Virgo: Zero Knowledge Proofs System without Trusted Setup

Jiaheng Zhang   Tiancheng Xie

***Abstract*—We present a new succinct zero knowledge argument scheme for layered arithmetic circuits without trusted setup. The prover time is $O(C + n \log n)$ and the proof size is $O(D \log C + \log^2 n)$ for a D-depth circuit with n inputs and C gates. The verification time is also succinct, $O(D \log C + \log^2 n)$, if the circuit is structured. Our scheme only uses lightweight cryptographic primitives such as collision-resistant hash functions and is plausibly post-quantum secure. We implement a zero knowledge argument system, Virgo, based on our new scheme and compare its performance to existing schemes. Experiments show that it only takes 53 seconds to generate a proof for a circuit computing a Merkle tree with 256 leaves, at least an order of magnitude faster than all other succinct zero knowledge argument schemes. The verification time is 50ms, and the proof size is 253KB, both competitive to existing systems.**

Underlying Virgo is a new transparent zero knowledge verifiable polynomial delegation scheme with logarithmic proof size and verification time. The scheme is in the interactive oracle proof model and may be of independent interest.

I. Introduction

Zero knowledge proof (ZKP) allows a powerful prover to convince a weak verifier that a statement is true, without leaking any extra information about the statement beyond its validity. In recent years, significant progress has been made to bring ZKP protocols from purely theoretical interest to practical implementations, leading to its numerous applications in delegation of computations, anonymous credentials, privacy-preserving cryptocurrencies and smart contracts.

Despite of these great success, there are still some limitations of existing ZKP systems. In SNARK [31], the most commonly adopted ZKP protocol in practice, though the proof sizes are of just hundreds of bytes and the verification times are of several milliseconds regardless of the size of the statements, it requires a trusted setup phase to generate structured reference string (SRS) and the security will be broken if the trapdoor is leaked.

To address this problem, many ZKP protocols based on different techniques have been proposed recently to remove the trusted setup, which are referred as *transparent* ZKP protocols. Among these techniques, ZKP schemes based on the doubly efficient interactive proof proposed by Goldwasser et al. in [25] (referred as *GKR* protocol in this paper) are particularly interesting due to their efficient prover time and sublinear verification time for statements represented as structured arithmetic circuits, making it promising to scale to large statements. Unfortunately, as of today we are yet to construct an efficient transparent ZKP system based on the GKR protocol with succinct[1] proof size and verification time. The transparent scheme in [35] has square-root proof size and verification time, while the succinct scheme in [36] requires a one-time trusted setup.

Our contributions. In this paper, we advance this line of research by proposing a transparent ZKP protocol based on GKR with succinct proof size and verification time, when the arithmetic circuit representing the statement is structured. The prover time of our scheme is particularly efficient, at least an order of magnitude faster than existing ZKP systems, and the verification time is merely tens of milliseconds. Our concrete contributions are:

- **Transparent zero knowledge verifiable polynomial delegation.** We propose a new zero knowledge verifiable polynomial delegation (zkVPD) scheme without trusted setup. Compared to existing pairing-based zkVPD schemes [30], [38], [39], our new scheme does not require a trapdoor and linear-size public keys, and eliminates heavy cryptographic operations such as modular exponentiation and bilinear pairing. Our scheme may be of independent interest, as polynomial delegation/commitment has various applications in areas such as verifiable secret sharing [5], proof of retrievability [37] and other constructions of ZKP [28].
- **Transparent zero knowledge argument.** Following the framework proposed in [39], we combine our new zkVPD protocol with the GKR protocol efficiently to get a transparent ZKP scheme. Our scheme only uses light-weight cryptographic primitives such as collision-resistant hash functions and is plausibly post-quantum secure.
- **Implementation and evaluation.** We implement a ZKP system, Virgo, based on our new scheme. We develop optimizations such that our system can take arithmetic circuits on the field generated by Mersenne primes, the operations on which can be implemented efficiently using integer additions, multiplications and bit operations in C++. We plan to open source our system.

A. Our Techniques

Our main technical contribution in this paper is a new transparent zkVPD scheme with $O(N \log N)$ prover time, $O(\log^2 N)$ proof size and verification time, where N is the size of the polynomial. We summarize the key ideas behind our construction. We first model the polynomial evaluation as the inner product between two vectors of size N: one defined by the coefficients of the polynomial and the other defined by the evaluation point computed on each monomial of the polynomial. The former is committed by the prover (or delegated to the prover after preprocessing in the case of

[1] "succinct" denotes poly-logarithmic in the size of the statement C.

delegation of computation), and the later is publicly known to both the verifier and the prover. We then develop a protocol that allows the prover to convince the verifier the correctness of the inner product between a committed vector and a public vector with proof size $O(\log^2 N)$, based on the univariate sumcheck protocol recently proposed by Ben-Sasson et al. in [10] (See Section II-D). To ensure security, the verifier needs to access the two vectors at some locations randomly chosen by the verifier during the protocol. For the first vector, the prover opens it at these locations using standard commitment schemes such as Merkle hash tree. For the second vector, however, it takes $O(N)$ time for the verifier to compute its values at these locations locally. In order to improve the verification time, we observe that the second vector is defined by the evaluation point of size only ℓ for a ℓ-variate polynomial, which is $O(\log N)$ if the polynomial is dense. Therefore, this computation can be viewed as a function that takes ℓ inputs, expands them to a vector of N monomials and outputs some locations of the vector. It is a perfect case for the verifier to use the GKR protocol to delegate the computation to the prover and validate the output, instead of computing locally. With proper design of the GKR protocol, the verification time is reduced to $O(\log^2 N)$ and the total prover time is $O(N \log N)$. We then turn the basic protocol into zero knowledge using similar techniques proposed in [4], [10]. The detailed protocols are presented in Section III.

II. Preliminaries

We use λ to denote the security parameter, and $\mathsf{negl}(\lambda)$ to denote the negligible function in λ. "PPT" stands for probabilistic polynomial time. For a multivariate polynomial f, its "variable-degree" is the maximum degree of f in any of its variables. We often rely on polynomial arithmetic, which can be efficiently performed via fast Fourier tranforms and their inverses. In particular, polynomial evaluation and interpolation over a multiplicative coset of size n of a finite field can be performed in $O(n \log n)$ field operations via the standard FFT protocol, which is based on the divide-and-conquer algorthim.

A. Interactive Proofs and Zero-knowledge Arguments

Interactive proofs. An interactive proof allows a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ the validity of some statement through several rounds of interaction. We say that an interactive proof is public coin if $\mathcal{V}$'s challenge in each round is independent of $\mathcal{P}$'s messages in previous rounds. The proof system is interesting when the running time of $\mathcal{V}$ is less than the time of directly computing the function f. We formalize interactive proofs in the following:

Definition 1. *Let f be a Boolean function. A pair of interactive machines $\langle \mathcal{P}, \mathcal{V} \rangle$ is an interactive proof for f with soundness ϵ if the following holds:*

- **Completeness.** *For every x such that $f(x) = 1$ it holds that* $\Pr[\langle \mathcal{P}, \mathcal{V} \rangle(x) = \mathsf{1}] = 1$.
- **ϵ-Soundness.** *For any x with $f(x) \neq 1$ and any $\mathcal{P}^*$ it holds that* $\Pr[\langle \mathcal{P}^*, \mathcal{V} \rangle = \mathsf{1}] \leq \epsilon$

Zero-knowledge arguments. An argument system for an NP relationship $\mathcal{R}$ is a protocol between a computationally-bounded prover $\mathcal{P}$ and a verifier $\mathcal{V}$. At the end of the protocol, $\mathcal{V}$ is convinced by $\mathcal{P}$ that there exists a witness w such that $(x; w) \in R$ for some input x. We focus on arguments of knowledge which have the stronger property that if the prover convinces the verifier of the statement validity, then the prover must know w. We use $\mathcal{G}$ to represent the generation phase of the public parameters pp. Formally, consider the definition below, where we assume R is known to $\mathcal{P}$ and $\mathcal{V}$.

Definition 2. *Let $\mathcal{R}$ be an NP relation. A tuple of algorithm $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a zero-knowledge argument of knowledge for $\mathcal{R}$ if the following holds.*

- ***Correctness***. *For every pp output by $\mathcal{G}(1^\lambda)$ and $(x, w) \in R$,*
$$\langle \mathcal{P}(\mathsf{pp}, w), \mathcal{V}(\mathsf{pp}) \rangle (x) = \mathsf{1}$$
- ***Soundness***. *For any PPT prover $\mathcal{P}$, there exists a PPT extractor ε such that for every pp output by $\mathcal{G}(1^\lambda)$ and any x, the following probability is $\mathsf{negl}(\lambda)$:*
$$\Pr[\langle \mathcal{P}(\mathsf{pp}), \mathcal{V}(\mathsf{pp}) \rangle (x) = \mathsf{1} \wedge (x, w) \notin \mathcal{R} | w \leftarrow \varepsilon(\mathsf{pp}, x)]$$
- ***Zero knowledge***. *There exists a PPT simulator $\mathcal{S}$ such that for any PPT algorithm $\mathcal{V}^*$, auxiliary input $z \in \{0,1\}^*$, $(x; w) \in \mathcal{R}$, pp output by $\mathcal{G}(1^\lambda)$, it holds that*
$$\mathsf{View}(\langle \mathcal{P}(\mathsf{pp}, w), \mathcal{V}^*(z, \mathsf{pp}) \rangle (x)) \approx \mathcal{S}^{\mathcal{V}^*}(x, z)$$

*We say that $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a **succinct** argument system if the running time of $\mathcal{V}$ and the total communication between $\mathcal{P}$ and $\mathcal{V}$ (proof size) are* $\mathsf{poly}(\lambda, |x|, \log |w|)$.

In the definition of zero knowledge, $\mathcal{S}^{\mathcal{V}^*}$ denotes that the simulator $\mathcal{S}$ is given the randomness of $\mathcal{V}^*$ sampled from polynomial-size space. This definition is commonly used in existing transparent zero knowledge proof schemes [4], [10], [18], [35].

B. Zero-Knowledge Verifiable Polynomial Delegation

Let $\mathbb{F}$ be a finite field, $\mathcal{F}$ be a family of ℓ-variate polynomial over $\mathbb{F}$, and d be a variable-degree parameter. We use $\mathcal{W}_{\ell,d}$ to denote the collection of all monomials in $\mathcal{F}$ and $N = |\mathcal{W}_{\ell,d}| = (d+1)^\ell$. A zero-knowledge verifiable polynomial delegation scheme (zkVPD) for $f \in \mathcal{F}$ and $t \in \mathbb{F}^\ell$ consists of the following algorithms:

- $\mathsf{pp} \leftarrow \mathsf{zkVPD.KeyGen}(1^\lambda)$,
- $\mathsf{com} \leftarrow \mathsf{zkVPD.Commit}(f, r_f, \mathsf{pp})$,
- $((y, \pi); \{0, 1\}) \leftarrow$ $\langle \mathsf{zkVPD.Open}(f, r_f), \mathsf{zkVPD.Verify}(\mathsf{com}) \rangle (t, \mathsf{pp})$

Note that unlike the zkVPD in [30], [38], [39], our definition is transparent and does not have a trapdoor in $\mathsf{zkVPD.KeyGen}$. π denotes the transcript seen by the verifier during the interaction with $\mathsf{zkVPD.Open}$, which is similar to the proof in non-interactive schemes in [30], [38], [39].

Definition 3. *A zkVPD scheme satisfies the following properties:*

- ***Completeness.*** *For any polynomial $f \in \mathcal{F}$ and value $t \in \mathbb{F}^\ell$,* $\mathsf{pp} \leftarrow \mathsf{zkVPD.KeyGen}(1^\lambda)$, $\mathsf{com} \leftarrow \mathsf{zkVPD.Commit}(f, r_f \mathsf{pp})$*, it holds that*

$$\Pr\left[\langle \mathsf{zkVPD.Open}(f, r_f), \mathsf{zkVPD.Verify}(\mathsf{com})\rangle(t, \mathsf{pp}) = 1\right] = 1$$

- ***Soundness.*** *For any PPT adversary $\mathcal{A}$,* $\mathsf{pp} \leftarrow \mathsf{zkVPD.KeyGen}(1^\lambda)$*, the following probability is negligible of λ:*

$$\Pr\begin{bmatrix} (f^*, \mathsf{com}^*, t) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp}) \\ ((y^*, \pi^*); 1) \leftarrow \langle \mathcal{A}(), \mathsf{zkVPD.Verify}(\mathsf{com}^*)\rangle(t, \mathsf{pp}) \\ \mathsf{com}^* = \mathsf{zkVPD.Commit}(f^*, \mathsf{pp}) \\ f^*(t) \neq y^* \end{bmatrix}$$

- ***Zero Knowledge.*** *For security parameter λ, polynomial $f \in \mathcal{F}$,* $\mathsf{pp} \leftarrow \mathsf{zkVPD.KeyGen}(1^\lambda)$*, PPT algorithm $\mathcal{A}$, and simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$, consider the following two experiments:*

$\mathsf{Real}_{\mathcal{A},f}(\mathsf{pp})$:
1) $\mathsf{com} \leftarrow \mathsf{zkVPD.Commit}(f, r_f, \mathsf{pp})$
2) $t \leftarrow \mathcal{A}(\mathsf{com}, \mathsf{pp})$
3) $(y, \pi) \leftarrow \langle \mathsf{zkVPD.Open}(f, r_f), \mathcal{A}\rangle(t, \mathsf{pp})$
4) $b \leftarrow \mathcal{A}(\mathsf{com}, y, \pi, \mathsf{pp})$
5) Output b
$\mathsf{Ideal}_{\mathcal{A},\mathcal{S}^{\mathcal{A}}}(\mathsf{pp})$:
1) $\mathsf{com} \leftarrow \mathcal{S}_1(1^\lambda, \mathsf{pp})$
2) $t \leftarrow \mathcal{A}(\mathsf{com}, \mathsf{pp})$
3) $(y, \pi) \leftarrow \langle \mathcal{S}_2, \mathcal{A}\rangle(t_i, \mathsf{pp})$*, given oracle access to* $y = f(t)$.
4) $b \leftarrow \mathcal{A}(\mathsf{com}, y, \pi, \mathsf{pp})$
5) Output b

For any PPT algorithm $\mathcal{A}$ and all polynomial $f \in \mathbb{F}$, there exists simulator $\mathcal{S}$ such that

$$|\Pr[\mathsf{Real}_{\mathcal{A},f}(\mathsf{pp}) = 1] - \Pr[\mathsf{Ideal}_{\mathcal{A},\mathcal{S}^{\mathcal{A}}}(\mathsf{pp}) = 1]| \leq \mathsf{negl}(\lambda).$$

C. Zero Knowledge Argument Based on GKR

In [36], Xie et al. proposed an efficient zero knowledge argument scheme named Libra. The scheme extends the interactive proof protocol for layered arithmetic circuits proposed by Goldwasser et al. [25] (referred as the *GKR* protocol) to a zero knowledge argument using multiple instances of zkVPD schemes. Our scheme follows this framework and we review the detailed protocols here.

Sumcheck protocol. The sumcheck protocol is a fundamental protocol in the literature of interactive proof that has various applications. The problem is to sum a polynomial $f : \mathbb{F}^\ell \rightarrow \mathbb{F}$ on the binary hypercube $\sum_{b_1, b_2, \ldots, b_\ell \in \{0,1\}} f(b_1, b_2, ..., b_\ell)$. Directly computing the sum requires exponential time in ℓ, as there are 2^ℓ combinations of $b_1, \ldots, b_\ell$. Lund et al. [27] proposed a *sumcheck* protocol that allows a verifier $\mathcal{V}$ to delegate the computation to a computationally unbounded prover $\mathcal{P}$, who can convince $\mathcal{V}$ the correctness of the sum. At the end of the sumcheck protocol, $\mathcal{V}$ needs an oracle access to the evaluation of f at a random point $r \in \mathbb{F}^\ell$ chosen by $\mathcal{V}$. The proof size of the sumcheck protocol is $O(d\ell)$, where d is the variable-degree of f, and the verification time of the protocol is $O(d\ell)$. The sumcheck protocol is complete and sound with $\epsilon = \frac{d\ell}{|\mathbb{F}|}$.

GKR protocol. Let C be a layered arithmetic circuit with depth D over a finite field $\mathbb{F}$. Each gate in the i-th layer takes inputs from two gates in the $(i+1)$-th layer; layer 0 is the output layer and layer D is the input layer. The GKR protocol proceeds layer by layer. Upon receiving the claimed output from $\mathcal{P}$, in the first round, $\mathcal{V}$ and $\mathcal{P}$ run a sumcheck protocol to reduce the claim about the output to a claim about the values in the layer above. In the i-th round, both parties reduce a claim about layer $i-1$ to a claim about layer i through sumcheck. Finally, the protocol terminates with a claim about the input layer D, which can be checked directly by $\mathcal{V}$. If the check passes, $\mathcal{V}$ accepts the claimed output.

Formally speaking, we denote the number of gates in the i-th layer as S_i and let $s_i = \lceil \log S_i \rceil$. We then define a function $V_i : \{0,1\}^{s_i} \rightarrow \mathbb{F}$ that takes a binary string $b \in \{0,1\}^{s_i}$ and returns the output of gate b in layer i, where b is called the gate label. With this definition, V_0 corresponds to the output of the circuit, and V_D corresponds to the input. As the sumcheck protocol works on $\mathbb{F}$, we then extend V_i to its *multilinear extension*, the unique polynomial $\tilde{V}_i : \mathbb{F}^{s_i} \rightarrow \mathbb{F}$ such that $\tilde{V}_i(x_1, x_2, ..., x_{s_i}) = V_i(x_1, x_2, ..., x_{s_i})$ for all $x_1, x_2, \ldots, x_{s_i} \in \{0,1\}^{s_i}$. As shown in prior work [23], the closed form of $\tilde{V}_i$ can be computed as:

$$\begin{aligned} &\tilde{V}_i(x_1, x_2, ..., x_{s_i}) \\ &= \sum_{b \in \{0,1\}^{s_i}} \prod_{i=1}^{s_i} [((1-x_i)(1-b_i) + x_i b_i) \cdot V_i(b)], \end{aligned} \tag{1}$$

where b_i is i-th bit of b.

With these definitions, we can express the evaluations of $\tilde{V}_i$ as a summation of evaluations of $\tilde{V}_{i+1}$:

$$\begin{aligned} &\alpha_i \tilde{V}_i(u^{(i)}) + \beta_i \tilde{V}_i(v^{(i)}) \\ &= \sum\nolimits_{x,y \in \{0,1\}^{s_{i+1}}} f_i(\tilde{V}_{i+1}(x), \tilde{V}_{i+1}(y)), \end{aligned} \tag{2}$$

where $u^{(i)}, v^{(i)} \in \mathbb{F}^{s_i}$ are random vectors and $\alpha_i, \beta_i \in \mathbb{F}$ are random values. Note here that f_i depends on $\alpha_i, \beta_i, u^{(i)}, v^{(i)}$ and we omit the subscripts for easy interpretation.

With Equation 2, the GKR protocol proceeds as follows. The prover $\mathcal{P}$ first sends the claimed output of the circuit to $\mathcal{V}$. From the claimed output, $\mathcal{V}$ defines polynomial $\tilde{V}_0$ and computes $\tilde{V}_0(u^{(0)})$ and $\tilde{V}_0(v^{(0)})$ for random $u^{(0)}, v^{(0)} \in \mathbb{F}^{s_0}$. $\mathcal{V}$ then picks two random values α_0, β_0 and invokes a sumcheck protocol on Equation 2 with $\mathcal{P}$ for $i = 0$. As described before, at the end of the sumcheck, $\mathcal{V}$ needs an oracle access to the evaluation of f_0 at $u^{(1)}, v^{(1)}$ randomly selected in $\mathbb{F}^{s_1}$. To compute this value, $\mathcal{V}$ asks $\mathcal{P}$ to send $\tilde{V}_1(u^{(1)})$ and $\tilde{V}_1(v^{(1)})$. Other than these two values, f_0 only depends on $\alpha_0, \beta_0, u^{(0)}, v^{(0)}$ and the gates and wiring in layer 0, which are all known to $\mathcal{V}$ and can be computed by $\mathcal{V}$ directly. In this way, $\mathcal{V}$ and $\mathcal{P}$ reduces two evaluations of $\tilde{V}_0$ to two evaluations of $\tilde{V}_1$ in layer 1. $\mathcal{V}$ and $\mathcal{P}$ then repeat the protocol recursively layer by layer. Eventually, $\mathcal{V}$ receives two claimed evaluations $\tilde{V}_D(u^{(D)})$ and $\tilde{V}_D(v^{(D)})$. $\mathcal{V}$ then checks the correctness of

Protocol 1 (Zero Knowledge Argument in [36]). *Let λ be the security parameter, $\mathbb{F}$ be a prime field. Let $C : \mathbb{F}^n \to \mathbb{F}$ be a layered arithmetic circuit over $\mathbb{F}$ with D layers, input* in *and witness w such that $|\mathsf{in}| + |w| \le n$ and $1 = C(\mathsf{in}; w)$.*

- $\mathcal{G}(1^\lambda)$: *set* pp *as* $\mathsf{pp} \leftarrow \mathsf{zkVPD.KeyGen}(1^\lambda)$.
- $\langle \mathcal{P}(\mathsf{pp}, w), \mathcal{V}(\mathsf{pp})\rangle(\mathsf{in})$:

1) *$\mathcal{P}$ selects a random bivariate polynomial R_D. $\mathcal{P}$ commits to the witness of C by sending $\mathsf{com}_D \leftarrow \mathsf{zkVPD.Commit}(\dot{V}_D, r_{V_D}, \mathsf{pp})$ to $\mathcal{V}$, where $\dot{V}_D$ is defined by Equation 3.*
2) *$\mathcal{P}$ randomly selects polynomials $R_i : \mathbb{F}^2 \to \mathbb{F}$ and $\delta_i : \mathbb{F}^{2s_{i+1}+1} \to \mathbb{F}$ for $i = 0, \ldots, D-1$. $\mathcal{P}$ commits to these polynomials by sending $\mathsf{com}_{i,1} \leftarrow \mathsf{zkVPD.Commit}(R_i, r_{R_i}, \mathsf{pp})$ and $\mathsf{com}_{i,2} \leftarrow \mathsf{zkVPD.Commit}(\delta_i, r_{\delta_i}, \mathsf{pp})$ to $\mathcal{V}$. $\mathcal{P}$ also reveals R_0 to $\mathcal{V}$, as V_0 is known to $\mathcal{V}$.*
3) *$\mathcal{V}$ evaluates $\dot{V}_0(u^{(0)})$ and $\dot{V}_0(v^{(0)})$ for randomly chosen $u^{(0)}, v^{(0)} \in \mathbb{F}^{s_0}$.*
4) *For $i = 0, \ldots, D-1$:*
   a) *$\mathcal{P}$ sends $H_i = \sum_{x,y\in\{0,1\}^{s_{i+1}}, z\in\{0,1\}} \delta_i(x, y, z)$ to $\mathcal{V}$.*
   b) *$\mathcal{V}$ picks $\alpha_i, \beta_i, \gamma_i$ randomly in $\mathbb{F}$.*
   c) *$\mathcal{V}$ and $\mathcal{P}$ execute a sumcheck protocol on Equation 4. At the end of the sumcheck, $\mathcal{V}$ receives a claim of f_i' at point $u^{(i+1)}, v^{(i+1)} \in \mathbb{F}^{s_{i+1}}, g_i \in \mathbb{F}$ selected randomly by $\mathcal{V}$.*
   d) *$\mathcal{P}$ opens $R_i(u^{(i)}, g_i)$, $R_i(v^{(i)}, g_i)$ and $\delta_i(u^{(i+1)}, v^{(i+1)}, g_i)$ using* zkVPD.Open. *$\mathcal{P}$ sends $\dot{V}_0(u^{(i+1)})$ and $\dot{V}_0(v^{(i+1)})$ to $\mathcal{V}$.*
   e) *$\mathcal{V}$ validates $R_i(u^{(i)}, g_i)$, $R_i(v^{(i)}, g_i)$ and $\delta_i(u^{(i+1)}, v^{(i+1)}, g_i)$ using* zkVPD.Verify. *If any of them outputs 0, abort and output* 0.
   f) *$\mathcal{V}$ checks the claim of f_i' using $R_i(u^{(i)}, g_i)$, $R_i(v^{(i)}, g_i)$, $\delta_i(u^{(i+1)}, v^{(i+1)}, g_i)$, $\dot{V}_0(u^{(i+1)})$ and $\dot{V}_0(v^{(i+1)})$. If it fails, output* 0.
5) *$\mathcal{P}$ runs $(y_1, \pi_1) \leftarrow \mathsf{zkVPD.Open}(\dot{V}_D, r_{V_D}, u^{(D)}, \mathsf{pp})$, $(y_2, \pi_2) \leftarrow \mathsf{zkVPD.Open}(\dot{V}_D, r_{V_D}, v^{(D)}, \mathsf{pp})$ and sends y_1, π_1, y_2, π_2 to $\mathcal{V}$.*
6) *$\mathcal{V}$ runs $\mathsf{Verify}(\pi_1, y_1, \mathsf{com}_D, u^{(D)}, \mathsf{pp})$ and $\mathsf{Verify}(\pi_2, y_2, \mathsf{com}_D, v^{(D)}, \mathsf{pp})$ and output* 0 *if either check fails. Otherwise, $\mathcal{V}$ checks $\dot{V}_D(u^{(D)}) = y_1$ and $\dot{V}_D(v^{(D)}) = y_2$, and rejects if either fails. If all checks above pass, $\mathcal{V}$ output* 1.

these two claims directly by evaluating $\tilde{V}_D$, which is defined by the input of the circuit. Let GKR.$\mathcal{P}$ and GKR.$\mathcal{V}$ be the algorithms for the GKR prover and verifier, we have the following theorem:

Lemma 1. *[23], [25], [33], [36]. Let $C : \mathbb{F}^n \to \mathbb{F}$ be a depth-D layered arithmetic circuit. $\langle \mathsf{GKR}.\mathcal{P}, \mathsf{GKR}.\mathcal{V}\rangle(C, x)$ is an interactive proof per Definition 1 for the function computed by C on input x with soundness $O(D \log |C|/|\mathbb{F}|)$. The total communication is $O(D \log |C|)$ and the running time of the prover $\mathcal{P}$ is $O(|C|)$. When C has regular wiring pattern[2], the running time of the verifier $\mathcal{V}$ is $O(n + D \log |C|)$.*

Extending GKR to Zero Knowledge Argument. There are two limitations of the GKR protocol: (1) It is not an argument system supporting witness from $\mathcal{P}$, as $\mathcal{V}$ needs to evaluate $\tilde{V}_D$ locally in the last round; (2) It is not zero knowledge, as in each round, both the sumcheck protocol and the two evaluations of $\tilde{V}_i$ leak information about the values in layer i.

To extend the GKR protocol to a zero knowledge argument, Xie et al. [36] address both of the problems using zero knowledge polynomial delegation. Following the approach of [35], [38], [39], to support witness w as the input to the circuit, $\mathcal{P}$ commits to $\tilde{V}_D$ using zkVPD before running the GKR protocol. In the last round of GKR, instead of evaluating $\tilde{V}_D$ locally, $\mathcal{V}$ asks $\mathcal{P}$ to open $\tilde{V}_D$ at two random points $u^{(D)}, v^{(D)}$ selected by $\mathcal{V}$ and validates them using zkVPD.Verify. In this way, $\mathcal{V}$ does not need to access w directly and the soundness still holds because of the soundness guarantee of zkVPD.

To ensure zero knowledge, using the techniques proposed by Chiesa et al. in [21], the prover $\mathcal{P}$ masks the polynomial $\tilde{V}_i$ and the sumcheck protocol by random polynomials so that the proof does not leak information. For correctness and soundness purposes, these random polynomials are committed using the zkVPD protocol and opened at random points chosen by $\mathcal{V}$. In particular, for layer i, the prover selects a random bivariate polynomial $R_i(x_1, z)$ and defines

$$
\begin{aligned}
&\dot{V}_i(x_1, \ldots, x_{s_i}) \\
&\overset{def}{=} \tilde{V}_i(x_1, \ldots, x_{s_i}) + Z_i(x_1, \ldots, x_{s_i}) \sum_{z\in\{0,1\}} R_i(x_1, z), \quad (3)
\end{aligned}
$$

where $Z_i(x) = \prod_{i=1}^{s_i} x_i(1 - x_i)$, i.e., $Z_i(x) = 0$ for all $x \in \{0,1\}^{s_i}$. $\dot{V}_i$ is known as the *low degree extension* of V_i, as $\dot{V}_i(x) = \tilde{V}_i(x) = V_i(x)$ for all $x \in \{0,1\}^{s_i}$. As R_i is randomly selected by $\mathcal{P}$, revealing evaluations of $\dot{V}_i$ does not leak information about V_i, thus the values in the circuit. Additionally, $\mathcal{P}$ selects another random polynomial $\delta_i(x_1, \ldots, x_{s_{i+1}}, y_1, \ldots, y_{s_{i+1}}, z)$ to mask the sumcheck protocol. Let $H_i = \sum_{x,y\in\{0,1\}^{s_{i+1}}, z\in\{0,1\}} \delta_i(x_1, \ldots, x_{s_{i+1}}, y_1, \ldots, y_{s_{i+1}}, z)$, Equation 2 to run sumcheck on becomes

$$
\begin{aligned}
&\alpha_i \dot{V}_i(u^{(i)}) + \beta_i \dot{V}_i(v^{(i)}) + \gamma_i H_i \\
&= \sum\nolimits_{x,y\in\{0,1\}^{s_{i+1}}, z\in\{0,1\}} f_i'(\dot{V}_{i+1}(x), \dot{V}_{i+1}(y), \\
&\quad R_i(u_1^{(i)}, z), R_i(v_1^{(i)}, z), \delta_i(x, y, z)), \quad (4)
\end{aligned}
$$

where $\gamma_i \in \mathbb{F}$ is randomly selected by $\mathcal{V}$, and f_i' is defined by $\alpha_i, \beta_i, \gamma_i, u^{(i)}, v^{(i)}, Z_i(u^{(i)}), Z_i(v^{(i)})$[3]. Now $\mathcal{V}$ and $\mathcal{P}$ can execute the sumcheck and GKR protocol on Equation 4. In each round, $\mathcal{P}$ additionally opens R_i and δ_i at $R_i(u_1^{(i)}, g^{(i)}), R_i(v_1^{(i)}, g^{(i)}), \delta_i(u^{(i+1)}, v^{(i+1)}, g^{(i)})$ for $g^{(i)} \in \mathbb{F}$ randomly selected by $\mathcal{V}$. With these values, $\mathcal{V}$ reduces the correctness of two evaluations $\dot{V}_i(u^{(i)}), \dot{V}_i(v^{(i)})$ to two evaluations $\dot{V}_i(u^{(i+1)}), \dot{V}_i(v^{(i+1)})$ on one layer above like before. In addition, as f_i is masked by δ_i, the sumcheck protocol is zero knowledge; as $\tilde{V}_i$ is masked by R_i, the two evaluations of $\dot{V}_i$ do not leak information. The full zero

[2] "Regular" circuits is defined in [23, Theorem A.1]. Roughly speaking, it means the mutilinear extension of its wiring predicates can be evaluated at a random point in time $O(\log |C|)$.

[3] Formally, f_i' is $I(0, z) f_i(\dot{V}_{i+1}(x), \dot{V}_{i+1}(y)) + I((x, y), 0)(\alpha_i Z_i(u^{(i)}) R(u_1^{(i)}, z) + \beta_i Z_i(v^{(i)}) R(v_1^{(i)}, z)) + \gamma_i \delta_i(x, y, z)$, where $I(a, b)$ is an identity polynomial $I(a, b) = 0$ iff $a = b$. We will not use f_i' explicitly in our constructions later.

knowledge argument protocol in [36] is given in Protocol 1. We have the following theorem:

Lemma 2. *[36]. Let $C : \mathbb{F}^n \to \mathbb{F}$ be a layered arithmetic circuit with D layers, input* in *and witness w. Protocol 1 is a zero knowledge argument of knowledge under Definition 2 for the relation defined by $1 = C(\mathsf{in}; w)$.*

The variable degree of R_i is $O(1)$. $\delta_i(x,y,z) = \delta_{i,1}(x_1) + \ldots + \delta_{i,s_{i+1}}(x_{s_{i+1}}) + \delta_{i,s_{i+1}+1}(y_1) + \ldots + \delta_{i,2s_{i+1}}(y_{s_{i+1}}) + \delta_{i,2s_{i+1}+1}(z)$ is the summation of $2s_{i+1}+1$ univariate polynomials of degree $O(1)$. Other than the zkVPD instantiations, the proof size is $O(D \log |C|)$ and the prover time is $O(|C|)$. When C is regular, the verification time is $O(n + D \log |C|)$.

D. Univariate Sumcheck

Our transparent zkVPD protocol is inspired by the univariate sumcheck protocol recently proposed by Ben-Sasson et al.in [10]. As the name indicates, the univariate sumcheck protocol allows the verifier to validate the result of the sum of a univariate polynomial on a subset $\mathbb{H}$ of the field $\mathbb{F}$: $\mu = \sum_{a \in \mathbb{H}} f(a)$. The key idea of the protocol relies on the following lemma:

Lemma 3. *[17]. Let $\mathbb{H}$ be a multiplicative coset[4] of $\mathbb{F}$, and let $g(x)$ be a univariate polynomial over $\mathbb{F}$ of degree strictly less that $|\mathbb{H}|$. Then $\sum_{a \in \mathbb{H}} g(a) = g(0) \cdot |\mathbb{H}|$.*

Because of Lemma 3, to test the result of $\sum_{a \in \mathbb{H}} f(a)$ for f with degree less than k, we can decompose f into two parts $f(x) = g(x) + Z_{\mathbb{H}}(x) \cdot h(x)$, where $Z_{\mathbb{H}}(x) = \prod_{a \in \mathbb{H}}(x - a)$ (i.e., $Z_{\mathbb{H}}(a) = 0$ for all $a \in \mathbb{H}$), and the degrees of g and h are strictly less than $|\mathbb{H}|$ and $k - |\mathbb{H}|$. This decomposition is unique for every f. As $Z_{\mathbb{H}}(a)$ is always 0 for $a \in \mathbb{H}$, $\mu = \sum_{a \in \mathbb{H}} f(a) = \sum_{a \in \mathbb{H}} g(a) = g(0) \cdot |\mathbb{H}|$ by Lemma 3. Therefore, if the claimed sum μ sent by the prover is correct, $f(x) - Z_{\mathbb{H}}(x) \cdot h(x) - \mu/|\mathbb{H}|$ must be a polynomial of degree less than $|\mathbb{H}|$ with constant term 0, or equivalently polynomial

$$p(x) = \frac{|\mathbb{H}| \cdot f(x) - |\mathbb{H}| \cdot Z_{\mathbb{H}}(x) \cdot h(x) - \mu}{|\mathbb{H}| \cdot x} \tag{5}$$

must be a polynomial of degree less than $|\mathbb{H}| - 1$. To test this, the univariate sumcheck uses a low degree test (LDT) protocol on Reed-Solomon (RS) code, which we define below.

Reed-Solomon Code. Let $\mathbb{L}$ be a subset of $\mathbb{F}$, an RS code is the evaluations of a polynomial $\rho(x)$ of degree less than m ($m < \mathbb{L}$) on $\mathbb{L}$. We use the notation $\rho|_{\mathbb{L}}$ to denote the vector of the evaluations $(\rho(a))_{a \in \mathbb{L}}$, and use $RS[\mathbb{L}, m]$ to denote the set of all such vectors generated by polynomials of degree less than m. Note that any vector of size $|\mathbb{L}|$ can be viewed as some univariate polynomial of degree less than $|\mathbb{L}|$ evaluated on $\mathbb{L}$, thus we use vector and polynomial interchangeably.

Low Degree Test Low degree test allows a verifier to test whether a polynomial/vector belongs to an RS code, i.e., the vector is the evaluations of some polynomial of degree less than m on $\mathbb{L}$.

In our constructions, we use the LDT protocol in [10, Protocol 8.2], which was used to transform an RS-encoded IOP to a regular IOP. It applies the LDT protocol proposed in [7] protocol to a sequence of polynomials $\vec{\rho}$ and their *rational constraint* p, which is a polynomial that can be computed as the division of the polynomials in $\vec{\rho}$. In the case of univariate sumcheck, the sequence of polynomials is $\vec{\rho} = (f, h)$ and the rational constraint is given by Equation 5. We state the properties of the protocol in the following lemma:

Lemma 4. *There exist an LDT protocol $\langle \mathsf{LDT}.\mathcal{P}(\vec{\rho}, p), \mathsf{LDT}.\mathcal{V}(\vec{m}, \deg(p))\rangle(\mathbb{L})$ that is complete and sound with soundness error $O(\frac{|\mathbb{L}|}{|\mathbb{F}|}) + \mathsf{negl}(\kappa)$, given oracle access to evaluations of each polynomial in $\vec{\rho}$ at κ points indexed by $\mathcal{I}$ in $\mathbb{L}$. The proof size and the verification time are $O(\log |\mathbb{L}|)$ other than the oracle access, and the prover time is $O(\mathbb{L})$.*

The LDT protocol can be made zero knowledge in a straight-forward way by adding a random polynomial of degree max in $\vec{\rho}$. That is, there exists a simulator $\mathcal{S}$ such that given the random challenges of $\mathcal{I}$ of any PPT algorithm $\mathcal{V}^*$, it can simulate the view of $\mathcal{V}^*$ such that $\mathsf{View}(\langle \mathsf{LDT}.\mathcal{P}(\vec{\rho}, p), \mathcal{V}^*(\vec{m}, \deg(p))\rangle(\mathbb{L})) \approx \mathcal{S}^{\mathcal{V}^*}(\deg(p))$. In particular, $\mathcal{S}$ generates $p^* \in RS[\mathbb{L}, \deg(p)]$ and can simulate the view of any sequence of random polynomials $\vec{\rho}^*$ subject to the constraint that their evaluations at points indexed by $\mathcal{I}$ are consistent with the oracle access of p^*.

Merkle Tree. Merkle hash tree proposed by Ralph Merkle in [29] is a common primitive to commit a vector and open it at an index with logarithmic proof size and verification time. It consists of three algorithms:

- $\mathsf{root}_c \leftarrow \mathsf{MT.Commit}(c)$
- $(c_{idx}, \pi_{idx}) \leftarrow \mathsf{MT.Open}(idx, c)$
- $(1, 0) \leftarrow \mathsf{MT.Verify}(\mathsf{root}_c, idx, c_{idx}, \pi_{idx})$

The security follows the collision-resistant property of the hash function used to construct the Merkle tree.

With these tools, the univariate sumcheck protocol works as follows. To prove $\mu = \sum_{a \in \mathbb{H}} f(a)$, the verifier and the prover picks $\mathbb{L}$, a multiplicative coset of $\mathbb{F}$ and a superset of $\mathbb{H}$, where $|\mathbb{L}| > k$. $\mathcal{P}$ decompose $f(x) = g(x) + Z_{\mathbb{H}}(x) \cdot h(x)$ as defined above, and computes the vectors $f|_{\mathbb{L}}$ and $h|_{\mathbb{L}}$. $\mathcal{P}$ then commits to these two vectors using Merkle trees. $\mathcal{P}$ then defines a polynomial $p(x) = \frac{|\mathbb{H}| \cdot f(x) - |\mathbb{H}| \cdot Z_{\mathbb{H}}(x) \cdot h(x) - \mu}{|\mathbb{H}| \cdot x}$, which is a rational constraint of f and h. As explained above, in order to ensure the correctness of μ, it suffices to test that the degree of $(f, h), p$ is less than $(k, k - |\mathbb{H}|), |\mathbb{H}| - 1$, which is done through the low degree test. At the end of the LDT, $\mathcal{V}$ needs oracle access to κ points of $f|_{\mathbb{L}}$ and $h|_{\mathbb{L}}$. $\mathcal{P}$ sends these points with their Merkle tree proofs, and $\mathcal{V}$ validates their correctness. The formal protocol and the lemma is presented in Appendix A. As shown in [10], it suffices to set $|\mathbb{L}| = O(|\mathbb{H}|)$.

[4] In [10], the protocols are mainly using additive cosets. We require $\mathbb{H}$ to be a multiplicative coset for our constructions over prime fields and extensions. The univariate sumsheck on multiplicative cosets is also stated in [10].

Protocol 2 (Verifiable Polynomial Delegation)**.** *Let $\mathcal{F}$ be a family of ℓ-variate polynomial over $\mathbb{F}$ with variable-degree d and $N = (d+1)^\ell$.We use $\mathcal{W}_{\ell,d} = \{W_i(x_1,\ldots,x_\ell)\}_{i=1}^N$ to denote the collection of all monomials in $\mathcal{F}$. $r_f = \perp$ and we omit if in the algorithms.*

- $\mathsf{pp} \leftarrow \mathsf{KeyGen}(1^\lambda)$*: Pick a hash function from the collision-resistant hash function family for Merkle tree. Find a multiplicative coset $\mathbb{H}$ of $\mathbb{F}$ such that $|\mathbb{H}| = (d+1)^\ell$. Find a multiplicative coset $\mathbb{L}$ of $\mathbb{F}$ such that $|\mathbb{L}| = O(|\mathbb{H}|) > 2|\mathbb{H}|$ and $\mathbb{H} \subset \mathbb{L} \subset \mathbb{F}$.*
- $\mathsf{com} \leftarrow \mathsf{Commit}(f, \mathsf{pp})$*: For a polynomial $f \in \mathcal{F}$ of the form $f(x) = \sum_{i=1}^N c_i W_i(x)$, find the unique univariate polynomial $l(x): \mathbb{F} \to \mathbb{F}$ such that $l|_\mathbb{H} = (c_1,\ldots,c_N)$. $\mathcal{P}$ evaluates $l|_\mathbb{L}$ and runs $\mathsf{root}_l \leftarrow \mathsf{MT.Commit}(l|_\mathbb{L})$. Output $\mathsf{com} = \mathsf{root}_l$.*
- $((\mu,\pi);\{0,1\}) \leftarrow \langle \mathsf{Open}(f), \mathsf{Verify}(\mathsf{com})\rangle(t, \mathsf{pp})$*: This is an interactive protocol between $\mathcal{P}$ and $\mathcal{V}$.*
  1) *$\mathcal{P}$ computes $\mu = f(t)$ and sends it to $\mathcal{V}$.*
  2) *$\mathcal{P}$ evaluates $T = (W_1(t),\ldots,W_N(t))$. $\mathcal{P}$ finds the unique univariate polynomial $q(x): \mathbb{F} \to \mathbb{F}$ such that $q|_\mathbb{H} = T$.*
  3) *$\mathcal{P}$ computes $l(x)\cdot q(x)$. $\mathcal{P}$ uniquely decomposes $l(x)\cdot q(x) = g(x) + Z_\mathbb{H}(x)\cdot h(x)$, where $Z_\mathbb{H}(x) = \prod_{a\in\mathbb{H}}(x-a)$ and the degrees of g and h are strictly less than $|\mathbb{H}|$ and $|\mathbb{H}|-1$. $\mathcal{P}$ evaluates $h|_\mathbb{L}$ and runs $\mathsf{root}_h \leftarrow \mathsf{MT.Commit}(h|_\mathbb{L})$ and sends root_h to $\mathcal{V}$.*
  4) *Let $p(x) = \frac{|\mathbb{H}|\cdot l(x)\cdot q(x) - \mu - |\mathbb{H}|\cdot \mathbb{Z}_\mathbb{H}(x)h(x)}{|\mathbb{H}|\cdot x}$. $\mathcal{P}$ and $\mathcal{V}$ invoke a low degree test: $\langle \mathsf{LDT}.\mathcal{P}((l\cdot q, h), p), \mathsf{LDT}.\mathcal{V}((2|\mathbb{H}|-1, |\mathbb{H}|-1), |\mathbb{H}|-1)\rangle(\mathbb{L})$. If the test fails, $\mathcal{V}$ aborts and output 0. Otherwise, at then end of the test, $\mathcal{V}$ needs oracle access to κ points of $l(x)\cdot q(x), h(x)$ and $p(x)$ at indices $\mathcal{I}$.*
  5) *For each index $i \in \mathcal{I}$, let a_i be the corresponding point in $\mathbb{L}$. $\mathcal{P}$ opens $(l(a_i), \pi_i^l) \leftarrow \mathsf{MT.Open}(i, l|_\mathbb{L})$ and $(h(a_i), \pi_i^h) \leftarrow \mathsf{MT.Open}(i, h|_\mathbb{L})$.*
  6) *$\mathcal{V}$ executes $\mathsf{MT.Verify}(\mathsf{root}_l, i, l(a_i), \pi_i^l)$ and $\mathsf{MT.Verify}(\mathsf{root}_h, i, h(a_i), \pi_i^h)$ for all points opened by $\mathcal{P}$. If any verification fails, abort and output 0.*
  7) *To complete the low degree test, $\mathcal{P}$ and $\mathcal{V}$ runs $\langle \mathsf{GKR}.\mathcal{P}, \mathsf{GKR}.\mathcal{V}\rangle(C, t)$, where circuit C computes the evaluations of $q|_\mathbb{L}$ and outputs the elements $q(a_i)$ for $i \in \mathcal{I}$ (see Figure 1). If any of the checks in GKR fails, $\mathcal{V}$ aborts and outputs 0.*
  8) *For each $i \in \mathcal{I}$, $\mathcal{V}$ computes $l(a_i)\cdot q(a_i)$. Together with $h(a_i)$, $\mathcal{V}$ completes the low degree test. If all checks above pass, $\mathcal{V}$ outputs 1.*

III. Transparent Zero Knowledge Polynomial Delegation

In this section, we present our main construction, a zero knowledge verifiable polynomial delegation scheme without trusted setup. We first construct a VPD scheme that is correct and sound, then extend it to be zero knowledge. Our construction is inspired by the univariate sumcheck [10] described in Section II-D.

Our main idea is as follows. To evaluate an ℓ-variate polynomial f with variable degree d at point $t = (t_1,\ldots,t_\ell)$, we model the evaluation as the inner product between the vector of coefficients in f and the vector of all monomials in f evaluated at t. Formally speaking, let $N = |\mathcal{W}_{\ell,d}| = (d+1)^\ell$ be the number of possible monomials in an ℓ-variate polynomial with variable degree d, and let $c = (c_1,\ldots,c_N)$ be the coefficients of f in the order defined by $\mathcal{W}_{\ell,d}$ such that $f(x_1,\ldots,x_\ell) = \sum_{i=1}^N c_i W_i(x)$, where $W_i(x)$ is the i-th monomial in $\mathcal{W}_{\ell,d}$. Define the vector $T = (W_1(t),\ldots,W_N(t))$, then naturally the evaluation equals $f(t) = \sum_{i=1}^N c_i\cdot T_i$, the inner product of the two vectors. We then select a multiplicative coset $\mathbb{H}$ such that $|\mathbb{H}| = N$, [5] and interpolate vectors c and T to find the unique univariate polynomials that evaluate to c and T on $\mathbb{H}$. We denote the polynomials as $l(x)$ and $q(x)$ such that $l|_\mathbb{H} = c$ and $q|_\mathbb{H} = T$. With these definitions, $f(t) = \sum_{i=1}^N c_i\cdot T_i = \sum_{a\in\mathbb{H}} l(a)\cdot q(a)$, which is the sum of the polynomial $l(x)\cdot q(x)$ on $\mathbb{H}$. The verifier can check the evaluation through a univariate sumcheck protocol with the prover. The detailed protocol is presented in step 1-4 of Protocol 2.

Up to this point, the construction for validating the inner product between a vector committed by $\mathcal{P}$ and a public vector is similar to and simpler than the protocols to check linear constraints proposed in [4], [10]. However, naively applying the univariate sumcheck protocol incurs a linear overhead for the verifier. This is because as described in Section II-D, at the end of the univariate sumcheck, due to the low degree test, the verifier needs oracle access to the evaluations of $l(x)\cdot q(x)$ at κ points on $\mathbb{L}$, a superset of $\mathbb{H}$. As $l(x)$ is defined by c, i.e. the coefficients of f, the prover can commit to $l|_\mathbb{L}$ at the beginning of the protocol, and opens to points the verifier queries with their Merkle tree proofs. $q(x)$, however, is defined by the public vector T, and the verifier has to evaluate it locally, which takes linear time. This is the major reason why the verification time in the zero knowledge proof schemes for generic arithmetic circuits in [4], [10] is linear in the size of the circuits.

Reducing the verification time. In this paper, we propose an approach to reduce the cost of the verifier to poly-logarithmic for VPD. We observe that in our construction, though the size of T and $q(x)$ is linear in N, it is defined by only $\ell = O(\log N)$ values of the evaluation point t. This means that the oracle access of κ points of $q(x)$ can be modeled as a function that: (1) Takes t as input, evaluates all monomials $W_i(t)$ for all $W_i \in \mathcal{W}_{\ell,d}$ as a vector T; (2) Extrapolates the vector T to find polynomial $q(x)$, and evaluates $q(x)$ on $\mathbb{L}$; (3) Outputs κ points of $q|_\mathbb{L}$ chosen by the verifier. Although the size of the function modeled as an arithmetic circuit is $\Omega(N)$ with $O(\log N)$ depth, and the size of its input and output is only $O(\log N + \kappa)$. Therefore, instead of evaluating the function locally, the verifier can delegate this computation to the prover, and validate the result using the GKR protocol, as presented in Section II-C. In this way, we eliminate the linear overhead to evaluate these points locally, making the verification time of the overall VPD protocol poly-logarithmic. The formal protocol is presented in Protocol 2.

To avoid any asymptotic overhead for the prover, we also design an efficient layered arithmetic circuit for the function mentioned above. The details of the circuit are presented in Figure 1. In particular, in the first part, each value t_i in the input t is raised to powers of $0, 1, \ldots, d$. Then they are expanded to T, the evaluations of all monomials in $\mathcal{W}_{\ell,d}$,

[5] If such coset does not exist, we can pad N to the nearest number with a coset of that size, and pad vector T with 0s at the end.

Input: $t = (t_1, \dots, t_\ell)$
Output: $q()$

1) Computing vector $T = (W_1(t), \dots, W_N(t))$:
   - Compute $(t_i^0, t_i^1, \dots, t_i^d)$ for $i = 1, \dots, \ell$.
   - Initialize vector $T_0 = (1)$.
   - For $i = 1, \dots, \ell$: $T_i = (t_i^0 \cdot T_{i-1}, \dots, t_i^d \cdot T_{i-1})$, where " $\cdot$ " here is scalar multiplication between a number and a vector and "," means concatenation. Set $T = T_\ell$.
2) Computing $q|_{\mathbb{L}}$:
   - $q|_{\mathbb{L}} = \mathsf{FFT}(\mathsf{IFFT}(T, \mathbb{H}), \mathbb{L})$
3) Outputting evaluations indexed by I_q:

Fig. 1: Arithmetic circuit C computing evaluations of $q(x)$ at κ points in $\mathbb{L}$ indexed by $\mathcal{I}$.

by multiplying one t_i at a time through a $(d+1)$-ary tree. The size of this part is $O(N) = O((d+1)^\ell)$ and the depth is $O(\log d + \ell)$. In the second part, the polynomial $q(x)$ and the vector $q|_{\mathbb{L}}$ is computed from T directly using FFTs. We first construct a circuit for an inverse FFT to compute the coefficients of polynomial $q(x)$ from its evaluations T. Then we run an FFT to evaluate $q|_{\mathbb{L}}$ from the coefficients of $q(x)$. We implement FFT and IFFT using the Butterfly circuit [22]. The size of the circuit is $O(N \log N)$ and the depth is $O(\log N)$. Finally, κ points are selected from $q|_{\mathbb{L}}$. As the whole delegation of the GKR protocol is executed at the end in Protocol 2 after these points being fixed by the verifier, the points to output are directly hard-coded into the circuit with size $O(\kappa)$ and depth 1. No heavy techniques for random accesses in the circuit is needed. Therefore, the whole circuit is of size $O(N \log N)$ and depth $O(\log N)$, with ℓ inputs and κ outputs.

Theorem 1. *Protocol 2 is a verifiable polynomial delegation protocol that is complete and sound under Definition 3.*

We will omit the proof due to limited space.

Efficiency. The running time of Commit is $O(N \log N)$. C in step 7 is a regular circuit with size $O(N \log N)$, depth $O(\ell + \log d)$ and size of input and output $O(\ell + \kappa)$. By Lemma 1 and 5, the prover time is $O(N \log N)$, the proof size and the verification time are $(\log^2 N)$.

Extending to other ZKP schemes. We notice that our technique can be potentially applied to generic zero knowledge proof schemes in [4], [10] to improve the verification time for circuits/constraint systems with succinct representation. As mentioned previously, the key step that introduces linear verification time in these schemes is to check a linear constraint system, i.e., $y = \mathbf{A}w$, where w is a vector of all values on the wires of the circuit committed by the prover, and $\mathbf{A}$ is a public matrix derived from the circuit such that $\mathbf{A}w$ gives a vector of left inputs to all multiplication gates in the circuit. (This check is executed 2 more times to also give right inputs and outputs.) To check the relationship, it is turned into a vector inner product $\mu = ry = r\mathbf{A} \cdot w$ by multiplying both sides by a random vector r. Similar to our naive protocol to check inner product, the verification time is linear in order

Protocol 3 (Zero Knowledge Verifiable Polynomial Delegation)**.** *Let $\mathcal{F}$ be a family of ℓ-variate polynomial over $\mathbb{F}$ with variable-degree d and $N = (d+1)^\ell$. We use $\mathcal{W}_{\ell,d} = \{W_i(x_1, \dots, x_\ell)\}_{i=1}^N$ to denote the collection of all monomials in $\mathcal{F}$.*

- $\mathsf{pp} \leftarrow \mathsf{zkVPD.KeyGen}(1^\lambda)$*: Same as* KeyGen *in Procotol 2. Define $\mathbb{U} = \mathbb{L} - \mathbb{H}$.*
- $\mathsf{com} \leftarrow \mathsf{Commit}(f, r_f, \mathsf{pp})$*: For a polynomial $f \in \mathcal{F}$ of the form $f(x) = \sum_{i=1}^N c_i W_i(x)$, find the unique univariate polynomial $l(x) : \mathbb{F} \to \mathbb{F}$ such that $l|_{\mathbb{H}} = (c_1, \dots, c_N)$. $\mathcal{P}$ samples a polynomial $r(x)$ with degree κ randomly and sets $l'(x) = l(x) + Z_{\mathbb{H}}(x) \cdot r(x)$, where $Z_{\mathbb{H}}(x) = \prod_{a \in \mathbb{H}}(x - a)$. $\mathcal{P}$ evaluates $l'|_{\mathbb{U}}$ and runs $\mathsf{root}_{l'} \leftarrow \mathsf{MT.Commit}(l'|_{\mathbb{U}})$. Output* $\mathsf{com} = \mathsf{root}_{l'}$.
- $((\mu, \pi); \{0,1\}) \leftarrow \langle \mathsf{Open}(f, r_f), \mathsf{Verify}(\mathsf{com}) \rangle (t, \mathsf{pp})$*: This is an interactive protocol between $\mathcal{P}$ and $\mathcal{V}$. It replaces the univariate sumscheck on $l(x) \cdot q(x)$ by $l'(x) \cdot q(x) + \alpha s(x)$ and $\mathbb{L}$ by $\mathbb{U}$ in Protocol 2.*

1) $\mathcal{P}$ computes $\mu = f(t)$ and sends it to $\mathcal{V}$.
2) $\mathcal{P}$ evaluates $T = (W_1(t), \dots, W_N(t))$. $\mathcal{P}$ finds the unique univariate polynomial $q(x) : \mathbb{F} \to \mathbb{F}$ such that $q|_{\mathbb{H}} = T$.
3) $\mathcal{P}$ samples randomly a degree $2|\mathbb{H}| + \kappa - 1$ polynomial $s(x)$. $\mathcal{P}$ sends $\mathcal{V}$ $S = \sum_{a \in \mathbb{H}} s(a)$ and $\mathsf{root}_s \leftarrow \mathsf{MT.Commit}(s|_{\mathbb{U}})$.
4) $\mathcal{V}$ picks $\alpha \in \mathbb{F}$ randomly and sends it to $\mathcal{P}$.
5) $\mathcal{P}$ computes $\alpha l'(x) \cdot q(x) + s(x)$. $\mathcal{P}$ uniquely decomposes it as $g(x) + Z_{\mathbb{H}}(x) \cdot h(x)$, where the degrees of g and h are strictly less than $|\mathbb{H}|$ and $|\mathbb{H}| + \kappa$. $\mathcal{P}$ evaluates $h|_{\mathbb{U}}$ and sends $\mathsf{root}_h \leftarrow \mathsf{MT.Commit}((h|_{\mathbb{U}})$ to $\mathcal{V}$.
6) Let $p(x) = \frac{|\mathbb{H}| \cdot (\alpha l'(x) \cdot q(x) + s(x)) - (\alpha\mu + S) - |\mathbb{H}| \cdot \mathbb{Z}_{\mathbb{H}}(x) h(x)}{|\mathbb{H}| \cdot x}$. $\mathcal{P}$ and $\mathcal{V}$ invoke the low degree test: $\langle \mathsf{LDT}.\mathcal{P}((l' \cdot q, h, s), p), \mathsf{LDT}.\mathcal{V}((2|\mathbb{H}| + \kappa, |\mathbb{H}| + \kappa, 2|\mathbb{H}| + \kappa), |\mathbb{H}| - 1) \rangle (\mathbb{U})$. If the test fails, $\mathcal{V}$ aborts and output 0. Otherwise, at the end of the test, $\mathcal{V}$ needs oracle access to κ points of $l'(x) \cdot q(x), h(x), s(x)$ and $p(x)$ at indices $\mathcal{I}$.
7) For each index $i \in \mathcal{I}$, let a_i be the corresponding point in $\mathbb{U}$. $\mathcal{P}$ opens $(l'(a_i), \pi_i^{l'}) \leftarrow \mathsf{MT.Open}(i, l'|_{\mathbb{U}})$, $(h(a_i), \pi_i^h) \leftarrow \mathsf{MT.Open}(i, h|_{\mathbb{U}})$ and $(s(a_i), \pi_i^s) \leftarrow \mathsf{MT.Open}(i, s|_{\mathbb{U}})$.
8) $\mathcal{V}$ executes $\mathsf{MT.Verify}(\mathsf{root}_{l'}, i, l'(a_i), \pi_i^{l'})$, $\mathsf{MT.Verify}(\mathsf{root}_h, i, h(a_i), \pi_i^h)$ and $\mathsf{MT.Verify}(\mathsf{root}_s, i, s(a_i), \pi_i^s)$ for all points opened by $\mathcal{P}$. If any verification fails, abort and output 0*.*
9) To complete the low degree test, $\mathcal{P}$ and $\mathcal{V}$ runs $\langle \mathsf{GKR}.\mathcal{P}, \mathsf{GKR}.\mathcal{V} \rangle (C, t)$, where circuit C computes the evaluations of $q|_{\mathbb{U}}$ and outputs the elements $q(a_i)$ for $i \in \mathcal{I}$. If any of the checks in GKR fails, $\mathcal{V}$ aborts and outputs 0.
10) For each $i \in \mathcal{I}$, $\mathcal{V}$ computes $l'(a_i) \cdot q(a_i)$. Together with $h(a_i)$ and $s(a_i)$, $\mathcal{V}$ completes the low degree test. If all checks above pass, $\mathcal{V}$ outputs 1.

to evaluate the polynomial defined by $r\mathbf{A}$ at κ points. With our new protocol, if the circuit can be represented succinctly in sublinear or logarithmic space, $\mathbf{A}$ can be computed by a function with sublinear or logarithmic number of inputs. We can use the GKR protocol to delegate the computation of $r\mathbf{A}$ and the subsequent evaluations to the prover in a similar way as in our construction, and the verification time will only depend on the space to represent the circuit, but not on the total size of the circuit. This is left as a future work.

A. Achieving Zero Knowledge

Our VPD protocol in Protocol 2 is not zero knowledge. Intuitively, there are two places that leak information about the polynomial f: (1) In step 6 of Protocol 2, $\mathcal{P}$ opens evaluations of $l(x)$, which is defined by the coefficients of f; (2) In step 4, $\mathcal{P}$ and $\mathcal{V}$ execute low degree tests on $(l(x) \cdot q(x), h(x)), p(x)$ and the proofs of LDT reveal information about the polynomials, which are related to f.

To make the protocol zero knowledge, we take the standard approaches proposed in [4], [10]. To eliminate the former leakage of queries on $l(x)$, the prover picks a random degree κ polynomial $r(x)$ and masks it as $l'(x) = l(x) + Z_{\mathbb{H}}(x) \cdot r(x)$, where as before, $Z_{\mathbb{H}}(x) = \prod_{a \in \mathbb{H}}(x - a)$. Note here that $l'(a) = l(a)$ for $a \in \mathbb{H}$, yet any κ evaluations of $l'(x)$ outside $\mathbb{H}$ do not reveal any information about $l(x)$ because of the masking polynomial $r(x)$. The degree of $l'(x)$ is $|\mathbb{H}| + \kappa$, and we denote domain $\mathbb{U} = \mathbb{L} - \mathbb{H}$.

To eliminate the latter leakage, $\mathcal{P}$ samples a random polynomial $s(x)$ of the same degree as $l'(x) \cdot q(x)$, sends $S = \sum_{a \in \mathbb{H}} s(a)$ to $\mathcal{V}$ and runs the univariate sumcheck protocol on their random linear combination: $\alpha\mu + S = \sum_{a \in \mathbb{H}}(\alpha l'(x) \cdot q(x) + s(x))$ for a random $\alpha \in \mathbb{F}$ chosen by $\mathcal{V}$. This ensures that both μ and S are correctly computed because of the random linear combination and the linearity of the univariate sumcheck, while leaking no information about $l'(x) \cdot q(x)$ during the protocol, as it is masked by $s(x)$.

One advantage of our construction is that the GKR protocol used to compute evaluations of $q(x)$ in step 7 of Protocol 2 remains unchanged in the zero knowledge version of the VPD. This is because $q(x)$ and its evaluations are independent of the polynomial f or any prover's secret input. Therefore, it suffices to apply the plain version of GKR without zero knowledge, avoiding any expensive cryptographic primitives.

The full protocol for our zkVPD is presented in Protocol 3. Note that all the evaluations are on $\mathbb{U} = \mathbb{L} - \mathbb{H}$ instead of $\mathbb{L}$, as evaluations on $\mathbb{H}$ leaks information about the original $l(x)$. $s(x)$ is also committed and opened using Merkle tree for the purpose of correctness and soundness. The efficiency of our zkVPD protocol is asymptotically the same as our VPD protocol in Protocol 2, and the concrete overhead in practice is also small. We have the following theorem:

Theorem 2. *Protocol 3 is a zero knowledge verifiable polynomial delegation scheme by Definition 3.*

We will omit the proof due to limited space.

IV. Zero Knowledge Argument

Following the framework of [36], we can instantiate the zkVPD in Protocol 1 with our new construction of transparent zkVPD in Protocol 3 to obtain a zero knowledge argument of knowledge scheme for layered arithmetic circuits without trusted setup.

The full protocol of our transparent zero knowledge argument scheme can be summarized by the following theorem:

Theorem 3. *For a finite field $\mathbb{F}$ and a family of layered arithmetic circuit $\mathcal{C}_{\mathbb{F}}$ over $\mathbb{F}$, Protocol 1 with the new zkVPD is a zero knowledge argument of knowledge for the relation*

$$\mathcal{R} = \{(C, x; w) : C \in \mathcal{C}_{\mathbb{F}} \wedge C(x; w) = 1\},$$

as defined in Definition 2.

Moreover, for every $(C, x; w) \in \mathcal{R}$, the running time of $\mathcal{P}$ is $O(|C| + n \log n)$ field operations, where $n = |x| + |w|$. The running time of $\mathcal{V}$ is $O(|x| + D \cdot \log |C| + \log^2 n)$ if C is regular with D layers. $\mathcal{P}$ and $\mathcal{V}$ interact $O(D \log |C|)$ rounds and the total communication (proof size) is $O(D \log |C| + \log^2 n)$. In case D is $\mathsf{polylog}(|C|)$, the protocol is a succinct argument.

Removing interactions. Similar to [36], our construction can be made non-interactive in the random oracle model using Fiat-Shamir heuristic [24]. As shown in recent work [11], [20], applying Fiat-Shamir on the GKR protocol only incurs a polynomial soundness loss in the number of rounds. f

V. Implementation and Evaluation

We implement Virgo, a zero knowledge proof system based on our construction in Section IV. The system is implemented in C++. There are around 700 lines of code for our transparent zkVPD protocol and 2000 lines for the GKR part.

Hardware. We run all of the experiments on AMD Ryzen™ 3800X Processor with 64GB RAM. Our current implementation is not parallelized and we only use a single CPU core in the experiments. We report the average running time of 10 executions, unless specified otherwise.

A. Choice of Field with Efficient Arithmetic

One important optimization we developed during the implementation is on the choice of the underlying field. Our scheme is transparent and does not use any discrete log or bilinear pairing as in [35], [36], [38], [39]. However, there is one requirement on the finite field: in order to run the low degree test protocol in [7], either the field is an extension of $\mathbb{F}_2$, or there exists a multiplicative subgroup of order 2^k in the field for large enough k (one can think of $2^k \geq |\mathbb{L}| = O(|\mathbb{H}|) = O(n)$). Existing zero knowledge proof systems that use the LDT protocol as a building block such as Stark [6] and Aurora [10] run on the extension fields $\mathbb{F}_{2^{64}}$ and $\mathbb{F}_{2^{192}}$. Modern CPUs (e.g., AMD Ryzen™ 3800X Processor) have built-in instructions for field arithmetics on these extension fields, which improves the performance of these systems significantly. However, the drawback is that the arithmetic circuits representing the statement of ZKP must also operate on the same field, and the additions (multiplications) are different from integer or modular additions (multiplications) that are commonly used in the literature. Because of this, Stark [6] has to design a special SHA-256 circuit on $\mathbb{F}_{2^{64}}$, and Aurora [10] only reports the performance versus circuit size (number of constraints), but not on any commonly used functions.

One could also use a prime field p with an order-2^k multiplicative subgroup. Equivalently, this requires that 2^k is a factor of $p - 1$. In fact, there exist many such primes and Aurora [10] also supports prime fields. However, the speed of field arithmetic is much slower than extension fields of $\mathbb{F}_2$ (see Table I).

In this paper, we provide an alternative to achieve the best of both cases. A first attempt is to use Mersenne primes,

primes that can be expressed as $p = 2^m - 1$ for integers m. As shown in [23], [33], multiplications modulo Mersenne primes is known to be very efficient. However, Mersenne primes do not satisfy the requirement of the LDT, as $p-1 = 2^m - 2 = 2 \cdot (2^{m-1} - 1)$ only has a factor 2^1. Instead, we propose to use the extension field of a Mersenne prime $\mathbb{F}_{p^2}$.The multiplicative group of $\mathbb{F}_{p^2}$ is a cyclic group of order $p^2-1 = (2^m-1)^2-1 = 2^{2m}-2^{m+1} = 2^{m+1}(2^{m-1}-1)$, thus it has a multiplicative subgroup of order 2^{m+1}, satisfying the requirement of LDT when m is reasonably large. Meanwhile, to construct an arithmetic circuit representing the statement of the ZKP, we still encode all the values in the first slot of the polynomial ring defined by $\mathbb{F}_{p^2}$. In this way, the additions and multiplications in the circuit are on $\mathbb{F}_p$ and our system can take the same arithmetic circuits over prime fields in prior work. Meanwhile, the LDT, zkVPD and GKR protocol are executed on $\mathbb{F}_{p^2}$, preserving the soundness over the whole field.

With this alternative approach, we can implement modular multiplications on $\mathbb{F}_{p^2}$ using 3 modular multiplications on $\mathbb{F}_p$. (The modular multiplication is analog to multiplications of complex numbers.) In our implementation, we choose Mersenne prime $p = 2^{61} - 1$, thus our system provides 100+ bits of security. We implement modular multiplications on $\mathbb{F}_p$ for $p = 2^{61} - 1$ with only one integer multiplication in C++ (two 64-bit integers to one 128-bit integer) and some bit operations. As shown in Table I, the field arithmetic on $\mathbb{F}_{p^2}$ is comparable to $\mathbb{F}_{2^{64}}$, 2× faster than $\mathbb{F}_{2^{192}}$ and 4× faster than a 128-bit prime field. Encoding numbers in $\mathbb{F}_p$ for $p = 2^{61} - 1$ is enough to avoid overflow for all computations used in our experiments in Section V-B. For other computations requiring larger field, one can set p as $2^{89}-1, 2^{107}-1$ or $2^{127}-1$, which incurs a moderate slow down. For example, the multiplication over $\mathbb{F}_{p^2}$ for $p = 2^{89} - 1$ is 2.7× slower than $p = 2^{61} - 1$.

This optimization can also be applied to Stark [6] and Aurora [10], which use the same LDT in [7]. Currently they run on $\mathbb{F}_{2^{64}}$ and $\mathbb{F}_{2^{192}}$ and their performances are reported in Section V-C. With our optimization, they can run on $\mathbb{F}_{p^2}$ with similar efficiency while taking arithmetic circuits in $\mathbb{F}_p$.

B. Performance of zkVPD

In this section, we present the performance of our new transparent zkVPD protocol, and compare it with the existing approach based on bilinear maps. We use the open-source code of [36], which implements the zkVPD scheme presented in [38]. For our new zkVPD protocol, we implement the univariate sumcheck and the low degree test described in Section II-D. We set the repetition parameter κ in Lemma 4 as 33, and the rate of the RS code as 32 (i.e., $|\mathbb{L}| = 32|\mathbb{H}|$). These parameters provide 100+ bits of security, based on Theorem 1.2 and Conjecture 1.5 in [7], and are consistent with the implementation of Aurora [10]. In addition, we use the field $\mathbb{F}_{p^2}$ with $p = 2^{61} - 1$, which has a multiplicative subgroup of order 2^{m+1}. Thus $|\mathbb{L}|$ can be as big as 2^{60} and the size of the witness $|\mathbb{H}|$ is up to 2^{55}. We pad the size of the witness to a power of 2, which introduces an overhead of at most 2×.

	128-bit prime	$\mathbb{F}_{2^{64}}$	$\mathbb{F}_{2^{192}}$	Our field
$+$	6.29ns	2.16ns	4.75ns	1.23ns
$\times$	30.2ns	7.29ns	15.8ns	8.27ns

TABLE I: Speed of basic arithmetic on different fields. The time is averaged over 100 million runs and is in nanosecond.

Figure 2 shows the prover time, verification time and proof size of the two schemes. We fix the variable degree of the polynomial as 1 and vary the number of variables from 12 to 20. The size of the multilinear polynomial is 2^{12} to 2^{20}. As shown in the figure, the prover time of our new zkVPD scheme is 8-10× faster than the pairing-based one. It only takes 11.7s to generate the proof for a polynomial of size 2^{20}. This is because our new scheme does not use any heavy cryptographic operations, while the scheme in [38] uses modular exponentiations on the base group of a bilinear map. In terms of the asymptotic complexity, though the prover time is claimed to be linear in [38], there is a hidden factor of $\log|\mathbb{F}|$ because of the exponentiations. The prover complexity of our scheme is $O(n \log n)$, which is strictly better than $O(n \log |\mathbb{F}|)$ field operations. Additionally, as explained in Section V-A, our scheme is on the extension field of a Mersenne prime, while the scheme in [38] is on a 254-bit prime field with bilinear maps, the basic arithmetic of which is slower.

The verification time of our zkVPD scheme is also comparable to that of [38]. For $n = 2^{20}$, it takes 12.4ms to validate the proof in our scheme, and 20.9ms in [38].

The drawback of our scheme is the proof size. As shown in Figure 2(c), the proof size of our scheme is 30-40× larger than that of [38]. This is due to the opening of the commitments using Merkle tree, which is a common disadvantage of all IOP-based schemes [4], [6], [10]. The proof size of our scheme can be improved by a factor of $\log n$ using the vector commitment scheme with constant-size proofs in [15], with a compromise on the prover time. This is left as a future work.

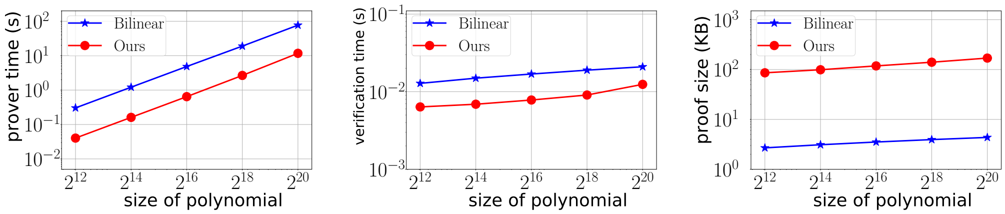
Finally, the scheme in [38] requires a trusted setup phase, which takes 12.6s for $n = 2^{20}$. We remove the trusted setup completely in our new scheme.

C. Performance of Virgo

In this section, we present the performance of our ZKP scheme, Virgo, and compare it with existing ZKP systems.

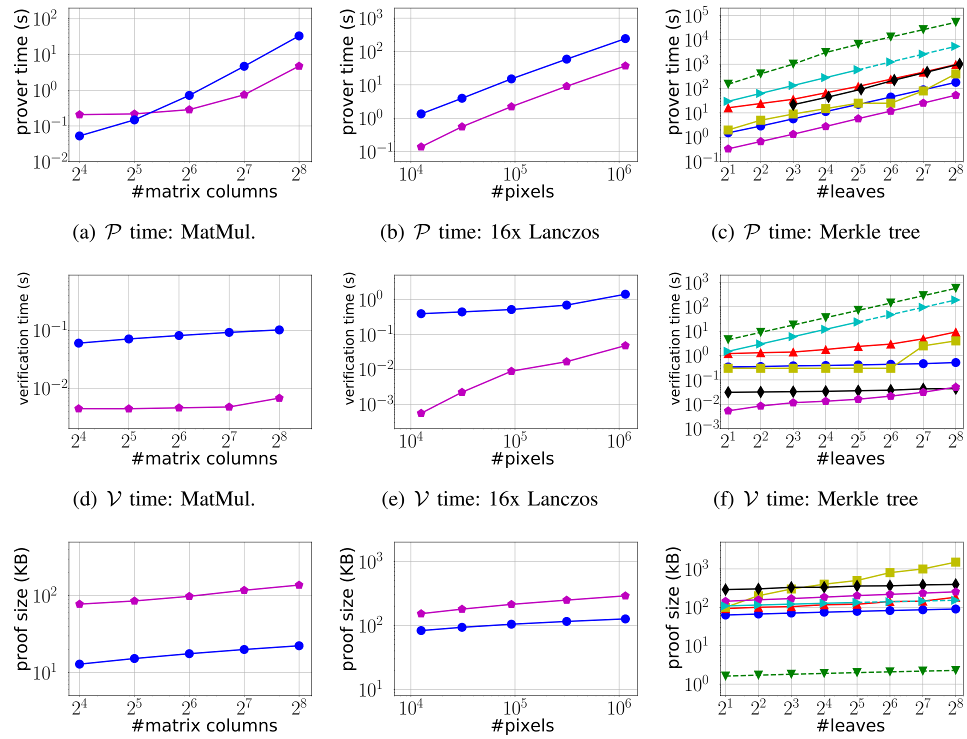
Methodology. We first compare with Libra [36], as our scheme follows the same framework and replaces the zkVPD with our new transparent one. We use the open-source implementation and the layered arithmetic circuits at [3] for all the benchmarks. The circuits are generated using [32].

We then compare the performance of Virgo to state-of-the-art transparent ZKP systems: Ligero [4], Bulletproofs [18], Hyrax [35], Stark [6] and Aurora [10]. We use the open-source implementations of Hyrax, Bulletproofs and Aurora at [1] and [2]. As the implementation of Aurora runs on $\mathbb{F}_{2^{192}}$, we execute the system on a random circuit with the same number of constraints. For Ligero, as the system is not open-source, we use the same number reported in [4] on computing hashes. For Stark, after communicating with the authors, we obtain numbers for proving the same number of hashes in the 3rd benchmark. The experiments were executed on a server



(a) $\mathcal{P}$ time (b) $\mathcal{V}$ time (c) Proof size

Fig. 2: Comparison of our zkVPD and the pairing-based zkVPD in [38].



(a) $\mathcal{P}$ time: MatMul. (b) $\mathcal{P}$ time: 16x Lanczos (c) $\mathcal{P}$ time: Merkle tree

(d) $\mathcal{V}$ time: MatMul. (e) $\mathcal{V}$ time: 16x Lanczos (f) $\mathcal{V}$ time: Merkle tree

(g) Proof size: MatMul. (h) Proof size: 16x Lanczos (i) Proof size: Merkle tree

Fig. 3: Comparisons of prover time, proof size and verification time between Virgo and existing ZKP systems.

with 512GB of DDR3 RAM (1.6GHz) and 16 cores (2 threads per core) at speed of 3.2GHz.

Benchmarks. We evaluate the systems on three benchmarks: matrix multiplication, image scaling and Merkle tree, which are used in [35], [36].

- Matrix multiplication: $\mathcal{P}$ proves to $\mathcal{V}$ that it knows two matrices whose product equals a public matrix. We evaluate on different dimensions from 4×4 to 256×256, and the size of the circuit is n^3.
- Image scaling: It computes a low-resolution image by scaling from a high-resolution image. We use the classic Lanczos re-sampling [34] method. It computes each pixel of the output as the convolution of the input with a sliding window and a kernel function defined as: $k(x) = \text{sinc}(x)/\text{sinc}(ax), \text{if} -a < x < a; k(x) = 0, \text{otherwise}$, where a is the scaling parameter and $\text{sinc}(x) = \sin(x)/x$. We evaluate by fixing the window size as 16×16 and increase the image size from 112x112 to 1072x1072.
- Merkle tree: $\mathcal{P}$ proves to $\mathcal{V}$ that it knows the value of the leaves of a Merkle tree that computes to a public root value [14]. We use SHA-256 for the hash function. We implement it with a flat circuit where each sub-computation is one instance of the hash function. The consistency of the input and output of corresponding hashes are then checked by the circuit. There are $2M - 1$ SHA256 invocations for a Merkle tree with M leaves. We increase the number of leaves from 16 to 256. The circuit size of each SHA256 is roughly 2^{19} gates and the size of the largest Merkle tree instance is around 2^{26} gates.

Comparing to Libra. Figure 3 shows the prover time, verification time and proof size of our ZKP system, Virgo, and compares it with Libra. The prover time of Virgo is 7-10$\times$ faster than Libra on the first two benchmarks, and 3-5$\times$ faster on the third benchmark. The speedup comes from our new efficient zkVPD. As shown in Section V-B, the prover time of our zkVPD is already an order of magnitude faster. Moreover, the GKR protocol for the whole arithmetic circuit must operate on the same field of the zkVPD. In Libra, it runs on a 254-bit prime field matching the base group of the bilinear map, though the GKR protocol itself is information-theoretic secure and can execute on smaller fields. This overhead is eliminated in Virgo, and both zkVPD and GKR run on our efficient extension field of Mersenne prime, resulting in an order of magnitude speedup for the whole scheme. It only takes 53.40s to generate the proof for a circuit of 2^{26} gates. Our improvement on the third benchmark is slightly less, as most input and values in the circuit are binary for SHA-256, which is more friendly to exponentiation used in Libra.

The verification time of Virgo is also significantly improved upon Libra, leading to a speedup of 10-30$\times$ in the benchmarks. This is because in Libra, the verification time of the zkVPD for the input layer is similar to that for the masking polynomials in each layer, both taking $O(\log C)$ bilinear pairings. Thus the overall verification time is roughly D times one instance of zkVPD verification. This is not the case in Virgo. We combine all the evaluations into one inner product through random linear combinations. Therefore, the verification time in Virgo is only around twice of the zkVPD verification time, ranging from 7ms to 50ms in all the benchmarks.

Because of the zkVPD, the proof size of Virgo is larger than Libra. For example, Virgo generates a proof of 253KB for Merkle tree with 256 leaves, while the proof size of Libra is only 90KB. However, the gap is not as big as the zkVPD schemes themselves in Section V-B, as the proof size of Libra is dominated by the GKR protocol of the circuit, which is actually improved by 2$\times$ in Virgo because of the smaller field. Finally, Libra requires a one-time trusted setup for the pairing-based zkVPD, while Virgo is transparent.

Comparing to other transparent ZKP Systems. Table II and Figure 3 show the comparison between Virgo and state-of-the-art transparent ZKP systems. As shown in Figure 3, Virgo is the best among all systems in terms of practical prover time, which is faster than others by at least an order of magnitude. The verification time of Virgo is also one of the best thanks to the succinctness of our scheme. It only takes 50ms to verify the proof of constructing a Merkle tree with 256 leaves, a circuit of size 2^{26} gates. The verification time is competitive to Stark, and faster than all other systems by 2 orders of magnitude. The proof size of Virgo is also competitive to other systems. It is larger than Bulletproofs [18] and is similar to Hyrax, Stark and Aurora.

In particular, our scheme builds on the univariate sumcheck proposed in [10]. Compared to the system Aurora, Virgo significantly improves the prover time due to our efficient field and the fact that the univariate sumcheck is only on the witness, but not on the whole circuit. For the computation in Figure 3, the witness size is 16$\times$ smaller than the circuit size. E.g., the witness size for one hash is around 2^{14} while the circuit size is 2^{18}. In the largest instance in the figure, the witness size is 2^{22} while the circuit size is 2^{26}. The verification time is also much faster as we reduce the complexity from linear to logarithmic. The proof size is similar to Aurora. Essentially the proof size is the same as that in Aurora on the same number of constraint as the witness size, plus the size of the GKR proofs in the zkVPD and for the whole circuit.

VI. Applications and Future Work

In this section, we discuss several applications of our new zkVPD and ZKP schemes.

A. Privacy on Blockchain

Zero knowledge proof is widely used in blockchain systems to provide privacy for cryptocurrencies (e.g., Zcash [8]), smart contracts (e.g., Hawk [26]) and zero knowledge contingent payment [19]. As mentioned in the introduction, the most commonly deployed ZKP scheme, SNARK [12], requires a trusted setup phase. A trusted party is usually absent in the setting of blockchains and an expensive "ceremony" [9] among multiple parties is usually deployed to generate the SRS. To address this issue, there are recent attempts to use transparent ZKP schemes. For example, in [16], Bünz et at. proposed Zether, which uses a variant of Bulletproofs [18] to hide account balances and provide confidentiality for applications such as auction. However, due to the high prover time and verification time of Bulletproofs for general computations, providing full anonimity still remain impractical.

As shown in Section V-C, among all transparent ZKP schemes, Virgo achieves the best prover time and one of the best verification time, which are critical for applications of ZKP on blockchains. Compared to existing GKR-based ZKP scheme, Virgo removes the trusted setup of Libra [36], and improves the verification time of both Libra and Hyrax [35] by 1-2 orders of magnitude. These make Virgo a good candidate to build privacy-preserving cryptocurrencies and smart contract without trusted setup. The overhead on the proof size is comparable to schemes based on IOPs, which is acceptable in scenarios such as permissioned blockchain and can be potentially reduced through proof composition [13].

B. Future work

1) GPU implementation: The current system we implemented can only run in single thread mode. It can be proved that it's efficiently parallelable. There are still some problems to address. Most important one is the cache efficiency for random circuit. We could address this problem by prepossessing the representation of the circuit. The other problem is huge memory consumption. Though we tried to optimize the memory overhead, and it's actually the best system now. The concrete memory consumption still large, it's larger than any customer level GPU memory.

[6]When the circuit is data parallel, the prover time of Hyrax [35] is $O(C + C' \log C')$ where C' is the size of each copy in the data parallel circuit. Hyrax has the option with proof size $O(D \log C + n^{\tau})$ and verification time $O(D \log C + n^{1-\tau})$ for $\tau \in [0, \frac{1}{2}]$.

	Ligero [4]	Bulletproofs [18]	Hyrax[6] [35]	Stark [6]	Aurora [10]	Virgo
$\mathcal{P}$ time	$O(C\log C)$	$O(C)$	$O(C\log C)$	$O(C\log^2 C)$	$O(C\log C)$	$O(C+n\log n)$
$\mathcal{V}$ time	$O(C)$	$O(C)$	$O(D\log C+\sqrt{n})$	$O(\log^2 C)$	$O(C)$	$O(D\log C+\log^2 n)$
Proof size	$O(\sqrt{C})$	$O(\log C)$	$O(D\log C+\sqrt{n})$	$O(\log^2 C)$	$O(\log^2 C)$	$O(D\log C+\log^2 n)$

TABLE II: Performance of transparent ZKP systems. C is the size of the regular circuit with depth D, and n is witness size.

Protocol 4 (Univariate Sumcheck)**.** *Let f be a degree k univariate polynomial on $\mathbb{F}$ with degree less than k and $\mathbb{H},\mathbb{L}$ be a multiplicative coset of $\mathbb{F}$ such that $\mathbb{H}\subset\mathbb{L}\subset\mathbb{F}$ and $|\mathbb{L}|>k$. To prove $\mu=\sum_{a\in\mathbb{H}}f(a)$, a univariate sumcheck protocol has the following algorithms.*

- $\mathsf{SC.com}\leftarrow\mathsf{SC.Commit}(f)$*:*
  1) *$\mathcal{P}$ computes polynomial h such that $f(x)=g(x)+Z_{\mathbb{H}}(x)\cdot h(x)$. $\mathcal{P}$ evaluates of $f|_{\mathbb{L}}$ and $h|_{\mathbb{L}}$.*
  2) *$\mathcal{P}$ commits to the vectors using Merkle tree $\mathsf{root}_f\leftarrow\mathsf{MT.Commit}(f|_{\mathbb{L}})$ and $\mathsf{root}_h\leftarrow\mathsf{MT.Commit}(h|_{\mathbb{L}})$. $\mathcal{P}$ sends $\mathcal{V}$ $\mathsf{com}=(\mathsf{root}_f,\mathsf{root}_h)$.*
- $\langle\mathsf{SC.Prove}(f),\mathsf{SC.Verify}(\mathsf{com},\mu)\rangle$*:*
  1) *Let $p(x)=\frac{|\mathbb{H}|\cdot f(x)-\mu-|\mathbb{H}|\cdot\mathbb{Z}_{\mathbb{H}}(x)h(x)}{|\mathbb{H}|\cdot x}$.*
  2) *$\mathcal{P}$ and $\mathcal{V}$ invoke the low degree test: $\langle\mathsf{LDT}.\mathcal{P}((f,h),p),\mathsf{LDT}.\mathcal{V}((k,k-|\mathbb{H}|),|\mathbb{H}|-1)\rangle(\mathbb{L})$. If the test fails, $\mathcal{V}$ aborts and output $\mathsf{0}$. Otherwise, at then end of the test, $\mathcal{V}$ needs oracle access to κ points of f,h and p in $\mathbb{L}$. We denote their indices as $\mathcal{I}$.*
  3) *For each index $i\in\mathcal{I}$, $\mathcal{P}$ opens $\mathsf{MT.Open}(i,f|_{\mathbb{L}})$ and $\mathsf{MT.Open}(i,h|_{\mathbb{L}})$.*
  4) *$\mathcal{V}$ executes $\mathsf{MT.Verify}$ for all points opened by $\mathcal{P}$. If any verification fails, abort and output $\mathsf{0}$.*
  5) *$\mathcal{V}$ completes the low degree test with these points. If all checks above pass, $\mathcal{V}$ outputs $\mathsf{1}$.*

2) Reducing the zkVPD prover complexity: In the zkVPD protocol, we need to encode RS-Code. The encoding involves FFT computation. The overall computation complexity is $O(n\log n)$. It's not an ideal solution. We would expect a efficient $O(n)$ solution in the future.

APPENDIX A
UNIVARIATE SUMCHECK PROTOCOL

The protocol of the univariate sumcheck in [10] is in Protocol 4. We have the following lemma:

Lemma 5. *Let $f:\mathbb{F}\to\mathbb{F}$ be a univariate poynomial with degree less than k and $\mathbb{H}\subseteq\mathbb{L}\subseteq\mathbb{F}$ and $|\mathbb{L}|>k$. Protocol 4 is an interactive proof to prove $\mu=\sum_{a\in\mathbb{H}}f(a)$ with soundness $O(\frac{\mathbb{L}}{\mathbb{F}}+\mathsf{negl}(\kappa))$. The proof size and the verification time are $O(\log^2|\mathbb{L}|)$ and the prover time is $O(|\mathbb{L}|\log|\mathbb{L}|)$.*

REFERENCES

[1] Hyrax reference implementation. https://github.com/hyraxZK/hyraxZK
[2] libiop. https://github.com/scipr-lab/libiop
[3] Libra implementation. https://github.com/sunblaze-ucb/fastZKP/tree/Libra
[4] Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (2017)
[5] Backes, M., Kate, A., Patra, A.: Computational verifiable secret sharing revisited. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 590–609. Springer (2011)
[6] Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint, 2018
[7] Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast reed-solomon interactive oracle proofs of proximity. In: 45th International Colloquium on Automata, Languages, and Programming (ICALP 2018). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2018)
[8] Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: Proceedings of the Symposium on Security and Privacy SP, 2014 (2014)
[9] Ben-Sasson, E., Chiesa, A., Green, M., Tromer, E., Virza, M.: Secure sampling of public parameters for succinct zero knowledge proofs. In: 2015 IEEE Symposium on Security and Privacy. pp. 287–304. IEEE (2015)
[10] Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent Succinct Arguments for R1CS. Cryptology ePrint, 2018
[11] Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Theory of Cryptography Conference. pp. 31–60. Springer (2016)
[12] Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture. In: Proceedings of the USENIX Security Symposium, 2014
[13] Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. In: CRYPTO 2014, pp. 276–294 (2014)
[14] Blum, M., Evans, W., Gemmell, P., Kannan, S., Naor, M.: Checking the correctness of memories. Algorithmica **12**(2-3), 225–244 (1994)
[15] Boneh, D., Bünz, B., Fisch, B.: Batching techniques for accumulators with applications to iops and stateless blockchains. Tech. rep., Cryptology ePrint Archive, Report 2018/1188, Tech. Rep (2018)
[16] Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. IACR Cryptology ePrint Archive **2019**, 191 (2019)
[17] Byott, N.P., Chapman, R.J.: Power sums over finite subspaces of a field. Finite Fields and Their Applications **5**(3), 254–265 (1999)
[18] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: Proceedings of the Symposium on Security and Privacy (SP), 2018. vol. 00, pp. 319–338
[19] Campanelli, M., Gennaro, R., Goldfeder, S., Nizzardo, L.: Zero-knowledge contingent payments revisited: Attacks and payments for services. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 229–243. ACM (2017)
[20] Canetti, R., Chen, Y., Holmgren, J., Lombardi, A., Rothblum, G.N., Rothblum, R.D.: Fiat-shamir from simpler assumptions. Cryptology ePrint Archive, Report 2018/1004 (2018)
[21] Chiesa, A., Forbes, M.A., Spooner, N.: A Zero Knowledge Sumcheck and its Applications. CoRR **abs/1704.02086** (2017), http://arxiv.org/abs/1704.02086
[22] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, Third Edition. The MIT Press, 3rd edn. (2009)
[23] Cormode, G., Mitzenmacher, M., Thaler, J.: Practical Verified Computation with Streaming Interactive Proofs. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. ITCS '12
[24] Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Crypto 1986
[25] Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating Computation: Interactive Proofs for Muggles. J. ACM **62**(4), 27:1–27:64 (Sep 2015)
[26] Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: Proceedings of Symposium on security and privacy (SP), 2016
[27] Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic Methods for Interactive Proof Systems. J. ACM **39**(4), 859–868 (Oct 1992)
[28] Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge snarks from linear-size universal and updateable structured reference strings. Cryptology ePrint Archive, Report 2019/099 (2019), https://eprint.iacr.org/2019/099
[29] Merkle, R.C.: A certified digital signature. In: CRYPTO 1989. pp. 218–238
[30] Papamanthou, C., Shi, E., Tamassia, R.: Signatures of correct computation. In: TCC 2013. pp. 222–242 (2013)
[31] Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: S&P 2013. pp. 238–252 (2013)
[32] Tange, O.: Gnu parallel - the command-line power tool. The USENIX Magazine **36**(1), 42–47 (Feb 2011), http://www.gnu.org/s/parallel
[33] Thaler, J.: Time-Optimal Interactive Proofs for Circuit Evaluation. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013 (2013)
[34] Turkowski, K.: Filters for common resampling tasks. In: Graphics gems. pp. 147–165. Academic Press Professional, Inc. (1990)
[35] Wahby, R.S., Tzialla, I., Shelat, A., Thaler, J., Walfish, M.: Doubly-efficient zkSNARKs without trusted setup. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 926–943. IEEE (2018)
[36] Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: Advances in Cryptology (CRYPTO) (2019)
[37] Yuan, J., Yu, S.: Proofs of retrievability with public verifiability and constant communication cost in cloud. In: Proceedings of the 2013 international workshop on Security in cloud computing. pp. 19–26. ACM (2013)
[38] Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: A Zero-Knowledge version of vSQL. Cryptology ePrint, 2017
[39] Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In: IEEE Symposium on Security and Privacy (S&P) 2017 (2017)