Line-Point Zero Knowledge and Its Applications

Samuel Dittmer* Yuval Ishai† Rafail Ostrovsky‡

Abstract

We introduce and study a simple kind of proof system 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 2-5 times the computation of evaluating the circuit in the clear (following an input-independent 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 [29] 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 [39]; see [43, 40, 20] 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 and reusable) interaction during an offline preprocessing phase, before the inputs are known. In

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

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

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

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 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 [30] or IOPs [8], 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" [35, 27, 19, 37], garbled circuits [24, 31], or interactive proofs [28, 46, 48], scale better to big verification tasks.

Minimizing prover complexity. Our goal is to push the advantages of non-succinct zero-knowledge proof systems to their limits, 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 physically connected 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 motivating scenario is one where the *statement* is short and simple, but is kept secret from the verifier. In this setting, which comes up in applications such as "commit-and-prove" and NISC on small inputs (which will be discussed later), the concrete overhead of "asymptotically succinct" systems is 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 [21]. 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. [12], 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 [46, 48, 47]. 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) [42, 36] and its vector variant (VOLE) [3]. 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 [13, 20]. This is motivated by recent techniques for securely realizing pseudorandom instances of (V)OLE with sublinear communication and good concrete overall cost [13, 14, 44, 15, 17]. However, these protocols for zero knowledge from (V)OLE still suffered from a high concrete overhead. For instance, the protocol from [20] requires 44 instances of OLE for each multiplication gate. Recent and concurrent works by Weng et al. [45] and Baum et al. [7] improved this state of affairs. We will discuss these works 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. [13], with improvements in [14, 44], 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 [13, 20]. In particular, previous protocols involve multiple VOLE instances and have a large (constant) overhead in communication and computation compared to the circuit size.

The goal of reducing NIZK to a single instance of VOLE motivates the key new tool we introduce: 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 $\mathbb{F}$ of length (or dimension) n. We define the LPZK model formally along with more refined cost metrics in Section 2.1.

Information-theoretic LPZK construction. We start by showing the existence of an LPZK for arithmetic circuit satisfiability (an NP-complete problem), where the dimension n and computational costs scale linearly with the circuit size.

Theorem 1.1 (LPZK for arithmetic circuit satisfiability)**.** *For any NP-relation $R(x, y)$ and finite 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 have an LPZK over $\mathbb{F}$ with dimension $n = O(|C|)$, soundness error $\varepsilon = O(1/|\mathbb{F}|)$, and where the prover and verifier can be implemented by arithmetic circuits of size $O(|C|)$.*

As an information-theoretic proof system, LPZK can be viewed as a simple instance of a (1-round) zero-knowledge *linear interactive proof* (LIP) [10], in which the verifier sends a single field element to the prover. Theorem 1.1 implies the first such system that is zero knowledge even against a *malicious verifier*.

From LPZK to NIZK over random VOLE. It is easy to convert an LPZK into an NIZK protocol in the *rVOLE-hybrid model*, namely with a trusted setup in which 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 $\mathbf{a}'\alpha + \mathbf{b}'$. This uses a standard reduction from VOLE to rVOLE; see Section 2.2 for details. We refer to the length of the vectors $\mathbf{a}', \mathbf{b}'$ as the *rVOLE length*.

The rVOLE setup, whose efficient implementation will be discussed later, allows the prover to compress the LPZK proof by eliminating entries that can be picked at random independently of the input. Using this and other optimizations, we obtain an information-theoretic NIZK protocol in the rVOLE-hybrid model with the following concrete efficiency features.

Theorem 1.2 (NIZK over a single random VOLE)**.** *Fix an integer $t \geq 1$. There exists an (unconditional, perfect zero-knowledge) 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 the following security and efficiency features:*

- **Soundness error:** $\varepsilon = 2t/|\mathbb{F}|$*;*
- **Communication:** $k + k' + (2 + \frac{1}{t})m$ *field elements from P to V;*
- **rVOLE length:** $n = k + 2m$ *field elements;*
- **Computation:** *Assuming the cost of field additions is negligible compared to multiplications, the computation of the prover is less than* 4 *times the cost of evaluation in the clear, and the computation of the verifier is less than* 5 *times the cost of evaluation in the clear.*

We give a more precise analysis of the computation cost in terms of the cost of additions and multiplications in § 4.4. In Appendix A, we combine this analysis with a consideration of circuit structure and computer architecture to show that, for reasonable choices of parameters, both the prover and verifier computation are between 2 and 5 times the cost of evaluation in the clear, depending on the number of operations of each kind.

VOLE instantiations. The random VOLE required by Theorem 1.2 can be instantiated in a variety of ways. For instance, one could use a 2-message protocol in the CRS model based on Paillier's encryption scheme, which yields *statistical* dv-NIZK arguments for NP from the DCRA assumption [20]. Other efficient VOLE implementations under different assumptions appear in [3, 23, 6]. In terms of asymptotic efficiency, random VOLE can be implemented with constant multiplicative computational overhead under plausible variants of the learning parity with noise (LPN) assumption over big fields [3, 13]. From a concrete efficiency viewpoint, the most appealing current VOLE implementations rely on pseudorandom correlation generators (PCGs) [13, 14, 44]. A PCG for VOLE enables a "silent" generation of a long random VOLE correlation by locally expanding a pair of short, correlated seeds. This local expansion can be done in near-linear or even linear time, and may be carried out in an offline phase before the statement is known. The secure generation of the correlated seeds can also be done by a concretely efficient, low-communication protocol. Optimized pseudorandom *function* analogs of PCG that enable random access to the outputs of a virtually unbounded VOLE correlation were recently considered in [17]. The above approaches generally lead to a *preprocessing* NIZK, where both the verifier and the prover are fixed in advance. However, using 2-round protocols for VOLE with security against malicious receivers [20, 14], LPZK can be compiled into dv-NIZK protocols in which the same (short) verifier message can be used by different provers.

1.2 Improving proof size in the random oracle model

Inspired by the concurrent[1] work of Weng et al. [45], 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 [45] and our work.

Theorem 1.3 (NIZK over random VOLE in the ROM)**.** *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:** $\varepsilon = \frac{2}{|\mathbb{F}|} + \frac{\ell}{|\mathbb{F}|^r}$*;*
- **Communication:** $k + k' + m + 2r$ *field elements from P to V;*
- **rVOLE length:** $n = k + m + r$ *field elements;*
- **Computation:** *Computation of $O(r|C|)$ field operations and 1 cryptographic hash call (from $\mathbb{F}^m \to \mathbb{F}^{mr}$) for both the prover and the verifier.*

1.3 Reusable NISC from LPZK via certified VOLE

A *non-interactive secure computation* (NISC) protocol [34] 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 either party 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 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 [39, 34, 2, 41] are *not* fully reusable, and this is in some sense inherent [20].

Chase et al. [20] recently showed how to realize reusable NISC by using parallel instances of VOLE instead of OT. This can be seen as a natural extension of the LPZK model, where the receiver randomly encodes its NISC input x into multiple points α_i and the sender randomly encodes is input y into corresponding lines $\mathbf{v}_i(t)$. Here reusability refers to fixing the VOLE inputs (points) α_i generated by an honest receiver on input x and reusing them in multiple interactions with a malicious sender.

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 [32, 4], is that it "natively" supports simple *arithmetic* computations over the VOLE field. This is contrasted with NISC protocols over OT [34, 2, 41], which apply to Boolean circuits and are expensive to adapt to arithmetic computations.

We provide an alternative construction of reusable NISC over VOLE that uses LPZK to protect against malicious senders. Our approach significantly simplifies the protocol from [20] and results in much better concrete constants.

[1] Most of the present work was done concurrently and independently of [45]. We explicitly point out the improvements that are based on ideas from [45].

NISC for bounded inner product. To illustrate the concrete efficiency potential of our NISC technique, we optimize it for a simple 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 of length 1000 over $\mathbb{F}$, with $|\mathbb{F}| \approx 2^{64}$, sender L_2 norm bounded by 2^{32} and the entries of the sender's vector each bounded by 1024, our protocol requires 1002 instances of VOLE with a total of 21,067 entries and communication of 36,139 field elements (roughly 290 kB) after the offline generation of VOLE instances. Given recent methods for "silent" generation of multiple VOLE instances [14, 44, 15, 17], 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 realizing LPZK.

Constructing information-theoretic LPZK proofs. Our information-theoretic LPZK construction follows the general template of similar kinds of proof systems: the verification circuit is evaluated in two different ways that depend on secret randomness picked by the verifier, and the verifier accepts if the two evaluations are consistent. Zero knowledge is obtained by masking the values revealed to the verifier using randomness picked by the prover. This high level approach was used in previous information-theoretic zero-knowledge proof systems (such as succinct zero-knowledge linear PCPs [5, 33, 26, 10]), actively secure computation protocols (such as the SPDZ line of protocols [9, 22]), and circuits resilient to additive attacks [25]. Our LPZK systems most closely resemble the "homomorphic MAC" approach used for actively secure computation in the preprocessing model [9, 22], but differ in the low-level details.

More concretely, 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: 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 a_1.

We can "add" two encrypted wires $v_1(t) = a_1 t + b_1$ and $v_2(t) = a_2 t + b_2$ 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 $a_1 a_2 t + b$, 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 $b_3 t$, they can write $v_1(t)v_2(t) = tv_3(t) + v_4(t)$, with $v_3(t) = a_1 a_2 t + (b_1 a_2 + b_2 a_1 - b_3)$ and $v_4(t) = b_3 t + b_1 b_2$, so that $v_3(t)$ is the desired encryption of $a_1 a_2$ and satisfies $v_3(t) = (v_1(t)v_2(t) - v_4(t))/t$. The verifier learns $v_i(\alpha)$, for $1 \leq i \leq 4$ from the LPZK, and accepts if

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

and rejects otherwise. 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 LPZK to NIZK over random VOLE. These take advantage of special features of the LPZK to further 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 $\mathbf{b}'$ in the random VOLE 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 the technical sections (cf. Theroem 4.1, a refinement of Theorem 1.1) we use length parameters (n, n', n'') to give a more refined complexity measure for LPZK that takes the above optimizations into account. Here n denotes the total LPZK dimension (where $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$), n' denotes the number of entries in $\mathbf{a}$ and $\mathbf{b}$ for which the first and second optimization above do *not* apply, and by n'' denotes 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 multiplication 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 multiplication 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 *certified* variant of VOLE. This primitive is useful for invoking several parallel instances of VOLE while assuring the receiver 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 a weaker construction, *distributional VOLE with equality constraints*. This construction allows us to move all inputs to C to a single VOLE instance. The sender and receiver then prove that C is satisfied using LPZK NIZK.

This weaker variant, which we call eVOLE, is *distributional*, because it requires the VOLE inputs from the receiver to be chosen independently and uniformly at random. In general certified VOLE, which we call cVOLE, we use two additional evaluation points α, β, and perform an affine shift to the receiver's inputs, replacing $(\alpha_1, \ldots, \alpha_n)$ with $(\alpha + \alpha_1, \ldots, \alpha + \alpha_n, \alpha, \beta)$.

This forces all receiver inputs to be uniformly random, and every input besides β is independent of β. We move all inputs to C to the VOLE instance with receiver input β, and use the VOLE instance with input α to reverse the affine shift of the receiver's inputs.

From certified VOLE to NISC. Following [20], our NISC protocol is obtained from certified VOLE in a conceptually straightforward way: we start with existing protocols for arithmetic branching programs [32, 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 [20]; 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 concurrent work

In concurrent work, Weng, Yang, Katz and Wang [45] and Baum, Malozemoff, Rosen and Scholl [7] design and implement two concretely efficient VOLE-based ZK protocols, which they call Wolverine and Mac'n'Cheese, respectively. Both protocols have an online phase that can be made non-interactive in the random oracle model. These protocols share some high-level features with each other and with our LPZK-based protocol, but there are some important differences in scope and design, and each work offers unique optimizations and extensions.

The following comparison refers to the online phase of the protocols, once a random VOLE correlation has already been generated. In the random oracle model, our LPZK protocol requires at least 2 times less communication and computation per multiplication gate than either Wolverine or Mac'n'Cheese, while offering the possibility of eliminating entirely the "cryptographic" overhead of the online phase. Our information theoretic LPZK NIZK requires similar communication to the best variant of Wolverine described in [45], with at least 2-3 times less multiplications, and no calls to a cryptographic hash function. We give additional details below.

Baum et al. [7] provide an additional optimization that allows stacking disjunctive statements, similar to the "stacked garbling" improvement over ordinary garbled circuits [31]. The efficiency gains from this optimization depend on the structure of the circuit, of course, and so the only comparison we make to Mac'n'Cheese is for general arithmetic circuits on a per-gate basis.

Weng et al. [45] 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.

We draw attention to one low-level detail in the distribution of information between the prover and verifier that distinguishes our approach. In LPZK, the prover holds a line $\mathbf{v}(t) = \mathbf{a}t + \mathbf{b}$ and the verifier queries a single point $\mathbf{v}(\alpha)$ on that line. In both [45] and [7], the prover instead holds $\mathbf{a}$, the verifier holds $\mathbf{b}$ and α, and the prover learns $\mathbf{v}(\alpha)$. Our construction better exploits the algebraic structure of VOLE, allowing local computation by the verifier to reduce the communication cost of verifying a multiplication gate.

We also note our work is the only to consider reusable NISC over VOLE, in addtion to NIZK.

For NIZK, both in the random oracle model and in the information theoretic setting, we compare our work to Wolverine and Mac'n'Cheese in communication cost and prover and verifier communication complexity.

Communication: Our LPZK protocol requires $1 + o(1)$ field elements per multiplication gate in the random oracle model, and $2 + 1/t + o(1)$ elements per multiplication gate for information theroetic NIZK, for some parameter $t \in \mathbb{Z}$ that affects soundness error. For large fields, we can choose $t = \log |\mathbb{F}|$, so the $1/t$ term is in practice small.

Mac'n'Cheese [7] requires $3+o(1)$ elements of communication per multiplication gate. Wolverine [45] requires $2 + o(1)$ elements of communication per multiplication gate, while a variant discussed in [45] that is more computationally efficient requires $4 + o(1)$ elements per multiplication gate. In all cases, the $o(1)$ term is controlled by the growth of circuit size relative to the input size.

Our information theoretic NIZK protocol has comparable communication to the Wolverine protocol in the random oracle setting, and our random oracle NIZK protocol has 2 times less communication.

Prover computation: Our information theoretic LPZK-based NIZK over random VOLE requires prover computation of 3 multiplications per multiplication gate.

Our NIZK protocol in the random oracle model requires $(2r+3)$ multiplications per multiplication gate, and a single call to a cryptographic hash function $H : \mathbb{F}^m \to \mathbb{F}^{mr}$, where m is the number of multiplication gates. The parameter $r \in \mathbb{N}$ affects soundness error, but for large fields we can set $r = 1$. The cryptographic hash function H can be evaluated in a streaming fashion to reduce memory costs.

The most computationally efficient variant in Wolverine [45] requires $(4r + 6)$ multiplications per multiplication gate, 2 cryptographic hash function calls $H_1 : \mathbb{F} \to \mathbb{F}^r$ per multiplication gate, and one more cryptographic hash function call $H_2 : \mathbb{F}^m \to \mathbb{F}^r$ for the entire protocol, with m as above. Mac'n'Cheese [7] requires 13 multiplications per multiplication gate and 3 cryptographic hash function calls $H_3 : \mathbb{F}^{k+1} \to \mathbb{F}$ per multiplication gate, where k is the number of messages sent so far in the protocol.

Verifier computation: Our information theoretic LPZK NIZK protocol requires verifier computation of 4 multiplications per multiplication gate.

In the random oracle model, our LPZK protocol requires $(r + 3)$ multiplications per multiplication gate, and a single cryptographic hash call. As above, for large enough field size, we can set $r = 1$ and the cryptographic hash function $H : \mathbb{F}^{2m} \to \mathbb{F}^{mr}$ can be evaluated in a streaming fashion to reduce memory costs.

The most computationally efficient variant in Wolverine [45] requires $7r$ multiplications per multiplication gate, 2 cryptographic hash function calls per multiplication gate, and one more cryptographic hash function call for the entire protocol, with the hash functions H_1, H_2 as above. Mac'n'Cheese [7] requires 10 multiplications per multiplication gate and 3 cryptographic hash function calls per multiplication gate, with the hash function H_3 as above.

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* [10] that restricts the verifier to sending a single field element.

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

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 $\Pr[\mathbf{v}(t) \xleftarrow{R} \mathsf{Prove}(\mathbb{F}, C, w) : \mathsf{Verify}(\mathbf{F}, C, \alpha, \mathbf{v}(\alpha)) = \mathsf{acc}] = 1$.
- **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 $\{\mathbf{v} : \mathbf{v} \xleftarrow{R} \mathsf{Sim}(\mathbb{F}, C, \alpha)\}$ 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.

Proof of Knowledge. For simplicity, we focus here on (reusable) soundness and ignore the additional *proof of knowledge* property. However, the LPZK systems we construct all satisfy this stronger notion of soundness (see [10] a definition of proofs of knowledge in the context of linear proof systems). More formally, there is an efficient *extractor* that can extract a valid witness from any (maliciously generated) line that makes the verifier accept with $> \varepsilon$ probability.

Computational LPZK. The above definition considers our main *information-theoretic* flavor of LPZK, with statistical soundness and perfect zero knowledge. Computational variants of LPZK can be defined analogously. In particular, we will later consider computationally sound LPZK in the random oracle model, which bounds the number of oracle queries made by a malicious prover.

Complexity measures for LPZK: (n, n', n'')-LPZK. As noted in § 1.4, 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 verifier 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 [13, 20], 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 (From LPZK to NIZK)**.** *Given (n, n', n'')-LPZK over $\mathbb{F}$ with soundness error ε, there is an NIZK protocol that uses a single instance of random VOLE of length $n - n''$ and requires communication of $n' + n''$ field elements from the prover to the verifier.*

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

We recall a simple self-reduction property of VOLE (see e.g. [13]) 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. The security completeness, soundness, and zero knowledge properties of the above NIZK protocol are inherited directly from the corresponding properties of the LPZK. □

UC security. While we only consider here a standard standalone security definition for NIZK proofs [29, 11], all of our LPZK-based NIZK protocols are in fact *UC-secure* NIZK protocols (e.g., in the sense of [18]) in the rVOLE-hybrid model. This is the typical situation for information-theoretic protocols.

Using a corruptible random VOLE functionality. When using a pseudorandom correlation generator (PCG) for generating the random VOLE correlation with sublinear communication complexity [13, 16, 44], 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 [13] 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 flavor of ZK, which is meaningful even for finite functions. Note that our LPZK construction is adapted from this single gate example, rather than directly built up from it, so this proof and the proof in Section 4 can be read independently.

A commit-and-prove protocol for the above relation R has the same syntax as LPZK, and should satisfy the following loosely stated properties (see, e.g., [38] for a formal definition).

- **Completeness.** If the prover runs honestly on (x, y, z) such that $z = xy$, then the verifier always accepts.
- **Binding.** There is a determinstic extractor that given a line picked by a (potentially malicious) prover outputs effective inputs (x^*, y^*, z^*) such that the following holds. Any attempt of the prover to "explain" a different input triple (x', y', z') (by revealing its randomness) would lead to an inconsistent verifier view, except with the binding error probability (over the choice of α).
- **Soundness.** For any malicious prover, if the extracted values (x^*, y^*, z^*) satisfy $z^* \neq x^*y^*$, then the verifier rejects except for the soundness error probability (over the choice of α).
- **Zero knowledge.** For any choice of α, the verifier's evaluation on an honestly generated line can be simulated without knowing x, y, z.

Random evaluation of the line picked by a prover (even a malicious one) effectively commits the prover to unique values of x, y, z, in the sense that except for the binding error probability it cannot reveal randomness that consistently explains different (x', y', z')), and moreover the verifier rejects unless $z = xy$ (except with soundness error probability).

3.1 Protocol

We construct our commit-and-prove protocol for the relation $R(x, y, z) := xy - z$ as a $(5, 4, 1)$-LPZK over $\mathbb{F}$ with binding and soundness error $\leq 2/|\mathbb{F}|$.

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 a 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. 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

$$\begin{aligned} v_1(t)v_2(t) - tv_3(t) - v_4(t) - v_5(t) = {} & (x^*y^* - z^*)t^2 \\ & + \big(x^*b_2^* + y^*b_1^* - b_3^* - a_4^*\big)t \\ & + b_1^*b_2^* - b_4^* - b_5^* \end{aligned}$$

is a non-trivial polynomial in t of degree 2. This polynomial 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 simulate 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. $\square$

3.3 Complexity

Communication. The communication cost of implementing $(5, 4, 1)$-LPZK over random VOLE 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 $(x + b_1)(y + b_2) - z - b_1b_2$).

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

Remark 3.1 (Extension to general arithmetic circuits)**.** We can convert this protocol to an LPZK for arithmetic circuits by placing all intermediate wire values into $\mathbf{a}$ and running the commit-and-prove protocol for each multiplication gate. The binding property ensures that the wire values match the values x, y, z for which the prover demonstrates $xy = z$. For all multiplication gates whose inputs are intermediate values, the verifier no longer needs to learn the values v_1, v_2 masking the inputs x, y from VOLE, but can instead compute them as a linear combination of previous multiplication gate outputs. This therefore gives a communication cost of 3 field elements per multiplication gate. We improve on this by batching together verification messages into blocks of size t, as we show in the next section.

4 Information-Theoretic LPZK for Arithmetic Circuits

In this section we prove Theorem 1.1 by describing an information-theoretic LPZK for proving the satisfiability of arithmetic circuits.

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

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

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

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 $1 \le j \le 2m + k'$ and $1 \le i \le m$, 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 $2t/|\mathbb{F}|$, 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 $a_0 = 1$ and $b_0 = 0$. 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

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

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

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

4.3 Proof

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

Theorem 4.1 (LPZK for arithmetic circuit satisfiability)**.** *For any NP-relation $R(x, y)$ and finite 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 computation of the prover is less than 4 times the cost of evaluation in the clear, and the computation of the verifier is less than 5 times the cost of evaluation in the clear.*

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 $b_{4i-3} b_{4i-2} \neq 0$, 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

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

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

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

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 $C(\mathbf{w}^*) \neq \mathbf{0}$. 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

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

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

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

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 $2T(C) + (4 - \frac{1}{t})m\ T(*) + (k + 6m)\ T(+)$, which assuming the cost of additions is negligible, is less than 4 times the cost of evaluation in the clear.

Verifier computation: Similarly, the verifier's work is $T(C) + \left((5 - \frac{1}{t})m + k\right)\ T(*) + 2m\ T(+)$, which assuming the cost of additions is negligible, is less than 5 times the cost of evaluation in the clear. Included in this cost is a $2m + k\ T(*)$ term from the conversion from random to fixed VOLE.

Remark 4.1. Assuming that the random VOLE instances are already precomputed, or alternatively that such compressed instances can be "unpacked" in a streaming fashion, our NIZK protocol does not require the vectors $\mathbf{a}, \mathbf{b}, \mathbf{v}$ to be computed or stored in their entirety. Instead, the entries corresponding to each gate can be computed on the fly. 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 can be made sublinear in the circuit size[2] in the circuit size (see [13, 14, 44, 45, 17] for possible trade-offs and optimizations), 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 LPZK in the Random Oracle Model

In the section we prove Theorem 1.3, which gives an improved NIZK over random VOLE in the random oracle model (ROM). This follows by applying the compiler of Lemma 2.1 to the LPZK in following theorem.

Theorem 5.1 (LPZK in the ROM)**.** *For any positive integer r, there exists an LPZK in the ROM for arithmetic circuit satisfiability, with the following size parameters (n, n', n'') and soundness error. If C has k inputs, k' outputs, and m multiplication gates, we have $n = k + k' + m + 2r$, $n' = k$, $n'' = k' + m + 2r$. For any malicious prover making ℓ calls to a random oracle $H : \mathbb{F}^m \to \mathbb{F}^{mr}$, the soundness error is $\varepsilon = \frac{2}{|\mathbb{F}|} + \frac{\ell}{|\mathbb{F}|^r}$. Moreover, the computation of both the prover and the verifier consists of $O(r|C|)$ field operations and a single call to H.*

At a high level, the LPZK construction begins by setting $\mathbf{a}$ equal to the wire values in the circuit evaluation, and choosing $\mathbf{b}$ at random, as in § 4.2. To convince the verifier that all multiplication gates have been evaluated correctly, the prover must show that a sequence of quadratic polynomials whose coefficients are determined by $\mathbf{a}$ and $\mathbf{b}$ each have leading term zero, i.e. that this sequence of quadratics is actually a sequence of linear polynomials. The protocol uses LPZK to reveal to the verifier a vector $\mathbf{s}$ of the evaluations of those quadratics at α and then the prover must show they have vectors $\mathbf{y}, \mathbf{z}$ such that $\mathbf{s} = \mathbf{y}\alpha + \mathbf{z}$. In other words, the prover must show that $\mathbf{y}, \mathbf{z}$ as VOLE inputs give $\mathbf{s}$ as a VOLE output.

To do this, prover and verifier choose a random $r \times m$ matrix $M := H(\mathbf{w})$ by evaluating a random oracle H on the prover messages $\mathbf{w}$ sent during the protocol. Then after adding random masks from the LPZK to $\mathbf{y}, \mathbf{z}, \mathbf{s}$, the verifier cheks that $M\mathbf{s} = M\mathbf{y}\alpha + M\mathbf{z}$.

5.1 The LPZK construction

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 reduces to a shorter $\hat{\mathbf{v}}$ that is used as VOLE input. For $0 \le i \le k + k' + 4m$, the prover defines a_i and b_i identically to their definitions in § 4.2, except each entry a_{k+4j} is chosen uniformly

[2]To this end one can either use the "primal" PCG for VOLE from [13, 44], which has at most quadratic stretch, or store multiple seeds of a higher-stretch PCG where each seed is expanded when processing a different (sublinear-size) segment of the circuit. Alternatively, one could use a pseudorandom correlation function [17].

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 partial redundancy between the $k+4j-1$th and $k+4j$th entry is to preserve the indexing of § 4.2 while enabling the reconstruction of v_{k+4i-1} from v_{k+4i} and the value of $a_{k+4j} - a_{k+4j-1}$, as described below.

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. These $m+2r$ entries, in other words, can be sent from the prover to the verifier directly without require any VOLE overhead.

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 a random oracle, 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 $1 \leq i \leq m$, 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 through 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 $1 \leq i \leq k$, we take $\hat{a}_i = a_i$ and $\hat{b}_i = b_i$. For $1 \leq i \leq m$ we take $\hat{a}_{k+i} = a_{k+4i}$ and $\hat{b}_{k+i} = b_{k+4i}$. For $1 \leq i \leq r$, 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 $1 \leq i \leq k'$, we set $\hat{b}_{k+m+r+i} = \mathbf{r}_{2m+i} \cdot \mathbf{b}$. For $1 \leq i \leq m$, we set $\hat{b}_{k+k'+m+r+i} = w_i = a_{k+4i-1} - a_{k+4i}$. Finally, for $1 \leq i \leq 2r$, 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 $1 \leq i \leq m$,

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

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

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

$$y_\alpha := (\mathbf{v}[k + k' + 5m + r + 1 : k + k' + 5m + 2r])$$

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

and returns rej unless $y\alpha + z = s$. Then for $1 \leq i \leq k'$, 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.

Remark 5.1. As with information-theoretic LPZK, we can convert the algorithm above to a streaming algorithm requiring $O(r)$ local memory beyond what would be required in a plaintext evaluation of the circuit.

The conversion is similar to the information-theoretic case, although we now have to also evaluate the random oracle H in a streaming fashion. We choose a pair of random oracles H_1, H_2, where $H_1 : \mathbb{F}^m \to \mathbb{F}$ can be evaluated in a streaming fashion, and $H_2 : \mathbb{F} \times \{1, \ldots, m\} \to \mathbb{F}^r$, and set the ith column of $H(\mathbf{w})$ equal to $H_2(H_1(\mathbf{w}), i)$.

An honest prover computes $H_1(\mathbf{w})$ before performing the rest of the protocol, and sends this along with the rest of the proof. Then the prover can compute $H(\mathbf{w})\mathbf{y}^T$ and $H(\mathbf{w})\mathbf{z}^T$ in a streaming fashion while executing the LPZK protocol, and the verifier can likewise compute $H(\mathbf{w})\mathbf{s}^T$ and verify the value of $H_1(\mathbf{w})$ sent by the prover based on the messages w_i sent during the protocol.

5.2 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.

5.3 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. This gives a total of $k + k' + m + 2r$ field elements of communication.

The prover performs 3 multiplications per multiplication gate, while the verifier performs 2 multiplications. The prover must multiply $H(\mathbf{w})$ by the vectors $\mathbf{y}^T$, $\mathbf{z}^T$, while the verifier does the same multiplication by $\mathbf{s}^T$. This matrix multiplication requires mr multiplications, giving an amortized cost of $2r$ multiplications per multiplication gate for the prover and r multiplications per multiplication gate for the verifier. Finally, the verifier must perform an additional $k + m + r$ multiplications to convert from random to fixed VOLE. This gives a total cost of $3 + 2r$ multiplications per multiplication gate for the prover, $3 + r$ for the verifier, and the single evaluation of H for both parties.

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 [20].

6.1 NISC definition

We start by giving a simplified definition of reusable NISC over VOLE, which strengthens the definition from [20]. 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. (The lines are the sender's VOLE inputs and the points are the receiver's VOLE inputs.)

At a high level, reusable security is ensured by preventing a malicious sender from making the receiver's output depend on its input beyond the dependence allowed by the ideal functionality. This is contrasted with OT-based NISC protocols, where the sender can learn a receiver's OT input by starting from an honest strategy and replacing one of the sender OT inputs by a random one.

We formulate the NISC definition for arithmetic functions f defined over an arbitrary field $\mathbb{F}$, where the security error vanishes with the field size. For simplicity we consider a single function f and information-theoretic security. The definition can be naturally generalized to take a function description as input and allow computational security.

Definition 6.1 (Reusable arithmetic NISC)**.** A reusable non-interactive secure computation (NISC) protocol over VOLE for an arithmetic function $f : \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^\ell$ is a triple of algorithms $(\mathit{R1}, \mathit{S}, \mathit{R2})$ with the following syntax:

- $\mathit{R1}(\mathbb{F}, \mathbf{x})$ is a PPT algorithm that, given an input $\mathbf{x} \in \mathbb{F}^n$, outputs points $(\alpha_1, \ldots, \alpha_{n'}) \in \mathbb{F}^{n'}$ and auxiliary information aux.
- $\mathit{S}(\mathbb{F}, \mathbf{y})$ is a PPT algorithm that, given $\mathbf{y} \in \mathbb{F}^m$, outputs n' pairs of vectors $\mathbf{a}_i, \mathbf{b}_i \in \mathbb{F}^s$, each specifying an affine line $\mathbf{v}_i(t) := \mathbf{a}_i t + \mathbf{b}_i$.
- $\mathit{R2}(\mathbb{F}, \mathsf{aux}, (\mathbf{v}_1, \ldots, \mathbf{v}_{n'}))$ is a polynomial-time algorithm that, given auxiliary information aux and evaluations $\mathbf{v}_i$, outputs either $\mathbf{z} \in \mathbb{F}^\ell$ or rej.

The algorithms $(\mathit{R1}, \mathit{S}, \mathit{R2})$ should satisfy the following security requirements:

- **Completeness**. When both parties follow the protocol, running the above algorithms in sequence, with $\mathbf{v}_i = \mathbf{v}_i(\alpha_i)$, results in the output $\mathbf{z} = f(\mathbf{x}, \mathbf{y})$.
- **Reusable ε-security against malicious sender**. There exists a polynomial-time extractor algorithm Ext such that for any field $\mathbb{F}$ and lines $\mathbf{v}_i^*(t) := \mathbf{a}_i^* t + \mathbf{b}_i^*$, the output of $\mathsf{Ext}(\mathbb{F}, (\mathbf{a}_1^*, \mathbf{b}_1^*), \ldots, (\mathbf{a}_{n'}^*, \mathbf{b}_{n'}^*))$ is $\mathbf{y}^* \in \mathbb{F}^m \cup \{\perp\}$ such that the following holds: for every honest receiver's input $\mathbf{x} \in \mathbb{F}^n$, the receiver's output when interacting with malicious sender strategy $\mathbf{v}_i^*(t)$ is equal to $f(\mathbf{x}, \mathbf{y}^*)$ except with $\leq \varepsilon$ probability over the receiver's randomness. Here we assume that the output on $\perp$ is rej. Unless otherwise specified, we assume $\varepsilon \leq O(1/|\mathbb{F}|)$. We will also use a **random-input** variant of the above definition, where the probability is over both the receiver's randomness and a *uniformly random* choice of $\mathbf{x} \in \mathbb{F}^n$.
- **Perfect security against malicious receiver**. There exist a polynomial-time extractor algorithm Ext and PPT simulator algorithm Sim such that, for any field $\mathbb{F}$ and malicious receiver points $\alpha_1^*, \ldots, \alpha_{n'}^* \in \mathbb{F}$, the extractor outputs an effective input $\mathbf{x}^* = \mathit{Ext}(\mathbb{F}, (\alpha_1^*, \ldots, \alpha_{n'}^*))$, where $\mathbf{x}^* \in \mathbb{F}^n$, such that the following holds. For every honest sender's input $\mathbf{y} \in \mathbb{F}^m$, the output distribution of $\mathit{Sim}(\mathbb{F}, f(\mathbf{x}^*, \mathbf{y}))\}$ is identical to $\{(\mathbf{v}_1(\alpha_1^*), \ldots, \mathbf{v}_{n'}(\alpha_{n'}^*)) : (\mathbf{v}_1(t), \ldots, \mathbf{v}_{n'}(t)) \xleftarrow{R} \mathit{S}(\mathbb{F}, \mathbf{y})\}$.

We note that instead of allowing the receiver to output *rej*, we could instead make the receiver use a default value for the sender input and compute the output of f. However, making the receiver reject whenever it detects cheating makes protocol descriptions more natural.

The definition above does not permit the sender to transmit additional values to the receiver in the clear. In order to simplify the definition and the proofs, we note that we can realize plaintext transmission from sender to receiver as a reusable NISC protocol over VOLE. The function $f(\mathbf{x}, \mathbf{y}) := \mathbf{y}$ prints the sender input, the algorithm $\mathsf{R1}(\mathbb{F}, \mathbf{x})$ outputs random points α_1, α_2, and the sender algorithm $\mathsf{S}(\mathbb{F}, \mathbf{y})$ outputs $\mathbf{a}_i := \mathbf{0}$ and $\mathbf{b}_i = \mathbf{y}$ for $i = 1, 2$. Finally, $\mathsf{R2}(\mathbb{F}, (\mathbf{v}_1, \mathbf{v}_2))$ rejects if $\mathbf{v}_1 \neq \mathbf{v}_2$, and outputs $\mathbf{v}_1$ otherwise. The security conditions are straightforward to verify.

In the proofs below, when we refer to "sending values in the clear", we formally mean the protocol above. In actual applications, of course, we will continue to send the plaintexts directly. We use direct transmission, rather than this more involved NISC protocol, in our analysis of computation and communication complexity.

Throughout this section, whenever we desire to refer to the jth entry of a vector $\mathbf{a}_i, \mathbf{b}_i, \mathbf{v}_i$, etc, we write the entry as a_i^j, b_i^j, v_i^j, etc.

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 a more specialized form, distributional certified VOLE with equality constraints, which we write as eVOLE. In this variant of certified VOLE, the arithmetic circuit on the family of VOLEs is restricted to a single equality constraint between two coefficients from $\mathbf{a}$ vectors. In eVOLE, we require additionally that R's inputs are uniformly distributed over $\mathbb{F}$ and independent. It is straightforward to extend this result to an arbitrary set of equality constraints on terms from $\mathbf{a}$ and $\mathbf{b}$ terms, and we explain the details below.

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 and 2. We state this result as the following two lemmas.

Lemma 6.1. *A receiver R and a sender S can realize the functionality $\mathcal{F}_{eVOLE}^{(\mathbb{F})}$ with parameters (ℓ_1, ℓ_2, i, j) in the rVOLE hybrid model with 2 instances of random VOLE of total length $\ell_1 + \ell_2 + 2$ and communication of 3 field elements from sender to receiver, in addition to any communication cost for transforming random VOLEs to the VOLEs with inputs $(\hat{\mathbf{a}}_1, \hat{\mathbf{b}}_1, \hat{\mathbf{a}}_2, \hat{\mathbf{b}}_2)$.*

Lemma 6.2. *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 $q_{\mathbf{a}}$ inputs from the $\hat{\mathbf{a}}_i$'s, $q_{\mathbf{b}}$ inputs from the $\hat{\mathbf{b}}_i$'s, q' outputs, and m multiplication gates, these VOLE instances have total length*

$$2m + 6q_{\mathbf{a}} + 7q_{\mathbf{b}} + \sum_{i=1}^{k} \ell_i,$$

Figure 1: Distributional certified VOLE with equality constraints

Functionality $\mathcal{F}_{eVOLE}^{(\mathbb{F})}$: Distributional certified VOLE with equality constraint

Parametrized by a finite field $\mathbb{F}$, length parameters (ℓ_1, ℓ_2), and integers i, j with $1 \leq i \leq \ell_1$ and $1 \leq j \leq \ell_2$.

- R sends $\mathbf{x} := (\alpha, \beta)$ to $\mathcal{F}_{eVOLE}^{(\mathbb{F})}$
  // Receiver security is only required for random inputs
- S sends $\mathbf{y} := (\hat{\mathbf{a}}_1, \hat{\mathbf{b}}_1, \hat{\mathbf{a}}_2, \hat{\mathbf{b}}_2)$ to $\mathcal{F}_{eVOLE}^{(\mathbb{F})}$, where $\hat{\mathbf{a}}_i, \hat{\mathbf{b}}_i \in \mathbb{F}^{\ell_i}$
- $\mathcal{F}_{cVOLE}^{(\mathbb{F})}$ verifies that $\hat{a}_1^i = \hat{a}_2^j$.
- If the input does not pass verification, the ideal functionality sends $\perp$ to R. Otherwise, $\mathcal{F}_{cVOLE}^{(\mathbb{F})}$ computes $\hat{\mathbf{v}}_1 := \hat{\mathbf{a}}_1\alpha + \hat{\mathbf{b}}_1$ and $\hat{\mathbf{v}}_2 := \hat{\mathbf{a}}_2\beta + \hat{\mathbf{b}}_2$ and sends $f(\mathbf{x}, \mathbf{y}) := (\mathbf{v}_1, \mathbf{v}_2)$ to R.

and the protocol requires communication of

$$(2 + \tfrac{1}{t})m + q' + 8q_{\mathbf{a}} + 9q_{\mathbf{b}} + 2\sum_{i=1}^{k} \ell_i$$

field elements from sender to receiver.

Figure 2: 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 $\mathbf{x} := (\alpha_1, \ldots, \alpha_k) \in \mathbb{F}^k$.

Send phases

- S sends $\mathbf{y} := (\hat{\mathbf{a}}_1, \ldots, \hat{\mathbf{a}}_k, \hat{\mathbf{b}}_1, \ldots, \hat{\mathbf{b}}_k)$ to $\mathcal{F}_{cVOLE}^{(\mathbb{F})}$, where $\hat{\mathbf{a}}_i, \hat{\mathbf{b}}_i \in \mathbb{F}^{\ell_i}$
- $\mathcal{F}_{cVOLE}^{(\mathbb{F})}$ verifies that $(\hat{\mathbf{a}}_1, \ldots, \hat{\mathbf{a}}_k, \hat{\mathbf{b}}_1, \ldots, \hat{\mathbf{b}}_k)$ 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 $\hat{\mathbf{v}}_i := \hat{\mathbf{a}}_i\alpha_i + \hat{\mathbf{b}}_i$ and sends $f(\mathbf{x}, \mathbf{y}) := (\hat{\mathbf{v}}_1, \ldots, \hat{\mathbf{v}}_k)$ to R.

6.2.2 The protocols

eVOLE. eVOLE is a special case of reusable arithmetic NISC where the receiver has no inputs, and $\mathsf{R1}(\mathbb{F})$ outputs uniformly random and independent points (α, β), and stores their values as the auxiliary information $\mathsf{aux} := (\alpha, \beta)$. The sender's input $\mathbf{y} := (\hat{\mathbf{a}}_1, \hat{\mathbf{b}}_1, \hat{\mathbf{a}}_2, \hat{\mathbf{b}}_2)$ is two existing VOLE inputs, and the algorithm $\mathsf{S}(\mathbb{F}, \mathbf{y})$ outputs vectors $(\mathbf{a}_1, \mathbf{b}_1, \mathbf{a}_2, \mathbf{b}_2)$ whose first ℓ, ℓ, ℓ', and ℓ' coordinates are equal to $(\hat{\mathbf{a}}_1, \hat{\mathbf{b}}_1, \hat{\mathbf{a}}_2, \hat{\mathbf{b}}_2)$, respectively. The remaining values are defined as $a_1^{\ell+1} := \hat{b}_2^j$, $a_2^{\ell'+1} := \hat{b}_1^i$, with $b_1^{\ell+1}$ and $b_2^{\ell'+1}$ chosen uniformly at random. In addition, the sender sends the value $b_1^{\ell+1} - b_2^{\ell'+1}$ in the clear.

In the algorithm $\mathsf{R2}(\mathbb{F}, \mathsf{aux}, (\mathbf{v}_1, \mathbf{v}_2))$, the receiver tests whether

$$\beta v_1^i - \alpha v_2^j + v_1^{\ell+1} - v_2^{\ell'+1} = b_1^{\ell+1} - b_2^{\ell'+1}.$$

The receiver rejects if the test fails, and otherwise outputs the vectors $\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2$ obtained by deleting the last element from $\mathbf{v}_1, \mathbf{v}_2$.

This protocol can be modified to prove constraints of the form $\hat{a}_1^i = \hat{b}_2^j$ or $\hat{b}_1^i = \hat{b}_2^j$ for the same communication cost and one or two additional multiplications, respectively, by the receiver. Indeed, by multiplying $\mathbf{v}_1$ by α^{-1} or $\mathbf{v}_2$ by β^{-1}, the receiver can locally obtain the VOLE outputs $\mathbf{w}_1 := \alpha^{-1}\mathbf{b}_1 + \mathbf{a}_1$ and $\mathbf{w}_2 := \beta^{-1}\mathbf{b}_2 + \mathbf{a}_2$, and the same construction above applies to the pair $\mathbf{v}_1, \mathbf{w}_2$ or the pair $\mathbf{w}_1, \mathbf{w}_2$.

Additionally, since the eVOLE protocol transforms VOLE inputs $\hat{\mathbf{a}}_i, \hat{\mathbf{b}}_i$ for the sender into extended VOLE inputs $\mathbf{a}_i, \mathbf{b}_i$ and delivers extended VOLE outputs $\mathbf{v}_i$ to the receiver, this protocol can be implemented repeatedly on the *same two instances* of VOLE, proving c equality constraints with VOLEs of length $\ell + c$, $\ell' + c$.

cVOLE. We write the receiver's inputs as $\mathbf{x} := (\alpha_1, \ldots, \alpha_k)$. The receiver's algorithm $\mathsf{R1}(\mathbb{F}, \mathbf{x})$ generates their VOLE inputs by choosing random independent values α, β, and then outputs $(\alpha + \alpha_1, \ldots, \alpha + \alpha_k, \alpha, \beta)$.

As in eVOLE, the sender defines $a_i^j := \hat{a}_i^j$ everywhere this is defined. We give the definition of b_i^j later. Then, for each input to C from the $\hat{\mathbf{a}}_i$'s, say $\hat{a}_i^{j_1}$, the sender chooses one entry of $\mathbf{a}_{k+1}$ and one entry of $\mathbf{a}_{k+2}$, say $a_{k+1}^{j_2}$ and $a_{k+2}^{j_3}$ respectively, and uses eVOLE to prove $\hat{a}_i^{j_1} = a_{k+2}^{j_3}$ and $a_{k+1}^{j_2} = a_{k+2}^{j_3}$. Since each of the pairs $(\alpha+\alpha_i, \beta)$ and (α, β) are uniformly random and independent, the conditions for eVOLE are satisfied.

Similarly, for an input $\hat{b}_i^{j_1}$ to C, the sender chooses entries $b_{k+1}^{j_2}$, $a_{k+2}^{j_3}$ and $a_{k+2}^{j_4}$ and proves $b_i^{j_1} = a_{k+2}^{j_3}$ and $b_{k+1}^{j_2} = a_{k+2}^{j_4}$. We now define $b_i^{j_1} := \hat{b}_i^{j_1} + b_{k+1}^{j_2}$. Upon subtracting $v_{k+1}^{j_2} := a_{k+1}^{j_2}\alpha + b_{k+1}^{j_2}$ from $v_i^{j_1} := a_i^{j_1}(\alpha + \alpha_i) + b_i^{j_1}$, the receiver holds

$$\hat{v}_i^{j_1} := v_i^{j_1} - v_{k+1}^{j_2} = a_i^{j_1}\alpha_i + (b_i^{j_1} - b_{k+1}^{j_2}) = \hat{a}_i^{j_1}\alpha_i + \hat{b}_i^{j_1}.$$

After deleting unneeded entries of the $\hat{\mathbf{v}}_i$'s receiver ends with the VOLE outputs $\hat{\mathbf{v}}_i := \hat{\mathbf{a}}_i\alpha_i + \hat{\mathbf{b}}_i$, as desired. In addition, the elements $a_i^{j_1}, b_i^{j_2}, b_{k+1}^{j_2}$ have all been transferred to entries of $\mathbf{a}_{k+2}$, so the receiver and sender extend the $(k+2)$nd instance of VOLE $\mathbf{v}_{k+2}$ with a NIZK proof that C is satisfied by $\hat{\mathbf{a}}_i, \hat{\mathbf{b}}_i$.

6.2.3 Proof of Lemma 6.1

Completeness: Writing the value the sender transmits in the clear as c in the protocol, we have

$$v_1^i\beta - v_2^j\alpha + v_1^{\ell+1} - v_2^{\ell'+1} - c = (\hat{a}_1^i - \hat{a}_2^j)\alpha\beta + (a_1^{\ell+1} - \hat{b}_2^j)\alpha + (a_2^{\ell'+1} - \hat{b}_1^i)\beta + \left((b_1^{\ell+1} - b_2^{\ell'+1}) - c\right).$$

If the sender is honest, each of the four terms on the right-hand side vanish, and the receiver accepts the message, as desired.

Perfect security against a malicious receiver: The receiver has no input, so the input extractor does nothing. A malicious receiver's output $f(\mathbf{x}^*, \mathbf{y})$ is the pair of truncations $\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2$ of the VOLE outputs $\mathbf{v}_1, \mathbf{v}_2$. The simulator $\mathsf{Sim}(\mathbb{F}, f(\mathbf{x}^*, \mathbf{y}))$ obtains $\mathbf{v}_1^*, \mathbf{v}_2^*$ from $\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2$ by appending a value chosen uniformly at random from $\mathbf{F}$ to each vector.

By construction, $\mathbf{v}_i^* = \mathbf{v}_i$ on every entry except the last. For the last entry, we note that, because $\hat{b}_1^{\ell+1}$ and $\hat{b}_2^{\ell'+1}$ are chosen independently and uniformly at random, the last entries of $\mathbf{v}_i$ and $\mathbf{v}_i^*$ each follow the uniform distribution, so the output distributions are identical, as desired.

Reusable ε-security against a malicious sender: The sender's input is two sets of VOLE inputs $((\hat{\mathbf{a}}_1, \hat{\mathbf{b}}_1), (\hat{\mathbf{a}}_2, \hat{\mathbf{b}}_2))$, which are then extended by one entry each in the algorithm $\mathsf{S}(\mathbb{F}, \mathbf{y})$. The extractor $\mathsf{Ext}(\mathbb{F}, (\mathbf{a}_1^*, \mathbf{b}_1^*), (\mathbf{a}_2^*, \mathbf{b}_2^*))$ sets $\mathbf{y}^* = \perp$ if any of the conditions $a_1^i = a_2^j$, $a_1^{\ell+1} = b_2^j$, $a_2^{\ell'+1} = b_1^i$, $c = b_1^{\ell+1} - b_2^{\ell'+1}$ are false, and otherwise computes $\mathbf{y}^*$ by truncation. We write tr for the truncation-by-one operator.

By completeness, whenever $\mathbf{y}^* \neq \perp$ we have $f(\mathbf{x}, \mathbf{y}^*) = tr(\mathbf{v}^*)$ which is equal to the receiver's output under an honest run of the protocol. Therefore the probability the receiver's output is not equal to $f(\mathbf{x}, \mathbf{y}^*)$ is precisely the probability that $\mathbf{y}^* = \perp$ and the receiver does not output rej.

This occurs precisely when the expression

$$(a_1^i - a_2^j)\alpha\beta + (a_1^{\ell+1} - b_2^j)\alpha + (a_2^{\ell'+1} - b_1^i)\beta + b_1^{\ell+1} - b_2^{\ell'+1} - c = 0,$$

but at least one of the four coefficients in this bivariate polynomial expression are nonzero. For all but at most one choice of α, either fixing α determines β or the statement is identically false. Since β is independent of α, the probability that the true value of β matches the value determined by α is $1/|\mathbb{F}|$. Together, this gives a soundness error bounded above by $2/|\mathbb{F}|$.

Complexity: The total VOLE length is $\ell + \ell' + 2$, by construction.

In addition to the cost of setting up the fixed VOLE on the first ℓ, ℓ' entries of $\mathbf{v}_1, \mathbf{v}_2$, respectively, the sender must communicate 3 field elements, 2 to set $a_1^{\ell+1}$ and $a_2^{\ell'+1}$, and the additional value $b_1^{\ell+1} - b_2^{\ell'+1}$ that is sent in the clear.

6.2.4 Proof of Lemma 6.2

Correctness and security against malicious sender are immediate from the correctness and security of the underlying protocols. For security against a malicious receiver, the extractor Ext obtains the values $\alpha_1, \ldots, \alpha_k$ by subtracting $\alpha_1^* - \alpha_{k+1}^*, \ldots, \alpha_k^* - \alpha_{k+1}^*$. The simulator for the receiver generates the entire vectors $\mathbf{v}_{k+1}, \mathbf{v}_{k+2}$ uniformly at random, which matches the distribution under an honest run of the protocol by the randomness of the values b_{k+1}^j, b_{k+2}^j. Vector elements $v_i^{j_1}$ are computed from the formula

$$v_i^{j_1} = \hat{v}_i^{j_1} + v_{k+1}^{j_2},$$

using the indexing from the previous subsection and writing $\hat{v}_i^{j_1}$ for entries of receiver output, i.e. of $f(\mathbf{x}^*, \mathbf{y})$, as above. All remaining entries of all vectors are chosen uniformly at random. Perfect security follows by the security of NIZK and eVOLE. We only need to check that eVOLE is only applied to pairs of VOLEs where the receiver VOLE inputs are uniformly random and independent. This is true for the pairs $(\alpha + \alpha_i, \beta)$ and the pair (α, β), as desired.

Complexity: For each input to the circuit C, whether from the $\mathbf{a}_i$'s or $\mathbf{b}_i$'s, we require two instances of eVOLE. For an entry of the $\hat{\mathbf{a}}_i$'s that is an input to C, we require an additional two entries of VOLE from $\mathbf{a}_{k+1}$ and $\mathbf{a}_{k+2}$. For an entry of the $\hat{\mathbf{b}}_i$'s that is an input to C, we require three additional entries. For a total of $q := q_{\mathbf{a}} + q_{\mathbf{b}}$ inputs, this requires

$$6q_{\mathbf{a}} + 7q_{\mathbf{b}} + \sum_{i=1}^{k} \ell_i$$

entries of VOLE and

$$8q_{\mathbf{a}} + 9q_{\mathbf{b}} + 2\sum_{i=1}^{k} \ell_i$$

field elements of communication.

The LPZK-NIZK proof of C is carried out on the $k+2$nd instance of VOLE, and the inputs to C have already been included in the cost, so this requires an additional $2m$ entries of VOLE, and an additional $k' + (2 + \frac{1}{t})m$ field elements of communication. 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 [20], 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 [20].

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 [32, 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 [20] in a simpler way.

Theorem 6.3 (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 $15 \sum_{i=1}^{\ell} d_i s_i^2 + 2 \sum_{i=1}^{\ell} s_i^3$.*
- *The simulation error (per invocation) is $\varepsilon = O(1/|\mathbb{F}|)$.*

Chase et al. [20] show how to bootstrap Theorem 6.3 to get reusable NISC over VOLE for general Boolean circuits, by making (a non-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 normalized 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 Functionality

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

$$\sum_{i=1}^{n} y_i^2 \leq L^2,$$

for some other constant L, so that the ℓ^2 norm satisfies

$$\|\mathbf{y}\|_2 \leq L.$$

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 L are powers of 2.

In the above description we assume the inputs to be vectors over non-negative integers. This functionality can be naturally embedded by considering vectors over a finite field $\mathbb{F}$ of prime order p, provided that p is bigger than the square-norm bound L^2 and an upper bound on the output size.

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 of squares $\sigma_y := \sum y_i^2$. We use $\lg K$ bits for the bit decompositions (c_{ij}) and $2 \lg L$ bits for bit decomposition (c_{sj}), which ensures that $\mathbf{y}$ satisfies the desired bounds if the bit decompositions are correct.

We give a slightly modified construction of cVOLE, optimized for this setting. R and S generate $n + 2$ instances of random VOLE. As in cVOLE, R chooses inputs $(x_1 + \alpha, \ldots, x_n + \alpha, \alpha, \beta)$, with $\alpha, \beta \in \mathbb{F}$ random and independent. The input of S to the ith instance of VOLE is y_i, for $1 \leq i \leq n$. Then S uses the entire vector $\mathbf{y}$ as inputs to the $(n + 1)$st and $(n + 2)$nd instance of VOLE. S also takes as inputs to the $(n + 2)$nd instance all constant terms from the first n VOLEs, the sum of the constant terms from the $(n + 1)$st instance, the squares y_i^2, and the bit decompositions (c_{ij}) and (c_{sj}). After this initial set up, R learns the following:

- $v_i^1 := y_i(x_i + \alpha) + z_i$, for $1 \leq i \leq n$, where z_i is a random element determined in the initial random VOLE set up, and thus requires no additional communication.
- $v_{n+1}^i := y_i\alpha + w_i$, for $1 \leq i \leq n - 1$, with w_i from the random VOLE.
- $v_{n+1}^n := y_n\alpha + w_n$, where w_n is chosen such that $\sum_{i=1}^n z_i = \sum_{i=1}^n w_i$.
- $v_{n+2}^i := y_i\beta + u_i$, for $1 \leq i \leq n$, with u_i from the random VOLE.
- $v_{n+2}^{n+i} := z_i\beta + u_{n+i}$, for $1 \leq i \leq n$, with u_i from the random VOLE.
- $v_{n+2}^{2n+1} := (\sum_{i=1}^n w_i)\beta + u_{2n+1}$, with u_{2n+1} from the random VOLE.

Additionally, $\mathbf{a}_{n+2}$ holds all of the bit decompositions and associated data mentioned above. To complete the verification step of the protocol, R and S execute eVOLE to ensure that all inputs that occur in multiple VOLE instances are, in fact, equal, and then S uses LPZK-NIZK on the $(n+2)$nd instance of VOLE to convince R of the validity of S's input.

The NIZK proof checks that all values y_i^2 sent by S are actually equal to the squares of the values y_i, and confirms that c_{ij} and c_{si} are in $\{0,1\}$ by evaluating the quadratic t^2-t on each entry. The proof then checks that the bit-decompositions are correct by computing and revealing $y_i - \sum_j c_{ij}2^j$ and $\sum y_i^2 - \sum_j c_{sj}2^j$, all of which are equal to zero when both parties behave honestly.

Finally, R computes the output value as

$$\mathbf{x}\cdot\mathbf{y} = \sum_{i=1}^{n} \left(v_i^1 - v_{n+1}^i\right).$$

6.5 Proof

We have

$$\sum_{i=1}^{n} \left(v_i^1 - v_{n+1}^i\right) = \sum_{i=1}^{n} y_i(x_i+\alpha) - y_i\alpha + z_i - w_i = \mathbf{x}\cdot\mathbf{y},$$

since $\sum z_i = \sum w_i$, which shows correctness.

Security against a malicious sender is an immediate consequence of the security of eVOLE and LPZK-NIZK, as in the proof of security of cVOLE.

The argument for security against a malicious receiver follows closely the argument for cVOLE. The simulator Sim can generate each of the elements v_i^j uniformly at random, due to the randomness of the z_i's, w_i's, and u_i's, except for v_{n+1}^n, which the simulator computes via

$$v_{n+1}^n = \left(\sum_{i=1}^{n-1} v_{n+1}^i - v_i^1\right) - v_n^1 - \mathbf{x}\cdot\mathbf{y}.$$

6.5.1 Complexity

The protocol requires the $4n+1$ entries of VOLE listed, which in turn require communication of $4n+2$ field elements to convert from random VOLE to fixed VOLE (one per entry, except we need two field elements for $\mathbf{v}_{n+1}^n$.)

The protocol contains additionally 3 field elements of communication and 2 entries of VOLE for each of the $3n+1$ equality constraints implemented under $\mathcal{F}_{eVOLE}^{(\mathbb{F})}$, and a LPZK-NIZK verification of a circuit with $2n + n\lg K + 2\lg(L)$ inputs and $n + n\lg K + 2\lg(L)$ multiplication gates.

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

$$(4n+1) + 2(3n+1) + n + n\lg K + 2\lg(L) = 11n + n\lg K + 2\lg L + 3$$

entries of VOLE and communication of

$$\begin{aligned}(4n+2) + 3(3n+1) + (2+\tfrac{1}{t})(n + n\lg K + 2\lg(L)) \\ = \left(15 + 2\lg K + \tfrac{1+\lg K}{t}\right)n + \left(4+\tfrac{2}{t}\right)\lg L + 5\end{aligned}$$

field elements. Taking $K = 2^{10}$, $L = 2^{32}$, $n = 1000$, and $t = 11$ gives total VOLE length $21,067$ and total communication $36,139$ field elements, as stated in the introduction.

We note that we can allow the bounds K and L 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 $O(n \lg K)$ multiplication gates in the LPZK-proof.

We remark that the main way we deviate here from our cVOLE construction is through summing the w_i terms on the $(n+1)$st instance of VOLE and then passing that sum to the $(n+2)$nd instance, instead of sending each of the w_i terms to the $(n+2)$nd instance of VOLE. This optimization reduces the entries of VOLE needed by $(n-1)$ and communication by $4(n-1)$.

Acknowledgements

Supported in part by DARPA 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 supported in part 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] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3), 1998.

[6] Carsten Baum, Daniel Escudero, Alberto Pedrouzo-Ulloa, Peter Scholl, and Juan Ramón Troncoso-Pastoriza. Efficient protocols for oblivious linear function evaluation from ring-lwe. In *SCN 2020*, pages 130–149, 2020.

[7] Carsten Baum, Alex J. Malozemoff, Marc Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for arithmetic circuits with nested disjunctions. Cryptology ePrint Archive, Report 2020/1410, 2020. `https://eprint.iacr.org/2020/1410`.

[8] 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.

[9] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, 2011.

[10] 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.

[11] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC 1988*, pages 103–112, 1988.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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`.

[18] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS 2001*, pages 136–145, 2001.

[19] 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.

[20] 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.

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

[22] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, 2012.

[23] Leo de Castro, Chiraag Juvekar, and Vinod Vaikuntanathan. Fast vector oblivious linear evaluation from ring learning with errors. *IACR Cryptol. ePrint Arch.*, 2020:685, 2020. URL: `https://eprint.iacr.org/2020/685`.

[24] 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.

[25] Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In *STOC 2014*, pages 495–504, 2014.

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

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

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

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

[30] 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.

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

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

[33] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient arguments without short PCPs. In *CCC*, 2007.

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

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

[36] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In *TCC 2009*, pages 294–314, 2009.

[37] 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.

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

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

[40] 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.

[41] 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.

[42] Moni Naor and Benny Pinkas. Oblivious polynomial evaluation. *SIAM Journal on Computing*, 35(5):1254–1281, 2006.

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

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

[45] 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`.

[46] 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.

[47] 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`.

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

A Concrete computational overhead

In actual implementations, computation cost is affected by the choice of field size, the computer architecture, and the circuit structure. When considering typical circuits over large finite fields, the ratio between the NIZK online computation cost (given a precomputed random VOLE correlation) and computing the circuit in the clear is typically between 2 and 5 for both the prover and verifier. We describe some possible parameter choices below. We remark, however, that when computation costs are this low, we expect that the overall cost will be typically dominated by online communication and offline computation rather than online computation.

The cost of evaluating a multiplication over $\mathbb{F}$, for generic field characteristic, will be dominated by the cost of the modular reduction, which will be far greater than the cost of an addition, and so it is reasonable in this case to treat the cost of additions as negligible.The ratio of NIZK computation cost to plain circuit evaluation now depends on the number of multiplication gates where one multiplicand is a public value. If 70% of the multiplication gates in C satisfy this condition, then, using the more detailed accounting of prover and verifier computation given in § 4.4, we obtain $T(C) = \frac{7}{3}mT(*)$ and cost of evaluation in the clear is $\frac{10}{3}mT(*)$. This then gives prover work of $\frac{26}{3}mT(*)$ and verifier work of $\frac{22}{3}mT(*)$, for a prover multiplier of 2.6 and a verifier multiplier of 2.2.

For carefully chosen primes, such as Mersenne primes, modular multiplication can be implemented much more efficiently. Suppose, in this setting, that the cost of a multiplication in the architecture's instruction set is 6 times the cost of an addition, and suppose the circuit has an equal number of addition, multiplication-by-secret-value, and multiplication-by-public-value gates. Then, using the more detailed accounting of prover and verifier computation given in § 4.4, we obtain $T(*) = 6T(+)$, $T(C) = 7mT(+)$, and cost of evaluation in the clear is $13mT(+)$. This then gives prover work of $44mT(+)$ and verifier work of $39mT(+)$, for a prover multiplier of 3.4 and a verifier multiplier of 3.0.

Note that when we compare the computation cost of evaluating a circuit in the clear to the prover or verifier computation cost, we restrict our attention to the case where all parties evaluate an arithmetic circuit over a finite field $\mathbb{F}$. In some cases, the circuit could be evaluated more

quickly in the clear by treating it as a circuit over $\mathbb{Z}$, and therefore avoiding the cost of finite field arithmetic. We leave an empirical cost analysis for realistic use-cases to future implementations.