SECO: Secure Inference With Model Splitting Across Multi-Server Hierarchy

Shuangyi Chen*, and Ashish Khisti *

*Department of Electrical and Computer Engineering, University of Toronto, {shuangyi.chen@mail.utoronto.ca, akhisti@ece.utoronto.ca}

Abstract—In the context of prediction-as-a-service, concerns about the privacy of the data and the model have been brought up and tackled via secure inference protocols. These protocols are built up by using single or multiple cryptographic tools designed under a variety of different security assumptions.

In this paper, we introduce SECO, a secure inference protocol that enables a user holding an input data vector and multiple server nodes deployed with a split neural network model to collaboratively compute the prediction, without compromising either party's data privacy. We extend prior work on secure inference that requires the entire neural network model to be located on a single server node, to a multi-server hierarchy, where the user communicates to a gateway server node, which in turn communicates to remote server nodes. The inference task is split across the server nodes and must be performed over an encrypted copy of the data vector.

We adopt multiparty homomorphic encryption and multiparty garbled circuit schemes, making the system secure against dishonest majority of semi-honest servers as well as protecting the partial model structure from the user. We evaluate SECO on multiple models, achieving the reduction of computation and communication cost for the user, making the protocol applicable to user's devices with limited resources.

Index Terms—Distributed machine learning, Security and privacy

I. INTRODUCTION

Prediction-as-a-service using neural network models enables a company to deploy a neural network on the cloud and allows users to upload their input and obtain the corresponding prediction results. However, this may lead to privacy concerns and legal issues. There are regulatory constraints such as Health Insurance Portability and Accountability Act (HIPAA) and EU General Data Protection Regulation (GDPR), restricting the way data could be used or shared. A secure inference protocol enables a user, holding its input, and servers deployed with the neural network to collaboratively evaluate the prediction result, preserving the privacy of both the input and the model parameters. Many cryptographic tools, such as Homomorphic Encryption (HE) [1] and secure multiparty computation (MPC), have been applied in this area. Homomorphic encryption allows the data holder to encrypt its data with the public key, any other party who obtains the encrypted data can compute on it without decryption, and only the party holding the secret key can decrypt the computed result. MPC is usually a mixed approach that combines secret sharing and Garbled Circuit [2], etc. Both HE and MPC can protect data privacy. Pure HE-based secure inference protocols [3][4][5][6] provides the data and model confidentiality as well

as protects the structure of the model. However, it involves heavy computation and does not immediately generalize to non-linear function computations used during predictions. One popular approach in HE-based protocols is to replace the activation functions in neural networks with approximated polynomials at the cost of reduced accuracy. While MPC protocols are not as computationally heavy as HE-based protocols, they require multiple rounds of communication and can also leak some information about the model to the users. Furthermore, there are hybrid approaches that combine HE and MPC. GAZELLE [7] relies on a hybrid approach by employing HE and two-party computation, performing linear function evaluation with HE and non-linear function evaluation with Garbled Circuit. DELPHI [8] builds upon techniques from GAZELLE and modifies it to a two-phase protocol. It reduces the use of heavy cryptographic tools in the online phase, thus speeding up online inference. However, these methods come with certain limitations. Firstly, they reveal the neural network's architecture to the user, which is problematic when the model's structure is intended to remain confidential, a common scenario with today's large language models. Secondly, they require user equipment to possess substantial computational power and communication capabilities. Additionally, they operate under the assumption that the model is hosted on a single server, which is impractical for very large models. For instance, processing a model with the size of GPT-4 necessitates a cluster of multiple GPUs. To process such a large model effectively, distributing the model parameters across several servers, each with limited GPU resources, can enhance processing power.

With these considerations, we propose SECO, a hybrid HE-MPC secure inference protocol that involves a hierarchy of servers as shown in Figure 1. The user communicates with a *gateway* server that stores a part (the parameters associated with the first few layers) of the neural network model. The remainder of the neural network model is stored on remote servers, with each server holding a share of the weights of the linear layers. Our setup extends the secure inference protocol proposed in [8] to such a multi-server scenario, which has several advantages. First, the user only needs to participate during the evaluation involving part of the network model stored on the gateway server. It does not need to participate during evaluations involving the network model stored on the remote servers. This reduces the communication and computational requirements for the user device while



Fig. 1. Protocol Architecture

increasing the load on the servers, which enables the use of SECO on lightweight user devices. Furthermore although as in prior works [8], the structures of the layers in the neural network model stored on the gateway server must be revealed to the user, the structures of the layers on the remote server are not revealed to the user. SECO offers the benefit of hiding most of the neural network's architecture, crucial for protecting the proprietary structures of large models like GPT-4, where both the model's parameters and the model's design are valuable assets. We demonstrate that a straightforward application of prior works to the hierarchical server setting can lead to information leakage. Our proposed setting involves storing a secret share of model weights at the remote server. However, for secure inference of the partial model on the remote servers, a straightforward extension of DELPHI's 2PC protocol to three-party setting with distributed weights is not privacypreserving due to the limitation of the plain HE scheme. To privately perform the execution of secure inference on the remote servers, we design a sub-protocol relying on multiparty HE, secret sharing, and multiparty Garbled Circuit schemes, ensuring the privacy of each individual data while facilitating efficient communication. We achieve this by 1) designing new secret sharing scheme and integrating it with multiparty HE for secure computation of linear functions 2) departing from the conventional approach for the use of Garbled Circuits (GC) in DELPHI by redesigning the assignment of roles for secure computation of non-linear functions. We formally demonstrate that our proposed method prevents information leakage in the presence of dishonest majority of honest-but-curious servers. We have implemented SECO and reported the performance. The results indicate that SECO significantly enhances efficiency by reducing the user's online participation time by up to $9.9 \times$ and decreasing the user's online communication costs by as much as $18.5 \times$ compared to DELPHI. It also provides a versatile tradeoff between the communication and computation loads between the user and server in addition to protecting more information about the neural network than prior works [8].

II. RELATED WORK

A. Privacy-Preserving Inference on Neural Networks

Secure Inference has received considerable attention in recent years. It is developed based on the following techniques: homomorphic encryption [9], secret sharing and garbled circuits [2]. Some earlier works utilize a single technique and their models are simple machine learning models such as linear regression [10] [11], linear classifiers [12] [13] [14], etc.

a) HE-based Protocols: Pure HE-based protocols usually protect the complete model information including the architecture but lack support for non-approximated activation functions. CryptoNets [3] was one of the earliest works to use homomorphic encryption on neural network inference. CryptoDL [4] improves upon CryptoNets by choosing the interval and the degree of approximations based on heuristics on the data distribution. CHET [5] also improves upon CryptoNets by modifying the approximated activation function that CryptoNets uses to $f(x) = ax^2 + bx$ with learnable parameters a and b. TAPAS [6] utilizes FHE [15] to allow arbitrary-layer neural network processing and proposes tricks to accelerate computation on encrypted data. nGraph-HE [16] develops HE-aware graph-compiler optimizations on general matrixmatrix multiplication operations and convolution-batch-norm operation. CHIMERA [17] uses different HE schemes for ReLU functions and other neural network functions. It needs to switch between two HE schemes to evaluate the model, leading to high computational costs. Cheetah [18] presents a set of optimizations to speed up HE-based secure inference, including auto-tuning HE security parameters, the order scheduler for HE operations, and a hardware accelerator architecture. Pure HE-based protocols involve heavy computation and their performance are not practical due to their costly homomorphic computations.

b) MPC-based Protocols: Pure MPC-based protocols are usually high-throughput but with high communication overhead due to multiplication operations. Furthermore, they usually need to expose the structure of the model to all parties. ABY [19] is a 2PC protocol that supports arbitrary neural network functions by combining Arithmetic, Boolean circuits, and Yao's garbled circuits. ABY3 [20] improves ABY by extending it to 3-party settings and optimizing the conversion between Arithmetic, Boolean circuits, and garbled circuits. 3PC protocol SecureNN [21] distributes 2-out-of-2 secret sharing of the input among two of three servers. However, it is only demonstrated on small DNNs. FALCON, a 3PC protocol proposed by Wagh et al.[22] uses replicated secret sharing to reduce the number of interactions. However, the use of replicated secret sharing in the context of predictionas-a-service necessitates an increase in communication costs for the user. To sum up, MPC-based protocols rely on assumptions such as non-collusion, or an honest majority among the servers. Such a strong assumption of the system might not always align with the practical environments. Conversely, SECO assumes dishonest majority among servers, meaning the user's input privacy can be protected even when most of the servers are corrupted by the adversary. Thus, SECO can offer a higher degree of trustworthiness to users than pure MPC-based protocols. Furthermore, these 3PC protocols require the user to divide the input into multiple random shares. Each input share is then dispatched to three different servers, thereby increasing the communication costs for the user.

c) HE-MPC-based Protocols: HE-MPC-based Protocols are designed by using both HE and MPC in a hybrid scheme. GAZELLE [7] designs schemes for HE neural network operations. It uses HE to evaluate linear layers and Garbled Circuits to evaluate non-linear layers, achieving efficient secure inference with high accuracy. DELPHI [8] modifies GAZELLE by designing two phases with the offline phase using HE, speeding up the online inference by evaluating the neural network in plaintext. It also proposes a novel neural network architecture search planner to determine where to approximate ReLU in the neural network, trading off between performance and accuracy. MP2ML [23] is proposed based on a novel combination of nGraph-HE [16] and ABY [19], combining additive secret sharing and CKKS [24] HE scheme. However, HE-MPC-based Protocols face several limitations. Firstly, they require revealing the neural network's architecture to users, problematic for confidential models like large language models. Secondly, they demand substantial computational power and communication capabilities from user equipment, which can be a barrier for those with limited resources. Thirdly, these methods assume the model is hosted on a single server, impractical for large models like GPT-4 that need multiple GPUs. SECO is derived from these 2PC HE-MPC-based Protocols and is optimized to address the mentioned limitations.

B. Split Neural Network

Split neural network is a specific type of neural network architecture. In this method, the neural network is divided into two or more sections, and each section is processed on different devices or servers. It is often used to reduce load for devices with limited resources. SplitFed [25] is a federated learning framework that combines federated learning with split learning for better model privacy. However, it applies differential privacy rather than cryptographic tools. Pereteanu et al. introduce Split HE [26], a framework where the server provides the user with the middle portion of the neural network. In this setup, the user conducts plaintext inference on the middle section of the network, while the server manages the inference of other parts under homomorphic encryption. However, this approach harms the privacy of the deployed model. Khan et al. propose a secure training protocol named Split Ways [27] based on U-shaped split learning, which incorporates homomorphic encryption on the client side to protect user input privacy. However, while they deploy only the fully-connected layer on the server, the user is still responsible for training the majority of the neural network. Split HE and Split Ways explore the integration of split neural networks with cryptographic tools. However, they do not fully utilize the server's computing resources and continue to depend significantly on the user's device.

III. PRELIMINARIES

A. Split Neural Network

Split Neural Network [28][29] involves splitting a neural network model at different intermediate layers. Consider a neural network function \mathbf{F} composed of a sequence of layers $\{\mathbf{F}_1, ..., \mathbf{F}_L\}$. Each layer \mathbf{F}_i can be a linear layer or an activation layer. The prediction result of the model for a given input \mathbf{x}_1 can be computed by $\mathbf{F}(\mathbf{x}_1) = \mathbf{F}_L(\mathbf{F}_{L-1}(...(\mathbf{F}_1(\mathbf{x}_1))))$. We split the model at the l^{th} layer to get two models $\mathbf{F}_I = \{\mathbf{F}_1, ..., \mathbf{F}_l\}$, $\mathbf{F}_{II} = \{\mathbf{F}_{l+1}, ..., \mathbf{F}_L\}$ which could be stored at two different nodes. The output can be expressed as $\mathbf{F}(\mathbf{x}_1) = \mathbf{F}_{II}(\mathbf{F}_I(\mathbf{x}_1))$. We adjust l to adjust the processing (training or inference) load for the parties deployed with \mathbf{F}_I and \mathbf{F}_{II} .

B. Cryptographic Blocks

We use the following cryptographic blocks to build the protocol.

1) The Plain BFV Homomorphic Encryption: The Brakerskil-Fan-Vercauteren scheme [1] is a Ring-Learning with Errors (RLWE)-based cryptosystem that supports additive and multiplicative homomorphic operations. We define the plaintext space to be $R_t = \mathbb{Z}_t[X]/(X^n+1)$ and the ciphertext space to be $R_q = \mathbb{Z}_q[X]/(X^n + 1)$, where n is a power of 2. We denote $\Delta = \begin{bmatrix} q \\ t \end{bmatrix}$. BFV Homomorphic Encryption scheme can be expressed as a tuple of functionalities HE = (HE.KeyGen, HE.Enc, HE.Dec, HE.Eval). The scheme is based on two kinds of distributions: the key distribution $R_3 = \mathbb{Z}_3[X]/(X^N + 1)$ with coefficients uniformly distributed in $\{-1, 0, 1\}$; the error distribution χ over R_a with coefficients distributed according to a centered discrete Gaussian.

• HE.KeyGen(Params) \rightarrow (pk, sk):It takes a security parameter as input and returns a public key pk and a secret key sk.

Algorithm 1 HE.KeyGen(params) \rightarrow (pk, sk)	
1: Sample $s \leftarrow R_3$	
2: Let $\mathbf{sk} = s$. Sample $p_1 \leftarrow R_q$, $e \leftarrow \chi$	
3: $pk = (p_0, p_1) = (-sp_1 + e, p_1)$. Output (pk, sk)	

 HE.Enc(pk, m) → ct: It takes as input the public key pk and message m, and outputs the ciphertext of m denoted as ct.

Algorithm 2 HE.Enc(pk, m) $\rightarrow ct$

1: Let $\mathsf{pk} = (p_0, p_1)$. Sample $u \leftarrow R_3, e_0, e_1 \leftarrow \chi$ 2: Output $ct = (\Delta m + up_0 + e_0, up_1 + e_1)$

 $2: \quad \Theta uput \ et = (\Delta m + wp_0 + e_0, wp_1 + e_0)$

 HE.Dec(sk, ct) → m: It takes as input the secret key sk and the ciphertext ct, and outputs message m.

Algorithm 3	HE.Dec(sk, ct	$) \rightarrow $	m
-------------	---------	--------	------------------	---

- 1: Let $\mathbf{sk} = s$, $ct = (c_0, c_1)$
- 2: Output $m = \left[\left\lfloor \frac{t}{q} [c_0 + c_1 s]_q \right] \right]_t$

• HE.Eval(pk, $\{ct_i, pt_i\}, f$) $\rightarrow ct_f$: It takes as input the public key pk, valid ciphertexts $\{ct_i\}$ encrypting $\{m_i\}$ or plaintexts $\{pt_i\}$ encoding $\{m_i\}$ and operation f, and returns a valid ciphertext ct_f encrypting $f(\{m_i\}_i)$. The class of function $f(\cdot)$ that are supported include elementwise addition Add, subtraction Sub and multiplication Mul and slot rotation Rot, as well as a combination of those basic operations.

2) Multiparty BFV Homomorphic Encryption (MPHE): The Multiparty version of BFV Homomorphic Encryption scheme (MPHE) [30] is extended from the plain BFV scheme. It enables multiple distributed parties, each with input in private, to collaboratively compute a function of those inputs

without leaking any party's input. In this scheme, a common public key collectively generated by parties is known to all parties, while the corresponding secret key csk is distributed among parties. Below we introduce the main functions used in the protocol. Consider a scenario that N parties want to collaboratively compute a function of their private inputs.

- MPHE.KeyGen(params) \rightarrow (pk_i, sk_i) :Each party runs HE.KeyGen(params) and generates a public key pk_i and secret key sk_i . This procedure requires a public polynomial p_1 , which is agreed upon by all N parties.
- MPHE.DKeyGen $(\{\mathsf{pk}_i\}_{i=0}^{N-1}) \rightarrow \mathsf{cpk}$: It returns a common public key cpk, for a set of public keys $\{pk_i\}_{i=1}^{N-1}$. Its corresponding secret key csk is the aggregation of $\{\mathsf{sk}_i\}_{i=1}^{N-1}$.

Algorithm 4 MPHE.DKeyGen $(\{pk_i\}_{i=0}^{N-1}) \to cpk$	
---	--

1: Let $\mathsf{pk}_i = (p_{0,i}, p_1)$ 2: Output $\mathsf{cpk} = ([\sum_{i=0}^{N-1} p_{0,i}]_q, p_1)$

- MPHE.Enc(cpk, m) $\rightarrow ct$: It returns a ciphertext ct by running HE.Enc(cpk, m).
- MPHE.Eval(cpk, $\{ct_i, pt_i\}, f$) $\rightarrow ct_f$: It returns a ciphertext ct_f by running HE.Eval(cpk, $\{ct_i, pt_i\}, f$).
- MPHE.Reconstruct $(ct, \mathbf{sk}_i) \rightarrow pd_i$:Used by each party *i*. It takes as input a publicly known ciphertext *ct*, party's secret key \mathbf{sk}_i , outputs a partial decryption pd_i .

Algorithm 5 MPHE.Reconstruct $(ct, sk_i) \rightarrow pd_i$	
1: Let $\mathbf{sk}_i = s_i, ct = (c_0, c_1)$. Sample $e_i \leftarrow \chi$	

2: Output $pd_i = s_ic_1 + e_i$

• MPHE.Dec($\{pd_i\}_{i=0}^{N-1}, ct$) $\rightarrow m'$: It takes as input a set of partial decryption $\{pd_i\}_{i=0}^{N-1}$ and the corresponding publicly known ciphertext ct. By aggregating the partial decryption, it outputs m' which is equal to HE.Dec(ct, csk), provided that the noise powers are again selected in a suitable manner.

3) Oblivious Transfer (OT): Oblivious transfer [31] [32] is a protocol involving two parties, a sender who has an input two messages m_0, m_1 , and a receiver who holds a bit b as input. Via OT, the receiver obtains m_b . The security of the protocol guarantees that the sender does not learn anything Algorithm 6 MPHE.Dec($\{pd_i\}_{i=0}^{N-1}, ct$) $\rightarrow m'$

1: Let $ct = (c_0, c_1)$

2: Compute

$$c_0 + c_1 s = c_0 + \sum_{i=0}^{N-1} p d_i = c_0 + c_1 (\sum_{i=0}^{N-1} s_i) + \sum_{i=0}^{N-1} e_i$$

3: Output $m' = [\lfloor \frac{t}{q} [c_0 + c_1 s]_q \rceil]_t$

about bit b and the receiver does not learn anything about the message m_{1-b} .

4) Garbled Circuits: A Garbled Circuit scheme [2] [33] is a cryptographic tool involving two parties joint computing a function $C(x_1, x_2)$ where x_1 and x_2 are inputs provided by the garbler and the evaluator. The scheme should keep the inputs fully private. It is consists of a tuple of algorithms GC=(GC.Garble, GC.Transfer, GC.Eval) with the following syntax:

- GC.Garble(Params, $C) \rightarrow (\tilde{C}, \text{label})$: Used by the garbler. Assume C is composed of G gates (e.g., XOR, AND, etc.) in total. The garbler garbles the circuit by 1) Assigning two random k-bit labels to each wire in the circuit C corresponding to 0 and 1 where k is the security parameter usually set to 128; 2) For each gate $g,g \in [G]$ in C, computing a garbled table \hat{C}_q with 4 rows corresponding to 4 combinations of inputs labels. Each row is the encryption of the output with two corresponding input labels as the encryption key [34]. Output $\tilde{C} = \{\tilde{C}_g\}_{g=0}^{G-1}$ and labels corresponding to input wires label.
- GC.Transfer(label, x_i) \rightarrow label_{x_i} : This function involves both the garbler and the evaluator. If i = 2, for input x_2 held by the garbler, the garbler maps x_2 with the labels generated for input wire corresponding to x_2 to obtain $|abe|_{x_2}$, then sends $|abe|_{x_2}$ to the evaluator. If i = 1, for input x_1 held by the evaluator, the evaluator cannot simply send x_1 to the garbler to obtain $|abe|_{x_1}$. This undermines input privacy. The garbler and the evaluator engage in an OT protocol where the garbler provides labels for inputs wires belonging to the evaluator and the garbler provides the bit-expression of x_1 . The evaluator will receive the labels corresponding to x_1 without exposing x_1 to the garbler via OT. This step outputs the correct labels $|abe|_{x_i}$ corresponding to actual input x_i .
- $GC.Eval(C, \{label_{x_i}\}_{i \in \{1,2\}}) \rightarrow y$: Used by the evaluator. The evaluator starts from the first gate and uses two input labels with the garbled table \tilde{C}_0 to obtain the output. Then the evaluator performs the evaluation for each gate in order until obtaining the output y of the last gate, where $y = C(x_1, x_2)$.

A two-party Garbled Circuit scheme can be easily extended to a multiparty computations scheme of arbitrary complexity with an arbitrary number of participants providing inputs. The input providers obtain the labels for their private inputs via OT from the garbler, then they transmit the labels to the evaluator, which reveals nothing about the inputs since the labels are random.

5) Additive Secret Share: Let q be a prime. A 2-of-2 additive secret sharing of $x \in \mathbb{Z}_q$ is a pair $(\langle x \rangle_1, \langle x \rangle_2) = (x-r,r) \in \mathbb{Z}_q^2$ for a random $r \in \mathbb{Z}_q$ such that $x = \langle x \rangle_1 + \langle x \rangle_2$. Additive secret sharing is perfectly hidden, for example, given a share $\langle x \rangle_1$ or $\langle x \rangle_2$, the value x is perfectly hidden. n-ofn additive secret sharing can be easily extended from 2-of-2 additive secret sharing. A n-of-n additive secret sharing of $x \in \mathbb{Z}_q$ can be constructed by generating $(r_1, ..., r_{n-1}) \in \mathbb{Z}_q^{n-1}$ uniformly at random and setting $r_n = x - \sum_{i=1}^{n-1} r_i \pmod{q}$.

C. Prior Work: DELPHI

Here we describe the protocol we built upon: DELPHI [8]. DELPHI is a cryptographic neural network inference protocol. There are two parties in the system of DELPHI: the user and the server. DELPHI uses additive secret sharing pre-built with homomorphic encryption to evaluate linear layers, and garbled circuits pre-built to evaluate activation functions. Let $\mathbf{F} = {\mathbf{F}_i}_{i=1}^L$ described in Section III-A be the deployed model on the server, $\mathbf{x}_i = \mathbf{F}_{i-1}(\mathbf{F}_{i-2}(...(\mathbf{F}_1(\mathbf{x}_1))))$ is the the intermediate prediction result of the first i - 1 layers where $\mathbf{x}_1 \in \mathbb{Z}_q$ is the input held by the user.

In the preprocessing phase, the client and the server precompute the secret share which will be used for the inference phase. For each linear layer $i \in [L]$, with the user providing the ciphertext of \mathbf{r}_i and the server providing the plaintext of \mathbf{F}_i and s_i , two parties interact to compute $F_i r_i - s_i$ where r_i and s_i are the random masks for building secret shares, resulting in each of the party holding one of the two secret shares of $\mathbf{F}_i \mathbf{r}_i$. For ReLUs following linear layers, the server constructs a garbled circuit \hat{C}_i based on circuit C computing ReLU function. Then the server transmits \hat{C}_i to the user and input labels of $\mathbf{F}_i \mathbf{r}_i - \mathbf{s}_i$ and \mathbf{r}_{i+1} via Oblivious Transfer to the user. We note that the algorithm for computing the ciphertext of $\mathbf{F}_i \mathbf{r}_i$ in DELPHI relies on GAZELLE's design [7] for HE linear operations (convolutional and fully-connected layer). We do not provide details of this algorithm but represent it as a function Lin-OP. With function Lin-OP, ciphertext $ct_{\mathbf{r}_i}$ encrypting \mathbf{r}_i and plaintext pt_{F_i} encoding \mathbf{F}_i as input, HE.Eval will output the ciphertext of $\mathbf{F}_i \mathbf{r}_i$.

In the inference phase, assuming \mathbf{x}_i is the prediction result of the first i - 1 layers, at the beginning of the i^{th} layer, the server and the user hold the shares of \mathbf{x}_i : $\mathbf{x}_i - \mathbf{r}_i$ and \mathbf{r}_i . The server evaluates to get $\mathbf{F}_i(\mathbf{x}_i - \mathbf{r}_i) + \mathbf{s}_i$, resulting in the server and the user holding the secret shares of $\mathbf{F}_i \mathbf{x}_i$. For evaluating ReLU, the server transmits the input labels corresponding to its secret share to the user. Then the client evaluates \tilde{C}_i to obtain a secret share of the ReLU output.

IV. SECURE INFERENCE: HIERARCHICAL SERVER CONFIGURATIONS

Our proposed setup involves splitting the neural network model across three server nodes. We assume that the user communicates with a gateway server that holds part of the neural network model. The remainder of the model is stored on remote servers that do not directly communicate with the user. One motivation for considering such a scenario is that the interactions of the user and the server are required in computing each layer in the DELPHI protocol. This increases the computational and communication load on the user. Furthermore, the user gets access to information about the neural network including the shape of each layer and the total number of model layers since the user needs to generate randomness for each linear layer to build up additive secret shares, which is undesirable. By deploying a subset of the model on remote servers that do not directly interact with the user we can partially address these concerns.

A. Baseline Configuration

We now explain why a straightforward extension of the DELPHI protocol that splits the neural network model across two server nodes as shown in Fig. 2 will lead to information leakage. In this setting, the user and the gateway server as well as the gateway server and the remote server respectively execute DELPHI to evaluate the two split models in order as Figure 2 shows. In this case, the user can be offline when two servers evaluate the second split model and the architecture of the second split model can be kept secret from the user.

Let us consider deploying \mathbf{F}_I , \mathbf{F}_{II} (described in Section III-A) on server A and B respectively. When evaluating \mathbf{F}_{II} involving server A and B, server B holds the masked intermediate prediction result $\mathbf{x}_i - \mathbf{r}_i$ while randomness \mathbf{r}_i is only contributed by server A. Note that if the two servers collude to recover \mathbf{x}_i they can get access to full parameters of the first i - 1 layers, then execute the white-box model inversion attack proposed in [35] to reconstruct the user's input.



Fig. 2. DELPHI extended to Split Neural Network

B. Proposed Scheme

We consider a setting with a hierarchy of servers where there are a gateway server and two remote servers as Figure 1 shows. We deploy \mathbf{F}_I on the gateway server A and the processing of it follows DELPHI. For the second split model \mathbf{F}_{II} , the linear layers are randomly divided into 2 shares. We distribute the shares among 2 remote servers. We summarize the main advantages of our proposed architecture as follows: 1) hide partial architecture information of the model from the user and reduce the user's computation and communication cost 2) defend the possible white-box model inversion attack resulting from the model leakage and intermediate prediction result recovery caused by server collusion. We give the detailed description of the protocol in Section VI.

V. SYSTEM OVERVIEW

As shown in Figure 1, there are four parties in our setting: a user, and hierarchical service providers including server A, B, and C. The hierarchy contains two layers of servers: gateway server A in the first layer, and server B and C in the second layer. Three servers cooperate to provide prediction service of a model **F**. The model **F** can be expressed as $\mathbf{F} = (\mathbf{F}_A : \{\mathbf{F}_i\}_{i \in \{1,...,l\}}, \mathbf{F}_{BC} : \{\mathbf{F}_i\}_{i \in \{l+1,...,L\}})$ where L is the total number of model layers. Server A holds the first l layers \mathbf{F}_A . For *i*th layer $i \in \{l + 1, ..., L\}$, Server B and C hold the random share of linear layer \mathbf{F}_i , which can be expressed as $\mathbf{F}_i = \sum_{j=2}^{3} \mathbf{F}_i^j$ (j = 2 for server B, j = 3 for server C).

A. Threat Model and Privacy Goals

a) Threat model.: We assume a passive-adversary model with corruption of a user or up to two servers (any two). The parties will not deviate from the protocol but the adversary can corrupt the user or up to 2 servers so can know corrupted parties' private data and the observations. We assume the adversary cannot corrupt the user and the server simultaneously. Additionally, we assume server A belongs to the service provider while server B and C are external. Deployment of random shares on external servers won't compromise the proprietary model privacy.

b) Privacy Goals.: For the user, we aim to design a protocol that enables the user to only learn the result of the inference and the architecture of the model on gateway server A. All other information about the model including the model parameters, the total number of layers, and the architecture of the layers on server B, C should be kept secret from the user.

Assuming up to 2 servers corrupted by the adversary, we have the privacy goals for the servers as the followings: 1) the adversary should not learn any honest party's model parameters or the intermediate prediction result \mathbf{x}_i ; 2) the adversary should not infer the user's input.

VI. THE PROTOCOL: FORMAL DESCRIPTION

In this section, we introduce our secure inference protocol. The protocol takes as input the user's data $\mathbf{x}_1 \in \mathbb{Z}_q$ and the split model ($\mathbf{F}_A = {\mathbf{F}_i}_{i \in {1,...l}}, \mathbf{F}_{BC} = {\mathbf{F}_i}_{i \in {l+1,...L}}$), and enables four parties to execute secure inference collaboratively. The protocol can be considered as the combination of a 2-PC protocol, executed between the user and the gateway server for the inference of \mathbf{F}_A , and a 3-PC protocol, executed between the gateway server sfor the inference of \mathbf{F}_{BC} . The 2-PC protocol is adapted from DELPHI [8] and the 3-PC protocol is designed to achieve the secure neural network computations in the presence of dishonest majority of honest-but-curious servers. The protocol consists of three phases as shown in Figure 3: a setup phase, a preprocessing phase, and an online inference phase.



Fig. 3. SECO's 3 phases

A. Setup Phase

The setup phase is executed before the user appears in the system.

1) Key Generation: This step is to generate keys for homomorphic computation.

- 1) Each server runs HE.KeyGen to generate key pair $(\mathsf{sk}_j,\mathsf{pk}_j)$ (j = 0 for the user, j = 1 for server A, j = 2 for server B, j = 3 for server C).
- Server A collects the public keys from the three servers, runs MPHE.DKeyGen to obtain a common public key cpk, then transmits cpk to the user, server B and C.

At the end of the step, each server holds a common public key cpk.

2) Model Preparation: : This step is the preparation for \mathbf{F}_{BC} which can be executed before the user appears in the system. Like in DELPHI, the key insight is to pre-compute the additive secret shares of $\mathbf{F}_i \mathbf{r}_i$ for each linear layer of \mathbf{F}_{BC} . Server A, B, and C collaboratively compute the secret shares of $\mathbf{F}_i \mathbf{r}_i$ with MPHE scheme, where \mathbf{F}_i is distributed among server B and C and the randomness \mathbf{r}_i is solely contributed by the user if i = l + 1 or jointly contributed by three servers if $i \in \{l+2, ..., L\}$. The preprocessing of the $l + 1^{\text{th}}$ linear layer is required for a secure transition from the inference of \mathbf{F}_A to \mathbf{F}_{BC} . Additionally, three servers construct garbled circuits for each activation layer in \mathbf{F}_{BC} .

a) Preprocessing for other linear layers.: For the i^{th} layer $i \in \{l + 2, ...L\}$, server A, B, and C contribute to the randomness \mathbf{r}_i and collaborate to compute three secret shares of $(\mathbf{F}_{l+1}^2 + \mathbf{F}_{l+1}^3)\mathbf{r}_{l+1}$. With server B and C providing the ciphertexts of private model parameter \mathbf{F}_i^j , input randomness \mathbf{r}_i^j and output randomness \mathbf{s}_i^j , server A computes its share $\mathbf{E}_i^1 = (\mathbf{F}_i^2 + \mathbf{F}_i^3)(\mathbf{r}_i^1 + \mathbf{r}_i^2 + \mathbf{r}_i^3) - \mathbf{s}_i^2 - \mathbf{s}_i^3$, resulting in building

the additive shares of $(\mathbf{F}_i^2 + \mathbf{F}_i^3)(\mathbf{r}_i^1 + \mathbf{r}_i^2 + \mathbf{r}_i^3)$ among three servers. See Algorithm 8 for details. In Algorithm 8, we use MPHE.DisDec (Algorithm 7) which is a protocol for the distribute decryption of MPHE scheme involving three servers.

b) Preprocessing for activation layers: Three servers collaborate to build the garbled circuit on server A and prepare the evaluation of the activation layers. With server B as the garbler, server C as the evaluator, and server A providing input as the third party, server B first prepares the garbled circuit \tilde{C}_i and labels by running GC.Garble(Params, C_i) where Params is the security parameters and C_i is described in Algorithm 9. To transmit the labels corresponding to actual input values, server B sends the labels of the input value provided by itself to server C through a public channel. For input values from server A and C, server A and C first obtain the actual labels from server B via OT. Then server C collects the labels obtained by server A through a public channel.

B. Preprocessing Phase

In the preprocessing phase, the user first generates a key pair $(\mathbf{sk}_0, \mathbf{pk}_0)$ and broadcasts \mathbf{pk}_0 to each server in the system through Server A. Server A sends common public key \mathbf{cpk} to the user. For preprocessing of \mathbf{F}_A , the user and server A collaboratively pre-compute the secret shares of weights for linear layers and construct garbled circuits for activation layers.

1) Preprocessing for \mathbf{F}_A : This step follows the approach in DELPHI.

a) Preprocessing for linear layers.: For the *i*th linear layer $i \in [1, ..., l]$, with the user providing the ciphertext of \mathbf{r}_i , server A computes the ciphertext of $\mathbf{F}_i \mathbf{r}_i - \mathbf{s}_i$ where \mathbf{r}_i is the randomness of the layer's input size and \mathbf{s}_i is the randomness of the layer's decrypts it using \mathbf{sk}_0 , resulting in the user holding $\mathbf{F}_i \mathbf{r}_i - \mathbf{s}_i$ and server A holding \mathbf{s}_i which are two additive shares of $\mathbf{F}_i \mathbf{r}_i$.

b) Preprocessing for activation layers.: After the preprocessing for linear layers, the user and server A collaborate to build the garbled circuits for activation layers in \mathbf{F}_A . C_i is the circuit to compute the activation function. Server A will execute **GC.Garble** to generate a set of the garbled tables \tilde{C}_i according to gates in circuit C_i and labels corresponding to all input wires in C_i . Then server A transmits the labels for actual input values to the user. For input values held by server A such as one-time password \mathbf{d}_{i+1} , server A generates labels based on the bit expression of \mathbf{d}_{i+1} and directly sends it to the user. For input values held by the user such as $\mathbf{F}_i \mathbf{r}_i - \mathbf{s}_i, \mathbf{r}_{i+1}$, server A and the user engage in OT protocol to let the user obtain the corresponding labels without server A knowing the actual value.

c) Preprocessing for the $l+1^{th}$ linear layer.: In this step, the protocol involves 4 parties to prepare for the transition from the inference of \mathbf{F}_A to \mathbf{F}_{BC} . The user, server A, B, and C collaborate to build the additive shares of $(\mathbf{F}_{l+1}^2 + \mathbf{F}_{l+1}^3)\mathbf{r}_{l+1}$ among three servers, that is, with server B and C providing the ciphertexts of their own \mathbf{F}_{l+1}^j and the ciphertxts of the output randomness \mathbf{s}_{i+1}^j , with the user providing the ciphertext of

Algorithm 7 MPHE.DisDec

Inputs: *ct*: the ciphertext of *m*; sk_j : the plain secret key generated by the contributor of cpk **Server A**: Transmit *ct* to server B and C MPHE.Reconstruct(*ct*, sk_1) $\rightarrow pd_1$ **Server B and C**: MPHE.Reconstruct(*ct*, sk_j) $\rightarrow pd_j$ Transmit pd_j to server A **Server A**: Run MPHE.Dec(*ct*, { pd_j }_{{ $j \in \{1,2,3\}\}}) to obtain$ *m*</sub>

input randomness \mathbf{r}_{l+1} , server A homomorphically computes the ciphertext of $\mathbf{E}_{l+1} = (\mathbf{F}_{l+1}^2 + \mathbf{F}_{l+1}^3)\mathbf{r}_{l+1} - \mathbf{s}_{l+1}^2 - \mathbf{s}_{l+1}^3$ and then involves server B and C to decrypt it. At the end of the preprocessing of the $l + 1^{\text{th}}$ linear layer, three server holds the respective secret share of $(\mathbf{F}_{l+1}^2 + \mathbf{F}_{l+1}^3)\mathbf{r}_{l+1}$. Note all the encryption in this stage are conducted with MPHE public key cpk.

Algorithm 8 Pre-Compute the share for the i^{th} linear layer, $i \in \{l+2, ...L\}$

Inputs: (q_i, w_i) : the input and output shape of the *i*th layer; cpk: common public key Server A: $\mathbf{r}_i^1 \leftarrow \mathbb{Z}_a^{q_i}$ Compute $ct_{r_i^1} = \mathsf{MPHE}.\mathsf{Enc}(\mathsf{cpk},\mathbf{r}_i^1)$ Server B and C: $\begin{array}{lll} \mathbf{r}_{i}^{j} \leftarrow \mathbb{Z}_{q}^{q_{i}}, \, \mathbf{s}_{i}^{j} \leftarrow \mathbb{Z}_{q}^{w_{i}} \\ \text{Compute} \quad ct_{r_{i}^{j}} & = & \mathsf{MPHE}.\mathsf{Enc}(\mathsf{cpk}, \mathbf{r}_{i}^{j}), \quad ct_{s_{i}^{j}} \end{array}$ $\mathsf{MPHE}.\mathsf{Enc}(\mathsf{cpk},\mathbf{s}_{i}^{j}),\ ct_{F^{j}}=\mathsf{MPHE}.\mathsf{Enc}(\mathsf{cpk},\mathbf{F}_{i}^{j})$ Transmit $(ct_{r_i^j}, ct_{s_i^j}, ct_{F_i^j})$ to server A Server A: MPHE.Eval $(ct_{F_i^2}, ct_{F_i^3}, Add) \rightarrow ct_{F_i}$ $\mathsf{MPHE}.\mathsf{Eval}(ct_{s_i^2}, ct_{s_i^3}, \mathsf{Add}) \rightarrow ct_{s_i}$ MPHE.Eval $(ct_{r_i}, ct_{r_i}, ct_{r_i}, Add) \rightarrow ct_{r_i}$ MPHE.Eval $(ct_{F_i}, ct_{r_i}, Lin-OP) \rightarrow ct_{F_ir_i}$ $\mathsf{MPHE}.\mathsf{Eval}(ct_{F_ir_i}, ct_{s_i}, \mathsf{Sub}) \to ct_{F_ir_i - s_i}$ Transmit $ct_{F_ir_i-s_i}$ to server B and C Server A, B, C: Run MPHE.DisDec $(ct_{F_ir_i-s_i}, \{sk_j\}_{j \in \{1,2,3\}})$ Server A obtains $\mathbf{E}_i = (\mathbf{F}_i^2 + \mathbf{F}_i^3)\mathbf{r}_i - \mathbf{s}_i^2 - \mathbf{s}_i^3$ where $\mathbf{r}_i =$ $r_{i}^{1} + r_{i}^{2} + r_{i}^{3}$

C. Online Inference Phase

This section introduces the inference phase in the order of model layers. First, as preamble, the user computes $x_1 - r_1$ and sends it to server A.

a) Inference of \mathbf{F}_A : The inference of \mathbf{F}_A involves the participation of the user and server A. Before evaluating the $i^{\text{th}}, i \in [1, ..., l]$ layer, server A holds the masked input $\mathbf{x}_i - \mathbf{r}_i$ and the user holds the input randomness \mathbf{r}_{i+1} for next layer. Server A first computes $\mathbf{F}_i(\mathbf{x}_i - \mathbf{r}_i) + \mathbf{s}_i$ to evaluate the linear part and sends the labels of it to the user. Collecting the labels of all the input wires in \tilde{C}_i , the user executes GC.Eval to obtain a masked ReLU output as the masked input for the next layer.

Algorithm 9 A circuit C_i that computes ReLU function of the i^{th} layer $i \in \{l + 1, ..., L\}$

	• • • • • • • •
	Input from server A: $(\mathbf{F}_i^2 + \mathbf{F}_i^3)\mathbf{r}_i - \mathbf{s}_i^2 - \mathbf{s}_i^3, \mathbf{r}_{i+1}^1$
	Input from server B: $\mathbf{F}_i^2(\mathbf{x}_i - \mathbf{r}_i) + \mathbf{s}_i^2, \mathbf{r}_{i+1}^2$
	Input from server C: $\mathbf{F}_i^3(\mathbf{x}_i - \mathbf{r}_i) + \mathbf{s}_i^3, \mathbf{r}_{i+1}^3$
1:	Compute $(\mathbf{F}_i^2 + \mathbf{F}_i^3)\mathbf{x}_i = (\mathbf{F}_i^2 + \mathbf{F}_i^3)\mathbf{r}_i - \mathbf{s}_i^2 - \mathbf{s}_i^3 + \mathbf{F}_i^2(\mathbf{x}_i - \mathbf{x}_i^2)\mathbf{r}_i - \mathbf{s}_i^2 - \mathbf{s}_i^3 - \mathbf{s}_i^2 - \mathbf{s}_i^3 + \mathbf{F}_i^2(\mathbf{x}_i - \mathbf{x}_i^2)\mathbf{r}_i - \mathbf{s}_i^2 -$
	$\mathbf{r}_i) + \mathbf{s}_i^2 + \mathbf{F}_i^3(\mathbf{x}_i - \mathbf{r}_i) + \mathbf{s}_i^3$
2:	Compute $\mathbf{x}_{i+1} = ReLU((\mathbf{F}_i^2 + \mathbf{F}_i^3)\mathbf{x}_i)$
3:	Output $\mathbf{x}_{i+1} - \mathbf{r}_{i+1} = \mathbf{x}_{i+1} - \mathbf{r}_{i+1}^1 - \mathbf{r}_{i+1}^2 - \mathbf{r}_{i+1}^3$

b) Transition: After the evaluation of \mathbf{F}_A , server A holds $\mathbf{x}_{l+1} - \mathbf{r}_{l+1}$. Server A forwards it to server B and C as a transition step from the inference of \mathbf{F}_A to \mathbf{F}_{BC} .

c) Inference of \mathbf{F}_{BC} : The inference of \mathbf{F}_{BC} involves the participation of server A, B, and C. At the beginning of the inference for i^{th} layer $i \in \{l + 1, ..., L\}$, server B and C hold $\mathbf{x}_i - \mathbf{r}_i$ where \mathbf{r}_i is solely contributed by the user when i = l + 1, and is contributed by server A, B and C for linear other layers of \mathbf{F}_{BC} . Server A, B and C collaborate to do inference on i^{th} layer $i \in \{l + 1, ..., L\}$. First server B and C will compute $\mathbf{E}_{i}^{j} = \mathbf{F}_{i}^{j}(\mathbf{x}_{i} - \mathbf{r}_{i}) + \mathbf{s}_{i}^{j}$, resulting in three servers holding three additive shares of the real prediction result for the current layer $(\mathbf{F}_i^2 + \mathbf{F}_i^3)\mathbf{x}_i$. The garbled circuit \tilde{C}_i (Algorithm 9) first aggregates shares to recover $(\mathbf{F}_i^2 + \mathbf{F}_i^3)\mathbf{x}_i$, then computes ReLU of it, finally masks the ReLU output with the randomness of the next layer contributed by three servers. The entire process is in the encrypted domain. The garbled circuit for each ReLU layer of \mathbf{F}_{BC} are located at server C. Thus server C can evaluate the garbled circuit when collecting all labels for input wires of \hat{C}_i in a secure way.

Algorithm 10 Inference for the i^{th} layer, $i \in \{l + 1, ..., L\}$

Inputs: $\mathbf{x}_i - \mathbf{r}_i$: held by server B and C; \tilde{C}_i : the garbled circuit of circuit in Algorithm 9; label: random labels for input wires of Algorithm 9; \mathbf{r}_{i+1}^1 : randomness from server A; label_{E_i^1}, label_{r_{i+1}^2}, label_{r_{i+1}^3}: labels for actual input values held by server A **Server B and C**: Compute $\mathbf{E}_i^j = \mathbf{F}_i^j(\mathbf{x}_i - \mathbf{r}_i) + \mathbf{s}_i^j$ GC.Transfer(label, \mathbf{E}_i^3) \rightarrow label_{E_i^3} Server C obtains label_{E_i^3} **Server C**: GC.Eval(\tilde{C}_i , label_{E_i^2}, label_{E_i^3}, label_{E_i^1}, label_{r_{i+1}^1}, label_{r_{i+1}^2}, label_{r_{i+1}^2} \cap **x**_{i+1} - $\mathbf{r}_{i+1}^i - \mathbf{r}_{i+1}^2$ Compute $\mathbf{x}_{i+1} - \mathbf{r}_{i+1} = \mathbf{x}_{i+1} - \mathbf{r}_{i+1}^1 - \mathbf{r}_{i+1}^2 - \mathbf{r}_{i+1}^3$ Transmit $\mathbf{x}_{i+1} - \mathbf{r}_{i+1}$ to server B

D. Output

After the evaluation of \mathbf{F}_{BC} , server A holds the share $(\mathbf{F}_L^2 + \mathbf{F}_L^3)\mathbf{r}_L - \mathbf{s}_L^2 - \mathbf{s}_L^3$, server B holds $\mathbf{F}_L^2(\mathbf{x}_L - \mathbf{r}_L) + \mathbf{s}_L^2$ and server C holds $\mathbf{F}_L^3(\mathbf{x}_L - \mathbf{r}_L) + \mathbf{s}_L^3$.

- 1) Server B and C computes $\text{HE}.\text{Enc}(\text{pk}_0, \mathbf{F}_L^i(\mathbf{x}_L \mathbf{r}_L) + \mathbf{s}_L^i)$ and sends this ciphertext to server A.
- 2) Server A homomorphically add the received ciphertext to compute the ciphertexts $ct_{\hat{\mathbf{y}}}$ of prediction result $\hat{\mathbf{y}}$ where $\hat{\mathbf{y}} = (F_L^2 + F_L^3)\mathbf{x}_L$, then transmits $ct_{\hat{\mathbf{y}}}$ to the user.

TABLE I

THE COLUMNS DEPICT THE THREE DISTINCT SEGMENTS OF THE DEPLOYED MODEL, EACH ILLUSTRATING CONTRIBUTIONS FROM THE INVOLVED PARTIES AND DIFFERENT COMPUTATIONAL GOALS.

	\mathbf{F}_A	\mathbf{F}_{BC}	Transition
User	\mathbf{r}_i		\mathbf{r}_i
Server A	$\mathbf{F}_i, \mathbf{s}_i$	\mathbf{r}_i^1	
Server B		$\mathbf{r}_i^2, \mathbf{F}_i^2, \mathbf{s}_i^2$	$\mathbf{F}_i^2, \mathbf{s}_i^2$
Server C		$\mathbf{r}_i^3, \mathbf{F}_i^3, \mathbf{s}_i^3$	$\mathbf{F}_{i}^{3}, \mathbf{s}_{i}^{3}$
Shares to compute	$\mathbf{F}_i \mathbf{r}_i - \mathbf{s}_i$	$\begin{array}{c} (\mathbf{F}_{i}^{2}+\mathbf{F}_{i}^{3})(\mathbf{r}_{i}^{1}+\mathbf{r}_{i}^{2}+\mathbf{r}_{i}^{3})\\ -(\mathbf{s}_{i}^{2}+\mathbf{s}_{i}^{3}) \end{array}$	$\begin{array}{c} (\mathbf{F}_i^2+\mathbf{F}_i^3)\mathbf{r}_i\\ -(\mathbf{s}_i^2+\mathbf{s}_i^3) \end{array}$

3) The user decrypts $ct_{\hat{\mathbf{y}}}$ with sk_0 to obtain the prediction result $\hat{\mathbf{y}}$.

E. Discussion

We next highlight key technical contributions and the motivations for our design choices.

a) Secret Sharing for 3PC and Integration with MPHE: To sum up, during the executions, the protocol processes model \mathbf{F}_A and \mathbf{F}_{BC} . The processing of \mathbf{F}_A follows DELPHI which uses plain HE scheme to build two secret shares of $\mathbf{F}_i \mathbf{r}_i$ among two parties in the preprocessing phase, where \mathbf{r}_i is randomly generated by the user; In the inference phase, \mathbf{r}_i will be used as the mask for the intermediate prediction result \mathbf{x}_i and it can be canceled by the secret share built in the preprocessing phase. Our protocol follows this idea to build three secret shares of $\{\mathbf{F}_i \mathbf{r}_i\}_{i=l+1}^L$ in the preprocessing phase of \mathbf{F}_{BC} , where each of the servers contributes its own part to the randomness \mathbf{r}_i . However, using a plain HE scheme to compute the secret shares is not privacy-preserving in this case because the model parameters and randomness are distributed. To ensure that each of the three servers exclusively accesses its designated information and no extraneous data, we design a new secret sharing scheme as in Table I, so the shares can be computed with MPHE using the supported operations as described in Section III-B1, enabling secure computation of linear functions.

b) Assignments of Servers in GC: For the secure computation of non-linear functions, we deviate from the conventional approach of assigning the layer's input provider as the evaluator in Garbled Circuits (GC), like in DELPHI. Instead, we appoint a remote server as the evaluator and another as the garbler. This approach substantially diminishes the communication complexity during the inference phase, as communication now occurs exclusively between these two remote servers. Additionally, our protocol maintains the security of user's input even in the event of two remote servers being compromised, because of the gateway server's contribution to the masking term during the setup phase. Last but not the least, we design a transition from user-server interaction to server-server interaction to ensure no information leakage in this process.

VII. INFORMATION LEAKAGE ANALYSIS

In this section, we show the protocol described in Section VI achieves the privacy goals described in Section V-A under a passive-adversary model. Based on Composition Theorem for semi-honest model (Theorem 7.3.3 in [36]), by proving the security of the protocol in each phase, the security of the entire protocol is proven.

We prove the security of the preprocessing and inference phase following the simulation paradigm [37]. According to the simulation paradigm, a protocol is secure if whatever can be computed by a party involved in the protocol can be computed given only the input and the output of the party. So the security can be modeled by defining a simulator that can generate the view of the party during the procedure given the input and the output. If the view of the party can be simulated based on the input and the output only and is computationally indistinguishable from the real view of the party, it implies the party only learns what can be computed given the input and the output, so the protocol is secure.

A. Setup phase

The setup phase is independent of the rest of the protocol. For a given user and the servers, it has to be run only once. The protocol used in the setup phase is a composition of HE.KeyGen(.) and MPHE.DKeyGen(.).

In Mouchet et al. [30], it shows that the protocol in the setup phase can securely and correctly generate a valid plain BFV key pair. Provided with valid plain BFV key pair, HE.Enc and MPHE.Enc can output valid BFV ciphertext, which guarantees the semantic security of HE and MPHE schemes described in Proposition VII.1 and Proposition VII.2. The semantic security of HE and MPHE schemes provides the security basis for the following two phases.

Proposition VII.1 (HE semantic security). For any two messages m_1, m_2 , no adversary has an advantage (better than 1/2 chance) in distinguishing between distributions $HE.Enc(pk, m_1)$ and $HE.Enc(pk, m_2)$.

Proposition VII.2 (MPHE-based MPC semantic security). For any subsets of at most colluded N - 1 clients, for any two messages m_1 and m_2 , no adversary has an advantage (better than 1/2 chance) in distinguishing between distributions MPHE.Enc(cpk, m_1) and MPHE.Enc(cpk, m_2).

B. Preprocessing and Inference Phase

In this section, we prove the privacy goals in Section V-A are achieved in the preprocessing and inference phase following the simulation paradigm [37].

We can prove the security of the preprocessing and inference phase by 1) providing the real view of the adversary for the case when the user or two servers are corrupted 2) describing simulators for cases where two servers or the user are corrupted 3) comparing each term in the view of the adversary and proving the indistinguishability. We have three privacy goals that are to preserve the privacy of 1) the user's input 2) the intermediate prediction result 3) the complete model

parameters. Intuitively, by distributing the model parameters, the honest server's partial model parameters are protected thus the adversary cannot obtain the complete model. The user's input and the intermediate prediction result are protected by the randomness. To prove they are fully secure, we can determine the indistinguishability of each element in the view separately. Note Composition Theorem [36] can be applied to the sequential compositions of arbitrary protocols involving multiple parties. Based on Composition Theorem [36], the security of the protocols for two phases is proven when the composing sub-protocols are secure. The security proof of cryptographic sub-protocols (section III-B) are provided in [30][38][39] assuming the passive adversary model.

We name the protocols of two phases as PRE and INF and define the corresponding ideal functionality as

$$f_{pre}(\mathsf{pk}_{0}, \mathsf{cpk}, \{C_{i}\}_{i=1}^{L}, \{\mathbf{F}_{i}\}_{i=1}^{l}, \{\mathbf{F}_{i}\}_{i=l+1, j=2, 3}^{i=L}) = \{\{\mathsf{share}_{i}^{U}\}_{i=1}^{l}, \{\mathsf{share}_{i}^{A}\}_{i=l+1}^{L}, \{\tilde{C}_{i}\}_{i=1}^{L}\}$$
(1)
$$f_{inf}(\mathbf{x}_{1}, \{\mathsf{share}_{i}^{U}\}_{i=1}^{l}, \{\mathsf{share}_{i}^{A}\}_{i=l+1}^{L}, \{\tilde{C}_{i}\}_{i=1}^{L})$$

$$= \{\{\mathsf{msk}_i\}_{i=1}^L\}$$
(2)

where $\{\operatorname{share}_{i}^{U} = \mathbf{F}_{i}\mathbf{r}_{i} - \mathbf{s}_{i}\}_{i=1}^{l}, \{\operatorname{share}_{i}^{A} = (\mathbf{F}_{i}^{2} + \mathbf{F}_{i}^{3})(\mathbf{r}_{i}^{1} + \mathbf{r}_{i}^{2} + \mathbf{r}_{i}^{3}) - \mathbf{s}_{i}^{2} - \mathbf{s}_{i}^{3}\}_{i=l+1}^{L}, \{\operatorname{msk}_{i} = \mathbf{x}_{i} - \mathbf{r}_{i}\}_{i=1}^{L}.$ Next we we discuss the cases where server A and B are corrupted and the user is corrupted. The proofs for other cases, wherein either servers B and C or servers A and C are corrupted, can be readily extended based on the methodology outlined in the following proof.

a) Server A, B corrupted: The real view v_{AB} of the adversary corrupting server A and B includes:

- 1) Ciphertexts set $ct_{S_{AB}} = \{\{ct_{r_i}\}_{i=1}^{l+1}, \{ct_{r_i^2}, ct_{r_i^3}\}_{i=l+2}^{L}, \{ct_{F_i^2}, ct_{F_i^3}, ct_{s_i^2}, ct_{s_i^2}\}_{i=l+2}^{L}, \{ct_{\mathsf{share}_i^A}\}_{i=l+1}^{L}\}$ 2) The masked input $\{\mathsf{msk}_i = \mathbf{x}_i \mathbf{r}_i\}_{i=1}^{l+1}, \{\mathsf{msk}_i = \mathbf{x}_i \mathbf{x}_i\}_{i=1}^{L+1}\}$
- $\mathbf{r}_{i}^{3}\}_{l+2}^{L+1}$
- 3) The secret shares set share ${}^{A}_{i} = (\mathbf{F}_{i}^{2} + \mathbf{F}_{i}^{3})\mathbf{r}_{i} \mathbf{s}_{i}^{2} \mathbf{s}_{i}^{3}, i \in$ $\{l+1, ..., L\}$
- 4) Garbled circuit $\{\tilde{C}_i\}_{i=1}^L$ and labels $\{|abel_i\}_{i=l+1}^L$

We define a simulator S_{AB} that simulates the view of an adversary corrupting server A and B. For a given value x, we denote the simulated equivalent by \tilde{x} and the encryption of it by ct_x . S_{AB} is given $\mathsf{pk}_0, \mathsf{cpk}, \mathbf{F}_A, \{\mathbf{F}_i^2\}_{i=l+1}^L, (q_i, w_i)_{i=1}^L$ and proceeds as follows:

- 1) S_{AB} chooses a uniform random tape for server A and B
- 2) In the preprocessing phase:
 - a) S_{AB} , as the user, computes $\{ct_{r_i}\}$ = HE.Enc($pk_0, \tilde{\mathbf{r}}_i$) $\}_{i=1}^{l+1}$ and sends it to server A where $\tilde{\mathbf{r}}_i \leftarrow \mathbb{Z}_q^{q_i}$. Then S_{AB} receives the evaluated ciphertext from server A.
 - b) With S_{AB} as the evaluator of the garbled circuits for $i^{\text{th}}, i \in \{1, ..., l\}$ non-linear layer, S_{AB} receives \tilde{C}_i from server A and labels corresponding to random input generated by S_{AB} .

c) Server B computes and sends the ciphertexts

$$ct_{F_i^2} = \mathsf{MPHE}.\mathsf{Enc}(\mathsf{cpk}, \mathbf{F}_i^2)$$

$$ct_{r_i^2} = \mathsf{MPHE}.\mathsf{Enc}(\mathsf{cpk}, \mathbf{r}_i^2)$$

$$ct_{s_i^2} = \mathsf{MPHE}.\mathsf{Enc}(\mathsf{cpk}, \mathbf{s}_i^2)$$

to server A.

- d) S_{AB} , as server C, sets $\tilde{\mathbf{F}}_{i}^{3} = \mathbf{0}$, then generates $\tilde{\mathbf{r}}_{i}^{3} \leftarrow \mathbb{Z}_{q}^{q_{i}}, \tilde{\mathbf{s}}_{i}^{3} \leftarrow \mathbb{Z}_{q}^{w_{i}}$ and encrypts them with Cpk; During distribute decryption, server B receives the evaluated ciphertext from server A and server A computes $\tilde{\mathbf{share}}_{i}^{A} = (\mathbf{F}_{i}^{2} + \tilde{\mathbf{F}}_{i}^{3})(\mathbf{r}_{i}^{1} + \mathbf{r}_{i}^{2} + \tilde{\mathbf{r}}_{i}^{3}) - \mathbf{s}_{i}^{2} - \tilde{\mathbf{s}}_{i}^{3}$
- 3) In the inference phase:
 - a) Preamble: S_{AB} , as the user, sends $-\tilde{\mathbf{r}}_1$ to server A to evaluate the first linear layer. Then S_{AB} receives the corresponding GC label.
 - b) S_{AB} evaluates C_i to obtain the output of garbled circuits and sends it to server A.
 - c) Transition: S_{AB} , as server C, receives the output of first l layers
 - d) S_{AB} obtains labels corresponding to $\tilde{\mathbf{s}}_i^3$ and $\tilde{\mathbf{r}}_{i+1}^3$ from server B, computes the output of garbled circuits and sends it to server B

Theorem VII.1. The view simulated by S_{AB} is computationally indistinguishable from the real view v_{AB} when the adversary is corrupting server A and B.

Proof. We observe that the PRE and INF protocol can be privately reducible to the HE, MPHE, GC and other protocols. Therefore, the security of PRE and INF protocol follow from the standalone security of each protocol by applying Composition Theorem for semi-honest model [36]. The indistinguishability of ciphertext is guaranteed by the semantic security (Proposition VII.1 and Proposition VII.2) of protocol HE and MPHE, and the indistinguishability of garbled circuits with labels is guaranteed by the security property of protocol GC [38]. We only need to show the indistinguishability of the masked input and the the secret shares, then we can prove that PRE and INF preserve their security. Due to using different random values which share identical distribution hence are statistically equivalent, we get $\{\widetilde{msk}_i\}_{i=1}^{L}$, $\{\widetilde{share}_i^A\}_{i=l+1}^L \stackrel{c}{=} \{share_i^A\}_{i=l+1}^L$.

The arguments above yields the following security property.

Proposition VII.3. SECO introduced in Section VI securely realizes f_{pre} and f_{inf} in the presence of semi-honest adversary controlling server A and B.

1) The user corrupted: We prove that the simulated view and the real view are computationally indistinguishable, thus information such as the model parameters which is beyond the view cannot be computed if the adversary corrupts the user.

The real view v_U of the adversary corrupting the user includes:

- 1) The masked input $\{\mathsf{msk}_i = \mathbf{x}_i \mathbf{r}_i \mathbf{d}_i\}_{i=1}^l$
- 2) The additive share $\{\text{share}_{i}^{U} = \mathbf{F}_{i}\mathbf{r}_{i} \mathbf{s}_{i}\}_{i=1}^{l}$
- 3) Garbled circuit $\{\tilde{C}_i\}_{i=1}^l$ and labels $\{|abel_i\}_{i=1}^l$

We define a simulator S_U that simulates the view of an adversary corrupting the user. S_U is given $\mathbf{x}_1, \mathbf{pk}_0, \mathbf{cpk}$ and proceeds as follows:

- 1) S_U chooses a uniform random tape for the user.
- 2) In the preprocessing phase:
 - a) S_U , as server A, receives $\{ct_{r_i} = \mathsf{HE}.\mathsf{Enc}(\mathsf{pk}_0,\mathbf{r}_i)\}_{i=1}^{l+1}$ with $\mathbf{r}_i \leftarrow \mathbb{Z}_q^{q_i}$. With $\tilde{\mathbf{F}}_i$ set to 0, the user receives the evaluated $\{ct_{s_i} = \mathsf{HE}.\mathsf{Enc}(\mathsf{pk}_0,-\tilde{\mathbf{s}}_i)\}_{i=1}^{l+1}$ from S_U for a randomly chosen $\tilde{\mathbf{s}}_i$ from $\mathbb{Z}_q^{w_i}$.
 - b) With S_U acting as the garbler of the garbled circuits for $i^{\text{th}}, i \in \{1, ..., l\}$ non-linear layer, the user receives \tilde{C}_i from S_U and labels corresponding to \mathbf{r}_i and $\tilde{\mathbf{s}}_i$. S_U sets the output of the circuit to be a random value.
- 3) In the inference phase:
 - a) Preamble: S_U , as server A, receives $\mathbf{x}_1 \mathbf{r}_1$ from the user
 - b) S_U , as server A, sets the $\mathbf{F}_i = 0$ evaluates to obtain $\tilde{\mathbf{s}}_i$ and sends the corresponding simulated labels to the user.
 - c) The user evaluates \tilde{C}_i to obtain the output of garbled circuits which is set randomly, then sends it to S_U .

Theorem VII.2. The view simulated by S_U is computationally indistinguishable from the real view v_U when the adversary is corrupting the user.

Proof. Similar to the proof of Theorem VII.1, the views are computationally indistinguishable based on the security properties of HE, MPHE, GC and the use of randomness that are statistically equivalent. \Box

The arguments above yields the following security property.

Proposition VII.4. SECO introduced in Section VI securely realizes f_{pre} and f_{inf} in the presence of semi-honest adversary controlling the user.

VIII. EXPERIMENTS

In this section, we experimentally evaluate the performance of SECO and present our experimental results comparing SECO and other protocols. Currently, SECO supports the deployment of CNN models.

A. Implementation

Our implementation is based on SEAL library [40] and DELPHI's open source code ¹. We first implemented the operations for multiparty key-generation and distribute-decryption for BFV scheme [30] in SEAL. Then we modified DEL-PHI's implementation to support the collaborative inference involving multiple parties as described in Section VI. For

¹https://github.com/mc2-project/delphi

convolutional functions, we employ modulus-switching to reduce the ciphertext size as in Cryptflow2 [41]. We use the same BFV security parameters as DELPHI uses, with 8192 as the degree of polynomial modulus and 2061584302081 as plaintext modulus.



Fig. 4. Comparison of SECO and DELPHI on execution time of ResNet32





Fig. 5. Comparison of SECO and DELPHI on communication cost of ResNet32

B. Setup

We run our benchmarks in the WLAN setting with four instances in Australia or North America. All the instances have Intel Xeon Processor running at 2.3 GHz with 32GB of RAM. Each experiment is conducted 5 times, and the results presented are the average of these runs.

C. Performance

1) Compare with HE-MPC-based Protocols: We experimentally evaluate SECO in terms of execution time and communication cost for preprocessing and inference phase with the deployed model split at different layers. We present the comparison results of SECO, DELPHI [8], and DELPHI-3. DELPHI-3 is a straightforward extension of DELPHI to 3PC setting. This extension employs MPHE for processing linear functions, while utilizing Garbled Circuits (GC) for nonlinear operations. A key distinction between DELPHI-3 and SECO, lies in the allocation of roles within the GC scheme. In SECO, two remote servers are designated as the garbler and the evaluator in the GC process. In DELPHI-3, the gateway server assumes the role of the evaluator, while a remote server functions as the garbler. This will lead to different performance of SECO and DELPHI-3 in only online inference phase. We evaluate three schemes on ResNet32 [42] (with 62 layers in total).

a) User's experience: Figure 4(a) shows the comparison of the online execution time. As illustrated in the figure, the blue bars indicate the duration for which the user must remain connected and actively process the computation while the light blue bars represent the time required to process the model on remote servers. The total height of the bars in the figure represents the user's waiting time, spanning from the start of the query to the receipt of the prediction result. We can tell that the fewer layers deployed on gateway server A, the less execution time of SECO in the preprocessing phase. When all the layers of the model are on server A, SECO is reduced to DELPHI and only the user and server A are involved in the computation. When the number of layers on server A is reduced to two, SECO offers the most efficient experience in terms of both the shortest waiting and participation time for the user comparing to DELPHI, thereby potentially minimizing the user's computational burden. We note that SECO can operate on a device with just 2 GB of memory as the user device (with less than 10 layers of ResNet32 on the gateway server), whereas DELPHI cannot support such a device, making meaningful comparison impossible. This capability allows SECO to utilize lightweight devices, such as the smart watch, as user devices. Depending on the numbers of layers on server A, SECO's results are between $2.6 \times -1 \times$ faster than DELPHI in terms of user's waiting time. Accordingly, SECO's efficiency ranges from $9.9 \times -1 \times$ faster than DELPHI in terms of user's participation time.

Figure 4(b) shows the comparison of the preprocessing execution time, revealing a similar trend as the observation in Figure 4(a). Depending on the numbers of layers on server A, SECO's results are between $11 \times -1 \times$ faster than DELPHI in terms of user's preprocessing time. We note that SECO's preprocessing only involves the preprocessing of \mathbf{F}_A . The three server pre-compute the secret shares for \mathbf{F}_{BC} before the user appears in the system.

As illustrated in Figure 5(a) and Figure 5(b), SECO demonstrates up to $18.5 \times$ more efficient compared to DELPHI in



(b) ResNet32 total online communication cost

Fig. 6. Comparison of SECO and DELPHI-3 on execution time and communication cost of ResNet32 $\,$

terms of the user's online communication cost, additionally, it is up to $16.6 \times$ more efficient than DELPHI regarding the user's preprocessing communication cost. Even though SECO's total communication cost in the online inference phase is higher than DELPHI's, SECO reduces the user's cost. Figure 5(a) shows that SECO can provide a tradeoff in communication load between user and server nodes, which might be desirable to service provider.

b) Effect of schemes with different assignments of servers in GC: Figure 6(a) and Figure 6(b) depict the online performance of SECO, DELPHI, and DELPHI-3. We can tell from figures, the time and the cost for processing \mathbf{F}_A are the same for SECO and DELPHI-3 as the 2PC protocol in both cases is derived from DELPHI. However, the time and cost for processing \mathbf{F}_{BC} are significantly reduced in SECO. The inefficiency in DELPHI-3 arises because the evaluator of the garbled circuit obtains labels for server A's share during \mathbf{F}_{BC} pre-computation, making server A's participation in the inference phase unnecessary. In contrast, SECO designates server C as the evaluator, involving only servers B and C in the inference phase. Our approach streamlines the process, avoiding the need for three-server involvement as in DELPHI-3.

2) Compare with MPC-based protocols: In this section, we compare SECO with pure MPC-based protocols. To make a fair comparison, we deploy all the layers of the model on remote servers in SECO. So the user can send the masked input and receive the prediction result without further interaction like in pure MPC-based protocols. Figure 7 shows the

 TABLE II

 Comparison of execution time and communication cost

		MiniONN	L	.eNet
	Time (s)	Comm. cost (MB)	Time (s)	Comm. cost
SecureNN Falcon SECO	183.493 8.426 4.776	177.264 1.097 50.479	9.303 4.862	1.611 58.888

communication cost for the user in protocols including SECO, Falcon [22], and SecureNN [21]. Although there is only a one-way transmission from user to servers, due to the use of replicated secret shares, the user in Falcon needs to duplicates each of the three input shares, and the user must transmit two distinct shares to the respective server. This process leads to a communication cost that is $6 \times$ greater compared to that in SECO. SecureNN uses a 2-out-of-2 secret sharing scheme between two servers, and the user needs to deliver each of two shares to the respective servers. It differs from the 2-out-of-2 secret sharing used between a user and a server in SECO. This process results in the communication cost $2\times$ greater compared to SECO. Furthermore, we assess the execution time starting from when the user transmits the input until the user receives the prediction result. In Table II, we evaluate 3 protocols on MiniONN and LeNet. The outperformance of SECO is attributed to the online inference only involving two (remote) servers while other protocols are executed between three servers. Although SECO is faster than other MPC-based protocols, it incurs a communication cost between servers that is 36 to 48 times higher than that of Falcon. We remind that SECO operates under a different system and threat model. SECO can protect the user's input even when the adversary corrupts two of the three servers while other protocols assume non-collusion between servers. The increase in communication cost can be justified by SECO's emphasis on enhancing the system's robustness and trustworthiness for the user.



Fig. 7. User's communication cost comparison

IX. CONCLUSION

In this paper, we design and implement SECO, a novel hybrid HE-MPC-based secure inference protocol with model splitting in a multi-server hierarchy setting. SECO ensures the confidentiality of the input data withstanding passive adversaries corrupting up to 2 servers, and also protects the model parameters from the user. Furthermore, SECO hides more information about the model architecture than other HE-MPC-based protocols and pure MPC-based protocols. In the experiments, SECO is shown to minimize the user device's obligation in computation and communication, making it applicable to devices with limited resources.

REFERENCES

- J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, 2012.
- [2] A. C.-C. Yao, "How to generate and exchange secrets (extended abstract)," in FOCS, 1986.
- [3] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International conference on machine learning*, pp. 201–210, PMLR, 2016.
- [4] E. Hesamifard, H. Takabi, and M. Ghasemi, "Cryptodl: Deep neural networks over encrypted data," 2017.
- [5] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz, "Chet: an optimizing compiler for fully-homomorphic neural-network inferencing," in *Proceedings of the* 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 142–156, 2019.
- [6] A. Sanyal, M. J. Kusner, A. Gascón, and V. Kanade, "Tapas: Tricks to accelerate (encrypted) prediction as a service," 2018.
- [7] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "Gazelle: A low latency framework for secure neural network inference," 01 2018.
- [8] P. Mishra, R. T. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *IACR Cryptol. ePrint Arch.*, 2020.
- [9] C. Gentry, "Fully homomorphic encryption using ideal lattices," in Proceedings of the forty-first annual ACM symposium on Theory of computing, pp. 169–178, 2009.
- [10] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, "Privacy-preserving ridge regression on hundreds of millions of records," in 2013 IEEE Symposium on Security and Privacy, pp. 334– 348, 2013.
- [11] D. Wu and J. Haven, "Using homomorphic encryption for large scale statistical analysis," FHE-SI-Report, Univ. Stanford, Tech. Rep. TRdwu4, 2012.
- [12] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data," *Cryptology ePrint Archive*, 2014.
- [13] J. W. Bos, K. Lauter, and M. Naehrig, "Private predictive analysis on encrypted medical data," *Journal of biomedical informatics*, vol. 50, pp. 234–243, 2014.
- [14] T. Graepel, K. Lauter, and M. Naehrig, "MI confidential: Machine learning on encrypted data," in *International Conference on Information Security and Cryptology*, pp. 1–21, Springer, 2012.
- [15] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *international conference on the theory and application of cryptology and information security*, pp. 3–33, Springer, 2016.
- [16] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski, "ngraph-he: A graph compiler for deep learning on homomorphically encrypted data," 2018.
- [17] C. Boura, N. Gama, and M. Georgieva, "Chimera: a unified framework for b/fv, tfhe and heaan fully homomorphic encryption and predictions for deep learning," *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 758, 2018.
- [18] B. Reagen, W.-S. Choi, Y. Ko, V. T. Lee, H.-H. S. Lee, G.-Y. Wei, and D. Brooks, "Cheetah: Optimizing and accelerating homomorphic encryption for private inference," in 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), pp. 26–39, 2021.
- [19] D. Demmler, T. Schneider, and M. Zohner, "Aby-a framework for efficient mixed-protocol secure two-party computation.," in NDSS, 2015.
- [20] P. Mohassel and P. Rindal, "Aby3: A mixed protocol framework for machine learning," in *Proceedings of the 2018 ACM SIGSAC conference* on computer and communications security, pp. 35–52, 2018.

- [21] S. Wagh, D. Gupta, and N. Chandran, "Securenn: 3-party secure computation for neural network training.," *Proc. Priv. Enhancing Technol.*, vol. 2019, no. 3, pp. 26–49, 2019.
- [22] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "Falcon: Honest-majority maliciously secure framework for private deep learning," *arXiv preprint arXiv:2004.02229*, 2020.
- [23] F. Boemer, R. Cammarota, D. Demmler, T. Schneider, and H. Yalame, "Mp2ml: A mixed-protocol machine learning framework for private inference," in *Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice*, PPMLP'20, (New York, NY, USA), p. 43–45, Association for Computing Machinery, 2020.
- [24] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International conference* on the theory and application of cryptology and information security, pp. 409–437, Springer, 2017.
- [25] C. Thapa, M. A. P. Chamikara, S. Camtepe, and L. Sun, "Splitfed: When federated learning meets split learning," 2022.
- [26] G.-L. Pereteanu, A. Alansary, and J. Passerat-Palmbach, "Split he: Fast secure inference combining split learning and homomorphic encryption," arXiv preprint arXiv:2202.13351, 2022.
- [27] T. Khan, K. Nguyen, and A. Michalas, "Split ways: Privacy-preserving training of encrypted data using split learning," 2023.
- [28] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," arXiv preprint arXiv:1812.00564, 2018.
- [29] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *Journal of Network and Computer Applications*, vol. 116, pp. 1–8, 2018.
- [30] C. Mouchet, J. Troncoso-Pastoriza, J.-P. Bossuat, and J.-P. Hubaux, "Multiparty homomorphic encryption from ring-learning-with-errors," *Proceedings on Privacy Enhancing Technologies*, vol. 4, pp. 291–311, 2021.
- [31] M. O. Rabin, "How to exchange secrets with oblivious transfer," IACR Cryptol. ePrint Arch., vol. 2005, p. 187, 2005.
- [32] S. Even, O. Goldreich, and A. Lempel, "A randomized protocol for signing contracts.," vol. 28, pp. 205–210, 01 1982.
- [33] M. Bellare, T. Hoang, and P. Rogaway, "Foundations of garbled circuits," pp. 784–796, 10 2012.
- [34] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, "Efficient garbling from a fixed-key blockcipher," in 2013 IEEE Symposium on Security and Privacy, pp. 478–492, 2013.
- [35] Z. He, T. Zhang, and R. B. Lee, "Model inversion attacks against collaborative inference," in *Proceedings of the 35th Annual Computer Security Applications Conference*, ACSAC '19, (New York, NY, USA), p. 148–162, Association for Computing Machinery, 2019.
- [36] O. Goldreich, "Foundations of cryptography. ii: Basic applications," vol. 2, 05 2004.
- [37] Y. Lindell, "How to simulate it-a tutorial on the simulation proof technique," *Tutorials on the Foundations of Cryptography*, pp. 277–346, 2017.
- [38] Y. Lindell and B. Pinkas, "A proof of security of yao's protocol for two-party computation," *Journal of Cryptology*, vol. 22, pp. 161–188, 04 2009.
- [39] M. O. Rabin, "How to exchange secrets with oblivious transfer," 2005. Harvard University Technical Report 81 talr@watson.ibm.com 12955 received 21 Jun 2005.
- [40] "Microsoft SEAL (release 3.6)." https://github.com/Microsoft/SEAL, Nov. 2020. Microsoft Research, Redmond, WA.
- [41] 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*, ACM, oct 2020.
- [42] 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, pp. 770–778, 2016.
- [43] 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, pp. 619-631, 2017.
- [44] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

APPENDIX A

NOTATION

Table III provides the summary of the symbols used throughout this work.

APPENDIX B

SECURITY PROOF

We provide the security proof for other cases when server B and C are corrupted or server A and C are corrupted.

a) Server B, C corrupted: The real view v_{BC} of the adversary corrupting server B and C includes:

- 1) Ciphertexts set $ct_{S_{BC}} = \{\{ct_{r_i^2}, ct_{r_i^3}\}_{i=l+2}^L, \}$ $\{ct_{F_i^2}, ct_{F_i^3}, ct_{s_i^2}, ct_{s_i^2}\}_{i=l+2}^L, \{ct_{\mathsf{share}_i^A}\}_{i=l+1}^L\}$
- 2) The masked input of $\mathsf{msk}_i = \mathbf{x}_i \mathbf{r}_i^1, i \in \{l+2, ..., L\}$
- 3) Garbled circuit $\{\hat{C}_i\}_{i=l+1}^L$ and $\{|abe|_i\}_{i=l+1}^L$ obtained by Server B and C

We define a real-world simulator S_{BC} that simulates the view of an adversary corrupting server B and C. S_{BC} is given $\{\mathbf{F}_{i}^{2}, \mathbf{F}_{i}^{3}\}_{i=l+1}^{L}, \mathsf{pk}_{0}, \mathsf{cpk}, \{(q_{i}, w_{i})\}_{i=l+1}^{L}, \mathsf{msk}_{l+1} \text{ and }$ proceeds as follows:

- 1) S_{BC} chooses a uniform random tape for server B and C.
- 2) In the preprocessing phase:
 - a) S_{BC} , as server A, receives ciphertexts MPHE.Enc(cpk, \mathbf{F}_{i}^{j}), MPHE.Enc(cpk, \mathbf{r}_{i}^{j}), MPHE.Enc(cpk, \mathbf{s}_{i}^{j}) from server B and C.
 - b) S_{BC} generates $\tilde{\mathbf{r}}_i^1 \leftarrow \mathbb{Z}_Q^{q_i}$ and computes $\widetilde{\mathsf{share}}_i^A = (\mathbf{F}_i^3 + \mathbf{F}_i^2)(\tilde{\mathbf{r}}_i^1 + \mathbf{r}_i^2 + \mathbf{r}_i^3) \mathbf{s}_i^3 \mathbf{s}_i^2$
 - c) Server B sends the garbled circuit \hat{C}_i and labels for inputs from server B to server C; S_{BC} obtains the labels from server B and forwards it to server С
- 3) In the inference phase:
 - a) Transition: S_{BC} generates a random value as the output of the first l layers msk_{l+1} and sends it to server B and C

Theorem B.1. The view simulated by S_{BC} is computationally indistinguishable from the real view v_{BC} when the adversary corrupting server B and C.

Proof. Similar to the proof of Theorem VII.1, the views are computationally indistinguishable based on the security properties of HE, MPHE, GC, and the use of randomness that are statistically equivalent.

The arguments above yields the following security property.

Proposition B.1. SECO introduced in Section VI securely realizes f_{pre} and f_{inf} in the presence of semi-honest adversary controlling server B and C.

b) Server A, C corrupted: The real view v_{AC} of the adversary corrupting server A and C includes:

1) Ciphertexts set $ct_{S_{AC}} = \{\{ct_{r_i}\}_{i=1}^{l+1}, \{ct_{r_i^2}, ct_{r_i^3}\}_{i=l+2}^{L}, \{ct_{F_i^2}, ct_{F_i^3}, ct_{s_i^2}, ct_{s_i^2}\}_{i=l+2}^{L}, \{ct_{\mathsf{share}_i^A}\}_{i=l+1}^{L}\}$

- 2) The masked input of the $i^{\text{th}}, i \in \{l + 2, ..., L\}$ layer $\mathsf{msk}_i = \mathbf{x}_i - \mathbf{r}_i^2$
- 3) The additive secret shares set share $_{i}^{A} = (\mathbf{F}_{i}^{2} + \mathbf{F}_{i}^{3})\mathbf{r}_{i} \mathbf{F}_{i}^{2}$ $\begin{aligned} \mathbf{s}_i^2 - \mathbf{s}_i^3, i \in \{l+1, ..., L\} \\ \text{4) Garbled circuit } \{\tilde{C}_i\}_{i=1}^L \text{ and } \{\mathsf{label}_i\}_{i=l+1}^L \text{ for garbled} \end{aligned}$

We define a real-world simulator S_{AC} that simulates the view of an adversary corrupting server A and C. S_{AC} is given $\{\mathbf{F}_{i}^{3},(q_{i},w_{i})\}_{i=l+1}^{L}, \mathbf{F}_{A},\mathsf{pk}_{0},\mathsf{cpk} \text{ and proceeds as follows:}$

- 1) S_{AC} chooses a uniform random tape for server A and С.
- 2) In the preprocessing phase:
 - a) S_{AC} , as the user, computes $\mathsf{HE}.\mathsf{Enc}(\mathsf{pk}_0, \tilde{\mathbf{r}}_i), i \in$ $\{1, ..., l + 1\}$ to server A where $\tilde{\mathbf{r}}_i$ is randomly chosen from $\mathbb{Z}_Q^{q_i}$. Then S_{AC} receives the evaluated ciphertext from server A.
 - b) With S_{AC} as the evaluator of the garbled circuits for $i^{\text{th}}, i \in \{1, ..., l\}$ non-linear layer, S_{AC} receives $C_i, i \in \{1, ..., l\}$ from server A and labels corresponding to random input generated by S_{AC} .
 - c) S_{AC} , as server B, sets $\tilde{\mathbf{F}}_i^2 = \mathbf{0}$, then generates $ilde{\mathbf{r}}_i^2 \leftarrow \mathbb{Z}_Q^{q_i}, ilde{\mathbf{s}}_i^2 \leftarrow \mathbb{Z}_Q^{w_i}$ and encrypts them respectively with cpk; For processing distribute decryption, server C receives the evaluated ciphertext from server A and $\widetilde{\mathsf{share}}_{i}^{A} = (\mathbf{F}_{i}^{3} + \widetilde{\mathbf{F}}_{i}^{2})(\mathbf{r}_{i}^{3} + \widetilde{\mathbf{r}}_{i}^{2} + \widetilde{\mathbf{F}}_{i}^{2})$ \mathbf{r}_i^1) – \mathbf{s}_i^3 – $\tilde{\mathbf{s}}_i^2$ from S_{AB}
 - d) S_{AC} , as server B, runs the simulator for garbled circuits and generates $\tilde{C}_i, i \in \{l+1, ..., L\}$, then sends labels corresponding to $\tilde{\mathbf{r}}_i^2$ to server C
- 3) In the inference phase:
 - a) Preamble: S_{AC} sends $-\tilde{\mathbf{r}}_1$ to server A to evaluate the first linear layer.
 - b) S_{AC} , as the user, obtains labels from server A
 - c) S_{AC} , as the user, sends a random value as the output of garbled circuits back to server A.
 - d) Transition: S_{AC} , as server B, receives the output of the first *l* layers
 - e) S_{AC} , as server B, sends labels corresponding to \tilde{s}_i^2 to server C
 - f) S_{AC} receives the output of garbled circuits from server C

Theorem B.2. The view simulated by S_{AC} is computationally indistinguishable from the real view v_{AC} when the adversary corrupting server A and C.

Proof. Similar to the proof of Theorem VII.1, the views are computationally indistinguishable based on the security properties of HE, MPHE, GC, and the use of randomness that are statistically equivalent.

The arguments above yields the following security property.

Proposition B.2. SECO introduced in Section VI securely realizes f_{pre} and f_{inf} in the presence of semi-honest adversary controlling server A and C.

Symbol	Description
j	Party index (0 for the user, 1 for server A, 2 for server B, 3 for server C)
\mathbb{Z}_q	$[-\frac{1}{2},\frac{1}{2})$
R_t	Plaintext space for HE scheme
R_q	Ciphertext space for HE scheme
L	Total number of layers of the model
l	The number of layers stored on server A
q_i	Input size of <i>i</i> th layer
w_i	Output size of i^{th} layer
\mathbf{F}_A	The partial model on server A
\mathbf{F}_{BC}	The partial model on server B and C
\mathbf{F}_{i}	Model parameter of the i^{th} layer
\mathbf{F}_{i}^{j}	Model parameter of the i^{th} layer on Party j
\mathbf{x}_i	The input of the model's i^{th} layer
$\hat{\mathbf{y}}$	The prediction result
\mathbf{r}_i	Randomness masking the input of i^{th} layer
\mathbf{s}_i	Randomness masking the output of i^{th} layer
(sk_j, pk_j)	Key pair generated by party j
cpk	Common public key
ct_x	Ciphertext of message x
C_i	Circuit for computing the i^{th} ReLU layer
\tilde{C}_i	Garbled circuit correspongind to C_i
$label_x$	Garbled circuit input labels corresponding to value x

TABLE III Notation Summary

APPENDIX C DELPHI-3

DELPHI-3 is a 3PC protocol directly expanded from DEL-PHI. DELPHI-3 sets server B as the garbler and server A as the evaluator for GCs.

a) Preprocessing for \mathbf{F}_{BC} : The preprocessing of linear layers is the same with SECO as described in Algorithm 8. The preprocessing of ReLU is as follows: with server B as the garbler, server A as the evaluator, and server C providing input as the third party, server B first prepares the garbled circuit \tilde{C}_i and labels by running GC.Garble(Params, C_i) where Params is the security parameters and C_i is described in Algorithm 9 in Appendix F. To transmit the labels corresponding to actual input values, server B sends the labels of the input value provided by itself to server A through a public channel. For input values from server A and C, server A and C first obtain the actual labels from server B via OT. Then server A collects the labels obtained by server C through a public channel.

b) Online-inference for \mathbf{F}_{BC} : The inference of \mathbf{F}_{BC} involves the participation of server A, B, and C. At the beginning of the inference for i^{th} layer $i \in \{l+1, ..., L\}$, server B and C hold $\mathbf{x}_i - \mathbf{r}_i$ where \mathbf{r}_i is solely contributed by the user when i = l + 1, and is contributed by server A, B and C for linear other layers of \mathbf{F}_{BC} . Server A, B and C collaborate to do inference on i^{th} layer $i \in \{l + 1, ..., L\}$. First server B and C will compute $\mathbf{F}_i^j(\mathbf{x}_i - \mathbf{r}_i) + \mathbf{s}_i^j$, resulting in three servers holding three additive shares of the real prediction result for the current layer $(\mathbf{F}_i^2 + \mathbf{F}_i^3)\mathbf{x}_i$. The garbled circuit \tilde{C}_i on server A (Algorithm 9 in Appendix F) first aggregates shares to recover $(\mathbf{F}_i^2 + \mathbf{F}_i^3)\mathbf{x}_i$, then computes ReLU of it, finally masks the ReLU output with the randomness of the next layer contributed by three servers. The entire process is in the encrypted domain. Server A can evaluate the garbled circuit when collecting all labels for input wires of \tilde{C}_i . Note the inefficiency of this method is that despite three servers holding the respective additive shares of the real prediction result, the evaluator of the garbled circuit already obtains the labels for server A's share during the pre-computation for \mathbf{F}_{BC} . Thus server A does not need to involve in the inference phase of \mathbf{F}_{BC} . We observe this inefficiency and set server C as the evaluator in SECO instead.

APPENDIX D Setup phase in SECO

The setup phase in SECO includes the MPHE key generation and the preparation for \mathbf{F}_{BC} . This phase involves three servers. In Table IV and Table V, we provide the execution time and communication cost for the setup phase.

APPENDIX E Structures of Neural Networks

a) MiniONN: This is a 4 layer network with 2 convolutional and 2 fully-connected layers selected from prior work MiniONN [43]. The structure is provided in Table VI. It has around 10,500 parameters in total.

b) LeNet: This network, proposed by LeCun et al. [44] contains 2 convolutional layers and 2 fully connected layers with about 431K parameters. Table VII shows the structure.

APPENDIX F PSEUDOCODE

Numbers of layers on service	ver B,C 60	56	52	4	18	44	4	40	36	32
Execution time (s) Comm. cost (GB)	290.202 21.609	2 271.366 19.920	250.965 18.230	5 233 16.	.168 541	215.9 14.85	94 192 51 13	2.566 .161	175.395 11.747	162.568 10.621
	SE	TUP PHASE O	TABLE V OF SECO ON	N RESNE	et32 (2	2)				
Numbers of layers	s on server B,C	28	24	20		16	12	8	4	_
Execution Comm. co	time (s) st (GB)	148.177 9.495	136.214 8.368	117.16 7.242	69	1.340 5.822	68.487 4.404	41.719 2.985	9 16.042 5 1.567	
Layer	· Туре	Input Siz	ze Outpu	ıt Size		D	etails			
1	Convolution	28×28×	1 24×2	4×16		5×5 filt	ers, stride	1		
2	Subsampling	24×24×1	16 12×1	2×16	2×2	average	pooling,	stride 2		
3	ReLU	1×230	4 1 ×	2304						
4	Convolution	12×12×1	16 8×8	×50		5×5 filt	ers, stride	1		
5	Subsampling	8×8×16	6 4×4	×16	2×2	average	pooling,	stride 2		
6	ReLU	1×256	1×	256						
7	Fully Connected	256	10	00						
8	ReLU	100	10	00						
9	Fully Connected	100	1	0						
			TADIEVI							

TABLE IV SETUP PHASE OF SECO ON RESNET32 (1)

TABLE VI STRUCTURE OF MINIONN

Layer	Туре	Input Size	Output Size	Details
1	Convolution	28×28×1	$24 \times 24 \times 20$	5×5 filters, stride 1
2	Subsampling	$24 \times 24 \times 20$	$12 \times 12 \times 20$	2×2 average pooling, stride 2
3	ReLU	1×2880	1×2880	
4	Convolution	12×12×20	8×8×50	5×5 filters, stride 1
5	Subsampling	8×8×50	$4 \times 4 \times 50$	2×2 average pooling, stride 2
6	ReLU	1×800	1×800	
7	Fully Connected	800	500	
8	ReLU	500	500	
9	Fully Connected	500	10	

TABLE VII STRUCTURE OF LENET

> Algorithm 12 Pre-Compute the share for the *i*th linear layer, $i \in \{1, ...l\}$

Inputs: (q_i, w_i) : the input and output shape of the i^{th} layer; (sk_0, pk_0) : the plain HE key pair generated by the user

The user:

 $\mathbf{r}_i \leftarrow \mathbb{Z}_q^{q_i}$

Encrypt $ct_{r_i} = \mathsf{HE}.\mathsf{Enc}(\mathsf{pk}, \mathbf{r}_i)$ Transmit ct_{r_i} to server A

Server A:

 $\mathbf{s}_i \leftarrow \mathbb{Z}_q^{w_i}$ Compute HE.Enc(pk, $\mathbf{F}_i \mathbf{r}_i - \mathbf{s}_i$) and transmit this ciphertext to the user

The user:

Decrypt the ciphertext to obtain $\mathbf{F}_i \mathbf{r}_i - \mathbf{s}_i$ using sk_0

Algorithm 11 A circuit C_i that computes ReLU function of the i^{th} layer, $i \in \{1, ..., l\}$

Input from the user: $\mathbf{F}_i \mathbf{r}_i - \mathbf{s}_i, \mathbf{r}_{i+1}$

Input from server A:
$$\mathbf{F}_i(\mathbf{x}_i - \mathbf{r}_i) + \mathbf{s}_i, \mathbf{d}_{i+1}$$

1: Compute
$$\mathbf{F}_i \mathbf{x}_i = \mathbf{F}_i (\mathbf{x}_i - \mathbf{r}_i) + \mathbf{s}_i + \mathbf{F}_i \mathbf{r}_i - \mathbf{s}_i$$

- 2: Compute $\mathbf{x}_{i+1} = ReLU(\mathbf{F}_i \mathbf{x}_i)$
- 3: Output $\mathbf{x}_{i+1} \mathbf{r}_{i+1} \mathbf{d}_{i+1}$

Algorithm 13 Pre-Compute the share for the l + 1th linear layer

Inputs: (q_{l+1}, w_{l+1}) : the input and output shape of the $l + 1^{\text{th}}$ layer; cpk: common public key; The user: $\mathbf{r}_{l+1} \leftarrow \mathbb{Z}_q^{q_{l+1}}$ Compute $ct_{r_{l+1}} = \mathsf{MPHE}.\mathsf{Enc}(\mathsf{cpk}, \mathbf{r}_{l+1})$ Transmit $ct_{r_{l+1}}$ to server A Server B and C: $\mathbf{s}_{l+1}^{j} \leftarrow \mathbb{Z}_{q}^{w_{l+1}}$ Compute MPHE.Enc(cpk, \mathbf{s}_{l+1}^{j}) \rightarrow $ct_{s_{l+1}^j},$ $\mathsf{MPHE}.\mathsf{Enc}(\mathsf{cpk},\mathbf{F}_{l+1}^j) \to ct_{F_{l+1}^j}$ Transmit $(ct_{s_{l+1}^j}, ct_{F_{l+1}^j})$ to server A Server A: $\mathsf{MPHE}.\mathsf{Eval}(ct_{F_{l+1}^2}, ct_{F_{l+1}^3}, \mathsf{Add}) \rightarrow ct_{F_{l+1}}$ $\mathsf{MPHE}.\mathsf{Eval}(ct_{s_{l+1}^2}, ct_{s_{l+1}^3}^{i+1}, \mathsf{Add}) \rightarrow ct_{s_{l+1}}$ MPHE.Eval(MPHE.Eval($ct_{F_{l+1}}, ct_{r_{l+1}}, Lin-OP$) $(ct_{s_{l+1}}, Sub) \rightarrow ct_{F_{l+1}r_{l+1}-s_{l+1}}$ Server A, B, C: $\begin{array}{l} \mathsf{MPHE.DisDec}(ct_{F_{l+1}r_{l+1}-s_{l+1}},\{sk_j\}_{j\in\{1,2,3\}})\\ \mathsf{Server}\ \mathsf{A}\ \mathsf{obtains}\ (\mathbf{F}_{l+1}^2+\mathbf{F}_{l+1}^3)\mathbf{r}_{l+1}-\mathbf{s}_{l+1}^2-\mathbf{s}_{l+1}^3 \end{array}$

Algorithm 15 Inference for the i^{th} layer, $i \in \{1, ..., l\}$

Inputs: $(\mathbf{F}_i, \mathbf{x}_i - \mathbf{r}_i, \mathbf{s}_i)$: held by server A; \tilde{C}_i : the garbled circuit of Algorithm 11; label: random labels for input wires of Algorithm 11; $|abe|_{\mathbf{F}_i \mathbf{r}_i - \mathbf{s}_i}$, $|abe|_{\mathbf{r}_{i+1}}$, $|abe|_{\mathbf{d}_{i+1}}$: labels tranmitted to the user Server A: Compute $\mathbf{F}_i(\mathbf{x}_i - \mathbf{r}_i) + \mathbf{s}_i$ Server A and the user: GC. Transfer(label, \mathbf{d}_i , $\mathbf{F}_i(\mathbf{x}_i - \mathbf{r}_i) + \mathbf{s}_i) \rightarrow \mathsf{label}_{d_i}$, $\mathsf{label}_{F_i(x_i-r_i)+s_i}$ The user obtains $|abe|_{\mathbf{F}_i(\mathbf{x}_i - \mathbf{r}_i) + \mathbf{s}_i}$, $|abe|_{d_i}$ The user: $\mathsf{GC}.\mathsf{Eval}(\hat{C}_i, \mathsf{label}_{F_i(x_i-r_i)+s_i}, \mathsf{label}_{d_i}, \mathsf{label}_{r_{i+1}},$ $\mathsf{label}_{F_ir_i-s_i}) \to \mathbf{x}_{i+1} - \mathbf{r}_{i+1} - \mathbf{d}_{i+1}$ Send $\mathbf{x}_{i+1} - \mathbf{r}_{i+1} - \mathbf{d}_{i+1}$ to Server A Server A: Compute $\mathbf{x}_{i+1} - \mathbf{r}_{i+1}$

Algorithm 14 Construct Garbled Circuit for the i^{th} activation layer, $i \in \{1, ..., l\}$

Inputs: Params: Garbled Circuit security parameters, C_i : the circuit in Algorithm 11; $(\mathbf{F}_i \mathbf{r}_i - \mathbf{s}_i, \mathbf{r}_{i+1})$: the precomputed additive share of i^{th} layer and the randomness for next layer held by server A ; q_i : the input shape of i^{th} layer

```
Server A:

\mathbf{d}_{i+1} \leftarrow \mathbb{Z}_q^{q_i}

\operatorname{GC.Garble}(\operatorname{Params}, C_i) \to \tilde{C}_i, label

\operatorname{Transmit} \tilde{C}_i to the user

The user and server A:

\operatorname{GC.Transfer}(\operatorname{label}, \mathbf{F}_i \mathbf{r}_i - \mathbf{s}_i) \to \operatorname{label}_{\mathbf{F}_i \mathbf{r}_i - \mathbf{s}_i}

\operatorname{GC.Transfer}(\operatorname{label}, \mathbf{r}_{i+1}) \to \operatorname{label}_{\mathbf{r}_{i+1}}

\operatorname{GC.Transfer}(\operatorname{label}, \mathbf{d}_{i+1}) \to \operatorname{label}_{\mathbf{d}_{i+1}}

\operatorname{The} user obtains \operatorname{label}_{\mathbf{F}_i \mathbf{r}_i - \mathbf{s}_i}, \operatorname{label}_{\mathbf{r}_{i+1}}, \operatorname{label}_{\mathbf{d}_{i+1}}
```

Algorithm 16 Construct the Garbled Circuits for the i^{th} activation layer $i \in \{l + 1, ..., L\}$

Inputs: C_i : the circuit in Algorithm 9; \mathbf{E}_i^1 : the precomputed additive share held by server A; \mathbf{r}_{i+1}^2 : the randomness generated by server B Server B: GC.Garble(Params, C_i) $\rightarrow \tilde{C}_i$, label Transmit \tilde{C}_i to server C Server A and B: GC.Transfer(label, \mathbf{E}_i^1) \rightarrow label \mathbf{E}_i^1 Server A obtains label \mathbf{E}_i^1 transmits it to server C Server B and C: GC.Transfer(label, \mathbf{r}_{i+1}^3) \rightarrow label \mathbf{r}_{i+1}^3 Server C obtains label \mathbf{r}_{i+1}^3