Improving Line-Point Zero Knowledge: Two Multiplications for the Price of One

Samuel Dittmer* Yuval Ishai† Steve Lu‡ Rafail Ostrovsky§

Abstract

Recent advances in fast protocols for *vector oblivious linear evaluation* (VOLE) have inspired a family of new VOLE-based lightweight designated-verifier NIZK protocols (Weng et al., S&P 2021, Baum et al., Crypto 2021, Dittmer et al., ITC 2021, Yang et al., CCS 2021). In particular, the Line-Point Zero Knowledge (LPZK) protocol of Dittmer et al. has the advantage of being entirely non-cryptographic given a single instance of a random VOLE correlation.

We present improvements to LPZK through the introduction of additional structure to the correlated randomness. Using an efficiently realizable variant of the VOLE correlation, we reduce the online proof size of LPZK by roughly 2x: from roughly 2 field elements per multiplication gate, or 1 element in the random oracle variant, to only 1 or $\frac{1}{2}$ elements respectively. In particular, we get the first practical VOLE-based NIZK that breaks the 1-element-per-multiplication barrier.

We implemented an optimized version of our protocol and compared it with other recent VOLE-based NIZK protocols. In the typical case where communication is the bottleneck, we get at least 2x performance improvement over all previous VOLE-based protocols. When prover computation is the bottleneck, we outperform all non-LPZK protocols by at least 2-3x and (our optimized implementation of) LPZK by roughly 30%, obtaining a 2-3x slowdown factor compared to plain circuit evaluation.

1 Introduction

There has been a recent flurry of designated-verifier non-interactive zero-knowledge (NIZK) proof systems [17, 8, 3, 18] taking advantage of improvements in secure generation of correlated randomness, in particular, "silent" generation of a random vector oblivious linear evaluation (VOLE) correlation [4, 15, 5, 7]. VOLE-based NIZK protocols significantly reduce the prover's computational overhead while allowing execution in a streaming fashion, with a memory footprint comparable to that of evaluating the circuit in the clear.

Among the VOLE-based NIZK protocols, Line-Point Zero Knowledge (LPZK) [8] achieved the best communication complexity, requiring slightly more than 1 field element of communication per gate for arithmetic circuits over large enough fields. This was extended to Boolean circuits by Quicksilver [18], whose optimized implementation gave unprecedented end-to-end runtimes for large-scale instances of zero-knowledge proofs. In particular, when considering the task of (designated-verifier) NIZK over fast networks, VOLE-based protocols significantly outperform all existing *succinct* proof ZK systems based on linear PCPs (such as SNARKs and STARKs), interactive proofs and IOPs (see §1.3 for discussion and literature pointers and § 5.3 for benchmarking comparisons).

*Stealth Software Technologies Inc. samdittmer@stealthsoftwareinc.com

†Department of Computer Science, Technion. yuvali@cs.technion.ac.il

‡Stealth Software Technologies Inc. steve@stealthsoftwareinc.com

§University of California, Los Angeles. Department of Computer Science and Mathematics. rafail@cs.ucla.edu

Protocol	Comm.	Prover comp.			Prover speed
		$\times$	$+$	$H(\cdot)$	
Plain circuit eval.		**1**	**0**	**0**	**49.6 M/sec**
Mac 'n' Cheese [3]	3	23	15	3	3.6 M/sec
Wolverine [17]	2	13	16	3	0.96 M/sec
IT-LPZKv1 [8]	$(2+\frac{1}{t})$	4	7	0	**19.6 M/sec**
Quicksilver [18]	1	15	13	< 1	7.8 M/sec
IT-LPZKv2	$(1+\frac{1}{t})$	3	5	0	**21.8 M/sec**
ROM-LPZKv2	$\frac{1}{2}$	8.5	8.5	1	**9.8 M/sec**

Table 1: Online communication and prover computation for VOLE-based NIZK. Communication is given in field elements per multiplication gate, ignoring lower order terms depending on input and output size. "Prover comp." columns count the number of multiplications, additions, and calls to a cryptographic hash function, respectively per multiplication gate (an addition gate requires two additions). Speeds are given in millions of multiplication gates per second on a single-threaded machine; see § 1.4 and § 5.1 for further discussion. The parameter t in the IT-LPZK protocols corresponds to a soundness error of $(2t+1)/|\mathbb{F}|$. All estimates and benchmarks are done using a computational security parameter of $\kappa = 128$ and a field $\mathbb{F} = \mathbb{F}_{2^{61}-1}$. Numbers in red come from this paper. Verifier costs are lower than prover costs, see § 5.2.

An additional advantage of VOLE-based protocols is that their low overhead does not only hold when considering big instances of zero knowledge, but kicks in even when (sequentially) proving many small statements over committed data. This can be useful for achieving security against malicious parties in secure computation protocols [8] and other applications. The key advantage of VOLE-based protocols is that VOLE generation can be done offline, thereby maximizing the speed of online proof generation.

1.1 Our contribution

We present two variants of a new designated-verifier NIZK protocol in the preprocessing model that improve on LPZK in terms of communication complexity and online computation. We implemented and benchmarked both variants, demonstrating a concrete improvement over Quicksilver in terms of the prover and verifier's online runtimes.

In order to fairly compare our improvements when comparing to the original LPZK paper of [8], we also implemented the original protocol as well (which we call LPZKv1), which was previously not implemented to the best of our knowledge. The two variants presented in this paper, which we call IT-LPZKv2 and ROM-LPZKv2, to refer to the information-theoretic and random oracle variants, respectively, require approximately half of the online communication of the corresponding LPZKv1 variants, and approximately 30% less online computation (see §5.1 for further discussion). We give informal theorem statements below, and formal theorem statements of the IT and ROM variants in Sections 3 and 4, respectively. See Table 1 for a summary of the performance of our new protocols compared to previous VOLE-based protocols.

Our protocols take advantage of a variant of the VOLE correlation that we call *quadratically certified* VOLE, or qVOLE. We propose two efficient methods for securely realizing the qVOLE correlation required by both variants of our protocol. The first method bootstraps off of an existing instance of VOLE, and requires a linear amount of communication in the online phase (effectively pushing 50% of the communication of LPZKv1 to an offline phase). The other method uses ring-

LPN to give a sublinear-communication qVOLE generation protocol that is concretely efficient in the SIMD setting or circuits with repeated subcircuits (such as hash trees). See Section 1.2 for further discussion.

Our information-theoretic protocol, IT-LPZKv2, requires $1 + \frac{1}{t}$ elements of communication, $(3 - \frac{1}{t})$ multiplications by the prover, and 2 multiplications by the verifier per multiplication gate, where t is a parameter guaranteeing soundness error $(2t + 1)/|\mathbb{F}|$.

Our random oracle model protocol, ROM-LPZKv2, requires $\frac{1}{2}$ elements of communication and $4.5 + 2r$ multiplications by both the prover and verifier (where $r \geq 1$ is approximately the computational security parameter divided by the log of the field size) for *layered circuits*, where each gate is assigned to some layer k, and all the inputs to gates at layer k are outputs to gates at layer $k-1$. The approach of ROM-LPZKv2 can be applied to general circuits, with a variable amount of communication savings depending on the circuit structure. Indeed, as we discuss in §4.1, for a random circuit made up entirely of multiplication gates, we still achieve an approximately 38% reduction in communication. For layered circuits, 50% is in fact a *lower bound* on the total communications savings.

Finally, we remark that the results below can be extended from arithmetic circuits containing only fan-in 2 addition and multiplication gates to circuits with arbitrary degree 2 polynomial gates by using the technique of [18] (applied to those degree 2 polynomial gates rather than the entire circuit). In fact, we will give the protocols and proofs in terms of degree 2 polynomial gates, but write our informal theorem statements below in terms of the usual addition and multiplication gates for clarity.

Theorem 1.1 (NIZK over qVOLE, informal)**.** *Fix an integer $t \geq 1$. There exists an (unconditional, perfect zero-knowledge) NIZK protocol in the qVOLE-hybrid model (defined in §2.2) that proves the satisfiability of an arithmetic circuit C over a field $\mathbb{F}$, where C has k inputs, k' outputs and m multiplication gates, with the following security and efficiency features:*

- **Soundness error:** $\varepsilon = (2t + 1)/|\mathbb{F}|$*;*
- **Communication:** $k + (1 + \frac{1}{t})m$ *field elements from P to V;*
- **qVOLE length:** $n = k + 2m$ *field elements;*
- **Computation:** *Assuming the cost of field additions is negligible compared to multiplications, the computation of the prover is less than* 3 *times the cost of evaluation in the clear, and the computation of the verifier is less than* 2 *times the cost of evaluation in the clear.*

Theorem 1.2 (NIZK over qVOLE in the ROM, informal)**.** *Fix an integer $r \geq 1$. There exists an (unconditional) NIZK protocol in the RO-qVOLE-hybrid model that proves the satisfiability of a layered arithmetic circuit C over a field $\mathbb{F}$, where C has k inputs and m multiplication gates, and ℓ is the number of oracle calls a malicious prover P^* makes, with the following features:*

- **Soundness error:** $\varepsilon = \frac{2}{|\mathbb{F}|} + \frac{\ell}{|\mathbb{F}|^r}$*;*
- **Communication:** $k + \frac{1}{2}m + 2r$ *field elements;*
- **qVOLE length:** $n = k + m + r$ *field elements;*
- **Computation:** *Computation of $O(r|C|)$ field operations and a cryptographic hash from $\mathbb{F}^m$ to $\mathbb{F}^{mr}$ for both the prover and the verifier.*

1.2 Overview of techniques

We now give a brief overview of the main technical ideas.

From certified LPZK to NIZK over qVOLE. In an LPZK proof system [8], the prover encodes the witness as a line $\mathbf{a}t + \mathbf{b}$ and the verifier evaluates the line at a random point α. An LPZK can be naturally compiled in to an NIZK protocol over a VOLE correlation. In a VOLE correlation, the prover holds *random* vectors $\mathbf{a}', \mathbf{b}'$ and the verifier receives $\mathbf{a}'\alpha + \mathbf{b}'$. We extend this to a *certified LPZK* proof system, where a malicious prover is required to output vectors $\mathbf{a}, \mathbf{b}$ such that the vector $\mathbf{a}$ satisfies a set of quadratic relations, and give a natural compiler from certified LPZK to NIZK over qVOLE. The qVOLE correlation is an instance of *certified VOLE* [8] that imposes quadratic constraints on the entries of $\mathbf{a}'$ (see §2.2 for formal definition).

As in [8], we take advantage of the algebraic structure of qVOLE, treating the expression $\mathbf{v} = \mathbf{a}\alpha + \mathbf{b}$ as a linear polynomial in α whose entries can be added and multiplied together, not only as a commitment scheme. Critically, we modify the approach of [8], by using the vector $\mathbf{b}$, rather than the vector $\mathbf{a}$ to hold the prover's witness and the intermediate wire values of the circuit. From the prover's perspective, the two choices are identical, but the verifier is able to get rid of one multiplication by α per multiplication gate (in addition to the computational savings from qVOLE). In fact, this improvement in verifier computation applies even to LPZK over standard VOLE.

Proofs of polynomial degeneracy. In order to reduce communication, both of our protocols rely on batched proofs of polynomial degeneracy functionalities, $\mathcal{F}_{PoPD}^{k,\mathbb{F}}$ in the ROM and $\mathcal{F}_{PoLD}^{1,\mathbb{F}}$ in the IT construction. These functionalities and their realizations are given implicitly in [8], but are given explicitly here to clarify the protocols and proofs, in §2.5.

In both funtionalities, the parties hold or construct shares where the prover holds the coefficients of a group of polynomials and the verifier holds the evaluation of those polynomials at a secret value α. The prover then seeks to convince the verifier that the leading term of all polynomials is 0.

The first functionality is more general, allowing polynomials of arbitrary degree, but is circuit dependent. The second functionality is defined only for linear polynomials, but is simpler to define and leads to information-theoretic security.

We show that the first functionality $\mathcal{F}_{PoPD}^{k,\mathbb{F}}$ can be realized with sublinear communication in the random oracle model, requiring only communication of kr elements for a polynomial of degree k of arbitrary length, where r is a parameter that controls soundness error. We also show that the second functionality $\mathcal{F}_{PoLD}^{1,\mathbb{F}}$ can be realized with sublinear communication in the information-theoretic setting, requiring only one element of communication for every t entries batched together.

Improvements in information-theoretic LPZK. The fundamental idea behind the use of qVOLE is to move the computation of certain products $a_i a_j$ out of the online phase and into the preprocessing phase.

The information-theoretic protocol for certified LPZK can be understood as a modification of either IT-LPZKv1, replacing one of the elements of communication with an entry of qVOLE, or of ROM-LPZKv1, replacing the call to $\mathcal{F}_{PoPD}^{k,\mathbb{F}}$ to a use of an entry of qVOLE and a call to $\mathcal{F}_{PoLD}^{1,\mathbb{F}}$.

Half-free multiplication. Applying qVOLE to the protocol in the random oracle model is more difficult, since we need to allow the prover and verifier to process some gates non-interactively.

We do so by dividing the wires $a\alpha + b$ up into two types. For some wires, the values a are determined purely by the correlated randomness. For other wires, the values a are determined by a function of the correlated randomness and the input.

As we discuss in more detail in §4.1, this can be conceptualized with a wire coloring, where the wires with a determined purely by correlated randomness are colored red, and the other wires are

colored blue.

The key observation, then, is that, by taking the product of the polynomial expression for two red wires, and using an appropriately constructed entry of qVOLE, it is possible to form the output wire, without any communication. The output wire, however, will be colored blue, and the same technique does not apply to a product of two blue wires, or of a blue and a red wire. The only communication required by the protocol is a call to $\mathcal{F}_{PoPD}^{k,\mathbb{F}}$ and the messages from prover to verifier adjusting the constant terms of red wires.

In the case of layered circuits, it is clear that half of the layers can be colored red and half colored blue. For simplicity, our proofs and protocols are written in terms of layered circuits, but we discuss the case of general circuits as well.

qVOLE generation. We propose two different approaches for efficiently realizing the qVOLE correlation. First, qVOLE can be realized from an existing instance of VOLE, with additional communication from the prover to the verifier that is linear in the circuit size. In fact, the communication costs in the offline phase of realizing qVOLE in this way are equal to the savings gained in the online phase; in LPZKv2 we move around half of the communication and one-third of the computation into the offline phase. This gives a strict improvement over LPZKv1 in typical settings where offline work is cheaper than online work.

Combining our online protocol with this first qVOLE generation method can be understood as a refinement of a *certified VOLE* functionality from [8], which was introduced in the context of an application to secure computation. Some alterations are required because of the reversal of the roles of the vectors $\mathbf{a}$ and $\mathbf{b}$, and further optimizations can be applied because we are restricted to certifications of degree 2 polynomials. We give these details in §2.3.

Our second approach for realizing the qVOLE correlation has sublinear communication cost and relies on the ring-LPN based pseudorandom correlation generators from [5] (Section 6.3), which are based on the Learning Parity with Noise assumption. The construction is concretely efficient when generating many copies of the same degree-2 correlated randomness, and hence is only applicable to the SIMD case (many instances of the same circuit). Using this realization of qVOLE reduces the *end-to-end* communication cost per multiplication of LPZKv2 by a factor of 2, with sublinear offline communication cost. Because of the ease with which VOLE-based proof systems can implement a "commit-and-prove" functionality, this approach also works well in a setting where circuits are made of a small family of subcircuits, repeated many times (e.g., when proving statements about hash trees). We give a detailed overview of the ring-LPN based realization of qVOLE in §2.4.

1.3 Related work

We now briefly compare our technique with related techniques from the literature.

VOLE-based ZK. In addition to LPZK and Quicksilver, which were discussed above, we compare our solution to Wolverine [17], a predecessor to Quicksilver, and Mac 'n' Cheese [3]. Wolverine was the first VOLE-based zero knowledge proof system to use subfield VOLE to transfer techniques from arithmetic circuits to boolean circuits. Mac 'n' Cheese allows for proving a nested disjunction, with communication cost proportional to the longest statement, using an approach that can be understood as analogous to stacked garbling [12]. LPZK is also noteworthy for providing information theoretic security non-interactively, after the initial VOLE setup. Quicksilver is noteworthy for applying the family of VOLE-based techniques to polynomials, which reduces the communication required to the input size plus the degree of the polynomial.

The notation differs slightly from protocol to protocol, but they share some key features. In each protocol, the prover uses the entries of VOLE to commit to their input *and* to intermediate wire

values in the circuit evaluation. This commitment produces a secret sharing of $w\alpha$, where w is the wire value and α is the secret element known to the verifier used in the VOLE. After, the prover uses additional entries to prove that the committed values satisfy the desired circuit relations. When we wish to refer to protocols that produce such a secret sharing for *all* intermediate wire values, we say such a protocol is operating in the *gate-by-gate paradigm*.

Polynomial-based ZK. The most direct way to relax the gate-by-gate paradigm is to allow for gates that represent polynomials of degree greater than 2. The ZK protocol in Quicksilver for polynomial sets can be understood as the special case where the entire circuit is replaced with a single polynomial of degree d, and requires only $|I| + dr$ communication, where the term r is used to control soundness error, and can be set equal to 1 for sufficiently large fields.

More generally, in a layered circuit, by introducing gates representing arbitrary polynomials of degree at most 2^c, you can remove all gates except for those at layers congruent to i mod c, for any choice of i. Choosing i to remove as many gates as possible, total communication will be $\leq |I| + |C|/c + 2^c r$. However, at least $2^c r$ computation is required per gate (this is the cost to simply print the polynomial in question and multiply it by an element of $\mathbb{F}^r$), and so reducing communication by a constant factor c increases computation by a factor of at least $2^c/c$.

In contrast, our "half-free multiplication" approach *reduces* computation in the online phase, and meets the informal definition of the gate-by-gate paradigm given above.

zkSNARKs and other succinct protocols. The field of non-VOLE-based zero knowledge is broad and deep, and we can list here only a few significant protocols oriented towards concrete efficiency.

Groth's SNARK [11], building on the blueprint of GGPR [10], was the culmination of years of improvements in SNARKs, and is considered nearly optimal by some measures, such as proof size. However, it requires a setup in the form of a long structured reference string, and efforts have been made to improve on its prover computation. Among these efforts we mention three protocols that remove the need for a trusted setup: Ligero [2], which is built on the MPC-in-the-head paradigm, Virgo [19], which is IOP-based, and also works on layered circuits, where the proof size and verifier computation depend linearly on the circuit depth, and Spartan [16] which uses an existing polynomial commitment scheme as a black box to build polynomial commitment schemes that can handle sparse polynomials efficiently. Ligero and Virgo each require $O(n \log n)$ prover work for a circuit of size n, while Spartan requires nearly linear work[1]. Spartan has recently been improved by Cerberus [14], which gives post-quantum SNARKs and faster concrete runtimes.

Most of these schemes require large circuits for their asymptotic efficiency to be realized, and are not efficient when realizing small instances of "commit-and-prove." Moreover, the schemes based on the GKR protocol incur a biger computational overhead when the witness size is close to the circuit size.

1.4 Applications

The general advantage of VOLE-based protocols is their significant improvement in prover computation over other protocols like Virgo and zk-SNARKs, at the cost of communication that is linear in the size of the circuit. Therefore VOLE-based protocols, in general, outperform other ZK protocols in settings with fast networks, or in settings where computational costs are significant.

Our protocols improve over the state-of-the-art in VOLE-based protocols for arithmetic fields in computation and, in the case of the ROM-LPZKv2, in communication as well. Because our

[1]More specifically, Spartan's protocol requires $O_\lambda(n)$ work, where λ is a computational security parameter, and Spartan defines $O_\lambda(*)$ notation to mean that that the constant inside big-O is a function of λ.

protocols use a form of correlated randomness that is more expensive to generate than VOLE, our protocols will perform best with fast networks. We give here several examples of real-world applications where the use of our protocols would be desirable.

Fast networks. As shown in §5.3, for networks faster than 200 MBps, the time required for prover computation is larger than the time for communication for the most efficient VOLE-based protocols (and so, *a fortiori*, for Virgo and zk-SNARKs protocols). For fast internet connections, local area networks, and direct physical connections (e.g. chip readers), a VOLE-based protocol will have the best performance in the online-offline model.

As discussed above, our LPZKv2 solution is especially suited to a SIMD setting. For example, a secure credit card could be required to submit a proof that privately held card data about credit limits and balances are correct before authorizing a transaction. The authorization circuit is fixed across all transactions, although the transaction history is variable.

Cloud computing pricing. Prominent cloud computing platforms, including AWS, charge for computing power but do not charge for certain classes of intra-cloud communication. Therefore in these settings, the cost of communication is irrelevant (as long as it does not render the application infeasible) and computational resources required are the only relevant metric for determining cost.

Additionally, these platforms have various forms of surge pricing for usage during peak hours and discounts for using the platform at fixed times, or for flexibly using spare computing power. These price incentives give advantages to the use of the online-offline model and encourage moving as much computation as possible into the offline step.

Computationally expensive gates. When we allow arbitrary degree 2 polynomial gates, it becomes possible for the computational effort required to compute a single gate to be quite large, for example if the gate represents the dot product of long vectors. This example is discussed in [18], where they show that the use of their polynomial technique reduces the communication cost of matrix multiplication from $O(n^3)$ to $O(n^2)$ field elements.

Our use of generic degree two gates achieves the same reduction, and extends it to other computations, e.g., computations that use matrix multiplication as a sub-protocol in a larger circuit. The computational cost of matrix multiplication remains super-quadratic in n, so for n sufficiently large, the cost of computation will dwarf the cost of communication, regardless of the speed of the network. This example is discussed in more detail in Remark 4.

1.5 Arithmetic circuit formats

Our result for arithmetic circuits in the information-theoretic settings holds for arbitrary arithmetic circuits, but to clarify the presentation we write a circuit $C := ((w_i), (\mathcal{G}_j), \mathbb{F})$ as a collection of wires w_i over $\mathbb{F}$ and arbitrary fan-in degree 2 polynomial gates $\mathcal{G}_j$ acting on the wires, instead of the more common construction consisting entirely of fan-in two addition and multiplication gates and fan-in one scalar multiplication gates.

This more common construction can be recovered from our definition by noting that the inputs to any multiplication gate in the standard construction can be represented as a linear combination over $\mathbb{F}$ of outputs from prior multiplication gates, and the product of two linear functions gives a degree 2 polynomial gate.

The authors of Quicksilver [18] noted that it is possible to construct a NIZK with communication sublinear in the circuit size in cases such as matrix multiplication where a functionality could be represented by a low degree polynomial. In the language of our modified circuit format, matrix multiplication requires n^3 multiplications, but only n^2 degree 2 polynomial gates. In other words, we have embedded the observation of [18] into the circuit format.

2 Preliminaries

In this section we give a formal definition of our new notion of a *certified* LPZK proof system and show how to compile such a proof system into a designated-verifier NIZK when given a random qVOLE correlation. We also show how the underlying qVOLE functionality can be realized either by an offline LPZK-NIZK step or by ring-LPN.

2.1 Defining certified LPZK

The definition of a certified LPZK proof system should be compared to the definition of an LPZK proof system in [8]. The primary difference is that the prover's algorithm is required to output vectors $\mathbf{a}, \mathbf{b}$ where the vector $\mathbf{a}$ satisfies a set of quadratic relations, and the verifier receives a guarantee (certification) that this set of quadratic relation does indeed hold. There is additionally some metadata tracked by the *Prove* and *Verify* algorithms that we use to simplify the protocol descriptions.

As in [8], we focus here on the case of arithmetic circuit satisfiability, and work in an arithmetic model in which probabilistic polynomial time (PPT) algorithms can sample a uniformly random element from a finite field $\mathbb{F}$ and perform field operations at a unit cost. All of the protocols we describe make a black-box use of the underlying field $\mathbb{F}$.

As in [8], we remark that the proof systems we construct satisfy the stronger *proof of knowledge* property, but the proof does not add much additional insight and we omit the details to streamline the presentation. Again, as in [8], we give only the definition for a certified LPZK proof system with statistical security, but a certified LPZK proof system with computational security can be defined similarly.

Definition 1 (Certified LPZK)**.** A *certified line-point zero-knowledge* (certified LPZK) proof system for arithmetic circuit satisfiability is a triple of algorithms (*Setup*, *Prove*, *Verify*) with the following syntax:

- *Setup*$(\mathbb{F}, C)$ is a polynomial time algorithm that given an arithmetic verification circuit $C : \mathbb{F}^k \to \mathbb{F}^{k'}$ outputs an integer n, a set $I \subseteq [1, n]$, and a sequence $Q := (f_i)_{i \in I}$, where each $f_i \in \mathbb{F}[x_i]_{i \notin I}$ is either zero or a homogeneous polynomial of degree 2.
- *Prove*$(\mathbb{F}, C, w, n, I, Q)$ is a PPT algorithm that given an arithmetic verification circuit $C : \mathbb{F}^k \to \mathbb{F}^{k'}$ and a witness $w \in \mathbb{F}^k$, outputs a pair of vectors $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$ that specify an affine line $\mathbf{v}(t) := \mathbf{a}t + \mathbf{b}$ such that $f_i(\mathbf{a}) = a_i$ for all $i \in I$. For $i \in I$ such that $f_i \neq 0$ the values b_i are chosen uniformly at random from $\mathbb{F}$, and for all $i \notin I$, the values a_i are chosen uniformly at random from $\mathbb{F}$.
- *Verify*$(\mathbb{F}, C, \alpha, \mathbf{v}_\alpha, n, I, Q)$ is a polynomial-time algorithm that, given an evaluation $\mathbf{v}_\alpha$ of the line $\mathbf{v}(t)$ at some point $\alpha \in \mathbb{F}$, outputs *acc* or *rej*.

We note that the restriction that certain entries a_i, b_i be randomly distributed is not inherently necessary. We impose it to simplify the definition of qVOLE and the compiler from certified LPZK to NIZK over qVOLE in §2.2 below. The algorithms (*Setup*, *Prove*, *Verify*) should satisfy the following:

- **Completeness**. For any arithmetic circuit $C : \mathbb{F}^k \to \mathbb{F}^{k'}$ and witness $w \in \mathbb{F}^k$ such that $C(w) = \mathbf{0}$, for $(n, I, Q) =$ *Setup*$(\mathbb{F}, C)$, and for any fixed $\alpha \in \mathbb{F}$, we have $\Pr[\mathbf{v}(t) \xleftarrow{R}$ *Prove*$(\mathbb{F}, C, w, n, I, Q) :$ *Verify*$(\mathbb{F}, C, \alpha, \mathbf{v}(\alpha), n, I, Q) =$ *acc*$] = 1$.

- **Reusable ε-soundness**. For every arithmetic circuit $C : \mathbb{F}^k \to \mathbb{F}^{k'}$ such that $C(w) \neq \mathbf{0}$ for all $w \in \mathbb{F}^k$, with $(n, I, Q) = \mathsf{Setup}(\mathbb{F}, C)$, and every (adversarially chosen) line $\mathbf{v}^*(t) = \mathbf{a}^* t + \mathbf{b}^*$, where the length n of $\mathbf{v}^*$ depends on C as above, and $f_i(\mathbf{a}^*) = a_i^*$ for all $i \in I$, we have $\Pr[\alpha \xleftarrow{R} \mathbb{F} : \mathsf{Verify}(\mathbb{F}, C, \alpha, \mathbf{v}^*(\alpha), n, I, Q) = \mathsf{acc}] \leq \varepsilon$. Moreover, for every $\mathbb{F}, C, \mathbf{v}^*(t)$ the probability of Verify accepting (over the choice of α) is either 1 or $\leq \varepsilon$. Unless otherwise specified, we assume $\varepsilon \leq O(1/|\mathbb{F}|)$.

- **Perfect zero knowledge**. There exists a PPT simulator Sim such that, for any arithmetic circuit $C : \mathbb{F}^k \to \mathbb{F}^{k'}$, any witness $w \in \mathbb{F}^k$ such that $C(w) = \mathbf{0}$, any $\alpha \in \mathbb{F}$, and $(n, I, Q) = \mathsf{Setup}(\mathbb{F}, C)$, the output of $\mathsf{Sim}(\mathbb{F}, C, \alpha)$ is distributed identically to $\mathbf{v}(\alpha)$, where $\mathbf{v}(t)$ is the affine line produced by $\mathsf{Prove}(\mathbb{F}, C, w, n, I, Q)$.

The *reusable* soundness requirement guarantees that even by observing the verifier's decision bit on a maliciously chosen circuit C, and line $\mathbf{v}^*(t) = \mathbf{a}^* t + \mathbf{b}^*$, the prover learns essentially nothing about the verifier's secret point α, which allows the same α to be reused without substantially compromising soundness.

Complexity measures for certified LPZK: (n, n', n'')-cLPZK. As in [8], in addition to the dimension/length parameter n, we use two other parameters n' and n'' as complexity measures for certified LPZK. We stress that n' and n'' do not correspond to the n' and n'' used for LPZK. Concretely, the parameter n' is the number of elements $i \in I$ for which $f_i \neq 0$, while n'' is the number of elements $i \in I$ for which $f_i = 0$.

2.2 Compiling certified LPZK to NIZK over qVOLE

We now give a simple compiler that converts a certified LPZK proof system into a (designated verifier) NIZK protocol by means of a certain variant of a certified VOLE functionality we define here called qVOLE (see [8, 6] for a general discussion of certified OLE and certified VOLE).

This compiler is similar to the compiler from LPZK to NIZK in [8], although we make one high-level, conceptual change. In [8], the wire values are held in the vector $\mathbf{a}$, and the vector $\mathbf{b}$ is used as a mask, while in this work, the vector $\mathbf{b}$ holds the wire values, and the vector $\mathbf{a}$ holds the masks. This reduces the cost for the verifier, since V no longer needs to multiply P's message by α.

Definition 2 (Quadratically certified random VOLE)**.** Quadratically certified random VOLE (qVOLE) is a two-party functionality that takes as input from the sender and receiver a triple (n, I, Q) that is equal to the output of $\mathsf{Setup}(\mathbb{F}, C)$ for some arithmetic verification circuit C. The functionality outputs to the sender a pair of vectors $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$ and to the receiver a value $\alpha \in \mathbb{F}$ and the vector $\mathbf{a}\alpha + \mathbf{b}$. The vector $\mathbf{a}$ satisfies $f_i(\mathbf{a}) = a_i$ for all $i \in I$, and $a_i \xleftarrow{R} \mathbb{F}$ for $i \notin I$. The vector $\mathbf{b}$ is chosen uniformly at random from $\mathbb{F}^n$.

Lemma 2.1 (From certified LPZK to NIZK)**.** *Given (n, n', n'')-cLPZK over $\mathbb{F}$ with soundness error ε, there is an NIZK protocol that uses a single instance of qVOLE of length $n - n''$ and requires communication of $n - n'$ field elements from the prover to the verifier.*

Proof. The prover and verifier are given a random qVOLE of length n, so that the prover holds $(\mathbf{a}', \mathbf{b}')$, and the verifier holds $\mathbf{v}' = \mathbf{a}'\alpha + \mathbf{b}'$ for a random $\alpha \in \mathbb{F}$.

By the randomness guarantees of cLPZK, the algorithm $\mathsf{Prove}(\mathbb{F}, C, w)$ can take $\mathbf{a} = \mathbf{a}'$ and then compute $\mathbf{b}$ from $\mathbf{a}$. The prover sends the vector $\mathbf{b} - \mathbf{b}'$ to the verifier, who then computes

$$\mathbf{v} = \mathbf{v}' + (\mathbf{b}' - \mathbf{b}).$$

The relations $f_i(\mathbf{a}) = a_i$ for $i \in I$ and the randomness of a_i for $i \notin I$ are guaranteed by the qVOLE protocol. This compiler requires communication of n elements to send the entire vector $\mathbf{b}' - \mathbf{b}$, and a qVOLE of length n.

However, we can set $b'_i = b_i$ for $i \in I$ such that $f_i \neq 0$, since those entries are randomly distributed under $\mathsf{Prove}(\mathbb{F}, C, w)$. This forces $\mathbf{b}' - \mathbf{b} = 0$ for those i, and so the prover can omit those values from the message to the verifier, reducing communication complexity to $n - n'$ elements. Similarly, a qVOLE instance is not required for the entries $i \in I$ such that $f_i = 0$, giving an adjusted qVOLE of length $n = n''$, as desired. The security completeness, soundness, and zero knowledge properties of the above NIZK protocol are inherited directly from the corresponding properties of the LPZK. □

For efficient end-to-end NIZK protocols, we need practically efficient protocols to realize the qVOLE functionality. We give two realizations below.

2.3 qVOLE from general certified VOLE

The qVOLE functionality above can be realized as a special case of the certified VOLE with a general relation established in [8][Lemma 6.2]. However, that general protocol is designed to certify arithmetic relations involving terms from distinct instances of VOLE, and is therefore not optimized for this setting. We prove the following lemma in Appendix §A.1.

Lemma 2.2. *Fix an integer $r \geq 1$. A receiver R and a sender S can realize the functionality $\mathcal{F}^{(\mathbb{F})}_{qVOLE}(n, I, Q)$, in the RO-rVOLE hybrid model with a single instance of random VOLE. This VOLE instance has total length*

$$n + |Q|$$

, and the protocol requires communication of

$$|Q| + 2r$$

field elements from sender to receiver and has $\frac{k}{|\mathbb{F}|} + \frac{\ell}{|\mathbb{F}|^r}$ soundness error, where ℓ is the number random oracle calls a malicious prover can make.

2.4 qVOLE from ring-LPN

As discussed in §1.2, the qVOLE functionality can be understand as a flavor of multiplication triple (MT)-type correlations, and these correlations can in turn be understood in terms of simple atomic operations under a "correlation calculus". Atomic operations consist of picking a random vector $\mathbf{x} \in F^n$, computing the scalar product $\beta\mathbf{x}$, computing the point-wise product $\mathbf{x} \cdot \mathbf{y}$, extending a vector by copying certain entries, sending a vector to a party, and sharing a vector between parties. Standard authenticated triples come from computing $\mathbf{z} := \mathbf{x} \cdot \mathbf{y}$ and $\beta\mathbf{z}$ and sharing all four vectors.

With that understanding, the proof of the following lemma follows from Theorem 6.4 of [5].

Lemma 2.3. *A receiver R and a sender S can produce a collection of realizations of the functionality $\mathcal{F}^{(\mathbb{F})}_{qVOLE}(n, I, Q)$ under the Ring-LPN assumption with communication costs sublinear in n.*

2.5 Batched proofs of polynomial degeneracy

In Figure 1, we give a proof of polynomial degeneracy functionality whereby a prover P's initial message $\mathbf{m}$ allows P to learn the coefficients of B polynomials of degree k, while a verifier V learns the evaluation of those polynomials on a secret value α (i.e. their VOLE input), and P convinces V that the leading term of all polynomials is equal to zero (i.e. that the polynomial is *degenerate*) if their initial message was formed correctly.

In practice, we want the message $\mathbf{m}$ that P sends in step 1 be sufficient for each party to determine their messages from $\mathcal{F}_{PoPD}^{k,\mathbb{F}}$ in step 2. To that end, we say that a pair of protocols (Π^P, Π^V) has *Property S* if the outputs of $(\Pi^P(\mathbf{m})$ are a series of vectors $(\mathbf{x}^0, \ldots, \mathbf{x}^k)$, and one of COOPERATE, DEFECT, the output of $\Pi^V(\mathbf{m})$ is $(\alpha, \mathbf{y})$, and together these values satisfy the conditions given in step 2.

In this paper (Π^P, Π^V) will always be obtained from the outputs of (*Prove*, *Verify*) corresponding to a block of B multiplication gates in the circuit, and $\mathbf{m}$ will be the part of P's proof corresponding to that block.

In Figure 3, we give a related functionality for the special case where $k = 1$. Our protocol realizing this functionality has information theoretic security, and so we no longer require an initialization message $\mathbf{m}$, which simplifies the definition.

In the following two lemmas, we give two protocols that realize these functionalities. These functionalities and their realizations are found implicitly in [8], but we include them here for completeness and clarity of presentation.

Lemma 2.4. *Let (Π_{PD}^P, Π_{PD}^V) be any pair of protocols with property S. The protocol in Figure 2 realizes the functionality $\mathcal{F}_{PoPD}^{k,\mathbb{F}}$ in the random oracle model with a random oracle $H : \mathbb{F}^s \times [1, \ldots, B] \to \mathbb{F}^r$, soundness error $\frac{k}{|\mathbb{F}|} + \frac{\ell}{|\mathbb{F}|^r}$, where ℓ is the number of calls a malicious prover can make to H, communication of kr field elements, Bkr multiplications by the prover, and $(2r+1)k + Br$ multiplications by the verifier.*

Proof. **Completeness:** This follows directly from the correctness of the OPE functionality and the relation

$$\sum_{i=0}^{k-1} \mathbf{x}^i \alpha^i = \mathbf{y}.$$

Zero Knowledge: The verifier's simulator generates each of z_i uniformly at random over $\mathbb{F}^r$, and computes $w := \left(\sum_{i=0}^{k} z_i \alpha^i\right) - \sum_{j=1}^{B} H(\mathbf{m}; j) y_j$. The distribution of the z_i's is uniformly random, since the c_i's are generated uniformly at random, and w must satisfy the above relation.

Soundness: The expression

$$w + \sum_{j=1}^{B} H(\mathbf{m}; j) y_j - \left(\sum_{i=0}^{k-1} z_i \alpha^i\right)$$

is a polynomial in α of degree k with leading coefficient

$$\sum_{j=1}^{B} H(\mathbf{m}; j) x_j^k.$$

If this coefficient is nonzero, then for a malicious prover P^* to violate soundness, they must guess one of the k (possibly repeated) roots of this polynomial, which P^* can do with probability at most $\frac{k}{|\mathbb{F}|}$.

Figure 1: k-degree proof of polynomial degeneracy

Functionality $\mathcal{F}_{PoPD}^{k,\mathbb{F}}$: k-degree proof of polynomial degeneracy

Parametrized by a finite field $\mathbb{F}$, a degree k, and a batch size B, with the following syntax:

1. P sends a message $\mathbf{m} \in \mathbb{F}^B$ and one of $\{\textsc{Cooperate}, \textsc{Defect}\}$ to $\mathcal{F}_{PoPD}^{k,\mathbb{F}}$.
2. $\mathcal{F}_{PoPD}^{k,\mathbb{F}}$ chooses $(\mathbf{x}^0, \mathbf{x}^1, \ldots \mathbf{x}^k, \alpha, \mathbf{y})$, with $\mathbf{x}^i, \mathbf{y} \in \mathbb{F}^B$, $\alpha \in \mathbb{F}$,
$$\mathbf{y} = \sum_{i=0}^{k} \mathbf{x}^i \alpha^i$$
and $\mathbf{x}^k = \mathbf{0}$ if P sent $\textsc{Cooperate}$ and $\mathbf{x}^k \neq \mathbf{0}$ otherwise.
3. $\mathcal{F}_{PoPD}^{k,\mathbb{F}}$ sends $(\mathbf{x}^0, \mathbf{x}^1, \ldots \mathbf{x}^k)$ to P and $(\alpha, \mathbf{y}, \mathbf{m})$ to V.
4. $\mathcal{F}_{PoPD}^{k,\mathbb{F}}$ sends $\perp$ to V if $\mathbf{x}^k \neq \mathbf{0}$ or if
$$\sum_{i=0}^{k-1} \mathbf{x}^i \alpha^i \neq \mathbf{y}.$$

Figure 2: k-degree proof of polynomial degeneracy protocol

Protocol $\Pi^{k,\mathbb{F}}_{PoPD}$: k-degree proof of polynomial degeneracy

Parametrized by a finite field $\mathbb{F}$, a degree k, a batch size B, and a pair of protocols (Π^P_{PD}, Π^V_{PD}) with property S.

1. P sends a message $\mathbf{m} \in \mathbb{F}^B$ to V.
2. P learns $(\mathbf{x}^0, \cdots, \mathbf{x}^k)$ from $\Pi^P_{PD}(\mathbf{m})$ and V learns $(\alpha, \mathbf{y})$ from part Π^V_{PD}.
3. P and V engage in Π_{VOLE} so that P holds $(\mathbf{a}, \mathbf{b})$ and V holds $\mathbf{v}$, with $\mathbf{v} = \mathbf{a}\alpha + \mathbf{b}$, and all vectors of length $k-1$, with indices from 1 to $k-1$.
4. For $i = 0, \ldots, k-1$, P sets $c_i := a_i + b_{i+1}$, with $a_0 := b_k := 0$.
5. V sets $w := \sum_{i=1}^{k} v_i \alpha^{i-1}$.
6. P sends
$$z_i := c_i + \sum_{j=1}^{B} H(\mathbf{m}; j) x_j^i$$
to V.
7. V verifies that
$$\left(\sum_{i=0}^{k-1} z_i \alpha^i \right) = w + \sum_{j=1}^{B} H(\mathbf{m}; j) y_j$$
and otherwise outputs $\perp$.

Figure 3: Proof of linear degeneracy

Functionality $\mathcal{F}_{PoLD}^{1,\mathbb{F}}$: Proof of linear degeneracy

Parametrized by a finite field $\mathbb{F}$ and a batch size B.

- Initially, P holds $(\mathbf{x}^0, \mathbf{x}^1)$ where each $\mathbf{x}^i \in \mathbb{F}^B$, and V holds $(\alpha, \mathbf{y})$, with $\alpha \in \mathbb{F}$ and $\mathbf{y} = \alpha\mathbf{x}^1 + \mathbf{x}^0$.
- P sends $(\mathbf{x}^0, \mathbf{x}^1)$ to $\mathcal{F}_{PoLD}^{1,\mathbb{F}}$.
- V sends $(\alpha, \mathbf{y})$ to $\mathcal{F}_{PoLD}^{1,\mathbb{F}}$.
- $\mathcal{F}_{PoLD}^{1,\mathbb{F}}$ sends $\perp$ to V if $\mathbf{x}^1 \neq \mathbf{0}$ or if $\mathbf{y} \neq \alpha\mathbf{x}^1 + \mathbf{x}^0$.

Because Π_{PD}^P satisfies property P, for any fixed choice of $\mathbf{m}$, the resulting vector $\mathbf{x}^k$ will have at least one nonzero entry, and so the probability that

$$\sum_{j=1}^{B} H(\mathbf{m}; j) x_j^k = 0$$

is $\frac{1}{|\mathbb{F}|}$. To boost soundness error, we repeat steps (2) through (6) on a family of hash functions H_i, for $i = 1, \ldots, r$.

Over ℓ queries to H (where on a single query, we allow P^* to learn the value of $H_i(\mathbf{m}; j)$ for all i, j, for the sake of simplicity), P^* has a probability of success of $\frac{\ell}{|\mathbb{F}|^r}$, as desired. □

Lemma 2.5. *The protocol in Figure 4 realizes the functionality $\mathcal{F}_{PoLD}^{1,\mathbb{F}}$ with soundness error $\frac{2B}{|\mathbb{F}|}$, communication of 1 field element, and $B - 1$ multiplications in $\mathbb{F}$ by each party.*

Proof. **Completeness:** If $\mathbf{x}^1 = \mathbf{0}$, then $x_i^0 = y_i$ for $1 \leq i \leq B$, and so the values of z match.

Perfect Zero Knowledge: The verifier's simulator generates the sender's message z by making the substitution $y_i \leftarrow 1$ whenever $y_i = 0$, and setting

$$z := \prod_{i=1}^{B} y_i.$$

This matches the verifier's view during an honest run of the protocol exactly.

Soundness: The verifier does not need to be convinced that $\mathbf{y} = \alpha\mathbf{x}^1 + \mathbf{x}^0$, since they are already guaranteed this from the initial setup. If a cheating prover holds $\mathbf{x}^1 \neq \mathbf{0}$, then let $I \subseteq [1, B]$ be the set of indices i such that at least one of x_i^0, x_i^1 is nonzero. Then the prover can compute the polynomial

$$f(t) := \prod_{i \in I} (x_i^1 t + x_i^0),$$

of degree at least 1 and at most B. Let z^* be the cheating prover's message, and let z be the product of the nonzero elements of $\mathbf{y}$. Then if $z^* = z$, either $z^* = f(\alpha)$, or for some $i \in I$, $y_i = x_i^1\alpha + x_i^0 = 0$.

Figure 4: Proof of linear degeneracy protocol

Protocol $\Pi^{1,\mathbb{F}}_{PoLD}$: Information-theoretic proof of linear degeneracy protocol

Parametrized by a finite field $\mathbb{F}$ and a batch size B.

1. Initially, P holds $(\mathbf{x}^0, \mathbf{x}^1)$ where each $\mathbf{x}^i \in \mathbb{F}^B$, and V holds $(\alpha, \mathbf{y})$, with $\alpha \in \mathbb{F}$ and $\mathbf{y} = \alpha\mathbf{x}^1 + \mathbf{x}^0$.
2. P substitutes $x^0_i \leftarrow 1$ whenever $x^0_i = 0$.
3. V substitutes $y_i \leftarrow 1$ whenever $y_i = 0$.
4. P computes $z := \prod_{i=1}^B x^0_i$.
5. V computes $z' := \prod_{i=1}^B y_i$.
6. P sends z to V.
7. V verifies that $z = z'$ and otherwise returns $\perp$.

The first possibility occurs with probability at most $B/|\mathbb{F}|$, since P does not know α, and $f(t) - z^*$ has at most B roots. By a union bound, the second possibility occurs with probability at most $B/|\mathbb{F}|$, since for any $i \in I$, the probability that $x^1_i\alpha + x^0_i = 0$ is equal to 0 if $x^1_i = 0$, and $1/|\mathbb{F}|$ otherwise.

□

Remark 3. The functionality $\mathcal{F}^{1,\mathbb{F}}_{PoLD}$ can be used to instead prove $\mathbf{x}^1 = 0$ by having P swap $\mathbf{x}^0$ and $\mathbf{x}^1$ and V assign $\mathbf{y} \leftarrow \alpha^{-1}\mathbf{y}$ and $\alpha \leftarrow \alpha^{-1}$, and then executing $\mathcal{F}^{1,\mathbb{F}}_{PoLD}$. This requires an additional B multiplications by the verifier. When performing multiple instances of $\mathcal{F}^{1,\mathbb{F}}_{PoLD}$, R can reduce computation by instead computing the value α^{-B} once and then computing

$$\alpha^{-B} \prod_{y_i \neq 0} y_i$$

for each instance of $\mathcal{F}^{1,\mathbb{F}}_{PoLD}$, at a cost of 1 additional multiplication per instance.

3 Certified LPZK

We now state and prove a more refined version of Theorem 1.1.

Let $C = ((w_i), (\mathcal{G}_j), \mathbb{F})$ be an arithmetic circuit with k inputs, k' outputs, and $|C|$ degree 2 polynomial gates, as described in §1.2. We write a gate $\mathcal{G}_j := (g_j, I_j, (w_i)_{I_j})$, where g_j is a degree 2 polynomial acting on the wires w_i, for i in the index set I_j, with output wire w_j.

The algorithm $\mathsf{Setup}(\mathbb{F}, C)$ sets $n := k + (2 + \frac{1}{t})m$, I equal to the set $[k - k' + m + 1, \ldots, k - k' + (2 + \frac{1}{t})m]$, and $Q := (f_j)_{j \in I}$ defined by setting f_j equal to the degree 2 part of $g_{j-k+k'-m}$ for gate $\mathcal{G}_{j-k+k'-m}$, for $k - k' + m + 1 \leq j \leq k - k' + 2m$, and $f_j = 0$ otherwise.

We give the algorithms Prove_{IT} and Verify_{IT} in Figures 5 and 6, respectively. We give the full proof of the following theorem in Appendix §A.2.

Figure 5: Prover algorithm for certified IT-LPZK

Algorithm $\mathsf{Prove}_{IT}(\mathbb{F}, n, I, Q, w)$

Parametrized by a finite field $\mathbb{F}$, a degree k, and a batch size B.

1. P chooses $\mathbf{a}^1, \mathbf{b}^1 \leftarrow_R \mathbb{F}^{k+m}$, and sets $\mathbf{b}^1 := \mathbf{w} = (w_i)_{1 \le i \le k+m-k'}$.
2. $\mathbf{a}^1, \mathbf{b}^1[m-k'+1:m] \leftarrow \mathbf{0}^{k'}$.
3. P chooses $\mathbf{b}^2 \leftarrow_R \mathbb{F}^m$ and sets $\mathbf{a}^2_j = Q(\mathbf{a}^1)$, for $j = 1, \ldots, m$.
4. For each gate $\mathcal{G}_j$, P computes the values
$$x^1_j := \left(\frac{dg_j}{dz}(\mathbf{a}^1 z + \mathbf{b}^1)(0)\right) - a^1_j - b^2_j$$
and
$$x^0_j = g_j(\mathbf{b}^1) - b^1_j.$$
5. P computes a message $\mathbf{z}$ using $\mathcal{F}^{1,\mathbb{F}}_{PoLD}(\mathbf{x}^0, \mathbf{x}^1)$ and sets $\mathbf{b}^3 := \mathbf{z}$.
6. P sets $\mathbf{a}^3 := \mathbf{0}$, a vector of length $\frac{m}{t}$.
7. Return $(\mathbf{a}^1 \cup \mathbf{a}^2 \cup \mathbf{a}^3, \mathbf{b}^1 \cup \mathbf{b}^2 \cup \mathbf{b}^3)$.

Theorem 3.1 (Certified LPZK for arithmetic circuit satisfiability)**.** *For any NP-relation $R(x, y)$ and finite field $\mathbb{F}$, there exists a certified LPZK system for R over $\mathbb{F}$ with soundness error $O(1/|\mathbb{F}|)$. Concretely, in the case of proving the satisfiability of an arithmetic circuit C over $\mathbb{F}$, we get LPZK over $\mathbb{F}$ with the following size parameters (n, n', n'') and soundness error ε for every integer $t \ge 1$. If C has k inputs, k' outputs, and m degree 2 polynomial gates, we have $n = k + (2 + \frac{1}{t})m$, $n' = m$, $n'' = \frac{m}{t} + k'$, $\varepsilon = 2t/|\mathbb{F}|$. Moreover, assuming that the cost of additions in the field are negligible compared to the cost of multiplications, the computation of the prover is less than 3 times the cost of evaluation in the clear, and the computation of the verifier is less than 2 times the cost of evaluation in the clear.*

3.1 Certified LPZK protocol

To simplify the indexing, we write the output from the algorithm $\mathsf{Prove}(\mathbb{F}, C, w)$ as eight vectors $\mathbf{a}, \mathbf{a}^1, \mathbf{a}^2, \mathbf{a}^3, \mathbf{b}, \mathbf{b}^1, \mathbf{b}^2, \mathbf{b}^3$, with $\mathbf{a} := \mathbf{a}^1 \cup \mathbf{a}^2 \cup \mathbf{a}^3$ and $\mathbf{b} := \mathbf{b}^1 \cup \mathbf{b}^2 \cup \mathbf{b}^3$. The vectors $\mathbf{a}^1, \mathbf{b}^1$ correspond to the indices $[1, k-k'+m]$, the vectors $\mathbf{a}^2, \mathbf{b}^2$ correspond to the indices $[k-k'+m+1, k-k'+2m]$, and the vectors $\mathbf{a}^3, \mathbf{b}^3$ correspond to the indices $[k-k'+2m+1, k+(2+\frac{1}{t})m]$. We then re-index each vector so that, e.g. a^1_j and b^2_j each will be consumed by the jth degree 2 polynomial gate.

The vectors $\mathbf{a}^1, \mathbf{b}^1$ correspond to all indices $i \notin I$ from $\mathsf{Setup}(\mathbb{F}, C)$. The vector $\mathbf{b}^1$ represents

Figure 6: Verifier algorithm for certified IT-LPZK

Algorithm *Verify*$_{IT}(\mathbb{F}, n, I, Q, \alpha, \mathbf{v}_\alpha)$

Parametrized by a finite field $\mathbb{F}$, a degree k, and a batch size B.

1. $\mathbf{v}_\alpha^1 \leftarrow \mathbf{v}_\alpha[1 : k+m]$, i.e. the first $k+m$ entries of $\mathbf{v}_\alpha$.
2. $\mathbf{v}_\alpha^1[m-k'+1 : m] \leftarrow \mathbf{0}^{k'}$
3. $\mathbf{v}_\alpha^2 \leftarrow \mathbf{v}_\alpha[k-k'+m+1 : k-k'+2m]$
4. $\mathbf{v}_\alpha^3 \leftarrow \mathbf{v}_\alpha[k-k'+2m+1 : k-k'+(2+\frac{1}{t})m]$
5. For each gate $\mathcal{G}_j$, V computes the values
$$y_j := g_j(\mathbf{v}_\alpha^1) - v_{\alpha,j}^2\alpha - v_{\alpha,j}^1.$$
6. V calls $\mathcal{F}_{PoLD}^{1,\mathbb{F}}(\alpha, \mathbf{y}, \mathbf{v}_\alpha^3)$ and returns rej if the protocol returns $\perp$.
7. Return acc.

all wire values in the circuit, and the vector $\mathbf{a}^1$ masks those wire values. By the definition, we have $\mathbf{a}^1$ chosen uniformly at random. We have $|\mathbf{a}^1| = |\mathbf{b}^1| = k+m-k'$, take $\mathbf{b} := \mathbf{w} = (w_i)_{i \le k+m-k'}$, and extend both vectors to length $k+m$ by adding zeros.

The vector $\mathbf{a}^2$ holds the action of the polynomial sequence Q on the vector $\mathbf{a}^1$. We have $|\mathbf{a}^2| = |\mathbf{b}^2| = m$, with the vector $|\mathbf{a}^2|$ defined by the polynomial sequence Q. The vector $\mathbf{b}^2$ is chosen uniformly at random from $\mathbb{F}^m$. For each gate $\mathcal{G}_j$, the prover computes the polynomial

$$g_j(\mathbf{a}^1 z + \mathbf{b}^1) - \alpha(a_j^2 z + b_j^2) - (a_j^1 z + b_j^1).$$

The term $(a_j^1 z + b_j^1)$ can be dropped when $\mathcal{G}_j$ is an output wire, since b_j^1 must be equal to 0, and so there is no need to hide the value.

By construction, this is a linear function in z, and P sets the result equal to $x_j^1 + x_j^0$, as shown in step (3) of the algorithm. Moreover, if P is honest, $x_j^0 = 0$, and so P executes a proof of linear degeneracy on $(\mathbf{x}^0, \mathbf{x}^1)$ to show that $\mathbf{x}^0 = \mathbf{0}$, using the modification of $\mathcal{F}_{PoLD}^{1,\mathbb{F}}$ described in Remark 3. The prover sets $\mathbf{b}^3$ equal to the message sent during the execution of $\mathcal{F}_{PoLD}^{1,\mathbb{F}}$ and sets $\mathbf{a}^3 := 0$.

In the algorithm *Verify*$(\mathbb{F}, C, \alpha, \mathbf{v}_\alpha)$, the verifier subdivides $\mathbf{v}_\alpha$ into $\mathbf{v}_\alpha^1, \mathbf{v}_\alpha^2, \mathbf{v}_\alpha^3$ whose values are equal to $\mathbf{a}^i\alpha + \mathbf{b}^i$, for $i \in \{1,2,3\}$ in an honest run of the protocol.

For each gate $\mathcal{G}_j$, the verifier computes the value

$$y_j := g_j(\mathbf{v}_\alpha^1) - v_{\alpha,j}^2\alpha - v_{\alpha,j}^1$$

and plays the part of the receiver in a proof of linear degeneracy. The expression for $\mathbf{y}$ is the evaluation of the polynomial $\mathbf{x}^1 z + \mathbf{x}^0$ computed by the prover at $z = \alpha$, and again the $v_{\alpha,j}^1$ term can be dropped when $\mathcal{G}_j$ is an output gate.

3.2 Cost analysis of certified LPZK

Communication: As described in §2.2, the communication complexity of (n, n', n'')-certified LPZK is $n - n'$, since P sends one field element to V for each element of $\mathbf{b}^1$ and $\mathbf{b}^3$, but sends nothing for $\mathbf{a}^2$ or $\mathbf{b}^2$. Thus total communication complexity is $k + (1 + \frac{1}{t})m$. Note that k' has no impact on communication complexity because the cost of verifying a degree 2 polynomial gate is the same as cost of verifying the gate and its output.

Prover computation: To run the protocol, the prover must compute the values

$$\left(\frac{dg_j}{dz}(\mathbf{a}^1 z + \mathbf{b}^1)(0)\right)$$

and $g_j(\mathbf{b}_1)$ and play the role of the prover in $\frac{m}{t}$ executions of $\mathcal{F}^{1,\mathbb{F}}_{PoLD}$. But the derivative evaluated at zero can be computed as:

$$\left(\frac{dg_j}{dz}(\mathbf{a}^1 z + \mathbf{b}^1)(0)\right) = g_j(\mathbf{a}^1 + \mathbf{b}^1) - f_j(\mathbf{a}^1) - g_j(\mathbf{b}^1),$$

and $f_j(\mathbf{a}^1)$ is computed in a preprocessing step as part of the qVOLE functionality.

Therefore the total computation cost for the prover is equal to two times the cost of evaluation in the clear plus the cost of $t - 1$ field multiplications per execution of $\mathcal{F}^{1,\mathbb{F}}_{PoLD}$, for a total cost of $(1 - \frac{1}{t})m + 2\mathsf{comp}(C)$ field multiplications.

Verifier computation: The values $\alpha v^2_{\alpha,j}$ can be computed from $\mathbf{v}^2_\alpha$ in a preprocessing step, as can the value α^{-t} required for $\mathcal{F}^{1,\mathbb{F}}_{PoLD}$, as described in Remark 3. The computations $g_j(\mathbf{v}^1_\alpha) - v^2_{\alpha,j}\alpha - v^1_{\alpha,j}$ can be done with the same amount of verifier work as executing the circuit in the clear, plus the cost of two subtractions per gate.

Additionally, the verifier requires t multiplications for each batch of t gates in $\mathcal{F}^{1,\mathbb{F}}_{PoLD}$, for a total cost of $m + \mathsf{comp}(C)$ field multiplications.

Remark 4. In the special case of certified-LPZK for $N \times N$ matrix multiplication, we have, using the standard schoolbook algorithm, $k = k' = m = N^2$, and $\mathsf{comp}(C) = N^3$ field multiplications. We therefore have $m = o(\mathsf{comp}(C))$, and for N sufficiently large, prover work is roughly equal to 2 times the cost of computation in the clear, and verifier work is roughly equal to the cost of computation in the clear.

The concrete efficiency parameters are $(3 + \frac{1}{t})N^2$ field elements of communication, $2N^3 + (1 - \frac{1}{t})N^2$ multiplications by the prover, and $N^3 + 2N^2$ multiplications by the verifier.

Because matrix multiplication is a homogeneous polynomial of degree 2, the cost of computing the functions $g_j(\mathbf{a}^1 + \mathbf{b}^1)$, $g_j(\mathbf{b}^1)$ is exactly equal to two times the cost of evaluating matrix multiplication. Therefore we can replace the standard schoolbook algorithm with any algorithm for matrix multiplication, reducing the computational cost for both parties to $O(N^{2.8074})$ with the Strassen algorithm, or to $O(N^{2.3728596})$ with the best asymptotic matrix multiplication algorithm due to Alman and Williams [1]. Communication cost will be $(3+\frac{1}{t})N^2 = o(\mathsf{comp}(C))$ field elements regardless of the matrix multiplication algorithm used.

For practical values of N, the standard algorithm or the Strassen algorithm should be used, with the Strassen algorithm becoming preferable somewhere between $N = 500$ and $N = 1000$, see e.g. [13, 9].

Figure 7: Prover algorithm for certified ROM-LPZK

Algorithm $\mathit{Prove}_{ROM}(\mathbb{F}, n, I, Q, w)$

Parametrized by a finite field $\mathbb{F}$, a degree k, and a batch size B.

1. P chooses $\mathbf{a}^1 \leftarrow_R \mathbb{F}^{k+m'}$ and sets $\mathbf{b}^1 := \mathbf{w} = (w_i)_{i \in K \cup I_\tau}$.
2. P chooses $\mathbf{b}^2 \leftarrow_R \mathbb{F}^{m-m'}$ and sets $\mathbf{a}^2_j = Q(\mathbf{a}^1)$, for $j = 1, \ldots, m$.
3. $\mathbf{a}^3, \mathbf{b}^3 \leftarrow \mathbf{0}^r$
4. $\mathbf{a}^4, \mathbf{b}^4 \leftarrow \mathbf{0}^{k+m-m'}$
5. $(\mathbf{a}^4, \mathbf{b}^4)[1:k] \leftarrow (\mathbf{a}^1, \mathbf{b}^1)[1:k]$
6. For each gate $\mathcal{G}_j$, in sequence:
   - If $j \in I_\tau$, the prover computes
   $$x^1_j := f_j(\mathbf{a}^4)$$
   and
   $$x^0_j = \left(\frac{dg_j}{dz}(\mathbf{a}^4 z + \mathbf{b}^4)(0)\right) - a^1_j.$$
   - If $j \in I_{1-\tau}$ the prover computes the values
   $$a^4_j := \left(\frac{dg_j}{dz}(\mathbf{a}^1 z + \mathbf{b}^1)(0)\right) - b^2_j$$
   and
   $$b^4_j := g_j(\mathbf{b}^1).$$
7. For each output gate $\mathcal{G}_j$, the prover computes the values
   $$x^1_j := f_j(\mathbf{a})$$
   and
   $$x^0_j = \frac{dg_j}{dz}(\mathbf{a}z + \mathbf{b})(0).$$
8. P calls $\mathcal{F}^{2,\mathbb{F}}_{PoPD}$ on $(\mathbf{x}^0, \mathbf{x}^1)$, producing the message $\mathbf{z}$, and sets $\mathbf{b}^3 := \mathbf{z}$.
9. P sets $\mathbf{a}^3 := \mathbf{0}$.
10. Return $(\mathbf{a}^1 \cup \mathbf{a}^2 \cup \mathbf{a}^3, \mathbf{b}^1 \cup \mathbf{b}^2 \cup \mathbf{b}^3)$.

4 Certified LPZK in the ROM

Let $C = ((w_i), (\mathcal{G}_j), \ell, \mathbb{F})$ be an arithmetic circuit with k inputs, k' outputs, and $|C|$ degree 2 polynomial gates, as described in §1.5. As above, we write a gate $\mathcal{G}_j := (g_j, I_j, (w_i)_{I_j})$, where g_j is a degree 2 polynomial acting on the wires w_i, for i in the index set I_j, with output wire w_j. Here $\ell : (w_i) \to \mathbb{N}$ is a grading that assigns each wire to a layer, with the requirement that $\ell(w_j) = \ell(w_i) + 1$, for each wire $w_i \in I_j$ that is an input to gate $\mathcal{G}_j$. For input wires w_i, we set $\ell(w_i) = 0$, and let ℓ_{max} be the maximum value of $\ell(w_i)$ (achieved on all output wires). Let $K = [1, \ldots, k]$ be the set of input wire indices, let I_0 be the set of indices i corresponding to wires w_i with $\ell(w_i)$ even and $0 < \ell(w_i) < \ell_{max}$, and let I_1 the set of indices i corresponding to wires w_i with $\ell(w_i)$ odd and $0 < \ell(w_i) < \ell_{max}$. Set

$$m' = \min\{|I_0|, |I_1|\} \leq \frac{m}{2},$$

and set $\tau = 0$ if $m' = |I_0|$ and $\tau = 1$ otherwise.

The algorithm $\mathsf{Setup}(\mathbb{F}, C)$ sets $n := k + m + r$, I equal to the set $[k + m' + 1, \ldots, k + m + r]$, and $Q := (f_j)_{j \in I}$ defined by setting f_j equal to the degree 2 part of g_i, where i is the jth element of I_τ, and g_i corresponds to gate $\mathcal{G}_i$, for $k + m' + 1 \leq j \leq k + m$, and $f_j = 0$ otherwise. We give the algorithms Prove_{ROM} and Verify_{ROM} in Figures 7 and 8, respectively. We give the proof of the following theorem in Appendix §A.3.

Theorem 4.1 (LPZK in the ROM)**.** *For any positive integer r, there exists an LPZK in the ROM for arithmetic circuit satisfiability for layered circuits, with the following size parameters (n, n', n'') and soundness error. If C has k inputs, k' outputs, and m multiplication gates, we have $n = k + m + r$, $n' = m'$, $n'' = r$. For any malicious prover making ℓ calls to a random oracle $H : \mathbb{F}^m \to \mathbb{F}^{mr}$, the soundness error is $\varepsilon = \frac{2}{|\mathbb{F}|} + \frac{\ell}{|\mathbb{F}|^r}$. Moreover, the computation of both the prover and the verifier consists of $O(r|C|)$ and a single call to H.*

4.1 Certified LPZK protocol in the ROM

To simplify the indexing, we write the output from the algorithm $\mathsf{Prove}(\mathbb{F}, C, w)$ as eight vectors $\mathbf{a}, \mathbf{a}^1, \mathbf{a}^2, \mathbf{a}^3, \mathbf{b}, \mathbf{b}^1, \mathbf{b}^2, \mathbf{b}^3$, with $\mathbf{a} := \mathbf{a}^1 \cup \mathbf{a}^2 \cup \mathbf{a}^3$ and $\mathbf{b} := \mathbf{b}^1 \cup \mathbf{b}^2 \cup \mathbf{b}^3$. The vectors $\mathbf{a}^1, \mathbf{b}^1$ correspond to the indices $[1, k + m']$ and the vectors $\mathbf{a}^2, \mathbf{b}^2$ correspond to the indices $[k + m' + 1, k + m]$, and the vectors $\mathbf{a}^3, \mathbf{b}^3$ correspond to the indices $[k + m + 1, k + m + r]$. Additionally, the prover computes another pair of vectors $(\mathbf{a}^4, \mathbf{b}^4)$ that are not included in the vectors $\mathbf{a}, \mathbf{b}$, with $|\mathbf{a}^4| = \mathbf{b}^4| = k + m - m'$. As above, we then re-label the indices of all vectors so that the element at index j will be consumed by the the jth multiplication gate.

The vectors $\mathbf{a}^1, \mathbf{b}^1$ correspond to all indices $i \notin I$ from $\mathsf{Setup}(\mathbb{F}, C)$. These represent the input wires and all wires at a level with parity τ, masked as entries of VOLE. By the definition, we have $\mathbf{a}^1$ chosen uniformly at random. We have $|\mathbf{a}^1| = |\mathbf{b}^1| = k + m'$, and take $\mathbf{b}^1 := \mathbf{w} = (w_i)_{i \in K \cup I_\tau}$.

We have $|\mathbf{a}^2| = |\mathbf{b}^2| = m - m'$, with the vector $|\mathbf{a}^2|$ defined by the polynomial sequence Q. The vector $\mathbf{b}^2$ is chosen uniformly at random from $\mathbb{F}^m$. These vectors hold the results of applying the degree 2 gates of the circuit to the vector $\mathbf{a}^1$, masked by the values $\mathbf{b}^2$.

The vectors $(\mathbf{a}^3, \mathbf{b}^3)$ only holds the message needed for $\mathcal{F}^{k,\mathbb{F}}_{PoPD}$, as in the information-theoretic protocol.

The first k entries of $\mathbf{a}^4$ and $\mathbf{b}^4$ are set equal to the first k entries of $\mathbf{a}^1$ and $\mathbf{b}^1$, respectively, for indexing purposes. The remaining entries of $\mathbf{a}^4$ and $\mathbf{b}^4$ hold the wire values of all wires at level with parity $1 - \tau$, together with the masks $\mathbf{a}^4$.

Figure 8: Verifier algorithm for certified ROM-LPZK

Algorithm $\mathit{Verify}_{ROM}(\mathbb{F}, n, I, Q, \alpha, \mathbf{v}_\alpha)$

Parametrized by a finite field $\mathbb{F}$, a degree k, and a batch size B.

1. $\mathbf{v}_\alpha^1 \leftarrow \mathbf{v}_\alpha[1 : k + m']$, i.e. the first $k + m'$ entries of $\mathbf{v}_\alpha$.
2. $\mathbf{v}_\alpha^2 \leftarrow \mathbf{v}_\alpha[k + m' + 1 : k + m]$
3. $\mathbf{v}_\alpha^3 \leftarrow \mathbf{v}_\alpha[k + m + 1 : k + m + r]$
4. $\mathbf{v}_\alpha^4 \leftarrow \mathbf{0}$, a vector of length $k + m - m'$.
5. $\mathbf{v}_\alpha^4[1 : k] \leftarrow \mathbf{v}_\alpha^1[1 : k]$.
6. For each gate $\mathcal{G}_j$, in sequence:
   - If $j \in I_\tau$, the verifier computes
   $$y_j := g_j(\mathbf{v}_\alpha^4) - v_{\alpha,j}^1.$$
   - If $j \in I_{1-\tau}$ the verifier computes
   $$v_{\alpha,j}^4 := g_j(v_{\alpha,j}^1) - v_{\alpha,j}^2 \alpha.$$
7. For each output gate $\mathcal{G}_j$, the verifier computes $y_j := g_j(\mathbf{v})$.
8. V calls $\mathcal{F}_{PoPD}^{2,\mathbb{F}}(\alpha, \mathbf{y}, \mathbf{v}_\alpha^3)$ and returns rej if the protocol returns $\perp$.
9. Return acc.

The vectors $\mathbf{a}^4$ and $\mathbf{b}^4$, and the corresponding vector $\mathbf{v}_\alpha^4$ held by the verifier, can be computed by the prover (resp. verifier) from the previous level's vectors $\mathbf{a}^1, \mathbf{v}^1$ and $\mathbf{a}^2, \mathbf{b}^2$ (resp. $\mathbf{v}_\alpha^1$ and $\mathbf{v}_\alpha^2$), without any communication. The correlated randomness held in $\mathbf{a}^2$ allows us to reduce the degree of the gate output of $g_j(\mathbf{a}^1 z + \mathbf{b}^1)$ without any communication. However, the resulting vector $\mathbf{a}^4$ depends upon the prover's input, and so we can no longer use correlated randomness can be used to get to next level. Therefore the prover must communicate to set the vector $\mathbf{b}^1$ on the next level, as in the information-theoretic variant, and then use $\mathcal{F}_{PoPD}^{2,\mathbb{F}}$ to show this was done correctly, and the cycle repeats.

For each gate $\mathcal{G}_j$, with $j \in I_\tau$, the prover computes the polynomial $g_j(\mathbf{a}^4 z + \mathbf{b}^4) - (a_j^1 z + b_j^1)$, and sets the z^2 and z coefficients to x_j^1 and x_j^0, respectively.

For gate $\mathcal{G}_j$ with $j \in_{1-\tau}$, the prover computes the polynomial $g_j(\mathbf{a}^1 z + \mathbf{b}^1) - \alpha(a_j^2 z + b_j^2)$ over z and sets (a_j^4, b_j^4) equal to the coefficients of z^1 and z^0, respectively (See step (6) of Figure 7.)

Finally, for each output gate $\mathcal{G}_j$, the prover computes the values

$$x_j^1 := f_j(\mathbf{a})$$

and

$$x_j^0 = \frac{dg_j}{dz}(\mathbf{a}z + \mathbf{b})(0),$$

and engages in a proof of linear degeneracy on $(\mathbf{x}^0, \mathbf{x}^1)$.

In the algorithm *Verify*$(\mathbb{F}, C, \alpha, \mathbf{v}_\alpha)$, the verifier subdivides $\mathbf{v}_\alpha$ into $\mathbf{v}_\alpha^1, \mathbf{v}_\alpha^2$, whose values are equal to $\mathbf{a}^i\alpha + \mathbf{b}^i$, for $i \in \{1, 2, 3\}$ in an honest run of the protocol, and creates the additional vector $\mathbf{v}_\alpha^4$, whose values will be computed by the verifier. The first k entries of $\mathbf{v}_\alpha^4$ are set equal to the first k entries of $\mathbf{v}_\alpha^1$.

For each gate $\mathcal{G}_j$, with $j \in I_\tau$ and with $j \in I_{1-\tau}$, the verifier computes the value corresponding to the evaluation of the prover's computed polynomial in z at $z = \alpha$ (see step (6) of Figure 8.)

Finally, for each output gate $\mathcal{G}_j$, the verifier computes the value

$$y_j := g_j(\mathbf{a})$$

and engages in a proof of linear degeneracy to verify that $\mathbf{y}\alpha = \mathbf{x}^1\alpha + \mathbf{x}^0$.

Remark 5. It is important to note that it is possible to apply this protocol to arbitrary circuits, not only layered circuits, by placing a gate index in I_τ when any of the input indices lie in $I_{1-\tau}$, and otherwise choosing whether to place it in I_τ or $I_{1-\tau}$. This can be stated as a coloring problem: Use the color red to denote wires for which the value a_i is determined purely by the correlated randomness, and use blue to denote wires for which the value a_i depends on the prover's input. Color the input wires red, then color the remaining wires of the circuit such that, for any gate with all blue inputs, or a mix of blue and red inputs, the output wire is red, while for a gate with all red inputs, the output wire may be red or blue. The communication cost using this approach is equal to the number of red wires.

For many potential applications, the savings are substantial. For example, let C be a circuit made up entirely of multiplication gates, where the two inputs to the ith gate are chosen at random from the outputs of gates $1, \ldots, (i-1)$. Then P_B, the probability a wire is blue, is asymptotically equal to P_R^2, where P_R is the probability a wire is red. But of course $P_B + P_R = 1$, and we can solve to find $P_R = (-1 + \sqrt{5})/2 \approx 0.62$, that is, we achieve a 38% reduction in communication.

4.2 Communication and computational complexity of certified LPZK in the ROM

Communication: As described in §2.2, the communication complexity of (n, n', n'')-certified LPZK is $n - n'$, since P sends one field element to V for each element of $\mathbf{b}^1$ and $\mathbf{b}^3$, but sends nothing for $\mathbf{a}^2$ or $\mathbf{b}^2$. Thus total communication complexity is

$$k + m' + r \leq k + \tfrac{m}{2} + r.$$

Prover computation: As in the non-ROM protocol, the prover must compute:

$$\left(\frac{dg_j}{dz}(\mathbf{a}^\iota z + \mathbf{b}^\iota)(0)\right) = g_j(\mathbf{a}^\iota + \mathbf{b}^\iota) - f_j(\mathbf{a}^\iota) - g_j(\mathbf{b}^\iota),$$

for each gate $\mathcal{G}_j$, and $\iota \in \{1, 4\}$. However, for gates $\mathcal{G}_j$ with $j \in I_\tau$, the value $f_j(\mathbf{a}^4)$ can no longer be computed in a preprocessing step, since $\mathbf{a}^4$ depends on P's witness.

Therefore the total computation cost for the prover is between $2\mathsf{comp}(C)$ and $3\mathsf{comp}(C)$ field multiplications, plus the cost of a proof of linear degeneracy of length $m' + k$, with $2(m' + k)r$ field multiplications and $m' + k$ calls to a random oracle.

Verifier computation: The computational cost for the verifier is equal to the cost of the non-ROM protocol, replacing the calls to $\mathcal{F}^{1,\mathbb{F}}_{PoLD}$ with a call to $\mathcal{F}^{k,\mathbb{F}}_{PoPD}$ of length $m' + k$. It therefore requires $(m' + k + 4)r + \mathsf{comp}(C)$ field multiplications and $m' + k$ calls to a random oracle.

5 Implementation and Benchmarking

We implemented our *Prove* and *Verify* algorithms in the IT and ROM variants, and benchmarked their performance. For our benchmarking, we worked in the preprocessing model, where the necessary qVOLE randomness had been generated in an offline step. The tests were done on an Amazon EC2 machine of type `m5.2xlarge` with a single thread.

5.1 Comparing with prior VOLE implementations

In Table 1, in the introduction, we compared the speed in millions of gates per second to those of Mac 'n' Cheese, Wolverine, and Quicksilver. The implementation of Quicksilver by Yang et al in [18] has since been surpassed on their Github, so we used their most recent numbers. Additionally, we implemented IT-LPZKv1, since to the best of our knowledge, this was not implemented by Dittmer et al in [8]. Finally, we implemented a plain circuit evaluation for comparison purposes. Quicksilver, like our work, used a single-threaded `m5.2xlarge` instance. Wolverine used a larger `m5.4xlarge` instance.

We note that the implementation of Mac 'n' Cheese, like our implementations of LPZK, distinguishes the actual silent generation of VOLE from on-line cost. Both Mac 'n' Cheese and Quicksilver include the cost of on-line communication, where our benchmarking tests are even more refined: we measure the *Prove* and *Verify* algorithms cost independently. We therefore use their benchmarking run on a LAN to give as equal of a comparison as possible.

The benchmarked estimates are all roughly consistent with the columns of computations, with two exceptions - the plain circuit evaluation and Wolverine are slower than would be expected from the number of operations required per gate. Even the plaintext circuit has to read the witness from memory before performing additions and multiplications and iterate through the gates, which may be a substantial portion of its cost. We remark that pipelining and multi-threading to read the

Protocol	Prime	Prover	Verifier	Ratio
IT-LPZKv1	$\mathbb{F}_{2^{61}-1}$	51.1	30.3	0.59
IT-LPZKv2	$\mathbb{F}_{2^{61}-1}$	45.8	27.4	0.60
IT-LPZKv1	P-384	3050	1813	0.59
IT-LPZKv2	P-384	2556	1511	0.59
ROM-LPZKv2	$\mathbb{F}_{2^{61}-1}$	102.5	88.1	0.86

Table 2: Prover and verifier times for LPZK variants over various primes, and the ratio between the runtimes. All times are given in ms / million multiplication gates

circuit in, read the witness in, and begin the proof and verification in parallel could close this gap. and, more importantly, speed up the LPZK protocols further.

Similarly, we note that we might expect a 33% improvement in runtime from IT-LPZKv1 to IT-LPZKv2, since the latter protocol requires 3 multiplications per gate instead of 4. Instead, we got approximately a 10% improvement. Again, it is possible that, since field operations over the Mersenne prime $2^{61}-1$ are highly optimized, the cost of additions, subtractions, control flow, and iterating over the various forms of randomness may be contributing.

It is not entirely clear why Wolverine is unusually slow. Ultimately, of course, the reason Wolverine is slow is that the authors turned their attention to optimizing the code of Quicksilver, and deprecated Wolverine. Additionally, Wolverine uses a slower VOLE generation procedure than later works, and includes this time in its benchmarking, as well as some communication and pre-processing that is linear in the circuit size. Quicksilver also includes the VOLE generation time in its cost, but uses the faster Silver procedure [7] so that the costs here become negligible. Finally, Quicksilver includes optimizations around hashing that are applicable to our setting as well but we have not implemented. Wolverine discusses various choices here (and in other parts of their protocol), but it is not clear which choices were actually implemented.

5.2 LPZK prover and verifier online runtimes

In Table 2, we give the prover and verifier runtimes for our LPZK protocols. For the IT-variant, we give runtimes over $\mathbb{F}_{2^{61}-1}$ and P-384 (the prime associated with the elliptic curve of the same name).

For all the information-theoretic protocols and choices of primes, the verifier run-time is very close to 0.59 times the prover's run time. For the random oracle protocol, that ratio is around 0.85, probably because the cost of hashing is equal for both parties.

We also note that the prover's advantage in IT-LPZKv2 versus IT-LPZKv1 improves from 10% to 16% when using the larger prime. We suspect that this ratio continues to increase towards 30% for large enough primes, as the cost of multiplication dominates.

5.3 Comparing with state-of-the-art NIZK protocols and zkSNARKs

In Table 3, we additionally compare our algorithms in this paper and the Quicksilver algorithm to three of the fastest ZK algorithms used today, Groth16 [11], Virgo [19], and Cerberus [14]. In the rightmost column, we give the slowest network that can send the proof as quickly as the prover can generate it. Since our prover times are at least 4-10x faster than these non-based VOLE protocols, when we have networks running around $100-200$ MBps or faster, we can expect that our protocols, in particular IT-LPZKv2, give the fastest online time for implementing ZK.

Protocol	Comm.	P's Time	V's Time	Bottleneck
Groth16 [11]	192 b	21 s	< 2 ms	0.009 kBps
Virgo [19]	271 kB	478 ms	12.4 ms	567 kBps
Cerberus [14]	2.8 MB	2.17 s	148 ms	1.29 MBps
Quicksilver [18]	8MB	128 ms	< 128 ms	62.5 MBps
IT-LPZKv2	8MB	45.8 ms	27.4 ms	174.7 MBps
ROM-LPZKv2	4MB	102.5 ms	88.1 ms	39.0 MBps

Table 3: Comparison of Quicksilver and LPZKv2 online times to state-of-the-art NIZK protocols operating on one million gates. The protocols are all executed over $\mathbb{F}_{2^{61}-1}$ except Cerberus, which is executed over a generic 128-bit field. The bottleneck point is the minimum network speed required for the prover computation to be the performance bottleneck.

5.4 Comparing end-to-end times

Although our protocols are especially designed for the online-offline model, we also obtain highly competitive protocols in end-to-end runtime under reasonable assumptions about setting and network speed.

When computation is a bottleneck, for example, on a physically connected device, we can use the qVOLE compiler from plain VOLE, and get numbers for end-to-end prover and verifier computation for both IT-LPZKv1 and IT-LPZKv2 that will be roughly 2x faster than any other protocol. The additional cost of VOLE generation must be included here, but as discussed elsewhere, advances in VOLE in [7] render this cost negligible.

When dealing with medium speed networks, communication will be a bottleneck for both Quicksilver and ROM-LPZKv2. When using the ring-LPN based approach for generating qVOLE, ROM-LPZKv2 has, asymptotically, a 2x advantage over Quicksilver in end-to-end communication.

Concretely, we can use Table 3 in [5] and the surrounding discussion, taking $c = 4$, $w = 64$ with active, 128-bit security, to estimate the seed size, total end-to-end computation, and total end-to-end communication needed for ROM-LPZKv2. For a circuit with 32 million multiplication gates over $\mathbb{F}_{2^{61}-1}$ and a network speed of 0.5 Mbps, Quicksilver will require 516 s in this setting, while ROM-LPZKv2 will require only 406 s. For slower networks and larger circuits, of course, ROM-LPZKv2's advantage will be even greater.

6 Acknowledgements

Distribution Statement "A" (Approved for Public Release, Distribution Unlimited). This material is based upon work supported by DARPA under Contract No. HR001120C0087. 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. R. Ostrovsky is supported in part by DARPA under Cooperative Agreement HR0011-20-2-0025. Y. Ishai is supported in part by ERC Project NTSC (742754), BSF grant 2018393, and ISF grant 2774/20.

References

[1] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms*

(SODA), pages 522–539. SIAM, 2021.

[2] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 2017 acm sigsac conference on computer and communications security*, pages 2087–2104, 2017.

[3] Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In *CRYPTO 2021*, pages 92–122, 2021.

[4] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In *CCS 2018*, pages 896–912, 2018.

[5] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In *Crypto 2020*, pages 387–416, 2020.

[6] Melissa Chase, Yevgeniy Dodis, Yuval Ishai, Daniel Kraschewski, Tianren Liu, Rafail Ostrovsky, and Vinod Vaikuntanathan. Reusable non-interactive secure computation. In *CRYPTO 2019, Part III*, pages 462–488, 2019.

[7] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In *CRYPTO 2021, Part III*, pages 502–534.

[8] Sam Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. In *ITC 2021*, 2021.

[9] Paolo D'Alberto and Alexandru Nicolau. Using recursion to boost atlas's performance. In *High-Performance Computing*, pages 142–151. Springer, 2005.

[10] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *EUROCRYPT*, 2013.

[11] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, pages 305–326, 2016.

[12] David Heath and Vladimir Kolesnikov. Stacked garbling for disjunctive zero-knowledge proofs. In *EUROCRYPT 2020, Part III*, pages 569–598, 2020.

[13] Jianyu Huang, Tyler M Smith, Greg M Henry, and Robert A Van De Geijn. Strassen's algorithm reloaded. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 690–701. IEEE, 2016.

[14] Jonathan Lee, Srinath Setty, Justin Thaler, and Riad Wahby. Linear-time and post-quantum zero-knowledge snarks for r1cs. *Cryptology ePrint Archive*, 2021.

[15] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-OLE: Improved constructions and implementation. In *CCS 2019*, pages 1055–1072, 2019.

[16] Srinath Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *Annual International Cryptology Conference*, pages 704–737. Springer, 2020.

[17] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1074–1091. IEEE, 2021.

[18] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In *CCS*, 2021. Full version: `https://eprint.iacr.org/2021/076`.

[19] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *2020 IEEE Symposium on Security and Privacy*, pages 859–876, 2020.

A Deferred Proofs

A.1 Proof of Lemma 2.2

Proof. For each degree 2 polynomial gate $\mathcal{G}_i$, let there be a corresponding entry of VOLE $v_i = a_i\alpha + b_i$. Then S sends to R the value $d = f_i(\mathbf{a}) - a_i$, and R adds $\alpha d + v_i$ to give $f_i(\mathbf{a})\alpha + b_i$, as desired.

Then the expression $f_i(\mathbf{a}\alpha + b) - \alpha^2 d - \alpha v_i$ is a quadratic in α, with leading coefficient zero under an honest run of the protocol. P and V call $\Pi^{k,\mathbb{F}}_{PoPD}$ of degree 2 to complete the protocol.

Security follows from the security of VOLE and $\Pi^{k,\mathbb{F}}_{PoPD}$.

□

A.2 Proof of Thereom 3.1

Completeness: If P has a valid witness $\mathbf{w}$ for C and follows the protocol, we have $g_j(\mathbf{b}^1) = b^1_j$ for all gates, and $g_j(\mathbf{b}^1) = 0$ for output gates. For each gate $\mathcal{G}_j$, the value $g_j(\mathbf{v}^1_\alpha) = g_j(\mathbf{a}^1\alpha + \mathbf{b}^1)$ is a quadratic polynomial in α, with coefficients

$$g_j(\mathbf{v}^1_\alpha) = f_j(\mathbf{a}^1)\alpha^2 + \left(\frac{dg_j}{dz}(\mathbf{a}^1 z + \mathbf{b}^1)(0)\right)\alpha + g_j(\mathbf{b}^1),$$

where the linear and constant terms are derived from the Taylor series expansion, and, by definition, f_j is the degree 2 part of g_j. For all gates we have

$$g_j(\mathbf{v}^1_\alpha) - v^2_{\alpha,j}\alpha - v^1_{\alpha,j} = \left(\frac{dg_j}{dz}(\mathbf{a}^1 z + \mathbf{b}^1)(0) - b^2_j - a^1_j\right)\alpha = x^1_j\alpha,$$

which does indeed have a constant term equal to zero, taking $v^1_{\alpha,j} = a^1_j = b^1_j = 0$ for output gates, and so the $\frac{m}{t}$ values sent in the execution of $\mathcal{F}^{1,\mathbb{F}}_{PoLD}$ are accepted as a verification that $\mathbf{x}^1\alpha + \mathbf{x}^0 = \mathbf{x}^1\alpha$.

Perfect Zero Knowledge: The simulator generates α and $\mathbf{v}^2_\alpha$ uniformly at random, matching the behavior of the simulator for certified VOLE. The simulator next generates $\mathbf{v}^1_\alpha$ uniformly at random. This matches exactly the distribution under an honest run of the protocol, since $\mathbf{a}^1$ is chosen uniformly at random by an honest prover. Then the simulator computes the values

$$y_j := g_j(\mathbf{v}^1_\alpha) - v^2_{\alpha,j}\alpha - v^1_{\alpha,j}$$

and generates the the vector $\mathbf{v}^3_\alpha$ using the simulator for $\mathcal{F}^{1,\mathbb{F}}_{PoLD}$ acting on the values y_j.

Soundness: We show the stronger proof-of-knowledge property. For a line $(\hat{\mathbf{a}}, \hat{\mathbf{b}})$ generated by a (potentially malicious) prover, we give an efficient extractor $E(\hat{\mathbf{a}}, \hat{\mathbf{b}})$ that extracts the witness $\hat{\mathbf{w}}$. In fact, we have $\hat{\mathbf{w}} := (\hat{a}^1_1, \ldots, \hat{a}^1_k)$, i.e. the extractor reads off the first k elements of $\hat{\mathbf{a}}^1$.

By the definition of certified LPZK, we require that even a maliciously chosen $(\hat{\mathbf{a}}, \hat{\mathbf{b}})$ satisfy the qVOLE property. Suppose V accepts $\alpha\hat{\mathbf{a}} + \hat{\mathbf{b}}$ and $C(\hat{\mathbf{w}}) \neq \mathbf{0}$. Then $g_j(\hat{b}^1) \neq \hat{b}^j_1$ for some gate. If P cheats on some non-output gate, we have

$$\begin{aligned} g_j(\hat{\mathbf{v}}^1_\alpha) - \hat{v}^2_{\alpha,j}\alpha - \hat{v}^1_{\alpha,j} &= \left(\frac{dg_j}{dz}(\hat{\mathbf{a}}^1 z + \hat{\mathbf{b}}^1)(0) - \hat{b}^2_j - \hat{a}^1_j\right)\alpha + g_j(\hat{\mathbf{b}}^1) - \hat{b}^1_j \\ &= x^1_j\alpha + x^0_j, \end{aligned}$$

for $x^0_j \neq 0$. The verifier will detect that $\mathbf{x}^0 \neq \mathbf{0}$ unless the prover successfully cheats during the execution of the $\mathcal{F}^{1,\mathbb{F}}_{PoLD}$ functionality, which adds $2t/|\mathbb{F}|$ to the soundness error. Finally, for the prover to cheat on any output gate, they must have a vector $\mathbf{b}^1$ such that the output gate $g_j(\mathbf{b}^1) \neq 0$, but send a value y_j such that

$$(x^1_j - y_j)\alpha + g_j(\mathbf{b}^1) = 0,$$

which is equivalent to guessing α, and adds $1/|\mathbb{F}|$ to the soundness error.

A.3 Proof of Thereom 4.1

Completeness: If P has a valid witness $\mathbf{w}$ for C and follows the protocol, we have $g_j(\mathbf{b}^1) = b^1_j$ for all gates, and $g_j(\mathbf{b}^1) = 0$ for output gates.

For each gate $\mathcal{G}_j$ with $j \in I_\tau$, we have

$$\begin{aligned} g_j(\mathbf{v}^4_\alpha) - v^1_{\alpha,j} &= f_j(\mathbf{a}^4)\alpha^2 + \left(\frac{dg_j}{dz}(\mathbf{a}^4 z + \mathbf{b}^4)(0) - a^1_j\right)\alpha + g_j(\mathbf{b}^4) - b^1_j \\ &= x^1_j\alpha^2 + x^0_j\alpha \\ &= y_j\alpha, \end{aligned}$$

as desired. For each gate $\mathcal{G}_j$ with $j \in I_{1-\tau}$, we have

$$\begin{aligned} g_j(\mathbf{v}^1_\alpha) - \alpha v^2_{\alpha,j} &= \left(f_j(\mathbf{a}^1) - a^2_j\right)\alpha^2 + \left(\frac{dg_j}{dz}(\mathbf{a}^1 z + \mathbf{b}^1)(0) - b^2_j\right)\alpha + g_j(\mathbf{b}^1) \\ &= a^4_j\alpha + b_j, \end{aligned}$$

as desired. Finally, for each output gate $\mathcal{G}_j$, we have

$$\begin{aligned} g_j(\mathbf{v}_\alpha) &= f_j(\mathbf{a}^1)\alpha^2 + \left(\frac{dg_j}{dz}(\mathbf{a}^1 z + \mathbf{b}^1)(0)\right)\alpha + g_j(\mathbf{b}^1) \\ &= x^1_j\alpha^2 + x^0_j\alpha \\ &= y_j\alpha, \end{aligned}$$

since $g_j(\mathbf{b}) = 0$. Therefore the $2r$ values sent in the execution of $\mathcal{F}^{k,\mathbb{F}}_{PoPD}$ are accepted as verification that $\mathbf{x}^1\alpha + \mathbf{x}^0 = \mathbf{y}\alpha$.

Zero Knowledge: The simulator generates α and $\mathbf{v}^2_\alpha$ uniformly at random, matching the behavior of the simulator for certified VOLE. The simulator next generates $\mathbf{v}^1_\alpha$ uniformly at random.

This matches exactly the distribution under an honest run of the protocol, since $\mathbf{a}^1$ is chosen uniformly at random by an honest prover. The verifier computes $\mathbf{v}_\alpha^4$ and $\mathbf{y}$ as in an honest run of the protocol, and computes $\mathbf{v}_\alpha^3$ from the simulator of $\mathcal{F}_{PoPD}^{k,\mathbb{F}}$.

Soundness: We show the stronger proof-of-knowledge property. For a line $(\hat{\mathbf{a}}, \hat{\mathbf{b}})$ generated by a (potentially malicious) prover, we give an efficient extractor $E(\hat{\mathbf{a}}, \hat{\mathbf{b}})$ that extracts the witness $\hat{\mathbf{w}}$. As above, we have $\hat{\mathbf{w}} := (\hat{a}_1^1, \ldots, \hat{a}_k^1)$, i.e. the extractor reads off the first k elements of $\hat{\mathbf{a}}^1$.

By the definition of certified LPZK, we require that even a maliciously chosen $(\hat{\mathbf{a}}, \hat{\mathbf{b}})$ satisfy the qVOLE property.

Suppose V accepts $\alpha\hat{\mathbf{a}} + \hat{\mathbf{b}}$ and $C(\hat{\mathbf{w}}) \neq \mathbf{0}$. Then either $g_j(\hat{b}^1) \neq \hat{b}_1^j$ for some gate $\mathcal{G}_j$ in I_τ or $I_{1-\tau}$, or $g_j(\hat{b}^1) \neq 0$ for some output gate. If P cheats on some gate $\mathcal{G}_j \in I_\tau$, we have

$$g_j(\hat{\mathbf{v}}_\alpha^4) - \hat{v}_{\alpha,j}^1 = x_j^1\alpha^2 + x_j^0\alpha + g_j(\hat{\mathbf{b}}^4) - \hat{b}_j^1,$$

which will be detected unless P successfully cheats in the execution of the $\mathcal{F}_{PoPD}^{k,\mathbb{F}}$ functionality. The prover cannot cheat on gates $\mathcal{G}_j \in I_{1-\tau}$, since V computes $\hat{\mathbf{v}}_{\alpha,j}^4$ without any communication from P. Finally, for output gates, if P cheats we have

$$g_j(\hat{\mathbf{v}}_\alpha) = \hat{x}_j^1\alpha^2 + \hat{x}_j^0\alpha + g_j(\hat{\mathbf{b}}),$$

which again will be detected unless P successfully cheats in the execution of the $\mathcal{F}_{PoPD}^{k,\mathbb{F}}$ functionality. Therefore the total soundness error is equal to the soundness error of $\mathcal{F}_{PoPD}^{k,\mathbb{F}}$.