Confidential and Verifiable Machine Learning Delegations on the Cloud

Wenxuan Wu
Texas A&M University

Soamar Homsi
US Air Force Research Laboratory
Information Warfare Division

Yupeng Zhang
University of Illinois
Urbana-Champaign

ABSTRACT

With the growing adoption of cloud computing, the ability to store data and delegate computations to powerful and affordable cloud servers have become advantageous for both companies and individual users. However, the security of cloud computing has emerged as a significant concern. Particularly, Cloud Service Providers (CSPs) cannot assure data confidentiality and computations integrity in mission-critical applications. In this paper, we propose a confidential and verifiable delegation scheme that advances and overcomes major performance limitations of existing Secure Multiparty Computation (MPC) and Zero Knowledge Proof (ZKP). Secret-shared Data and delegated computations to multiple cloud servers remain completely confidential as long as there is at least one honest MPC server. Moreover, results are guaranteed to be valid even if all the participating servers are malicious. Specifically, we design an efficient protocol based on interactive proofs, such that most of the computations generating the proof can be done locally on each server. In addition, we propose a special protocol for matrix multiplication where the overhead of generating the proof is asymptotically smaller than the time to evaluate the result in MPC. Experimental evaluation demonstrates that our scheme significantly outperforms prior work, with the online prover time being 1-2 orders of magnitude faster. Notably, in the matrix multiplication protocol, only a minimal 2% of the total time is spent on the proof generation. Furthermore, we conducted tests on machine learning inference tasks. We executed the protocol for a fully-connected neural network with 3 layers on the MNIST dataset and it takes 2.6 seconds to compute the inference in MPC and generate the proof, 88× faster than prior work. We also tested the convolutional neural network of Lenet with 2 convolution layers and 3 dense layers and the running time is less than 300 seconds across three servers. [1]

KEYWORDS

Zero-Knowledge Proof, Privacy-preserving Machine Learning, Secure Multiparty Computations

1 INTRODUCTION

Cloud computing has revolutionized the way organizations, businesses and individuals store, access, and process data. Rather than relying on traditional data centers that are centralized, expensive to setup, and costly to maintain and run, cloud computing offers a remote access to unlimited resources available on demand over the internet. Although cloud computing has numerous advantages, including elasticity, scalability, and cost savings, it comes with major cybersecurity risks and concerns. Specifically, they require end-to-end confidentiality of the data, meaning that the data should remain confidential from Cloud Services Providers (CSPs) and the integrity of the data and computations, meaning that the data and results should be valid even if the cloud servers are compromised. For example, governmental agencies procure cloud services based on compliance with specified standards that must be established as a trusted entity in providing secure services. Although the processes for meeting these standards are costly and time-consuming, they still cannot provide any mathematically-grounded assurances on data and computation security. In this paper, we aim to leverage widely available commercial CSPs without any dependence on their trustworthiness or security measures.

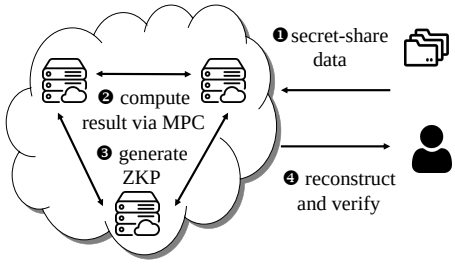

Figure 1: Confidential and Verifiable Delegation.

We thus seek to address the limitations of the state of the art applied cryptographic methods and to build the necessary building blocks that enable the development of secure and efficient applications such as Machine Learning (ML) evaluation. We primarily focus on confidential and verifiable ML prediction outsourced to commercial CSPs by combining primitives from secure multiparty computations (MPC) with zero-knowledge proofs (ZKP). Figure 1 illustrates an example of a user, with limited resources and energy, on a task that needs to utilize a ML model. To ease the burden of performing local data storing and processing, our scheme can delegate ML inference to multiple CSPs via methods, such as additive secret-sharing [40], to keep the model and data confidential. After the user secret-shares the data, the cloud servers compute the ML inference using an MPC protocol. In order to verify the correctness of the ML prediction, the servers generate a ZKP via an MPC protocol. Finally, the user reconstructs and verifies the results.

Attack model and security guarantees. In our setting, we consider attackers that can compromise any number of cloud servers, and which behave arbitrarily once compromised. First, we aim to guarantee the confidentiality of both the data and ML model in the presence of malicious attackers compromising all but one server

[1]Approved for Public Release on 16 March 2024; Distribution Is Unlimited; Case Number: AFRL-2024-1294.

Scheme	Prover Time	Prover Communication	Proof Size	Verifier Time	Setup Type
zkSNARK [35]	$O(C \log C)$	$O(C)$	$O(1)$	$O(1)$	per-circuit
Plonk [35]	$O(C \log C)$	$O(C)$	$O(1)$	$O(1)$	universal
Marlin [11]	$O(C \log C)$	$O(C)$	$O(1)$	$O(1)$	universal
[24]	$O(C)$	$O(C)$	$O(\log C)$	$O(C)$	transparent
[12]	$O(C \log C)$	$O(C \log C)$	$O(\log^2 C)$	$O(\log^2 C)$	transparent
Ours (General)	$O(C)$	$O(C)$	$O(d \log C)$	$O(d \log C)$	universal
Ours (Matrix)	$O(m^2)$	$O(m)$	$O(\log m)$	$O(\log m)$	universal

Table 1: Comparison to existing schemes. C denotes the circuit size and d is the depth of the circuit. The matrices are of size $m \times m$, and a circuit computing the matrix multiplication naively is of size $C = O(m^3)$.

under MPC protocols [26]. Second, we want to ensure the integrity of the ML inference results even if all servers are compromised. This property is not satisfied when only using malicious MPC protocols. We will make the MPC servers generate a ZKP to convince the verifier about the correctness of the result. Consequently, our scheme guarantees both the confidentiality of data and ML model, as long as there is at least one honest server, and the integrity of the inference results even if all servers are malicious. See Section 3.1 for formal definitions.

Applications. One concrete use case of the proposed scheme is on the ZKP generation of ML inferences for blockchain applications. ZKP plays an important role to perform computations off-chain and validate proofs on-chain, and has lead to massive adoptions of zkRollups and active developments of zkEVMs. Recently, the focus has shifted to ML functionalities allowing smart contracts to interact with ML models and use the result of ML inferences, see [1] and [3]. As generating ZKPs is very resource intensive, our scheme can be used to delegate the generation of ZKP for ML to cloud servers, while preserving the confidentiality of both the ML model and the users' data. A client with limited computing resource can obtain a ZKP from the cloud servers and post it on the blockchain, without revealing the sensitive information to the servers.

In this paper, we have made the following contributions:

- We proposed a confidential and verifiable outsourcing scheme for secure computations that are modeled as arithmetic circuits. The scheme is based on generic MPC protocols, such as SPDZ [26], and a ZKP scheme based on interactive proofs (IP) [21]. We designed a special MPC protocol to compute the messages of the IPs, such that most of the computations are done locally on each prover. This can greatly reduce the time to generate proof in a low bandwidth setting.
- We introduced a novel ZKP protocol for efficiently verifying matrix multiplication operations in ML. Given the result of the matrix multiplication of two $m \times m$ matrices computed via MPC, the additional computation and communication complexities to generate the proof are only $O(m^2)$ and $O(m)$, respectively. To the best of our knowledge, this is the first method that can generate the ZKP with less overhead and response time than those incurred when computing matrix multiplication over MPC.
- We implemented the system of confidential and verifiable ML delegation and evaluated its performance. Experiments show that our scheme improves the running time (the online prover time of MPC) of existing collaborative ZKP [35] by one to two orders of magnitude. Furthermore, for matrix multiplications, the running time of our scheme is completely dominated by the MPC evaluation. The additional time to ensure the integrity of the result using ZKP is only 2% of the total time. On a fully-connected neural network inference task with 3 layers, it only takes 2.6 seconds to run the full protocol, which is 88× faster than the prior work of [35]. On the convolutional neural network of Lenet [28], it takes 288 seconds for one inference, which is 74.8× faster than the prior work of [35].

1.1 Related Work

Papers that are mostly related to our work include [11, 12, 19, 24, 35, 39]. Schoenmakers et al. proposed a privacy-preserving verifiable computation scheme based on the SNARK [37] and an MPC protocol named Trinocchio [39]. Trinocchio assumes a circuit-specific trusted setup, while our scheme only requires a universal trusted setup. Another work, Kanjalkar et al. [24], introduced the concept of *auditable MPC* which proves to a third party that the results of an MPC protocol is correctly computed even if all the parties are malicious by generating the proof of the zkSNARK [10] using MPC. Concurrently, Ozdemir and Boneh [35] presented the notion of *collaborative ZK* based on Groth16 [22] and Plonk [18]. Later, Dayama et al. [12] considered the same setting and developed schemes based on interactive oracle proofs (IOP). More recently, Chiesa et al. [11] designed efficient schemes for zkSNARKs based on polynomial IOPs. Garg et al. [19] further improved the total running time utilizing multiple provers and the packed secret sharing techniques.

None of the above mentioned papers have studied schemes based on interactive proofs. Moreover, all of them assume that the *extended witness* (i.e., all values in the circuit) has been computed, secret-shared among parties, and only consider the step of proof generation. By contrast, we take the circuit evaluation using MPC into account and based on which we develop an end-to-end IP scheme. We show that our approach yields a small overhead as the time to compute the messages in IP is minimal compared to the time to evaluate the circuit in MPC. Table 1 shows the comparison among existing schemes excluding the communication overhead

of MPC. It is clear that our scheme has the best prover time which is linear in the size of the circuit. Moreover, the special protocol for matrix multiplication improves both the prover time and the communication by $O(m)$.

Concurrent work. In [31], Liu et al. proposed a collaborative ZK scheme based on the GKR protocol. The main protocol is similar to our approach for general circuits. They used packed secret sharing in order to achieve a faster running time, with a lower threshold on the number of malicious provers, while we use the standard additive sharing. Our main focus is on the machine learning delegations, and we propose a special protocol for matrix multiplications where the additional overhead to generate ZKP is small. Similar to [19, 35], the experiments in [31] does not consider the phase of circuit evaluation in MPC either.

Verifiable computation on encrypted data. An alternative solution to ensure both confidentiality and integrity is via verifiable computation on encrypted data by combining ZKPs with homomorphic encryptions [7, 15, 16, 20]. However, due to the heavy mechanism, they mainly remain theoretical and cannot scale to the ML computations considered in this paper.

There is a long line of work in the literature on privacy-preserving ML [23, 27, 33, 34] and zero-knowledge proofs for ML [13, 30, 45]. Nonetheless, schemes for confidential and verifiable ML inference based on special-purpose protocols to prove specific functions, like matrix multiplications have not been considered yet.

2 PRELIMINARIES

Let $\mathbb{F}_p$ be a finite field mod p. In our paper, we use capital letters, such as A to represent matrices and bold lower-case letters, such as $\mathbf{a}$ to represent vectors. We also use $f()$ to represent polynomial f. PPT stands for probabilistic polynomial time.

Bilinear pairings. Let $\mathbb{G}_1$,$\mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups of prime order q. Let $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ denote a bilinear map which maps two group elements from $\mathbb{G}_1$, $\mathbb{G}_1$ to target group from $\mathbb{G}_T$. Bilinear pairing satisfies the following properties:

- **Bilinearity:** $e(P^a, Q^b) = e(P, Q)^{ab}$ for all $P, Q \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$.
- **Non-degeneracy:** $e(g, g) \neq 1$,$g \in \mathbb{G}_1$.
- **Computability:** There is an efficient algorithm to compute $e(P, Q)$ for all $P, Q \in \mathbb{G}$.

2.1 Interactive Proofs

Our scheme is built upon ZKPs based on interactive proofs and polynomial commitments, and we describe the protocols in this section.

2.1.1 The Sumcheck Protocol. The sumcheck protocol is a fundamental interactive proof protocol in which the prover $\mathcal{P}$ tries to convince the verifier $\mathcal{V}$ that the sum H of a polynomial $f : \mathbb{F}^\ell \rightarrow \mathbb{F}$ on a binary hypercube is

$$H = \sum_{b_1, b_2, \ldots, b_\ell \in \{0,1\}} f(b_1, b_2, \ldots, b_\ell)$$

Protocol 1 (**Sumcheck protocol**). *The protocol proceeds in ℓ rounds.*

- *In the first round, $\mathcal{P}$ sends $\mathcal{V}$ a univariate polynomial*

$$g_1(x_1) \stackrel{def}{=} \sum_{b_2, \ldots, b_\ell \in \{0,1\}} g(x_1, b_2, \ldots, b_\ell),$$

  $\mathcal{V}$ checks $H = g_1(0) + g_1(1)$. Then $\mathcal{V}$ sends $\mathcal{P}$ a random challenge $r_1 \in \mathbb{F}$.
- *In the ith round, $\mathcal{P}$ sends $\mathcal{V}$ a univariate polynomial*

$$g_i(x_i) \stackrel{def}{=} \sum_{b_{i+1}, \ldots, b_\ell \in \{0,1\}} g(r_1, \ldots, r_{i-1}, x_i, b_{i+1}, \ldots, b_\ell),$$

  $\mathcal{V}$ checks $g_{i-1}(r_{i-1}) = g_i(0) + g_i(1)$, and sends $\mathcal{P}$ a random challenge $r_i \in \mathbb{F}$.
- *In the ℓth round, $\mathcal{P}$ sends a univariate polynomial*

$$g_\ell(x_\ell) \stackrel{def}{=} g(r_1, r_2, \ldots, r_{l-1}, x_\ell),$$

  $\mathcal{V}$ checks $g_{\ell-1}(r_{\ell-1}) = g_\ell(0) + g_\ell(1)$.
- *In the final round, $\mathcal{V}$ generates a random challenge $r_\ell \in \mathbb{F}$. and accept if and only if $g_\ell(r_\ell) = g(r_1, r_2, \ldots, r_\ell)$.*

Notice that computing this sum directly by $\mathcal{V}$ will take 2^ℓ work. The sumcheck protocol delegates this work to $\mathcal{P}$ and allows $\mathcal{P}$ to generate the proof in ℓ rounds as shown in Protocol 1.

2.1.2 The GKR Protocol. The GKR protocol [21] is an interactive proof protocol for layered circuits, which uses the sumcheck protocol as a building block. Let C be an arithmetic circuit with depth d over a finite field $\mathbb{F}$. The circuit is layered in such a way that the output wire from a gate at layer i can only be an input wire to a gate at layer $i-1$. The GKR protocol starts from the layer 0 (output layer) and proves the correctness of the circuit computation one layer at a time, until it reaches layer d (input layer). Specifically, $\mathcal{P}$ first sends a claim of the output to $\mathcal{V}$. Then, $\mathcal{P}$ reduces the claim of the output layer to a claim of wire value at the layer below using sumcheck protocol. In the i-th round, $\mathcal{P}$ will generally reduce the claim about the values of wires at the ith layer to a claim about wire values at the $(i+1)$-th layer.

Formally, we use S_i to denote the number of gates at layer i, and $s_i = \lceil log(S_i) \rceil$ denotes the number of bits that can represent all the gates in layer i. We also define a function $V_i : \{0,1\}^{s_i} \rightarrow \mathbb{F}$ that maps an input gate's index to its wire value. We also define two functions $add_i, mult_i : \{0,1\}^{s_{i-1}+2s_i} \rightarrow \{0,1\}$ that are called wiring predicates. For example, add_i takes three gates (a, b, c) such that a is from layer $i-1$ and b, c are from layer i. add_i outputs 1 if and only if a is an addition gate whose 2 inputs are outputs from gate b and c ($mult_i$ is defined similarly). With all these definitions, V_i can be written as:

$$V_i(z) = \sum_{x,y \in \{0,1\}^{s_{i+1}}} (add_{i+1}(g, x, y)(V_{i+1}(x) + V_{i+1}(y)) + mult_{i+1}(g, x, y)(V_{i+1}(x)V_{i+1}(y))) \quad (1)$$

for any $z \in \{0,1\}^{s_i}$.

Since equation V_i is written in a summation form, we can use sumcheck protocol to verify its correctness. Therefore we need to use the multilinear extension of the above equation, because sumcheck only works on finite field $\mathbb{F}$.

Definition 1 (**Multi-linear Extension**). *Let $V : \{0,1\}^\ell \to \mathbb{F}$ be a function. We denote $\tilde{V} : \mathbb{F}^\ell \to V$ as the multilinear extension of V such that $\tilde{V}$ and V agree on the binary hypercube $(x_1, x_2, \ldots, x_\ell) \in \{0,1\}^\ell$.*

The closed-form of $\tilde{V}$ is

$$\tilde{V}(g) = \sum\nolimits_{y \in \{0,1\}^\ell} \tilde{I}(g,y) V(y) \tag{2}$$

for all $g \in \mathbb{F}^\ell$, where $\tilde{I}(g,y)$ is the identity function such that $\tilde{I}(g,y) = 1$ if and only if $g = y$ for all $g \in \{0,1\}^\ell$. In particular, $\tilde{I}(g,y) = \prod_{i=1}^{\ell}(g_i y_i + (1-g_i)(1-y_i))$.

With the definition of multi-linear extensions, we have

$$\begin{aligned}\tilde{V}_i(g) &= \sum\nolimits_{x,y \in \{0,1\}^{s_{i+1}}} f_i(x,y) \\ &= \sum\nolimits_{x,y \in \{0,1\}^{s_{i+1}}} (\tilde{add}_{i+1}(g,x,y)(\tilde{V}_{i+1}(x) + \tilde{V}_{i+1}(y)) \\ &\quad + \tilde{mult}_{i+1}(g,x,y)(\tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y)))\end{aligned} \tag{3}$$

where $\tilde{V}_i$, $\tilde{add}_i$, and $\tilde{mult}_i$ are multilinear extensions of V_i, add_i, and $mult_i$ respectively.

The GKR protocol works as follows. The prover $\mathcal{P}$ initially sends the claimed output of the circuit to $\mathcal{V}$. Specifically, $\mathcal{P}$ defines a polynomial $V_0(x)$ based on the output wires' value. Then $\mathcal{V}$ takes $V_0(x)$ and computes $\tilde{V}_0(g)$ for $g \in \mathbb{F}^{s_0}$. Then, $\mathcal{P}$ and $\mathcal{V}$ engage in a sumcheck protocol to check this equation. The sumcheck in the ith layer takes $s_i + 2s_{i+1}$ rounds of interaction due to the number of variables in the polynomial. Recall at the end of each sumcheck, $\mathcal{V}$ needs to evaluate the polynomial $f_i(x,y)$ with randomness $w_1, w_2 \in \mathbb{F}^{2s_{i+1}}$, where w_1, w_2 are randomness selected in the sumcheck protocol. $\mathcal{V}$ can compute the wire predicate $\tilde{add}_i$ and $\tilde{mult}_i$ given they are public. $\mathcal{V}$ will need oracle access to values $\tilde{V}_{i+1}(w_1)$ and $\tilde{V}_{i+1}(w_2)$ from $\mathcal{P}$ in order to compute $f_i(w_1, w_2)$.

Combining two claims randomly. In the above protocol, observe that each sumcheck protocol reduces one claim into two claims. If we keep this protocol as it is, every sumcheck in one layer will induce two sumchecks in the following, resulting in an exponential increase of the numbers of sumcheck uses in GKR. Hence, we need to reduce these two claims into one before we proceed into the next layer. In Chiesa et al. [9], the authors proposed an approach using random linear combination to combine these two claims. After receiving both claims $\tilde{V}_i(w_1)$ and $\tilde{V}_i(w_2)$, $\mathcal{V}$ picks two randomness $\alpha_i, \beta_i \in \mathbb{F}$ and combine them as follows:

$$\begin{aligned}&\alpha_i \tilde{V}_i(w_1) + \beta_i \tilde{V}_i(v) \\ =&\alpha_i \sum_{x,y \in \{0,1\}^{s_{i+1}}} (\tilde{add}_{i+1}(w_1,x,y)(\tilde{V}_{i+1}(x) + \tilde{V}_{i+1}(y)) \\ &+ \tilde{mult}_{i+1}(w_1,x,y)(\tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y))) \\ +&\beta_i \sum_{x,y \in \{0,1\}^{s_{i+1}}} (\tilde{add}_{i+1}(w_2,x,y)(\tilde{V}_{i+1}(x) + \tilde{V}_{i+1}(y)) \\ &+ \tilde{mult}_{i+1}(w_2,x,y)(\tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y))) \\ =&\sum_{x,y \in \{0,1\}^{s_{i+1}}} ((\alpha_i \tilde{add}_{i+1}(w_1,x,y) + \beta_i \tilde{add}_{i+1}(w_2,x,y)) \\ &\times (\tilde{V}_{i+1}(x) + \tilde{V}_{i+1}(y)) \\ +&(\alpha_i \tilde{mult}_{i+1}(w_1,x,y) + \beta_i \tilde{mult}_{i+1}(w_2,x,y))(\tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y))\end{aligned} \tag{4}$$

The sumcheck protocol uses the combined Equation in 4 instead of Equation 3. Therefore, we reduce one claim into another claim.

2.1.3 Polynomial commitments.

Definition 2. *A Polynomial commitment is a tuple* (KeyGen, Commit, Open, Verify) *of PPT algorithms where:*

- $\mathsf{KeyGen}(1^\lambda, \mathcal{F}) \to \mathsf{pp}$*: The algorithm generates public parameters* (pp)*;*
- $\mathsf{Commit}(f, \mathsf{pp}) \to \mathsf{com}_f$*: The algorithm takes a secret polynomial $f(\mathbf{X})$ where $X = (X_0, \ldots, X_{\mu-1})$ and outputs a public commitment com_f;*
- $\mathsf{Open}(f, \mathbf{x}, \mathsf{pp}) \to (z, \pi_f)$*: The algorithm evaluates the polynomial $y = f(\mathbf{X})$ on a point $\mathbf{x}$ and generate a proof π_f;*
- $\mathsf{Verify}(\mathsf{com}_f, \mathbf{x}, z, \pi_f, \mathsf{pp}) \to b \in \{1, \emptyset\}$*: The algorithm verifies whether $f(\mathbf{x}) = z$ using* pp, com_f *and π_f;*

which satisfies the following properties:

- ***Completeness.*** *For any polynomial $f \in \mathcal{F}$ and $\mathbf{x} \in \mathbb{F}^\mu$, the following probability is* 1.

$$\Pr\left[\mathsf{Verify}(\mathsf{com}_f, \mathbf{x}, z, \pi_h, \mathsf{pp}) = 1 \;\middle|\; \begin{array}{r}\mathsf{pp} \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{F}) \\ : \mathsf{com}_f \leftarrow \mathsf{Commit}(f, \mathsf{pp}) \\ (z, \pi_f) \leftarrow \mathsf{Open}(f, \mathbf{x}, \mathsf{pp})\end{array}\right]$$

- ***Knowledge soundness.*** *For any PPT adversary $\mathcal{P}^*$, there exists a PPT extractor ϵ with access to $\mathcal{P}^*$'s messages during the protocol, the following probability is* $\mathsf{negl}(\lambda)$.

$$\Pr\left[\begin{array}{r}\mathsf{Verify}(\mathsf{com}^*, \mathbf{x}^*, z^*, \pi^*, \mathsf{pp}) = 1 \quad \mathsf{pp} \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{F}) \\ \wedge \mathsf{com}^* = \mathsf{Commit}(f^*, \mathsf{pp}) \;\; : (z^*, \mathbf{x}^*) \leftarrow \mathcal{P}^*(1^\lambda, \mathsf{pp}) \\ \wedge f^*(\mathbf{x}^*) \neq z^* \quad (com^*, \pi^*) \leftarrow \mathcal{P}^*(1^\lambda, \mathsf{pp}) \\ f^* \leftarrow \epsilon^{\mathcal{P}^*(\cdot)}(1^\lambda, \mathsf{pp})\end{array}\right]$$

- ***Zero-knowledge.*** *For all efficient adversary $\mathcal{A}$, there exists a simulator* Sim *such that for all* $\mathsf{KeyGen}(1^\lambda, \mathcal{F}) \to \mathsf{pp}$ *and all efficient*

distinguisher D

$$\Pr\begin{bmatrix} \mathcal{A}(\mathsf{pp}) \to f \\ \mathsf{Sim}(\mathsf{pp}) \to \mathsf{com}_f \ : D(z, \pi_f, \mathsf{com}_f) = 1 \\ \mathcal{A}(\mathsf{com}_f, \mathsf{pp}) \to \mathbf{x} \\ \mathsf{Sim}(\mathsf{pp}, z = f(\mathbf{x})) \to (z, \pi_f) \end{bmatrix}$$

$$- \Pr\begin{bmatrix} \mathcal{A}(\mathsf{pp}) \to f \\ \mathsf{Commit}(f, \mathsf{pp}) \to \mathsf{com}_f \ : D(z, \pi_f, \mathsf{com}_f) = 1 \\ \mathcal{A}(\mathsf{com}_f, \mathsf{pp}) \to \mathbf{x} \\ \mathsf{Open}(f, \mathbf{x}, \mathsf{pp}) \to (z, \pi_f) \end{bmatrix} \leq \mathsf{negl}(\lambda)$$

As shown in prior work [42, 43, 47], one can build a ZKP scheme by combining the GKR protocol with a polynomial commitment scheme. The prover commits to the witness of the circuit, executes the GKR protocol with the verifier, and finally opens the polynomial at a random point to check the correctness of the last claim of the GKR protocol. In this paper, we use the KZG polynomial commitment [25] and we present the protocol in Section 3.6. We denote this scheme as $(\mathsf{ZKP}.\mathcal{P}, \mathsf{ZKP}.\mathcal{V})$.

2.2 Secure Multiparty Computation

Secure multiparty computation [32, 44] allows a set of parties $\{P_1, \ldots, P_n\}$ to jointly compute a function f over their inputs, and reveals nothing other than the output. In this paper, we use MPC protocols based on a linear secret sharing scheme and primarily focus on additive sharing. In an additive secret sharing scheme, a value a is split into $(\langle a \rangle_1, \langle a \rangle_2, \ldots, \langle a \rangle_n)$ and distributed among all n parties such that $a = \langle a \rangle_1 + \langle a \rangle_2 + \ldots + \langle a \rangle_n \mod p$ in a field $\mathbb{F}_p$.

- $\mathsf{MPC.Add}(\langle a \rangle, \langle b \rangle)$: Given the shares of two values $\langle a \rangle$ and $\langle b \rangle$, their addition $c = a + b$ can be computed locally by the parties as $\langle c \rangle = \langle a \rangle + \langle b \rangle$ without any interaction because of the linearity of the secret sharing scheme. Similarly, subtraction and multiplication by a public value can also be computed locally.
- $\mathsf{MPC.Mult}(\langle a \rangle, \langle b \rangle)$: To compute the multiplication of two shared values, one can use Beaver's multiplication triplets [6]. In the offline phase, the parties precompute a multiplication triplet $\langle u \rangle, \langle v \rangle, \langle w \rangle$ such that $u \cdot v = w$. Then in the online phase, the parties exchange $\langle a \rangle + \langle u \rangle$ and $\langle b \rangle + \langle v \rangle$ and reconstruct $e = a+u, f = b+v$. Finally, each party sets $\langle c \rangle = e \cdot \langle b \rangle - f \cdot \langle u \rangle + \langle w \rangle$ locally, and it can be shown that $c = a \cdot b$.
- $\mathsf{MPC.InnerProduct}(\langle \mathbf{a} \rangle, \langle \mathbf{b} \rangle)$: The inner product of two shared vectors can be computed by applying MPC.Mult element-wise and then adding the shares locally.
- $\mathsf{MPC.ScalarProduct}(\langle \mathbf{a} \rangle, \langle b \rangle)$: The protocol multiplies every element of a shared vector by a shared value. It is realized using MPC.Mult element-wise.

3 CONFIDENTIAL AND VERIFIABLE DELEGATION

In this section, we present our confidential and verifiable delegation scheme. Section 3.1 gives the security definition of the scheme, and Section 3.2 gives an overview of our scheme combining MPC and ZKP. Section 3.3 reviews the algorithms that generates the proofs of GKR in linear time as in [43]. We present our scheme for general circuits in Section 3.4, and for matrix multiplications in Section 3.5. We describe the polynomial commitment scheme on shared data in Section 3.6.

3.1 Security Definitions

We first propose the security definition of a confidential and verifiable delegation scheme. The definition is adapted from collaborative zk-SNARKs proposed in [35].

Definition 3. *A confidential and verifiable delegation scheme for n provers, a verifier and a computation modeled as an arithmetic circuit C consists of 4 algorithms (*Setup, Share, Prove, Verify*):*

- $\mathsf{Setup}(1^\lambda, C) \to \mathsf{pp}$*: This algorithm outputs the public parameters.*
- $\mathsf{Share}(w) \to \langle w \rangle$*: This algorithm is called by the verifier to share the witness with the provers.*
- $\mathsf{Prove}(\langle w \rangle, C, x, \mathsf{pp}) \to (y, \pi)$*: This algorithm allows the provers to jointly evaluate the circuit C and output the result y and a proof π.*
- $\mathsf{Verify}(y, x, \pi, \mathsf{pp}) \to \{0, 1\}$ *This algorithm enables the verifier to verify the result and outputs 1 or 0.*

Completeness*: For all provers,*

$$\Pr\begin{bmatrix} \mathsf{Setup}(1^\lambda, C) \to \mathsf{pp} \\ \mathsf{Share}(w) \to \langle w \rangle \ : \mathsf{Verify}(y, x, \pi, \mathsf{pp}) \to 1 \\ \mathsf{Prove}(\langle w \rangle, C, x, \mathsf{pp}) \to (y, \pi) \end{bmatrix} = 1$$

Knowledge Soundness*: For all x, for all sets of efficient algorithms* $\vec{P} = \{P_1, \ldots, P_n\}$*, there exists an efficient extractor ε:*

$$\Pr\begin{bmatrix} \mathsf{Setup}(1^\lambda, C) \to \mathsf{pp} & \\ \vec{P}(C, x, \mathsf{pp}) \to (y, \pi) \ : & \mathsf{Verify}(y, x, \pi, \mathsf{pp}) \to 1 \\ \varepsilon(x, \pi, \mathsf{pp}) \to w & \wedge y \neq C(x, w) \end{bmatrix} \leq \mathsf{negl}(\lambda)$$

t-Zero-Knowledge*: For all efficient adversary* $\mathcal{A}$ *corrupting* $k \leq t$ *provers* $P_{i_1}, \ldots, P_{i_k}$*, there exists a simulator* Sim *such that for all x, w and C and all efficient distinguisher D*

$$\Pr\begin{bmatrix} \mathsf{Setup}(1^\lambda, C) \to \mathsf{pp} \\ \mathsf{Sim}(x, y, \langle w \rangle_{i_1}, \ldots, \langle w \rangle_{i_k}, \mathsf{pp}) \to \mathsf{tr} \ : D(\mathsf{tr}) = 1 \end{bmatrix}$$
$$- \Pr\begin{bmatrix} \mathsf{Setup}(1^\lambda, C) \to \mathsf{pp} \\ \mathsf{View}_{\mathcal{A}}(x, w) \to \mathsf{tr} \ : D(\mathsf{tr}) = 1 \end{bmatrix} \leq \mathsf{negl}(\lambda)$$

where tr *is a transcript,* $\mathsf{View}_{\mathcal{A}}(x, w)$ *denotes the view of* $\mathcal{A}$ *when provers* $P_1, ..., P_n$ *interact with input x and witnesses w.*

In Definition 3 above, the knowledge soundness guarantees that as long as the provers can generate a valid proof passing the verification, there exists an extractor that extracts the witness w such that $C(w, x) = y$ with an overwhelming probability. In other words, the protocol guarantees the integrity of the result y even if all provers are malicious, which is beyond the capability of malicious MPC. The

Protocol 2 (Confidential and Verifiable Delegation). *Let C be a layered arithmetic circuit. Let w be the witness and x be the public input of C. The protocol is between a verifier $\mathcal{V}$ and n provers $P_1, \ldots, P_n$.*

- $\mathsf{Setup}(1^\lambda)$*: run the* KeyGen *algorithm of the polynomial commitment in and output* pp.
- $\mathsf{Share}(w)$*: $\mathcal{V}$ shares w to all the provers $P_1, \ldots, P_n$ using a linear secret sharing scheme.*
- $\mathsf{Commit}(\langle w \rangle, x)$*: each prover defines the multilinear extension of $\langle w \rangle$ concatenated with $\langle x \rangle$ as $\langle \tilde{V}_d \rangle$. Each prover runs* $\mathsf{Commit}(\langle \tilde{V}_d \rangle, \mathsf{pp})$*. Output the combined commitment* com.
- $\mathsf{Prove}(\langle w \rangle, C, x, \mathsf{pp})$
  1. *The provers execute an MPC protocol to evaluate the circuit to obtain $y = C(x, w)$ and the shares of each wire value in the circuit.*
  2. *The provers run the GKR protocol on shares to generate the proof.*
  3. *For the input layer, the provers run* $\mathsf{Open}(\langle \tilde{V}_d \rangle, r_d, \mathsf{pp})$*, where r_d is the last random challenge for the input layer of the GKR protocol.*
  4. *The provers output y and all the proofs generated during the protocol.*
- $\mathsf{Verify}(\mathsf{com}, y, \pi, \mathsf{pp})$*: $\mathcal{V}$ verifies the proof of the GKR protocol and the proof of the polynomial commitment protocol. If all checks pass, output 1; otherwise, output 0.*

t-zero-knowledge guarantees that the proof leaks no information about the secret witness w even if the adversary corrupts up to t provers. In the application of ML inference, w includes both the ML model and the data sample.

3.2 Overview of Our Scheme

In order to build a confidential and verifiable delegation scheme, we combine the cryptographic primitives of MPC and ZKP. As shown in Protocol 2, the verifier first secret-shares the witness to all the provers. The provers then commit to the witness using a polynomial commitment on shared values. This step is necessary to derive randomness using the Fiat-Shamir heuristic [14] to make the protocol non-interactive, and also to allow the provers to have additional witness. Later, to perform the computation and generate a proof, the provers execute an MPC protocol to compute y without revealing w, and generate the proof using the shares of wire values in the circuit. As we are using the ZKP based on GKR and polynomial commitments, this step is divided into generating GKR proofs and polynomial commitment proofs on shared values in MPC.

The framework is similar to the constructions in [19, 35], where they use MPC to generate ZKPs based on zkSNARKs and Plonk. However, we are the first to consider interactive proofs and achieve better prover efficiency. Moreover, none of prior work considers the MPC step to evaluate the circuit. In the following, we present our protocols for GKR and polynomial commitments on shared values.

Algorithm 1 Phase 1

Input: Multilinear extensions $\tilde{V}_{i+1}$ and randomness $g = g_1, \ldots, g_{s_{i+1}}, u = u_1, \ldots, u_{s_{i+1}}$;
Output: Proofs of the sumcheck protocol $a_1, \ldots, a_{s_{i+1}}$, where each a_i contains 3 evaluations uniquely representing a degree 2 polynomial;

procedure $\mathbf{g} \leftarrow \mathsf{Precompute}(g)$ ▷ $\mathbf{g}$ is an array of size $2^{s_{i+1}}$.
  Set $\mathbf{g}[0] = 1$
  for $i = 0, \ldots, s_{i+1} - 1$ **do**
    for $b \in \{0,1\}^i$ **do**
      $\mathbf{g}[b, 0] = \mathbf{g}[b] \cdot (1 - g_{i+1})$
      $\mathbf{g}[b, 1] = \mathbf{g}[b] \cdot g_{i+1}$
  return $\mathbf{g}$;
procedure $\mathbf{h} \leftarrow \mathsf{Initialize\_Phase1}(\tilde{V}_{i+1}, \mathbf{g}))$
  $\forall x \in \{0,1\}^{s_{i+1}}$, set $\mathbf{h}[x] = 0$
  for every multiplication gate (z, x, y) such that $\tilde{mult}_{i+1}(z, x, y) = 1$ **do**
    $\mathbf{h}[x] = \mathbf{h}[x] + \mathbf{g}[z] \cdot \tilde{V}_{i+1}(y)$
  return $\mathbf{h}$;
procedure $(a_1, \ldots, a_{s_{i+1}}) \leftarrow \mathsf{Sumcheckproof}(\mathbf{h}, \tilde{V}_{i+1}, u)$
  Set vector $\mathbf{v}[b] = \tilde{V}_{i+1}(b)$ for all $b \in \{0,1\}^{s_{i+1}}$
  Set $a_{t,i} = 0$ for $i = 1, \ldots, s_{i+1}$ and $t = 0, 1, 2$
  for each round $i = 1, \ldots, s_{i+1}$ **do**
    for $b \in \{0,1\}^{s_{i+1}-i}$ **do** ▷ we use b as both a number and its binary representation
      for $t = 0, 1, 2$ **do**
        $a_{t,i} = a_{t,i} + (\mathbf{h}[b](1-t) + \mathbf{h}[b + 2^{s_{i+1}-i}]t)(\mathbf{v}[b](1-t) + \mathbf{v}[b + 2^{s_{i+1}-i}]t)$
      $\mathbf{h}[b] = \mathbf{h}[b](1 - u_i) + \mathbf{h}[b + 2^{s_{i+1}-i}]u_i$
      $\mathbf{v}[b] = \mathbf{v}[b](1 - u_i) + \mathbf{v}[b + 2^{s_{i+1}-i}]u_i$
  return $a_1, \ldots, a_{s_{i+1}}$.

3.3 GKR with a linear prover

We first describe the state-of-the-art algorithms to generate the GKR proofs in the plain setting without MPC, as proposed by Xie et al.in [43].

Recall that for each layer of the circuit, $\mathcal{P}$ and $\mathcal{V}$ needs to execute the sumcheck protocol on Equation 3. It can be divided into the sum of three terms $\tilde{add}_{i+1}(g, x, y)\tilde{V}_{i+1}(x)$, $\tilde{add}_{i+1}(g, x, y)\tilde{V}_{i+1}(y))$ and $\tilde{mult}_{i+1}(g, x, y)\tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y)$. For simplicity, we describe the algorithms for the third term consisting of the product of three polynomials:

$$\sum\nolimits_{x,y \in \{0,1\}^{s_{i+1}}} \tilde{mult}_{i+1}(g, x, y)\tilde{V}_{i+1}(x)\tilde{V}_{i+1}(y)\,, \tag{5}$$

and the first two terms are special cases with only two polynomials. Notice that directly executing the sumcheck protocol would take quadratic time because $\tilde{mult}_{i+1}(g, x, y)$ contains $2^{2s_{i+1}} = S_{i+1}^2$ variables. In [43], the sumcheck was divided into two phases where

Algorithm 2 Phase 2

Input: Multilinear extensions $\tilde{V}_{i+1}$, evaluation $\tilde{V}_{i+1}(u)$ from Phase 1, and randomness
$g = g_1, \ldots, g_{s_{i+1}}, u = u_1, \ldots, u_{s_{i+1}}, v = v_1, \ldots, v_{s_{i+1}}$;
Output: Proofs of the sumcheck protocol $a'_1, \ldots, a'_{s_{i+1}}$;

1: $\mathbf{g} \leftarrow$ Precompute(g)
2: $\mathbf{u} \leftarrow$ Precompute(u)
3: **procedure** $\mathbf{m} \leftarrow$ Initialize_Phase2$(\mathbf{g}, \mathbf{u})$)
4: $\quad \forall y \in \{0,1\}^{s_{i+1}}$, set $\mathbf{m}[y] = 0$
5: $\quad$ **for** every multiplication gate (z, x, y) such that $\tilde{mult}_{i+1}(z, x, y) = 1$ **do**
6: $\quad\quad \mathbf{m}[y] = \mathbf{m}[y] + \mathbf{g}[z] \cdot \mathbf{u}[x]$
7: $\quad$ **return m**;
8: Compute the scalar product $\tilde{V}_{i+1}(u) \cdot \tilde{V}_{i+1}(y)$ for all $y \in \{0,1\}^{s_{i+1}}$
9: $(a'_1, \ldots, a'_{s_{i+1}}) \leftarrow$ Sumcheckproof$(\tilde{V}_{i+1}(u) \cdot \tilde{V}_{i+1}, \mathbf{m}), v)$
10: **return** $a'_1, \ldots, a'_{s_{i+1}}$.

Equation 5 can be written as:

$$\sum_{x \in \{0,1\}^{s_{i+1}}} \tilde{V}_{i+1}(x) \sum_{y \in \{0,1\}^{s_{i+1}}} \tilde{mult}_{i+1}(g, x, y) \tilde{V}_{i+1}(y) = \sum_{x \in \{0,1\}^{s_{i+1}}} \tilde{V}_{i+1}(x) h_{i+1}(x) \tag{6}$$

where $h_{i+1}(x) = \sum_{y \in \{0,1\}^{s_{i+1}}} \tilde{mult}_{i+1}(g, x, y) \tilde{V}_{i+1}(y)$.

Phase 1. In this phase, $\mathcal{P}$ executes the sumcheck protocol on Equation 6 summing over $x \in \{0,1\}^{s_{i+1}}$. It has been shown by Justin Thaler in [41] that given the evaluations of two multilinear polynomials on the Boolean hypercube (i.e., $\forall x \in \{0,1\}^{s_{i+1}}$), the sumcheck protocol can be executed with a linear prover time using a dynamic programming algorithm. The evaluations of $\tilde{V}_{i+1}$ are known by the prover as the values of the circuit, but the evaluations of $h_{i+1}(x)$ is not clear. Therefore, [43] proposed an initialization algorithm to compute them in linear time, and the full algorithm is shown in Algorithm 1. Procedure Precompute generates a vector of linear size using the randomness of the sumcheck protocol. This is exactly the evaluation of the identity polynomial $\tilde{I}$ such that $\mathbf{g}[b] = \tilde{I}(g, b)$ for all $b \in \{0,1\}^{s_{i+1}}$. It is then used together with $\tilde{V}_{i+1}$ to compute the evaluations of h_{i+1} on the Boolean hypercube as a vector $\mathbf{h}$. Finally, procedure Sumcheckproof is the dynamic programming algorithm in [41] generating the proof in each round using the vectors $\mathbf{h}$ and $\mathbf{v}$ and updating them using the randomness in the next round. The running time of all three procedures is linear in the size of the circuit $O(S_{i+1})$, and therefore the prover time of Phase 1 is linear.

Phase 2. After Phase 1, "variable x has been fixed to randomness u and the remaining element in $\mathbf{v}$ is $\mathbf{v}[0] = \tilde{V}_{i+1}(u)$. The remaining task in phase 2 is to run the sumcheck protocol for the following equation:

$$\sum_{y \in \{0,1\}^{s_{i+1}}} \tilde{mult}_{i+1}(g, u, y) \tilde{V}_{i+1}(u) \tilde{V}_{i+1}(y) \tag{7}$$

Note that $\tilde{V}_{i+1}(u)$ is a single value and Equation 7 is again the product of two multilinear polynomials. [43] proposed another algorithm to compute the evaluations of $\tilde{mult}_{i+1}(g, u, y)$ on the Boolean hypercube $\forall y \in \{0,1\}^{s_{i+1}}$ in linear time, and the proofs can be then generated by the same dynamic programming algorithm in [41]. The algorithm of Phase 2 is presented in Algorithm 2 using the Precompute and Sumcheckproof procedures in Algorithm 1.

3.4 GKR in MPC for General Circuits

In this section,we present our protocol for the provers to compute the proofs of GKR when the values are secret-shared. We follow the two-phase approach in [43] and observe that most of the steps in Algorithm 1 and 2 are linear. For example, as the randomness is either sent by the verifier in the interactive setting or computed using Fiat-Shamir [14] in the non-interactive setting, it is known by all the provers. Therefore, Procedure Precompute(g) in Algorithm 1 can be computed locally by each server. Moreover, in the initialization procedure, e.g., line 11 of Algorithm 1, the equation is a linear function of $\tilde{V}_{i+1}(y)$ as $\mathbf{g}$ is a public vector. Similarly, the equations to update the vectors in line 20 and 21 are linear as well. The only place that requires an MPC protocol is line 19 multiplying a value computed from $\mathbf{h}$ and a value computed from $\mathbf{v}$, which are both secret-shared. We use an MPC protocol for inner product to perform this step. We present the protocol of Phase 1 for each server in Algorithm 3. The shared values are indicated using $\langle\rangle$ and we omit the subscript of the server in the algorithm. As shown in the algorithm, only step 15 and 16 are executed using an MPC protocol and all other steps are computed locally. At the end of this phase, the provers hold the shares of $\langle \tilde{V}_{i+1}(u) \rangle = \langle \mathbf{v} \rangle [0]$.

After Phase 1, the provers execute Phase 2 on shared values and the algorithm is shown in 4. Surprisingly, all the steps can be done locally after a scalar multiplication in MPC. This is because in Phase 2, the initialization only depends on $\tilde{mult}_{i+1}$, which is defined by the circuit and is known to each prover. Thus the vector $\mathbf{m}$ is publicly known by all the provers. After the scalar multiplication in Step 8, the remaining Sumcheckproof procedure is executed on a shared vector and a public vector, and thus no MPC protocol is involved at all.

Combining two points randomly. Recall that as explained in Section 2.1.2, at the end of the sumcheck, $\mathcal{V}$ needs to combine two evaluations of $\tilde{V}_{i+1}$ via a random linear combination. Upon receiving the two claims $\tilde{V}_i(u)$ and $\tilde{V}_i(v)$, $\mathcal{V}$ selects $\alpha_i, \beta_i \in \mathbb{F}$ randomly and computes $\alpha_i \tilde{V}_i(u) + \beta_i \tilde{V}_i(v)$. Using random linear combination, the equation can be written as

$$\begin{aligned} &\alpha_i \tilde{V}_i(u) + \beta_i \tilde{V}_i(v) \\ =&\alpha_i \sum_{x,y \in \{0,1\}^{s_{i+1}}} (\tilde{add}_{i+1}(u, x, y)(\langle \tilde{V}_{i+1} \rangle(x) + \langle \tilde{V} \rangle_{i+1}(y)) \\ &+ \tilde{mult}_{i+1}(u, x, y)(\langle \tilde{V}_{i+1} \rangle(x) \langle \tilde{V}_{i+1} \rangle(y))) \\ +&\beta_i \sum_{x,y \in \{0,1\}^{s_{i+1}}} (\tilde{add}_{i+1}(v, x, y)(\langle \tilde{V}_{i+1} \rangle(x) + \langle \tilde{V}_{i+1} \rangle(y)) \\ &+ \tilde{mult}_{i+1}(v, x, y)(\langle \tilde{V}_{i+1} \rangle(x) \langle \tilde{V}_{i+1} \rangle(y))) \end{aligned} \tag{8}$$

Algorithm 3 Phase 1 in MPC

Input: Shared multilinear extension $\langle \tilde{V}_{i+1} \rangle$ and randomness $g = g_1, \ldots, g_{s_{i+1}}, u = u_1, \ldots, u_{s_{i+1}}$;
Output: Proofs of the sumcheck protocol $a_1, \ldots, a_{s_{i+1}}$, where each a_i contains 3 evaluations uniquely representing a degree 2 polynomial;

1: $\mathbf{g} \leftarrow$ Precompute(g)
2: **procedure** $\langle \mathbf{h} \rangle \leftarrow$ Initialize_Phase1($\langle \tilde{V}_{i+1} \rangle, \mathbf{g}$))
3: $\forall x \in \{0,1\}^{s_{i+1}}$, set $\langle \mathbf{h} \rangle[x] = 0$
4: **for** every multiplication gate (z, x, y) such that $\tilde{mult}_{i+1}(z, x, y) = 1$ **do**
5: $\langle \mathbf{h} \rangle[x] = \langle \mathbf{h} \rangle[x] + \mathbf{g}[z] \cdot \langle \tilde{V}_{i+1}(y) \rangle$
6: **return** $\langle \mathbf{h} \rangle$;
7: **procedure** $(a_1, \ldots, a_{s_{i+1}}) \leftarrow$ Sumcheckproof($\langle \mathbf{h} \rangle, \langle \tilde{V}_{i+1} \rangle, u$)
8: Set vector $\langle \mathbf{v} \rangle[b] = \langle \tilde{V}_{i+1}(b) \rangle$ for all $b \in \{0,1\}^{s_{i+1}}$
9: **for** each round $i = 1, \ldots, s_{i+1}$ **do**
10: **for** $b \in \{0,1\}^{s_{i+1}-i}$ **do** //we use b as both a number and its binary representation
11: **for** $t = 0, 1, 2$ **do**
12: Compute $\langle \mathbf{h} \rangle[b](1-t) + \langle \mathbf{h} \rangle[b + 2^{s_{i+1}-i}]t$ and $\langle \mathbf{v} \rangle[b](1-t) + \langle \mathbf{v} \rangle[b + 2^{s_{i+1}-i}]t$. Store them in vectors $\langle \mathbf{h} \rangle_t$ and $\langle \mathbf{v} \rangle_t$.
13: $\langle \mathbf{h} \rangle[b] = \langle \mathbf{h} \rangle[b](1-u_i) + \langle \mathbf{h} \rangle[b + 2^{s_{i+1}-i}]u_i$
14: $\langle \mathbf{v} \rangle[b] = \langle \mathbf{v} \rangle[b](1-u_i) + \langle \mathbf{v} \rangle[b + 2^{s_{i+1}-i}]u_i$
15: **for** $t = 0, 1, 2$ **do**
16: $a_{t,i} =$ MPC.InnerProduct($\langle \mathbf{h} \rangle_t, \langle \mathbf{v} \rangle_t$)
17: **return** $a_1, \ldots, a_{s_{i+1}}$.

Algorithm 4 Phase 2 in MPC

Input: Shared multilinear extension $\langle \tilde{V}_{i+1} \rangle$, shared evaluation $\langle \tilde{V}_{i+1}(u) \rangle$ from Phase 1, and randomness $g = g_1, \ldots, g_{s_{i+1}}, u = u_1, \ldots, u_{s_{i+1}}, v = v_1, \ldots, v_{s_{i+1}}$;
Output: Proofs of the sumcheck protocol $a'_1, \ldots, a'_{s_{i+1}}$;

1: $\mathbf{g} \leftarrow$ Precompute(g)
2: $\mathbf{u} \leftarrow$ Precompute(u)
3: **procedure** $\mathbf{m} \leftarrow$ Initialize_Phase2($\mathbf{g}, \mathbf{u}$))
4: $\forall y \in \{0,1\}^{s_{i+1}}$, set $\mathbf{m}[y] = 0$
5: **for** every multiplication gate (z, x, y) such that $\tilde{mult}_{i+1}(z, x, y) = 1$ **do**
6: $\mathbf{m}[y] = \mathbf{m}[y] + \mathbf{g}[z] \cdot \mathbf{u}[x]$
7: **return** $\mathbf{m}$;
8: Compute MPC.ScalarProduct($\langle \tilde{V}_{i+1}(u) \rangle, \langle V_{i+1} \rangle(y)$) for all $y \in \{0,1\}^{s_{i+1}}$. Denote the result as $\langle \mathbf{v}' \rangle$.
9: **procedure** $(a'_1, \ldots, a'_{s_{i+1}}) \leftarrow$ Sumcheckproof($\langle \mathbf{v}' \rangle, \mathbf{m}, v$)
10: Set $\langle a_{t,i} \rangle = 0$ for $i = 1, \ldots, s_{i+1}$ and $t = 0, 1, 2$
11: **for** each round $i = 1, \ldots, s_{i+1}$ **do**
12: **for** $b \in \{0,1\}^{s_{i+1}-i}$ **do** ▷ we use b as both a number and its binary representation
13: **for** $t = 0, 1, 2$ **do**
14: $\langle a'_{t,i} \rangle = \langle a_{t,i} \rangle + (\langle \mathbf{v}' \rangle[b](1-t) + \langle \mathbf{v}' \rangle[b + 2^{s_{i+1}-i}]t)(\mathbf{m}[b](1-t) + \mathbf{m}[b + 2^{s_{i+1}-i}]t)$
15: $\langle \mathbf{v}' \rangle[b] = \langle \mathbf{v}' \rangle[b](1-v_i) + \langle \mathbf{v}' \rangle[b + 2^{s_{i+1}-i}]v_i$
16: $\mathbf{m}[b] = \mathbf{m}[b](1-v_i) + \mathbf{m}[b + 2^{s_{i+1}-i}]v_i$
17: Reconstruct $\langle a'_1 \rangle, \ldots, \langle a'_{s_{i+1}} \rangle$
18: **return** $a'_1, \ldots, a'_{s_{i+1}}$.

$$
\begin{aligned}
&= \sum_{x,y \in \{0,1\}^{s_{i+1}}} ((\alpha_i \tilde{add}_{i+1}(u,x,y) + \beta_i \tilde{add}_{i+1}(v,x,y))) \\
&\quad (\langle \tilde{V}_{i+1} \rangle(x) + \langle \tilde{V}_{i+1} \rangle(y)) \\
&+ ((\alpha_i \tilde{mult}_{i+1}(u,x,y) + \beta_i \tilde{mult}_{i+1}(v,x,y)))(\langle \tilde{V}_{i+1} \rangle(x) \langle \tilde{V}_{i+1} \rangle(y))
\end{aligned}
$$

The polynomials $\tilde{add}$ and $\tilde{mult}$ are defined by the circuit and known by all provers. Hence, we can rewrite the Equation as:

$$
\begin{aligned}
\sum_{x,y \in \{0,1\}^{s_{i+1}}} &(\tilde{ADD}_{i+1}(v,x,y)(\langle \tilde{V}_{i+1} \rangle(x) + \langle \tilde{V}_{i+1} \rangle(y)) \\
&+ \tilde{MULT}_{i+1}(v,x,y) \langle \tilde{V}_{i+1} \rangle(x) \langle \tilde{V}_{i+1} \rangle(y))
\end{aligned} \tag{9}
$$

Note that Equation 9 is of the same form as Equation 3 on shared values of $\langle \tilde{V}_{i+1} \rangle$ and our new algorithms can be applied with the new public $\tilde{ADD}_{i+1}$ and $\tilde{MULT}_{i+1}$.

Complexity Analysis. As most of the steps are computed locally, the prover time of both Phase 1 and Phase 2 remains $O(C)$, linear in the size of the circuit. The additional MPC protocols in Step 15-16 of Algorithm 3 and Step 8 of Algorithm 4 do not introduce any asymptotic overhead on the running time. The communication complexity between the provers in Phase 1 is $O(C)$ per prover if these MPC protocols are implemented naively, and can be further reduced to $O(1)$ per prover using Shamir's packed secret sharing. Excluding the scalar multiplication, there is no communication in Phase 2 other than sending the proofs and receiving the randomness. The round complexity remains $O(d \log C)$ as in the original GKR protocol.

The protocol can be made non-interactive via the Fiat-Shamir heuristic [14]. Namely, in each round in Algorithm 3 (i.e., line 13 and 14), instead of receiving the randomness u_i from the verifier, it is computed non-interactively by hashing all the previous messages computed by the provers. It can also be further modified to achieve zero-knowledge by adding masking polynomials as proposed in [9, 46]. These modifications do not introduce any overhead asymptotically to the computation and communication of the provers, and we instantiate Step (2) in Protocol 2 with the non-interactive and zero-knoweledge GKR protocol on shared values.

3.5 GKR in MPC for Matrix Multiplications

In this section, we further propose a confidential and verifiable delegation scheme tailored for matrix multiplications. For simplicity, we assume that A, B and D are all $m \times m$ matrices such that $A \times B = D$. A and B are secret-shared by $\mathcal{V}$ to the provers, or intermediate shared values during a larger computation consisting of the matrix multiplication as one step. The provers first execute an MPC protocol to compute the shares of D. Using the special MPC

Algorithm 5 Confidential and Verifiable Matrix Multiplication

Input: $\langle A\rangle, \langle B\rangle$, randomness
$g = g_1, \ldots, g_{\log m}, u = u_1, \ldots, u_{\log m}, v = v_1, \ldots, v_{\log m}$;
Output: Proofs of the sumcheck protocol $a_1, \ldots, a_{\log m}$;

1: $\mathbf{g} \leftarrow$ Precompute(g)
2: $\mathbf{u} \leftarrow$ Precompute(u)
3: **procedure** $(\langle \mathbf{A}\rangle, \langle \mathbf{B}\rangle) \leftarrow$ Initialize_MM$(\langle A\rangle, \langle B\rangle)$
4: Initialize $\langle \mathbf{A}\rangle, \langle \mathbf{B}\rangle$ as all 0s
5: **for** $\ell \in [m]$ **do**
6: **for** $k \in [m]$ **do**
7: $\langle \mathbf{A}\rangle[\ell] = \langle \mathbf{A}\rangle[\ell] + \langle A\rangle_{k,\ell} \cdot \mathbf{g}[k]$
8: $\langle \mathbf{B}\rangle[\ell] = \langle \mathbf{B}\rangle[\ell] + \langle B\rangle_{\ell,k} \cdot \mathbf{u}[k]$
9: **procedure** $(a_1, \ldots, a_{\log m}) \leftarrow$ Sumcheckproof$(\langle \mathbf{A}\rangle, \langle \mathbf{B}\rangle, v)$
10: **for** each round $i = 1, \ldots, \log m$ **do**
11: **for** $b \in \{0,1\}^{\log m - i}$ **do** ▷ we use b as both a number and its binary representation
12: **for** $t = 0, 1, 2$ **do**
13: Compute $\langle \mathbf{A}\rangle[b](1-t) + \langle \mathbf{A}\rangle[b + 2^{s_{i+1}-i}]t$ and $\langle \mathbf{B}\rangle[b](1-t) + \langle \mathbf{B}\rangle[b + 2^{s_{i+1}-i}]t$. Store them in vectors $\langle \mathbf{A}\rangle_t$ and $\langle \mathbf{B}\rangle_t$.
14: $\langle \mathbf{A}\rangle[b] = \langle \mathbf{A}\rangle[b](1-v_i) + \langle \mathbf{A}\rangle[b + 2^{s_{i+1}-i}]v_i$
15: $\langle \mathbf{B}\rangle[b] = \langle \mathbf{B}\rangle[b](1-v_i) + \langle \mathbf{B}\rangle[b + 2^{s_{i+1}-i}]v_i$
16: **for** $t = 0, 1, 2$ **do**
17: $a_{t,i} =$ MPC.InnerProduct$(\langle \mathbf{A}\rangle_t, \langle \mathbf{B}\rangle_t)$
18: **return** $a_1, \ldots, a_{\log m}$.

protocol [26], the computation complexity on each server is $O(m^3)$, and the communication complexity is $O(m^2)$.

After computing D, as observed by Justin Thaler in [41], the matrix multiplication relationship can be modeled as a sumcheck equation:

$$\tilde{D}(x,y) = \sum_{z\in\{0,1\}^{\log m}} \tilde{A}(x,z)\tilde{B}(z,y), \tag{10}$$

where $\tilde{A}, \tilde{B}, \tilde{D}$ are multilinear extensions $\{0,1\}^{\log m} \times \{0,1\}^{\log m} \to \mathbb{F}$ defined by the matrices A, B, D. In particular, $\tilde{A}(x,y) = A_{\bar{x},\bar{y}}$, $\tilde{B}(x,y) = B_{\bar{x},\bar{y}}$ and $\tilde{D}(x,y) = D_{\bar{x},\bar{y}}$ for all $x, y \in \{0,1\}^{\log m}$ and $\bar{x}, \bar{y}$ denote their corresponding value in $[m]$. It is not hard to see that Equation 10 holds for all $x, y \in \{0,1\}^{\log m}$ by the definition of the polynomials, and thus the equation holds for the multilinear extensions.

To verify the correctness of the matrix multiplication, the verifier picks $g, u \in \mathbb{F}^{\log m}$ and evaluates $\tilde{D}(g,u)$. Again this value can also be sent by the prover during the GKR protocol in a larger computation. The verifier then runs a sumcheck protocol with the provers to check:

$$\tilde{D}(g,u) = \sum_{z\in\{0,1\}^{\log m}} \tilde{A}(g,z)\tilde{B}(z,u). \tag{11}$$

We propose an efficient MPC protocol to execute this sumcheck when the matrices A and B are secret-shared between n provers. Observe that Equation 11 is again a sumcheck protocol on the product of two polynomials, and in the first step, we would like to evaluate both polynomials on the Boolean hypercube, i.e., $\tilde{A}(g,z)$ and $\tilde{B}(z,u)$ for all $z \in \{0,1\}^{\log m}$. For each z, we have

$$\tilde{A}(g,z) = \sum_{t\in\{0,1\}^{\log m}} \tilde{I}(g,t)A_{t,z}, \tag{12}$$

where $\tilde{I}(g,t)$ is the identity polynomial defined in Section 2. The equation above holds for all $g \in \{0,1\}^{\log m}$ by the definition of $\tilde{A}$, and thus holds for all $g \in \mathbb{F}^{\log m}$ because both sides are multilinear extensions. This equation suggests that we can precompute $\tilde{I}(g,t)$ for all $t \in \{0,1\}^{\log m}$, and then sum up $\tilde{I}(g,t)A_{t,z}$ for each z, as shown in Steps 1-8 in Algorithm 5. Crucially, as g is known by all the provers, this initialization is a linear function and can be computed locally on each server. At the end of this step, each prover ends up with a shared vector $\langle \mathbf{A}\rangle$ of size m. The same algorithm applies to matrix B.

After the initialization, each prover holds the shares of two vectors storing the evaluations of $\tilde{A}(g,z)$ and $\tilde{B}(z,u)$ on the Boolean hypercube. We then apply the dynamic programming technique to compute the proofs of the sumcheck protocol. The algorithm is presented in Steps 9-18 in Algorithm 5 and is similar to the protocol in Algorithm 3. Only Steps 16 and 17 involve an MPC protocol to compute the inner product.

Complexity Analysis. The main advantage of our scheme is that the overhead to generate the proof is minimal compared to the time of computing the matrix multiplication in MPC. The computation complexity on each prover is $O(m^2)$, dominated by Procedure Initialize_MM. The communication complexity is only $O(m)$ to compute the inner products. These are smaller than the MPC protocol by a factor of $O(m)$. The proof size and the number of rounds are both $O(\log m)$.

3.6 Polynomial Commitments in MPC

In the last round of the GKR protocol, $\mathcal{V}$ receives a claim about the evaluation of a multilinear extension defined by the witness at a random point from the provers. In order to complete the protocol, the provers further prove that the evaluation is correct using a polynomial commitment scheme. We use multivariate polynomial commitment proposed by Papamanthou et al. [36], which is a generalization of the KZG polynomial commitment [25] to the multivariate case.

Similar to [35], we also observe that all computations in the multivariate polynomial commitment are linear and can be computed locally by each prover when the witness is secret-shared. The protocol of each prover is exactly the same as the original polynomial commitment as long as we set the field of the arithmetic secret sharing the same as the order of the base group of the bilinear map.

The protocol is presented in Protocol 3. The key generation remains exactly the same as the original scheme. To commit to a shared polynomial, it suffices to compute the shared commitent $\langle \mathrm{com}\rangle$ on each prover by raising the public parameters to shares of the multilinear extension evaluations. The final commitment is the product of all shared commitments when using the additive sharing. It can also be reconstructed via public Lagrange polynomials when using the Shamir sharing.

Protocol 3 (Confidential and Verifiable Multivariate Polynomial Commitment). *Let f be a multilinear extension polynomial with ℓ variables. Each prover holds the share of each evaluation of f on the Boolean hypercube, denoted as $\langle f \rangle$.*

(1) $\mathsf{KeyGen}(1^\lambda, \ell)$: *Run* $\mathsf{bp} \leftarrow \mathsf{BilGen}(1^\lambda)$. *Select* $s_1, \cdots, s_\ell \in \mathbb{F}$ *randomly and compute* $\mathsf{pp} = \mathsf{bp}, \{g^{\prod_{i=1}^{k}(b_i s_i + (1-b_i)(1-s_i))}\}$ *for all* $b \in \{0,1\}^k, k = 1, 2, \ldots, \ell$.

(2) $\mathsf{Commit}(\langle f \rangle, \mathsf{pp})$: *Compute* $\langle \mathsf{com} \rangle = g^{\langle f \rangle(s_1, \cdots, s_\ell)}$. *The commitment* com *is the product of the commitments* $\langle \mathsf{com} \rangle$ *from all provers.*

(3) $\mathsf{Open}(\langle f \rangle, t, \mathsf{pp})$: *On input* $t = (t_1, \cdots, t_\ell)$, *compute* $\langle y \rangle = \langle f \rangle(t)$. *Compute polynomials* $\langle q \rangle_i(x)$ *for* $i = 1, \cdots, \ell$ *such that*

$$\langle f \rangle(x_1, \cdots, x_\ell) - \langle f \rangle(t_1, \cdots, t_\ell) = \sum_{i=1}^{\ell} (x_i - t_i) \cdot \langle q \rangle_i(x)$$

Compute the shared proof as $\langle \pi \rangle := (g^{\langle q \rangle_1(s)}), \ldots, g^{\langle q \rangle_\ell(s)})$.
Reconstruct y and output the final proof π as the product of the shared proofs from all provers.

(4) $\mathsf{Verify}(\mathsf{com}, y, t, \pi, \mathsf{pp})$: *Parse the proof* π *as* $(\pi_1, \cdots, \pi_\ell)$. *If* $e(\mathsf{com}/g^y, g) = \prod_{i=1}^{\ell} e(g^{s_i - t_i), p i_i})$, *output 1, otherwise, output 0.*

$\mathcal{F}_f(\langle a \rangle_1, \langle a \rangle_2, \ldots, \langle a \rangle_n)$

The functionality interacts with n parties $P_1, \ldots P_n$.

1: Upon receiving shares $\langle a \rangle_1, \langle a \rangle_2, \ldots, \langle a \rangle_n$, reconstruct $a = \langle a \rangle_1 + \langle a \rangle_2 + \ldots + \langle a \rangle_n \mod p$, where a can be a vector of values.
2: Compute $y = f(a)$.
3: Sample random values $\langle y \rangle_1, \langle y \rangle_2, \ldots, \langle y \rangle_{n-1}$ from $\mathbb{F}_p$. Set $\langle y \rangle_n = y - (\langle y \rangle_1 + \langle y \rangle_2 + \ldots + \langle y \rangle_{n-1})$.
4: Send $\langle y \rangle_i$ to P_i.

Figure 2: Ideal functionality $\mathcal{F}$ of function f.

After receiving the evaluation point t from $\mathcal{V}$, each prover computes the partial quotient polynomials $\langle q \rangle_i(x)$ as shown in the protocol using the long division by $x_i - t_i$ for $i = 1, \ldots, \ell$. As the divisor is the same, it is easy to see that $f(x_1, \cdots, x_\ell) - f(t_1, \cdots, t_\ell) = \sum_{i=1}^{\ell} (x_i - t_i) \cdot q_i(x)$ when reconstructed. Then the partial proofs can be computed by raising pp to the corresponding multilinear evaluations of $\langle q_i \rangle(x)$. Finally, the Verify algorithm remains exactly the same as the original polynomial commitment in the plain setting.

Complexity Analysis. Protocol 3 does not involve any MPC and all the computations are done locally other than reconstructing the commitment and the proof. The prover time of each prover is $O(2^\ell)$. The proof size and the verifier time are both $O(\ell)$.

3.7 Putting Everything Together

Theorem 1. *Protocol 2 is a confidential and verifiable delegation scheme under Definition 3.*

Proof. We first define the security of the MPC protocols using the framework of Universal Composition(UC) [8]. The ideal functionality $\mathcal{F}$ is defined in Figure 2. It receives the shares of the input from all parties, reconstructs the values, evaluates the function and shares the output to the parties. We consider a malicious adversary $\mathcal{A}$ that can corrupt up to t provers.

Security is defined by comparing a real interaction and an ideal interaction. Let $\mathbf{real}[\mathcal{Z}, \mathcal{A}, \Pi]$ denote the output bit of the environment $\mathcal{Z}$ interacting with adversary $\mathcal{A}$ and honest parties executing protocol Π in the real world. Let $\mathbf{ideal}[\mathcal{Z}, \mathcal{S}, \mathcal{F}]$ denote the output bit of the environment $\mathcal{Z}$ interacting with a simulator $\mathcal{S}$ and the ideal functionality $\mathcal{F}$ where all parties forward their inputs to $\mathcal{F}$ and forwards the output to the environment. We say that a protocol Π securely realizes a functionality $\mathcal{F}$ if for every adversary $\mathcal{A}$ attacking the real interaction, there exists a simulator $\mathcal{S}$ attacking the ideal interaction, such that for all environments $\mathcal{Z}$,

$$|\Pr[\mathbf{real}[\mathcal{Z}, \mathcal{A}, \Pi] = 1] - \Pr[\mathbf{ideal}[\mathcal{Z}, \mathcal{S}, \mathcal{F}] = 1]| \leq \mathbf{negl}(\lambda).$$

As we are using maliciously secure MPC protocols as blackboxes, we have the following lemma:

Lemma 2. *Protocols* MPC.Add, MPC.Mult, MPC.InnerProduct, MPC.ScalarProduct *securely realizes the ideal functionalities $\mathcal{F}$ with f being addition, multiplication, inner product and scalar product.*

Completeness. We show the completeness by comparing Algorithm 1 with Algorithm 3. By the ideal functionality of MPC.InnerProduct and MPC.Add (and subtraction as well), the output of Algorithm 3 after reconstruction is the same as the output of Algorithm 1. Similar arguments can be made for Algorithm 4 and the polynomial commitment in Protocol 3. Therefore, $\Pr[\mathsf{Verify}(y, x, \pi, \mathsf{pp}) \rightarrow 1] = 1$.

Knowledge Soundness. The knowledge soundness follow from the fact that the view of $\mathcal{V}$ is identical to the original protocol between one verifier and one prover. All the algorithms and protocols proposed in this section are computing the same messages in the proof when the witness is secret-shared. Therefore, we view $\vec{P} = \{P_1, \ldots, P_n\}$ as a single prover $\mathcal{P}^*$. As proven in [43], for any malicious $\mathcal{P}^*$, there exist an extractor ε to extract the witness from the proof with an overwhelming probability in the algebraic group model (AGM) [17] based on the q-strong bilinear Diffie-Hellman assumption. Therefore, Protocol 2 is knowledge sound using the same extractor ε.

***t*-zero-knowledge.**

Lemma 3. Prove() *in Protocol 2 securely realizes* $\mathbf{ZKP}.\mathcal{P}$*, the prover protocol of the ZKP scheme based on GKR and polynomial commitments 2.1.*

By Lemma 2, MPC.Add, MPC.Mult, MPC.InnerProduct, MPC.ScalarProduct securely realizes the ideal functionalities $\mathcal{F}$ with the corresponding computation f. By the composition theorem of the UC framework [8], Algorithm 3 securely realizes the ideal functionality with f defined by Algorithm 1. Similarly, Algorithm 4 securely realizes Algorithm 2, and Protocol 3 securely realizes the KZG polynomial commitment. Therefore, Prove() securely realizes $\mathbf{ZKP}.\mathcal{P}$.

Lemma 4 ([35]). *If* $(\text{Setup}, \mathcal{P}, \text{Verify})$ *in the plain setting is a zk-SNARK, and* Prove *on shared values securely realizes* $\mathcal{P}$*, then the scheme is t-zero-knowledge.*

Finally, by Lemma 4 shown in [35, Theorem 1], the scheme is t-zero-knowledge, which completes the proof.

4 EXPERIMENTS

We have fully implemented our confidential and verifiable delegation scheme for both general circuits and matrix multiplications, and we present the experimental results in this section.

Implementation and Setting. The system is implemented in C++. The sumcheck protocol and the GKR protocol are based on the open-source code of Libra [2]. For MPC computations, we adopt the MP-SPDZ library [4] to implement an additive secret-sharing protocol with malicious security and all but one honest party. Our implementation involves targeted modifications to the original Libra code, specifically focusing on locations requiring MPC computations. We integrate MP-SPDZ's C++ interface for share computations and merge proofs before transmission to the verifier. To enhance circuit evaluation efficiency, we optimize by "grouping" multiplication gates in a layer, enabling the use of MP-SPDZ's inner product interface.

We ran all experiments on an AWS c5.9xlarge instance with an Intel Xeon Platinum 8000 processor and 72GB of RAM. We simulated multiple provers in a LAN setting with 10 Gbps network bandwidth using the tc command. We report the average running time over 10 executions.

4.1 Performance of Our System

We first benchmark our confidential and verifiable delegation system in this section.

General circuits. We executed our system with 2 provers on circuits ranging from 2^{17} to 2^{20} gates. We break down prover's runtime and communication (column Comm.) and show each component in Table 2. For a circuit size of 2^{18}, our confidential and verifiable delegation protocol's end-to-end prover time is 163 seconds. Approximately 11% of this time is spent on evaluating the circuit in MPC, while the majority (89%) is dedicated to the MPC protocol for ZKP generation. This is because although most of the steps in Algorithm 3 and 4 are performed locally, the size of the MPC computation is still linear in the size of the circuit. Note that the time reported in the table includes both the offline phase and the online phase of the MPC protocol. The time of the local computation to generate the proofs is very small compared to the other two parts.

# of gates	Prover	Circ Eval	ZKP (MPC)	ZKP (local)	Comm.
2^{17}	78.5	8.61	69.8	0.02	30.34
2^{18}	162.71	17.25	145.43	0.035	60.41
2^{19}	314.5	34.27	281.73	0.074	120.53
2^{20}	631.21	68.48	562.59	0.15	240.76

Table 2: Breakdown of prover time in seconds and communication in megabytes for general circuits.

Matrix Size	Prover	Matrix Mult	ZKP (MPC)	ZKP (Local)	Comm.
64×64	0.139	0.101	0.0359	0.00017	1.605
128×128	0.254	0.18	0.074	0.0006	4.617
256×256	0.601	0.591	0.15	0.003	18.398

Table 3: Breakdown of prover time in seconds and communication in megabytes for matrix multiplications.

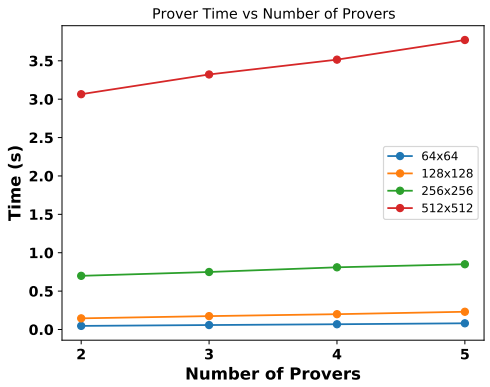

Figure 3: Prover time with various number of provers

The proof size is from 13.8KB for $C = 2^{17}$ to 14.9KB for $C = 2^{20}$, which grows logarithmically with the size of the circuit. Similarly, the verifier ranges from 13ms to 15ms.

Matrix Multiplications. In Table 3, we demonstrate the performance of our specialized matrix multiplication protocol by varying the matrix size from from 64×64 to 256×256. As shown in the table, the breakdown is completely different from the case of general circuits. In particular, for $m = 256$, 98.3% of the prover time is spent on computing the matrix multiplication in MPC. The additional overhead to generate the ZKP is minimal, thanks to our special protocol that improves both the computation and the communication by $O(m)$. Moreover, the special protocol is generally faster than the protocol for general circuits in all aspects. For example, implementing a 64×64 matrix multiplication would require a circuit of size more than 2^{18}, and the prover time of the special protocol

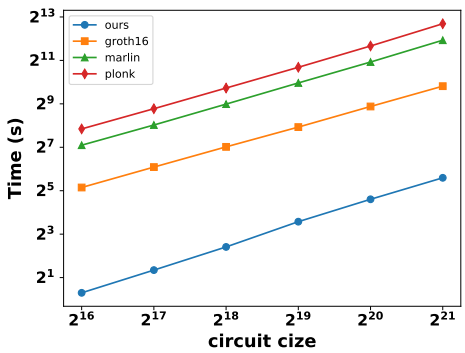

Figure 4: Comparison to schemes in [35]. The time is for the online phase of 2 provers.

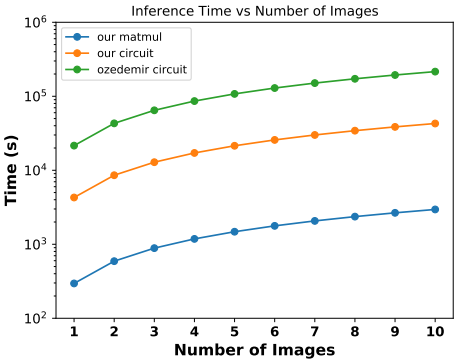

Figure 6: Lenet inference. The time is total time ranging from 1 to 10 images.

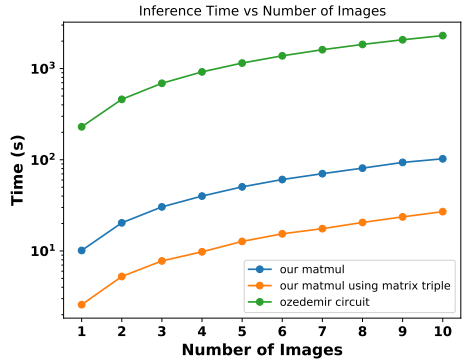

Figure 5: Neural network inference. The time is total time ranging from 1 to 10 images.

is 1170× faster than the protocol in Table 2 for $C = 2^{18}$, and the communication is 37× smaller. Finally, the proof size ranges from 0.15KB to 0.2KB, and the verifier time is from 0.02ms to 0.036ms, both are succinct.

Scaling to multiple servers. We demonstrate the performance of the matrix multiplication scheme with varying numbers of provers (2 to 5) in Figure 3. The running time scales linearly with the number of provers, remaining practical. For a 512×512 matrix multiplication on 5 provers, even with all but one potentially malicious, it takes only 3.77 seconds.

4.2 Comparison to Prior Work

In this section, we further compare the performance of our system to the prior work of [35], which is implemented in the Github repository at [5]. As [35] only reports the online time of MPC assuming the multiplication triplets have been generated, we also only show the online time of our scheme to get a fair comparison. Figure 4 shows the comparison of the prover time to the three schemes in [35]. As shown in the figure, our scheme improves the prover time significantly. For a circuit with 2^{20} gates[2], our online prover time is 10× faster than the scheme based on Groth16, 80× faster than the one based on Marlin and 133× faster than the one based on Plonk. The comparison demonstrates the lightweight use of MPC in our confidential and verifiable delegation scheme to generate the ZKP based on interactive proofs.

However, the proof size of our scheme is admittedly larger than these schemes because of the underlying cryptographic techniques. The proof size of these schemes is only several hundreds of bytes, while it is tens of KBs in our scheme and it further grows with the depth of the circuit.g

4.3 Machine Learning Inference

We implemented confidential and verifiable machine learning inferences by secret-sharing a pre-trained neural network model to 3 provers via additive sharing. The experiment utilized the MNIST dataset [29] comprising hand-written digit images with dimensions 28×28. The images, flattened to a vector of size 784, are secret-shared to the servers. The neural network model, trained using PyTorch, consists of three fully connected layers (128, 84, and 10 neurons) with ReLU activation functions.

In Figure 5, a single inference and its proof generation on three provers take only 2.6 seconds. Comparing our protocol for general circuits with the new matrix multiplication protocol, our scheme is 12.7× faster. Additionally, it is 88× faster than the prior work [35] (Groth16). Our implementation supports multiple image inferences, taking 26.9 seconds for 10 images.

We also tested our scheme on the convolutional neural network of Lenet [28]. It consists of 2 convolution layers with 5×5 kernels and 2 padding, 2 average pooling layers, and 3 fully connected layers of size 120,84, and 10 respectively. We implement convolutions as matrix multiplications and apply our special protocol, thus the dimension is much larger than the fully connected neural network above. There are special MPC protocols for convolutions with convolution triples [38], but we were not able to obtain efficiency

[2]Groth16 and Marlin use the rank-1-constraint-systems (R1CS) instead of circuits, and we execute them on R1CS with the same number of constraints.

improvement in our setting, and designing special protocols for convolutions is left as a future work.

As shown in Figure 6, the running time for 1 inference is 288 seconds on three provers. Likewise, we include the performance of implementing the lenet inference as a circuit. A single convolution on a 28×28 image with kernel size 5 will result in a 1024×1024 square matrix multiplication, which translate into 2^{20} gates. The running time of our scheme with the new protocol for matrix multiplication is $14.8\times$ faster than the protocol for general circuit, and $74.8\times$ faster than the prior work of [35]. Our implementation takes 2888 seconds to inference 10 images.

5 CONCLUSIONS

This paper introduces a confidential and verifiable delegation scheme by integrating MPC and ZKP. We demonstrate efficient proof generation for interactive-proof-based protocols within MPC, emphasizing a specialized protocol for matrix multiplications with asymptotically smaller proof generation time than MPC evaluation. Since we observe that MPC is typically the dominant cost, future work could explore enhancing ZKP generation in MPC by utilizing multiple provers for improved running time.

6 ACKNOWLEDGMENTS

This work is funded by the AFRL/RI, Rome, NY, USA under Contract Number FA8750-22-2-0267. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air. Approved for Public Release on 16 Mar 2024. Distribution is Unlimited. Case Number: AFRL-2024-1294.

REFERENCES

[1] EZKL . https://cli.gizatech.xyz/frameworks/ezkl.
[2] Libra. https://github.com/sunblaze-ucb/Libra.
[3] Modulus . https://medium.com/@ModulusLabs.
[4] Spdz. https://github.com/data61/MP-SPDZ.
[5] collaborative-zksnark . https://github.com/alex-ozdemir/collaborative-zksnark, 2021.
[6] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, 1991.
[7] Alexandre Bois, Ignacio Cascudo, Dario Fiore, and Dongwoo Kim. Flexible and efficient verifiable computation on encrypted data. In *IACR International Conference on Public-Key Cryptography*, pages 528–558. Springer, 2021.
[8] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.
[9] Alessandro Chiesa, Michael A. Forbes, and Nicholas Spooner. A Zero Knowledge Sumcheck and its Applications. *CoRR*, abs/1704.02086, 2017.
[10] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zksnarks with universal and updatable srs. In *EUROCRYPT 2020*, pages 738–768, 2020.
[11] Alessandro Chiesa, Ryan Lehmkuhl, Pratyush Mishra, and Yinuo Zhang. EOS: Efficient private delegation of zkSNARK provers. In *32st USENIX Security Symposium (USENIX Security 23)*, 2023.
[12] Pankaj Dayama, Arpita Patra, Protik Paul, Nitin Singh, and Dhinakaran Vinayagamurthy. How to prove any np statement jointly? efficient distributed-prover zero-knowledge protocols. In *Proceedings on Privacy Enhancing Technologies*, 2022.
[13] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. Zen: An optimizing compiler for verifiable, zero-knowledge neural network inferences. *Cryptology ePrint Archive*, 2021.
[14] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Crypto 1986*.
[15] Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently verifiable computation on encrypted data. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 844–855, 2014.
[16] Dario Fiore, Anca Nitulescu, and David Pointcheval. Boosting verifiable computation on encrypted data. In *Public-Key Cryptography–PKC 2020: 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4–7, 2020, Proceedings, Part II 23*, pages 124–154. Springer, 2020.
[17] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part II 38*, pages 33–62. Springer, 2018.
[18] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, 2019.
[19] Sanjam Garg, Aarushi Goel, Abhishek Jain, Guru-Vamsi Policharla, and Sruthi Sekar. zkSaaS: Zero-Knowledge SNARKs as a service. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4427–4444, Anaheim, CA, 2023.
[20] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings 30*, pages 465–482. Springer, 2010.
[21] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating Computation: Interactive Proofs for Muggles. *J. ACM*, 62(4):27:1–27:64, September 2015.
[22] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT 2016*, pages 305–326, 2016.
[23] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. {GAZELLE}: A low latency framework for secure neural network inference. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1651–1669, 2018.
[24] Sanket Kanjalkar, Ye Zhang, Shreyas Gandlur, and Andrew Miller. Publicly auditable mpc-as-a-service with succinct verification and universal setup. In *2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 386–411. IEEE, 2021.
[25] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT 2010*, pages 177–194.
[26] Marcel Keller. Mp-spdz: A versatile framework for multi-party computation. 2020.
[27] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems*, 34:4961–4973, 2021.
[28] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
[29] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.
[30] Tianyi Liu, Xiang Xie, and Yupeng Zhang. Zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2968–2985, 2021.
[31] Xuanming Liu, Zhelei Zhou, Yinghao Wang, Bingsheng Zhang, and Xiaohu Yang. Scalable collaborative zk-snark: Fully distributed proof generation and malicious security. Cryptology ePrint Archive, Paper 2024/143, 2024. https://eprint.iacr.org/2024/143.
[32] Silvio Micali, Oded Goldreich, and Avi Wigderson. How to play any mental game. In *Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC*, pages 218–229. ACM New York, NY, USA, 1987.
[33] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: a cryptographic inference system for neural networks. In *Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice*, pages 27–30, 2020.
[34] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *IEEE S and P*, 2017.
[35] Alex Ozdemir and Dan Boneh. Experimenting with collaborative {zk-SNARKs}:{Zero-Knowledge} proofs for distributed secrets. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 4291–4308, 2022.
[36] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In *TCC 2013*, pages 222–242, 2013.
[37] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *S&P 2013*, pages 238–252, 2013.
[38] Marc Rivinius, Pascal Reisert, Sebastian Hasler, and Ralf Küsters. Convolutions in overdrive: Maliciously secure convolutions for mpc. *Cryptology ePrint Archive*, 2023.
[39] Berry Schoenmakers, Meilof Veeningen, and Niels de Vreede. Trinocchio: Privacy-preserving outsourcing by distributed verifiable computation. In *Applied Cryptography and Network Security: 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings 14*, pages 346–366. Springer, 2016.

[40] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
[41] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. 2013.
[42] Riad S Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 926–943. IEEE, 2018.
[43] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *Advances in Cryptology (CRYPTO)*, 2019.
[44] Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.
[45] Jiaheng Zhang, Zhiyong Fang, Yupeng Zhang, and Dawn Song. Zero knowledge proofs for decision tree predictions and accuracy. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 2039–2053, 2020.
[46] Jiaheng Zhang, Tiancheng Xie, Y. Zhang, and D. Song. Transparent polynomial delegation and its applications to zero knowledge proof. *2020 IEEE Symposium on Security and Privacy (SP)*, pages 859–876, 2020.
[47] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *IEEE Symposium on Security and Privacy (S&P) 2017*, 2017.