Linear-time and post-quantum zero-knowledge SNARKs for R1CS

Jonathan Lee
Nanotronics Imaging

Srinath Setty
Microsoft Research

Justin Thaler
Georgetown University

Riad Wahby
Stanford University

Abstract

This paper studies zero-knowledge SNARKs for NP, where the prover incurs $O(N)$ finite field operations to prove the satisfiability of an N-sized R1CS instance. We observe that recent work of Bootle, Chiesa, and Groth (BCG, TCC 20) provides a polynomial commitment scheme that, when combined with the linear-time interactive proof system of Spartan (CRYPTO 20), yields linear-time IOPs and SNARKs for R1CS. Specifically, for security parameter λ, and for an N-sized R1CS instance over a field of size $\exp(\lambda)$ and fixed $\epsilon > 0$, the prover incurs $O(N)$ finite field operations to produce a proof of size $O_\lambda(N^\epsilon)$ that can be verified in $O_\lambda(N^\epsilon)$—after a one-time preprocessing step, which requires $O(N)$ finite field operations. This reestablishes the main result of BCG. Arguably, our approach is conceptually simpler and more direct. Additionally, the polynomial commitment scheme that we distill from BCG is of independent interest; it improves over the prior state of the art by offering the first scheme where the time to commit to an N-sized polynomial is $O(N)$ finite field operations. We further observe that one can render the aforementioned SNARK zero knowledge and reduce the proof size and verifier time to polylogarithmic—while maintaining a linear-time prover—by outsourcing the verifier's work via one layer of proof composition with an existing zkSNARK as the "outer" proof system. A similar result can be derived from recent work of Bootle, Chiesa, and Liu (ePrint 2020/1527).

We implement the aforementioned polynomial commitment scheme with $\epsilon = 1/2$ and combine it with Spartan's interactive proof system to obtain a SNARK for R1CS. We refer to this combination as *Cerberus*. It uses Reed-Solomon codes in the polynomial commitment scheme, and hence the prover is *not* asymptotically linear-time. Nonetheless, Cerberus features the fastest known prover (the only exception is Spartan when proving large instances over 256-bit fields), and is plausibly post-quantum secure.

1 Introduction

A zero-knowledge SNARK (zkSNARK) [Kil92, Mic94, GW11, BCCT12] is a cryptographic primitive that enables a *prover* to prove to a *verifier* the knowledge of a satisfying witness to an NP statement by producing a proof π such that: (1) π reveals no information beyond what is implied by the validity of the NP statement; and (2) the size of π and the cost to verify it are both sub-linear (ideally, at most polylogarithmic) in the size of the statement. Given their practical applications, constructing zkSNARKs with excellent asymptotics and concrete efficiency is a highly active area of research.

As with much of the literature on zkSNARKs, we focus on SNARKs for the following NP-complete problems: rank-1 constraint satisfiability (R1CS)[1] and arithmetic circuit satisfiability over a finite field $\mathbb{F}$. These problems are powerful "intermediate representations", in that they are amenable to efficient checking via zkSNARKs and are highly expressive. For example, in theory, any non-deterministic random access machine running in time T can be transformed into an R1CS or an arithmetic-circuit-satisfiability instance of size "close" to T. In practice, there exist highly efficient transformations and compiler toolchains to transform applications of interest to these representations [SVP+12, PGHR13, BFR+13, BCGT13, WSR+15, SAGL18a, LNS20].

Our focus in this work is designing zkSNARKs for these problems with the fastest possible prover. We also wish to avoid a trusted setup, and desire a verifier that runs in at most polylogarithmic time. Since the verifier must at least read the statement that is being proven, we allow a one-time public preprocessing phase for general, unstructured, R1CS or circuit-satisfiability instances, where the verifier computes a *computation commitment*, a cryptographic commitment to the structure of a circuit or R1CS matrices [Set20]. After the

[1] R1CS is implicit in the QAPs of Gennaro et al. [GGPR13], but was made explicit in subsequent works [SBV+13, BCR+19].

pre-processing phase, the verifier must run in polylogarithmic time. Furthermore, the pre-processing phase should be at least as efficient as the SNARK prover. Subsequent works to Spartan [Set20] refer to such public preprocessing to achieve fast verification as leveraging *holography* [CHM+20, COS20, BCG20a].

Formalizing "fastest possible" provers. How fast can we hope for the prover in a zkSNARK to be? Letting N denote the size of the R1CS or arithmetic-circuit-satisfiability instance over an *arbitrary* finite field $\mathbb{F}$,[2] a lower bound on the prover's runtime is N operations in $\mathbb{F}$. This is because any prover that knows a witness w for the instance has to at least convince *itself* (much less the verifier) that w is valid. We refer to this procedure as *native evaluation* of the instance. So the natural goal, roughly speaking, is to achieve a zkSNARK prover that is only a constant-factor slower than native evaluation. Such a prover is said to run in *linear-time*. We anticipate that the development of linear-time provers is a prerequisite to deploying zkSNARKs on significantly larger statements than is currently feasible.

Achieving a linear-time prover may sound like a simple and well-defined goal, but it is in fact quite subtle to formalize, because one must be precise about what operations can be performed in one "time-step", as well as the soundness error achieved and the choice of the finite field.

In known zkSNARKs, the bottleneck for the prover (both asymptotically and concretely) is typically one or more of the following operations (we discuss exceptions later): (1) Performing an FFT over a vector of length $O(N)$. (2) Building a Merkle-hash tree over a vector consisting of $O(N)$ elements of $\mathbb{F}$. (3) Performing a multiexponentiation of size $O(N)$ in a cryptographic group $\mathbb{G}$; in this case, the field $\mathbb{F}$ is of prime order p and $\mathbb{G}$ is typically an elliptic curve group (or subgroup) of order p.

Should any of these operations count as "linear-time"?

FFTs. It is clear that an FFT of length $\Theta(N)$ over $\mathbb{F}$ should *not* count as linear-time, because the fastest known algorithms require $\Theta(N \log N)$ operations over $\mathbb{F}$, which is a $\log N$ factor, rather than a constant factor, larger than native evaluation.

However, the remaining operations are somewhat trickier to render judgment upon, because they do not refer to field operations.

Merkle-hashing. Regarding (2), a first observation is that to build a Merkle tree over a vector of $O(N)$ elements of $\mathbb{F}$, computing $O(N)$ cryptographic hashes is necessary and sufficient, assuming the hash function takes as input $O(1)$ elements of $\mathbb{F}$. However, this is only "linear-time" if hashing $O(N)$ elements of $\mathbb{F}$ can be done in time comparable to $O(N)$ operations over $\mathbb{F}$.

Bootle, Cerulli, Ghadafi, Groth, Hajiabadi and Jakobsen [BCG+17] observe that (assuming the intractability of certain lattice problems over $\mathbf{F}_2$, specifically finding a low-Hamming vector in the kernel of a sparse matrix), a collision-resistant hash family of Applebaum, Haramaty, Ishai, Kushilevitz, and Vaikuntanathan [AHI+17] is capable of hashing strings consisting of $k \gg \lambda$ bits in $O(k)$ bit operations, with security parameter λ. This means that a vector of $O(N)$ elements of $\mathbb{F}$ can be Merkle-hashed in $O(N \log |\mathbb{F}|)$ bit operations, which Bootle et al. [BCG+17] consider comparable to the cost of $O(N)$ operations in $\mathbb{F}$. Following their lead, we consider this to be "linear-time". However, note that these hash functions may be of primarily theoretical interest because they can be orders of magnitude slower than standard hash functions (e.g., SHA-256 or blake).[3]

Multiexponentation. Pippenger's algorithm can perform an $O(N)$-sized multiexponentiation in a group $\mathbb{G}$ of size $\sim 2^\lambda$ by performing $O(N \cdot \lambda / \log(N \cdot \lambda))$ group operations. Typically, one thinks of the security parameter λ as $\omega(\log N)$ (so that 2^λ is superpolynomial in N, ensuring the intractability of problems such as discrete logarithm in $\mathbb{G}$), and so $O(N \cdot \lambda / \log(N \cdot \lambda))$ group operations is considered $\omega(N)$ group operations. Each group operation is in practice at least as expensive (in fact, several times slower) than a field operation,[4] and hence for purposes of this work, we do *not* consider this to be linear time.

[2] The size of an arithmetic-circuit-satisfiability instance is the number of gates in the circuit. The size of an R1CS instance of the form $Az \circ Bz = Cz$ is the number of non-zero entries in A, B, C, where $\circ$ denotes the Hadamard (entry-wise) product.

[3] The security of these hash functions is based on novel lattice assumptions.

[4] Typically, an operation in the elliptic-curve group $\mathbb{G}$ requires performing a constant number of field operations within a field that is of similar size to, but different than, then prime-order field $\mathbb{F}$ over which the circuit or R1CS instance is defined.

However, note that *for a fixed value of the security parameter* λ, the cost of a multiexponentiation of size N performed using Pippenger's algorithm scales only linearly (in fact, *sub*linearly) with N. That is, Pippenger's algorithm incurs $\Theta(N \cdot (\lambda/\log(N\lambda))) = \Theta_\lambda(N/\log N)$ group operations and in turn this cost is comparable up to a constant factor to the same number of operations over a field of size $\exp(\lambda)$ (see Footnote 4). In practice, protocol designers fix a cryptographic group (and hence fix λ), and then apply the resulting protocol to R1CS instances of varying sizes N. For this reason, systems (such as Spartan [Set20]) whose dominant prover cost is a multiexponentiation of size N will scale (sub-)linearly as a function of N. Specifically, in the experimental results [Set20], Spartan's prover exhibits the behavior of a linear-time prover (as the cost of native evaluation of the instance also scales linearly as a function of N).

Nonetheless, as a theoretical matter, λ should be thought of as $\omega(\log N)$, and hence we do not consider a multiexponentation of size N to be a linear-time operation.

Prior work. According to our taxonomy above, the problem of designing zkSNARKs with a linear-time prover remains open (the paragraphs ahead clarify this statement further). The following works have come closest to this goal: (1) Hyrax [WTS+18] and Libra [XZZ+19] for low-depth, uniform circuits; (2) Spartan [Set20] and Xiphos [SL20] for arbitrary, non-uniform statements represented with R1CS; (3) The recent work of Kothapalli et al. [KMP20] for a variant of R1CS; and (4) The recent work of Bootle et al. [BCG20a, BCL20] for R1CS. All these schemes combine the sum-check protocol [LFKN90, BCR+19] with cryptographic commitments (e.g., polynomial commitments).

In Hyrax and Libra, beyond performing $O(N)$ finite field operations,[5] the prover performs $O(d \log N + W)$ exponentiations in a group, where W is the size of the witness to the circuit. As a result, these schemes achieve a linear-time prover so long as $d \log N + W \leq (N \cdot \log W)/\lambda$ (i.e., for circuits of sub-linear depth and with sub-linear sized witnesses, and when $|\mathbb{F}| = 2^{\Theta(\lambda)}$). Two downsides remain. First, their underlying interactive proof [GKR08, CMT12, WJB+17] has communication cost $\Theta(d \log N)$, which places a lower bound on the proof length (regardless of the polynomial commitment scheme used). Second, they require uniform circuits (e.g., data-parallel circuits) to achieve verifier's costs that are sub-linear in the circuit size.

In contrast, Spartan and Xiphos apply to arbitrary R1CS and when using appropriate polynomial commitment schemes (such as Dory [Lee20]), their proof length is $O(\log N)$ group elements. Furthermore, they achieve $O_\lambda(\log N)$ verification times, after a one-time preprocessing step to create a commitment to R1CS matrices.[6] However, after performing $O(N)$ operations over $\mathbb{F}$, Spartan's and Xiphos's provers perform an $O(N)$-sized multiexponentiation, which stems from their use of a polynomial commitment schemes applied to a multilinear polynomial with $O(N)$ coefficients.[7] As a result, according to the aforementioned accounting, they do not achieve a linear-time prover even though concretely these zkSNARKs have a prover time that is in fact the state of the art in most cases.

In a recent theoretical work, Kothapalli et al [KMP20] propose a zkSNARK that achieves $O_\lambda(1)$ proof sizes and verification times for a variant of R1CS, but their prover performs an $O(N)$-sized multiexponentiation for an N-sized R1CS instance, so it does not achieve a linear-time prover according to the above accounting. Additionally, their scheme requires a one-time trusted setup.

Another recent theoretical work by Bootle, Chiesa, and Groth [BCG20a] give an *interactive oracle proof* (IOP) [BCS16] in which the prover's work is $O(N)$ finite field operations for an N-sized R1CS instance over any finite field of size $\Omega(N)$ and with a multiplicative subgroup of size N.[8] By applying standard transformations to their construction, one can obtain a SNARK in the random oracle model with similar prover costs, or an interactive argument assuming linear-time computable cryptographic hash functions [AHI+17]. However, to achieve a

[5] Hyrax employs Giraffe [WJB+17] that shows how to implement the prover via $O(N)$ finite field operations for *data-parallel* circuits. Libra showed how to achieve this runtime bound for arbitrary circuits.

[6] Similar preprocessing applies to Hyrax and Libra to achieve an $O_\lambda(d \log N)$-time verifier even for non-uniform, depth-d circuits; see Figure 1.

[7] There exist polynomial commitment schemes, e.g., [ZXZS20], that do not require a multiexponentiation, but they require the prover to perform an FFT over a vector of length N, which as discussed is not linear-time, and in practice is slower than an $O(N)$-sized multiexponentiation for large enough N.

[8] An interactive oracle proof (IOP) [BCS16, RRR16] is a generalization of an interactive proof, where in each round, the prover sends a string as an oracle, and the verifier may read one or more entries in the oracle.

soundness error that is negligible in the security parameter λ, one must either sacrifice the linear-time prover, or restrict to R1CS instances over a finite field where $|\mathbb{F}| = 2^{\Theta(\lambda)}$.[9] Furthermore, their construction does not achieve zero-knowledge nor polylogarithmic proofs and verification times (the proof sizes and verification times are $O_\lambda(N^{1/t})$, where t is a constant, and not $O_\lambda(\log N)$ or $O_\lambda(1)$).

Concurrent with our work, Bootle, Chiesa, and Liu [BCL20] address the latter issue by both achieving zero-knowledge as well as polylogarithmic proof sizes and verification times via a host of new techniques. A more detailed discussion of the relationship between our results and those of [BCL20] is in Section 1.3.

In summary, existing works leave open the following problems: (1) designing an IOP (or a zkSNARK) with a linear-time prover for R1CS instances defined over an arbitrary finite field; and (2) designing a zkSNARK for R1CS, which in addition to a linear-time prover, achieves concrete prover performance better than existing high-speed zkSNARKs [WTS+18, XZZ+19, Set20, SL20] for "reasonable" instance sizes (e.g., $N \leq 2^{30}$).

1.1 Our results

We make progress towards these problems by building on existing high-speed zkSNARKs.

Our first result is an interactive oracle proof that achieves the following efficiency. For security parameter λ, and for an N-sized R1CS instance over a field of size $\exp(\lambda)$ and any fixed $\epsilon > 0$, the prover incurs $O(N)$ finite field operations to produce a proof of size $O_\lambda(N^\epsilon)$ that can be verified in $O_\lambda(N^\epsilon)$—after a one-time preprocessing step, which requires $O(N)$ finite field operations.[10] This reestablishes the main result of Bootle, Chiesa, and Groth [BCG20a], but we believe that our approach is simpler and more direct. It also reuses an existing high-speed, linear-time interactive proof system for R1CS as well as its approach to achieve sub-linear verification costs [Set20]. As with any IOP, one can obtain a SNARK in the random oracle model via standard transformations, and heuristically in the plain model by instantiating the random oracle with an appropriate hash function.[11]

Second, we observe that one can render the aforementioned SNARK zero knowledge and reduce the proof size and verifier time to at most polylogarithmic—while maintaining a linear-time prover—by outsourcing the verifier's work via one layer of proof composition with an existing zkSNARK as the "outer" proof system.

Along the way, we distill from [BCG20a] a polynomial commitment scheme with the following efficiency characteristics (this commitment scheme is little more than a rephrasing of the results in [BCG20a]). For security parameter λ, and for a $\log N$-variate multilinear polynomial over a field of size $\exp(\lambda)$ and any fixed $t > 0$, the time to commit to a polynomial and prove an evaluation of a committed polynomial are both $O(N)$ finite field operations, the size of a commitment is $O_\lambda(1)$, and the time to verify a proof of an evaluation and the size of the proof are both $O_\lambda(N^{1/t})$. This improves over the prior state-of-the-art polynomial commitment schemes [KZG10, ZGK+17, WTS+18, ZXZS20, BFS20, SL20, Lee20] by offering the first polynomial commitment scheme where the time to commit to a polynomial is linear in the size of the polynomial. Our stated result is for multilinear polynomials, but the scheme generalizes to arbitrary polynomials such as univariate polynomials (see e.g., [Lee20]).

For completeness, we provide a self-contained description and an analysis of the distilled polynomial commitment scheme for the case of $t = 2$. The scheme and its analysis is far simpler than the scheme for general integers $t > 0$ in [BCG20a]. Moreover, even the $t = 2$ case suffices to achieve all of our results that make use of one layer of recursive composition (as to achieve these results, it is sufficient for the proof length and verifier time of the "inner SNARK" to be smaller than N by any factor that dominates $\lambda / \log N$). The

[9] Since [BCG20a] achieves a linear-time prover only when $|\mathbb{F}| > \exp(\lambda)$, the prover's work in bit operations has an implicit dependence on λ. Of course, in this case the cost of native evaluation in terms of bit operations has the same implicit dependence on λ.

[10] The IOP yields constant soundness error over field sizes as small as $\Theta(N)$, but parts of the protocol would need to be repeated $\Theta(\lambda)$ times to obtain λ bits of security, and these repetitions would not preserve linear prover time.

[11] Because the IOP uses logarithmically many rounds rather than constantly many, the transformation into a SNARK in the random oracle model may introduce some loss of security unless one shows the IOP satisfies an additional security property called round-by-round soundness [CCH+, BCS16]. Our IOP is based on the sum-check protocol [LFKN90], which is known to be round-by-round sound [CCH+]; however, for brevity we omit details of a full analysis of round-by-round soundness of our IOP.

distilled polynomial commitment scheme for the $t = 2$ case is already arguably implicit in the argument system (for the arithmetic circuit satisfiability problem) of [BCG+17], an ancestor of [BCG20a].

Figure 1 depicts the asymptotic efficiency of our schemes and compares it with prior work.

Implementation and experimental results. We implement the aforementioned polynomial commitment scheme with $\epsilon = 1/2$ to investigate its concrete efficiency. We then combine it with Spartan's interactive proof system [liba] to obtain a SNARK for R1CS. We refer to this combination as *Cerberus*. It uses the Reed-Solomon code in the polynomial commitment scheme, and hence the prover is *not* asymptotically linear-time. Nonetheless, Cerberus features the fastest prover (the only exception is Spartan when proving large instances over 256-bit fields). In particular, when working over 256-bit fields (compatible with cryptographic groups where the discrete logarithm problem is hard), Cerberus is faster than Spartan on smaller instances ($N < 2^{20}$), but not on larger instances. However, Cerberus can prove R1CS instances over any finite field that supports FFTs, and is faster than any other (zero-knowledge) argument system. Furthermore, Cerberus is plausibly post-quantum secure. Cerberus's proof sizes are generally larger than prior schemes, but its verification times are competitive with those of prior schemes.

We find this performance to be surprisingly attractive, because the components of Cerberus have arguably been known for some time. In particular, the polynomial commitment scheme when instantiated with the Reed-Solomon code is arguably implicit in Ligero [AHIV17, Tha20], and our implementation merely combines that polynomial commitment scheme with Spartan's polynomial IOP [Set20].

Additionally, we demonstrate that Cerberus is a *complete* replacement to Ligero [AHIV17], both asymptotically and concretely for prover times, verification times, proof sizes, and assumptions (e.g., post-quantum security).

Cerberus's implementation is not currently zero-knowledge; however, it can be rendered zero-knowledge using standard techniques with minimal overhead [AHIV17, XZZ+19, CFS17]. We leave the completion of a zero-knowledge implementation to near-term future work.

1.2 Our approach

Distilling a linear-time polynomial commitment scheme. We observe that a core technique in the work of Bootle, Chiesa, and Groth [BCG20a] can be used, in a straightforward manner, to build a non-interactive argument of knowledge for a specific type of inner product relations between two vectors over $\mathbb{F}$, where one of the vectors is committed with a binding commitment and the other is the tensor product of a constant number of smaller vectors. Specifically, an untrusted prover commits to a vector $z \in \mathbb{F}^N$ by producing an $O_\lambda(1)$-sized commitment, and then later proves that $v = \langle q, z \rangle$, where $q \in \mathbb{F}^N$ is any vector that is a tensor product of t vectors of length $N^{1/t}$ (for a constant t) and $v \in \mathbb{F}$. For N-sized vectors, the cost to commit and to later prove an inner product relation are both $O(N)$ operations over $\mathbb{F}$; the size of an evaluation proof and the time to verify are both $O_\lambda(N^{1/t})$ where λ is the security parameter (the verification time is sub-linear because it exploits the tensor structure in q).

We further observe that the above non-interactive proof system implies a polynomial commitment scheme for multilinear polynomials with efficiency characteristics stated earlier. In a nutshell, one can express the evaluations of a multilinear polynomial using the special inner product operation (§4).[12]

Constructing linear-time SNARKs. Recent works have considered several generalizations of IOPs, such as "polynomial IOPs" [BFS20] (in which in each round the prover may send the verifier a polynomial, and the verifier may query the polynomial to obtain its evaluation at any desired point) and "tensor IOPs" [BCG20a] (in which each round the prover may send a vector to the verifier and the verifier can pose "tensor queries" to the vector, meaning the verifier request the inner product of the vector with any vector possessing "tensor structure"). These formalisms are meant to capture the most general type of protocols that we know how to transform into SNARKs. For example, Bootle, Chiesa, and Groth [BCG20a] give a transformation from any tensor IOP to a standard IOP, in a manner that preserves linear runtime for the prover. The latter is in

[12] A similar polynomial commitment scheme is implicit in earlier work of Rothblum and Ron-Zewi [RR20], albeit with a worse dependence of polynomial evaluation proof lengths on both the security parameter λ and the constant parameter t.

turn known to be transformable into SNARKs [Mic94, Val08, BCS16] (also in a manner that preserves linear runtime for the prover). Similarly, Bünz, et al [BFS20] provide a compiler from polynomial IOPs to SNARKs using an (extractable) polynomial commitment scheme.

To construct SNARKs with a linear-time prover, we take two separate routes (both produce the same result).

1. We observe that the prover's work in Spartan [Set20], excluding its work as part of a polynomial commitment scheme, is $O(N)$ finite field operations for an N-sized R1CS instance. Thus, by employing the linear-time polynomial commitment scheme that we distill from [BCG20a] in Spartan, we can obtain SNARKs for R1CS with a linear-time prover. To formalize this, we first observe that Spartan [Set20] gives a polynomial IOP for R1CS with a linear-time prover (§3). We then apply the compiler of [BFS20] using the linear-time polynomial commitment scheme.

2. We observe that any polynomial IOP is also a tensor IOP (§7). This observation is in fact implicit or explicit in many prior works, e.g., [AHIV17, WTS+18, RR20, Lee20, Tha20]. Hence, applying the transformation of [BCG20a] to Spartan's polynomial IOP yields a SNARK with linear-time prover.

Finally, our recursively-composed zkSNARKs are obtained via a standard application of (one level of) recursive composition with known zkSNARKs. Prior work [CFH+15, BCG+20b, KMP20] also applies one level of proof composition to achieve similar properties as us, but in different contexts.

Bootle, Chiesa, and Groth [BCG20a] remark that Spartan [Set20] combined with a subset of their techniques could potentially establish their main result, but suggest that this could be a complicated endeavor as one must distill a tensor IOP from Spartan. We believe that our work shows otherwise. Additionally, our first route offers a more direct way of achieving the same result. This route makes no explicit reference to the tensor IOP formalism, and by extension may render the core technique of [BCG20a] more easily applicable in diverse settings, as polynomial commitment schemes are now well-understood primitives with many applications [BGH19, ZXZS20, BFS20, BCMS20, BDFG20].

1.3 Additional comparisons

On cryptographic assumptions and detailed comparison to [BCL20]. Bootle, Chiesa, and Liu [BCL20] improve on the IOP given in [BCG20a] by achieving zero-knowledge, and reducing the proof length and verifier runtime to polylogarithmic in N. As previously discussed, standard transformations can be applied to the IOP to obtain a zkSNARK that is unconditionally secure in the random oracle model. However, for real-world use the SNARK must ultimately be instantiated in the plain model, with the random oracle replaced with a hash function supporting linear-time evaluation. The hash function used [AHI+17] is collision-resistant based on certain lattice assumptions.

Our first (non-zero-knowledge) SNARK satisfies the same security property as [BCL20, BCG20a], in the sense that we give an IOP, and this can be transformed into a SNARK that is unconditionally secure in the random oracle model. However, our other SNARKs do not, owing to their use of (one layer of) recursive composition. Rather, they are knowledge-sound assuming our first SNARK remains knowledge sound in the plain model when the random oracle is instantiated by the appropriate concrete hash function. This is a well-known issue that arises whenever one recursively composes two SNARKs, where the "inner" SNARK makes use of random oracles [Val08, BCCT13, COS20, BCMS20]. The random oracle used by the inner SNARK must be instantiated before recursive composition can occur, and knowledge soundness of the composed SNARK requires knowledge soundness of the inner SNARK.

From a practical perspective, this difference in security of the resulting SNARKs is unimportant for the reasons indicated above. Namely, security of any real-world linear-time instantiation of the SNARKs that are unconditionally sound in the random oracle model (e.g., the SNARKs that can be obtained from [BCL20, BCG20a], or our first SNARK) will ultimately need to assume knowledge soundness when the random oracle is instantiated with a hash function that can be evaluated in linear time, just as our recursively-composed SNARKs do.

On a technical level, Bootle et al. [BCL20] obtain a zero-knowledge IOP with polylogarithmic proof length by applying proof composition to interactive oracle proofs (IOPs) that one can later transform into zkSNARKs

	prover time	encoder time	proof size/ verifier time	ZK?	assumptions	computational model
Hyrax [WTS+18]	$O(W + d \log N)$ $\mathbb{G}$-exp $O(N)$ $\mathbb{F}$-ops	N/A	$O_\lambda(\sqrt{W} + d \log N)$	✓	DLOG, RO	data-parallel circuits with low-depth d
Libra [XZZ+19]†	$O(W + d \log N)$ $\mathbb{G}$-exp $O(N)$ $\mathbb{F}$-ops	N/A	$O_\lambda(d \log N)$	✓	q-type, RO	uniform circuits with low-depth d
Virgo [ZXZS20]	$O(N + W \log W)$ $\mathbb{F}$-ops $O(W)$ hashes	N/A	$O_\lambda(d \log N + \log^2 W)$	✓	RO	uniform circuits with low-depth d
Spartan [Set20]	$O(N)$ $\mathbb{G}$-exp	$O(N)$ $\mathbb{G}$-exp	$O_\lambda(\sqrt{N})$	✓	DLOG, RO	R1CS
Spartan++ [SL20]	$O(N)$ $\mathbb{G}$-exp	$O(N)$ $\mathbb{F}$-ops	$O_\lambda(\sqrt{N})$	✓	DLOG, RO	R1CS
Xiphos [SL20]	$O(N)$ $\mathbb{G}$-exp	$O(N)$ $\mathbb{F}$-ops	$O_\lambda(\log N)$	✓	SXDH, RO	R1CS
[KMP20]†	$O(N)$ $\mathbb{G}$-exp	$O(N)$ $\mathbb{G}$-exp	$O_\lambda(1)$	✓	q-type, RO*	ACS
Hyrax⋆	$O(W + d \log N)$ $\mathbb{G}$-exp $O(N)$ $\mathbb{F}$-ops	$O(N)$ $\mathbb{F}$-ops	$O_\lambda(d \log N)$	✓	SXDH, RO	non-uniform circuits with low-depth d
[BCG20a]	$O(N)$ $\mathbb{F}$-ops $O(N)$ hashes	$O(N)$ $\mathbb{F}$-ops $O(N)$ hashes	$O_\lambda(N^\epsilon)$	✗	RO	R1CS
[BCL20]	$O(N)$ $\mathbb{F}$-ops $O(N)$ hashes	$O(N)$ $\mathbb{F}$-ops $O(N)$ hashes	$O_\lambda(\log^c(N))$	✓	RO	R1CS
This work	$O(N)$ $\mathbb{F}$-ops $O(N)$ hashes	$O(N)$ $\mathbb{F}$-ops	$O_\lambda(N^\epsilon)$	✗	RO	R1CS
This work	$O(N)$ $\mathbb{F}$-ops $O(N)$ hashes	$O(N)$ $\mathbb{F}$-ops	$O_\lambda(\log^2 N)$	✓	RO*	R1CS
This work	$O(N)$ $\mathbb{F}$-ops $O(N)$ hashes	$O(N)$ $\mathbb{F}$-ops	$O_\lambda(\log N)$	✓	RO*, SXDH	R1CS
This work†	$O(N)$ $\mathbb{F}$-ops $O(N)$ hashes	$O(N)$ $\mathbb{F}$-ops	$O_\lambda(1)$	✓	RO*, q-type	R1CS

† Requires universal trusted setup

Figure 1: Asymptotic efficiency of prior zkSNARKs (we borrow style from [Set20, SL20, BCG20a]). The depicted costs are for an NP statement of size N over a finite field $\mathbb{F}$, where $|\mathbb{F}| = \exp(\Theta(\lambda))$ and $\lambda \geq \omega(\log N)$ is the security parameter. $\mathbb{F}$-ops refers to field multiplications or additions; $\mathbb{G}$-exp refers to an exponentiation in a group $\mathbb{G}$ whose scalar field is $\mathbb{F}$. We focus on zkSNARKs with a linear number of operations of a certain type. We do not depict schemes that do not achieve sub-linear verification costs [BBB+18, BCG+17, WYKW20, BMRS20, DIO20]. For Hyrax [WTS+18] and Libra [XZZ+19], we assume a layered arithmetic circuit, with N gates, depth d, and a non-deterministic witness of size W. The parameters $\epsilon \in (0, 1)$ and $c > 0$ are constants. ACS refers to a specialization of R1CS [KMP20]. Spartan++ and Xiphos use an untrusted assistant (e.g., the prover) to accelerate the encoder, thereby avoiding $O(N)$ $\mathbb{G}$-exp operations [SL20]. Hyrax⋆ refers to Hyrax with the following modifications from subsequent works: (1) linear-time sum-checks for non-uniform circuits from Libra [XZZ+19]; (2) computation commitments from Spartan [Set20] to preprocess the structure of a non-uniform circuit; and (3) polynomial commitment scheme from Dory [Lee20], which is also used in Xiphos [SL20]. Bootle et al. [BCG20a, BCL20] and our first scheme give IOPs, and the table refers to SNARKs that can be obtained thereof via standard transformations. Our first scheme is Theorem 6. The latter three schemes are obtained by applying one-level of proof composition to our first scheme using one of three existing zkSNARKs as the "outer" proof system: Spartan$_{\text{RO}}$ [Set20], Xiphos [SL20], and Groth16 [Gro16]. RO in the "assumptions" column refers to the random oracle model (rows for which RO is the only assumption yield an unconditionally knowledge sound protocol in the random oracle model, and *interactive* protocols in the plain model that are knowledge sound assuming CRHFs. Rows for which RO* appears in the assumption column require assuming plain-model security when the random oracle is instantiated with a concrete hash function). To achieve a linear-time prover by using a linear-time hash function for Merkle hashes, our schemes and the schemes of Bootle et al. [BCG20a, BCL20] require assuming the hardness of certain lattice problems, which are not listed in the "assumptions" column for brevity (§1).

via Merkle hashing and the Fiat-Shamir transformation [Mic94, Val08, BCS16]. In contrast, we obtain zkSNARKs with analogous costs by transforming our first (non-zero-knowledge) IOP into a SNARK *before* performing recursive proof composition.

Additional related work. GGPR-based zkSNARKs [GGPR13, PGHR13, Gro16], which build on earlier work [IKO07, Gro10, Lip12, SMBW12], offer the best proof sizes and verification times in practice, but they require the prover to perform an FFT of length $\Theta(N)$ and also require a trusted setup (a trusted authority, or a set of authorities of which at least one is honest, that creates public parameters using a secret trapdoor that they later forget). Other approaches to zero-knowledge arguments not discussed above either require the prover to devote $\Theta(N \log N)$ field operations to FFTs [AHIV17, BCR$^+$19, GWC19, CHM$^+$20], or they require a trusted setup [GWC19, CHM$^+$20], or they do not achieve sub-linear verification costs [BBB$^+$18, BCG$^+$17]. Some works [GMO16, CDG$^+$17, WYKW20, BMRS20, DIO20] achieve a linear-time prover analogous to [BCG20a], but they do not achieve sub-linear verification costs nor proof sizes; many of these works do not achieve non-interactivity nor publicly-verifiable proofs.

1.4 Roadmap

Section 3 describes a linear-time polynomial IOP for R1CS from Spartan [Set20]. Section 4 describes a linear-time polynomial commitment scheme distilled from [BCG20a]; Section 5 extends it to handle sparse polynomials efficiently using techniques from Spartan [Set20]. Section 6 describes our first route to construct linear-time SNARKs, and Section 7 describes our second route to construct linear-time SNARKs. Performance results for our implemented SNARK are detailed in Section 8. Appendix A explicitly describes the linear-time polynomial commitment scheme when $t = 2$. Appendix B provides a concretely refined analysis of the binding property of the polynomial commitment scheme when instantiated with the Reed-Solomon code as used in our implementation.

2 Preliminaries

We use $\mathbb{F}$ to denote a finite field, λ to denote the security parameter, and $\mathrm{negl}(\lambda)$ to denote a negligible function in λ. Unless we specify otherwise, $|\mathbb{F}| = 2^{\Theta(\lambda)}$.

Polynomials. We recall a few basic facts about polynomials. Detailed treatment of these facts can be found elsewhere [Tha20].

- A polynomial g over $\mathbb{F}$ is an expression consisting of a sum of *monomials* where each monomial is the product of a constant (from $\mathbb{F}$) and powers of one or more variables (which take values from $\mathbb{F}$); all arithmetic is performed over $\mathbb{F}$.
- The degree of a monomial is the sum of the exponents of variables in the monomial; the (total) degree of a polynomial g is the maximum degree of any monomial in g. Furthermore, the degree of a polynomial g in a particular variable x_i is the maximum exponent that x_i takes in any of the monomials in g.
- A *multivariate* polynomial is a polynomial with more than one variable; otherwise it is called a *univariate* polynomial. A multivariate polynomial is called a *multilinear* polynomial if the degree of the polynomial in each variable is at most one.
- A multivariate polynomial g over a finite field $\mathbb{F}$ is called *low-degree* if the degree of g in each variable is bounded above by a constant.

Definition 2.1. Suppose $f : \{0,1\}^\ell \to \mathbb{F}$ is a function that maps ℓ-bit strings to an element of $\mathbb{F}$. A *polynomial extension* of f is a low-degree ℓ-variate polynomial $\widetilde{f}(\cdot)$ such that $\widetilde{f}(x) = f(x)$ for all $x \in \{0,1\}^\ell$.

Definition 2.2. A *multilinear extension (MLE)* of a function $f : \{0,1\}^\ell \to \mathbb{F}$ is a low-degree polynomial extension where the extension is a multilinear polynomial.

It is well-known that every function $f : \{0,1\}^\ell \to \mathbb{F}$ has a unique multilinear extension, and similarly every ℓ-variate multilinear polynomial of $\mathbb{F}$ extends a unique function mapping $\{0,1\}^\ell \to \mathbb{F}$. In the rest of the document, for a function f, we use $\widetilde{f}$ to denote the unique MLE of f.

For an ℓ-variate multilinear polynomial $\widetilde{f}$ extending a function f, let v_f denote the 2^ℓ-dimensional vector containing all 2^ℓ evaluations of f. We refer to v_f as the representation of $\widetilde{f}$ in the Lagrange basis, because the

entries of v_f are the (unique) set of coefficients of $\widetilde{f}$ when $\widetilde{f}$ is expressed as a linear combination of Lagrange basis polynomials.

The sum-check protocol. Recall that the sum-check protocol [LFKN90] is an interactive proof system for proving claims of the form: $T = \sum_{x \in \{0,1\}^\ell} G(x)$, where G is ℓ-variate polynomial over $\mathbb{F}$ with the degree in each variable at most μ, and $T \in \mathbb{F}$. In the sum-check protocol, the verifier $\mathcal{V}_{SC}$ interacts with the prover $\mathcal{P}_{SC}$ over a sequence of ℓ rounds. At the end of this interaction, $\mathcal{V}_{SC}$ must evaluate $G(r)$ where $r \in_R \mathbb{F}^\ell$ is a vector of random field elements chosen by $\mathcal{V}_{SC}$. Other than evaluating $G(r)$, $\mathcal{V}_{SC}$ performs just $O(\ell \cdot \mu)$ field operations. As in prior work, it is natural to view the sum-check protocol as a mechanism to transform claims of the form $\sum_{x \in \{0,1\}^m} G(x) \stackrel{?}{=} T$ to the claim $G(r) \stackrel{?}{=} e$, where $e \in \mathbb{F}$. This is because in most cases, the verifier uses an auxiliary protocol to verify the latter claim.

Rank-1 constraint satisfiability (R1CS). R1CS refers to the following problem.

Definition 2.3. An R1CS instance is a tuple $(\mathbb{F}, A, B, C, M, N, \mathsf{io})$, where $A, B, C \in \mathbb{F}^{M \times M}$, $M \geq |\mathsf{io}| + 1$, io denotes the public input and output, and there are at most $N = \Omega(M)$ non-zero entries in each matrix.

We denote the set of R1CS (instance, witness) pairs as:

$$\mathcal{R}_{\text{R1CS}} = \{\langle (\mathbb{F}, A, B, C, \mathsf{io}, m, n), w\rangle : A \cdot (w, 1, \mathsf{io}) \circ B \cdot (w, 1, \mathsf{io}) = C \cdot (w, 1, \mathsf{io})\}$$

In the rest of the manuscript, WLOG, we assume that M and N are powers of 2, and that $M = |\mathsf{io}| + 1$. Throughout this manuscript, all logarithms are to base 2.

3 A linear-time polynomial IOP for R1CS from Spartan

This section recapitulates the results of Spartan [Set20] using a subsequent formalism, a polynomial IOP [BFS20]. This is a variant of IOPs [BCS16, RRR16] where in each round, the prover sends a polynomial as an oracle, and the verifier query may request an evaluation of the polynomial at a point in its domain.

The following theorem formalizes the polynomial IOP at the core of Spartan.

For an R1CS instance, $\mathbb{X} = (\mathbb{F}, A, B, C, M, N, \mathsf{io})$, we interpret the matrices A, B, C as functions mapping domain $\{0,1\}^{\log M} \times \{0,1\}^{\log M}$ to $\mathbb{F}$ in the natural way. That is, an input in $\{0,1\}^{\log M} \times \{0,1\}^{\log M}$ is interpreted as the binary representation of an index $(i, j) \in [M] \times [M]$, where $[M] \coloneqq \{1, \ldots, M\}$ and the function outputs the (i, j)'th entry of the matrix.

Theorem 1 ([Set20])**.** *For any finite field $\mathbb{F}$, there exists a polynomial IOP for $\mathcal{R}_{R1CS}$, with the following parameters, where M denotes the dimension of the R1CS coefficient matrices, and N denotes the number of non-zero entries in the matrices:*

- *soundness error is $O(\log M)/|\mathbb{F}|$*
- *round complexity is $O(\log M)$;*
- *at the start of the protocol, the prover sends a single $(\log M - 1)$-variate multilinear polynomial $\widetilde{W}$, and the verifier has a query access to three additional $2 \log M$-variate multilinear polynomials $\widetilde{A}, \widetilde{B}$, and $\widetilde{C}$;*
- *the verifier makes a single evaluation query to each of the four polynomials $\widetilde{W}, \widetilde{A}, \widetilde{B}$, and $\widetilde{C}$, and otherwise performs $O(\log M)$ operations over $\mathbb{F}$;*
- *the prescribed prover performs $O(N)$ operations over $\mathbb{F}$ to compute its messages over the course of the polynomial IOP (and to compute answers to the verifier's four queries to $\widetilde{W}, \widetilde{A}, \widetilde{B}$, and $\widetilde{C}$).*

Proof. Let $s = \log M$. For an R1CS instance, $\mathbb{X} = (\mathbb{F}, A, B, C, M, N, \mathsf{io})$ and a purported witness W, let $Z = (W, 1, \mathsf{io})$. As explained prior to the theorem statement, we can interpret A, B, C as functions mapping $\{0,1\}^s \times \{0,1\}^s$ to $\mathbb{F}$, and similarly we interpret Z and $(1, \mathsf{io})$ as functions with the following respective

signatures in the same manner: $\{0,1\}^s \to \mathbb{F}$ and $\{0,1\}^{s-1} \to \mathbb{F}$. It is easy to check that the MLE $\widetilde{Z}$ of Z satisfies

$$\widetilde{Z}(X_1, \ldots, X_{\log M}) = (1 - X_1) \cdot \widetilde{W}(X_2, \ldots, X_{\log M}) + X_1 \cdot \widetilde{(1, \mathsf{io})}(X_2, \ldots, X_{\log M}) \tag{1}$$

Indeed, the right hand side of Equation (1) is a multilinear polynomial, and it is easily checked that $\widetilde{Z}(x_1, \ldots, x_{\log M}) = Z(x_1, \ldots, x_{\log M})$ for all $x_1, \ldots, x_{\log M}$ (since the first half of the evaluations of Z are given by W and the second half are given by the vector $(1, \mathsf{io})$). Hence, the right hand side of Equation (1) must be the unique multilinear extension of Z.

From [Set20, Theorem 4.1], checking if $(\mathbb{X}, W) \in \mathcal{R}_{\text{R1CS}}$ is equivalent, except for a soundness error of $\log M/|\mathbb{F}|$ over the choice of $\tau \in \mathbb{F}^s$, to checking if the following identity holds:

$$0 \stackrel{?}{=} \left(\sum_{x \in \{0,1\}^s} \widetilde{eq}(\tau, x) \cdot \left(\left(\sum_{y \in \{0,1\}^s} \widetilde{A}(x, y) \cdot \widetilde{Z}(y) \right) \cdot \left(\sum_{y \in \{0,1\}^s} \widetilde{B}(x, y) \cdot \widetilde{Z}(y) \right) - \sum_{y \in \{0,1\}^s} \widetilde{C}(x, y) \cdot \widetilde{Z}(y) \right) \right), \tag{2}$$

where $\widetilde{eq}$ is the MLE of $eq : \{0,1\}^s \times \{0,1\}^s \to \mathbb{F}$:

$$eq(x, e) = \begin{cases} 1 & \text{if } x = e \\ 0 & \text{otherwise.} \end{cases}$$

That is, if $(\mathbb{X}, W) \in \mathcal{R}_{\text{R1CS}}$, then Equation (2) holds with probability 1 over the choice of τ, and if $(\mathbb{X}, W) \notin \mathcal{R}_{\text{R1CS}}$, then Equation (2) holds with probability at most $O(\log M/|\mathbb{F}|)$ over the random choice of τ.

Consider computing the right hand side of Equation (2) by applying the sum-check protocol to the polynomial

$$g(x) \coloneqq \widetilde{eq}(\tau, x) \cdot \left(\left(\sum_{y \in \{0,1\}^s} \widetilde{A}(x, y) \cdot \widetilde{Z}(y) \right) \cdot \left(\sum_{y \in \{0,1\}^s} \widetilde{B}(x, y) \cdot \widetilde{Z}(y) \right) - \sum_{y \in \{0,1\}^s} \widetilde{C}(x, y) \cdot \widetilde{Z}(y) \right).$$

From the verifier's perspective, this reduces the task of computing the right hand side of Equation (2) to the task of evaluating g at a random input $r_x \in \mathbb{F}^s$. Note that the verifier can evaluate $\widetilde{eq}(\tau, r_x)$ unassisted in $O(\log M)$ field operations, as it is easily checked that $\widetilde{eq}(\tau, r_x) = \prod_{i=1}^{s} (\tau_i r_{x,i} + (1 - \tau_i)(1 - r_{x,i}))$. With $\widetilde{eq}(\tau, r_x)$ in hand, $g(r_x)$ can be computed in $O(1)$ time given the three quantities

$$\sum_{y \in \{0,1\}^s} \widetilde{A}(x, y) \cdot \widetilde{Z}(y),$$

$$\sum_{y \in \{0,1\}^s} \widetilde{B}(x, y) \cdot \widetilde{Z}(y),$$

and

$$\sum_{y \in \{0,1\}^s} \widetilde{C}(x, y) \cdot \widetilde{Z}(y).$$

These three quantities can be computed by applying the sum-check protocol three more times in parallel, once to each of the following three polynomials (using the same random vector of field elements, $r_y \in \mathbb{F}^s$, in each of the three invocations):

$$\widetilde{A}(r_x, y) \cdot \widetilde{Z}(y),$$
$$\widetilde{B}(r_x, y) \cdot \widetilde{Z}(y),$$
$$\widetilde{B}(r_x, y) \cdot \widetilde{Z}(y).$$

To perform the verifier's final check in each of these three invocations of the sum-check protocol, it suffices for the verifier to evaluate each of the above 3 polynomials at the random vector r_y, which means it suffices for

the verifier to evaluate $\widetilde{A}(r_x, r_y)$, $\widetilde{B}(r_x, r_y)$, $\widetilde{C}(r_x, r_y)$, and $\widetilde{Z}(r_y)$. The first three evaluations can be obtained via the verifier's assumed query access to $\widetilde{A}$, $\widetilde{B}$, and $\widetilde{C}$. $\widetilde{Z}(r_y)$ can be obtained from one query to $\widetilde{W}$ and one query to $\widetilde{(1, \mathsf{io})}$ via Equation (1).

In summary, we have the following polynomial IOP:

1. $\mathcal{P} \to \mathcal{V}$: a $(\log M - 1)$-variate multilinear polynomial $\widetilde{W}$ as an oracle.
2. $\mathcal{V} \to \mathcal{P}$: $\tau \in_R \mathbb{F}^s$
3. $\mathcal{V} \leftrightarrow \mathcal{P}$: run the sum-check reduction to reduce the check in Equation (2) to checking if the following hold, where r_x, r_y are vectors in $\mathbb{F}^s$ chosen at random by the verifier over the course of the sum-check protocol:
   - $\widetilde{A}(r_x, r_y) \stackrel{?}{=} v_A$, $\widetilde{B}(r_x, r_y) \stackrel{?}{=} v_B$, and $\widetilde{C}(r_x, r_y) \stackrel{?}{=} v_C$; and
   - $\widetilde{Z}(r_y) \stackrel{?}{=} v_Z$.
4. $\mathcal{V}$:
   - check if $\widetilde{A}(r_x, r_y) \stackrel{?}{=} v_A$, $\widetilde{B}(r_x, r_y) \stackrel{?}{=} v_B$, and $\widetilde{C}(r_x, r_y) \stackrel{?}{=} v_C$, with one query to each of $\widetilde{A}, \widetilde{B}, \widetilde{C}$;
   - check if $\widetilde{Z}(r_y) \stackrel{?}{=} v_Z$ by checking if: $v_Z = (1 - r_y[1]) \cdot v_W + r_y[1] \cdot \widetilde{(io, 1)}(r_y[2..])$, where $r_y[2..]$ refers to a slice of r_y without the first element of r_y, and $v_W \leftarrow \widetilde{W}(r_y[2..])$ via an oracle query (see Equation (1)).

Completeness. Perfect completeness follows from perfect completeness of the sum-check protocol and the fact that Equation (2) holds with probability 1 over the choice of τ if $(\mathbb{X}, W) \in \mathcal{R}_{\text{R1CS}}$.

Soundness. Applying a standard union bound to the soundness error introduced by probabilistic check in Equation (2) with the soundness error of the sum-check protocol [LFKN90], we conclude that the soundness error for the depicted polynomial IOP as at most $O(\log M)/|\mathbb{F}|$.

Round and communication complexity. The sum-check protocol is applied 4 times (although 3 of the invocations occur in parallel and in practice combined into one [Set20]). In each invocation, the polynomial to which the sum-check protocol is applied has degree at most 3 in each variable, and the number of variables is $s = \log M$. Hence, the round complexity of the polynomial IOP is $O(\log M)$. Since each polynomial has degree at most 3 in each variable, the total communication cost is $O(\log M)$ field elements.

Verifier time. The asserted bounds on the verifier's runtime are immediate from the verifier's runtime in the sum-check protocol, and the fact that $\widetilde{eq}$ can be evaluated at any input $(\tau, r_x) \in \mathbb{F}^{2s}$ in $O(\log M)$ field operations.

Prover Time. [Set20] shows how to implement the prover's computation in the polynomial IOP in $O(N)$ $\mathbb{F}$-ops using prior techniques for linear-time sum-checks [Tha13, XZZ+19] (see also [Tha20, Section 7.5.2] for an exposition). This includes the time required to compute $\widetilde{A}(r_x, r_y)$, $\widetilde{B}(r_x, r_y)$, $\widetilde{C}(r_x, r_y)$, and $\widetilde{Z}(r_y)$ (i.e., to compute answers to the verifier's queries to the polynomials $\widetilde{A}$, $\widetilde{B}$, $\widetilde{C}$, and $\widetilde{Z}$). □

4 Linear-time commitments for multilinear polynomials

This section describes a polynomial commitment scheme from multilinear polynomials (see [Set20, BFS20] for definitions). This is essentially a direct application of the "tensor IOP to point-query IOP" transformation in

Bootle et al. [BCG20a] to the case of multilinear polynomial evaluations. Figure 2 compares the asymptotics of this scheme with prior polynomial commitment schemes.

For ease of exposition, instead of first constructing an argument system for special inner products and then using it for polynomial commitments (§1), we directly prove the existence of a linear-time polynomial commitment scheme, using results from [BCG20a]. Furthermore, our focus here is on multilinear polynomials, but the scheme described here generalizes to other polynomials such as univariate polynomials (see e.g., [Lee20]).

Background on tensor IOPs. Recall that an interactive oracle proof (IOP) [BCS16, RRR16] is a generalization of an interactive proof (IP), in which in each round the i prover sends a (possibly long) message string Π_i, and the verifier is given query access to Π_i. [BCG20a] define a generalization of IOPs called *tensor* IOPs. To distinguish tensor IOPs from the standard notion of IOPs that they generalize, [BCG20a] refers to standard IOPs as *point-query* IOPs.

Definition 4.1 ([BCG20a])**.** A $(\mathbb{F}, k, t)$-tensor IOP is an IOP except for the following modifications: (1) the prover's message in each round i consists of a message m_i that the verifier reads in full, followed optionally by a vector $\Pi_i \in \mathbb{F}^{k^t}$; and (2) a verifier query to Π_i may request the value $\langle q_1 \otimes q_2 \otimes \ldots \otimes q_t, \Pi_i \rangle$ for a chosen round i and chosen vectors $q_1, \ldots, q_t \in \mathbb{F}^k$.

Polynomial evaluation as a tensor query to the coefficient vector. For an ℓ-variate multilinear polynomial g represented in the Lagrange basis via a vector $z \in \mathbb{F}^n$ (where $n = 2^\ell$), given an evaluation point $r \in \mathbb{F}^\ell$, $g(r)$ can be evaluated using the following tensor product identity:

$$g(r) = \langle ((r_1, 1 - r_1) \otimes (r_2, 1 - r_2) \otimes \ldots \otimes (r_\ell, 1 - r_\ell)), z \rangle.$$

The above equality expresses $g(r)$ as the inner product of the coefficient vector z of the polynomial with another vector described as a tensor product of dimensionality ℓ. However, the identity generalizes to any dimensionality $t \leq \ell$ (for simplicity of presentation, we assume henceforth that $t|\ell$). Specifically, given $r \in \mathbb{F}^\ell$ and $t \in (1, \ell)$, there always exist vectors $q_1, \ldots, q_t \in \mathbb{F}^{n^{1/t}}$ (which can be computed from r using $O(n^{1/t})$ operations over $\mathbb{F}$) such that the following holds:

$$(r_1, 1 - r_1) \otimes (r_2, 1 - r_2) \otimes \ldots \otimes (r_\ell, 1 - r_\ell) = (q_1 \otimes q_2 \otimes \ldots \otimes q_t). \tag{3}$$

Equation (3) implies:

$$g(r) = \langle (q_1 \otimes q_2 \otimes \ldots \otimes q_t), z \rangle.$$

Indexed relations. The completeness and soundness requirements of IPs and IOPs are typically defined with respect to a language $\mathcal{L}$. For languages, completeness requires that for every input $x \in \mathcal{L}$, there exists a prover strategy that causes the verifier to output 1 with high probability. Soundness requires that for every input $x \notin \mathcal{L}$ and for every prover strategy, the verifier outputs 0 with high probability.

Recall that a relation $\mathcal{R}$ is a set of tuples $(\mathbb{X}, W)$, where $\mathbb{X}$ is the instance and W is the witness. Moreover, for any relation $\mathcal{R}$, there is a corresponding language $\mathcal{L}_\mathcal{R}$, which is the set of $\mathbb{X}$ for which there exists a witness W such that $(\mathbb{X}, W) \in \mathcal{R}$. In this section, we consider a generalized notion of relations called *indexed relations*, which are implicit in [Set20] but formalized in [COS20]. An indexed relation $\mathcal{R}$ is a set of triples $(\mathbb{I}, \mathbb{X}, W)$, where $\mathbb{I}$ is the index, $\mathbb{X}$ is the instance, and W is the witness. Naturally, there is an *indexed language* $\mathcal{L}_\mathcal{R}$ associated with an indexed relation, namely the set of tuples $(\mathbb{I}, \mathbb{X})$ for which there exists a witness W such that $(\mathbb{I}, \mathbb{X}, W) \in \mathcal{R}$.

Tensor IOPs for multilinear polynomial evaluation. Consider the following indexed relation.

Definition 4.2 (Multilinear evaluation indexed relation)**.** The indexed relation $\mathcal{R}_{MLE}$ is the set of all triples

$$(\mathbb{I}, \mathbb{X}, W) = ((\mathbb{F}, Z), (\ell, r, e), \perp),$$

where $\mathbb{F}$ is a finite field, $Z \in \mathbb{F}^N$ for $n = 2^\ell$, $r \in \mathbb{F}^\ell$, $v \in \mathbb{F}$, such that

$$e = \langle Z, (r_1, 1 - r_1) \otimes (r_2, 1 - r_2) \otimes \ldots \otimes (r_\ell, 1 - r_\ell) \rangle.$$

	commit time	commit size	$\mathcal{P}_{\mathsf{Eval}}$	π_{Eval}	$\mathcal{V}_{\mathsf{Eval}}$	assumptions
vSQL-VPD [ZGK+17]†	$O(N)$ $\mathbb{G}$-exp	$O_\lambda(1)$	$O(N)$ $\mathbb{G}$-exp	$O_\lambda(\log N)$	$O_\lambda(\log N)$	q-type
BP-PC [BBB+18, BGH19]	$O(N)$ $\mathbb{G}$-exp	$O_\lambda(1)$	$O(N)$ $\mathbb{G}$-exp	$O_\lambda(\log N)$	$O_\lambda(N)$	DLOG
Hyrax-PC [WTS+18]	$O(N)$ $\mathbb{G}$-exp	$O_\lambda(\sqrt{N})$	$O(N)$ $\mathbb{F}$-ops	$O_\lambda(\log N)$	$O_\lambda(\sqrt{N})$	DLOG
Virgo-VPD [ZXZS20]	$O(N \log N)$ $\mathbb{F}$-ops	$O_\lambda(1)$	$O(N \log N)$ $\mathbb{F}$-ops	$O_\lambda(\log^2 N)$	$O_\lambda(\log^2 N)$	CRHF
Kopis-PC [SL20]	$O(N)$ $\mathbb{G}$-exp	$O_\lambda(1)$	$O(N)$ $\mathbb{F}$-ops	$O_\lambda(\log N)$	$O_\lambda(\sqrt{N})$	SXDH
Dory-PC [Lee20]	$O(N)$ $\mathbb{G}$-exp	$O_\lambda(1)$	$O(N)$ $\mathbb{F}$-ops	$O_\lambda(\log N)$	$O_\lambda(\log N)$	SXDH
Scheme in Section 4 (distilled from [BCG20a])	$O(N)$ $\mathbb{F}$-ops $O(N)$ hashes	$O_\lambda(1)$	$O(N)$ $\mathbb{F}$-ops	$O_\lambda(N^{1/t})$	$O_\lambda(N^{1/t})$	CRHF

† Requires trusted setup

Figure 2: Asymptotic efficiency of prior polynomial commitment schemes. The depicted costs are for an ℓ-variate multilinear polynomial ($N = 2^\ell$) over a finite field $\mathbb{F}$, where $|\mathbb{F}| = \exp(\Theta(\lambda))$ and $\lambda \geq \omega(\log N)$ is the security parameter. $\mathbb{F}$-ops refers to field multiplications or additions; $\mathbb{G}$-exp refers to an exponentiation in a group $\mathbb{G}$ whose scalar field is $\mathbb{F}$. The parameter t refers to a constant. For non-interactivity, all schemes except vSQL-VPD [ZGK+17] assume the random oracle model; vSQL-VPD assumes q-type, knowledge of exponent assumptions. Furthermore, to achieve a linear-time commit time by using a linear-time hash function for Merkle hashes, our scheme requires assuming the hardness of certain lattice problems, which are not listed in the "assumptions" column for brevity (§1). Finally, one can reduce the verifier time and proof size in our scheme by using one layer of proof composition with an existing zkSNARK [Gro16, Set20, SL20], but we do not depict those variants here.

Theorem 2. *For a finite field $\mathbb{F}$ and positive integers k, t, there exists a $(\mathbb{F}, k, t)$-tensor IOP for $\mathcal{R}_{MLE}$ with the following parameters, where $N = 2^\ell = k^t$ and ℓ is a parameter in an instance:*

- *soundness error is 0;*
- *round complexity is $O(1)$;*
- *index length is $O(N)$ elements in $\mathbb{F}$;*
- *query complexity is $O(1)$;*
- *the prover performs $O(N)$ operations over $\mathbb{F}$; and*
- *the verifier performs $O(N^{1/t})$ operations over $\mathbb{F}$, given a tensor-query access to the index.*

Proof. Suppose that the evaluation point is $r \in \mathbb{F}^\ell$. Consider the following tensor IOP.

1. $\mathbb{I}$: a vector Π specifying the coefficients of the ℓ-variate multilinear polynomial in the Lagrange basis.
2. $\mathcal{V} \to \mathcal{P}$: a tensor query $(q_1, \ldots, q_t)$ to $\mathcal{P}$, where $(r_1, 1 - r_1) \otimes (r_2, 1 - r_2) \otimes \ldots \otimes (r_\ell, 1 - r_\ell) = (q_1 \otimes q_2 \otimes \ldots \otimes q_t)$. This query is answered with an evaluation $e \in \mathbb{F}$ equal to $\langle (q_1, \ldots, q_t), \Pi \rangle$.

The index consists of $O(N)$ elements in $\mathbb{F}$. The round complexity and the query complexity of the depicted tensor IOP is $O(1)$; soundness error is 0. In terms of time complexity, the verifier starts with an evaluation point $r \in \mathbb{F}^\ell$, which it transforms to a set of vectors $(q_1, \ldots q_t)$, where each $q_i \in \mathbb{F}^{N^{1/t}}$; this costs $O(N^{1/t})$ operations over $\mathbb{F}$ to the verifier. The prover performs $O(N)$ operations over $\mathbb{F}$ to compute a response to the tensor query. □

The theorem below requires a linear code, and to achieve the stated asymptotics, the linear code must support linear-time encoding. As noted in prior work, explicit constructions of such codes are known [Spi96, DI14]. One can also use Reed-Solomon codes, but it introduces a superlinear encoding time and hence a superlinear prover. In fact, in the case of $t = 2$ and using Reed-Solomon codes, the tensor IOP (Theorem 3) and resulting polynomial commitment scheme (Theorem 4) given below is implicit in Ligero [AHIV17] and explicitly distilled in [Tha20, Section 9.6]. Meanwhile, the case of $t = 2$ using general linear codes is implicitly considered in [BCG+17].

Theorem 3. *For security parameter λ, given an $(\mathbb{F}, k, t)$-tensor IOP for $\mathcal{R}_{MLE}$ with $N = 2^\ell = k^t$ (where ℓ is the size parameter) and fixed value of t, and a linear code over $\mathbb{F}$ with rate $\rho = k/n$, relative distance $\delta = d/n$, and encoding time k, there exists a point-query IOP for $\mathcal{R}_{MLE}$ with the following parameters:*

- *soundness error is $O(d^t/|\mathbb{F}| + (1 - \frac{\delta^t}{2})^\lambda)$;*
- *round complexity is $O(1)$;*
- *index length is $O(N)$ elements in $\mathbb{F}$;*
- *query complexity is $O(N^{1/t})$;*
- *the indexer performs $O(N)$ operations over $\mathbb{F}$;*
- *the prover performs $O(N)$ operations over $\mathbb{F}$; and*
- *the verifier performs $O(N^{1/t})$ operations over $\mathbb{F}$.*

Proof. Applying [BCG20a, Theorem 3] to the tensor IOP from Theorem 2 provides the desired result. □

Theorem 4. *For security parameter λ and a positive integer t, given a hash function that can compute a Merkle-hash of N elements of $\mathbb{F}$ with the same time complexity as $O(N)$ $\mathbb{F}$-ops, there exists a linear-time polynomial commitment scheme for multilinear polynomials. Specifically, there exists an algorithm that, given as input the coefficient vector of an ℓ-variate multilinear polynomial over $\mathbb{F}$ over the Lagrange basis, with $N = 2^\ell$, commits to the polynomial, where:*

- *the size of the commitment is $O_\lambda(1)$; and*
- *the running time of the commit algorithm is $O(N)$ operations over $\mathbb{F}$.*

Furthermore, there exists a non-interactive argument of knowledge in the random oracle model to prove the correct evaluation of a committed polynomial with the following parameters:

- *the running time of the prover is $O(N)$ operations over $\mathbb{F}$;*
- *the running time of the verifier is $O_\lambda(N^{1/t})$ operations over $\mathbb{F}$; and*
- *the proof size is $O_\lambda(N^{1/t})$.*

Proof. The desired commit algorithm and its claimed efficiency follows from from applying the BCS transform [BCS16] (with a linear-time hash function [AHI+17, BCG+17]) to the indexer in the point-query IOP from Theorem 3 (obtained by using any linear-time encodable linear error-correcting code of constant rate and relative distance). Similarly, the non-interactive argument of knowledge along with its claimed efficiency follows from applying the BCS transform [BCS16] (with a linear-time hash function [AHI+17, BCG+17]) to the point-query IOP from Theorem 3. □

Theorem 4 obtains the claimed polynomial commitment scheme using the tensor-IOP-to-point-IOP transformation of [BCG20a] as a black box, and hence provides the reader with essentially no information about how the polynomial commitment scheme actually operates. For concreteness, we provide a self-contained description of the polynomial commitment scheme and its analysis for the case of $t = 2$ in Appendix A. The analysis for $t = 2$ is far simpler than the more general case of arbitrary t considered in [BCG20a].

5 Linear-time commitments for sparse multilinear polynomials

To construct SNARKs under both routes (§1), we need a commitment scheme for multilinear polynomials $\widetilde{A}$, $\widetilde{B}$, and $\widetilde{C}$ capturing the R1CS instance. Specifically, we desire a polynomial commitment scheme that when applied to these polynomials, the time to commit and prove an evaluation are both $O(N)$.

The previous section gave a commitment scheme for ℓ-variate multilinear polynomials where the prover runs in time $O(2^\ell)$. However, $\widetilde{A}$, $\widetilde{B}$, and $\widetilde{C}$ are each defined over $\ell = 2 \log M$ variables. So if the commitment

scheme is applied naively to these polynomials, the runtime of the committer is $O(M^2)$, which is superlinear in the size of the R1CS instance (unless a constant fraction of the entries of the matrices A, B, C are non-zero). We call this an efficient commitment scheme for "dense" polynomials, since the prover's runtime is linear in the number of coefficients of the polynomial, irrespective of how many of those coefficients are non-zero.

Fortunately, for each of these three polynomials, only $O(N)$ of coefficients over the Lagrange basis are non-zero. If $N \ll M^2$, we call such polynomials *sparse*. What we require is a commitment scheme for sparse polynomials in which the committer runs in time proportional to the number of non-zero coefficients in both the commitment phase and the evaluation phase. Spartan [Set20] gave a technique that transforms any efficient polynomial commitment scheme for dense polynomials into an efficient polynomial commitment scheme for sparse polynomials. By applying Spartan's construction to the polynomial commitment scheme of the previous section, we obtain our desired sparse polynomial commitment scheme.

We first state the result from Spartan [Set20, Lemma 7.6] in a more general form; below, provide a detailed proof of this result for completeness. An alternate perspective on this result is in [Tha20, §13].

Theorem 5 ([Set20])**.** *Given a polynomial commitment scheme for $\log M$-variate multilinear polynomials with the following parameters (where M is a positive integer and WLOG a power of 2):*

- *the size of the commitment is $\mathsf{c}(M)$;*
- *the running time of the commit algorithm is $\mathsf{tc}(M)$;*
- *the running time of the prover to prove a polynomial evaluation is $\mathsf{tp}(M)$;*
- *the running time of the verifier to verify a polynomial evaluation is $\mathsf{tv}(M)$; and*
- *the proof size is $\mathsf{p}(M)$,*

there exists a polynomial commitment scheme for $2\log M$-variate multilinear polynomials that evaluate to a non-zero value at at most $N = \Omega(M)$ locations over the Boolean hypercube $\{0,1\}^{2\log M}$, with the following parameters, assuming that the commit algorithm is run by an honest entity:

- *the size of the commitment is $O(\mathsf{c}(N))$;*
- *the running time of the commit algorithm is $O(\mathsf{tc}(N))$;*
- *the running time of the prover to prove a polynomial evaluation is $O(\mathsf{tp}(N))$;*
- *the running time of the verifier to verify a polynomial evaluation is $O(\mathsf{tv}(N))$; and*
- *the proof size is $O(\mathsf{p}(N))$.*

The following corollary follows from applying the above theorem to the polynomial commitment scheme for $\log M$-variate multilinear polynomials (Theorem 4).

Corollary 1. *Given a polynomial commitment scheme for $\log M$-variate multilinear polynomials with the following parameters (where M is a positive integer and WLOG a power of 2, t is a constant, and λ is the security parameter):*

- *the size of the commitment is $O_\lambda(1)$;*
- *the running time of the commit algorithm is $O(M)$ operations over $\mathbb{F}$;*
- *the running time of the prover to prove a polynomial evaluation is $O(M)$ operations over $\mathbb{F}$;*
- *the running time of the verifier to verify a polynomial evaluation is $O_\lambda(M^{1/t})$ operations over $\mathbb{F}$; and*
- *the proof size is $O_\lambda(M^{1/t})$,*

there exists a polynomial commitment scheme for $2\log M$-variate multilinear polynomials that evaluate to a non-zero value at at most $N = \Omega(M)$ locations over the Boolean hypercube $\{0,1\}^{2\log M}$, with the following parameters, assuming that the commit algorithm is run by an honest entity:

- *the size of the commitment is $O_\lambda(1)$;*

– *the running time of the commit algorithm is $O(N)$ operations over $\mathbb{F}$;*

– *the running time of the prover to prove a polynomial evaluation is $O(N)$ operations over $\mathbb{F}$;*

– *the running time of the verifier to verify a polynomial evaluation is $O_\lambda(N^{1/t})$ operations over $\mathbb{F}$; and*

– *the proof size is $O_\lambda(N^{1/t})$.*

Representing sparse polynomials with dense polynomials. Let D denote a $2\log M$-variate multilinear polynomial that evaluates to a non-zero value at at most $N = \Omega(M)$ locations over $\{0,1\}^{2\log M}$. For any $r \in \mathbb{F}^{2\log M}$, we can express the evaluation of $D(r)$ as follows. Interpret $r \in \mathbb{F}^{2\log M}$ as a tuple (r_x, r_y) in a natural manner, where $r_x, r_y \in \mathbb{F}^{\log M}$.

$$D(r_x, r_y) = \sum_{(i,j) \in \{0,1\}^{\log M} \times \{0,1\}^{\log M} \,:\, D(i,j) \neq 0} D(i,j) \cdot \widetilde{eq}(i, r_x) \cdot \widetilde{eq}(j, r_y). \tag{4}$$

Claim 1. *Given a $2\log M$-variate multilinear polynomial D that evaluates to a non-zero value at at most N locations over $\{0,1\}^{2\log M}$, there exist three $\log N$-variate multilinear polynomials $\mathsf{row}, \mathsf{col}, \mathsf{val}$ such that the following holds for all $r_x, r_y \in \mathbb{F}^s$:*

$$D(r_x, r_y) = \sum_{k \in \{0,1\}^{\log N}} \mathsf{val}(k) \cdot \widetilde{eq}(\mathsf{row}(k), r_x) \cdot \widetilde{eq}(\mathsf{col}(k), r_y). \tag{5}$$

Moreover, the polynomials' coefficients in the Lagrange basis can be computed in $O(N)$ time.

Proof. Let $R, C, V \in \mathbb{F}^n$. Since D evaluates to a non-zero value at at most N locations over $\{0,1\}^{2\log M}$, D can be represented uniquely with N tuples of the form $(i, j, D(i,j)) \in (\{0,1\}^{\log M}, \{0,1\}^{\log M}, \mathbb{F})$. By using a natural injection from $\{0,1\}^{\log M}$ to $\mathbb{F}$, we can view entries in these tuples as elements of $\mathbb{F}$. Furthermore, these tuples can be represented with three N-sized vectors R, C, V, where tuple k (for all $k \in [N]$) is stored across the three vectors at the kth location in the vector, i.e., the first entry in the tuple is stored in R, the second entry in C, and the third entry in V. Take row as the unique MLE of R viewed as a function $\{0,1\}^{\log N} \to \mathbb{F}$. Similarly, col is the unique MLE of C, and val is the unique MLE of V. The claim holds by inspection since Equations (4) and (5) are both multilinear polynomials in r_x and r_y and agree with each other at every pair $r_x, r_y \in \{0,1\}^{\log M}$. □

A first attempt at the commit phase. Here is a first attempt at designing the commit phase. To commit to D, the committer can send commitments to the three $\log N$-variate multilinear polynomials $\mathsf{row}, \mathsf{col}, \mathsf{val}$ from Claim 1. Using the provided polynomial commitment scheme, this costs $O(N)$ finite field operations, and the size of the commitment to D is $O_\lambda(1)$. As we will see, to aid in the evaluation phase, we will ultimately have to extend the commit phase to include commitments to several additional polynomials.

A first attempt at the evaluation phase. Given $r_x, r_y \in \mathbb{F}^{\log M}$, to prove an evaluation of a committed polynomial, i.e., to prove that $D(r_x, r_y) = v$ for a purported evaluation $v \in \mathbb{F}$, consider the following polynomial IOP, where assume that the verifier has oracle access to the three $\log N$-variate multilinear polynomial oracles that encode D $(\mathsf{row}, \mathsf{col}, \mathsf{val})$:

1. $\mathcal{P} \to \mathcal{V}$: two $\log N$-variate multilinear polynomials E_{rx} and E_{ry} as oracles. These polynomials are purported to respectively equal $\widetilde{eq}(\mathsf{row}(k), r_x)$ and $\widetilde{eq}(\mathsf{row}(k), r_y)$.
2. $\mathcal{V} \leftrightarrow \mathcal{P}$: run the sum-check reduction to reduce the check that

$$v = \sum_{k \in \{0,1\}^{\log N}} \mathsf{val}(k) \cdot E_{\mathsf{rx}}(k) \cdot E_{\mathsf{ry}}(k)$$

to checking if the following hold, where $r_z \in \mathbb{F}^{\log N}$ is chosen at random by the verifier over the course of the sum-check protocol:

- $\mathsf{val}(r_z) \stackrel{?}{=} v_{\mathsf{val}}$;
- $E_{\mathsf{rx}}(r_z) \stackrel{?}{=} v_{E_{\mathsf{rx}}}$ and $E_{\mathsf{ry}}(r_z) \stackrel{?}{=} v_{E_{\mathsf{ry}}}$. Here, v_{val}, $v_{E_{\mathsf{rx}}}$, and $v_{E_{\mathsf{ry}}}$ are values provided by the prover at the end of the sum-check protocol.

3. $\mathcal{V}$: check if the three equalities hold with an oracle query to each of $\mathsf{val}, E_{\mathsf{rx}}, E_{\mathsf{ry}}$.

If the prover is honest, it is easy to see that it can convince the verifier about the correct of evaluations of D. Unfortunately, the two oracles that the prover sends in the first step of the depicted polynomial IOP can be completely arbitrary. To fix, this, $\mathcal{V}$ must *additionally* check that the following two conditions hold.

- $\forall k \in \{0,1\}^{\log N}$, $E_{\mathsf{rx}}(k) = \widetilde{eq}(\mathsf{row}(k), r_x)$; and
- $\forall k \in \{0,1\}^{\log N}$, $E_{\mathsf{ry}}(k) = \widetilde{eq}(\mathsf{col}(k), r_y)$.

A core insight of Spartan [Set20] is to check these two conditions using memory-checking techniques [BEG+91]. In particular, the RHS in each of the above conditions can be viewed as N memory lookups over an M-sized memory, initialized to contain the values $\{\widetilde{eq}(\mathsf{row}(k), r_x) \colon k \in \{0,1\}^{\log N}\}$ and $\{\widetilde{eq}(\mathsf{row}(k), r_x) \colon k \in \{0,1\}^{\log N}\}$ (note that all of these values can be computed in $O(N)$ total time using standard techniques).

Specifically, focusing on the RHS of the first condition, the M-sized memory $\mathsf{mem}_{\mathsf{rx}}$ is initialized to satisfy $\mathsf{mem}_{\mathsf{rx}}[i] = \widetilde{eq}(i, r_x)$ for all $i \in \{0,1\}^{\log M}$, and for memory lookup $k \in [N]$, the memory address that is read is $\mathsf{row}(k)$. Similarly, in the second condition, the M-sized memory $\mathsf{mem}_{\mathsf{ry}}$ is the evaluations of $\widetilde{eq}(j, r_y)$ for all $j \in \{0,1\}^{\log M}$, and for memory lookup $k \in [N]$, the memory address read is $\mathsf{col}(k)$.

We take a detour to introduce prior results that we rely on here.

Detour: offline memory checking. Recall that in the offline memory checking algorithm of [BEG+91], a *trusted checker* issues operations to an untrusted memory. For our purposes, it suffices to consider only operation sequences in which each memory address is initialized to a certain value, and all subsequent operations are read operations. To enable efficient checking using set-fingerprinting techniques, the memory is modified so that in addition to storing a value at each address, the memory also stores a timestamp with each address. Moreover, each read operation is followed by a write operation that updates the timestamp associated with that address (but not the value stored there). This is captured in the codebox below.

Local state of the checker:

- a timestamp counter ts initialized to 0;
- Two sets: RS and WS, which are initialized as follows.[13] $RS = \{\}$, and for an M-sized memory, WS is initialized to the following set of tuples: for all $i \in [M]$, the tuple $(i, v_i, 0)$ is included in WS, where v_i is the value stored at address i, and the third entry in the tuple, 0, is an "initial timestamp" associated with the value (intuitively capturing the notion that v_i was written to address i at time step 0, i.e., at initialization).

Read operations and an invariant. For a read operation at address a, suppose that the untrusted memory responds with a value-timestamp pair (v, t). Then the checker updates its local state as follows:

1. $RS \leftarrow RS \cup \{(a, v, t)\}$;
2. $ts \leftarrow \max(ts, t) + 1$;
3. store (v, ts) at address a in the untrusted memory; and

[13] The checker in [BEG+91] maintains a fingerprint of these sets, but for our exposition, we let the checker maintain full sets.

4. $WS \leftarrow WS \cup \{(a, v, ts)\}$.

The following claim captures the invariant maintained on the sets of the checker:

Claim 2. *There exists a set S with cardinality M consisting of tuples of the form (k, v_k, t_k) for all $k \in [M]$ such that $WS = RS \cup S$ if and only if for every read operation the untrusted memory returns the tuple last written to that location.*

Proof. A proof of a more general claim is at [SAGL18b, Lemma C.1]. □

Observe that if the untrusted memory is honest, S can be constructed trivially from the current state of the memory. It is simply the current state of the memory viewed as a set of address-value-timestamp tuples.

Timestamp polynomials. To aid the polynomial evaluation proof of the sparse polynomial commitment scheme, the commit algorithm of the sparse polynomial commitment commits to additional multilinear polynomials, beyond val, row, and col. We now describe these additional polynomials and how they are constructed.

Observe that given the size of memory M and a list of N addresses involved in read operations, one can locally compute three vectors $T_r \in \mathbb{F}^N, T_w \in \mathbb{F}^N, T_f \in \mathbb{F}^M$ defined as follows. For $k \in [N]$, $T_r[k]$ stores the timestamp that would have been returned by the untrusted memory if it were honest during the kth read operation. Let $T_w[k]$ store the timestamp that the checker stores in step (3) of its specification when processing the kth read operation. Similarly, for $k \in [M]$, let $T_f[k]$ store the final timestamp stored at memory location k of the untrusted memory (if the untrusted memory were honest) at the termination of the N read operations. Computing these three vectors requires computation comparable to $O(N)$ operations over $\mathbb{F}$.

Let $\mathsf{read} = \widetilde{T_r}, \mathsf{write} = \widetilde{T_w}, \mathsf{final} = \widetilde{T_f}$. We refer these polynomials as *timestamp polynomials*, which are unique for a given memory size M and a list of N addresses involved in read operations.

The actual commit algorithm. Given a $2 \log M$-variate multilinear polynomial D that evaluates to a non-zero value at at most N locations over $\{0,1\}^{2 \log M}$, the commitment algorithm commits to D by committing to seven $\log N$-variate multilinear polynomials $(\mathsf{row}, \mathsf{col}, \mathsf{val}, \mathsf{read}_{\mathsf{row}}, \mathsf{write}_{\mathsf{row}}, \mathsf{read}_{\mathsf{col}}, \mathsf{write}_{\mathsf{col}})$, two $\log M$-variate multilinear polynomials $(\mathsf{final}_{\mathsf{row}}, \mathsf{final}_{\mathsf{col}})$, where $\mathsf{row}, \mathsf{col}, \mathsf{val}$ are described in Claim 1, and $(\mathsf{read}_{\mathsf{row}}, \mathsf{write}_{\mathsf{row}}, \mathsf{final}_{\mathsf{row}})$ and $(\mathsf{read}_{\mathsf{col}}, \mathsf{write}_{\mathsf{col}}, \mathsf{final}_{\mathsf{col}})$ are respectively the timestamp polynomials for the N addresses specified by row and col over a memory of size M.

There are two crucial subtleties unique to the setting of holography that we are exploiting in the above commitment procedure. First, that the timestamp polynomials $(\mathsf{read}_{\mathsf{row}}, \mathsf{write}_{\mathsf{row}}, \mathsf{final}_{\mathsf{row}}, \mathsf{read}_{\mathsf{col}}, \mathsf{write}_{\mathsf{col}}, \mathsf{final}_{\mathsf{col}})$ depend only on the sparse polynomial being committed, and in particular not on the evaluation point (r_x, r_y). Second, in the holography context, the commit algorithm is run by an honest entity. This means that the commit phase does not need to include any proof that the committed timestamp polynomials actually equal the polynomials $\widetilde{T_r}$, $\widetilde{T_w}$, and $\widetilde{T_f}$ described above. This appears to be essential for avoiding superlinear operations such as sorting.[14]

In total, using the provided polynomial commitment, the commit algorithm incurs $O(N)$ finite field operations, and the commitment size is $O_\lambda(1)$.

[14] If desired in other contexts, such a proof can be produced with $O(N \log N)$ operations over $\mathbb{F}$. We are unaware of a mechanism to produce a proof faster than this. Straightforward approaches either require breaking field elements into their binary representations (which introduces $\log N$ factor to the number of coefficients of the polynomials being committed) or require the prover to sort a vector of timestamps, which is a superlinear-time operation. The challenging issue is to ensure after every read operation, the timestamp associated with the memory location gets updated to the maximum of the returned timestamp ts and the current timestamp t.

The actual evaluation procedure. To prove the correct evaluation of a $2\log M$-variate multilinear polynomial D, in addition to performing the polynomial IOP depicted earlier in the proof, the core idea is to check if the two oracles sent by the prover satisfy the conditions identified earlier using Claim 2.

Claim 3. *Given a $2\log M$-variate multilinear polynomial, suppose that $(\mathsf{row}, \mathsf{col}, \mathsf{val}, \mathsf{read}_{\mathsf{row}}, \mathsf{write}_{\mathsf{row}}, \mathsf{final}_{\mathsf{row}}, \mathsf{read}_{\mathsf{col}}, \mathsf{write}_{\mathsf{col}}, \mathsf{final}_{\mathsf{col}})$ denote multilinear polynomials committed by the commit algorithm. Then, for any $r_x \in \mathbb{F}^{\log M}$, checking that $\forall k \in \{0,1\}^{\log N}$, $E_{\mathsf{rx}}(k) = \widetilde{eq}(\mathsf{row}(k), r_x)$ is equivalent to checking $WS = RS \cup S$, where*

- $WS = \{(i, \widetilde{eq}(i, r_x), 0) \colon i \in [M]\} \cup \{(\mathsf{row}(k), E_{\mathsf{rx}}(k), \mathsf{write}_{\mathsf{row}}(k)) \colon k \in [N]\}$;
- $RS = \{(\mathsf{row}(k), E_{\mathsf{rx}}(k), \mathsf{read}_{\mathsf{row}}(k)) \colon k \in [N]\}$; *and*
- $S = \{(i, \widetilde{eq}(i, r_x), \mathsf{final}_{\mathsf{row}}(i)) \colon i \in [M]\}$.

Similarly, for any $r_y \in \mathbb{F}^{\log M}$, checking that $\forall k \in \{0,1\}^{\log N}$, $E_{\mathsf{ry}}(k) = \widetilde{eq}(\mathsf{col}(k), r_y)$ is equivalent to checking $WS' = RS' \cup S'$, where

- $WS' = \{(j, \widetilde{eq}(j, r_y), 0) \colon j \in [M]\} \cup \{(\mathsf{col}(k), E_{\mathsf{ry}}(k), \mathsf{write}_{\mathsf{col}}(k)) \colon k \in [N]\}$;
- $RS' = \{(\mathsf{col}(k), E_{\mathsf{ry}}(k), \mathsf{read}_{\mathsf{col}}(k)) \colon k \in [N]\}$; *and*
- $S' = \{(j, \widetilde{eq}(j, r_y), \mathsf{final}_{\mathsf{col}}(i)) \colon j \in [M]\}$.

Proof. The desired result follows from a straightforward application of the invariant in Claim 2. □

There is no direct way to prove that the checks on sets in Claim 3 hold. Instead, we rely on public-coin, multiset hash functions to compress RS, WS, and S into a single element of $\mathbb{F}$ each. Specifically:

Claim 4 ([Set20])**.** *Given two multisets A, B where each element is from $\mathbb{F}^3$, checking that $A = B$ is equivalent to checking the following, except for a soundness error of $O(|A| + |B|)/|\mathbb{F}|)$ over the choice of γ, τ: $\mathcal{H}_{\tau,\gamma}(A) = \mathcal{H}_{\tau,\gamma}(B)$, where $\mathcal{H}_{\tau,\gamma}(A) = \prod_{(a,v,t)\in A}(h_\gamma(a,v,t) - \tau)$, and $h_\gamma(a,v,t) = a \cdot \gamma^2 + v \cdot \gamma + t$. That is, if $A = B$, $\mathcal{H}_{\tau,\gamma}(A) = \mathcal{H}_{\tau,\gamma}(B)$ with probability 1 over randomly chosen values τ and γ in $\mathbb{F}$, while if $A \neq B$, then $\mathcal{H}_{\tau,\gamma}(A) = \mathcal{H}_{\tau,\gamma}(B)$ with probability at most $O(|A| + |B|)/|\mathbb{F}|)$.*

We are now ready to depict a polynomial IOP for proving evaluations of a committed sparse multilinear polynomial. Given $r_x, r_y \in \mathbb{F}^{\log M}$, to prove that $D(r_x, r_y) = v$ for a purported evaluation $v \in \mathbb{F}$, consider the following polynomial IOP, where assume that the verifier has an oracle access to multilinear polynomial oracles that encode D (namely, $\mathsf{row}, \mathsf{col}, \mathsf{val}, \mathsf{read}_{\mathsf{row}}, \mathsf{write}_{\mathsf{row}}, \mathsf{final}_{\mathsf{row}}, \mathsf{read}_{\mathsf{col}}, \mathsf{write}_{\mathsf{col}}, \mathsf{final}_{\mathsf{col}}$).

1. $\mathcal{P} \to \mathcal{V}$: two $\log N$-variate multilinear polynomials E_{rx} and E_{ry} as oracles.
2. $\mathcal{V} \leftrightarrow \mathcal{P}$: run the sum-check reduction to reduce the check that
$$v = \sum_{k \in \{0,1\}^{\log N}} \mathsf{val}(k) \cdot E_{\mathsf{rx}}(k) \cdot E_{\mathsf{ry}}(k)$$
to checking that the following equations hold, where $r \in \mathbb{F}^{\log N}$ chosen at random by the verifier over the course of the sum-check protocol:
   - $\mathsf{val}(r_z) \stackrel{?}{=} v_{\mathsf{val}}$; and
   - $E_{\mathsf{rx}}(r_z) \stackrel{?}{=} v_{E_{\mathsf{rx}}}$ and $E_{\mathsf{ry}}(r_z) \stackrel{?}{=} v_{E_{\mathsf{ry}}}$. Here, v_{val}, $v_{E_{\mathsf{rx}}}$ and $v_{E_{\mathsf{ry}}}$ are values provided by the prover at the end of the sum-check protocol.
3. $\mathcal{V}$: check if the three obligations hold with an oracle query each to $\mathsf{val}, E_{\mathsf{rx}}, E_{\mathsf{ry}}$.
4. // The following steps check if E_{rx} is well-formed
5. $\mathcal{V} \to \mathcal{P}$: $\tau, \gamma \in_R \mathbb{F}$.

6. $\mathcal{V} \leftrightarrow \mathcal{P}$: run the layered sum-check reduction for "grand products" [Tha13, §5.3.1] to reduce the check that $\mathcal{H}_{\tau,\gamma}(WS) = \mathcal{H}_{\tau,\gamma}(RS) \cdot \mathcal{H}_{\tau,\gamma}(S)$, where RS, WS, S are as defined in Claim 3 and $\mathcal{H}$ is defined in Claim 4 to checking if the following hold, where $r_M \in \mathbb{F}^{\log M}, r_N \in \mathbb{F}^{\log N}$ chosen at random by the verifier over the course of the sum-check protocol:
   - $\widetilde{eq}(r_M, r_x) \stackrel{?}{=} v_{eq}$
   - $E_{\mathsf{rx}}(r_N) \stackrel{?}{=} v_{E_{\mathsf{rx}}}$
   - $\mathsf{row}(r_N) \stackrel{?}{=} v_{\mathsf{row}}$; $\mathsf{write}_{\mathsf{row}}(r_N) \stackrel{?}{=} v_{\mathsf{write}_{\mathsf{row}}}$; $\mathsf{read}_{\mathsf{row}}(r_N) \stackrel{?}{=} v_{\mathsf{read}_{\mathsf{row}}}$; and $\mathsf{final}_{\mathsf{row}}(r_M) \stackrel{?}{=} v_{\mathsf{final}_{\mathsf{row}}}$
7. $\mathcal{V}$: directly check if the first equality holds, which can be done with $O(\log M)$ field operations; check the remaining equations hold with an oracle query to each of $E_{\mathsf{rx}}, \mathsf{row}, \mathsf{write}_{\mathsf{row}}, \mathsf{read}_{\mathsf{row}}, \mathsf{final}_{\mathsf{row}}$.
8. // The following steps check if E_{ry} is well-formed
9. $\mathcal{V} \to \mathcal{P}$: $\tau', \gamma' \in_R \mathbb{F}$.
10. $\mathcal{V} \leftrightarrow \mathcal{P}$: run the sum-check reduction for "grand products" [Tha13, SL20] to reduce the check that $\mathcal{H}_{\tau',\gamma'}(WS') = \mathcal{H}_{\tau',\gamma'}(RS') \cdot \mathcal{H}_{\tau',\gamma'}(S')$, where RS', WS', S' are as defined in Claim 3 and $\mathcal{H}$ is defined in Claim 4 to checking if the following hold, where $r'_M \in \mathbb{F}^{\log M}, r'_N \in \mathbb{F}^{\log N}$ chosen at random by the verifier over the course of the sum-check protocol:
    - $\widetilde{eq}(r'_M, r_y) \stackrel{?}{=} v'_{eq}$
    - $E_{\mathsf{ry}}(r'_N) \stackrel{?}{=} v_{E_{\mathsf{ry}}}$
    - $\mathsf{col}(r'_N) \stackrel{?}{=} v_{\mathsf{col}}$; $\mathsf{write}_{\mathsf{col}}(r'_N) \stackrel{?}{=} v_{\mathsf{write}_{\mathsf{col}}}$; $\mathsf{read}_{\mathsf{col}}(r'_N) \stackrel{?}{=} v_{\mathsf{read}_{\mathsf{col}}}$; and $\mathsf{final}_{\mathsf{col}}(r'_M) \stackrel{?}{=} v_{\mathsf{final}_{\mathsf{col}}}$
11. $\mathcal{V}$: directly check if the first equality holds, which can be done with $O(\log M)$ field operations; check the remaining equations hold with an oracle query to each of $E_{\mathsf{ry}}, \mathsf{col}, \mathsf{write}_{\mathsf{col}}, \mathsf{read}_{\mathsf{col}}, \mathsf{final}_{\mathsf{col}}$.

Completeness. Perfect completeness follows from perfect completeness of the sum-check protocol and the fact that the multiset equality checks using their fingerprints hold with probability 1 over the choice of τ, γ if the prover is honest.

Soundness. Applying a standard union bound to the soundness error introduced by probabilistic multiset equality checks with the soundness error of the sum-check protocol [LFKN90], we conclude that the soundness error for the depicted polynomial IOP as at most $O(N)/|\mathbb{F}|$.

Round and communication complexity. There are three sum-check reductions. First, it is applied on a polynomial with $\log N$ variables where the degree is at most 3 in each variable, so the round complexity is $O(\log N)$ and the communication cost is $O(\log N)$ field elements.

Second, it is applied to compute four "grand products" in parallel. Two of the grand products are over vectors of size M and the remaining two are over vectors of size N. We use the interactive proof for grand products of [Tha13], for which the round complexity is $O(\log^2 N)$ with a communication cost of $O(\log^2 N)$ field elements.

Third, the depicted IOP runs four additional "grand products", which incurs the same costs as above.

In total, the round complexity of the depicted IOP is $O(\log^2 N)$ and the communication cost is $O(\log^2 N)$ field elements.[15]

[15] Employing the sum-check reduction for grand products in [SL20] results in a complexity of $O(\log N)$ with a communication cost of $O(\log N)$ field elements and a verifier runtime of $O(\log N)$.

Verifier time. The verifier's runtime is dominated by its runtime in the grand product sum-check reductions, which is $O(\log^2 N)$.

Prover Time. Using linear-time sum-checks [Tha13, XZZ$^+$19] in all three sum-check reductions (and exploiting the linear-time prover in the grand product interactive proof [Tha13]), the prover's time is $O(N)$ finite field operations.

Finally, to prove Theorem 5, applying the compiler of [BFS20] to the depicted polynomial IOP with the given polynomial commitment primitive, followed by the Fiat-Shamir transformation [FS86], provides the desired non-interactive argument of knowledge for proving evaluations of committed sparse multilinear polynomials, with efficiency claimed in the theorem statement.

6 Linear-time SNARKs for R1CS from polynomial commitments

This section describes our first route to construct linear-time SNARKs for R1CS (§1). The following theorem captures our main result.

Theorem 6. *Assuming that $|\mathbb{F}| = 2^{\Theta(\lambda)}$ there exists a preprocessing SNARK for $\mathcal{R}_{R1CS}$ in the random oracle model, with the following efficiency characteristics, where M denotes the dimensions of the R1CS matrices, N denotes the number of non-zero entries, and a fixed positive integer t:*

- *the preprocessing cost to the verifier is $O(N)$ $\mathbb{F}$-ops;*
- *the running time of the prover is $O(N)$ $\mathbb{F}$-ops;*
- *the running time of the verifier is $O_\lambda(N^{1/t})$ $\mathbb{F}$-ops; and*
- *the proof size is $O_\lambda(N^{1/t})$.*

Proof. From applying [BFS20, Theorem 8] to the polynomial IOP in Theorem 1 using polynomial commitment schemes from Theorem 4 and Corollary 1, there exists a public-coin interactive argument for $\mathcal{R}_{\text{R1CS}}$ with witness-extended emulation. Applying the Fiat-Shamir transform [FS86] to the public-coin interactive argument results in the claimed SNARK for $\mathcal{R}_{\text{R1CS}}$.

The verifier, in a preprocessing step, commits to three $2\log M$-variate polynomials that evaluate to a non-zero value at at most N locations over the Boolean hypercube $\{0,1\}^{2\log M}$; this costs $O(N)$ $\mathbb{F}$-ops.

The prover: (1) commits to a $O(\log M)$-variate polynomial (which costs $O(M)$ $\mathbb{F}$-ops); (2) participates in the sum-check protocol in the polynomial IOP in Theorem 1 (which costs $O(N)$ $\mathbb{F}$-ops); and (3) proves evaluations of one $(\log M - 1)$-variate multilinear polynomial and three $2\log M$-variate multilinear polynomials from the preprocessing step (which costs $O(N)$ $\mathbb{F}$-ops). Together, the prover incurs $O(N)$ $\mathbb{F}$-ops.

The verifier to verify a proof: (1) participates in the sum-check protocol in the polynomial IOP in Theorem 1 (which costs $O(\log M)$ $\mathbb{F}$-ops); and (2) verifies the proofs of evaluations of one $(\log M - 1)$-variate multilinear polynomial and three $2\log M$-variate multilinear polynomials from the preprocessing step (which costs $O_\lambda(N^{1/t})$ $\mathbb{F}$-ops). Together, the verifier incurs $O_\lambda(N^{1/t})$ $\mathbb{F}$-ops.

Finally, the proof size is the sum of the proof sizes from the sum-check protocol in the polynomial IOP from Theorem 1 and the proof sizes from polynomial commitment schemes. In total, the proof size is $O(\log M) + O_\lambda(N^{1/t}) = O_\lambda(N^{1/t})$. □

We obtain zkSNARKs with the cost profiles and cryptographic assumptions asserted in the final three rows of Figure 1 by composing the SNARK of Theorem 6 with known zkSNARKs [Set20, SL20, Gro16]. Specifically, the prover in the composed SNARKs proves that it knows a proof π that would convince the SNARK verifier in Theorem 6 to accept. Perfect zero-knowledge of the resulting composed SNARK is immediate from the zero-knowledge property of the SNARKs from these prior works [Set20, SL20, Gro16]. Perfect completeness follows from the perfect completeness properties of these prior works and of Theorem 6. Knowledge soundness

follows from a standard argument [Val08, BCCT13]: one composes the knowledge extractors of the two constituent SNARKs to get a knowledge extractor for the composed SNARK.

7 Linear-time SNARKs for R1CS from compiling tensor queries

This section describes our second route to construct linear-time SNARKs for R1CS (§1). Below, we rely on the definition of tensor IOPs and how to represent polynomial evaluations with tensor queries from Section 4.

Polynomial IOPs are Tensor IOPs. For the moment, as we did in Theorem 1, let us assume that the verifier has query access to the polynomials $\widetilde{A}$, $\widetilde{B}$, and $\widetilde{C}$ capturing an R1CS instance $(\mathbb{F}, A, B, C, M, N, \mathsf{io})$. Under this assumption, Theorem 1 gave a linear-time polynomial IOP in which the prover begins by sending a multilinear, $(\log M - 1)$-variate polynomial $\widetilde{W}$, and then the prover and verifier apply the sum-check protocol interactive proof to $O(1)$ many polynomials. Across all invocations of the sum-check protocol, the verifier runs in $O(\log M)$ time and queries $\widetilde{W}$ at a single point.

Equation (3) implies that this polynomial IOP can be transformed, in a straightforward manner, into a $(\mathbb{F}, k, t)$-tensor IOP for any constant $t > 0$. Specifically, the honest prover's first message Π_1 in the tensor IOP is simply the coefficient vector of $\widetilde{W}$ over the Lagrange basis, which in turn is simply the witness vector $W \in \mathbb{F}^{M/2}$ (the prover sets m_1 to the empty string). When the verifier needs to learn the evaluation of $\widetilde{W}$ at a point $r \in \mathbb{F}^{\log M - 1}$, Equation (3) shows that the evaluation can be obtained with a single tensor query $(q_1, \ldots q_t)$ to Π_1. Moreover, in terms of time complexity, the verifier can compute the set of vectors $(q_1, \ldots q_t)$, in $O(N^{1/t})$ operations over $\mathbb{F}$.

The following theorem captures the tensor IOP obtained from a straightforward transformation of the polynomial IOP in Theorem 1.

Theorem 7. *For any finite field $\mathbb{F}$ and positive integers k, t, there exists a $(\mathbb{F}, k, t)$-tensor IOP for $\mathcal{R}_{R1CS}$ with the following parameters, where $N = 2^\ell = k^t$.*

- *completeness error is 0;*
- *soundness error is $O(\log(M))/|\mathbb{F}|$;*
- *round complexity is $O(\log M)$;*
- *query complexity is $O(1)$;*
- *the proof length is $O(M)$ elements of $\mathbb{F}$;*
- *the prover performs $O(N)$ operations over $\mathbb{F}$; and*
- *the verifier performs $O(N^{1/t})$ operations over $\mathbb{F}$, given a tensor-query access to the R1CS matrices.*

Section 5 provides a polynomial IOP for proving evaluations of sparse multilinear polynomials, where the verifier is given an oracle access to a dense representation of the sparse multilinear polynomial. From the aforementioned discussion, this polynomial IOP can be viewed as a holographic tensor IOP for proving evaluations of sparse multilinear polynomials. Using this tensor IOP to implement the tensor-query access to the R1CS matrices results in the following holographic tensor IOP:

Theorem 8. *For any finite field $\mathbb{F}$ and positive integers k, t, there exists a $(\mathbb{F}, k, t)$-holographic-tensor IOP for $\mathcal{R}_{R1CS}$ with the following parameters, where $N = 2^\ell = k^t$.*

- *completeness error is 0;*
- *soundness error is $O(N)/|\mathbb{F}|$;*
- *round complexity is $O(\log^2 N)$;*
- *query complexity is $O(1)$;*
- *the index length is $O(N)$ elements of $\mathbb{F}$;*

- *the indexer performs $O(N)$ operations over $\mathbb{F}$;*
- *the proof length is $O(N)$ elements of $\mathbb{F}$;*
- *the prover performs $O(N)$ operations over $\mathbb{F}$; and*
- *the verifier performs $O(N^{1/t})$ operations over $\mathbb{F}$.*

Bootle, Chiesa, and Groth [BCG20a] show how to transform any tensor IOP such as the above into a point-query IOP while preserving holography, which is in turn known to be transformable into preprocessing SNARKs. Both transformations preserve a linear-time prover. Specifically, applying these transformations to the tensor IOP of Theorem 8 yields the following two results.

Theorem 9. *For security parameter λ, given an $(\mathbb{F}, k, t)$-tensor IOP for $\mathcal{R}_{R1CS}$ with $N = k^t$ and fixed value of t, and a linear code over $\mathbb{F}$ with rate $\rho = k/n$, relative distance $\delta = d/n$, and encoding time $O(k)$, there exists a point-query IOP for $\mathcal{R}_{R1CS}$ with the following parameters:*

- *soundness error is $O(N/|\mathbb{F}| + d^t/|\mathbb{F}| + (1 - \frac{\delta^t}{2})^\lambda)$;*
- *round complexity is $O(\log^2 N)$;*
- *query complexity is $O(N^{1/t})$;*
- *the index length is $O(N)$ elements of $\mathbb{F}$;*
- *the indexer performs $O(N)$ operations over $\mathbb{F}$;*
- *the proof length is $O(N)$ elements of $\mathbb{F}$;*
- *the prover performs $O(N)$ operations over $\mathbb{F}$;*
- *the verifier performs $O(N^{1/t})$ operations over $\mathbb{F}$.*

Proof. Applying [BCG20a, Theorem 3] to the tensor IOP from Theorem 8 provides the desired result. □

Theorem 10. *For security parameter λ and a positive integer t, given a hash function that can compute a Merkle-hash of N elements of $\mathbb{F}$ with the same time complexity as $O(N)$ $\mathbb{F}$-ops, there exists a preprocessing SNARK in the random oracle model with the following efficiency properties.*

- *the running time of the prover is $O(N)$ operations over $\mathbb{F}$;*
- *the preprocessing time for the verifier is $O(N)$ operations over $\mathbb{F}$;*
- *after pre-processing, the running time of the verifier is $O_\lambda(N^{1/t})$ operations over $\mathbb{F}$; and*
- *the proof size is $O_\lambda(N^{1/t})$.*

Proof. It is known that, over any finite field, there are linear-time encodable linear codes with constant rate and relative distance [Spi96, DI14]. The theorem then follows by combining the resulting point-query IOP of Theorem 9 with the BCS transform [BCS16] from point-query IOPs to SNARKs in the random oracle model (with a linear-time hash function [AHI+17, BCG+17]). □

8 Implementation and evaluation

Implementation. We implement the polynomial commitment scheme from Section 4 with $t = 2$ as a Rust library in about 1,000 lines of code. For a linear code, we use Reed-Solomon codes with rate $\rho = 1/4$. This does *not* achieve a linear-time prover since the encoding procedure of Reed-Solomon codes requires an FFT, which is a super-linear operation (§1). We refer to this scheme as *Ligero-PC*, since as discussed earlier, the polynomial commitment scheme is arguably implicit in Ligero [AHIV17]. We set parameters in Ligero-PC using the analysis in Appendix B. Unless we specify otherwise, our choice of parameters provide at least 128 bits of security. Our implementation is generic over the hash function, but in our experiments, we use `blake3`.

We integrate Ligero-PC with the open-source implementation of Spartan [liba] to obtain a SNARK library for R1CS. We refer to this variant of Spartan as *Cerberus*.

As discussed in Section 1.1, our SNARK implementation is not currently zero-knowledge. However, it can be rendered zero-knowledge using standard techniques with minimal overhead [AHIV17, CFS17, XZZ+19]. We leave the completion of a zero-knowledge implementation to near-term future work.

Metrics, method, and baselines. As is standard in the SNARKs literature, our metrics are: (1) the prover time to produce a proof; (2) the verifier time to verify a proof; (3) proof sizes; and (4) the verifier's preprocessing costs. As baselines, we consider two types of SNARKs: (1) schemes that achieve verification costs sub-linear in the size of the statement (which implies sub-linear proof sizes); and (2) schemes that only achieve sub-linear proof sizes. We refer to the latter type of schemes as $\frac{1}{2}$-*SNARKs*. Additionally, we focus on schemes that do not require a trusted setup (we refer the reader to Spartan [Set20] for a comparison between our baselines with state-of-the-art SNARKs with trusted setup).

The reader may wonder why we include results for our $\frac{1}{2}$-SNARK given that our implementation is not yet zero-knowledge; after all, the verifier runtime in any non-zero-knowledge $\frac{1}{2}$-SNARK is commensurate with that of the trivial proof system in which the prover explicitly sends the NP-witness to the verifier. The answer is three-fold. First, our $\frac{1}{2}$-SNARK actually *can* save the verifier time relative to the trivial proof system for structured R1CS instances. In particular, if the R1CS is data parallel, then the verifier can run in time proportional to the size of a single sub-computation, independent of the number of times the sub-computation is performed (this is entirely analogous to how prior proof systems save the verifier work for structured computations [Tha13, BBHR19]). Second, since our $\frac{1}{2}$-SNARK can be rendered zero-knowledge with minimal overhead, we expect that the reported performance results are indicative of the performance of a future zero-knowledge implementation. Third, the proof length in our $\frac{1}{2}$-SNARK is smaller than the witness size for sufficiently large instances ($N \geq 2^{13}$).

Unless we specify otherwise, we run our experiments on an Azure Standard F16s_v2 virtual machine (16 vCPUs, 32 GB memory) with Ubuntu 20.10. We report results from a single-threaded configuration since not all our baselines leverage multiple cores. As with prior work [BCR+19, Set20, COS20, SL20], we vary the size of the R1CS instance by varying the number of constraints and variables m and maintain the ratio n/m to approximately 1.

8.1 Performance of Cerberus's $\frac{1}{2}$-SNARK scheme

Prior state-of-the-art schemes in this category include: Ligero [AHIV17], Bulletproofs [BBB+18], Aurora [BCR+19], SpartanNIZK [Set20], and Lakonia [SL20]. Note that for uniform computations (e.g., data-parallel circuits), Hyrax [WTS+18] and STARK [BBHR19] are SNARKs, but for computations without any structure, they are $\frac{1}{2}$-SNARKs. We do not report results from Bulletproofs or STARK as they feature a more expensive prover than other baselines considered here [BBB+18, BCR+19]. Hyrax [WTS+18] supports only layered arithmetic circuits, so as used in prior work [Set20] for comparison purposes, we translate R1CS to depth-1 arithmetic circuits (without any structure). None of the $\frac{1}{2}$-SNARKs we consider require a preprocessing step for the verifier.

Below, we provide comparison with Wolverine [WYKW20] and Mac'n'Cheese [BMRS20], which unlike schemes considered here do *not* support proof sizes sub-linear in the instance size. Another potential baseline is Virgo [ZXZS20], which like Hyrax [WTS+18] applies only to low-depth circuits as they both share the same information-theoretic component [GKR08, CMT12, WJB+17, XZZ+19]. In Section 8.3, we provide a brief qualitative comparison between Cerberus's polynomial commitment scheme and Virgo's polynomial commitment scheme.

For Aurora and Ligero, we use their open-source implementations from `libiop` [libc], configured to provide provable security. For Hyrax, we use its reference (i.e., unoptimized) implementation [libb]. For SpartanNIZK, we use its open-source implementation [liba]. For all schemes we use 256-bit prime fields.

We first experiment with Cerberus and its baselines with varying R1CS instance sizes up to 2^{20} constraints defined over a 256-bit prime field. Figures 3, 4, and 5 depict respectively the prover time, the proof sizes, and

the verifier time from these experiments. We find the following from these experiments.

- Cerberus's prover is the fastest at all instance sizes we measure—though we expect SpartanNIZK (whose bottleneck is a multiexponentiation in a cryptographic group) to "cross over" and become more efficient than Cerberus (whose bottleneck is FFTs) at larger instances sizes than we experiment with. However, compared to baseslines that are plausibly post-quantum secure (Ligero and Aurora), Cerberus's prover is over an order of magnitude faster.
- Cerberus's verifier is competitive with that of SpartanNIZK and Lakonia, and is well over an order of magnitude faster than the other plausibly post-quantum 1/2-SNARK baselines (Ligero and Aurora).
- Cerberus's proof size is larger than all other tested systems other than Ligero. Still, our proofs are substantially smaller than the size of the NP-witness for instance sizes $N \geq 2^{13}$.

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Ligero [AHIV17]	0.1	0.2	0.4	0.8	1.6	2	4	8	17	35	69
Hyrax [WTS+18]	1	1.7	2.8	5	9	18	36	61	117	244	486
Aurora [BCR+19]	0.5	0.8	1.6	3.2	6.5	13.3	27	56	116	236	485
SpartanNIZK [Set20]	0.01	0.02	0.04	0.07	0.14	0.25	0.5	0.8	1.7	3	6
Lakonia [SL20]	0.2	0.2	0.4	0.5	0.8	1	2	3	6	10	19
Cerberus ($\frac{1}{2}$-SNARK)	0.004	0.008	0.02	0.03	0.07	0.1	0.3	0.5	1	2.2	4.4

Figure 3: Prover time (in seconds) for varying R1CS instance sizes under different schemes.

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Ligero [AHIV17]	546	628	1,076	1,169	2,100	3,169	5,788	5,662	10,527	10,736	19,828
Hyrax [WTS+18]	14	16	17	20	21	26	28	37	38	56	58
Aurora [BCR+19]	447	510	610	717	810	931	1,069	1,179	1,315	1,473	1,603
SpartanNIZK [Set20]	6	6	7	8	10	10	15	15	23	24	40
Lakonia [SL20]	7	7	8	8	9	9	10	10	11	11	11
Cerberus ($\frac{1}{2}$-SNARK)	129	217	425	441	834	870	1,655	1,703	3,271	3,360	6,484

Figure 4: Proof sizes (KBs) for Cerberus and its baselines.

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Ligero [AHIV17]	49	96	172	357	680	976	1,900	3,700	7,300	15,000	31,000
Hyrax [WTS+18]	195	229	262	317	388	502	510	1,200	1,900	3,500	7,700
Aurora [BCR+19]	186	316	574	933	1,800	3,500	6,700	13,000	27,000	54,000	108,000
SpartanNIZK [Set20]	3	4	5	6	9	15	25	44	87	166	347
Lakonia [SL20]	27	29	34	38	48	60	85	115	179	281	517
Cerberus ($\frac{1}{2}$-SNARK)	2	2	3	4	11	18	35	61	121	227	471

Figure 5: Verifier time (in ms) under different schemes.

Performance for larger instance sizes. To demonstrate Cerberus's scalability to larger instance sizes, we experiment with Cerberus and SpartanNIZK for instance sizes beyond 2^{20} constraints.

For these larger-scale experiments, we use an Azure Standard F32s_v2 VM which has 32 vCPUs and 64 GB memory. Figures 6, 7, and 8 depict results from these larger-scale experiments. Our findings from these experiments are similar to results from the smaller-scale results.

Performance over small fields. To demonstrate flexibility with different field sizes, we also run Cerberus with a prime field where the prime modulus is 128 bits. For the latter case, our choice of parameters achieve at least 100 bits security. We depict these results together with results from our larger-scale experiments (Figures 6, 7, and 8).

	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}	2^{26}
SpartanNIZK (256-bit field)	6	12	23	44	88	171	347
Cerberus ($\frac{1}{2}$-SNARK)(256-bit field)	4.3	9	18	36	73	150	300
Cerberus ($\frac{1}{2}$-SNARK)(128-bit field)	2	4	8	16	33	67	137

Figure 6: Prover time (in seconds) for varying R1CS instance sizes under Cerberus ($\frac{1}{2}$-SNARK) and its baselines.

	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}	2^{26}
SpartanNIZK (256-bit field)	40	41	74	74	140	140	272
Cerberus ($\frac{1}{2}$-SNARK)(256-bit field)	6,484	6,655	12,882	13,216	25,670	26,332	51,240
Cerberus ($\frac{1}{2}$-SNARK)(128-bit field)	2,802	2,932	5,540	5,793	10,995	11,494	21,894

Figure 7: Proof sizes (in KBs) under Cerberus and its baselines.

	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}	2^{26}
SpartanNIZK (256-bit field)	0.3	0.7	1.3	2.6	5	10	20
Cerberus ($\frac{1}{2}$-SNARK)(256-bit field)	0.5	0.9	1.8	3.6	7.3	14.4	29
Cerberus ($\frac{1}{2}$-SNARK)(128-bit field)	0.2	0.3	0.6	1.2	2.4	4.7	9.4

Figure 8: Verifier time (in seconds) for varying R1CS instance sizes under different schemes.

Recall that our asymptotic results require $|\mathbb{F}| > \exp(\Omega(\lambda))$ to achieve a linear-time prover, because if the field is smaller than this, certain parts of the protocol need to be repeated $\omega(1)$ times to drive the soundness error below $\exp(-\lambda)$.[16] However, our implementation (which is not asymptotically linear-time since it uses Reed-Solomon codes) is quite efficient over small fields. The reason is that only some parts of the protocol need to be repeated to drive the soundness error below $\exp(-\lambda)$ and those repetitions produce only low-order effects on the prover's runtime and the proof length. This means that for a fixed security level, our prover is faster over small fields than large fields, because the effect of faster field arithmetic dominates the overhead due to the need to repeat parts of the protocol to drive down soundness error. Similar observations appear in prior work [BBHR19]. See Appendix B for details.

Comparison with Wolverine and Mac'n'Cheese. There does not appear to be open-source implementations of these baselines, so we rely on prior performance reports for this comparison. Since we do not measure the performance of all schemes on the same hardware platform, one must treat this as a rough comparison, distinct from the more rigorous comparison to other systems earlier in this section.

We begin with a qualitative comparison.

- Both baselines require interaction between the verifier and the prover, so unlike Cerberus, they do not produce proofs that any verifier can verify. In other words, both baselines do not produce publicly-verifiable proofs. The verifier's runtime is asymptotically the same as Cerberus's verifier ($\frac{1}{2}$-SNARK variant), but Cerberus's verifier is concretely far cheaper as noted below.
- Proof sizes are $O_\lambda(N)$ for an N-sized instance for both baselines, whereas Cerberus's proofs are $O_\lambda(\sqrt{N})$.
- Asymptotically, the prover in both baselines uses memory proportional to that needed to produce the witness and evaluate the circuit non-cryptographically. Cerberus does not have this property, though for worst-case circuits, Cerberus and the baselines have the same asymptotic memory consumption (this includes, for example, circuit-satisfiability instances whose witness size is linear in the circuit size). Concretely, both baselines report better memory usage than Cerberus.

More concretely, for a circuit with 2^{27} (multiplication) gates over a prime field with $p = 2^{127} - 1$, Mac'n'Cheese reports that the prover time is about 650 seconds to produce a proof of size 6 GB, and the verifier's run

[16] To achieve *constant* soundness error, we in fact only need $|\mathbb{F}| \gg N$ (see Claim 6 in Appendix B).

time is 650 seconds. On a generic 128-bit field, Cerberus's prover runs in 278 seconds to produce a proof of size 22.9 MB, and the verifier's run time is 19 seconds. For the same sized computation over $p = 2^{61} - 1$, Wolverine's prover time is 320 seconds, the proof size is 4.2 GB, and the verifier's run time is 320 seconds. Note that Wolverine reports results at 40 bits of security whereas Cerberus achieves over 100 bit of security.

8.2 Performance of Cerberus's SNARK scheme

Prior state-of-the-art schemes in this category include: Spartan [Set20], SuperSonic [BFS20], Fractal [COS20], Kopis [SL20], and Xiphos [SL20]. For Fractal, we use its open-source implementations from `libiop` [libc], configured to provide provable security. For SuperSonic, there is no prior implementation, so we use prior estimates of their costs based on microbenchmarks (See [SL20] for a detailed discussion of how they estimate these costs). For Spartan, we use its open-source implementation [liba]. For preprocessing costs, we ignore the use of "untrusted assistant" technique [SL20], which applies to all schemes considered here.

Figures 9, 10, 11, 12 depict respectively the prover time, the proof size, the verifier time, and the verifier's preprocessing time for varying R1CS instance sizes for Cerberus and its baselines. We find the following from our experimental results.

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Spartan [Set20]	0.1	0.2	0.3	0.5	0.9	2	3	7	12	24	45
SuperSonic [BFS20]	86	163	311	599	1,160	2,240	4,360	8,500	16,600	32,500	63,800
Fractal [COS20]	0.8	1.5	2.9	5.9	12	25	51	104	216	–	–
Kopis [SL20]	1.0	1.3	2.1	3.1	5.3	8.3	15	25	48	87	168
Xiphos [SL20]	1.2	2.0	2.5	4.2	5.7	10.2	15.4	28	49	93	169
Cerberus	0.05	0.09	0.2	0.4	0.8	1.5	3	6	13	24	50

Figure 9: Prover time (in seconds) for varying R1CS instance sizes under different schemes. Fractal's prover runs out of memory at 2^{18} constraints and beyond.

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Spartan [Set20]	32	37	41.7	48	54	63	72	85	98	120	142
SuperSonic [BFS20]	31	33	34	36	38	40	41	43	45	47	49
Fractal [COS20]	1,069	1,226	1,359	1,490	1,665	1,817	1,962	2,147	2,316	–	–
Kopis [SL20]	25	26	27	29	30	32	33	34	36	37	39
Xiphos [SL20]	40	44	45	48	49	51	53	55	57	59	61
Cerberus	705	1,026	1,441	1,947	2,735	3,681	5,270	7,047	10,205	13,764	20,828

Figure 10: Proof sizes in KBs.

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Spartan [Set20]	8	9	11	14	18	22	30	38	53	68	97
SuperSonic [BFS20]	1,400	1,500	1,600	1,700	1,900	2,000	2,100	2,200	2,300	2,500	2,600
Fractal [COS20]	148	120	163	168	141	184	188	165	205	–	–
Kopis [SL20]	68	73	87	94	117	129	165	185	236	278	390
Xiphos [SL20]	53	54	55	57	57	60	60	63	63	65	65
Cerberus	11	14	20	25	37	47	71	92	141	181	280

Figure 11: Verifier's performance (in ms).

We have the largest proof sizes amongst the displayed proof systems, but for large enough R1CS instances our proof sizes (larger than 2^{19}) are sublinear in the size of the NP-witness. Compared to the other plausibly post-quantum full SNARK tested (Fractal), our prover is more than an order of magnitude more efficient. Compared to discrete-logarithm-based systems, our prover is faster on small instance sizes (less than 2^{20}), but scales worse with the instance size N. This is consistent with the prover's bottleneck in the discrete-log-based systems being a multiexponentiation of size N, which costs $O(N\lambda/\log(\lambda N))$ group operations; for small

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Spartan	0.1	0.1	0.2	0.3	0.6	1	2	3	7	10	20
SuperSonic	35	64	117	216	400	747	1,400	2,600	4,900	9,400	17,900
Fractal	0.3	0.6	1.2	2.5	5.4	11.5	24	51	107	227	–
Kopis	0.6	0.7	1.3	1.5	2.7	3.4	6.2	8.6	16	24	46
Xiphos	0.8	1	2	3	3	6	7	13	18	32	49
Cerberus	0.03	0.07	0.14	0.3	0.6	1.2	2.4	5	10	21	43

Figure 12: Verifier's preprocessing time (i.e., encoder's time) in seconds for varying R1CS instance sizes under different schemes.

values of N this is slower than the $O(N \log N)$ runtime of *Ligero-PC*, but for large values of N it is faster. The comparison for Encoder time amongst the systems is similar to the prover time.

The verifier in our SNARK is faster than the other plausibly post-quantum protocol tested, Fractal, despite having larger proofs. Compared to discrete-logarithm-based systems, our new SNARK is broadly competitive with the exception of Xiphos, which is specifically designed to have a fast verifier (logarithmically many group operations).

8.3 Other benefits of Cerberus

STARK [BBHR19], Aurora [BCR+19], Virgo [ZXZS20], and Fractal [COS20] employ polynomial commitments based on FRI [BBHR18],[17] and they require the prover to perform an FFT of length $\Theta(N)$ for an NP instance of size N.[18] Besides being the bottleneck for the prover for large values of N, FFTs are notoriously space-intensive and challenging to parallelize and distribute. In contrast, Cerberus's prover performs $\Theta(\sqrt{N})$ independent FFTs each of length $\Theta(\sqrt{N})$. While this results in the same asymptotic runtime as a single FFT of length N, it is trivial to parallelize across as many as $\Theta(\sqrt{N})$ threads. This benefit is not evident in our reported results, which are all single-threaded. We leave it to the near-term future work to report results from a parallel version of Cerberus.

Relative to schemes that employ polynomial commitments based on cryptographic groups (Spartan, SuperSonic, Kopis, Lakonia, and Xiphos), Cerberus has the following two advantages. First, it is plausibly post-quantum secure. Second, it applies over any field (for efficiency, the field must support FFTs). In contrast, group-based arguments cannot support R1CS instances over non-prime-order fields or fields smaller than $\exp(\lambda)$ (though recent theoretical work seeks to apply similar techniques in more general settings [BCS21]).

Acknowledgments

We thank Jonathan Bootle, Alessandro Chiesa, and Jens Groth for answering our questions on [BCG20a], and Eran Tromer for helpful discussions on recursive composition. Justin Thaler was supported in part by NSF CAREER award CCF-1845125. Both Justin Thaler and Riad Wahby were supported in part by DARPA under Agreement No. HR00112020022. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the United States Government or DARPA.

[17] These works, with the exception of Virgo [ZXZS20], do not treat their use of FRI in combination with Merkle trees as a polynomial commitment scheme, but for ease of exposition, we ignore this aspect.

[18] Although Virgo [ZXZS20] applies a polynomial commitment scheme only over the (non-deterministic) input to the circuit, in the worst case and for realistic circuits, the non-deterministic input size is similar to the circuit size.

References

[AHI+17] Benny Applebaum, Naama Haramaty, Yuval Ishai, Eyal Kushilevitz, and Vinod Vaikuntanathan. Low-complexity cryptographic hash functions. In *Proceedings of the Innovations in Theoretical Computer Science (ITCS)*, 2017.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthu Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2017.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2018.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast Reed-Solomon interactive oracle proofs of proximity. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, 2018.

[BBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2019.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the Innovations in Theoretical Computer Science (ITCS)*, 2012.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKs and proof-carrying data. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2013.

[BCG+17] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, 2017.

[BCG20a] Jonathan Bootle, Alessandro Chiesa, and Jens Groth. Linear-time arguments with sub-linear verification from tensor codes. In *Theory of Cryptography Conference (TCC)*, 2020.

[BCG+20b] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2020.

[BCGT13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. Fast reductions from RAMs to delegatable succinct constraint satisfaction problems: Extended abstract. In *Proceedings of the Innovations in Theoretical Computer Science (ITCS)*, 2013.

[BCI+20] Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. Proximity gaps for Reed-Solomon codes. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 2020.

[BCL20] Jonathan Bootle, Alessandro Chiesa, and Siqi Liu. Zero-knowledge succinct arguments with a linear-time prover. ePrint Report 2020/1527, 2020.

[BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data from accumulation schemes. In *Theory of Cryptography Conference (TCC)*, 2020.

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2019.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive Oracle Proofs. In *Theory of Cryptography Conference (TCC)*, 2016.

[BCS21] Jonathan Bootle, Alessandro Chiesa, and Katerina Sotiraki. Sumcheck arguments and their applications. Cryptology ePrint Archive, Report 2021/333, 2021.

[BDFG20] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo Infinite: Recursive zk-SNARKs from any Additive Polynomial Commitment Scheme. Cryptology ePrint Archive, Report 2020/1536, 2020.

[BEG+91] Manuel Blum, Will Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 1991.

[BFR+13] Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. Verifying computations with state. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2013.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2020.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019.

[BMRS20] 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.

[CCH+] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*.

[CDG+17] 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 *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2017.

[CFH+15] Craig Costello, Cedric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. Geppetto: Versatile verifiable computation. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, May 2015.

[CFS17] Alessandro Chiesa, Michael A. Forbes, and Nicholas Spooner. A zero knowledge sumcheck and its applications. *CoRR*, abs/1704.02086, 2017.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2020.

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the Innovations in Theoretical Computer Science (ITCS)*, 2012.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2020.

[DI14] Erez Druk and Yuval Ishai. Linear-time encodable codes meeting the Gilbert-Varshamov bound and their cryptographic applications. In *Proceedings of the Innovations in Theoretical Computer Science (ITCS)*, pages 169–182, 2014.

[DIO20] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. Cryptology ePrint Archive, Report 2020/1446, 2020.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Proceedings of the International Cryptology Conference (CRYPTO)*, pages 186–194, 1986.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2013.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2008.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster zero-knowledge for Boolean circuits. In *Proceedings of the USENIX Security Symposium*, 2016.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, 2010.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2016.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 99–108, 2011.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. ePrint Report 2019/953, 2019.

[IKO07] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient arguments without short PCPs. In *IEEE Conference on Computational Complexity*, 2007.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1992.

[KMP20] Abhiram Kothapalli, Elisaweta Masserova, and Bryan Parno. A direct construction for asymptotically optimal zkSNARKs. Cryptology ePrint Archive, Report 2020/1318, 2020.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 177–194, 2010.

[Lee20] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. Cryptology ePrint Archive, Report 2020/1274, 2020.

[LFKN90] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, October 1990.

[liba] Spartan: High-speed zkSNARKs without trusted setup. `https://github.com/Microsoft/Spartan`.

[libb] libfennel. Hyrax reference implementation. `https://github.com/hyraxZK/fennel`.

[libc] libiop. A C++ library for IOP-based zkSNARK. `https://github.com/scipr-lab/libiop`.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *Theory of Cryptography Conference (TCC)*, 2012.

[LNS20] Jonathan Lee, Kirill Nikitin, and Srinath Setty. Replicated state machines without replicated execution. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2020.

[Mic94] Silvio Micali. CS proofs. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 1994.

[PGHR13] Bryan Parno, Craig Gentry, Jon Howell, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, May 2013.

[RR20] Noga Ron-Zewi and Ron Rothblum. Local proofs approaching the witness length. In *Foundations of Computer Science (FOCS)*, 2020.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 49–62, 2016.

[SAGL18a] Srinath Setty, Sebastian Angel, Trinabh Gupta, and Jonathan Lee. Proving the correct execution of concurrent services in zero-knowledge. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, October 2018.

[SAGL18b] Srinath Setty, Sebastian Angel, Trinabh Gupta, and Jonathan Lee. Proving the correct execution of concurrent services in zero-knowledge (extended version). ePrint Report 2018/907, September 2018.

[SBV+13] Srinath Setty, Benjamin Braun, Victor Vu, Andrew J. Blumberg, Bryan Parno, and Michael Walfish. Resolving the conflict between generality and plausibility in verified computation. In *Proceedings of the ACM European Conference on Computer Systems (EuroSys)*, April 2013.

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2020.

[SL20] Srinath Setty and Jonathan Lee. Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Report 2020/1275, 2020.

[SMBW12] Srinath Setty, Richard McPherson, Andrew J. Blumberg, and Michael Walfish. Making argument systems for outsourced computation practical (sometimes). In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, February 2012.

[Spi96] Daniel A Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1731, 1996.

[SVP+12] Srinath Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J. Blumberg, and Michael Walfish. Taking proof-based verified computation a few steps closer to practicality. In *Proceedings of the USENIX Security Symposium*, August 2012.

[Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2013.

[Tha20] Justin Thaler. Proofs, arguments, and zero-knowledge. `http://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.html`, 2020.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Theory of Cryptography Conference (TCC)*, pages 1–18, 2008.

[WJB+17] Riad S. Wahby, Ye Ji, Andrew J. Blumberg, Abhi Shelat, Justin Thaler, Michael Walfish, and Thomas Wies. Full accounting for verifiable outsourcing. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2017.

[WSR+15] Riad S. Wahby, Srinath Setty, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. Efficient RAM and control flow in verifiable outsourced computation. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2015.

[WTS+18] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2018.

[WYKW20] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for Boolean and arithmetic circuits. Cryptology ePrint Archive, Report 2020/925, 2020.

[XZZ+19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2019.

[ZGK+17] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2017.

[ZXZS20] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2020.

A Description of the polynomial commitment scheme for $t = 2$

Notation. g is a multilinear polynomial with n coefficients. We assume for simplicity that $n = m^2$ for some integer m. Let u denote the coefficient vector of g in the Lagrange basis (equivalently, u is the vector of all evaluations of g over inputs in $\{0,1\}^{\log n}$). Recalling that $[m] = \{1, \ldots, m\}$, we can naturally index entries of u by elements of the set $[m]^2$. As per Section 4, for any input r to g there exist vectors $q_1, q_2 \in \mathbb{F}^m$ such that

$$g(r) = \langle (q_1 \otimes q_2), u \rangle.$$

For each $i \in [m]$, let us view u as an $m \times m$ matrix, and let u_i denote the ith row of this matrix, i.e., $u_i = \{u_{i,j}\}_{j \in [m]}$.

Let $N = \rho^{-1} \cdot m$, and let $\mathsf{Enc} \colon \mathbb{F}^m \to \mathbb{F}^N$ denote the encoding function of a linear code with constant rate $\rho > 0$ and constant relative distance $\gamma > 0$. We assume that Enc runs in time proportional to that required to perform $O(N)$ operations over $\mathbb{F}$. We assume for simplicity that Enc is systematic, since explicit systematic codes with the properties we require are known [Spi96].

Commitment phase. Let $\hat{u} = \{\mathsf{Enc}(u_i)\}_{i \in [m]} \in \left(\mathbb{F}^N\right)^m$ denote the vector obtained by encoding each row of u. In the IOP setting, the commitment to u is just the vector $\hat{u}$, i.e., the prover sends $\hat{u}$ to the verifier, and the verifier is given point query access to $\hat{u}$. In the derived polynomial commitment scheme in the plain or random oracle model, the commitment to u will be the Merkle-hash of the vector $\hat{u}$. As with u, we may view $\hat{u}$ as a matrix, with $\hat{u}_i \in \mathbb{F}^N$ denoting the ith row of $\hat{u}$ for $i \in [m]$.

Testing phase. Upon receiving the commitment message, the IOP verifier will interactively test it to confirm that each "row" of u is indeed (close to) a codeword of Enc. We describe this process as occurring in a separate "testing phase" so as to keep the commitment size constant in the plain or random oracle models. In practice, the testing phase can occur during the commit phase, during the evaluation phase, or sometime in between the two.

The verifier sends the prover a random vector $r \in \mathbb{F}^m$, and the prover sends a vector $u' \in \mathbb{F}^m$ claimed to equal the random linear combination of the m rows of u, in which the coefficients of the linear combination are given by r. The verifier reads u' in its entirety.

Next, the verifier tests u' for consistency with $\hat{u}$. That is, the verifier will pick $\ell = \Theta(\lambda)$ random entries of the codeword $\mathsf{Enc}(u') \in \mathbb{F}^N$ and confirm that $\mathsf{Enc}(u')$ is consistent with $v \in \mathbb{F}^N$ at those entries, where v is:

$$\sum_{i=1}^{m} r_i \hat{u}_i \in \mathbb{F}^N. \tag{6}$$

Observe that, by definition of v (Equation (6)), any individual entry v_j of v can be learned by querying m entries of $\hat{u}$ (we refer to these m entries as the "j'th column" of $\hat{u}$). Meanwhile, since the verifier reads u' in its entirety; $\mathcal{V}$ can compute $\mathsf{Enc}(u')_j$ for all desired $j \in [N]$ in $O(m)$ time.

Evaluation phase. As per Section 7, let $q_1, q_2 \in \mathbb{F}^m$ be such that

$$g(r) = \langle (q_1 \otimes q_2), u \rangle.$$

The evaluation phase is identical to the testing phase, except that r is replaced with q_1 (and the verifier uses fresh randomness to choose the sets of coordinates used for consistency testing). Let $u'' \in \mathbb{F}^m$ denote the vector that the prover sends in this phase, which is claimed to equal $\sum_{i=1}^{m} q_{1,i} \cdot u_i$. If the prover is honest, then u'' satisfies $\langle u'', q_2 \rangle = \langle (q_1 \otimes q_2), u \rangle$. Hence, if the verifier's consistency tests all pass in the testing and evaluation phases, the verifier outputs $\langle u'', q_2 \rangle$ as $g(r)$.

Description of polynomial commitment in the language of IOPs. Following standard transformations [Kil92, Mic94, Val08, BCS16], in the actual polynomial commitment scheme, vectors sent by the prover in the IOP may be replaced with a Merkle-commitment to that vector, and each query the verifier makes to a vector is answered by the prover along with Merkle-tree authentication path for the answer. Each phase of the scheme can be rendered non-interactive using the Fiat-Shamir transformation [FS86].

Commit phase.

- $\mathcal{P} \to \mathcal{V}$: a vector $\hat{u} = (\hat{u}_1, \ldots, \hat{u}_m) \in \left(\mathbb{F}^N\right)^m$. If $\mathcal{P}$ is honest, each "row" $\hat{u}_i$ of $\hat{u}$ contains a codeword in Enc.

Testing phase.

- $\mathcal{V} \to \mathcal{P}$: a random vector $r \in \mathbb{F}^m$.
- $\mathcal{P} \to \mathcal{V}$ sends a vector $u' \in \mathbb{F}^m$ claimed to equal $v = \sum_{i=1}^m r_i \cdot u_i \in \mathbb{F}^m$.
- //Now $\mathcal{V}$ probabilistically checks consistency between $\hat{u}$ and u' ($\mathcal{V}$ reads u' in entirety).
- $\mathcal{V}$: chooses Q to be a random set of size $\ell = \Theta(\lambda)$ with $Q \subseteq [N]$. For each $j \in Q$:
  - $\mathcal{V}$ queries all m entries of the corresponding "column" of $\hat{u}$, namely $\hat{u}_{1,j}, \ldots, \hat{u}_{m,j}$.
  - $\mathcal{V}$ confirms that $\mathsf{Enc}(u')_j = \sum_{i=1}^m r_i \cdot \hat{u}_{i,j}$, halting and rejecting if not.

Evaluation phase.

- Let $q_1, q_2 \in \mathbb{F}^m$ be such that $g(r) = \langle (q_1 \otimes q_2), z \rangle$.
- The evaluation phase is identical to the testing phase, except that r is replaced with q_1 (and fresh randomness is used to choose a set Q' of columns for use in consistency checking).
- If all consistency tests pass, then $\mathcal{V}$ outputs $\langle u', q_2 \rangle$ as $g(r)$.

Concrete optimizations to the commitment scheme. Here are optimizations that can reduce the "authentication paths" provided in the testing and evaluation phases by constant factors without affecting the correctness guarantees of the commitment scheme. First, in settings where the evaluation phase will only be run once, the testing phase and evaluation phase can be run in parallel and the same query set Q can be used for both testing and evaluation. This saves roughly a factor of 2 in the proof size. Second, in settings where the commitment is trusted (e.g., applying the polynomial commitment to achieve holography as per Section 5), the testing phase can be omitted. An additional concrete optimization that applies when working over fields of size smaller than $\exp(\lambda)$ is described in Appendix B.

Soundness analysis for the testing phase. The following claim roughly states that if $\hat{u} = (\hat{u}_1, \ldots, \hat{u}_m) \in \left(\mathbb{F}^N\right)^m$, then if even a single $\hat{u}_i$ is far from all codewords in Enc, then a random linear combination of the $\hat{u}_i$'s is also far from all codewords with high probability.

Claim 5. *(Ames, Hazay, Ishai, and Venkitasubramaniam [AHIV17]) Let $\hat{u} = (\hat{u}_1, \ldots, \hat{u}_m) \in \left(\mathbb{F}^N\right)^m$ and for each $i \in [m]$ let c_i be the closest codeword in Enc to $\hat{u}_i$. Let E with $|E| \leq (\gamma/4)N$ be a subset of the columns $j \in [N]$ of $\hat{u}$ on which there is even one row $i \in [m]$ such that $\hat{u}_{i,j} \neq c_{i,j}$. With probability at least*

$$1 - (|E| + 1)/|\mathbb{F}| > 1 - N/|\mathbb{F}|$$

over the choice of $r \in \mathbb{F}^m$, $\sum_{i=1}^m r_i \cdot \hat{u}_i$ has distance at least $|E|$ from any codeword in Enc.

Lemma 1. *If the prover passes all of the checks in the testing phase with probability at least*

$$N/|\mathbb{F}| + (1 - \gamma/4)^\ell,$$

then there is a sequence of m codewords $c_1, \ldots, c_m$ in Enc such that

$$E := |\{j \in [N] \colon \exists i \in [m] \text{ such that } c_{i,j} \neq \hat{u}_{i,j}\}| \leq (\gamma/4)N. \tag{7}$$

Proof. Let $d(b, c)$ denote the relative Hamming distance between two vectors $b, c \in \mathbb{F}^N$. Assume by way of contradiction that Equation (7) does not hold. We explain that the prover passes the consistency tests during the testing phase with probability less than $N/|\mathbb{F}| + (1 - \gamma/4)^\ell$.

Recall that v denotes $\sum_{i=1}^{m} r_i \hat{u}_i$. By Claim 5, the probability over the verifier's choice of r that there exists a codeword a satisfying $d(a, v) > \gamma/4$ is less than $N/|\mathbb{F}|$. If no such a exists, then $d(\mathsf{Enc}(u'), v) \geq \gamma/4$. In this event, all of the verifier's consistency tests pass with probability at most $(1 - \gamma/4)^{\ell}$.

□

Completeness and binding of the polynomial commitment scheme. Completeness holds by design.

To argue binding, recall from the analysis of the testing phase that c_i denotes the codeword in Enc that is closest to row i of $\hat{u}$, and let $w \coloneqq \sum_{i=1}^{m} q_{1,i} \cdot c_i$. We show that, if the prover passes the verifier's checks in the testing phase with probability more than $N/|\mathbb{F}| + (1 - \gamma/4)^{\ell}$ and passes the verifier's checks in the evaluation phase with probability more than $(1 - (3/4)\gamma)^{\ell}$, then $w = \mathsf{Enc}(u'')$.

If $w \neq \mathsf{Enc}(u'')$, then w and $\mathsf{Enc}(u'')$ are two distinct codewords in Enc and hence they can agree on at most $(1-\gamma) \cdot N$ coordinates. Denote this agreement set by A. The verifier rejects in the evaluation phase if there is any $j \in Q'$ such that $j \notin A \cup E$, where E is as in Equation (7). $|A \cup E| \leq |A| + |E| \leq (1-\gamma) \cdot N + (\gamma/4)N = (1 - (3/4)\gamma)N$, and hence a randomly chosen column $j \in [N]$ is in $A \cup E$ with probability at most $1 - (3/4)\gamma$. It follows that u'' will pass the verifier's consistency checks in the evaluation phase with probability at most $(1 - (3/4)\gamma)^{\ell}$.

In summary, we have shown that if the prover passes the verifier's checks in the commitment phase with probability at least $N/|\mathbb{F}| + (1 - \gamma/4)^{\ell}$, then, in the following sense, the prover is *bound* to the polynomial g^* whose coefficients in the Lagrange basis are given by $c_{1,1}, \ldots, c_{m,m}$, where $c_i \in \mathbb{F}^N$ denotes the closest codeword to row i of the vector $\hat{u}$ sent in the commitment phase: on evaluation query r, the verifier either outputs $g^*(r)$, or else rejects in the evaluation phase with probability at least $1 - (1 - (3/4)\gamma)^{\ell}$. Furthermore, the polynomial commitment scheme provides standard extractability properties. This is because with the transformation of [Kil92, Mic94, Val08, BCS16], informally speaking, given a prover that convinces a verifier, one can efficiently extract the IOP proof strings from the prover; row-by-row decoding of the prover's proof string in the commitment phase provides the coefficients of the multilinear polynomial that the prover is bound to.

B Tighter Analysis for the Reed-Solomon Code

By invoking a state of the art analysis in place of Claim 5, one can concretely improve the number of columns that must be opened by the verifier in the polynomial commitment when the error-correcting code used is the Reed-Solomon code. The following claim is a rephrasing of [BCI+20, Theorem 1.6].

Claim 6. *(Ben-Sasson, Carmon, Ishai, Kopparty, and Saraf [BCI+20]) Let Enc be the encoding function of a Reed-Solomon code over $\mathbb{F}$ with message length k, blocklength N, and rate $\rho = (k+1)/N$. Let $\hat{u} = (\hat{u}_1, \ldots, \hat{u}_m) \in \left(\mathbb{F}^N\right)^m$ and for each $i \in [m]$ let c_i be the closest codeword in Enc to $\hat{u}_i$. Let $\delta \in (0, \frac{1-\rho}{2}]$. let E with $|E| < \delta N$ be a subset of the columns $j \in [N]$ of $\hat{u}$ on which there is even one row $i \in [m]$ such that $\hat{u}_{i,j} \neq c_{i,j}$. Let $\epsilon = N/|\mathbb{F}|$. With probability at least $1 - \epsilon$ over the choice of $r \in \mathbb{F}^m$, $\sum_{i=1}^{m} r_i \cdot \hat{u}_i$ has distance at least $|E|$ from any codeword in Enc.*

Lemma 2. *Let ρ and $\epsilon = N/|\mathbb{F}|$ be as in Claim 6 and let $\delta \in (0, \frac{1-\rho}{2}]$. If the prover passes all of the checks in the testing phase with probability at least*

$$\epsilon + (1 - \delta)^{\ell},$$

then there is a sequence of m codewords $c_1, \ldots, c_m$ in Enc such that

$$E \coloneqq |\{j \in [N] \colon \exists i \in [m] \text{ such that } c_{i,j} \neq \hat{u}_{i,j}\}| \leq \delta N. \tag{8}$$

Proof. Let $d(b, c)$ denote the relative Hamming distance between two vectors $b, c \in \mathbb{F}^N$. Assume by way of contradiction that Equation (8) does not hold. We explain that the prover passes the consistency tests during the testing phase with probability less than $\epsilon + (1 - \delta)^{\ell}$.

Recall that v denotes $\sum_{i=1}^{m} r_i \hat{u}_i$. By Claim 6, the probability over the verifier's choice of r that there exists a codeword a satisfying $d(a, v) \leq \delta$ is less than ϵ. If no such a exists, then $d(\mathsf{Enc}(u'), v) > \delta$. In this event, all of the verifier's consistency tests pass with probability at most $(1-\delta)^{\ell}$.

□

Completeness and binding of the polynomial commitment scheme. Completeness holds by design.

To argue binding, recall from the analysis of the testing phase that c_i denotes the codeword in Enc that is closest to row i of $\hat{u}$, and let $w \coloneqq \sum_{i=1}^{m} q_{1,i} \cdot c_i$. Let ρ, δ and $\epsilon = N/|\mathbb{F}|$ be as in Lemma 2. We show that, if the prover passes the verifier's checks in the testing phase with probability more than $\epsilon + (1-\delta)^{\ell}$ and passes the verifier's checks in the evaluation phase with probability more than $(\rho+\delta)^{\ell}$, then $w = \mathsf{Enc}(u'')$.

If $w \neq \mathsf{Enc}(u'')$, then w and $\mathsf{Enc}(u'')$ are two distinct codewords in Enc and hence they can agree on at most $\rho \cdot N$ coordinates. Denote this agreement set by A. The verifier rejects in the evaluation phase if there is any $j \in Q'$ such that $j \notin A \cup E$, where E is as in Equation (8). $|A \cup E| \leq |A| + |E| \leq \rho \cdot N + \delta N = (\rho+\delta)N$, and hence a randomly chosen column $j \in [N]$ is in $A \cup E$ with probability at most $\rho + \delta$. It follows that u'' will pass the verifier's consistency checks in the evaluation phase with probability at most $(\rho+\delta)^{\ell}$.

In summary, we have shown that if the prover passes the verifier's checks in the testing phase with probability at least $\epsilon + (1-\delta)^{\ell}$, then, in the following sense, the prover is *bound* to the polynomial g^* whose coefficients in the Lagrange basis are given by $c_{1,1}, \ldots, c_{m,m}$, where $c_i \in \mathbb{F}^N$ denotes the closest codeword to row i of the vector $\hat{u}$ sent in the commitment phase: on evaluation query r, the verifier either outputs $g^*(r)$, or else rejects in the evaluation phase with probability at least $1 - (\rho+\delta)^{\ell}$. Furthermore, the polynomial commitment scheme provides standard extractability properties. This is because with the transformation of [Kil92, Mic94, Val08, BCS16], informally speaking, given a prover that convinces a verifier, one can efficiently extract the IOP proof strings from the prover; row-by-row decoding of the prover's proof string in the commitment phase provides the coefficients of the multilinear polynomial that the prover is bound to. Setting $\delta = \frac{1-\rho}{2}$, we obtain the following theorem.

Theorem 11. *Consider the polynomial commitment scheme described in Appendix A using the Reed-Solomon code. If the prover passes the verifier's checks in the testing phase with probability at least $N/|\mathbb{F}| + (\frac{1+\rho}{2})^{\ell}$, then, in the following sense, the prover is* bound *to the polynomial g^* whose coefficients in the Lagrange basis are given by $c_{1,1}, \ldots, c_{m,m}$, where $c_i \in \mathbb{F}^N$ denotes the closest codeword to row i of the vector $\hat{u}$ sent in the commitment phase: on evaluation query r, the verifier either outputs $g^*(r)$, or else rejects in the evaluation phase with probability at least $1 - \left(\frac{1+\rho}{2}\right)^{\ell}$. The polynomial commitment is extractable.*

An optimization over small fields. The probability $N/|\mathbb{F}| + (\frac{1+\rho}{2})^{\ell}$ appearing in Theorem 11 can be driven arbitrarily close to $N/|\mathbb{F}|$ by increasing ℓ. However, for small fields, $N/|\mathbb{F}|$ may be larger than the desired soundness error $\epsilon = \exp(-\lambda)$. In this event, one can modify the polynomial commitment scheme as follows so as to replace $N/|\mathbb{F}| + (\frac{1+\rho}{2})^{\ell}$ with $(N/|\mathbb{F}|)^{\eta} + \eta \cdot (\frac{1+\rho}{2})^{\ell}$ for any constant $\eta > 1$. The Testing Phase is repeated η times in parallel, but with the same random subset $Q \subseteq [N]$ of columns (with $|Q| = \ell = \Theta(\lambda)$) used in all η repetitions. The proof of Theorem 11 is easily extended to show that Theorem 11 applies to this modification of the polynomial commitment, with $N/|\mathbb{F}| + (\frac{1+\rho}{2})^{\ell}$ replaced with $(N/|\mathbb{F}|)^{\eta} + \eta \cdot (\frac{1+\rho}{2})^{\ell}$.

By using the same set Q in all η invocations of the testing phase, one avoids a factor-η blowup in the proof length (as revealing all "columns" in Q of $\hat{u}$ is the bottleneck in the proof length). When using the Reed-Solomon code, the repetitions of the Testing Phase also do not substantially increase the prover time, because the bottleneck in the prover time is computing $\hat{u}$ in the Commit Phase, not the Test Phase (this holds both asymptotically when using Reed-Solomon codes, and concretely for large enough instance sizes N).