Asokan, "Prada: protecting against dnn model stealing attacks," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 512–527.

[31] X. Gong, Q. Wang, Y. Chen, W. Yang, and X. Jiang, "Model extraction attacks and defenses on cloud-based machine learning models," *IEEE Communications Magazine*, vol. 58, no. 12, pp. 83–89, 2021.

[32] Y. Zhu, Y. Cheng, H. Zhou, and Y. Lu, "Hermes attack: Steal {DNN} models with lossless inference accuracy," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[33] X. Zhang, C. Chen, Y. Xie, X. Chen, J. Zhang, and Y. Xiang, "A survey on privacy inference attacks and defenses in cloud-based deep neural network," *Computer Standards & Interfaces*, vol. 83, p. 103672, 2023.

[34] T. Nayan, Q. Guo, M. Al Duniawi, M. Botacin, S. Uluagac, and R. Sun, "{SoK}: All you need to know about {On-Device}{ML} model extraction-the gap between research and practice," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 5233–5250.

[35] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 619–631.

[36] T. Orekondy, B. Schiele, and M. Fritz, "Knockoff nets: Stealing functionality of black-box models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4954–4963.

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

[38] M. Nawaz, A. Gulati, K. Liu, V. Agrawal, P. Ananth, and T. Gupta, "Accelerating 2pc-based ml with limited trusted hardware," *arXiv preprint arXiv:2009.05566*, 2020.

[39] J. Katz, S. Ranellucci, M. Rosulek, and X. Wang, "Optimizing authenticated garbling for faster secure two-party computation," in *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part III 38*. Springer, 2018, pp. 365–391.

[40] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl, "Efficient two-round ot extension and silent non-interactive secure computation," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 291–308.

[41] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl, "Efficient pseudorandom correlation generators from ring-lpn," in *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II 40*. Springer, 2020, pp. 387–416.

[42] M. Keller, V. Pastro, and D. Rotaru, "Overdrive: Making spdz great again," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 158–189.

[43] Z. Huang, W.-j. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure {Two-Party} deep neural network inference," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 809–826.

[44] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing, "Spd: efficient mpc mod for dishonest majority," in *Annual International Cryptology Conference*. Springer, 2018, pp. 769–798.

Appendix A
Non-linear protocol in SIMC

The nonlinear protocol details of SIMC are shown in Figure 7.

Preamble: The function f is such that $f : \mathbb{F}_p \to \mathbb{F}_p$. Consider a boolean circuit Comp^f that takes additive shares of $a \in \mathbb{F}_p$, i.e., $\langle a\rangle_0, \langle a\rangle_1 \in \mathbb{F}_p$, as input and outputs $(a, f(a))$. Let $\text{Trim}_n : \{0,1\}^\lambda \to \{0,1\}^n$ be a function that outputs the last n bits of input.
Input: P_0 inputs $\langle s\rangle_0 \in \mathbb{F}_p^n$ and $\alpha \in \mathbb{F}_p$. P_1 inputs $\langle s\rangle_1 \in \mathbb{F}_p^n$.
Output: P_b learns $\langle \alpha s\rangle_b$, $\langle f(s)\rangle_b$, $\langle \alpha f(s)\rangle_b$ for $b \in \{0,1\}$.
Protocol:

1) **Garbled Circuit Phase:**
   - P_0 computes $(\text{GC}, \{\{\text{lab}^{\text{in}}_{i,j}\}_{i\in[2k]}, \{\text{lab}^{\text{out}}_{i,j}\}_{i\in[2k]}\}_{j\in\{0,1\}}) \leftarrow \text{Garble}(1^\lambda, \text{Comp}^f)$.
   - P_0 and P_1 invoke OT^k_λ where P_0 is the sender and P_1 is the receiver with inputs $\{\text{lab}^{\text{in}}_{i,0}\}_{i\in\{\kappa+1,\ldots,2k\}}$ and $\langle s\rangle_1$, respectively. P_1 learns $\{\widetilde{\text{lab}}^{\text{in}}_i\}_{i\in[\kappa+1,\ldots,2k]}$. P_0 sends GC and $\{\text{lab}^{\text{in}}_i = \text{lab}^{\text{in}}_{i,\langle s\rangle_0[i]}\}_{i\in[\kappa]}$ to P_1.
   - P_1 computes $\{\widetilde{\text{lab}}^{\text{out}}_i\}_{i\in[2k]} \leftarrow \text{GCEval}(\text{GC}, \{\widetilde{\text{lab}}^{\text{in}}_i\}_{i\in[2k]})$.
2) **Authentication Phase:**
   - For $i \in [\kappa]$, P_0 chooses $\eta_{i,0}, \delta_{i,0}, v_{i,0} \in \mathbb{F}_p$ and sets $(\eta_{i,1}, \delta_{i,1}, v_{i,1}) = (1 + \eta_{i,0}, \alpha + \delta_{i,0}, \alpha + v_{i,0})$.
   - For $i \in [2\kappa]$ and $j \in \{0,1\}$, P_0 parses $\text{lab}^{\text{out}}_{i,j}$ as $p_{i,j}\|k_{i,j}$ where $p_{i,j} \in \{0,1\}$ and $k_{i,j} \in \{0,1\}^{\lambda-1}$.
   - For $i \in [\kappa], j \in \{0,1\}$, P_0 sends $ct_{i,p_{i,j}} = v_{i,j} \oplus \text{Trim}_n(k_{i,j})$ and $\widehat{ct}_{i,p_{i,j}} = (\eta_{i,j}\|\delta_{i,j}) \oplus \text{Trim}_{2k}(k_{i+\kappa,j})$.
   - For $i \in [2\kappa]$, P_1 parses $\widetilde{\text{lab}}^{\text{out}}_i$ as $\widetilde{p}_i\|\widetilde{k}_i$ where $\widetilde{p}_i \in \{0,1\}$ and $\widetilde{k}_i \in \{0,1\}^{\lambda-1}$.
   - For $i \in [\kappa]$, P_1 computes $c_i = ct_{i,\widetilde{p}_i} \oplus \text{Trim}_n(\widetilde{k}_i)$ and $(d_i\|e_i) = \widehat{ct}_{i,\widetilde{p}_i} \oplus \text{Trim}_{2k}(\widetilde{k}_{i+\kappa})$.
3) **Local Computation Phase:**
   - P_0 outputs $\langle z_1\rangle_0 = -\left(\sum_{i\in[\kappa]} v_{i,0}2^{i-1}\right)$, $\langle z_2\rangle_0 = -\left(\sum_{i\in[\kappa]} \eta_{i,0}2^{i-1}\right)$, and $\langle z_3\rangle_0 = -\left(\sum_{i\in[\kappa]} \delta_{i,0}2^{i-1}\right)$.
   - P_1 outputs $\langle z_1\rangle_1 = \left(\sum_{i\in[\kappa]} c_i2^{i-1}\right)$, $\langle z_2\rangle_1 = \left(\sum_{i\in[\kappa]} d_i2^{i-1}\right)$, and $\langle z_3\rangle_1 = \left(\sum_{i\in[\kappa]} e_i2^{i-1}\right)$.

Fig. 7: Protocol $\pi^f_{\text{SIMC-Non-lin}}$

Appendix B
maxpool experiment

As noted in Remark 3.1, our framework adopts a modular design that supports the substitution of specific nonlinear functions and underlying protocols. To demonstrate the generalizability of CRISP, we evaluate the performance of the MaxPool function on Benchmark A.

TABLE V: Compare of maxpool

	Comm. (KB)		Lantency (ms)	
	SIMC	CRISP	SIMC	CRISP
Benchmark A	31983	54	325.66	63.39

Table V shows that the MaxPool function implemented using the CRISP framework achieves approximately a 5× improvement in computational efficiency and nearly two orders of magnitude reduction in communication overhead compared to the GC-based MaxPool. These results are consistent with the performance gains observed in our ReLU experiments.

Appendix C
additive operate

Due to the necessity of ensuring synchronous computation of verification labels in the complex domain, additional processing is required during constant addition operations. In machine learning scenarios, additive constants are model parameters owned by the server, such as bias B. Therefore, the server must construct $B + \mu B$j during the offline phase, extending b into the complex domain to accommodate addition operations. We illustrate this process using a fully connected layer with bias as an example, with specific details presented in Figure 8.

Input: P_0 holds $\langle\mathbf{m}\rangle_0 \in \mathbb{C}^n_N$, $\mathbf{W} \in \mathbb{Z}^{n'\times n}_N$, $\mu \in \mathbb{Z}_N$ and processsed bias $B' = \texttt{Cod}(B+\mu B\text{j}) \in \mathbb{Z}^m_N$. The plaintext corresponding to $\mathbf{m}$ is $\mathbf{x} + \mu\mathbf{x}$. P_1 holds $\langle\mathbf{m}\rangle_1 \in \mathbb{C}^n_N$.
Output: P_b learns $\langle\mathbf{Wm} + B'\rangle_b$ for $b \in \{0,1\}$.

- P_1 sends the encryption $\mathbf{C}_1 \leftarrow \texttt{Enc}(\text{pk}, \langle\mathbf{m}\rangle_1)$ to P_0.
- P_0 sets $\mathbf{m} = \mathbf{C}_1 + \texttt{Ecd}(\langle\mathbf{m}\rangle_0)$
- P_0 homomorphically evaluates the ciphertexts $\mathbf{Wm}+B'$, which plaintext is $\mathbf{xW}+B+\mu(\mathbf{xW}+B)$j.
- P_0 samples $\langle\mathbf{Wm} + B'\rangle_0 \in_R \mathbb{Z}^{n'}_N$.
- P_0 computes $\mathbf{C}_2 \leftarrow \texttt{Enc}(\text{pk}, \mathbf{Wm} + B' - \langle\mathbf{Wm} + B'\rangle_0)$, and sends to P_1.
- P_b outputs $\langle\mathbf{Wm} + B'\rangle_b$ for $b \in \{0,1\}$.

Fig. 8: Fully connected layer with addition operation