Line-Point Zero Knowledge and Its Applications

Samuel Dittmer* Yuval Ishai† Rafail Ostrovsky‡

November 16, 2020

Abstract

We introduce and study a simple kind of proof systems called *line-point zero knowledge* (LPZK). In an LPZK proof, the prover encodes the witness as an affine line $\mathbf{v}(t) := \mathbf{a}t + \mathbf{b}$ in a vector space $\mathbb{F}^n$, and the verifier queries the line at a single random point $t = \alpha$. LPZK is motivated by recent practical protocols for *vector oblivious linear evaluation* (VOLE), which can be used to compile LPZK proof systems into lightweight designated-verifier NIZK protocols.

We construct LPZK systems for proving satisfiability of arithmetic circuits with attractive efficiency features. These give rise to designated-verifier NIZK protocols that require only 2-3 times the computation of evaluating the circuit in the clear (following a "silent" preprocessing phase), and where the prover communicates roughly 2 field elements per multiplication gate, or roughly 1 element in the random oracle model with a modestly higher computation cost. On the theoretical side, our LPZK systems give rise to the first *linear interactive proofs* (Bitansky et al., TCC 2013) that are zero knowledge against a malicious verifier.

We then apply LPZK towards simplifying and improving recent constructions of *reusable non-interactive secure computation* (NISC) from VOLE (Chase et al., Crypto 2019). As an application, we give concretely efficient and reusable NISC protocols over VOLE for *bounded inner product*, where the sender's input vector should have a bounded L_2-norm.

1 Introduction

Zero-knowledge proofs, introduced by Goldwasser, Micali, and Rackoff [19] in the 1980s, are commonly viewed as a gem of theoretical computer science. For many years, they were indeed confined to the theory domain. However, in the past few years we have seen explosive growth in research on concretely efficient zero-knowledge proof systems. This research is motivated by a variety of real-world applications. See [1] for relevant pointers.

Designated-verifier NIZK. There are many different kinds of zero-knowledge proof systems. Here we mainly consider the setting of *designated-verifier, non-interactive* zero knowledge (dv-NIZK), where the proof consists of a single message from the prover to the verifier, but verification requires a secret verification key that is known only to the verifier and is determined during a (reusable) setup phase. Moreover, we consider by default computationally sound proofs, also known as *arguments*. Designated-verifier NIZK has a rich history starting from [27]; see [30, 28, 14] and references therein for recent works in the area. We will typically consider a more restrictive setting, sometimes referred to as *preprocessing NIZK*, where also the prover needs to hold secret information. In this variant of dv-NIZK the prover and the verifier engage in a (typically inexpensive

*Stealth Software Technologies Inc. samuel.dittmer@gmail.com

†Department of Computer Science, Technion yuval.ishai@gmail.com

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

and reusable) interaction during an offline preprocessing phase, before the inputs are known. In the end of the interaction the prover and the verifier obtain *correlated secret randomness* that is consumed by an online protocol in which the prover can prove multiple statements to the verifier. While this preprocessing model will be our default model for NIZK, our results are relevant to both kinds of dv-NIZK.

Efficiency of proof systems. We are primarily motivated by the goal of improving the *efficiency* of zero-knowledge proofs. There are several different metrics for measuring efficiency of proof systems. Much of the research in this area focuses on improving *succinctness*, which refers both to the proof length and to the verifier's running time. This is highly relevant to the case of publicly verifiable proofs that are generated once and verified many times. However, in the case of a proof that is verified once by a designated verifier, other complexity metrics such as prover's running time and space, can become the main performance bottlenecks. Indeed, state-of-the-art succinct proof systems, such as zk-SNARKs based on pairings [20] or IOPs [5], typically incur high concrete prover computation costs when scaled to large verification tasks. Moreover, they require a big amount of space, and are not compatible with a "streaming" mode of operation in which the proof is generated on the fly together with the computation being verified. On the other hand, non-succinct or semi-succinct proof systems based on the "MPC-in-the-head" [24, 17, 13, 25], garbled circuits [16, 21], or interactive proofs [18, 33, 35], scale better to big verification tasks.

Minimizing prover complexity. Our goal is to push the advantages of non-succinct zero-knowledge proof systems to an extreme, focusing mainly on optimizing the *prover's computation*. This can be motivated by settings in which the prover and the verifier are connected via a fast local network. An extreme case is that of interlocked devices, for which the distinction between computation and communication is blurred. Alternatively, one can think of scenarios in which the proofs can be generated and stored *offline* on the prover side and only verified at a later point, or possibly not at all. Another setting of interest is one where the *statement* is short but is kept secret from the verifier, who cannot perform the (simple) computation on its own. In this setting, which will come up later in this work, the concrete overhead of "asymptotically succinct" systems will be too high. Finally, if the witness is secret-shared between multiple provers and the proof needs to be generated in a distributed way, the prover's computation is likely to become a bottleneck. All of the above reasons motivate a systematic study of minimizing the prover's complexity in zero-knowledge proofs.

Achieving constant computational overhead. We consider the goal of zero-knowledge proofs with *constant computational overhead*, namely where the total computational cost (and in particular the prover's computation) is only a constant times bigger than the cost of performing the verification in the clear. In the case of proving the satisfiability of a Boolean circuit, this question is still open, and the best computational overhead is polylogarithmic in a statistical security parameter [15]. However, when considering arithmetic circuits over a big finite field $\mathbb{F}$ and settling for $O(1/|\mathbb{F}|)$ soundness error, this goal becomes much easier. The first such proof system was given by Bootle et al. [7], who also achieved "semi-succinctness." However, the underlying multiplicative constants are very big, and this system is not considered practical. A more practical approach uses variants of the GKR interactive proofs protocol [33, 35, 34]. Here the concrete computational overhead is smaller, but still quite big: roughly 20x overhead in the best-case scenario of "layered" arithmetic circuits. On top of that, this overhead is only relevant when the verification circuit is much bigger than the witness size. In some of the applications we consider (such as the NISC application discussed

below), this will not be the case.

A third approach, which is most relevant to our work, relies on *oblivious linear evaluation* (OLE) and its vector variant (VOLE). An OLE is an arithmetic variant of oblivious transfer, allowing the receiver, on input α, to learn a linear combination $a\alpha + b$ of two ring elements held by the sender. VOLE is a natural vector analogue of OLE: the receiver learns $\mathbf{a}\alpha + \mathbf{b}$ for a pair of vectors $\mathbf{a}, \mathbf{b}$ held by the sender. The idea of using *random* precomputed instances of OLE and VOLE towards zero-knowledge proofs with constant computational overhead was suggested in [8, 14]. This is motivated by recent techniques for securely realizing (V)OLE at a low amortized cost [3, 8, 9, 31, 10, 12]. However, these protocols for zero knowledge from (V)OLE still suffered from a high concrete overhead. For instance, the protocol from [14] requires 44 instances of OLE for each multiplication gate. A recent and concurrent work by Weng et al [32] improved this state of affairs. We will discuss this work in Section 1.5 below.

1.1 Our contribution

Motivated by the goal of minimizing prover complexity in zero-knowledge proofs, we introduce and study a simple kind of proof systems called *line-point zero knowledge*. We then apply this proof system towards obtaining simple, concretely efficient, and reusable protocols for *non-interactive secure computation*. We elaborate on these results below.

Line-point zero knowledge. A recent work of Boyle et al. [8], with improvements in [9, 31], has shown how to securely generate a long, pseudorandom instance of a vector oblivious linear evaluation (VOLE) correlation with low communication complexity (sublinear in the vector length) and good concrete efficiency. Here we show how to use this for implementing simple and efficient dv-NIZK protocols for circuit satisfiability, improving over similar protocols from [8, 14]. In particular, previous protocols involve multiple VOLE instances and have a large (constant) overhead in communication and computation compared to the circuit size.

The key new tool we introduce is a simple kind of information-theoretic proof system that we call *line point zero knowledge* (LPZK). In an LPZK proof, the prover P generates from the witness w (a satisfying assignment) an affine line $\mathbf{v}(t) := \mathbf{a}t + \mathbf{b}$ in an n-dimensional vector space $\mathbb{F}^n$. The verifier queries a single point $\mathbf{v}(\alpha) = \mathbf{a}\alpha + \mathbf{b}$ on this line, and determines whether to accept or reject. We call this proof system LPZK over $\mathbb{F}$ with length parameters n, n', and n'' and soundness error ε. The length parameter n denotes that $\mathbf{a}$, $\mathbf{b} \in \mathbb{F}^n$, and n' and n'' are additional parameters defined in Section 2 related to efficiency optimizations. In the constructions in this paper, we generally use $\varepsilon = O(1/|\mathbb{F}|)$. We define this proof system formally and explain how to compile any LPZK into a designated-verifier NIZK protocol over a random VOLE in Section 2.

As an information-theoretic proof system, LPZK can be viewed as a simple instance of (1-round), zero-knowledge *linear interactive proof* (LIP) [6], in which the verifier sends a single field element to the prover. Our LPZK consruction gives the first such system that is zero-knowledge even agianst a malicious verifier.

LPZK constructions. We start by stating our main result on the feaibility and efficiency of LPZK.

Theorem 1.1 (LPZK for arithmetic circuit satisfiability)**.** *For any NP-relation $R(x, y)$ and field $\mathbb{F}$, there exists an 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 $t \geq 1$. If C has k*

inputs, k' outputs, and m multiplication gates, we have $n = k + k' + (2 + \frac{1}{t})m$, $n' = k + 2m$, $n'' = \frac{m}{t} + k'$, $\varepsilon = 2t/|\mathbb{F}|$. Moreover, assuming that the cost of additions in the field are negligible compared to the cost of multiplications, the local computation of the prover is less than 3 times the cost of evaluation in the clear, and the local computation of the verifier is less than 2 times the cost of evaluation in the clear.

Our LPZK construction has the following direct application to dv-NIZK in the *rVOLE-hybrid model*, where in a trusted setup, the prover P receives a random pair of vectors $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$, while the verifier V receives a random field element $\alpha \in \mathbb{F}$ and the vector $\alpha\mathbf{a} + \mathbf{b}$.

Corollary 1.2 (Designated-verifier NIZK over random VOLE)**.** *Fix an integer $t \geq 1$. There exists an (unconditional) NIZK protocol in the rVOLE-hybrid model 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 local computation and soundness error as above in the LPZK case, and with:*

- *Communication: $k + k' + (2 + \frac{1}{t})m$ field elements from P to V;*
- *rVOLE length: $n = k + 2m$ over $\mathbb{F}$.*

1.2 Improving proof size in the random oracle model

Inspired by the concurrent[1] work of Weng et al. [32], we can improve the communication cost of our proofs in the random oracle model by a factor of 2 (asymptotically) at the cost of a modest increase of prover and verifier computation, in the form of calls to a cryptographic hash function. Note that other attractive features of LPZK such as space- and streaming-friendliness are maintained. See § 1.5 below for a detailed comparison between the results of [32] and our work.

Theorem 1.3. *Fix an integer $r \geq 1$. There exists an (unconditional) NIZK protocol in the RO-rVOLE-hybrid model that proves the satisfiability of an 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 $\frac{2}{|\mathbb{F}|} + \frac{\ell}{|\mathbb{F}|^r}$;*
- *Communication of $k + k' + m + 2r$ field elements from P to V;*
- *Local computation of $O(r|C|)$ field operations and $O(1)$ cryptographic hash calls (from $\mathbb{F}^m \to \mathbb{F}^{mr}$) for both the prover and the verifier;*
- *rVOLE length $n = k + m + r$ over $\mathbb{F}$.*

1.3 Reusable NISC from LPZK via certified VOLE

A *non-interactive secure computation* (NISC) protocol [23] is a two-party protocol that securely computes a function $f(x, y)$ using two messages: a message by a *receiver*, encrypting its input x, followed by a message by a *sender*, that depends on its input y. The output $f(x, y)$ is only revealed to the receiver. A major challenge is making such protocols secure even when both parties can be malicious. Another challenge is to make such protocols *reusable*, in the sense that the same encrypted input x can be used to perform computations with many sender inputs y_i without violating security. This should hold even when a malicious sender can learn partial information about

[1] Most of the present work was done concurrently and independently of [32]. We explicitly point out the improvements that are based on ideas from [32].

the honest receiver's output, such as whether the receiver "aborts" after detecting an inconsistent sender behavior. Existing NISC (or even NIZK) protocols based on parallel calls to oblivious transfer (OT) and symmetric cryptography [27, 23, 2, 29] are *not* fully reusable, and this is in some sense inherent [14].

Chase et al. [14] recently showed how to realize reusable NISC by using parallel instances of VOLE instead of OT. Here reusability refers to fixing the VOLE inputs of the receiver, which are picked once and for all based on its NISC input x. On top of the reusability feature, another advantage of the VOLE-based protocol, which is inherited from earlier protocols with security against semi-honest senders [22, 4], is that it "natively" supports simple *arithmetic* computations over the VOLE field. This is contrasted with NISC protocols over OT [23, 2, 29], which apply to Boolean circuits.

We provide an alternative construction of reusable NISC over VOLE. Our approach uses LPZK to significantly simplify the protocol from [14] and results in much better concrete constants.

NISC for bounded inner product. To illustrate the concrete efficiency potential of our approach, we showcase it for a simple and useful application scenario. Consider an "inner product" functionality that measures the level of similarity (or correlation) between receiver feature vector x and a sender feature vector y, where the same x can be reused with multiple sender inputs y_i. Here we view both x and y as integer vectors that are embedded in a sufficiently large finite field. An obvious problem is that the ideal functionality allows a malicious sender to scale its real input by an arbitrary multiplicative factor, thereby increasing the perceived similarity. To prevent this attack, we modify the functionality to bound the L_2 norm of the sender's input. In this way, the sender's strategy is effectively restricted to choosing the direction of a unit vector, where the bound on the norm determines the level of precision. For this *bounded inner product* functionality, we obtain a concretely efficient protocol that offers reusable malicious security. Even when considering malicious security alone, without reusability, previous techniques for NISC are much less efficient for such simple arithmetic functionalities. To give just one data point, for vectors length 1000, sender L_2 norm bounded by 2^{80} and the entries of the sender's vector each bounded by 1024, our protocol requires 1002 instances of VOLE of total length 19,000 field elements (roughly 190 kB given given random VOLE setup) and additional communication of 33,000 field elements (roughly 330 kB). Given recent methods for "silent" generation of multiple VOLE instances [9, 31, 10, 12], the amortized cost of setting up the required VOLE instances is small.

1.4 Overview of techniques

From LPZK to NIZK via random VOLE. An LPZK proof system can be directly realized by a single instance of VOLE, where the prover's line $\mathbf{v}(t) := \mathbf{a}t + \mathbf{b} \in \mathbb{F}^n$ determines the VOLE sender's input $(\mathbf{a}, \mathbf{b})$ and the verifier's point α is used as the VOLE receiver's input. A further observation is that this single VOLE instance can be easily reduced to a *random* VOLE functionality that assigns to the prover a uniformly random pair of vectors $(\mathbf{a}', \mathbf{b}')$ each in $\mathbb{F}^n$ and to the verifier a uniformly random value $\alpha \in \mathbb{F}$ and $\mathbf{v}' = \mathbf{a}'\alpha + \mathbf{b}'$. Indeed, the prover can send $(\mathbf{a} - \mathbf{a}')$ and $(\mathbf{b} - \mathbf{b}')$ to the verifier, who computes $\mathbf{v}(\alpha) = \mathbf{v}' + (\mathbf{a} - \mathbf{a}')\alpha + (\mathbf{b} - \mathbf{b}')$. This requires communication of $2n$ field elements on top of the pre-processing step required to set up the random VOLE instance. Combined with efficient protocols for generating long instances of random VOLE, this gives rise to dv-NIZK protocol in which the offline phase consists of secure generation of random VOLE and the online phase uses the random VOLE as a "one-time pad" for LPZK.

Constructing information-theoretic LPZK proofs. At a high level, we construct LPZK for proving the satisfiability of an arithmetic circuit C by encoding intermediate wire values in the vector $\mathbf{a}$ and masking these values with randomness in $\mathbf{b}$. This is an information-theoretic encryption, i.e. if the verifier holds $v_1(\alpha) := a_1\alpha + b_1$ and α, where a_1 is sampled from some distribution and b_1 is chosen uniformly at random from $\mathbb{F}$, the distribution of $v_1(\alpha)$ holds no information about the distribution of a_1.

We can "add" two encrypted wires $v_1(t) = a_1t + b_1$ and $v_2(t) = a_2t + b_2$ non-interactively for free; the prover adds to obtain $(a_1 + a_2)t + (b_1 + b_2)$, and the verifier adds $v_1(\alpha) + v_2(\alpha) = (a_1 + a_2)\alpha + (b_1 + b_2)$.

To multiply v_1 and v_2, the prover seeks to construct the encrypted wire a_1a_2t+b, for some value b. When the prover multiplies $v_1(t) \cdot v_2(t)$ they obtain a quadratic in t. By adding and subtracting a masking term b_3t, they can write $v_1(t)v_2(t) = tv_3(t) + v_4(t)$, with $v_3(t) = (v_1(t)v_2(t) - v_4(t))/t$ the desired encryption of a_1a_2. The verifier then computes

$$v_3(\alpha) = \frac{v_1(\alpha)v_2(\alpha) - v_4(\alpha)}{\alpha}.$$

Finally, to open the value of an encrypted wire $v(t) = at + b$, the prover sends b to the verifier who computes $a = (v(\alpha) - b)/\alpha$.

LPZK-NIZK optimizations. There are two optimizations in the compiler from random VOLE to LPZK we can make based on the desired structure of the LPZK to reduce communication costs. When the LPZK only requires an entry of $\mathbf{a}$ or $\mathbf{b}$ to be chosen uniformly at random over $\mathbb{F}$, independently of previous entries, we can leave the corresponding element of $\mathbf{a}'$ or $\mathbf{b}'$ unchanged, reducing communication costs by one. And, when the LPZK requires an entry of $\mathbf{a}$ to be equal to zero, we can instead send the corresponding entry of $\mathbf{b}$ in the clear, shortening our random VOLE length by one and reducing communication costs by one.

In our length parameters (n, n', n''), we denote by n' the number of entries in $\mathbf{a}$ and $\mathbf{b}$ where the first and second optimization above do *not* apply, and by n'' the number of entries where the second optimization *does* apply.

We perform additional optimizations for the general NIZK construction by batching together tests for multiple gates at once to reduce communication costs. We batch in blocks of t gates, giving an amortized cost of $2 + \frac{1}{t}$ field elements of communication per gate and increasing the soundness error by a factor of t. In the random oracle setting, we batch together all multiplication gates into a single block, giving an amortized cost of 1 field elements per gate plus an additional $2r$ field elements total, where r is some fixed parameter. This approach is theoretically vulnerable to a malicious prover who makes repeated calls to the random oracle until they find a collision. However, when we choose parameters r and $|\mathbb{F}|$ appropriately, we can bound soundness error reasonably, as we show in Section 5.

Certified VOLE. As a building block for NISC, we build a family of *certified VOLEs*, where we prove that a given circuit C is satisfied when its inputs are a certain subset of the entries of the VOLEs.

We construct fully general certified VOLE from weaker versions. We obtain general certified VOLE by combining *VOLE with equality constraints*, where the circuit C is made up entirely of equality conditions, with our LPZK NIZK construction by moving all inputs to C to a single VOLE instance and then proving that C is satisfied.

We obtain VOLE with equality constraints from a *weak* and *strong* variant of *distributional certified VOLE with equality constraints*. In weak distributional VOLE, we prove equality constraints

on two VOLEs whose evaluation points α_1, α_2 are each uniformly distributed, but are correlated such that $\alpha_1 - \alpha_2$ is a fixed constant. In strong distributional VOLE, we prove equality constraints on two VOLEs whose evaluation points α_1, α_2 are uniformly distributed and independent.

To construct general VOLE with equality constraints, we add two additional instances of VOLE. The first new instance's evaluation point α is used as an offset for the other instances of VOLE. The second new instance's evaluation point β is chosen uniformly at random and independent of all other values. We use weak distributional VOLE to move all equality constraints on $\mathbf{a}_i$'s to the first new instance of VOLE, and then use strong distributional VOLE twice, to move equality constraints on b_i's to the second new instance, and then from the second instance to the first.

From certified VOLE to NISC. Following [14], our NISC protocol is obtained from certified VOLE in a conceptually straightforward way: we start with existing protocols for arithmetic branching programs [22, 4] that achieve security against a malicious receiver and *semi-honest sender*. We then protect the receiver against a malicious sender by using certified VOLE to enforce honest behavior. This yields a statistically secure reusable NISC protocol for "simple" arithmetic functions represented by polynomial-size arithmetic branching programs. We can bootstrap this to get reusable NISC over VOLE for general Boolean circuits using the approach of [14]; however, this comes at the cost of making a non-black-box use of a pseudorandom generator and losing the concrete efficiency features of the arithmetic variant of the protocol.

1.5 Comparison with [32]

In concurrent work, Weng, Yang, Katz and Wang [32] design and implement a concretely efficient VOLE-based ZK protocol with an online phase that can be made non-interactive in the random oracle model. While their design principle shares similar high-level features to our LPZK-based design, there are several differences beyond the different abstraction.

In comparison with [32], our NIZK protocol requires 2 times less communication while offering the possibility of entirely eliminating the "cryptographic" overhead of the online phase. Computationally, the protocol from [32] requires more multiplications per gate and additionally requires 3 invocations of a cryptographic hash function (instantiating the random oracle). The protocol is at least 3 times more expensive computationally than the basic version of our protocol, where the gap can be much bigger when the hash function (such as SHA256) is more costly than a single multiplication. It is possible to apply the random oracle model optimization to our protocol as well, getting an additional improvement in communication, at a cost in computation, as we show in Theorem 5.1. It is also possible to remove the random oracle model from [32] using our ideas. In all measures, our protocol is faster and has better communication. We give additional details below.

Weng et al. [32] use subfield VOLE and other optimizations to improve the cost of soundness amplification when working over small fields. Similar techniques could be applied to our approach, but we chose here to focus on the simpler case of arithmetic circuits over large fields. (Alternatively, one can repeat a proof over a smaller field to amplify soundness, though this will typically eliminate the concrete efficiency benefits of our protocol.) We note that in the context of proofs, mixing Boolean and arithmetic operations is easier than in a typical secure computation setting, since the prover can provide a bit-decomposition of an arithmetic value which can be easily verified by an arithmetic circuit.

Communication:

- Random oracle model: [32] requires $2 + o(1)$ elements of communication per gate, while our approach requires $(1+o(1))$ elements per gate. A variant in [32] that is more computationally efficient requires r elements per gate.
- Without random oracle model: [32] would require $6 + o(1)$ elements per gate, while ours requires $2 + 1/t$.

Prover computation:

- Random oracle model: The more computationally efficient variant in [32] requires $(3r + 8)$ multiplications per gate + 3 cryptographic hash calls, while our approach requires $(2r + 3)$ multiplications and a total of 2 cryptographic hash calls for the entire proof.
- Without random oracle model: [32] requires 5 multiplications per gate, while our approach requires 3 multiplications per gate.

Verifier computation:

- random oracle model: The more computationally efficient variant in [32] requires $(3r + 11)$ multiplications per gate + 3 cryptographic hash calls, while our approach requires $(r + 2)$ multiplications and a single cryptographic hash call for the entire proof.
- Without random oracle model: [32] requires 7 multiplications per gate + while our approach requires 2 multiplications per gate.

2 LPZK and random VOLE

In this section we give a formal definition of our new notion of LPZK proof system and show how to compile such a proof system into a designated-verifier NIZK when given a random VOLE correlation.

2.1 Defining LPZK

While an LPZK proof system can be defined for any NP-relation, we focus here on the case of arithmetic circuit satisfiability that we use for describing our constructions. Our definition can be seen as a simple restriction of the more general notion of (1-round) zero-knowledge *linear interactive proof* [6] that restricts the verifier to sending a single field element.

Definition 2.1 (LPZK)**.** A *line-point zero-knowledge* (LPZK) proof system for arithmetic circuit satisfiability is a pair of algorithms (Prove, Verify) with the following syntax:

- $\mathsf{Prove}(\mathbb{F}, C, w)$ 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 $\mathbf{v}(t) := \mathbf{a}t + \mathbf{b}$. We assume that the dimension n is determined by C.
- $\mathsf{Verify}(\mathbb{F}, C, \alpha, \mathbf{v}_\alpha)$ 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.

The algorithms (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 $C(w) = \mathbf{0}$, and for any fixed $\alpha \in \mathbb{F}$, we have $\Pr[\mathbf{v}_\alpha \xleftarrow{R} \mathsf{Prove}(\mathbb{F}, C, w) : \mathsf{Verify}(\mathbf{F}, C, \alpha, \mathbf{v}_\alpha) = \mathsf{acc}] = 1$.

- **Reusable ε-soundness**. For every arithmetic circuit $C : \mathbb{F}^k \to \mathbb{F}^{k'}$ such that $C(w) \neq \mathbf{0}$ for all $w \in \mathbb{F}^k$, 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, we have $\Pr[\alpha \xleftarrow{R} \mathbb{F} : \mathsf{Verify}(\mathbb{F}, C, \alpha, \mathbf{v}^*(\alpha)) = \mathsf{acc}] \leq \varepsilon$. 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 $C(w) = \mathbf{0}$, and any $\alpha \in \mathbb{F}$, the output of $\mathsf{Sim}(\mathbb{F}, C, \alpha)$ is a vector $\mathbf{v}$ such that $\{\mathbf{v} : \mathbf{v} \xleftarrow{R} \mathsf{Sim}(\mathbb{F}, C)\}$ and $\{\mathbf{v}(\alpha) : \mathbf{v}(t) \xleftarrow{R} \mathsf{Prove}(\mathbb{F}, C, w)\}$ are identically distributed.

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. While we ignore here the additional *proof of knowledge* property for simplicity, the LPZK systems we construct all satisfy this stronger notion of soundness (see [6] for a formalization in the context of linear proof systems).

Complexity measures for LPZK. In addition to the dimension/length parameter n, we use two other parameters n' and n'' as complexity measures for LPZK. These will help us obtain a more efficient compiler from LPZK to NIZK that takes advantage of receiver outputs that are either known by the prover (namely, are independent of α) or entries of $\mathbf{a}, \mathbf{b}$ that can be picked at random independently of w. Concretely, the parameter n'' is the number of entries of $\mathbf{a}$ that are always equal to zero; we assume without loss of generality that these are the last n'' entries. The parameter n' measures the total number of entries of the first $n - n''$ entries of $\mathbf{a}$ and $\mathbf{b}$ that functionally depend on w. That is, we assume that the remaining $2n - 2n'' - n'$ entries are picked uniformly and independently at random, and then these n' entries are determined by w and the random entries. We will assume that the parameters (n, n', n'') as well as the identity of the entries of each type are determined by the public information C.

2.2 Compiling LPZK to NIZK over random VOLE

We now describe and analyze a simple compiler that takes an LPZK proof system as defined above and converts it into a (designated verifier) NIZK protocol that relies on a *random VOLE* correlation, where the prover gets a *random* pair of vectors $\mathbf{a}', \mathbf{b}' \in \mathbb{F}^n$ specifying an affine line $\mathbf{a}'t + \mathbf{b}'$ in $\mathbb{F}^n$ and the verifier gets the value of the line at a random point $\alpha \in \mathbb{F}$, namely $\mathbf{v}' = \mathbf{a}'\alpha + \mathbf{b}'$. Similarly to previous VOLE-based compilers from [8, 14], we rely on the simple known reduction from VOLE to random VOLE. Our compiler takes advantage of the extra parameters n' and n'' of the LPZK, which help reduce the cost of the NIZK below the $2n$ field elements communicated by the natural generic compiler.

Lemma 2.1. *We can realize (n, n', n'')-LPZK over $\mathbb{F}$ with soundness error ε by using a single instance of random VOLE of length $n - n''$ and additional communication of $n' + n''$ field elements.*

Proof. Let $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$ be the vectors for the prover's line $\mathbf{a}t + \mathbf{b}$. The prover and verifier generate a random VOLE of length n, so that the prover holds $(\mathbf{a}', \mathbf{b}')$, and the verifier holds $\mathbf{v}' = \mathbf{a}'\alpha + \mathbf{b}'$.

We recall a simple self-reduction property of VOLE (see e.g. [8]) that allows us to replace a random pair $(\mathbf{a}', \mathbf{b}')$ with the pair $(\mathbf{a}, \mathbf{b})$ as follows. The prover sends vectors $\mathbf{a}' - \mathbf{a}$ and $\mathbf{b}' - \mathbf{b}$ to

the verifier, who then computes

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

Finally, the prover sends the final n'' values of $\mathbf{b}$ to the verifier in the clear, and the verifier appends these values to $\mathbf{v}$.

For any entry of $\mathbf{a}, \mathbf{b}$ that should be chosen randomly for LPZK, the prover sets the corresponding entry of $\mathbf{a}' - \mathbf{a}$ or $\mathbf{b}' - \mathbf{b}$ to zero, and so no communication is required for those entries. The entire reduction requires a random VOLE of length n with communication of $n' + n''$ field elements, as desired. □

Using a corruptible random VOLE functionality. When using a pseudorandom correlation generator (PCG) for generating the random VOLE correlation [8, 11], what is actually realized is a so-called "corruptible" random VOLE functionality that allows the malicious party to choose its output, and then samples the honest party's output conditioned on this choice. The transformation of Lemma 2.1 remains secure even when using this corruptible VOLE functionality. Indeed, it was already observed in [8] that the reduction of VOLE to random VOLE remains secure even when using corruptible random VOLE, and the LPZK to NIZK transformation builds on this reduction.

3 Single gate example

To clarify the exposition, we begin with an example where the prover wishes to convince the verifier that they hold a triple of values x, y, z satisfying $xy = z$. More precisely, the prover and verifier realize a commit-and-prove functionality for the triple (x, y, z) and the relation $R(x, y, z) := xy - z$. We prove that our single gate example satisfies this stronger notion of ZK. Note that our NIZK construction is adapted from this single gate example, rather than directly built up from it, so this proof and the proof in section 6 can be read independently.

3.1 Protocol

We construct our commit-and-prove protocol for the relation $R(x, y, z) := xy - z$ as a $(4, 4, 1)$-LPZK over $\mathbb{F}$ with soundness error $2/|\mathbb{F}|$.

The (honest) prover chooses some triple (x, y, z) and constructs a line $\mathbf{a}t + \mathbf{b}$ by setting

$$\mathbf{a} = (a_1, a_2, a_3, a_4, a_5) := (x, y, z, xb_2 + yb_1 - b_3, 0)$$

with b_1, b_2, b_3, b_4 chosen uniformly at random and $b_5 := b_1 b_2 - b_4$. We write

$$\mathbf{v}(t) := \mathbf{a}t + \mathbf{b},$$

for the line held by the prover, and $\mathbf{v} = \mathbf{a}\alpha + \mathbf{b}$ for the point received by the verifier, for some random $\alpha \in \mathbb{F}$. We likewise write the prover's view of the entries as

$$\mathbf{v}(t) = (v_1(t), v_2(t), v_3(t), v_4(t), v_5(t)),$$

and write v_i for $v_i(\alpha)$. The verifier now checks whether

$$v_1 v_2 - \alpha v_3 - v_4 - v_5 = 0.$$

3.2 Proof

To prove that this is a commit-and-prove protocol for the relation $R(x, y, z) = xy - z$ we give a deterministic extractor that takes the line $(\mathbf{a}^*, \mathbf{b}^*)$ generated by a malicious prover, and extracts effective inputs $W^* := (x^*, y^*, z^*)$, and must prove the extractor and protocol satisfy completeness, binding, soundness, and zero knowledge (see e.g. [26]). The extractor is simple: it reads off W^* as the first three entries of $\mathbf{a}^*$.

Completeness. If the prover is honest, we have

$$\begin{aligned} v_1 v_2 - \alpha v_3 - v_4 - v_5 = \; & (xy - z)\alpha^2 + (xb_2 + yb_1 - b_3 - (xb_2 + yb_1 - b_3))\alpha \\ & + b_1 b_2 - b_4 - b_5 \\ = \; & 0 \end{aligned}$$

identically, as long as $xy - z = 0$.

Binding. For any $W' \neq W^*$, the verifier's values $\mathbf{a}^*\alpha + \mathbf{b}^*$ are consistent with W' only if $a_i^*\alpha + b_i^* = a_i'\alpha + b_i'$, for $i \in \{1, 2, 3\}$. For any choice $(\mathbf{a}', \mathbf{b}') \neq (\mathbf{a}^*, \mathbf{b}^*)$, the prover can compute a corresponding guess $\alpha^* = (b_i^* - b_i')/(a_i^* - a_i')$. Since α is chosen uniformly at random, independent of the prover, the probability that $\alpha = \alpha^*$ is at most $1/|\mathbb{F}|$.

Soundness.

If the extracted input W^* does not satisfy R, then the expression

$$\begin{aligned} v_1(t)v_2(t) - tv_3(t) - v_4(t) - v_5(t) = \; & (x^*y^* - z^*)t^2 \\ & + \big(xb_2 + yb_1 - b_3 - (xb_2 + yb_1 - b_3)\big)t \\ & + b_1 b_2 - b_4 - b_5 \end{aligned}$$

is a non-trivial polynomial in t of degree 2. This poylnomial is only satisfied if α is one of the at most 2 roots, which gives a soundness error of at most $2/|\mathbb{F}|$.

Zero knowledge. V can simiulate their view by generating v_1, v_2, v_3 and v_5 uniformly at random, and computing $v_4 = v_1 v_2 - \alpha v_3 - v_5$. We know that v_1, v_2, v_3 and v_5 are uniformly random because of the uniform randomness of b_1, b_2, b_3 and b_4, respectively. □

3.3 Complexity

Communication. The communication cost of $(4, 4, 1)$-LPZK is 5 field elements, as explained in Section 2.

Prover computation. 3 multiplications, 2 additions, and 7 subtractions (counting the computation of $xy = z$, and computing $xb_2 + yb_1$ as $(x + b_1)(y + b_2) - z - b_1 b_2$).

Verifier computation. 2 multiplications, 2 subtractions, and one equality test.

4 Information-Theoretic LPZK for Arithmetic Circuit Satisfiability

4.1 Setup

An arithmetic circuit C over a field $\mathbb{F}$ with k input wires, k' output wires, m multiplication gates, and arbitrarily many addition gates can be converted into an ordered triple $(\mathbf{a}, Q_C, R_C)$, where $\mathbf{a} = (a_0, a_1, \ldots, a_{k+k'+4m})$ represent wire values, Q_C is a collection of m degree 2 polynomials, with the ith polynomial defined as

$$q_i(\mathbf{a}) := a_{k+4i-1} - a_{k+4i-3}a_{k+4i-2},$$

and R_C is a set of linear relations defining certain a_i's in terms of previous elements. Formally, we write R_C as $2m + k'$ vectors $\mathbf{r}_i$ corresponding to the relations

$$\mathbf{r}_{2i-j} \cdot \mathbf{a} = a_{k+4i-2-j},$$

for $j \in \{0, 1\}$, and $1 \leq i \leq m$, where the only nonzero entries of $\mathbf{r}_{2i-j}$ occur at indices $\leq k + 4i - 4$, and

$$\mathbf{r}_{2m+i} \cdot \mathbf{a} = 0,$$

for $1 \leq i \leq k'$.

The wires a_{k+4i} are not needed for the insecure evaluation of the circuit, but we introduce them now to keep indices consistent. We require that each of $\mathbf{r}_j$ have zero at each of their entries in positions $k + 4i$, for $1 \leq j \leq 2m + k'$ and $1 \leq i \leq m$, i.e. the relations in R_C cannot depend on the unused a_{k+4i} wires. We set $a_0 = 1$ so that the relations R_C can include addition by constant terms.

We construct a NIZK in this setting. Using a $(k + 2m, k + 2m, \frac{m}{t} + k')$-LPZK with soundness error $t/|\mathbb{F}|$, a prover P will convince a verifier V that they hold a witness $\mathbf{w} = (w_1, \ldots, w_k)$ of circuit inputs to C such that the k' entries $a_{k+4m+i} = 0$, for $1 \leq i \leq k'$. The circuit C and associated data k, k', m and Q are public.

4.2 The LPZK construction

To begin, the prover constructs a pair of vectors $(\mathbf{a}, \mathbf{b}) \in \mathbb{F}^{k+(4+\frac{1}{t})m+2}$, with $a_0 = 0$ and $b_0 = 1$. The next k elements of $\mathbf{a}$ are set equal to the witness $\mathbf{w}$, and the corresponding elements of $\mathbf{b}$ are chosen uniformly at random. Using the relations in R_C, for the ith multiplication gate, and for $j \in \{0, 1\}$, the prover defines

$$a_{k+4i-2-j} := \mathbf{r}_{2i-j} \cdot \mathbf{a}$$

$$b_{k+4i-2-j} := \mathbf{r}_{2i-j} \cdot \mathbf{b}$$

$$a_{k+4i-1} := a_{k+4i-3} a_{k+4i-2}$$

$$a_{k+4i} := a_{k+4i-3} b_{k+4i-2} + a_{k+4i-2} b_{k+4i-3} - b_{k+4i-1},$$

with b_{k+4i-j} chosen uniformly at random, for $j \in \{0, 1\}$. Then, for $1 \leq i \leq k'$, P sets $a_{k+4m+i} = 0$ and

$$b_{k+4m+i} := \mathbf{r}_{2m+i} \cdot \mathbf{b}.$$

Next, P constructs a vector $\mathbf{c}$ of length m and defines

$$c_i := b_{k+4i-3} b_{k+4i-2} - b_{k+4i},$$

if this value is not equal to zero, and $c_i = 1$ otherwise, for $1 \leq i \leq m$. Finally, for $i = 1, \ldots, m/t$, P sets $a_{k+k'+4m+i} = 0$ and defines

$$b_{k+k'+4m+i} := \prod_{j=t*i}^{t*i+t-1} c_j.$$

After constructing $(\mathbf{a}, \mathbf{b})$, the prover constructs a shortened pair of vectors $(\hat{\mathbf{a}}, \hat{\mathbf{b}})$ of length $k + k' + (2 + \frac{1}{t})m + 1$ by deleting the zeroth entry and the entries $k + 4i - 2 - j$, for $1 \leq i \leq m$ and $j \in \{0, 1\}$, and performs LPZK with the verifier so that the verifier learns $\hat{\mathbf{v}} = \alpha \hat{\mathbf{a}} + \hat{\mathbf{b}}$.

The verifier then computes from $\hat{\mathbf{v}}$ a vector $\mathbf{v}$ of length $k + k' + (4 + \frac{1}{t})m + 2$ by re-indexing to match the indexing of $\mathbf{a}$ and $\mathbf{b}$, setting $v_0 = 1$, and computing

$$v_{k+4i-2-j} := \mathbf{r}_{2i-j} \cdot \mathbf{v},$$

for $1 \leq i \leq m$ and $j \in \{0, 1\}$.

Then for $1 \leq i \leq k'$, the verifier checks that $\mathbf{r}_{2m+i} \cdot \mathbf{v} = v_{k+4m+i}$. Finally, the verifier defines, for $1 \leq i \leq m$, the values

$$x_i := v_{k+4i-3}v_{k+4i-2} - \alpha v_{k+4i-1} - v_{k+4i},$$

when this is nonzero, and $x_i := 1$ otherwise, and checks that

$$\prod_{j=t*i}^{t*i+t-1} x_j = v_{k+k'+4m+i}.$$

4.3 Proof of Theorem 1.1

Completeness: If P has a valid witness $\mathbf{w}$ for C and follows the protocol, then, for the output gates, for $1 \leq i \leq k'$, we have

$$\mathbf{r}_{2m+i} \cdot \mathbf{v} = \alpha \mathbf{r}_{2m+i} \cdot \mathbf{a} + \mathbf{r}_{2m+i} \cdot \mathbf{b} = b_{k+4m+i} = v_{k+4m+i}.$$

For the multiplication gates, for $1 \leq i \leq m$, as in Section 3, we have

$$\begin{aligned}(v_{k+4i-3})(v_{k+4i-2}) &= (\alpha a_{4i-3} + b_{4i-3})(\alpha a_{4i-2} + b_{4i-2}) \\ &= \alpha^2 a_{4i-3}a_{4i-2} + \alpha(a_{4i-3}b_{4i-2} + a_{4i-2}b_{4i-3}) + b_{4i-3}b_{4i-2} \\ &= \alpha v_{k+4i-1} + v_{k+4i} + c_i,\end{aligned}$$

when $c_i \neq 0$, and $(v_{k+4i-3})(v_{k+4i-2}) = \alpha v_{k+4i-1} + v_{k+4i}$ otherwise. Thus $c_i = x_i$ for all i, and we have

$$\prod_{j=t*i}^{t*i+t-1} x_j = \prod_{j=t*i}^{t*i+t-1} c_j = v_{k+k'+4m+i},$$

as desired.

Perfect Zero Knowledge: The simulator generates α and $v_1, \ldots, v_k$ uniformly at random from $\mathbb{F}$. Since $v_i = \alpha a_i + b_i$ for $1 \leq i \leq k$ under an honest run of the protocol, and $b_1, \ldots, b_k$ are generated uniformly at random and independently, the v_i's are also uniformly random and independent under an honest run of the protocol. Therefore the distribution produced by the simulator matches the distribution produced by an honest run for $v_1, \ldots, v_k$.

Next, the simulator generates v_{k+4i-j} uniformly at random, for $1 \leq i \leq m$ and $j \in \{0, 1\}$, which again matches exactly the distribution under an honest run of the protocol, by the uniform randomness of the b_{k+4i-j}'s. Then, working from left to right, the simulator computes

$$v_{k+4i-2-j} := \mathbf{r}_{2i-j} \cdot \mathbf{v},$$

for $j \in \{0, 1\}$ and

$$x_i := v_{k+4i-3}v_{k+4i-2} - \alpha v_{k+4i-1} - v_{k+4i}.$$

The simulator then generates

$$v_{k+4m+i} := \mathbf{r}_{2m+i} \cdot \mathbf{v}$$

for $1 \leq i \leq k'$ and

$$v_{k+k'+4m+i} := \prod_{j=t*i}^{t*i+t-1} x_j$$

for $1 \leq i \leq m/t$ and outputs accept.

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 $\mathbf{w}^*$. In fact, we have $\mathbf{w}^* := (a_1^*, \ldots, a_k^*)$, i.e. the extractor reads off the first k elements of $\hat{\mathbf{a}}^*$. As in Section 3, we write $\hat{\mathbf{v}}(t)$ and $\mathbf{v}(t)$ for the lines the prover holds on which the verifier queries the points $\hat{\mathbf{v}}$ and $\mathbf{v}$.

Suppose V accepts $\alpha\hat{\mathbf{a}}^* + \hat{\mathbf{b}}^*$ and $C(\mathbf{w}^*) \neq 1$. Then either $a_{k+4i-1} \neq a_{k+4i-3}a_{k+4i-2}$, for some $i \leq m$, or $\mathbf{a} \cdot \mathbf{r}_{2m+i} \neq 0$, for some $i \leq k'$.

In the first case, the corresponding expression

$$x_i(t) = v_{k+4i-3}(t)v_{k+4i-2}(t) - \alpha v_{k+4i-1}(t) - v_{k+4i}(t)$$

reduces to a nontrivial polynomial of degree 2 over α, and so for some i the expression

$$v_{k+k'+4m+i}(t) - \prod_{j=t*i}^{t*i+t-1} x_j(t)$$

is a nontrivial polynomial over t of degree at least 2 and at most $2t$. This gives a soundness error of at most $2t/|\mathbb{F}|$.

In the second case, we have

$$a_{k+4m+i}t + b_{k+4m+i} = v_{k+4m+i}(t) = \mathbf{v}(t) \cdot \mathbf{r}_{2m+i} = \mathbf{a}(t) \cdot \mathbf{r}_{2m+i}t + \mathbf{b}(t) \cdot \mathbf{r}_{2m+i},$$

which simplifies to a linear polynomial over t, which corresponds to a soundness error of at most $1/|\mathbb{F}|$.

4.4 Complexity

We give complexity bounds for the online stage of the protocol, for a circuit C with k inputs and m multiplication gates.

Let $T(C)$ denote the time to evaluate the addition and scalar multiplication gates of a circuit C in the clear, $T(*)$ the cost of a multiplication and $T(+)$ the cost of an addition, so that the total cost of evaluation in the clear is $T(C) + mT(*)$.

Communication complexity: For the $k + 2m$ one-side-fixed VOLE, we require the communication of $k + 2m$ field elements. Our checks of multiplication gates require an additional m/t field elements, and the final output wire checks require k' more elements, for a total of $k + k' + (2 + \frac{1}{t})m$ field elements.

Prover computation: Applying § 3.3 and accounting for the additional cost of updating the $a_{k+4i-2}, a_{k+4i-3}, b_{k+4i-2}, b_{k+4i-3}$ terms, the prover's work is $2T(C) + 3m\ T(*) + (k + 6m)\ T(+)$, which assuming the cost of additions is negligible, is less than 3 times the cost of evaluation in the clear.

Verifier computation: Similarly, the verifier's work is $T(C) + 2m\ T(*) + 2m\ T(+)$, which assuming the cost of additions is negligible, is less than 2 times the cost of evaluation in the clear.

Remark 4.1. Because the VOLE instance can be compressed and unpacked in a streaming fashion, when implementing this NIZK protocol the vectors $\mathbf{a}, \mathbf{b}, \mathbf{v}$ do not need to ever be computed or stored in their entirety. This makes our protocol friendly to streaming and space considerations.

In particular, computations like $a_{k+4i-2-j} = \mathbf{r}_{2i-j} \cdot \mathbf{a}$ should be treated as shorthand for hard-coded evaluation of addition and scalar multiplication gates, and should certainly not be implemented by actually storing $\mathbf{r}_{2i-j}$ in memory and performing a dot product.

In fact, besides the memory costs of the VOLE, which are sublinear in the circuit size (see [8] for an exploration of trade-offs and optimizations available here), the only values that the verifier needs to store beyond what would be required for execution of the program in the clear is a single field element holding the product of the x_j terms in the current batch. The prover's memory cost is double the verifier's, since P has to store the values from both $\mathbf{a}$ and $\mathbf{b}$ that are currently "in-scope" along with the product of the c_j terms in the current batch.

5 Proof of Theorem 1.3

5.1 LPZK in the random oracle model

The desired result for LPZK from random VOLE is a consequence of the compiler to LPZK from random VOLE given in Lemma 2.1 and the following theorem.

Theorem 5.1 (LPZK in the ROM)**.** *For any NP-relation $R(x, y)$ and field $\mathbb{F}$, there exists an 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 $r \geq 1$. If C has k inputs, k' outputs, and m multiplication gates, we have $n = k + k' + m + 2r$, $n' = k$, $n'' = k' + m + 2r$, and $\varepsilon = \frac{2}{|\mathbb{F}|} + \frac{\ell}{|\mathbb{F}|^r}$. Moreover, the local computation of both the prover and the verifier consists of $O(r|C|)$ field operations and $O(1)$ cryptographic hash calls, for a cryptographic hash function $H : \mathbb{F}^m \to \mathbb{F}^{mr}$.*

5.2 Protocol

Since the batching in this algorithm is similar to the previous algorithm, and to simplify the presentation, we give the protocol without batching, and explain how to alter it for batching at the end.

Similar to § 4.2, the prover begins by constructing a line $\mathbf{v}(t) := \mathbf{a}t + \mathbf{b}$ with $\mathbf{v} \in \mathbb{F}^{k+k'+5m+3r+1}$, and then reduce to a shorter $\hat{\mathbf{v}}$ that we use for LPZK. For $0 \leq i \leq k + k' + 4m$, the prover defines a_i and b_i identically to their definitions in § 4.2, except each entry a_{k+4j} is chosen uniformly at random from $\mathbb{F}$, for $1 \leq j \leq m$, and each entry b_{k+4j} is chosen so that $b_{k+4j} = b_{k+4j-1}$.

The next r entries of $\mathbf{a}$ and $\mathbf{b}$ are chosen uniformly at random from $\mathbb{F}$. The remaining $m + 2r$ entries of $\mathbf{a}$ are all set equal to zero, and the remaining $m + 2r$ entries of $\mathbf{b}$ will be given explicitly later.

For $1 \leq i \leq m$, the prover computes

$$y_i := b_{k+4i-1} - a_{k+4i-3}b_{k+4i-2} - a_{k+4i-2}b_{k+4i-3}$$

and

$$z_i := -b_{k+4i-3}b_{k+4i-2},$$

and defines $\mathbf{y} = (y_i)$ and $\mathbf{z} = (z_i)$, where i ranges from 1 to m. For r the positive integer fixed in the statement of the theorem, let $H : \mathbb{F}^m \to \mathbb{F}^{mr}$ be some random crytographic hash function, and treat the output of H as a matrix in $M_{r\times m}(\mathbb{F})$. The prover then defines $\mathbf{w} := (w_i) := (a_{k+4i-1} - a_{k+4i})$, where i ranges from 1 to m. The prover then sets

$$y := (a_{k+k'+4m+1}, \ldots, a_{k+k'+4m+r})^T + H(\mathbf{w})\mathbf{y}^T$$

and

$$z := (b_{k+k'+4m+1}, \ldots, b_{k+k'+4m+r})^T + H(\mathbf{w})\mathbf{z}^T.$$

For $1 \le i \le m$, the prover sets

$$b_{k+k'+4m+r+i} := a_{k+4i-1} - a_{k+4i},$$

then the prover sets

$$\mathbf{b}[k + k' + 5m + r + 1 : k + k' + 5m + 2r] = y,$$

and

$$\mathbf{b}[k + k' + 5m + 2r + 1 : k + k' + 5m + 3r] = z,$$

writing $\mathbf{b}[i, j]$ for the projection onto coordinates i thru j inclusive.

Next, the prover computes from the pair $(\mathbf{a}, \mathbf{b})$ a line in a lower-dimensional space $\hat{\mathbf{v}}(t) := \hat{\mathbf{a}}t + \hat{\mathbf{b}} \in \mathbb{F}^{k+k'+2m+3r}$. For $1 \le i \le k$, we take $\hat{a}_i = a_i$ and $\hat{b}_i = b_i$. For $1 \le i \le m$ we take $\hat{a}_{k+i} = a_{k+4i}$ and $\hat{b}_{k+i} = b_{k+4i}$. For $1 \le i \le r$, we take $\hat{a}_{k+m+i} = a_{k+k'+4m+i}$ and $\hat{b}_{k+m+i} = b_{k+k'+4m+i}$. The remaining $k' + m + 2r$ values of $\mathbf{a}$ we set equal to zero. For $1 \le i \le k'$, we set $\hat{b}_{k+m+r+i} = \mathbf{r}_{2m+i} \cdot \mathbf{b}$. For $1 \le i \le m$, we set $\hat{b}_{k+k'+m+r+i} = w_i = a_{k+4i-1} - a_{k+4i}$. Finally, for $1 \le i \le 2r$, we set $\hat{b}_{k+k'+2m+r+i} = b_{k+k'+5m+r+i}$.

Now, having constructed $\hat{\mathbf{v}}(t)$, the prover and verifier run LPZK so that the verifier learns $\hat{\mathbf{v}}(\alpha)$, and, similar to § 4.2, expands $\hat{\mathbf{v}}(\alpha)$ to a vector $\mathbf{v} = \mathbf{a}\alpha + \mathbf{b}$. The verifier reconstructs v_{k+4i-1} as

$$v_{k+4i-1} = v_{k+4i} + \alpha v_{k+k'+m+r+i},$$

and the other missing values as in § 4.2.

The verifier now computes, for $1 \le i \le m$,

$$s_i := v_{k+4i-1}\alpha - v_{k+4i-3}v_{k+4i-2},$$

the vector $\mathbf{s} = (s_i)$, and the value

$$\begin{aligned} s &:= (v_{k+k'+4m+1}, \ldots, v_{k+k'+4m+r})^T + H(\mathbf{w})\mathbf{s}^T, \\ y_\alpha &:= (\mathbf{v}[k + k' + 5m + r + 1 : k + k' + 5m + 2r]) \\ z_\alpha &:= (\mathbf{v}[k + k' + 5m + 2r + 1 : k + k' + 5m + 3r]) \end{aligned}$$

and returns rej unless $y\alpha + z = s$. Then for $1 \le i \le k'$, the verifier checks that $\mathbf{r}_{2m+i} \cdot \mathbf{v} = v_{k+4m+i}$ and returns rej if any test fails, and acc otherwise.

5.3 Proof

Completeness: Each of the tests $\mathbf{r}_{2m+1} \cdot \mathbf{v} = v_{k+4m+i}$ are the same as in the previous section, and completeness follows by the same argument. If the prover is honest, the expression $y_\alpha\alpha + z_\alpha$ is equal to $y\alpha + z$. Comparing $y\alpha + z$ to s, we have $v_{k+k'+4m+r+i} = a_{k+4i-1} - a_{k+4i}$ by construction and $y_i\alpha + z_i = s_i$ by the same argument used in the previous section. We have

$$\begin{aligned} (v_{k+k'+4m+1}, \ldots, v_{k+k'+4m+r}) = {} & (a_{k+k'+4m+1}, \ldots, a_{k+k'+4m+r})\alpha \\ & +(b_{k+k'+4m+1}, \ldots, b_{k+k'+4m+r}), \end{aligned}$$

since the underlying elements are terms from the random VOLE, so $y\alpha + z = s$, as desired.

Zero Knowledge: The simulator chooses the values of v_i uniformly at random from $\mathbb{F}$, for $1 \leq i \leq k$, and likewise chooses v_{k+4i} uniformly at random for $1 \leq i \leq m$, and chooses $v_{k+k'+4m+i}$ uniformly at random, for $1 \leq i \leq r+m$. These values are all distributed uniformly at random and independently under an honest run of the protocol; the v_i's are distributed uniformly by the randomness of the b_i's, for $1 \leq i \leq k$, the v_{k+4i}'s by the randomness of b_{k+4i}, for $1 \leq i \leq m$, the $v_{k+k'+4m+i}$'s by the randomness of a_{k+4i}, for $1 \leq i \leq m$, and the $v_{k+k'+5m+r+i}$'s by the randomness of $b_{k+k'+5m+r+i}$, for $1 \leq i \leq r$. The simulator then computes $v_{k+4i-3}, v_{k+4i-2}, v_{k+4i-1}$, and s_i, for $1 \leq i \leq m$, and s and y_α, all as the verifier computes them during the actual protocol, and then computes $z_\alpha = s - \alpha y$. Finally, the simulator sets

$$\mathbf{v}[k+k'+5m+2r+1 : k+k'+5m+3r] = z_\alpha,$$

and $v_{k+4m+i} = \mathbf{r}_{2m+1} \cdot \mathbf{v}$ for $1 \leq i \leq k'$ and outputs acc.

Soundness: Define the vector

$$\mathbf{u}(t) := \left(v_{k+4i-1}(t)t - v_{k+4i-2}(t)v_{k+4i-3}(t)\right),$$

where i ranges from 1 to m. We write

$$s(t) := yt + z + H(\mathbf{w}) \cdot (\mathbf{u}(t))^T$$

for the system of r quadratic equations in t the prover implicitly constructs in the course of the protocol and the verifier evaluates at the point α. We write $\mathbf{u} := (a_{k+4i-1} - a_{k+4i-2}a_{k+4i-3})$ for the quadratic term of $\mathbf{u}(t)$, and note that, since a_{k+4i} is random, and each of a_{k+4i-2} and a_{k+4i-3} are determined by wire values to the left, $\mathbf{u}$ is determined by $\mathbf{w}$ along with randomness outside of a cheating prover's control.

If the prover cheats on a multiplication gate, there is some i for which

$$(v_{k+4i-1}(t)t - v_{k+4i-2}(t)v_{k+4i-3}(t)) - (y_i t - z_i)$$

is a nontrivial quadratic in t. Then either $s(t)$ is also a nontrivial quadratic in t, which gives a soundness error of at most $2/|\mathbb{F}|$ (i.e. if exactly one of the r quadratics is nontrivial), or else

$$0 = H(\mathbf{w}) \cdot (\mathbf{u})^T,$$

i.e. $\mathbf{u}^T$ lies in the kernel of $H(\mathbf{w})$. By the randomness of the oracle, the kernel is a random subspace of $\mathbb{F}^m$ of codimension r, and the probability that $\mathbf{u}$ lies in this subspace is $1/|\mathbb{F}|^r$, since $\mathbf{u}$ is entirely determined from $\mathbf{w}$. Over ℓ evaluations of H by a malicious prover, this gives a soundness error of $\ell/|\mathbb{F}|^r$, as desired.

5.4 Complexity

The computation of complexity is similar to the previous section.

The prover P needs to send y and z to the verifier, giving a communication cost of $2r$ elements for the verification of the multiplication gates, in addition to the $k + k' + m$ communication for wire values.

- Local computation of $O(r|C|)$ field operations and $O(1)$ cryptographic hash calls for both the prover and the verifier;
- Soundness error $\frac{2}{|\mathbb{F}|} + \frac{\ell}{|\mathbb{F}|^r}$;
- Communication of $k + k' + m + 2r$ field elements from P to V;
- rVOLE length $n = k + m + r$ over $\mathbb{F}$.

6 Non-Interactive Secure Computation

In this section we apply LPZK towards simplifying and improving the efficiency of the reusable protocol for non-interactive secure computation (NISC) from [14].

6.1 NISC definition

We start by giving a simplified definition of reusable NISC over VOLE, which strengthens the definition from [14]. The definition can be seen as a natural extension of the definition of LPZK to the case of secure computation, where both the sender and the receiver have secret inputs. Instead of the prover encoding its witness as a line and the verifier picking a random point, here the sender encodes its input as multiple lines and the receiver encodes its input as multiple points, one for each line. At a high level, reusable security is ensured by requiring that a malicious sender *know* whether the receiver will abort based on the lines that it picks (or VOLE inputs), independently of the receiver's inputs.

6.2 Certified VOLE

The main building block for NISC is a *certified* variant of VOLE, allowing the sender and the receiver to invoke multiple parallel instances of VOLE while assuring the receiver that the sender's VOLE inputs satisfy some global consistency relation.

6.2.1 Definitions and results

In its general form, *certified VOLE with a general arithmetic relation* the VOLE consistency requirement is specified by a general arithmetic circuit. We write cVOLE for this form of certified VOLE.

We begin with two more specialized forms, *weak* and *strong distributional certified VOLE with equality constraints*, which we write as wVOLE and sdVOLE, respectively. In both variants of distributional VOLE, the arithmetic circuit on the family of VOLEs are restricted to a set of equality constraints between coefficients. In wVOLE, the receivers' inputs are all offset by a random value α (so that R's inputs are uniformly distributed over $\mathbb{F}$, but not necessarily independent). In sdVOLE, we require that the receiver's inputs be uniformly distributed *and* independent.

Certified VOLE of these flavors can be realized by extending a family of random VOLEs with a NIZK proof that the random VOLEs satisfy the desired constraints. We give more precise definitions of these forms of certified VOLE as ideal functionalities in Figures 1, 2 and 3. We state this result as the following three lemmas.

Lemma 6.1. *A receiver R and a sender S can realize the functionality $\mathcal{F}_{wVOLE}^{(\mathbb{F})}$ with parameters $(\ell_1, \ell'_1, \ell_2, \ell'_2, I_a)$ in the rVOLE hybrid model with 2 instances of random VOLE of total length $\ell_1 + \ell_2$ and communication of*

$$\ell_1 + \sum_{i=1}^{2} \left(\ell_i - \ell'_i\right) + |I_a|$$

field elements from sender to receiver, as long as $|I_a| + \ell'_i \leq \ell_i$, for $i \in \{1, 2\}$. In particular, when $|I_a| = \ell_1 - \ell'_1 = \ell_2 - \ell'_2$, the communication cost is $\ell_1 + 3|I_a|$.

Lemma 6.2. *A receiver R and a sender S can realize the functionality $\mathcal{F}_{sVOLE}^{(\mathbb{F})}$ with parameters $(\ell_1, \ell'_1, \ell_2, \ell'_2, I_a, I_b)$ in the rVOLE hybrid model with 2 instances of random VOLE of total*

length $\ell_1 + \ell_2$ *and communication of*

$$\sum_{i=1}^{2} \left(\ell_i - \ell_i'\right) + |I_a| + |I_b|$$

field elements from sender to receiver, as long as $|I_a| + |I_b| + \ell_i' \leq \ell_i$*, for* $i \in \{1, 2\}$*. In particular, when* $|I_a| + |I_b| = \ell_1 - \ell_1' = \ell_2 - \ell_2'$*, the communication cost is* $3|I_a| + 3|I_b|$*.*

Lemma 6.3. *Fix an integer* $t \geq 1$*. A receiver* R *and a sender* S *can realize the functionality* $\mathcal{F}_{cVOLE}^{(\mathbb{F})}$*, in the rVOLE hybrid model with* $k + 2$ *instances of random VOLE. For a circuit* C *with* q *inputs,* q' *outputs, and* m *multiplication gates, these VOLE instances have total length*

$$\left(\sum_{i=1}^{k} \ell_i\right) + 2m + 12q,$$

and the protocol requires additional communication of

$$2\left(\sum_{i=1}^{k} \ell_i\right) + (2 + \tfrac{1}{t})m + 18q + q'$$

field elements from sender to receiver.

Figure 1: Weak distributional certified VOLE with equality constraints

Functionality $\mathcal{F}_{wVOLE}^{(\mathbb{F})}$: Weak distributional certified VOLE with equality constraints

Parametrized by a finite field $\mathbb{F}$, length parameters $(\ell_1, \ell_1'), (\ell_2, \ell_2')$, a sequence of ordered pairs $I_a := ((i, j))$ with $1 \leq i \leq \ell_1 - \ell_1'$ and $1 \leq j \leq \ell_2 - \ell_2'$ such that the projection of I_a onto its first or second coordinates gives no repeated elements.

Setup phase

- $\mathcal{F}_{wVOLE}^{(\mathbb{F})}$ chooses random $\alpha \in \mathbb{F}$ and vectors $\mathbf{a}^1, \mathbf{b}^1 \in \mathbb{F}^{\ell_1}$, $\mathbf{a}^2, \mathbf{b}^2 \in \mathbb{F}^{\ell_2}$, where we use superscripts for instances of VOLE, to distinguish from subscripts, which represent position within a vector.
- $\mathcal{F}_{wVOLE}^{(\mathbb{F})}$ sends α to R and the first $\ell_i - \ell_i'$ entries of $\mathbf{a}^i, \mathbf{b}^i$ to S, for $i \in \{1, 2\}$.
- R sends to $\mathcal{F}_{wVOLE}^{(\mathbb{F})}$ its input $\alpha_1 \in \mathbb{F}$.

Send phase

- S sends $(\hat{\mathbf{a}}^1, \hat{\mathbf{a}}^2)$ to $\mathcal{F}_{wVOLE}^{(\mathbb{F})}$, which are zero for the final ℓ_1' and ℓ_2' coordinates, respectively.
- $\mathcal{F}_{wVOLE}^{(\mathbb{F})}$ sends $\mathbf{v}^1 := (\mathbf{a}^1 + \hat{\mathbf{a}}^1) \cdot (\alpha + \alpha_1) + \mathbf{b}^1$ and $\mathbf{v}^2 := (\mathbf{a}^2 + \hat{\mathbf{a}}^2)\alpha + \mathbf{b}^2$ to R, and the remaining entries of $\mathbf{a}^i, \mathbf{b}^i$ to S, for $i \in \{1, 2\}$.
- For each pair $(i, j) \in I_a$, the ideal functionality checks that $a_i^1 + \hat{a}_i^1 = a_j^2 + \hat{a}_j^2$ and sends (i, j) to R and S if the check *does not* pass.
- If all pairs pass verification, the ideal functionality sends acc to R and S.

Figure 2: Strong distributional certified VOLE with equality constraints

Functionality $\mathcal{F}_{sVOLE}^{(\mathbb{F})}$: Strong distributional certified VOLE with equality constraints

Parametrized by a finite field $\mathbb{F}$, length parameters $(\ell_1, \ell_1'), (\ell_2, \ell_2')$, and two disjoint sequences of ordered pairs $I_a := ((i,j))$, $I_b := ((i',j'))$, with $1 \leq i, i' \leq \ell_1 - \ell_1'$ and $1 \leq j, j \leq \ell_2 - \ell_2'$ such that the projection of $I_a \cup I_b$ onto its first or second coordinates gives no repeated elements.

Setup phase

- $\mathcal{F}_{sVOLE}^{(\mathbb{F})}$ chooses uniformly random and independent $\alpha, \beta \in \mathbb{F}$ and vectors $\mathbf{a}^1, \mathbf{b}^1 \in \mathbb{F}^{\ell_1}$, $\mathbf{a}^2, \mathbf{b}^2 \in \mathbb{F}^{\ell_2}$.
- $\mathcal{F}_{sVOLE}^{(\mathbb{F})}$ sends α, β to R and the first $\ell_i - \ell_i'$ entries of $\mathbf{a}^i, \mathbf{b}^i$ to S, for $i \in \{1,2\}$.

Send phase

- S sends $(\hat{\mathbf{a}}^1, \hat{\mathbf{b}}^1, \hat{\mathbf{a}}^2)$ to $\mathcal{F}_{sVOLE}^{(\mathbb{F})}$, which are zero for the final ℓ_1', ℓ_1', and ℓ_2' coordinates, respectively.
- $\mathcal{F}_{sVOLE}^{(\mathbb{F})}$ sends $\mathbf{v}^1 := (\mathbf{a}^1 + \hat{\mathbf{a}}^1)\alpha + (\mathbf{b}^1 + \hat{\mathbf{b}}^1)$ and $\mathbf{v}^2 := (\mathbf{a}^2 + \hat{\mathbf{a}}^2)\beta + \mathbf{b}^2$ to R and the remaining entries of $\mathbf{a}^i$,$\mathbf{b}^i$ to S, for $i \in \{1,2\}$.
- For each pair $(i,j) \in I_a$, the ideal functionality checks that $a_i^1 + \hat{a}_i^1 = a_j^2 + \hat{a}_j^2$ and sends (i,j) to R and S if the check fails.
- For each pair $(i,j) \in I_b$, the ideal functionality checks that $b_i^1 + \hat{b}_i^1 = a_j^2 + \hat{a}_j^2$ and sends (i,j) to R and S if the check fails.
- If all pairs pass verification, the ideal functionality sends acc to R and S.

Figure 3: Certified VOLE with a general arithmetic relation

Functionality $\mathcal{F}_{cVOLE}^{(\mathbb{F})}$:

Parametrized by a finite field $\mathbb{F}$, a sequence of k positive integers $\ell_1, \ldots, \ell_k$, and an arithmetic circuit C on $q \leq 2\sum_{i=1}^{k} \ell_i$ inputs.

Setup phase

- R sends to $\mathcal{F}_{cVOLE}^{(\mathbb{F})}$ its input $(\alpha_1, \ldots, \alpha_k) \in \mathbb{F}^k$.

Send phases

- S sends $(\mathbf{a}_1, \ldots, \mathbf{a}_k, \mathbf{b}_1, \ldots, \mathbf{b}_k)$ to $\mathcal{F}_{cVOLE}^{(\mathbb{F})}$, where $\mathbf{a}_i, \mathbf{b}_i \in \mathbb{F}^{\ell_i}$
- $\mathcal{F}_{cVOLE}^{(\mathbb{F})}$ verifies that $(\mathbf{a}_1, \ldots, \mathbf{a}_k, \mathbf{b}_1, \ldots, \mathbf{b}_k)$ is a satisfying assignment for C.
- If the input does not pass verification, the ideal functionality sends $\perp$ to R. Otherwise, $\mathcal{F}_{cVOLE}^{(\mathbb{F})}$ computes $\mathbf{v}_i := \mathbf{a}_i\alpha_i + \mathbf{b}_i$ and sends $(\mathbf{v}_1, \ldots, \mathbf{v}_k)$ to R.

6.2.2 The protocols

wVOLE. When $|I_a| = 0$, we realize $\mathcal{F}_{wVOLE}^{(\mathbb{F})}$ with length parameters $\ell_1' \leq \ell_1$ and $\ell_2' \leq \ell_2$, as follows. R and S engage in two random VOLE protocols, so that S holds $\mathbf{v}^1(t) := \mathbf{a}^1 t + \mathbf{b}^1 \in \mathbb{F}^{\ell_1}$ and $\mathbf{v}^2(t) := \mathbf{a}^2 t + \mathbf{b}^2 \in \mathbb{F}^{\ell_2}$ and R holds $\mathbf{v}^1(\omega)$, $\mathbf{v}^2(\alpha)$, for random $\omega, \alpha \in \mathbb{F}$.

As in [14], to convert from ω to a receiver input $\alpha + \alpha_1$, R sends $\alpha + \alpha_1 - \omega$ to S, and S responds with $\mathbf{s} := \mathbf{a}^1(\alpha + \alpha_1 - \omega) + \mathbf{c}^1 - \mathbf{b}^1$, for some random vector $\mathbf{c}^1 \in \mathbb{F}^{\ell_1}$. Then S holds $\hat{\mathbf{w}}^1(t) := \mathbf{a}^1 t + \mathbf{c}^1$ and R holds $\mathbf{w}^1(\alpha + \alpha_1) = \mathbf{v}^1(\omega) + \mathbf{s}$. To complete this simple protocol, S takes $\hat{\mathbf{a}}^i := \mathbf{0}$, and R and S output acc.

For $|I_a| > 0$, we define our protocol by induction. Let $\mathcal{F}_{wVOLE}^{(\mathbb{F})}(\ell_1, \ell_1' + 1, \ell_2, \ell_2' + 1, I_a)$ be a realization of the weak distributional certified VOLE functionality, and let (i, j) be chosen disjoint from I_a. To realize the functionality $\mathcal{F}_{wVOLE}^{(\mathbb{F})}(\ell_1, \ell_1', \ell_2, \ell_2', I_a \cup \{(i, j)\})$, R and S first follow $\mathcal{F}_{wVOLE}^{(\mathbb{F})}(\ell_1, \ell_1' + 1, \ell_2 + 1, \ell_2', I_a)$ to construct lines $\mathbf{v}^1(t) := \mathbf{a}^1 t + \mathbf{b}^1$ and $\mathbf{v}^2(t) := \mathbf{a}^2 t + \mathbf{b}^2$, with R holding $\mathbf{v}^1(\alpha + \alpha_1)$ and $\mathbf{v}^2(\alpha)$ and S holding $\mathbf{a}^i, \mathbf{b}^i$ for $i \in \{1, 2\}$.

In the send phase, S sends

$$(c_1, c_2, c_3) := (b_j^2 - a_{\ell_1 - \ell_1'}^1, b_i^1 - a_{\ell_2 - \ell_2'}^2, b_{\ell_1 - \ell_1'}^1 - b_{\ell_2 - \ell_2'}^2)$$

to R. To certify the VOLE, R defines

$$(w_1^1, w_1^2, w_2^1, w_2^2) := (v_i^1, v_j^2, c_1 \cdot (\alpha + \alpha_1) + v_{\ell_1 - \ell_1'}^1, c_2 \alpha + v_{\ell_2 - \ell_2'}^2)$$

and tests whether

$$w_1^1 \alpha + w_2^1 - w_1^2(\alpha + \alpha_1) - w_2^2 = c_3.$$

This is a realization of the ideal functionality with $(\hat{\mathbf{a}}^1, \hat{\mathbf{a}}^2)$ defined exactly as in the previous round of induction, except for the entries in position $\ell_1 - \ell_1'$ and $\ell_2 - \ell_2'$, respectively, which are defined by c_1 and c_2, respectively.

sVOLE. As above, when $|I_a \cup I_b| = 0$, we set $\hat{\mathbf{a}}^1 = \hat{\mathbf{a}}^2 = \mathbf{0}$, and implementing $\mathcal{F}_{sVOLE}^{(\mathbb{F})}$ becomes equivalent to implementing two random VOLEs of lengths ℓ_1 and ℓ_2, respectively, S received random $\mathbf{v}^i(t) := \mathbf{a}^i t + \mathbf{b}^i \in \mathbb{F}^{\ell_i}$ and R receives random $\alpha, \beta \in \mathbb{F}$ and $\mathbf{v}^1(\alpha), \mathbf{v}^2(\beta)$.

In the inductive step, we add entries to I_a as in wVOLE. To add an entry to I_b, we observe that S can compute the line $\mathbf{w}^1(t) := \mathbf{b}^1 t + \mathbf{a}^1 = t \cdot \mathbf{v}^1(t^{-1})$ and R can compute the point $\mathbf{w}^1(\alpha^{-1}) = \alpha^{-1} \cdot \mathbf{v}^1(\alpha)$ without any additional communication. α^{-1} and β are uniformly random and independent, since α and β are, so we can apply the same construction to the pair $(\mathbf{w}^1, \mathbf{v}^2)$.

Concretely, in the send phase, S sends

$$(c_1, c_2, c_3) := (b_j^2 - b_{\ell_1 - \ell_1'}^1, a_i^1 - a_{\ell_2 - \ell_2'}^2, a_{\ell_1 - \ell_1'}^1 - b_{\ell_2 - \ell_2'}^2)$$

to R. To certify the VOLE, R defines

$$(w_1^1, w_1^2, w_2^1, w_2^2) := (\alpha^{-1} v_i^1, v_j^2, c_1 \alpha^{-1} + v_{\ell_1 - \ell_1'}^1 \alpha^{-1}, c_2 \beta + v_{\ell_2 - \ell_2'}^2)$$

and tests whether

$$w_1^1 \beta + w_2^1 - w_1^2 \alpha^{-1} - w_2^2 = c_3.$$

cVOLE. Generate $k+2$ instances of random VOLE. Take I_a to be the set of triples (h,i,j) corresponding to entries (a_i^h, b_i^h) that are inputs to C, together with the certification that $a_i^h = a_j^{k+1}$.

As in wVOLE, we begin with a 2-round stage of communication to adjust the receiver's points of evaluation for the VOLEs. We set the receiver's inputs to the VOLEs to be $\alpha_1+\alpha, \ldots, \alpha_k+\alpha, \alpha, \beta$, where $\alpha, \beta \in \mathbb{F}$ are generated uniformly at random.

R and S run weak distributional certified VOLE with equality constraints on each pair of VOLEs $(h, k+1)$ and the subset of I_a with first entry h, for $1 \leq h \leq k$, to construct the certifications for each of the triples in I_a. That is, after running $\mathcal{F}_{wVOLE}$, for each triple (h,i,j), the receiver holds $a_i^h \cdot (\alpha + \alpha_h) + b_i^h$ and $a_j^{k+1}\alpha + b_j^{k+1}$, and can subtract to obtain $a_i^h\alpha_h + b_i^h - b_j^{k+1}$.

After suitable rearranging, these entries $a_i^h\alpha_h + b_i^h - b_j^{k+1}$ make up the final family of certified VOLEs. Already we can evaluate any circuit on the a_i^h's by running LPZK on the $k+1$st instance of our orignal VOLE. We now need the remaining inputs to C, that is, the $b_i^h - b_j^{k+1}$ terms.

We obtain this via three instances of strong distributional certified VOLE. R and S run strong distributional certified VOLE first on each pair of VOLEs $(h, k+2)$, for $1 \leq h \leq k$, to move all entries b_i^h to $\mathbf{a}^{k+2}$. Running strong distributional certified VOLE on the pair $(k+1, k+2)$ moves the terms b_j^{k+1} to $\mathbf{a}^{k+2}$. Now, each party can locally subtract to obtain the desired $b_i^h - b_j^{k+1}$ terms as entries of $\mathbf{a}^{k+2}$. Running strong distributional certified VOLE again on the pair $(k+1, k+2)$ moves those values to $\mathbf{a}^{k+1}$, where we can apply LPZK-NIZK to prove that C is satisfied, as desired.

6.2.3 Proof of Lemma 6.1

Correctness: Recall the definitions of $(w_1^1, w_1^2, w_2^1, w_2^2)$ from the VERIFY step of the wVOLE protocol above. By the correctness of the random VOLE protocol and the fixed VOLE from random VOLE conversion, the receiver will compute

$$\begin{aligned} w_1^1\alpha + w_2^1 - w_1^2(\alpha+\alpha_1) - w_2^2 = (a_i^1 - a_j^2)(\alpha+\alpha_1)\alpha_1 + (b_i^1 - a^2_{\ell_2-\ell_2'} - c_2)(\alpha+\alpha_1) \\ +(c_1 + a^1_{\ell_1-\ell_1'} - b_j^2)\alpha_{k+1} + (b^1_{\ell_1-\ell_1'} - b^2_{\ell_2-\ell_2'}). \end{aligned}$$

If the sender is honest, the first three terms on the right-hand side vanish, and we are left with

$$w_1^1\alpha + w_2^1 - w_1^2 \cdot (\alpha+\alpha_1) - w_2^2 = b^1_{\ell_1-\ell_1'} - b^2_{\ell_2-\ell_2'} = c_3,$$

as desired.

Security against a malicious receiver:

We give a simulator Sim_R of S that generates for a corrupted receiver $\hat{R}$ a view indistinguishable from an honest run of the protocol. The simulator, like the protocol, is constructed by induction on $|I_a|$. When $|I_a| = 0$, we take Sim_R to be the standard simulator for the reduction from VOLE to random VOLE (see e.g. [8, 14]), ignoring the final ℓ_i' entries of $\mathbf{v}^i$, for $i \in \{1,2\}$. Now, suppose $|I_a| > 0$.

Let Sim'_R be a simulator of S in the protocol for some fixed set of parameters $(\ell_1, \ell_1'+1)$, $(\ell_2, \ell_2'+1)$, and I_a. Let (i,j) be chosen disjoint from I_a. Then Sim_R uses Sim'_R to generate $\mathbf{v}^1, \mathbf{v}^2$ as a proof of every equality constraint in $I_a \cup \{(i,j)\}$ except for (i,j).

Finally, for the pair (i,j), Sim_R generates c_1, c_2, w_1^2 and w_2^2 uniformly at random. Sim_R then computes the associated value c_3 as $w_1^1\alpha + w_2^1 - w_1^2 \cdot (\alpha+\alpha_1) - w_2^2$.

From the definition of the ideal wVOLE functionality, the values $a^1_{\ell_i-\ell_i'}, b^1_{\ell_i-\ell_i}$ are never given to Sim'_R, and so are independent of $\mathbf{v}^1, \mathbf{v}^2$.

Since $a^1_{\ell_1-\ell_1'}$ and $a^2_{\ell_2-\ell_2'}$ are chosen uniformly at random from $\mathbb{F}$ in an honest run of the protocol, c_1 and c_2 are distributed independently of $\mathbf{v}^1, \mathbf{v}^2$ and uniformly at random under an honest run

of the protocol as well as under Sim_R. Similarly, the uniform randomness of b and b guarantees the independence and uniform randomness of w_2^1 and w_2^2. Therefore the only way to distinguish between Sim_R and an honest run of the protocol on $I_a \cup \{(i,j)\}$ is to distinguish the output of Sim'_R and an honest run of the protocol on I_a, and we are done by induction.

Security against a malicious sender: We give a simulator Sim_S of R that generates for a corrupted sender $\hat{S}$ a view indistinguishable from an honest run of the protocol.

As in the case of a malicious receiver, proof proceeds by induction, with the case $|I_a| = 0$ following from the reduction from VOLE to random VOLE. For $|I_a| > 0$, the simulator Sim_S begins with a simulator Sim'_S that simulates $\hat{S}$'s view during the execution of the wVOLE protocol. The only additional work that Sim_S does is to inspect their values $(\mathbf{a}^1, \mathbf{b}^2, \mathbf{a}^2, \mathbf{b}^2)$ and output $\perp$ to $\hat{S}$ if any of the equality constraints are violated.

For $\hat{S}$ to distinguish Sim_S and an honest run of the protocol, he must cheat in such a way that R does not detect it. If he does not cheat, both R and Sim_S will always accept, and when he does cheat, Sim_S always detects it. By induction, we restrict ourself to the case where $\hat{S}$ cheats on the pair (i,j) just added to I_a. The receiver R only fails to detect this cheating when $a_i^1 \neq a_j^2$ but $w_1^1\alpha + w_2^1 - w_1^2(\alpha + \alpha_1) - w_2^2 = c_3$. But writing $\hat{S}$'s view as $w_1^1 t + w_2^1 - w_1^2(t + t') - w_2^2$, this reduces to a quadratic in t^2 with leading term

$$\left(a_i^1 - a_j^2\right) t^2.$$

When $\hat{S}$ cheats, we have $\left(a_i^1 - a_j^2\right) \neq 0$, so to persuade R the input is valid $\hat{S}$ must force a nontrivial quadratic in t to be equal to zero. This is equivalent to guessing two values in $\mathbb{F}$, one of which is equal to t. When R is honest, t is chosen uniformly at random from $\mathbb{F}$, so that the ability of $\hat{S}$ to distinguish Sim_S and the behavior of an honest protocol is bounded above by $2/\mathbb{F}$.

Complexity: The total VOLE length is $\ell_1 + \ell_2$, by construction.

Converting the initial random VOLE to fixed VOLE contributes $\sum_{i=1}^{2} (\ell_i - \ell'_i)$ to the communication cost, and changing the receiver input for the first VOLE from a random element ω to $\alpha + \alpha_1$ requires communication of an additional ℓ_1 field elements. For each step of the induction, S sends three values (c_1, c_2, c_3) and the values ℓ'_1, ℓ'_2 are decreased by one, so that increasing $|I_a|$ by one increases the expression

$$\ell_1 + \sum_{i=1}^{2} \left(\ell_i - \ell'_i\right) + |I_a|$$

by three, and therefore this expression is equal to the communication cost, by induction.

The bounds $|I_a| + \ell'_i \leq \ell_i$ hold trivially for $|I_a| = 0$, and are preserved by the induction step, which we can perform as long as $\ell_i - \ell'_i > 0$ for $i \in \{1, 2\}$.

6.2.4 Proof of Lemma 6.2

Correctness: We need to check only the verification step for the entries of $|I_b|$, since the rest of the verification is dealt with in the previous subsection. Recall the definitions of $w_1^1, w_2^1, w_1^2, w_2^2$ for the sVOLE protocol. The receiver computes

$$\begin{aligned} w_1^1\beta + w_2^1 - (w_1^2\alpha + w_2^2) = (b_i^1 - a_j^2)\alpha^{-1}\beta + (a_i^1 - a_{\ell_2-\ell'_2}^2 - c_2)\beta + \\ (c_1 - b_j^2 + b_{\ell_1-\ell'_1}^1)\alpha^{-1} + (a_{\ell_1-\ell'_1}^1 - b_{\ell_2-\ell'_2}^2). \end{aligned}$$

If the sender is honest, the first three terms on the right-hand side vanish, and we are left with

$$w_1^1\beta + w_2^1 - (w_1^2\alpha + w_2^2) = a^1_{\ell_1-\ell_1'} - b^2_{\ell_2-\ell_2'} = c_3,$$

as desired.

Security against a malicious receiver: As in the case of $\mathcal{F}^{(\mathbb{F})}_{wVOLE}$, we make the simulator Sim_R choose $c_1, c_2 w_2^1, w_2^2$ from the uniform distribution. The ret of the argument is identical, except with $b^1_{\ell_1-\ell_1'}$ and $a^2_{\ell_2-\ell_2'}$ guaranteeing the randomness of c_1 and c_2, respectively, and $a^1_{\ell_1-\ell_1'}$ and $b^2_{\ell_2-\ell_2'}$ guaranteeing the randomness of w_2^2 and w_2^2, respectively. The rest of the argument is identical.

Security against a malicious sender: As with $\mathcal{F}^{(\mathbb{F})}_{wVOLE}$, we choose a simulator Sim_S that always detects when a corrupted sender $\hat{S}$ cheats, and need to show this matches the honest run of the program with a cheating sender.

If $\hat{S}$ chooses values (c_1, c_2, c_3) that pass verification, then $\hat{S}$ can derive the bivariate polynomial expression

$$\left(\left(b_i^1 - a_j^2\right)u + \left(c_1 - b_j^2 + b^1_{\ell_1-\ell_1'}\right)\right)t = \left(a_i^1 - a^2_{\ell_2-\ell_2'} - c_2\right)u + a^1_{\ell_1-\ell_1'} - b^2_{\ell_2-\ell_2'} - c_3,$$

which is satisfied by $t = \alpha^{-1}$, $u = \beta$. When R is honest, α^{-1} and β are independent and uniform on $\mathbb{F}$. When $\hat{S}$ cheats, we have $b_i^1 - a_j^2 \neq 0$, and the probability that $\hat{S}$ can force the coefficient of t on the lefthand side to be zero is $1/|\mathbb{F}|$. When the coefficient of t is nonzero, dividing gives t as a function of u. But, by the independence of α^{-1} and β, the probability that this function is correct is again $1/|\mathbb{F}|$. By a union bound, the probability that $\hat{S}$ deceives R in an honest run of the protocol is at most $2/|\mathbb{F}|$.

Thus, as in the proof for $\mathcal{F}^{(\mathbb{F})}_{wVOLE}$, a malicious $\hat{S}$ has at most a $2/|\mathbb{F}|$ probability of distinguishing between Sim_S and an honest run of the protocol.

Complexity: The total VOLE length is $\ell_1 + \ell_2$, by construction. We again require $\sum_{i=1}^{2}(\ell_i - \ell_i')$ field elements of communication to convert from fixed VOLE to random VOLE, and because both receiver's inputs are chosen uniformly at random, we do not need the additional ℓ_1 term from the wVOLE case.

As in the case of wVOLE, each entry of $I_a \cup I_b$ requires communicating three elements and decreasing the values ℓ_1', ℓ_2' by one, so that the expression

$$\sum_{i=1}^{2}\left(\ell_i - \ell_i'\right) + |I_a| + |I_b|$$

is equal to the communication cost, by induction.

Again, the bounds $|I_a| + |I_b| + \ell_i' \leq \ell_i$ hold trivially for $|I_a| + |I_b| = 0$, and are preserved by the induction step, which we can perform as long as $\ell_i - \ell_i' > 0$ for $i \in \{1, 2\}$.

6.2.5 Proof of Lemma 6.3

Correctness and security against malicious sender and receiver are immediate from the correctness and security of the underlying protocols. We only need to check that the sequence $\alpha_1 + \alpha, \ldots, \alpha_k + \alpha, \alpha, \beta$ satisfy the conditions for wVOLE and sVOLE each time we invoke them. We invoke wVOLE on pairs $(\alpha_i + \alpha, \alpha)$, which obviously satisfy the wVOLE condition. For pairs $(\alpha_i + \alpha, \beta)$,

the values $(\alpha_i + \alpha)^{-1}$ and *beta* are each uniformly distributed and independent. Likewise when performing sVOLE between the $(k+1)$st and $(k+2)$nd instance of VOLE, the values α and β are uniformly distributed and independent.

Complexity: The cVOLE protocol is built up of an instance of wVOLE and 3 instances of sVOLE, each of which imposes q equality constraints, and an LPZK-NIZK proof.

For each of these $4q$ equality constraints, we require an additional entry of VOLE in either $\mathbf{v}^{k+1}$ or $\mathbf{v}^{k+2}$, and two additional entries of VOLE to supply the randomness for the distributional VOLE proof.

This gives a total VOLE length of

$$\sum_{i=1}^{k} \ell_i + 12q$$

prior to the construction of the LPZK proof. The inputs to C have already been included in $\mathbf{a}^{k+1}$, so the LPZK proof requires an additional $2m$ VOLE instances.

Likewise, the communication cost consists of $3 \cdot 4q + \sum_{i=1}^{k} \ell_i$ elements for the distributional VOLE, another $4q$ for the entries of VOLE in $\mathbf{v}^{k+1}$ and $\mathbf{v}^{k+2}$ that the distributional VOLE instances are certifying, and $2q + \sum_{i=1}^{k} \ell_i$ for the conversion for the first k instances of VOLE from random receiver inputs to the list $(\alpha_1 + \alpha, \ldots, \alpha_k + \alpha)$. Finally the LPZK-NIZK requires communication of an additional $(2 + \frac{1}{t})m + q'$ elements. Combining these terms gives the VOLE length and communication costs as desired.

6.3 Reusable NISC over VOLE

In this section we build on certified VOLE to compile NISC protocols with security against semi-honest senders into reusable NISC protocols in the fully malicious setting. We follow the same high level approach of [14], but present the compiler at a higher level of generality and with a more refined efficiency analysis.

Consider a two-party sender-receiver functionality $f(\mathbf{x}, \mathbf{y})$ where the receiver R holds $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{F}^n$ and the sender S hold inputs $\mathbf{y} = (y_1, \ldots, y_n) \in \mathbb{F}^m$. The function f is arithmetic, in the sense that its outputs are defined by a sequence of ℓ arithmetic branching programs $P_1, \ldots, P_\ell$ over $\mathbb{F}$, where program P_i has s_i nodes. (Note that such an arithmetic program P_i can simulate any arithmetic formula with s_i additions and multiplication gates.)

The goal is to securely evaluate f using only parallel instances of VOLE. (The ideal VOLE instances can be implemented using the same kind of cryptographic compilers we used in the context of LPZK.) We also require the NISC protocol to be *reusable* in the sense that if the receiver's input is fixed but the sender's input changes, the same VOLE inputs of the receiver can be securely reused, even if the sender can obtain partial information about the receiver's outputs in the different invocations. This feature is impossible to achieve in the information-theoretic setting when we use OT instead of VOLE [14].

To get a reusable NISC for f, we take the following two-step approach:

1. Using a so-called "Decomposable Affine Randomized Encoding" (DARE) for branching programs [22, 4] (an arithmetic variant of information-theoretic garbling), we get a NISC protocol for f with n instances of VOLE, each of length $S_i = \sum_{i=1}^{\ell} s_i^2$. (In fact, for the j-th VOLE instance, it suffices to sum over the output indices i that depend on input j.) This protocol is secure against a malicious receiver R and a *semi-honest* sender R.

2. To obtain reusable security against a malicious S (while maintaining security against malicious R) we replace the parallel VOLE in the previous protocol by *certified* VOLE, where the circuit C specifying the consistency relation takes the sender's input $\mathbf{y}$ and randomness in the previous protocol as a witness, and checks that the sender's VOLE inputs are obtained by applying the honest sender's algorithm to the witness. Using naive matrix multiplication, this requires a circuit C of size $S = \sum_{j=1}^{n} S_j + \sum_{i=1}^{\ell} s_i^3$. Applying our protocol for $\mathcal{F}_{cVOLE}^{(\mathbb{F})}$ with the arithmetic relation specified by C, we ensure that whenever a malicious sender does not provide a witness that "explains" its VOLE inputs by an honest sender strategy, the receiver outputs $\perp$ except with $O(1/\mathbb{F})$ probability. In particular, a (reusable) simulator for a malicious sender interacting with the $\mathcal{F}_{cVOLE}^{(\mathbb{F})}$ functionality either outputs the input $\mathbf{y}$ found in the witness, if the consistency check specified by C passes, or $\perp$ if C fails.

Combining the above two steps, we derive the feasibility result from [14] in a simpler way.

Theorem 6.4 (Reusable arithmetic NISC over VOLE)**.** *Suppose $f : \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^\ell$ is a sender-receiver functionality whose i-th output can be computed by an arithmetic branching programs over $\mathbb{F}$ of size s_i that depends on d_i inputs. Then f admits a reusable NISC protocol over VOLE with the following efficiency and security features:*

- *The protocol uses $n + 2$ parallel VOLE instances.*
- *The total length of the VOLE instances is $14 \sum_{i=1}^{\ell} d_i s_i^2 + 2 \sum_{i=1}^{\ell} s_i^3$.*
- *The simulation error (per invocation) is $\varepsilon = O(1/|\mathbb{F}|)$.*

Chase et al. [14] show how to bootstrap Theorem 6.4 to get reusable NISC over VOLE for general Boolean circuits, by making (a noon-black-box) use of any pseudorandom generator, or equivalently a one-way function.

6.4 NISC Example: Bounded Inner Product

In this section we showcase the usefulness of reusable arithmetic NISC by presenting an optimized construction for a natural functionality: an inner product between the receiver's input vector and the sender's input vector, where the sender's vector is restricted to have a bounded L_2 norm. This functionality is useful for measuring similarity between two feature vectors. The bound on the sender's input is essential for preventing a malicious sender from inflating the level of similarity by scaling its input.

6.4.1 Setup

Let R hold inputs $\mathbf{x} = (x_1, \ldots, x_n)$ and S hold inputs $\mathbf{y} = (y_1, \ldots, y_n)$ such that $y_i \in \{0, 1, \ldots, K\}$ and

$$\sum_{i=1}^{n} y_i \leq C,$$

for some other constant C. R desires to compute the dot product $\mathbf{x} \cdot \mathbf{y}$ (as a measure of the similarity of R and S's inputs). To simplify the protocol, we restrict to the case where K and C are powers of 2.

6.4.2 Protocol

S begins with a sequence of n inputs (y_i), and selects associated random masks z^i.

First S engages in pre-processing of their data by computing the bit decomposition (c_{ij}) of each element y_i and the bit decomposition (c_{sj}) of the sum $y := \sum y_i$.

R and S generate $n+2$ instances of random $VOLE$. Write α for the $(n+1)$st random input of R, and β for the $(n+2)$nd. Then R and S jointly apply $\mathcal{F}_{wVOLE}^{(\mathbb{F})}$ to move all entries y_i to the $(n+1)$st instance of VOLE. They next preform $\mathcal{F}_{sVOLE}^{(\mathbb{F})}$ to move each of the entries z_i from $\mathbf{b}_i$ to $\mathbf{a}_{k+2}$, so that R learns the following:

- $\mathbf{v}_1^i := y_i(x_i+\alpha)+z_i$, for $1 \le i \le n$, and z_i a random element from the original random VOLE.
- $\mathbf{v}_i^{n+1} := y_i\alpha + w_i$, for $1 \le i \le n-1$, with w_i from the random VOLE.
- $\mathbf{v}_n^n := y_n\alpha + w_n$, where w_n is chosen such that $\sum_{i=1}^n z_i = \sum_{i=1}^n w_i$.
- $\mathbf{v}_i^{n+2} := z_i\beta + v_i$, for $1 \le i \le n$, with v_i from the random VOLE.
- $\mathbf{v}_{n+1}^{n+1} := y_{n+1}\alpha + w_{n+1}$, with y_{n+1} and w_{n+1} from the random VOLE.
- $\mathbf{v}_{n+1}^{n+2} := w_{n+1}\beta + v_{n+1}$, for v_{n+1} from the random VOLE.

S sends the values $y := \sum_{i=1}^{n+1} y_i$, $w := \sum_{i=1}^{n+1} w_i$ and $v := \sum_{i=1}^{n+1} v_i$ in the clear. The receiver then verifies that

$$\sum_{i=1}^{n+1} \mathbf{v}_{n+1}[i] = y\alpha + w$$

and that

$$\sum_{i=1}^{n+1} \mathbf{v}_{n+2}[i] = w\beta + v,$$

along with verifying that $\mathcal{F}_{wVOLE}$ and $\mathcal{F}_{sVOLE}$ were executed correctly.

Next, R and S execute LPZK-NIZK on the $(n+1)$st instance of VOLE, running the NIZK proof on the values already transfered onto $\mathbf{a}_{n+1}$ along with the values c_{ij} and c_{si}. The proof checks that c_{ij} and c_{si} are in $\{0,1\}$ by evaluating the quadratic $t^2 - t$ on each entry, and then checks that the bit-decomposition is correct by computing and revealing $y_i - \sum_j c_{ij}2^j$ and $y - \sum_j c_{sj}2^j$.

Finally, R computes the output value

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{n} \left(\mathbf{v}_i[1] - \mathbf{v}_{n+1}[i]\right).$$

6.5 Proof

Correctness is straightforward, and we omit the details.

For security against a malicious receiver, the simulator Sim_R chooses y, w, v at random and all values $\mathbf{v}_i[j]$ at random except for $\mathbf{v}_{n+1}[n]$, $\mathbf{v}_{n+1}[n+1]$ and $\mathbf{v}_{n+2}[n+1]$, which we learn from the equations for $\mathbf{x} \cdot \mathbf{y}$, $y\alpha + w$ and $w\beta + v$, respectively. This matches the distribution of R's view under an honest run of the protocol, since the w_i's and v_i's are all chosen uniformly at random subject to the three constraints above.

For security against a malicious sender, we need to check only that $\hat{S}$ cannot fool an honest R by giving it incorrect values of y, w, or v. But if the sender cheats, the expressions for $y\alpha + w$ and

$w\beta + v$ reduce to a nontrivial linear equation in α or β, respectively, that would imply that $\hat{S}$ had guessed α or β, which can only happen with probability $1/\mathbb{F}$. The rest of the security argument follows from the security of $\mathcal{F}_{wVOLE}^{(\mathbb{F})}$, $\mathcal{F}_{sVOLE}^{(\mathbb{F})}$ and LPZK-NIZK.

6.5.1 Complexity

The protocol requires the $3n + 2$ entries of VOLE listed, which in turn require communication of $3n + 2$ field elements to convert from random VOLE to fixed VOLE (one per entry, except we need two field elements for $\mathbf{v}_n[n]$ and none for $\mathbf{v}_{n+1}[n + 1]$.)

The protocol contains additionally 3 field elements of communication for y, w, v, $2n + 1$ equality constraints implemented under $\mathcal{F}_{sVOLE}^{(\mathbb{F})}$, and a LPZK-NIZK verification of a circuit with $n + n \log K + \log(C)$ inputs and $n + n \log K + \log(C)$ multiplication gates.

Because our construction avoids moving $|I_b|$ entries from $\mathbf{a}_{n+2}$ to $\mathbf{a}_{n+1}$, we pay the same cost in VOLE entries and communication for the equality constraints on a_i's and b_i's.

Combining the results throughout this paper, this gives us a total of

$$(3n + 2) + 3(2n + 1) + n \log K + \log(C) + 1 = 9n + n \log K + O(1)$$

entries of VOLE and communication of

$$\begin{aligned}(3n + 2) + 3 + 4(2n + 1) + (2 + \tfrac{1}{t})(n + n \log K + \log(C)) + 1 \\ = (13 + 2 \log K + \tfrac{1+\log K}{t})n + O(1)\end{aligned}$$

field elements.

We note that we can allow the bounds K and C to hold values other than powers of two by evaluating a boolean comparison circuit on the bit decomposition of the y_i's and the sum y, at the cost of an additional $O(n \log K)$ multiplication gates in the LPZK-proof.

Acknowledgements

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. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited). Y. Ishai additionally supported by ERC Project NTSC (742754), NSF-BSF grant 2015782, BSF grant 2018393, and ISF grant 2774/20.

References

[1] *ZKProof Standards*, 2020. URL: `https://zkproof.org`.

[2] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In *EUROCRYPT 2014*, pages 387–404, 2014.

[3] Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. Secure arithmetic computation with constant computational overhead. In *CRYPTO 2017, Part I*, pages 223–254, 2017.

[4] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In *FOCS 2011*, pages 120–129, 2011.

[5] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In *CRYPTO 2019, Part III*, Lecture Notes in Computer Science, pages 701–732, 2019.

[6] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC 2013*, pages 315–333, 2013.

[7] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In *ASIACRYPT 2017, Part III*, pages 336–365, 2017.

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

[9] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *CCS 2019*, pages 291–308, 2019.

[10] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-lpn. In *CRYPTO 2020, Part II*, pages 387–416.

[11] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO 2019, Part III*, pages 489–518, 2019.

[12] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In *FOCS 2020*, 2020. Full version: `https://eprint.iacr.org/2020/1417`.

[13] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *CCS 2017*, pages 1825–1842, 2017.

[14] 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.

[15] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *EUROCRYPT 2010*, pages 445–465, 2010.

[16] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In *EUROCRYPT 2015, Part II*, pages 191–219, 2015.

[17] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *USENIX Security 2016)*, 2016.

[18] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4):27:1–27:64, 2015.

[19] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[20] 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.

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

[22] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP 2002*, pages 244–256, 2002.

[23] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. In *EUROCRYPT 2011*, pages 406–425, 2011.

[24] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM J. Comput.*, 39(3):1121–1152, 2009.

[25] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *CCS 2018*, pages 525–537. ACM, 2018.

[26] Dakshita Khurana, Rafail Ostrovsky, and Akshayaram Srinivasan. Round optimal black-box "commit-and-prove". In *Theory of Cryptography Conference*, pages 286–313, 2018.

[27] Joe Kilian, Silvio Micali, and Rafail Ostrovsky. Minimum resource zero-knowledge proof. In *CRYPTO 1989*, pages 545–546. Springer, 1989.

[28] Alex Lombardi, Willy Quach, Ron D. Rothblum, Daniel Wichs, and David J. Wu. New constructions of reusable designated-verifier nizks. In *CRYPTO 2019*, pages 670–700.

[29] Payman Mohassel and Mike Rosulek. Non-interactive secure 2pc in the offline/online and batch settings. In *EUROCRYPT 2017, Part III*, pages 425–455, 2017.

[30] Willy Quach, Ron D. Rothblum, and Daniel Wichs. Reusable designated-verifier nizks for all NP from CDH. In *EUROCRYPT 2019*, pages 593–621.

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

[32] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. Cryptology ePrint Archive, Report 2020/925, 2020. `https://eprint.iacr.org/2020/925`.

[33] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *CRYPTO 2019, Part III*, pages 733–764, 2019.

[34] Jiaheng Zhang, Weijie Wang, Yinuo Zhang, and Yupeng Zhang. Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. Cryptology ePrint Archive, Report 2020/1247, 2020. `https://eprint.iacr.org/2020/1247`.

[35] 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.