Ligero++: A New Optimized Sublinear IOP

Rishabh Bhadauria
Bar-Ilan University
rishabh.bhadauria@biu.ac.il

Zhiyong Fang
Texas A&M University
zhiyong.fang.1997@tamu.edu

Carmit Hazay
Ligero Inc., Bar-Ilan University
carmit@ligero-inc.com

Muthuramakrishnan Venkitasubramaniam
Ligero Inc., University of Rochester
muthu@ligero-inc.com

Tiancheng Xie
UC Berkeley
tianc.x@berkeley.edu

Yupeng Zhang
Texas A&M University
zhangyp@tamu.edu

ABSTRACT

This paper follows the line of works that design concretely efficient transparent sublinear zero-knowledge Interactive Oracle Proofs (IOP). Arguments obtained via this paradigm have the advantages of not relying on public-key cryptography, not requiring a trusted setup, and resistance to known quantum attacks. In the realm of transparent systems, Ligero and Aurora stand out with incomparable advantages where the former has a fast prover algorithm somewhat succinct proofs and the latter has somewhat fast prover and succinct proofs. In this work, we introduce Ligero++ that combines the best features of both approaches to achieve the best of both worlds. We implement our protocol and benchmark the results.

CCS CONCEPTS

• **Security and privacy → Cryptography**.

KEYWORDS

IOP; Zero-Knowledge; MPC-in-the-Head, SNARKs

ACM Reference Format:
Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. 2020. Ligero++: A New Optimized Sublinear IOP. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20), November 9–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3372297.3417893

1 INTRODUCTION

Verifying outsourced computations is important for tasks and scenarios when there is an incentive for the party performing the computation to report incorrect answers. In this work, we present a concretely efficient argument protocol for NP whose communication complexity is polylogarithmic to the size of a circuit verifying the NP witness, combining tools from [5] and [55]. Our argument system is a transparent zero-knowledge argument of knowledge that is public-coin, i.e. it only requires the verifier to send random coins to the prover in each round. The latter feature implies that it can be made non-interactive via the Fiat-Shamir transform [30], yielding an efficient implementation of zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs [24]) without a trusted setup.

Our paper follows a line of works on transparent sublinear zero-knowledge arguments from symmetric-key primitives (collision-resistant hash-functions). Building on Kilian [43] and Micali [47], recent works [11, 14, 16, 18, 20] have shown how to obtain efficient sublinear arguments for NP from so-called probabilistically checkable proofs (PCPs) [6, 7, 9]. Classical PCPs have been extended to allow additional interaction with the prover, first in the model of interactive PCP (IPCP) [41] and then in the more general setting of interactive oracle proofs (IOP) [21], also known as probabilistically checkable interactive proofs (PCIP) [49]. Arguments obtained via PCPs and IOPs have the advantage of not relying on heavy public-key cryptography or a trusted setup. Additionally, they are resistant to known quantum attacks.

With the growing list of zero-knowledge constructions, this work serves as a evidence that for real world scenarios the optimum construction will involve the right combination (i.e. composition) of known constructions. In the realm of transparent systems, Ligero [5] and Aurora [20] stand out with incomparable advantages. While Ligero brings the best prover performance concretely, Aurora's proof length scales polylogarithmically with circuit size. In this work, we combine the best features of both approaches and introduce Ligero++, an optimized sublinear IOP that achieves the best of both worlds. While there are several ways to compose systems, the main technical novelty here is a way to compose that harnesses the benefits of several previous IOP-based systems and achieve a good tradeoff between the prover complexity and proof length.

Our first instantiation combines Ligero and Aurora where we rely on the Ligero proof system to "fold" the original computation of size $|C|$ into k computations of size $\frac{|C|}{k}$ that are "uniform", which we prove using the Aurora proof system. This results in a system with prover efficiency close to the Ligero system and succinctness similar to the Aurora system.

As a second instance of this paradigm, we borrow some features from another line of work based on MPC-in-the-head, ZKBoo [32]. We demonstrate that the vanilla Ligero proof system yields good proof length for small circuits by repeating the entire proof to reduce soundness. Concretely, this yields a new instantiation of post-quantum signatures that is competetive with previous works.

CCS '20, November 9–13, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7089-9/20/11…$15.00
https://doi.org/10.1145/3372297.3417893

We demonstrate our performance via an implementation and believe that our composition techniques could potentially lead to more systems with better concrete and asymptotic performance.

1.1 Our Techniques

We first recall the Ligero proof system. This is an instantiation of the MPC-in-the-head paradigm of Ishai et al. [40] where the underlying MPC is instantiated with an optimized variant of the protocol of Damgård and Ishai [29]. A key feature of the underlying MPC protocol is that its *total* communication complexity between the parties is independent of the number of parties and is roughly equal to the size of the circuit being evaluated. Now, letting the number of parties be the square root of the circuit size, results in communication per party that is also roughly the square root of the circuit size. This translates into a ZKIPCP with analogous parameters and highly competitive prover's complexity and concrete running time.

In the first step in Ligero, the prover computes an "extended" witness that incorporates all intermediate computations (namely, output of each "gate") which it encodes using the Interleaved Reed-Solomon code. Interpreting each element of the interleaved code as a column, the entire code can be viewed as a matrix, say U. The prover computes the proof oracle where each element is one "column" of this code. Next, the verifier provides randomness and obtains a randomized aggregated summary of the "rows" of the code from the prover which it checks by fruther querying a random subset of columns in this oracle and matching the aggregates across the columns against the aggregated summary. Soundness follows from the "error correction" property of the Reed-Solomon code while zero-knowledge follows from the "secret-sharing" properties of the underlying code (i.e. interpreting the encoding as a Shamir secret-sharing scheme).

In more detail, the column consistency check in the Ligero proof system requires the verifier to compute a linear function (more precisely, an inner product) between each queried column, and a challenge "random" vector r chosen by the verifier. The number of queries directly effects the soundness level while the dimensions of U heavily influence the proof size and the parties' complexities. By setting the parameters of the Interleaved Reed-Solomon code appropriately, it is possible to have the size of the summarized row and the queried columns be roughly equal to $\sqrt{|C|}$ and the number of queries $O(\lambda)$.

Our first observation is that by choosing a bigger alphabet size of the interleaved code, we can make the row size any size γ where the alphabet size is $|C|/\gamma$. This will yield a proof size of $O(|C|/\gamma)$ if γ is small. In order to reduce the overall proof length, we compose the Ligero proof system with a second zero-knowledge proof system to establish the column consistency check, namely the Aurora proof system. More precisely, we rely on the inner product argument (IPA) recently proposed by Zhang et al. in [55] that in turn relies on the Aurora proof system.[1] Furthermore, we combine all of the column checks into one check by folding it via another random linear combination.

Our main observation is that the code rate in Aurora is much smaller than in Ligero since Aurora employs a low degree test (where the prover time is proportional to the inverse of the code rate). Through our composition, we manage to reduce the inner product argument in Aurora to a small instance (a subset of columns), thus improving the proof size significantly without compromising the prover time by too much.

Performance. Our scheme aims to provide a good trade-off between the two techniques, achieving the best of both systems. The prover time compared to Ligero is only slower by 2x, yet the proof size is reduced to polylogarithmic and is only between 100-200KBs. The proof size is already better than Ligero for circuits with hundreds of gates, and is improved by 32x on large circuits with millions of gates. Compared to Aurora, the prover time is improved by 11-12x, and the proof size is surprisingly even smaller.

Concretely, on a circuit with 2^{22} multiplication gates, the proof size of our new scheme is 184KB, smaller than the 276KB proof of Aurora and much smaller than the 6MB proof of Ligero. Meanwhile, it only takes 80s to generate the proof in our scheme, which is 2.3× slower than the prover time of 34s in Ligero, and much faster than that of 1004s in Aurora.

In our second instantiation, our signature scheme is competitive compared to existing candidates for post quantum signatures. It takes 42ms to sign a message and 8ms to validate the signature using MiMC block ciphers, and the signature size is 210KB. The size of our signature starts to be smaller than the Picnic signature schemes when using block ciphers with larger circuits. Our signature size is estimated to be 111KB using the standard AES block cipher. We refer to Section 4 for more details on the performance and the comparisons.

Applications. Our basic benchmark (of validating Merkle decommitments) already demonstrates our performance compared to Ligero and Aurora systems (See Section 4). We have further estimated the performance of our scheme on the application of linear regression, where the verification time grows sublinearly with the size of the instance. On a large instance with 2^{28} gates, our prover time is around 12,000s, the proof size is 257KB and the verification time is around 1s. We also report the performance of Ligero++ for DNA profile matching and batch verification of ECDSA signatures in the appendix.

1.2 Related Work

Zero-knowledge proofs were introduced by Goldwasser et al. in [36] and generic constructions based on PCPs were proposed in the seminal work of Kilian [43] and Micali [47] in the early days. In recent years, zero-knowledge proofs have evolved from purely theoretical constructions to practical implementations. Other than the schemes based on IOP [5, 14, 20], another line of work is based on the interactive proofs proposed by Goldwasser, Kalai, and Rothblum [34] (following a rich line of works on interactive proofs with computationally unbounded provers [35, 45, 50]), usually referred as the GKR protocol. The GKR protocol provides sublinear communication and efficiently verifiable proofs for low-depth polynomial-time computations. It has been extended to the case of NP statements by Zhang et al. [56] using polynomial commitment schemes, and several GKR-based zero-knowledge argument schemes are proposed in subsequent works [52, 54, 57]. The most relevant work in this line is a recent scheme by Zhang et al. [55], where a new polynomial

[1] We note that the underlying machinery of this IPA scheme is similar to Aurora while making use of the important FRI low degree test protocol [12].

commitment scheme based on symmetric cryptographic primitives without trusted setup is proposed. The zero knowledge argument scheme in [55] falls into the model of IOP where their underlying implementation relies on the FRI protocol of [20]. The communication complexity of all GKR-based schemes grows linearly with the depth of the circuit and they only support layered circuits.

Based on the MPC-in-the-head approach, ZKBoo [32] introduce linear proof size in the verification circuit. Followup works [27, 42] optimize the communication cost for post-quantum signature schemes, where the latter uses preprocessing-based MPC. Following the work of [38, 39, 44], Gennaro et al. [31] introduced Quadratic Arithmetic Programs (QAPs) which form the basis of a sequence of protocols [19, 22, 23, 28, 48, 51]. While the proof size of these protocols is constant and the verification time depends on input size, the prover's running time is much higher and a separate setup phase is required. Another line of works is based on the hardness of discrete log [10, 25, 26, 37]. Among these, Bulletproof [26] generates a logarithmic proof size but has high prover and verifier times due to a number of cryptographic operations per gate.

Our zero-knowledge proof scheme proposed in this paper is based on IOP. We mainly compared the properties with other IOP-based schemes. Please refer to [52, 54, 55] for more details on the performance and comparisons of other ZKP schemes.

2 PRELIMINARIES

We use $\mathsf{negl}(\cdot)$ to denote the negligible function, where for each positive polynomial $f(\cdot)$, $\mathsf{negl}(k) < \frac{1}{f(k)}$ for sufficiently large integer k. λ denotes the security parameter and "PPT" standards for probabilistic polynomial time. We use lower-case letters such as a, b, x, y to represent vectors, and x_i denotes the i-th element in vector x. We use capital letters such as A, B to represent matrices. A_i denotes the i-th row, $A[j]$ denotes the j-th column and A_{ij} denotes the (i, j)-th element of matrix A.

Merkle hash tree. Merkle hash tree is a primitive proposed by Ralph Merkle [46] to commit a vector and open it at an index with a logarithmic-size proof. It consists of three algorithms:

- $\mathsf{root}_c \leftarrow \mathsf{MT.Commit}(c)$
- $(c_i, \pi_i) \leftarrow \mathsf{MT.Open}(i, c)$
- $(1, 0) \leftarrow \mathsf{MT.Verify}(\mathsf{root}_c, i, c_i, \pi_i)$

2.1 Coding Notations

For a code $C \subseteq \Sigma^n$ and vector $v \in \Sigma^n$, denote by $d(v, C)$ the minimal distance of v from C, namely the number of positions in which v differs from the closest codeword in C, and by $\Delta(v, C)$ the set of positions in which v differs from such a closest codeword (in case of ties, take the lexicographically first closest codeword), and by $\Delta(V, C) = \bigcup_{v \in V} \{\Delta(v, C)\}$. We further denote by $d(V, C)$ the minimal distance between a vector set V and a code C, namely $d(V, C) = \min_{v \in V} \{d(v, C)\}$.

Reed-Solomon code. For a linear code $\mathsf{C} \subseteq \Sigma^n$ and vector $v \in \Sigma^n$, we use $d(\mathsf{C}, v)$ to denote the minimal distance of v from C. Formally $d(\mathsf{C}, v) = \min_{c \in \mathsf{C}} h(c, v)$, where $h(c, v)$ is the hamming distance between c and v.

Definition 2.1 (Reed-Solomon Code). For positive integers n, k, field $\mathbb{F}$ and vector $\eta = (\eta_0, ..., \eta_{n-1}) \in \mathbb{F}^n$ of distinct field elements, the Reed-Solomon (RS) code $RS_{\mathbb{F},n,k,\eta}$ is the $[n, k, n-k+1]$ linear code over $\mathbb{F}$ that consists of all n-tuples $(p(\eta_0), p(\eta_1), ..., p(\eta_{n-1}))$ where p is a polynomial of degree $< k$ over $\mathbb{F}$.

Definition 2.2 (Encoded message). Let $L = RS_{\mathbb{F},n,k,\eta}$ be an RS code and $\zeta = (\zeta_1, ..., \zeta_k)$ be a sequence of distinct elements in $\mathbb{F}$. For a codeword $u \in L$, we define the message $Dec_\zeta(u)$ to be $(p_u(\zeta_1), ..., p_u(\zeta_k))$, where p_u is the polynomial (of degree $< k$) corresponding to u. For $U \in L^m$ with rows $u_1, ..., u_m \in L$, we let $Dec_\zeta(U)$ be the length-mk vector $x = (x_{11}, ..., x_{1k}, ..., x_{m1}, ..., x_{mk})$ such that $(x_{i1}, ..., x_{ik}) = Dec_\zeta(u^i), i \in [m]$. Finally we say that U encodes x if $x = Dec_\zeta(U)$, we use $Dec(U)$ when ζ is clear from the context.

In our protocol, we set $\eta_i = \omega^i$ where ω is a generator of a multiplicative group in field $\mathbb{F}$. We can evaluate $(p(\eta_0), p(\eta_1), ..., p(\eta_{n-1}))$ using the fast Fourier transform (FFT), which takes $O(n \log n)$ field operations. We use $\mathsf{RS}(a)$ to denote the RS encoding of message a.

2.2 Zero-knowledge Arguments

A zero-knowledge 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 $(x; w) \in \mathcal{R}$ for some input x, and learns nothing beyond that. 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 $\mathcal{R}$ is known to $\mathcal{P}$ and $\mathcal{V}$.

Definition 2.3. Let $\mathcal{R}$ be an NP relation. A tuple of algorithm $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is an argument of knowledge for $\mathcal{R}$ if the following holds.

- **Correctness**. For every pp output by $\mathcal{G}(1^\lambda)$ and $(x, w) \in R$,
$$\langle \mathcal{P}(\mathsf{pp}, w), \mathcal{V}(\mathsf{pp}) \rangle(x) = 1$$
- **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 $\mathsf{negl}(\lambda)$:
$$\Pr[\langle \mathcal{P}(w), \mathcal{V} \rangle(\mathsf{pp}, x) = 1 \wedge (x, w) \notin \mathcal{R} | w \leftarrow \varepsilon^{\mathcal{P}}(\mathsf{pp}, x)]$$

It is a zero-knowledge argument of knowledge it additionally satisfies:

- **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
$$\mathsf{View}(\langle \mathcal{P}(w), \mathcal{V}^* \rangle(\mathsf{pp}, x, z)) \approx \mathcal{S}^{\mathcal{V}^*}(\mathsf{pp}, x, z)$$

Here $\varepsilon^{\mathcal{P}}$ denotes that the extractor ε has access to the entire executing process and the randomness of $\mathcal{P}$. $\mathcal{S}^{\mathcal{V}^*}$ denotes that the simulator $\mathcal{S}$ sees the randomness from a polynomial-size space of $\mathcal{V}^*$. We say that $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a **succinct** argument system if the total communication between $\mathcal{P}$ and $\mathcal{V}$ (proof size) are $\mathsf{poly}(\lambda, |x|, \log |w|)$.

2.3 Interactive Oracle Proofs

Interactive Oracle Proofs (IOP) [21, 49] is a type of proof system that combines the aspects of Interactive Proofs (IP) [8, 35] along with Probabilistic Checkable Proofs (PCP) [6, 7, 9] as well generalizes

Interactive PCPs (IPCP) [41]. In this model, like the PCP model, the verifier does not need to read the whole proof and instead can query the proof at some random locations while similar to IP model, the prover and verifier interact over several rounds.

A k-round IOP has k rounds of interaction. In the i^{th} round of interaction, the verifier sends a uniform public message m_i to the prover and the prover generates π_i. After running k rounds of interaction, the verifier makes some queries to the proofs via oracle access and will either accept it or reject it.

Definition 2.4. Let $\mathcal{R}(x, \omega)$ be an NP relation corresponding to an NP language $\mathcal{L}$. An IOP system for a relation $\mathcal{R}$ with round complexity k and soundness ϵ is a pair of PPT algorithms $(\mathcal{P}, \mathcal{V})$ if it satisfies the following properties:

- *Syntax:* On common input x and prover input ω, $\mathcal{P}$ and $\mathcal{V}$ run an interactive protocol of k rounds. In each round i, $\mathcal{V}$ sends a message m_i and $\mathcal{P}$ generates π_i. Here the $\mathcal{V}$ has oracle access to $\{\pi_1, \pi_2, \ldots, \pi_k\}$. We can express $\pi = (\pi_1, \pi_2, \ldots, \pi_k)$. Based on the queries from these oracles, $\mathcal{V}$ accepts or rejects.
- *Completeness:* If $(x, \omega) \in \mathcal{R}$ then,

$$\Pr[(\mathcal{P}(x, \omega), \mathcal{V}^{\pi}(x)) = 1] = 1$$

- *Soundness:* For every $x \notin \mathcal{L}$, every unbounded algorithm $\mathcal{P}^*$ and proof $\tilde{\pi}$

$$\Pr[(\mathcal{P}^*, \mathcal{V}^{\tilde{\pi}}) = 1] \leq \mathsf{negl}(\lambda)$$

The notion of IOP can be extended to provide zero-knowledge property as well. Next we define the definition of zero-knowledge IOP.

Definition 2.5. Let $\langle \mathcal{P}, \mathcal{V} \rangle$ be an IOP for $\mathcal{R}$. We say that $\langle \mathcal{P}, \mathcal{V} \rangle$ is a (honest verifier) zero-knowledge IOP if there exists a PPT simulator $\mathcal{S}$, such that for any $(x, \omega) \in \mathcal{R}$, the output of $\mathcal{S}(x)$ is distributed identically to the view of $\mathcal{V}$ in the interaction $(\mathcal{P}(x, \omega), \mathcal{V}(x))$.

2.4 Inner Product Arguments

Inner product arguments (IPA) allow a verifier to validate the inner product of a committed vector from the prover and a public vector. Our protocols use the inner product arguments recently proposed by Zhang et al. in [55] as a building block. The scheme is a Reed-Solomon encoded interactive oracle proof based on the work of Aurora[20], and does not require a trusted setup. Let $y = \langle a, b \rangle$ be the inner product of two vectors. The scheme consists of the following algorithms:

- $\mathsf{pp} \leftarrow \mathsf{IPA.KeyGen}(1^\lambda)$,
- $\mathsf{com}_a \leftarrow \mathsf{IPA.Commit}(a, \mathsf{pp})$,
- $(y, \pi) \leftarrow \mathsf{IPA.Prove}(a, b, \mathsf{pp})$,
- $\{0, 1\} \leftarrow \mathsf{IPA.Verify}(\mathsf{pp}, y, \mathsf{com}_a, b, \pi)$

Theorem 2.6 ([55]). *There exists an inner product argument scheme satisfying the following properties:*

- ***Completeness.*** *For any private vector $a \in \mathbb{F}^n$, public vector $b \in \mathbb{F}^n$, $\mathsf{pp} \leftarrow \mathsf{IPA.KeyGen}(1^\lambda)$, $\mathsf{com} \leftarrow \mathsf{IPA.Commit}(a, \mathsf{pp})$, $\{y, \pi\} \leftarrow \mathsf{IPA.Prove}(a, b, \mathsf{pp})$, it holds that*

$$\Pr\left[\mathsf{IPA.Verify}(\mathsf{pp}, y, \mathsf{com}_a, b, \pi) = 1\right] = 1$$

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

$$\Pr\left[\begin{matrix}(a^*, \mathsf{com}^*, b, y^*, \pi^*) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp}) \\ \mathsf{IPA.Verify}(\mathsf{pp}, y^*, \mathsf{com}^*, b, \pi^*) = 1\end{matrix} : \begin{matrix}\mathsf{com}^* = \mathsf{IPA.Commit}(a^*, \mathsf{pp}) \\ \wedge \langle a^*, b \rangle \neq y^*\end{matrix}\right]$$

Complexity. *Let the size of the vectors be n. The running time of Commit and* Prove *is $O(n \log n)$ time for the prover, and the running time of* Verify *is $O(n)$ for the verifier. The proof size is $O(\log^2 n)$.*

In particular, in the commitment of the IPA in [55], a is encoded into $\mathsf{RS}(a)$, and committed by setting $\mathsf{com}_a = \mathsf{MT.Commit}(\mathsf{RS}(a))$.

2.5 Ligero

Our construction follows the general structure of Ligero [5], and we highlight the idea of the construction here. Ligero is a zero knowledge argument protocol for NP based on the MPC in the head method. It allows the prover to convince the verifier that a statement is valid in zero knowledge.

We use arithmetic circuits to model our computation as done in Ligero. We use prime fields in our construction. Our technique is explained here for arithmetic circuits but can be extended to boolean circuits as well.

At a very high level, the Ligero prover arranges the circuit wire values in an matrix. It then encodes each row using the Reed-Solomon code. The verifier challenges the prover to reveal the linear combinations of the entries of the matrix, and randomly checks t entries of the revealed combination, by asking the prover for the corresponding columns. Therefore the total communication is one row and t columns. By setting the dimensions of the matrix to be a $\sqrt{C} \times \sqrt{C}$ matrix, the overall communication is $O(t\sqrt{C})$. Moreover, the prover needs to encode each row, thus the overall encode time is $O(\sqrt{C} \times \sqrt{C} \log C) = O(C \log C)$. One simple way to reduce this complexity is by reducing the row size. Nevertheless, this will increase the proof size since the prover needs to transfer t columns. It is a major challenge in our paper to reduce both the computation time and the communication size.

3 OUR CONSTRUCTION

In this section, we present our main construction, a zero-knowledge argument protocol for NP relations expressed as R1CS. Similar to [5], the construction consists of three components: testing interleaved linear code, testing linear constraints and testing quadratic constraints. We present these components in Section 3.1-3.3, followed by the whole protocol in Section 3.4. Finally, we show how to turn the protocol into zero-knowledge in Section 3.5.

3.1 Testing Interleaved Linear Code

We start with our protocol for testing interleaved linear code. The purpose of this protocol is to check that each row of a matrix U constructed by the prover is an RS code. Formally speaking, let $L \subset \mathbb{F}^n$ be an $[n, k, d]$ linear code over $\mathbb{F}$, the $[n, mk, d]$ interleaved code L^m over $\mathbb{F}^m$ is the code whose codewords are all $m \times n$ matrices U such that every row U_i of U satisfies $U_i \in L$.

Following the idea in [5], the protocol to test whether the constructed matrix U is an interleaved linear code contains two checks. In the first check, the verifier asks the prover to take a random linear

combination of the rows in matrix U, and sends the combined codeword $w = r^T U$ to the verifier, where r is a random vector chosen by the verifier. Due to the property of linear codes, the verifier verifies that w is a valid RS code (by decoding w for example.) In the second check, the verifier aims to ensure that the combined codeword w sent by the prover is actually consistent with the matrix U. This is done by randomly opening some columns of U. In particular, the verifier picks $Q \subseteq [n]$, a set of random indices. The prover sends all columns of $U[j]$ for $j \in Q$ with their consistency proofs to the commitment of U. The verifier checks that $\forall j \in Q$, $w_j = \langle r, U[j] \rangle$.

Intuitively, the reason the protocol works is that if the matrix U is far from an interleaved code, then if w sent by the prover in the first check is indeed rU, it is not a valid RS code with high probability; otherwise if w is not $r^T U$, then with high probability some w_j will be inconsistent with $\langle r, U[j] \rangle$. Regarding the proof size, the communication in the first check is linear to the row size, and the communication in the second check is linear to the size of columns in U. Therefore, the communication of the interleaved test in [5] is optimal when the sizes of the rows and columns are set to be roughly the same.

Improving proof size. We observe that the second check is actually computing multiple inner products between vectors from the prover and a public vector selected by the verifier. (I.e., for all $j \in Q$, $w_j = \langle r, U[j] \rangle$.) Then, instead of sending these vectors directly as in [5], the prover and the verifier can invoke inner product arguments to perform the second check. Using the inner product argument scheme described in Section 2.4 reduces the communication of the second check to be logarithmic in the size of the columns in U. The check is utilized in Step 5 in Protocol 1.

Looking ahead, when using our new interleaved linear code test, the witness of our zero-knowledge argument will be encoded into an interleaved code such that the size of the matrix is asymptotically the same as the size of the circuit/R1CS, denoted as C. Therefore, we will encode the witness into a matrix of size $\frac{C}{\text{polylog}(C)} \times \text{polylog}(C)$. In this way, the communication of both checks are $\text{polylog}(C)$ and the proof size of the zero-knowledge argument is $\text{polylog}(C)$, instead of $O(\sqrt{C})$ in [5]. Our full protocol for testing interleaved linear code is presented in Protocol 1.

We next prove the following theorem:

Theorem 3.1. *Protocol 1 is an interleaved linear code testing with the following properties:*

- ***Completeness.*** *For any valid interleaved Reed-Solomon code $U \in L^m$, the verifier accepts U with probability one.*
- ***Soundness.*** *([5, 55]) Let $\delta = \text{negl}(\lambda)$ be the soundness error of the IPA protocol and let e be a positive integer such that $e < d/3$. Suppose $d(U^*, L^m) > e$. Then, for any malicious $\mathcal{P}^*$ strategy, the oracle U^* is rejected by $\mathcal{V}$ except with probability $(1 - e/n)^t + d/|\mathbb{F}| + \delta$.*

Proof. We will prove both properties.

Completeness. First, the completeness of the interleaved test directly follows from the properties of linear codes and correctness of inner product argument.

Soundness. Let $\mathcal{P}^*$ be a malicious strategy for the prover. Towards analyzing the soundness error, we will formally define the following event Honest defined as follows. This event holds if the prover can provide randomness that "proves" it performed Step 5 correctly. Meaning, assume that the transcript trans is already fixed right before this step, then there exists randomness r for which when running $\mathcal{P}^*(\text{trans}; r)$ then $\mathcal{P}^*$ follows the honest strategy for the prover $\mathcal{P}$ as defined in Protocol 1. We split the analysis into two cases.

- **Case I**. In this case we condition on the event Honest. Let $e < d/3$ and $d(U^*, L^m) > e$. Then we will consider two subcases here.
  - $d(w^*, L) = e' > e$**.** Note that in this case, $\mathcal{V}$ will only accept the interleaved test if it queries all the t values are from the remaining $n - e'$ positions. The probability of that is $\frac{\binom{n-e'}{t}}{\binom{n}{t}}$. This can be generalized to get the probability,
  $$\Pr[\mathcal{V} \text{ accepts} \mid d(w^*, L) > e] \leq \frac{\binom{n-e-1}{t}}{\binom{n}{t}}.$$
  - $d(w^*, L) < e$**.** This case is covered by the following lemma proven in [5].

  Lemma 3.2 ([5]). *Let L be a Reed-Solomon code with minimal distance $d = n - k + 1$ and e be a positive integer such that $e < d/3$. Suppose $d(U, L^m) > e$. Then, for a random w^* in the row-span of U, we have*
  $$\Pr[d(w^*, L) \leq e] \leq d/|\mathbb{F}|.$$

  To conclude, using a union bound argument, we can bound the probability of this case by,
  $$\begin{aligned} \Pr[\mathcal{V} \text{ accepts } U^*] &\leq \Pr[\mathcal{V} \text{ accepts } U^* \mid d(w^*, L) > e]] \\ &\quad + \Pr[d(w^*, L) \leq e] \\ &\leq \frac{\binom{n-e-1}{t}}{\binom{n}{t}} + d/|\mathbb{F}| \\ &\leq (1 - e/n)^t + d/|\mathbb{F}|. \end{aligned}$$
- **Case II**. In this case we will condition on the event $\overline{\text{Honest}}$ where $\mathcal{V}$ accepts with probability bounded by δ due to the soundness error induced from IPA protocol.

Combining both cases using a union bound argument, we conclude with the following soundness error,

$$(1 - e/n)^t + d/|\mathbb{F}| + \delta.$$

□

Complexity. Let C be the size of matrix U. Then the prover time is $O(C \log C)$, the communication size is $O(\text{polylog} C)$ and the verifier time is $O(C)$.

3.2 Testing Linear Constraints

In this section we present our protocol for testing a linear constraint on an interleaved linear code. In particular, the protocol checks if an encoded message x satisfies $Ax = 0$ where $A \in \mathbb{F}^{m\ell \times m\ell}$ is a public sparse matrix with $O(m\ell)$ non-zero elements.

The main idea of our protocol is similar to that of our interleaved test. Following the construction in [5], the verifier asks the prover to randomly combine the rows of U and tests the linear constraint on the combined codeword; the verifier then queries a random subset of columns to check the consistency between the combined

Protocol 1 (Interleaved linear code test). *$\mathbb{F}$ is a prime field and $L \subset \mathbb{F}^n$ is a $[n, k, d]$ RS code. Let $U \in \mathbb{F}^{m \times n}$ be the matrix to be tested.*

- $\mathsf{pp} \leftarrow \mathsf{KeyGen}(1^\lambda)$.
- ***Interleaved testing:***

(1) $\mathcal{V}$ generates a random vector $r \in \mathbb{F}^m$ and sends it to $\mathcal{P}$.
(2) $\mathcal{P}$ computes $w = r^T U \in \mathbb{F}^n$ and sends it to $\mathcal{V}$.
(3) $\mathcal{V}$ checks that $w \in L$.
(4) $\mathcal{V}$ generates a random set $Q \subseteq [n]$ and $|Q| = t$ and sends it to $\mathcal{P}$.
(5) $\mathcal{V}$ checks the consistency of w. In particular, for $j \in Q$, $\mathcal{P}$ and $\mathcal{V}$ invoke an IPA protocol on $U[j]$ and r. $\mathcal{V}$ accepts if all the checks pass, and rejects otherwise.

codeword and U. Using inner product arguments, we again avoid sending the columns of U directly and improve both the proof size and the prover time.

In particular, the verifier first sends a random vector $r \in \mathbb{F}^{m\ell}$ to the prover so that the linear constraint becomes $(r^T A)x = 0$. (If $Ax \neq 0$, then $(r^T A)x = 0$ with probability $\frac{1}{\mathbb{F}}$.) Let $a = r^T A$, then the prover parses a into a matrix of size $m \times \ell$ and interpolates each row of the matrix into a polynomial $a_i(\cdot)$. The prover also interpolates each row of matrix U into polynomial $p_i(\cdot)$. In this way, the inner product of a and x is computed by $\sum_{j=1}^{\ell} \sum_{i=1}^{m} a_i(\zeta_j) p_i(\zeta_j)$.

The prover sends the polynomial $q(\cdot) = \sum_{i=1}^{m} a_i(\cdot) p_i(\cdot)$ in Step 3 so that the verifier checks whether the linear constraint is satisfied. This is done by checking that $\sum_{j=1}^{\ell} q(\zeta_j) = 0$ i.e. checking if summation at all interpolation point equates to 0 (see Step 4). The verifier also checks consistency between $q(\cdot)$ and U by randomly picking a subset Q of columns and checking that $q(\eta_j) = \sum_{i=1}^{m} a_i(\eta_j) U_{ij}$, which are the inner products between the columns of U and the public vectors (see Step 6).

The full protocol is presented in Protocol 2.

Theorem 3.3. *Protocol 2 is a linear constraint testing protocol with the following properties:*

- ***Completeness.*** *For any valid x, such that $Ax = 0$, and it's Reed-Solomon code $U \in L^m$, the verifier accepts with probability 1.*
- ***Soundness.*** *([5, 55]) Let e be a positive integer such that $e < d/2$, where d is the minimal distance of the RS code. Suppose that a (badly formed) matrix U^* is e-close to a codeword $U \in L^m$ that encodes $x \in F^{mk}$ such that $Ax \neq 0$. Then, for any malicious $\mathcal{P}^*$ strategy, U^* is rejected by $\mathcal{V}$ except with probability at most $((e + k + \ell)/n)^t + 1/|\mathbb{F}| + \delta$.*

The proof follows similarly to the proof outline from the previous section and the following lemma.

Lemma 3.4 ([5]). *Let e be a positive integer such that $e < d/2$. Suppose that a (badly formed) oracle U^* is e-close to a codeword $U \in L^m$ encoding $x \in \mathbb{F}^{m\ell}$ such that $Ax \neq b$. Then, for any malicious $\mathcal{P}^*$ strategy, U^* is rejected by $\mathcal{V}$ except with at most $1/\mathbb{F} + ((e+k+\ell)/n)^t$ probability.*

Complexity. Let C be the size of matrix U and assuming a can be computed in linear time. Then the prover time is $O(C \log C)$, the communication size is $O(\text{polylog} C)$ and the verifier time is $O(C)$.

3.3 Testing Quadratic Constraints

In this section we describe a test for verifying that messages x, y, z encoded by U_x, U_y, U_z satisfy the constraint $x \odot y - z = 0$, where $\odot$ denotes point-wise product. The constraint reduces to checking that $U^x \odot U^y - U^z$. We present this test in a self-contained way in Protocol 3. The protocol again utilizes the inner product arguments and the complexity is the same as the linear constraint test.

The main idea of this protocol is similar to that of our interleaved test. Following the constructions in [5], the verifier asks the prover to randomly combine the rows of $U^x \odot U^y - U^z$ and tests the linear constraint on the combined codeword; the verifier then queries a random subset of columns to check the consistency between the combine codeword and $U^x \odot U^y - U^z$. Using inner product arguments, we again avoid sending the columns of U directly and improve both the proof size and the prover time.

In particular, the verifier first sends a random vector $r \in \mathbb{F}^m$ to the prover. The prover interpolates each row of matrix U^x, U^y and U^z into polynomial $p_i^x(\cdot)$, $p_i^y(\cdot)$ and $p_i^z(\cdot)$ respectively . In this way, the inner product of r and $U[j]$ is $\sum_{j=1}^{\ell} \sum_{i=1}^{m} r_i \cdot p_i(\zeta_j)$ where $p_i(\cdot) = p_i^x(\cdot) p_i^y(\cdot) - p_i^z(\cdot)$.

The prover sends a polynomial $q(\cdot) = \sum_{i=1}^{m} r_i \cdot p_i(\cdot)$, where $p_i(\cdot) = p_i^x(\cdot) p_i^y(\cdot) - p_i^z(\cdot)$ in Step 2 and the verifier checks whether the linear constraint is satisfied. This is done by check if for all $j \in [\ell]$, $q(\zeta_j) = 0$ i.e. checking the polynomial evaluates to 0 at the interpolation point (in Step 3). The verifier also checks consistency of $q(\cdot)$ by randomly picking a subset Q of columns and check $q(\eta_j) = \sum_{i=1}^{m} r_i \cdot U_{i,j}$, where $Ui, j = U^x i, j * U^y i, j - U^z i, j$ which are inner products between columns of U ($U = U^x \odot U^y - U^z$) and public vectors r (in Step 5).

The full protocol is presented in Protocol 3.

Theorem 3.5. *Protocol 3 is a quadratic constraint testing protocol with the following properties:*

- ***Completeness.*** *For any valid x, y, z, such that $x \odot y - z = 0$, and it's Reed-Solomon code $U^x, U^y, U^z \in L^m$, the verifier accepts with probability one.*
- ***Soundness.*** *([5, 55]) Let e be a positive integer such that $e < d/2$. Let U^{x*}, U^{y*}, U^{z*} be badly formed matrix and let $U^* \in F^{3m \times n}$ be the matrix obtained by vertically juxtaposing the corresponding $m \times n$ matrices. Suppose $d(U^*, L^{3m}) \leq e$, and let U^x, U^y, U^z, respectively, be the (unique) codewords in L^m that are closest to U^{x*}, U^{y*}, U^{z*}. Suppose U^x, U^y, U^z encode x, y, z such that $x \odot y - z \neq 0$. Then, for any malicious $\mathcal{P}^*$ strategy, (U^{x*}, U^{y*}, U^{z*}) is rejected by $\mathcal{V}$ except with at most $1/|\mathbb{F}| + (e + 2k)^t + \delta$ probability.*

The proof follows similarly to the proof outline from the previous section and the following lemma.

Lemma 3.6. *([5]) Let e be a positive integer such that $e < d/2$. Let U^{x*}, U^{y*}, U^{z*} be badly formed oracles and let $U^* \in \mathbb{F}^{3m \times n}$ be the matrix obtained by vertically juxtaposing the corresponding $m \times n$*

Protocol 2 (Testing linear constraints over Interleaved RS Codes). *Let $L[n,k,d]$ be an RS code and $U \in L^m$ be an interleaved code that encodes the message x. $A \in \mathbb{F}^{m\ell \times m\ell}$ is a public matrix such that $Ax = 0$.*

- *Run* $\mathsf{pp} \leftarrow \mathsf{IPA.KeyGen}(1^\lambda)$.
- ***Interleaved testing:***

(1) *$\mathcal{V}$ picks a random value $r \in \mathbb{F}^{m\ell}$ and sends r to $\mathcal{P}$.*
(2) *Both $\mathcal{P}$ and $\mathcal{V}$ computes $a \leftarrow r \times A$ and calculates polynomials $a_i(\cdot)$ such that $a_i(\zeta_j) = a_{i\ell+j-1}$ for all $i \in [m], j \in [\ell]$.*
(3) *$\mathcal{P}$ computes polynomials $p_i(\cdot)$ such that $p_i(\eta_j) = U_{ij}$ for $i \in [m], j \in [n]$. $\mathcal{P}$ constructs polynomial $q(x) = \sum_{i=1}^{m} a_i(x) \cdot p_i(x)$ and sends it to $\mathcal{V}$.*
(4) *$\mathcal{V}$ checks that $\sum_{j\in[\ell]} q(\zeta_j) = 0$.*
(5) *$\mathcal{V}$ generates a random set $Q \subseteq [n]$ and $|Q| = t$ and sends it to $\mathcal{P}$.*
(6) *Let b_j denote the vector $(a_0(\eta_j), \ldots, a_{m-1}(\eta_j))$. $\mathcal{V}$ checks the consistency for $q(\cdot)$. In particular, for $j \in Q$, $\mathcal{P}$ and $\mathcal{V}$ invoke an IPA protocol on $U[j]$ and b_j. $\mathcal{V}$ accepts if all the checks pass, and rejects otherwise.*

Protocol 3 (Testing quadratic constraints over Interleaved RS Codes). *Let λ be the security parameter, $\mathbb{F}$ be a prime field. $L[n,k,d]$ be the intended codeword space. $U^x \in L^m$ encodes the message x, $U^y \in L^m$ encodes the message y, $U_z \in L^m$ encodes the message z. t be the repeat parameter depend on λ. x, y and z satisfies $x \odot y - z = 0$. The testing will accept if U^x, U^y, U^z correctly encodes x, y, z, let η be a root of unity of order n. Let $p_i^x(\cdot)$ be the corresponding polynomial of U_i^x, and we define $p_i^y(\cdot)$ and $p_i^z(\cdot)$ similarly.*

- *Run* $\mathsf{pp} \leftarrow \mathsf{IPA.KeyGen}(1^\lambda)$.
- ***Interleaved testing:***

(1) *$\mathcal{V}$ picks a random value $r \in \mathbb{F}^m$ and sends r to $\mathcal{P}$.*
(2) *$\mathcal{P}$ construct polynomial $q(\cdot)$ defined by $q(\cdot) = \sum_{i=1}^{m} r_i \cdot p_i(\cdot)$, where $p_i(\cdot) = p_i^x(\cdot)p_i^y(\cdot) - p_i^z(\cdot)$ send the polynomial q to the verifier.*
(3) *$\mathcal{V}$ checks that $\forall i \in [\ell]$, $q(\zeta_j) = 0$.*
(4) *$\mathcal{V}$ generates a random set $Q \subseteq [n]$ and $|Q| = t$ and sends it to $\mathcal{P}$.*
(5) *$\mathcal{V}$ checks the consistency for $q(\cdot)$. In particular, for $j \in Q$, $\mathcal{P}$ and $\mathcal{V}$ invoke an IPA protocol on $U[j]$ and r where $U[j] = U^x[j] * U^y[j] - U^z[j]$. $\mathcal{V}$ accepts if all the checks pass, and rejects otherwise.*

matrices. Suppose $d(U^*, L^{3m}) \le e$, and let U^x, U^y, U^z, respectively, be the (unique) codewords in L^m that are closest to U^{x*}, U^{y*}, U^{z*}. Suppose U^x, U^y, U^z encode x, y, z such that $x \odot y + a \odot z \neq b$. Then, for any malicious $\mathcal{P}$ strategy, (U^{x*}, U^{y*}, U^{z*}) is rejected by $\mathcal{V}$ except with at most $1/|\mathbb{F}| + ((e+2k)/n)^t$ probability.

3.4 Putting Everything Together

We combine all the components and present our new zero knowledge argument protocol here. Let $C : \mathbb{F}^{|C|} \to \mathbb{F}$ be an arithmetic circuit. Without loss of generality we can assume that the circuit only contains addition and multiplication gates with fan-in two. The full protocol is described in Protocol 5.

Regarding the oracles generation, we generate them for our IOP protocol in Protocol 4 in a similar fashion as in [5]. Recall that the prover rearranges the witness w into a matrix and encode each row using Reed-Solomon codes, and then generates the matrix U. Originally, in [5], the prover opens a subset of columns for executing a consistency check. In our new protocol, this consistency check is replaced by an IPA protocol. The IPA protocol here checks that $\langle a, b\rangle = c$ where a is a secret vector, b is a public vector and c is a public value. This protocol extends the vector a into a Reed-Solomon code and queries some of these elements. To model this in our IOP model, we extend the matrix U into an oracle $\mathcal{U}$ by converting each column into a Reed-Solomon code as well, where this oracle $\mathcal{U}$ is now queried in the IPA protocol during our tests. Note that whenever we run an IPA protocol on a column $U[j]$ and a vector r to check whether $\langle U[j], r\rangle = c$, the IPA in essence proves that $\mathcal{U}[j]$ represents a vector v such that $\langle v, r\rangle = c$. Therefore a consistency proof between U and $\mathcal{U}$ is not required.

We have the following theorem:

Theorem 3.7. *Let δ be the soundness of underlying IPA protocol, e be any positive integer such that $e < d/3$ and suppose there exists no input* inp *such that $C(\mathsf{inp}) = 1$, then for every (unbounded) prover strategy $\mathcal{P}^*$, $\mathcal{V}$ accepts with probability at most $(d+2)/|\mathbb{F}| + (1 - e/n)^t + 2(e+2k)^t + 3\delta$.*

Proof. We argue that soundness by combining the soundness errors in the following cases.

- **Case** $d(U, L^{4m}) > e$**:** Since $e < d/3$, we can conclude from Soundness property of Theorem 3.1 that verifier accepts with probability :$(1 - e/n)^t + d/|\mathbb{F}| + \delta$
- **Case** $d(U, L^{4m}) \le e$**:** Let $U^w, U^x, U^y, U^z \in L^m$ be codes that are codes that are close to $U^{w*}, U^{*x}, U^{y*}, U^{z*}$, and encode w,x,y,z. As we know there is no inp such that $C(\mathsf{inp}) = 1$, then w,x,y,z cannot satisfy all the linear and quadratic constructs namely :
  - $x = P_x w$, $y = P_y w$, $z = P_z w$
  - $P_a dd w = 0^{m\ell}$
  - $x \odot y - z = 0$

  We rely on the soundness analysis of Theorems 3.3 and 3.5 to bound the tests above using a union bound over a linear test and a quadratic test.

$$((e+k+\ell)/n)^t + 1/|\mathbb{F}| + \delta + ((e+2k)/n)^t + 1/|\mathbb{F}| + \delta < 2 \cdot \left(1/|\mathbb{F}| + ((e+2k)/n)^t + \delta\right).$$

Combining the two cases together using union bound gives us the soundness :

$$(d+2)/|\mathbb{F}| + (1 - e/n)^t + 2(e+2k)^t + 3\delta$$

□

Protocol 4 (Oracle Generation). *Let $\mathbb{F}$ is a prime field, let RS be Reed-Solomon code and w be the vector to be encoded.*

- $\mathsf{pp} \leftarrow \mathsf{KeyGen}(1^\lambda)$.
- *$\mathcal{P}$ divides the vector w into matrix W of size $m \times \ell$ where $m \cdot \ell > |w|$.*
- *$\mathcal{P}$ encodes each row a in W as $\mathsf{RS}(a)$ where $\mathsf{RS}(a)$ is the reed solomon encoding of message a. This is done by using interpolation set $\mathcal{I}_1$ where $|\mathcal{I}_1| = \ell$ to generate a polynomial of size k ($k > \ell$). Then we evaluate this polynomial on an evaluation set $\mathcal{E}_1$ where $|\mathcal{E}_1| = n$ ($n > k$). We call this matrix U^w which represents the encoding of vector w.*
- *$\mathcal{P}$ repeats the encoding on the columns of matrix U. Basically $\mathcal{P}$ encodes each column a in U as $\mathsf{RS}(a)$ where $\mathsf{RS}(a)$ is the reed solomon encoding of message a. This is done by using interpolation set $\mathcal{I}_2$ where $|\mathcal{I}_2| = m$ to generate a polynomial of size ℓ. Then we evaluate this polynomial on an evaluation set $\mathcal{E}_2$ where $|\mathcal{E}_2| = c$ ($c > m$). We call this matrix $\mathcal{U}^w$ and this represents the oracle of vector w.*

Protocol 5 (Our new argument construction for arithmetic circuits.). *Let $w = \{\alpha_1, \alpha_2, ..., \alpha_{|C|}\}$ be the circuit wire value (witness).*

(1) $\mathsf{pp} \leftarrow \mathsf{IPA.KeyGen}(1^\lambda)$ *generate public parameters for* IPA.

(2) Convert the input circuit to the Ligero matrix:

(a) Let m, ℓ be the parameter used in previous protocols. ($|C| = m \times \ell$).

(b) We define a system of constraints that contains the following constraints:

(i) Multiplication:

$$\alpha_a \times \alpha_b - \alpha_c = 0$$

(ii) Addition:

$$\alpha_a + \alpha_b - \alpha_c = 0$$

where a, b, c are the wire id of the circuit.

(c) $\mathcal{P}$ constructs x, y, z used in Protocol 3 in the following way:

(i) Let $\alpha_c = \alpha_a \times \alpha_b$ be the i-th multiplication gate, $\mathcal{P}$ sets $x[j] = \alpha_a$, $y[j] = \alpha_b$, $z[j] = \alpha_c$.

(d) $\mathcal{P}, \mathcal{V}$ construct the matrices $P_x, P_y, P_z \in \mathbb{F}^{m\ell \times m\ell}$ such that:

$$x = P_x w,\ y = P_y w,\ z = P_z w$$

(e) $\mathcal{P}, \mathcal{V}$ construct the matrix $P_{add} \in \mathbb{F}^{mk \times mk}$ to handle addition in the following way:

(i) For j-th addition gate $\alpha_a + \alpha_b - \alpha_c = 0$, then set $P_{add}[j][a] = P_{add}[j][b] = 1$ and $P_{add}[j][c] = -1$.

(ii) Set the other positions to zero.

(3) Encode and generate the oracle:

(a) Encode vectors w, x, y, z into encoded matrix $U^w, U^x, U^y, U^z \in \mathbb{F}^{m \times n}$ and generate oracles $\mathcal{U}^w, \mathcal{U}^x, \mathcal{U}^y, \mathcal{U}^z \in \mathbb{F}^{c \times n}$ by using Protocol 4. We would like to state that the oracles $\mathcal{U}^w, \mathcal{U}^x, \mathcal{U}^y, \mathcal{U}^z$ would only be invoked within the IPA protocols.

(4) Test U^w, U^x, U^y, U^z is e-close to a valid interleaved code by engaging Protocol 1 on the following input:

$$U^w, U^x, U^y, U^z$$

(5) Test addition gate, engage Protocol 2 on the following input:

$$(P_{add}, U^w)$$

(6) Test multiplication gate by using Protocol 2 and Protocol 3 in the following way:

(a) Let $[I_{m\ell} | -P_x]$ be a horizontally concatenated matrix of size $m \times (2\ell)$, and $\begin{Bmatrix} U^x \\ U^w \end{Bmatrix}$ be a vertically concatenated matrix of size $2m \times n$.

(b) We define $[I_{m\ell} | -P_y]$, $[I_{m\ell} | -P_z]$, $\begin{Bmatrix} U^y \\ U^w \end{Bmatrix}, \begin{Bmatrix} U^z \\ U^w \end{Bmatrix}$ in a similar way.

(c) run Protocol 2 on inputs:

$$([I_{m\ell} | -P_x], \begin{Bmatrix} U^x \\ U^w \end{Bmatrix}), ([I_{m\ell} | -P_y], \begin{Bmatrix} U^y \\ U^w \end{Bmatrix}), ([I_{m\ell} | -P_z], \begin{Bmatrix} U^z \\ U^w \end{Bmatrix})$$

(d) this will check $x = P_x w$, $y = P_y w$, $z = P_z w$.

(e) Run Protocol 3 on input: (U^x, U^y, U^z) in Protocol 3.

(f) this will check $z = x \odot y$

Complexity. Overall, the prover's complexity is $O(C \log C)$ field operations, the proof size is $O(\text{polylog}C)$ field elements and the verifier time is $O(C)$ field operations.

Extension to boolean circuits. We can extend our IOP argument to a boolean circuit in a similar fashion as done in [5]. In essence, we can use linear test to check all gate constraints (XOR and AND) [5]. This in turn increases the size of extended witness where instead of 1 element, two elements are used for each gate. Again we also would like to check that each element in the extended witness is a boolean value i.e., either 0 or 1. This check can be represented by introducing a quadratic constraint of the form $\beta^2 = \beta$ and can therefore be checked using the quadratic test.

3.5 Achieving Zero-Knowledge

Finally, in this section, we present the modifications on the protocols to achieve zero knowledge. We use standard techniques from existing works [5, 20, 55]. In particular, there are two parts of the

protocols that leak information about the witness. Taking the interleaved test in Protocol 1 as an example, in the first check in Step 2, $\mathcal{V}$ receives a linear combination of all rows in U, which encodes the witness in the full protocol. In addition, in the second check in Step 5, the verifier validates the consistency of the linear combination in the first check with a subset of columns in U.

To mitigate the first leakage, the prover appends a random codeword in L as an additional row to U. Now the linear combination in the first check is still a codeword, but leaks no information about the witness. The dimension of the interleaved code increases by 1 in this approach, i.e., $U \in L^{m+1}$. To mitigate the second leakage, we increase the degree of the RS code L by t, such that opening any t points of the codeword does not leak any information of the message. Note that if we do so, the inner product argument scheme does not have to be zero knowledge, as even revealing all the vectors leaks no information about the witness encoded in U. Alternatively, we can keep the same degree of L, but applies a zero knowledge inner product arguments.

Below we present the modifications required for the modules.

(1) Modifications to Protocol 1.
   (a) Modify the original input U to U', where U' contains an additional row of random codeword u'. $U' = \begin{Bmatrix} U \\ u' \end{Bmatrix}$.
   (b) The new random combined codeword in Protocol 1 Step 2 becomes $w' = w + u'$.
(2) Modifications to Protocol 2. The verification algorithm for the linear constraint $Ax = b$ samples a random vector r, obtains rAx and compares it with rb. Protocol 2 Step 6 will test $\sum_{i\in[n]} q(\zeta^i) = 0$. The individual evaluation of $q(\cdot)$ will reveal random linear combination about the input.
   Polynomial $q(\cdot)$ is essentially a random linear combination of rows of U. We can mask it by adding a random row u' that encodes a message $(\sigma_1, ..., \sigma_\ell)$ such that $\sum_{i=1}^{\ell} \sigma_i = 0$, and modify A such that it adds an additional constraint to ensure $\sum_{i=1}^{\ell} \sigma_i = 0$.
(3) Modifications to Protocol 3. We modify Protocol 3 in the same way as for Protocol 1, however, matrix U is a virtual matrix composed by U_x, U_y, U_z. Instead of adding a row to U, we add 3 random rows $u_{x'}, u_{y'}, u_{z'}$ to U_x, U_y, U_z, where $u_{x'}, u_{y'}, u_{z'}$ encode 3 random message x', y', z' such that it holds that $x' \odot y' - z' = 0$.

To obtain the final zero-knowledge version, we can replace the three modules of Protocol 5 with their corresponding zero-knowledge versions. We present these modified modules in Protocol 6, Protocol 7, Protocol 8 and highlight the differences compared with the non-ZK versions using a purple font.

3.6 From ZKIOP to ZK

Transforming ZKIOP into ZK can be done using the standard technique based on Merkle hash tree. In this approach, the prover commits to each element in the oracle using a statistically hiding commitment scheme and then compresses the commitment using a Merkle hash tree. Both of these commitments can be instantiated by any family of collision-resistant hash functions. Note that we cannot commit column wise as in Ligero, since the IPA protocol requires opening a small subset of elements within a column.

3.7 Optimizations

In Section 3.5, we discussed the leakage concerns in our IOP protocol and how to convert it into a zero-knowledge protocol. Taking interleaved test in Protocol 1 as an example, we discussed how the second check (Step 5) leaks information about the columns of the encoded witness. This is tackled by increasing the degree of the RS code L by t, such that opening any t points of the codeword does not leak any information of the message. Alternatively, we can use a zero-knowledge version of the IPA protocol (zk-IPA). This ensures no information is revealed about the columns and in turn the witness. This enables us to reduce the rate of codeword in Ligero framework to 2 (as we need the codeword to be at least as big as the polynomial degree in quadratic test which is has the highest degree among the three tests). This approach allows us to run the IPA protocol on all columns . As we are opening all the columns, each test (Interleaved test, linear test and quadratic test) would have a soundness error of $1/|\mathbb{F}| + \delta$ reducing the overall soundness error to $3/|\mathbb{F}| + 3\delta$. This gives the flexibility of better proof sizes in case that the number of columns is less than t (where t is linear to security parameter). As our number of columns is (polylog$C$), most of the benchmark circuits will have optimum column size that is less than t, enabling us to reduce the proof size.

4 IMPLEMENTATION AND EVALUATIONS

We fully implement our new zero knowledge proof scheme, Ligero++, and present the experimental results of the system in this section.

Software and hardware. The system is implemented in C++. There are around 1,600 lines of code for the main protocol, including the protocols for interleaved tests, linear tests, quadratic tests, Merkle Trees and the inner product arguments. We use the implementation of the inner product argument in [55]. We use the compiler of libsnark [1] to write the statements of zero-knowledge proofs. The statements are actually represented as the Rank-1-Constraint-System (R1CS). The number of constraints in R1CS maps to the number of multiplication gates in arithmetic circuits. Our protocols support both representations and in this section, we state the size of the statements in the number of R1CS constraints.

We ran all experiments on an AMD Ryzen™ 3800X processor with 64GM RAM. The experiments were not parallelized and only utilized a single CPU core. We report the average running time over 10 executions.

Choice of parameters. The rate of the RS code for the inner product argument is set to 8, and the repetition parameter of the low degree test is set to 43. These parameters provide a security level of 100+ bits based on Theorem 1.2 and Conjecture 1.5 in [12], and Conjecture 7.3 in [15], which is consistent with Virgo [55] and Aurora[20]. Without the conjectures, based on the proven theorem in [15, Theorem 7.2], we would need to repeat the inner product argument protocol twice to achieve the same security level of 100+ bits. This would increase the prover time, proof size and the verification time of the whole Ligero++ protocol by less than 2×, and the scheme still yields good trade-off on the prover time and the proof size compared to Ligero and Aurora. For our main protocol, we set the repetition parameter $t = 300$ and the rate of the RS code as 4. Our field is the same as the one used in Virgo[55], which is $\mathbb{F}_{p^2}$ where $p = 2^{61} - 1$.

Protocol 6 (Zero knowledge interleaved linear code test). *$\mathbb{F}$ is a prime field and $L \subset \mathbb{F}^n$ is a $[n, k, d]$ RS code. Let $U \in \mathbb{F}^{m \times n}$ be the matrix to test.*

- pp $\leftarrow$ KeyGen(1^λ)
- ***Interleaved testing:***

(1) *$\mathcal{P}$ Appends a random codeword $U_{u'}$ to U.*
(2) *$\mathcal{P}$ computes $w = r^T U \in \mathbb{F}^n$ and sends it to $\mathcal{V}$.*
(3) *$\mathcal{V}$ checks that $w \in L$.*
(4) *$\mathcal{V}$ generates a random set $Q \subseteq [n]$ and $|Q| = t$ and sends it to $\mathcal{P}$.*
(5) *$\mathcal{V}$ checks the consistency of w. In particular, for $j \in Q$, $\mathcal{P}$ and $\mathcal{V}$ invoke an IPA protocol on $U[j]$ and r. $\mathcal{V}$ accepts if all the checks pass, and rejects otherwise.*

Protocol 7 (Zero knowledge testing linear constraints over Interleaved RS Codes). *Let λ be the security parameter, $\mathbb{F}$ be a prime field. $L[n, k, d]$ be the intended codeword space. U encodes the message x. t be the repeat parameter depend on λ. m defined in Definition 2.2, and A, b satisfies $Ax = b$. The testing will return 1 if U correctly encodes x such that $Ax = b$.*

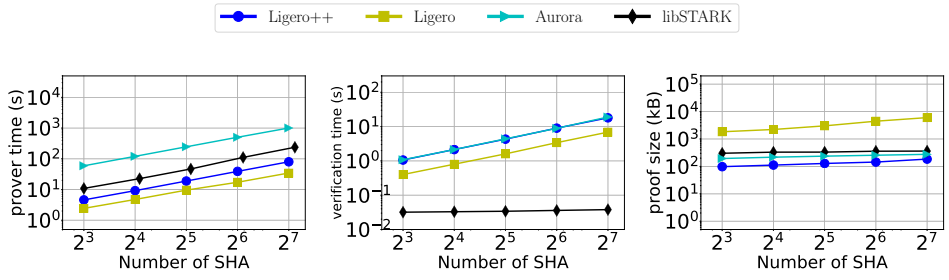
- pp $\leftarrow$ IPA.KeyGen(1^λ)
- ***Interleaved testing:***

(1) *$\mathcal{P}$ generates a random message $u' \in \mathbb{F}^\ell$ such that $\sum_{i=1}^{\ell} u'[\ell] = 0$, encodes into codeword $U_{u'}$ and appends it to U.*
(2) *Both sides modify the matrix A, adding the constraint $\sum_{i=1}^{\ell} u'[\ell] = 0$.*
(3) *$\mathcal{V}$ picks a random value $r \in \mathbb{F}^{m\ell}$ and sends r to $\mathcal{P}$.*
(4) *Both $\mathcal{P}$ and $\mathcal{V}$ computes $a \leftarrow r \times A$ and calculates polynomials $a_i(\cdot)$ such that $a_i(\zeta_j) = a_{i\ell+j-1}$ for all $i \in [m+1], j \in [\ell]$.*
(5) *$\mathcal{P}$ computes polynomials $p_i(\cdot)$ such that $p_i(\eta_j) = U_{ij}$ for $i \in [m+1], j \in [n]$. $\mathcal{P}$ constructs polynomial $q(x) = \sum_{i=1}^{m+1} a_i(x) \cdot p_i(x)$ and sends it to $\mathcal{V}$.*
(6) *$\mathcal{V}$ checks that $\sum_{j \in [\ell]} q(\zeta_j) = 0$.*
(7) *$\mathcal{V}$ generates a random set $Q \subseteq [n]$ and $|Q| = t$ and sends it to $\mathcal{P}$.*
(8) *Let b_j denote the vector $(a_0(\eta_j), \ldots, a_m(\eta_j))$. $\mathcal{V}$ checks the consistency for $q(\cdot)$. In particular, for $j \in Q$, $\mathcal{P}$ and $\mathcal{V}$ invoke an IPA protocol on $U[j]$ and b_j. $\mathcal{V}$ accepts if all the checks pass, and rejects otherwise.*

Protocol 8 (Zero knowledge testing quadratic constraints). *Let λ be the security parameter, $\mathbb{F}$ be a prime field. $L[n, k, d]$ be the intended codeword space. $U^x \in L^m$ encodes the message x, $U^y \in L^m$ encodes the message y, $U_z \in L^m$ encodes the message z. t be the repeat parameter depend on λ. x, y and z satisfies $x \odot y - z = 0$. The testing will accept if U^x, U^y, U^z correctly encodes x, y, z, let η be a root of unity of order n. Let $p_i^x(\cdot)$ be the corresponding polynomial of U_i^x, and we define $p_i^y(\cdot), p_i^z(\cdot)$ similarly.*

- pp $\leftarrow$ IPA.KeyGen(1^λ):
- ***Interleaved testing:***

(1) *$\mathcal{P}$ randomly samples vectors $x', y', z' \in \mathbb{F}^\ell$ such that $x' \odot y' - z' = 0$, and then encodes them and appends to U_x, U_y, U_z.*
(2) *$\mathcal{V}$ picks a random value $r \in \mathbb{F}^m$ and sends r to $\mathcal{P}$.*
(3) *$\mathcal{P}$ construct polynomial $q(\cdot)$ defined by $q(\cdot) = \sum_{i=1}^{m+1} r_i \cdot p_i(\cdot)$, where $p_i(\cdot) = p_i^x(\cdot)p_i^y(\cdot) - p_i^z(\cdot)$ send the polynomial q to the verifier.*
(4) *$\mathcal{V}$ checks that $\forall i \in [\ell], q(\zeta_j) = 0$.*
(5) *$\mathcal{V}$ generates a random set $Q \subseteq [n]$ and $|Q| = t$ and sends it to $\mathcal{P}$.*
(6) *$\mathcal{V}$ checks the consistency for $q(\cdot)$. In particular, for $j \in Q$, $\mathcal{P}$ and $\mathcal{V}$ invoke an IPA protocol on $U[j]$ and r where $U[j] = U^x[j] * U^y[j] - U^z[j]$. $\mathcal{V}$ accepts if all the checks pass, and rejects otherwise.*



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

Figure 1: Comparisons of prover time, proof size and verification time.

4.1 Performance of Ligero++

We first present the performance of Ligero++, and compare it with other IOP-based general-purpose zero knowledge proof systems, including Ligero [5], Aurora [20], Stark [14] and Virgo [55].

Benchmark. Our benchmark in this section is the path validations of a Merkle tree. The prover proves to the verifier that she knows a leaf and its validation path in a Merkle tree. This function is widely used in practice to prove memberships in zero knowledge (e.g., in the protocol of Zcash [17]). We use SHA-256 for the hash function and each hash takes around 27,000 constraints to implement in the R1CS. The R1CS are generated by the compiler provided by libsnark [23]. We increase the length of the path from 8 to 128 to show the scalability of the systems. In our largest instance, there are around $3,600,000 \simeq 2^{21.7}$ constraints.

Methodology. As Ligero is not open-source, we implement the scheme ourselves with same parameters specified above. We use the open-source implementation of Aurora in libiop [2]. We report the performance of the system on R1CS with the same number of constraints for each data point. We run the Aurora on 8-64 hashes on our 64GiB RAM machine, and interpolate the 128 hashes case, the interpolation may be larger than actual data since Aurora's implementation take advantage of large memory. For Stark, we obtain numbers for proving the same number of hashes from the authors. The experiments are executed on a server with 512GB of DDR3 RAM (1.6GHz) and 16 cores (2 threads per core) at speed of 3.2GHz. We obtained the implementation of Virgo from the authors.

The performance of Ligero++and the comparisons to existing zero knowledge proof systems are presented in Figure 1. As shown in the figure, the prover time of our system Ligero++is one of the fastest among all IOP-based ZKP schemes. It only takes 82s to generate the proof for a Merkle tree path with 128 hashes. The prover time of Ligero++is $11-12\times$ faster than Aurora, and $2-3\times$ faster than Stark on all data points we tested. Our prover time is only $2\times$ slower than Ligero, as we need to perform additional inner product arguments in our protocols.

The proof size of our scheme is the smallest among all schemes. In particular, the proof size for a Merkle path with 128 hashes is 184KB. This is slightly smaller than the 276KB in Aurora. This is because our protocol uses the inner product argument on a smaller instance, and the inner product argument is based on the same underlying techniques as Aurora (low degree test and univariate sumcheck). Meanwhile, the proof size of Ligero is 6MB, significantly larger than ours. Therefore, the experiments demonstrate that our new scheme provides a good trade-off between the proof size and the prover time. The proof size is a little smaller than Aurora, while the prover time is close to that of Ligero, achieving the best of both schemes.

Finally, the verifier time of Ligero++, Ligero and Aurora all scales linearly with the size of the circuit. It takes 18s in Ligero++for 128 hashes, which is the same as Aurora and $3\times$ slower than Ligero. Stark operates on a different computational model and the verification time is sublinear to the size of the computation. Therefore, the verification time of Stark is only tens of milliseconds.

Comparing to Virgo. Virgo operates on a different representation of layered arithmetic circuits. Following the framework of [52, 54, 56, 57], it combines the doubly efficient interactive proof by [33] (GKR protocol) with a polynomial commitment scheme to obtain a zero knowledge argument. Because of the fast prover time and the sublinear verification time for structured circuit of the GKR protocol, we anticipate that as a whole system, Virgo's prover time and verification time are faster on the benchmark, and its proof size is similar.

	Size of Polynomial	Prover Time	Verfication Time	Proof Size
Virgo	2^{12}	0.04s	0.006s	86KB
Ours		0.02s	0.0012s	49.1KB
Virgo	2^{16}	0.64s	0.007s	118.2KB
Ours		0.31s	0.0025s	90.2KB
Virgo	2^{20}	11.7s	0.012s	169.7KB
Ours		5.5s	0.0043s	180.0KB
Virgo	2^{22}	39.66s	0.020s	199.2KB
Ours		23.0s	0.0054s	229.4KB

Table 1: Performance of the polynomial commitment scheme.

However, we can actually use our protocols to construct a polynomial commitment scheme, and combine it with the GKR protocol in the same way as Virgo. This yields a new zero knowledge argument scheme similar to Virgo on layered arithmetic circuits. Table 1 shows the performance of our polynomial commitment comparing to the one used in Virgo. As shown in the table, the prover time is improved by 1.7-2.1×, while the verification time is slower by 2-3.6×. The proof size of our polynomial commitment is smaller than Virgo when the size of the polynomial is small, and slightly larger than Virgo when the size of the polynomial is large. This is because we optimize the parameters to achieve the best proof size in our scheme. When the size of the polynomial is small, the length of the rows of the matrix in our protocols is set to be relatively large. Thus the inner product arguments are executed on small columns and the proof size of Ligero++ is only around 1/2 of Virgo. When the size of the polynomial is large, the lengths of the rows stops growing, and the proof of the inner product arguments on columns becomes dominating. Therefore, the proof size becomes slightly larger than Virgo. Overall, we believe this provides a better comparison to schemes based on the GKR protocol, and our new zero knowledge argument scheme for layered arithmetic circuits will have faster prover time with similar proof size and verification time.

4.2 Post-Quantum Signature Schemes

An important application of our new ZKP scheme is building post-quantum signatures. Recent advances in quantum computers introduce potential threats to traditional digital signature schemes and NIST has initiated a process to solicit post-quantum signatures. One of the approaches to construct a post-quantum scheme is to utilize zero knowledge proofs that are plausibly post-quantum secure. Such a scheme, named Picnic, has advanced to the second round and becomes one of the nine candidates for standardization.

As described in [27, 42] the idea of the signature scheme is as follows. The private key is randomly select from $\mathsf{sk} \in \{0,1\}^\lambda$, and the public key is $\mathsf{pk} = \mathrm{PRF}_{\mathsf{sk}}(0^\lambda)$, where PRF is a pseudo-random function instantiated by a block cipher. The signature of a message m is $\mathrm{PRF}_{\mathsf{sk}}(m)$, together with the zero knowledge proof that the

	Sign	Verify	Signature size
Picnic1 [27]	25ms	17ms	118.5KB
Picnic2 [42]	28ms	28ms	45.9KB
Ours (LowMC)			202KB (est.)
Ours (MiMC)	42ms	8ms	210KB
Picnic1-AES	0.10s	0.07s	469KB
Picnic2-AES	0.11s	0.11s	182KB
Ours	0.256s	0.056s	224 KB (est.)

Table 2: Post-Quantum Signature

signer knows the private key sk that generates both $\mathrm{PRF}_{\mathsf{sk}}(m)$ and pk. The verification of the signature is simply the verification of the zero knowledge proof. The zero knowledge proof schemes used in [27, 42] are based on the technique of MPC-in-the-head. The prover time, proof size and the verification time are all linear to the size of the statement.

Methodology. In this section, we instantiate the post-quantum signature scheme described above for a slight variant of the Ligero zero-knowledge system. The main difference in our estimated in this section over [5] is in computing the optimum parameters. We consider an additional dimension of optimization where the entire proof is repeated for reducing soundness. This is inspired from the works of [27, 32, 42]. As the scheme works on R1CS, we use the MiMC block cipher [3] as the pseudo-random function. We implement the signature scheme on a 256-bit prime field, providing the same security level[2] as the Picnic schemes in [27, 42]. The performance of the Picnic schemes is obtained from [42, Table 3]. Note that their ZKP schemes work on Boolean circuits and they use the LowMC block cipher [4] as the PRF.

Table 2 displays the performance of our signature schemes and compares it to the Picnic schemes. As shown in the table, the signing time of our scheme is 42ms, which is 1.7× slower than Picnic1 [27] and 1.5× slower than Picnic2 [42]. The verifying time is much $2-3\times$ faster. Finally, the size of our signature is 210KB.

To further demonstrate the AES signature scheme, we increase the size of the statement for the ZKP scheme by using a standard block cipher, AES. AES requires roughly 1400 multiplication gates over $GF(2^8)$. We can embed this in the Ligero proof system by considering an instantiation over $GF(2^{32})$. This yields a proof length of 224KB.

Overall, the experiments demonstrate that the performance of the post-quantum signature scheme built using our new ZKP scheme is competitive to existing candidates. The advantage of our scheme on the signature size is more significant when signing multiple messages or using block ciphers with larger circuits.

4.3 Applications for SIMD Computations

Our scheme has a linear verification time to the number of R1CS constraints. However, for data parallel computations, or Single-Instruction-Multiple-Data (SIMD) computations, our scheme can actually achieve sublinear verification time. We demonstrate this feature in this section with three additional applications: linear regression, DNA profile matching and batch verification of ECDSA signatures.

[2]We instantiate the "interactive" version of our scheme with statistical security of 512-bits. Applying Fiat-Shamir reduces the security to 256-bit computational security.

n	Prover Time(s)	Verfication Time(s)	Proof Size(KB)
20	12	0.7	120
200	113	0.84	163
2,000	1,064	0.97	210
20,000	12,388*	1.1*	257*

Table 3: Performance of Ligero++ for linear regressions. $d = 100$. * **denotes estimation due to the memory limitation.**

Verifying linear regression models. Linear regression is a popular machine learning algorithm that captures the linear relationship between the data and the predictions. The n training data samples with d features each are represented as a $n \times d$ matrix X, with the corresponding $n \times 1$ label Y. Linear regression is trying to find a $d \times 1$ model w such that $Xw \approx Y$. "$\approx$" is formally defined as minimizing the distance function $\frac{1}{n} \min_w \sum_{i=1}^{n} (X_i \cdot w - Y_i)^2$, where X_i and Y_i are the i-th row of X and Y and $\cdot$ denotes vector inner products.

The closed-form solution to the optimization problem is $w = (X^T X)^{-1} X^T Y$. Therefore, to verify whether a linear regression model is correctly computed, we can validate the following equation: $(X^T X)w = X^T Y$.

Table 3 shows the performance of our system on proving the correctness of linear regression models. We fix the number of features d as 100, and vary the number of data samples n. The number of constraints for verifying linear regression is $2nd^2$. As shown in the table, the prover time for $n = 2,000$ data samples with $d = 100$ features is 12,388s. For the same size of the input, the DIZK paper [53] generates proof in 30,000s with 64 machines.

Most importantly, the verification time grows sublinearly with the size of the computation. It only takes 1.1s to verify the proof for $n = 20,000$, a computation with 2^{28} constraints. This is because when the computation is SIMD, in our scheme described in Protocol 5, the same matrix P_x, P_y, P_z generated from a single copy of the computation is used multiple times in the SIMD computation. Thus the verifier's work in the interleaved test, linear test and quadratic test only depends on the size of one copy of the computation, and the overall complexity of the verifier is $O(C' \log C' + B)$, where n' is the size of one copy and B is the number of copies. Similarly, the proof size can also be improved to $O(\log^2 C' + B)$, and is only 257KB for $n = 20,000$. These features are inherited from the protocol of Ligero from [5].

In addition, we also estimate the performance of Ligero++ for the applications of DNA profile matching described in [14] and batched verification of ECDSA signatures. We show the performance in Tables 4 and 5 in Appendix A.

ACKNOWLEDGMENTS

This material is based upon work supported by DARPA under Contract No. HR001120C0087 and No. N66001-15-C-4066. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA. This project was conducted when the third and fourth authors were at Ligero Inc.

REFERENCES

[1] 2014. libsnark. https://github.com/scipr-lab/libsnark. (2014).
[2] 2019. libiop. https://github.com/scipr-lab/libiop. (2019).
[3] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. 2016. MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity. In *ASIACRYPT*. 191–219.
[4] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. 2015. Ciphers for MPC and FHE. In *EUROCRYPT*. 430–454.
[5] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. 2017. Ligero: Lightweight Sublinear Arguments Without a Trusted Setup. In *CCS*. 2087–2104.
[6] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. 1998. Proof Verification and the Hardness of Approximation Problems. *J. ACM* 45, 3 (1998), 501–555.
[7] Sanjeev Arora and Shmuel Safra. 1998. Probabilistic Checking of Proofs: A New Characterization of NP. *J. ACM* 45, 1 (1998), 70–122.
[8] László Babai. 1985. Trading Group Theory for Randomness. In *STOC*. 421–429.
[9] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. 1991. Checking Computations in Polylogarithmic Time. In *STOC*. 21–31.
[10] Stephanie Bayer and Jens Groth. 2012. Efficient Zero-Knowledge Argument for Correctness of a Shuffle. In *EUROCRYPT*. 263–280.
[11] Eli Ben-Sasson, Iddo Bentov, Alessandro Chiesa, Ariel Gabizon, Daniel Genkin, Matan Hamilis, Evgenya Pergament, Michael Riabzev, Mark Silberstein, Eran Tromer, and Madars Virza. 2017. Computational Integrity with a Public Random String from Quasi-Linear PCPs. In *EUROCRYPT*. 551–579.
[12] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2018. Fast Reed-Solomon Interactive Oracle Proofs of Proximity. In *ICALP*. 14:1–14:17.
[13] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2018. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptology ePrint Archive* 2018 (2018), 46.
[14] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2019. Scalable Zero Knowledge with No Trusted Setup. In *CRYPTO*. 701–732.
[15] Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. 2020. Proximity Gaps for Reed-Solomon Codes. Cryptology ePrint Archive, Report 2020/654. (2020). https://eprint.iacr.org/2020/654.
[16] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. 2016. Short Interactive Oracle Proofs with Constant Query Complexity, via Composition and Sumcheck. *IACR Cryptology ePrint Archive* 2016 (2016), 324.
[17] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *IEEE Symposium on Security and Privacy*. 459–474.
[18] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. 2013. On the concrete efficiency of probabilistically-checkable proofs. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*. 585–594.
[19] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. 2013. SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge. In *CRYPTO*. 90–108.
[20] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. 2019. Aurora: Transparent Succinct Arguments for R1CS. In *EUROCRYPT*. 103–128.
[21] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. 2016. Interactive Oracle Proofs. In *TCC*. 31–60.
[22] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. Scalable Zero Knowledge via Cycles of Elliptic Curves. In *CRYPTO*. 276–294.
[23] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture. In *USENIX*. 781–796.
[24] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. 2013. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *STOC*. 111–120.
[25] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. 2016. Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting. In *EUROCRYPT*. 327–357.
[26] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. 2018. Bulletproofs: Short Proofs for Confidential Transactions and More. In *S&P*. 315–334.
[27] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. 2017. Post-Quantum Zero-Knowledge and Signatures from Symmetric-Key Primitives. In *CCS*. 1825–1842.
[28] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. 2015. Geppetto: Versatile Verifiable Computation. In *2015 IEEE Symposium on Security and Privacy, SP*. 253–270.
[29] Ivan Damgård and Yuval Ishai. 2006. Scalable Secure Multiparty Computation. In *CRYPTO*. 501–520.
[30] Amos Fiat and Adi Shamir. 1986. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO*. 186–194.
[31] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. 2013. Quadratic Span Programs and Succinct NIZKs without PCPs. In *EUROCRYPT*. 626–645.
[32] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. 2016. ZKBoo: Faster Zero-Knowledge for Boolean Circuits. In *USENIX*. 1069–1083.
[33] Shafi Goldwasser, Yael Tauman Kalai, and Guy Rothblum. 2008. Delegating computation: interactive proofs for muggles. In *STOC 2008*. 113–122.
[34] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. 2015. Delegating Computation: Interactive Proofs for Muggles. *J. ACM* 62, 4 (2015), 27:1–27:64.
[35] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1985. The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract). In *STOC*. 291–304.
[36] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1989. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. Comput.* 18, 1 (1989), 186–208.
[37] Jens Groth. 2009. Linear Algebra with Sub-linear Zero-Knowledge Arguments. In *CRYPTO*. 192–208.
[38] Jens Groth. 2010. Short Pairing-Based Non-interactive Zero-Knowledge Arguments. In *ASIACRYPT*. 321–340.
[39] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. 2007. Efficient Arguments without Short PCPs. In *CCC*. 278–291.
[40] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. 2007. Zero-knowledge from secure multiparty computation. In *STOC*. 21–30.
[41] Yael Tauman Kalai and Ran Raz. 2008. Interactive PCP. In *ICALP*. 536–547.
[42] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. 2018. Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures. In *CCS*. 525–537.
[43] Joe Kilian. 1992. A Note on Efficient Zero-Knowledge Proofs and Arguments (Extended Abstract). In *STOC*. 723–732.
[44] Helger Lipmaa. 2012. Progression-Free Sets and Sublinear Pairing-Based Non-Interactive Zero-Knowledge Arguments. In *TCC*. 169–189.
[45] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. 1990. Algebraic Methods for Interactive Proof Systems. In *FOCS*. 2–10.
[46] Ralph C. Merkle. 1989. A Certified Digital Signature. In *CRYPTO*. 218–238.
[47] Silvio Micali. 1994. CS Proofs (Extended Abstracts). In *FOCS*. 436–453.
[48] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2013. Pinocchio: Nearly Practical Verifiable Computation. In *S&P*. 238–252.
[49] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. 2016. Constant-round interactive proofs for delegating computation. In *STOC*. 49–62.
[50] Adi Shamir. 1990. IP=PSPACE. In *FOCS*. 11–15.
[51] Riad S. Wahby, Srinath T. V. Setty, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. 2015. Efficient RAM and control flow in verifiable outsourced computation. In *NDSS*.
[52] Riad S Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. 2018. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 926–943.
[53] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. 2018. DIZK: A Distributed Zero Knowledge Proof System. In *USENIX*. 675–692.
[54] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. 2019. Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation. In *Advances in Cryptology (CRYPTO)*.
[55] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. 2020. Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof. In *S&P 2020*.
[56] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. vSQL: Verifying Arbitrary SQL Queries over Dynamic Outsourced Databases. In *IEEE Symposium on Security and Privacy*. 863–880.
[57] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. A Zero-Knowledge Version of vSQL. Cryptology ePrint, 2017. (2017).

A ADDITIONAL EXPERIMENTAL RESULTS

DNA profile matching (DPM). The DNA profile match (DPM) is an application considered in [13]. The server holds a database of DNA profile releases the commitment of the database. A client can query the database, outputting three potential results: match, no match and partial match. The server then uses a zero knowledge proof scheme to prove that the query result is carried out correctly, and the public can verify this statement.

The detailed computation is described in [13, Appendix E]. The number of constraints to test each DNA entry is 90,954 [13, Figure 4]. We estimate the performance of Ligero++ based on the size of R1CS constraints and present the numbers in Table 4.

number of entries	Prover Time(s)	Verfication Time(s)	Proof Size(KB)
16	63	0.84	142
32	118	0.89	163
64	200	0.92	178

Table 4: Performance of Ligero++ for DPM

Batched verification of ECDSA signatures. Elliptic Curve Digital Signature Algorithm (ECDSA) is commonly used in cryptocurrencies to sign transactions. An important application of zero knowledge proofs is to aggregate multiple ECDSA signatures and generate a proof for batched verification. The verification of each ECDSA signature consists of one hashing and several modulo exponentiation, and the number of constraints for one verification is 32,768 using SHA-256 as the hash function. We estimate the performance of Ligero++ for batched verification of ECDSA signatures in Table 5. As shown in the table, both the verification time and the proof size grow sub-linearly. As each ECDSA signature is 64 bytes and takes around 3.8ms to verify, we can see that the verification time of our batch verification is already faster than verifying ECDSA signatures directly for $n = 64$ signatures. We also estimate that the proof size will be smaller than the total size of signatures for $n \geq 3,200$ which are relevant for blockchain applications, and the performance of our scheme demonstrates that we could potentially use the batched verification to improve both the proof size and the verification time.

n	Prover Time(s)	Verfication Time(s)	Proof Size(KB)
1	2.6	0.15	76
4	5.1	0.17	100
16	15.7	0.21	122
64	50.7	0.23	146

Table 5: Performance of Ligero++ for batched verification of ECDSA.