Brakedown: Linear-time and post-quantum SNARKs for R1CS*

Alexander Golovnev
Georgetown University

Jonathan Lee
Nanotronics Imaging

Srinath Setty
Microsoft Research

Justin Thaler
Georgetown University

Riad S. Wahby
Stanford University

Abstract

This paper introduces *Brakedown*,[1] the first built system that provides *linear-time* SNARKs for NP, meaning the prover incurs $O(N)$ finite field operations to prove the satisfiability of an N-sized R1CS instance. Brakedown's prover is faster, both concretely and asymptotically, than prior SNARK implementations. Brakedown does not require a trusted setup and is plausibly post-quantum secure. Furthermore, it is compatible with *arbitrary* finite fields of sufficient size; this property is new amongst implemented arguments with sublinear proof sizes.

To design Brakedown, we observe that recent work of Bootle, Chiesa, and Groth (BCG, TCC 2020) provides a polynomial commitment scheme that, when combined with the linear-time interactive proof system of Spartan (CRYPTO 2020), yields linear-time IOPs and SNARKs for R1CS (a similar theoretical result was previously established by BCG, but our approach is conceptually simpler, and crucial for achieving high-speed SNARKs). A core ingredient in the polynomial commitment scheme that we distill from BCG is a linear-time encodable code. Existing constructions of such codes are believed to be impractical. Nonetheless, we design and engineer a new one that is practical in our context.

We also implement a variant of Brakedown that uses Reed-Solomon codes instead of our linear-time encodable codes; we refer to this variant as *Shockwave*. Shockwave is *not* a linear-time SNARK, but it provides shorter proofs and lower verification times than Brakedown (it also provides a faster prover than prior plausibly post-quantum SNARKs).

As a modest additional contribution, we observe that one can render the aforementioned SNARK zero knowledge and reduce the proof size and verifier time from $O(\sqrt{N})$ to polylog(N)—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.

*This manuscript is an update to [LSTW21]. It contains substantial new results and updated presentation, including the design and implementation of a concretely efficient error-correcting code with linear-time encoding, and associated SNARK.

[1]In the Transformers universe, Brakedown and Shockwave are powerful (and fast) Decepticons.

1 Introduction

A SNARK [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 the size of π and the cost to verify it are both sub-linear (ideally, at most polylogarithmic) in the size of the witness. Given their many applications, constructing SNARKs with excellent asymptotics and concrete efficiency is a highly active area of research. Still, one of the key bottlenecks preventing application of existing SNARKs to large NP statements is the prover's asymptotic and concrete cost. This has limited practical uses of SNARKs to applications in which NP statements of interest are relatively small (for example, cryptocurrencies).

As with much of the literature on SNARKs, we focus on the following NP-complete problems: rank-1 constraint satisfiability (R1CS)[2] 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 SNARKs 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 efficient transformations and compiler toolchains to transform applications of interest to these representations [SVP$^+$12, PGHR13, BFR$^+$13, BCGT13, WSR$^+$15, SAGL18a, LNS20, OBW20].

Our focus in this work is designing SNARKs for these problems with the fastest possible prover. We also wish to avoid a trusted setup, and desire a verifier that runs in time sub-linear in the size of the R1CS instance. 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. In this phase, the verifier computes a *computation commitment*, a cryptographic commitment to the structure of a circuit or R1CS instance [Set20]. After the pre-processing phase, the verifier must run in time sub-linear in the size of the R1CS instance. 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, BCG20].[3]

A second focus of our work is designing SNARKs that can operate over arbitrary (sufficiently large) finite fields. Prior SNARKs apply over fields that are "discrete-log friendly" or "FFT-friendly", or otherwise require one or many multiplicative or additive subgroups of specified sizes. Yet many cryptographic applications naturally work over fields that do not satisfy these properties. Examples include proofs regarding encryption or signature schemes that themselves work over fields that do not satisfy the properties needed by the SNARK. Indeed, most practically relevant elliptic curve groups are defined over fields that are not FFT-friendly. Even in applications where SNARK designers do have flexibility in field choice, field size restrictions can still create engineering challenges or inconveniences, as well as performance overheads. For example, they may limit the size of R1CS statements that can be handled over the chosen field, or force instance sizes to be padded to a length corresponding to the size of a subgroup.

In this work we design transparent SNARKs that asymptotically have the fastest possible prover, are plausibly post-quantum secure, and work over arbitrary (sufficiently large) finite fields. To the best of our knowledge, this latter property is new amongst implemented arguments with sublinear proof size and even *quasi*linear runtime. We optimize and implement these SNARKs, and demonstrate the fastest prover performance in the SNARK literature (even compared to SNARKs that require FFT-friendly or discrete-log–friendly fields).

Formalizing "fastest possible" provers. How fast can we hope for the prover in a SNARK to be? Letting N denote the size of the R1CS or arithmetic-circuit-satisfiability instance over an *arbitrary* finite field $\mathbb{F}$,[4] 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 SNARK prover that is only a constant factor slower than native evaluation. Such a prover is said to run in *linear-time.*

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

[3] For "structured" computations, our work, like several prior works, can avoid this pre-processing phase.

[4] 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.

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 SNARKs, 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 $\mathbb{F}$. It is not clear whether or not applying a standard hash function such as SHA-256 or BLAKE to hash a field element should be considered comparable to performing a single field operation.

Theoretical work of Bootle, Cerulli, Ghadafi, Groth, Hajiabadi and Jakobsen [BCG$^+$17] sidesteps this issue by observing 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.[5] 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 $\mathbb{F}$.

The aforementioned hash functions appear to be of primarily theoretical interest because they can be orders of magnitude slower than standard hash functions (e.g., SHA-256 or BLAKE). Hence, in this paper our implementations make use of standard hash functions, and with this choice, Merkle-hashing is not the concrete bottleneck in our implementations. Accordingly, and to simplify discussion, we consider our implemented Merkle-hashing procedure to be linear-time, even if this may not be strictly justified from a theoretical perspective for the reasons described above.

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,[6] and hence for purposes of this work, we do *not* consider this to be linear time.

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 6). 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

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

[6] 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.

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.

In summary, in this paper we consider FFTs and multiexponentations of size $O(N)$ *not* to be linear-time operations, but *do* consider Merkle-hashing of vectors of size $O(N)$ to be linear-time.

Related work. Closely related prior works are as follows (we cover additional related work in Section 10).

Building on Bootle et al. [BCG$^+$17], Bootle, Chiesa, and Groth [BCG20] give an *interactive oracle proof* (IOP) [BCS16] with constant soundness error, in which the prover's work is $O(N)$ finite field operations for an N-sized R1CS instance over any finite field of size $\Omega(N)$.[7,8] Applying standard transformations to their IOP, one can obtain a SNARG in the random oracle model with similar prover costs, or an interactive argument assuming linear-time computable cryptographic hash functions [AHI$^+$17]. Unlike prior SNARGs (even those with a *quasi*-linear time prover), the resulting protocol does not require the field to be FFT-friendly nor discrete-log friendly. Their IOP 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 $O_\lambda(1)$). Bootle, Chiesa, and Liu [BCL20] address these issues by achieving zero-knowledge as well as polylogarithmic proof sizes and verification times (a more detailed discussion of the relationship between our results and those of [BCL20] is in Section 10). Both [BCG20] and [BCL20] are theoretical in nature; they do not implement their schemes nor report concrete performance results.

There is also very recent work related to our goal of working over arbitrary finite fields. Ben-Sasson, Carmon, Kopparty, and Levit [BCKL21] improve the efficiency of FFT-like algorithms that apply over fields with no smooth order root of unity, by a factor of $\exp(\log^* N)$. An explicit motivation for their work is to improve the efficiency of known SNARKs that perform FFTs (e.g., Fractal [COS20]) when operating over "non-FFT-friendly" fields. These results do not eliminate the superlinearity of the prover's runtime in their target SNARKs. Depending on the SNARK used, this also does not totally eliminate the need to work over fields possessing multiplicative or additive subgroups of appropriate sizes (e.g., SNARKs such as Fractal [COS20] require a subgroup of size $\Theta(N)$ not only to perform FFTs, but for other reasons as well.) The algorithms given in [BCKL21] also perform significant pre-computation that is field-specific. We seek (and achieve) high-speed SNARKs that require only black-box access to the addition and multiplication operations of the field, with the only additional information required being a lower bound on the field size to ensure soundness.

In summary, existing works leave open the problem of designing a concretely efficient SNARK that works over arbitrary (sufficiently large) finite fields, much less one with a linear-time prover.

1.1 Results and contributions

We address the above problems with *Brakedown*, a new linear-time SNARK that we design, implement, optimize, and experimentally evaluate. Concretely, Brakedown achieves the fastest SNARK prover in the literature, with additional important properties such as the ability to prove and verify R1CS instances over arbitrary fields and plausible post-quantum security. In addition, we implement and evaluate *Shockwave*, a variant of Brakedown that reduces proof sizes and verification times at the cost of sacrificing a linear-time prover, but nonetheless provides a faster prover than prior plausibly post-quantum secure SNARKs.

Design of linear-time SNARKs. We first distill from [BCG20] a polynomial commitment scheme with a linear-time commitment procedure, and show that it satisfies extractability, a key property required in the

[7] 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.

[8] To achieve a soundness error that is negligible in the security parameter λ, one must restrict to R1CS instances over a "sufficiently large" finite field i.e., where $|\mathbb{F}| = 2^{\Theta(\lambda)}$, or else sacrifice the linear-time prover. Since [BCG20] 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 dependence on λ. The problem of achieving a linear-time prover in an IOP or a SNARK for R1CS over arbitrary fields remains open.

context of SNARKs (the commitment scheme itself is little more than a rephrasing of the results in [BCG20], though [BCG20] did not analyze extractability).[9] 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 in which the time to commit to a polynomial is linear in the size of the polynomial.

We then describe a linear-time *polynomial IOP* for R1CS that is implicit in Spartan [Set20], a work that predates [BCG20]. Finally, we combine the linear-time polynomial IOP for R1CS with the linear-time polynomial commitment scheme in a standard way [BFS20, CHM+20] to obtain linear-time SNARKs.

From an asymptotic perspective, the above methodology recovers the main result of [BCG20], namely an IOP with a linear-time prover that, for security parameter λ, and for an N-sized R1CS instance over any field of size $\exp(\lambda)$ and any fixed $\epsilon > 0$ produces a proof of size $O_\lambda(N^\epsilon)$ that can be verified in time $O_\lambda(N^\epsilon)$—after a one-time preprocessing step, which requires $O(N)$ finite field operations. As a secondary asymptotic result, 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.

We believe that our approach is simpler and more direct than [BCG20]. In particular, we are able to reuse an existing high-speed, linear-time polynomial IOP for R1CS from Spartan [Set20]. This is important both for simplicity and for concrete efficiency. Moreover, it means that our goal of building a concretely fast SNARK prover boils down to optimizing our linear-time polynomial commitment scheme.

A new and concretely fast linear-time encodable code. A major component in the linear-time polynomial commitment scheme that we distill from [BCG20] is a linear-time encodable linear code. Unfortunately, to the best of our knowledge, existing linear-time encodable codes are highly impractical. We therefore design a new linear-time encodable code that is concretely fast in our context. Our code is based on classic results [GDP73, Spi96, DI14], but designing this code was non-trivial and represents a significant technical contribution. We achieve a fast linear code that works over any (sufficiently large) field by leveraging the following four observations: (1) In our setting, to achieve sublinear sized proofs, it is sufficient for the code to achieve relative Hamming distance only a small constant, rather than very close to 1 (higher minimum distance would improve Brakedown's proof length by a constant factor, but would not meaningfully reduce the prover time); (2) Efficient decoding is not necessary, as the prover and verifier only execute the encoding procedure of the code (this observation also appears in prior work [BCG20]); (3) We can (and indeed want to) work over large fields, say, of size at least 2^{127}; and (4) We can use randomized constructions instead of deterministic constructions of pseudorandom objects, so long as the probability that the construction fails to satisfy the necessary distance properties is cryptographically small (e.g., $\leq 2^{-100}$).

Observations (1)–(4) together allow us to strip out much of the complexity of prior constructions. For example, Spielman's celebrated work [Spi96] is focused on achieving both linear-time encoding and decoding, while Druk and Ishai [DI14] focus on improving the minimum distance of Spielman's code. On top of the this, we further optimize and simplify the code construction, and provide a detailed, quantitative analysis to show that the probability our code fails to achieve the necessary minimum distance is cryptographically small.

Implementation, optimization, and experimental results. We implement the aforementioned linear-time SNARK, yielding a system we call Brakedown. Because our linear-time code works over any (sufficiently large) field, and the polynomial IOP from Spartan does as well, Brakedown also works over any field; Brakedown is the first built system to achieve this property. It is also the first built system with a linear-time prover and sub-linear proof sizes and verification times.

We also implement Shockwave,[10] a variant of Brakedown that uses Reed-Solomon codes instead of the fast linear-time code introduced above. Since Shockwave uses Reed-Solomon codes, it is not a linear-time SNARK and requires an FFT-friendly finite field, but it provides concretely shorter proofs and lower verification times than Brakedown and is faster than prior plausibly post-quantum secure SNARKs.

[9] We focus on multilinear polynomials, but the scheme generalizes to arbitrary polynomials such as univariate polynomials (see e.g., [Lee20]).

[10] A prior version of this paper [LSTW21] referred to a (less optimized version) of Shockwave as Cerberus.

Both Shockwave and Brakedown contain simple but crucial concrete optimizations to the polynomial commitment scheme to reduce proof sizes. Neither Shockwave's nor Brakedown's implementations are currently zero-knowledge. However, Shockwave can be rendered zero-knowledge using standard techniques with minimal overhead [AHIV17, XZZ+19, CFS17]. Brakedown could be rendered zero-knowledge while maintaining linear prover time by using one layer of recursive composition with zkShockwave (or another zkSNARK); it is also plausible that it could be rendered zero-knowledge more directly using techniques from [BCL20]. We leave the completion of zero-knowledge implementations to near-term future work.

In terms of experimental results, Brakedown achieves a faster prover than all prior SNARKs for R1CS. Its primary downside relative to prior SNARKs is that its proofs are on the larger side, but they are still far smaller than the size of the NP-witness for R1CS instance sizes beyond several million constraints. Shockwave reduces Brakedown's proof sizes and verification times by about a factor of $6\times$, at the cost of a slower prover (both asymptotically and concretely). Nonetheless, Shockwave already features a concretely faster prover than prior plausibly post-quantum SNARKs. Furthermore, although Shockwave's proof sizes are somewhat larger than most prior schemes with sublinear proof size, they are surprisingly competitive with prior post-quantum schemes such as Fractal [COS20] and Aurora [BCR+19] that have lower *asymptotic* proof size ($\text{polylog}(N)$ rather than $\Theta_\lambda(\sqrt{N})$). Its verification times are competitive with discrete-logarithm based schemes, and in fact superior to prior plausibly post-quantum SNARKs.

On knowledge soundness and extractable polynomial commitments. The work of Bootle, Chiesa, and Groth [BCG20] is concerned only with obtaining an IOP for R1CS satisfying standard soundness; it does not consider knowledge soundness. To transform the IOP into a succinct non-interactive argument *of knowledge* (SNARK), one actually requires the IOP to satisfy knowledge soundness (equivalently, one requires the polynomial commitment scheme that we derive from [BCG20] to satisfy a property called *extractability*). As mentioned earlier, in this work, we observe that this is indeed the case.

The polynomial commitment scheme that we derive from [BCG20] makes use of any linear error-correcting code with constant rate and relative distance. We provide two different knowledge extractors, depending on whether or not the code used has a polynomial-time decoding procedure. The first knowledge extractor uses the code's decoding procedure, and is extremely simple and moreover "straight-line" (the extractor need not rewind the prover). The second knowledge extractor is more complex and non-straight-line, but works without invoking the code's decoding procedure.

The Reed-Solomon code used in Shockwave does have efficient decoding (e.g., via the Berlekamp-Welch algorithm) and hence Shockwave is a SNARK for which the knowledge extractor is straight-line. The code used in Brakedown does not have a (known) efficient decoding procedure. However, Brakedown is still a SNARK, but not via a straight-line extractor. Section 5 provides details.

1.2 Roadmap

Section 3 describes a linear-time polynomial IOP for R1CS implicit in Spartan [Set20]. Section 4 describes a linear-time polynomial commitment scheme distilled from [BCG20] with proof size N^ϵ for any desired constant $\epsilon > 0$. Section 5 gives a self-contained description and security analysis (including extractability) of the the polynomial commitment scheme when $\epsilon = 1/2$, along with important concrete optimizations. (Appendix A provides a concretely improved security analysis of the polynomial commitment scheme when instantiated with the Reed-Solomon code as used in one of our two implemented systems, namely Shockwave.) Section 6 describes the construction and analysis of our concretely efficient linear-time-encodable linear error-correcting code used in our polynomial commitment scheme implementation within Brakedown. Section 7 extends the polynomial commitment scheme to handle sparse polynomials efficiently using techniques from Spartan [Set20]. Section 8 obtains linear-time SNARKs for R1CS by combining the polynomial IOP from Spartan with the polynomial commitment schemes derived in Sections 4-7.

Performance results for our implemented SNARKs (Brakedown and Shockwave) are detailed in Section 9.

2 Preliminaries

We use $\mathbb{F}$ to denote a finite field, λ to denote the security parameter, and $\text{negl}(\lambda)$ 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 $(\mathbb{X}, W) \in \mathcal{R}_{\mathrm{R1CS}}$ 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 $(\mathbb{X}, W) \in \mathcal{R}_{\mathrm{R1CS}}$, then Equation (2) holds with probability 1 over the choice of τ, and if $(\mathbb{X}, W) \notin \mathcal{R}_{\mathrm{R1CS}}$, 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

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

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

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

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

and

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

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):

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

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

Overview. In this section, we observe that a core technique in the work of Bootle, Chiesa, and Groth [BCG20] 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).[11]

Our focus here is on multilinear polynomials, but the scheme described here generalizes to other polynomials such as univariate polynomials (see e.g., [Lee20]). Figure 1 compares the asymptotics of this scheme with prior polynomial commitment schemes.

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. [BCG20] define a generalization of IOPs called *tensor* IOPs. To distinguish tensor IOPs from the standard notion of IOPs that they generalize, [BCG20] refers to standard IOPs as *point-query* IOPs.

Definition 4.1 ([BCG20])**.** 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

[11]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.

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

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

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.*

	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 [BCG20])	$O(N)$ $\mathbb{F}$-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 1: 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.

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. $\square$

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].

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 [BCG20, 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 [BCG20] 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 a quantitative analysis for the case of $t = 2$ in Section 5, which is the case used in our implementations. The analysis given there also considers extractability properties of the commitment scheme, and describes several simple but crucial optimizations to the scheme that result, concretely, in an orders-of-magnitude reduction in proof length relative to a naive implementation.

We remark that the case of $t = 2$ 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[12] of the "inner SNARK" to be smaller than N by any factor that dominates $\lambda/\log N$). The 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 [BCG20].

5 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., $u_i = \{u_{i,j}\}_{j \in [m]}$.

[12] More precisely, the size of the verifier's checks when implemented as an R1CS instance.

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:

$$\sum_{i=1}^{m} r_i \hat{u}_i \in \mathbb{F}^N. \tag{4}$$

Observe that, by definition of v (Equation (4)), 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. 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 proof size in the testing and evaluation phases by large 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, while for simplicity in this section we have described the commitment scheme in the setting where u is indexed by $[m]^2$, i.e., u was viewed as a square matrix, this is not a requirement, and the proof size in the testing and evaluation phases can be substantially reduced by exploiting this flexibility. Specifically, if r and c denote the number of rows and columns of u, so that the number of entries in u is $c \cdot r = N$, then the proof length of the commitment scheme is roughly $2c + r \cdot \ell$ field elements where ℓ is the number of columns of the encoded matrix opened by the verifier. Here, the $2c$ term comes from the prover sending two different linear combination of the rows of u, one in the commitment phase and one in the evaluation phase, while the $r \cdot \ell$ term comes from the verifier querying ℓ different columns of u in the testing and evaluation phases. (This optimization appeared in Ligero [AHIV17] in the context of the Reed-Solomon code).

  To minimize proof length, one should set $c \approx r\ell/2$, or equivalently, one should set $r \approx \sqrt{2/\ell} \cdot \sqrt{N}$ and $c \approx \sqrt{\ell/2} \cdot \sqrt{N}$. This reduces the proof length from roughly $\ell \cdot \sqrt{N}$ if a square matrix is used, to roughly $\sqrt{2\ell} \cdot \sqrt{N}$, a savings of a factor of $\sqrt{\ell/2}$. Asymptotically, this means the proof length falls from $\Theta(\lambda\sqrt{N})$ if a square matrix is used, down to $\Theta(\sqrt{\lambda N})$, a quadratic improvement in the dependence on λ.

  To achieve soundness error, say, 2^{-100}, ℓ will be on the order of hundreds or thousands depending on the relative Hamming distance of the code used, and hence this optimization will lead to a reduction in proof length relative to the use of square matrices by one or more orders of magnitude.[13]

[13] An earlier version of this manuscript missed this optimization, and hence reported over 10x larger proof sizes than necessary for our Reed-Solomon-based implementation, Shockwave.

- Third, in settings where the commitment is trusted (e.g., applying the polynomial commitment to achieve holography as per Section 7), 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 A.
- Fourth, if $\mathcal{P}$ naively commits to the vector $\hat{u} \in (\mathbb{F}^N)^m$ with a Merkle-tree, then revealing all entries of ℓ columns of $\hat{u}$ in the Testing and Evaluation phases would require providing $m \cdot \ell$ Merkle-authentication paths. Naively, this may require $\mathcal{P}$ to send up to $\Theta(m \cdot \ell \cdot \log m)$ hash values. However, by arranging the vector $\hat{u}$ in column-major order before Merkle-hashing it, the communication cost of revealing ℓ columns of $\hat{u}$ can be reduced to just the $m \cdot \ell$ requested field elements plus $O(\log m)$ hash values (a similar optimization appears in prior work [BBHR19]).

Soundness analysis for the testing phase. The following claim roughly states that if $\hat{u} = (\hat{u}_1, \ldots, \hat{u}_m) \in (\mathbb{F}^N)^m$, 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 1. *(Ames, Hazay, Ishai, and Venkitasubramaniam [AHIV17], Roth and Zémor) Let $\hat{u} = (\hat{u}_1, \ldots, \hat{u}_m) \in (\mathbb{F}^N)^m$ and for each $i \in [m]$ let c_i be the closest codeword in Enc to $\hat{u}_i$. Let E with $|E| \le (\gamma/3)N$ 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*

$$N/|\mathbb{F}| + (1-\gamma/3)^\ell,$$

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

$$E := |\{j \in [N] \colon \exists i \in [m] \text{ such that } c_{i,j} \neq \hat{u}_{i,j}\}| \le (\gamma/3)N. \tag{5}$$

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 (5) does not hold. We explain that the prover passes the consistency tests during the testing phase with probability less than $N/|\mathbb{F}| + (1-\gamma/3)^\ell$.

Recall that v denotes $\sum_{i=1}^m r_i \hat{u}_i$. By Claim 1, the probability over the verifier's choice of r that there exists a codeword a satisfying $d(a,v) > \gamma/3$ is less than $N/|\mathbb{F}|$. If no such a exists, then $d(\mathsf{Enc}(u'), v) \ge \gamma/3$. In this event, all of the verifier's consistency tests pass with probability at most $(1-\gamma/3)^\ell$.

□

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 $w := \sum_{i=1}^m q_{1,i} \cdot c_i$. We show that, if the prover passes the verifier's checks in the testing phase with probability more than $N/|\mathbb{F}| + (1-\gamma/3)^\ell$ and passes the verifier's checks in the evaluation phase with probability more than $(1-(2/3)\gamma)^\ell$, 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 (5). $|A \cup E| \le |A| + |E| \le (1-\gamma)\cdot N + (\gamma/3)N = (1-(2/3)\gamma)N$, and hence a randomly chosen column $j \in [N]$ is in $A \cup E$ with probability at most $1-(2/3)\gamma$. It follows that u'' will pass the verifier's consistency checks in the evaluation phase with probability at most $(1-(2/3)\gamma)^\ell$.

In summary, we have shown that if the prover passes the verifier's checks in the commitment phase with probability at least

$$N/|\mathbb{F}| + (1-\gamma/3)^\ell, \tag{6}$$

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

$$1 - (1 - (2/3)\gamma)^{\ell}. \tag{7}$$

The polynomial commitment scheme provides standard extractability properties. We show this by giving two different extractors.

Extractability via efficient decoding. The first is a simple straight-line extractor that is efficient if the error-correcting code Enc has a polynomial-time decoding procedure that can correct up to a $\gamma/4$ fraction of errors. This is because with the IOP-to-succinct-argument transformation of [Kil92, Mic94, Val08, BCS16], it is known that, given a prover $\mathcal{P}$ that convinces the argument-system verifier to accept with non-negligible probability, there is an efficient straight-line extractor capable of outputting IOP proof string π that "opens" the Merkle commitment sent by the argument system prover in the commitment phase. Moreover, there is an IOP prover strategy $\mathcal{P}'$ for the testing and evaluation phases by which $\mathcal{P}'$ can convince the IOP verifier in those phases to accept with non-negligible probability when the first IOP message is π ($\mathcal{P}'$ merely simulates the argument-system prover $\mathcal{P}$ in those phases).

Our analysis of the testing phase of the polynomial commitment scheme (Lemma 1) then guarantees that each row of the extracted string π has relative Hamming distance at most $\gamma/3$ from some codeword. Hence, row-by-row decoding provides the coefficients of the multilinear polynomial that the prover is bound to. So long as the decoding procedure runs in polynomial time, the resulting extractor is efficient.

Extractability without decoding. If the error-correcting code does not support efficient decoding, then even though one can efficiently extract the IOP proof string π underlying the Merkle-comittment sent in the commitment phase of the commitment scheme, one can not necessarily decode (each row of) the string to efficiently extract from π the polynomial that the commiter is bound to.

Instead, the extractor can proceed as follows. We assume throughout the below that Expressions (6) and (7) are negligible (say, exponentially small in the security parameter λ), which holds so long as $|\mathbb{F}| \geq \exp(\lambda)$ and the number of column openings is $\ell = \Theta(\lambda)$.

The testing phase of the commitment scheme can be viewed as a 3-move public-coin argument in which the verifier moves first. First, the verifier sends a challenge vector $r \in \mathbb{F}^m$. Second, the prover responds with a vector claimed to equal $\sum_{i=1}^m r_i u_i$. Third, the verifier chooses a set Q of random columns to use in the consistency test, and performs the consistency test by querying the committed proof string π at all entries of the columns in Q.

Given any efficient prover strategy that passes the verifier's checks in the testing phase with non-negligible probability, we show in the following lemma that there is a polynomial-time extraction procedure capable of outputting m linearly independent challenge vectors $r_1, \ldots, r_m \in \mathbb{F}^m$ from the testing phase of the protocol, and m response vectors $u'_1, \ldots, u'_m \in \mathbb{F}^m$ of the prover, each of which pass the verifier's consistency checks in the testing phase with non-negligible probability.

Lemma 2. *Suppose there is a deterministic prover strategy $\mathcal{P}$ that, following the commitment phase of the polynomial commitment scheme, passes the verifier's checks in the testing phase of the polynomial commitment scheme with non-negligible probability ϵ. Then there is a randomized extraction procedure $\mathcal{E}$ that runs in time poly$(m, 1/\epsilon)$ and such that the following holds. Given the ability to repeatedly rewind $\mathcal{P}$ to the start of the testing phase, with probability at least $9/10$, $\mathcal{E}$ outputs m linearly independent challenge vectors $r_1, \ldots, r_m \in \mathbb{F}^m$ from the testing phase, and m corresponding response vectors $u'_1, \ldots, u'_m \in \mathbb{F}^m$ of the prover, each of which pass the verifier's checks in the testing phase with probability at least ϵ.*

Before proving Lemma 2, we explain how to extract the desired polynomial given the extracted challenge vectors $r_1, \ldots, r_m \in \mathbb{F}^m$ and m response vectors $u'_1, \ldots, u'_m \in \mathbb{F}^m$. Observe that the testing phase and the evaluation phase of the polynomial commitment scheme are identical up to how the challenge vector is selected. In addition, for each challenge r_i the prover's response u'_i passes the verifier's consistency checks with non-negligible probability. Hence, the binding analysis for the commitment scheme implies that $u'_1, \ldots, u'_m$ are all consistent with the evaluations of a fixed multilinear polynomial g^*, i.e., for $i = 1, \ldots, m$, $u'_i = r_i^T \cdot C$ where C is the coefficient matrix of g^* in the Lagrange basis. Since the r_i vectors are linearly independent, these m linear equations uniquely specify C, and in fact C can be found in polynomial time using Gaussian elimination.

Proof of Lemma 2. Fix the extracted proof string $\pi = (\pi_1, \ldots, \pi_m) \in (\mathbb{F}^N)^m$ that "opens" the Merkle-commitment sent by the committer during the commitment phase.

Observe that for any verifier challenge $r' \in \mathbb{F}^m$ and prover response u', one can efficiently compute the probability (over the random choice of column set Q) that u' will pass the verifier's consistency checks. Specifically, if η is the the number of columns i such that $\left(\sum_{j=1}^m r'_j \pi_j\right)_i = u'_i$, and ℓ is the number of columns selected by the verifier, then this probability is $(\eta/N)^\ell$ (here, for simplicity let us assume columns are selected with replacement, but an exact expression can also be given when columns are selected without replacement).

Let T denote the set of all challenges r such that $\mathcal{P}$'s response u to r passes the consistency checks with probability at least $\epsilon/2$. By averaging, since $\mathcal{P}$ passes all checks in the testing phase with probability at least ϵ, $|T| \geq (\epsilon/2) \cdot |\mathbb{F}|^m$. The extractor's goal is to efficiently identify a subset $S = \{r_1, \ldots, r_m\}$ of T that spans $\mathbb{F}^m$.

The extractor $\mathcal{E}$ works by repeatedly picking challenge vectors r uniformly at random from $\mathbb{F}^m$, and running $\mathcal{P}$ on challenge r to get a response u; this enables $\mathcal{E}$ to determine whether $r \in T$, and if so, $\mathcal{E}$ adds r to S. Since $|T| \geq (\epsilon/2) \cdot |\mathbb{F}|^m$, with probability at least $9/10$, $\mathcal{E}$ will identify at least m vectors to add to S after trying at most $18/\epsilon$ vectors r.

We now argue that with overwhelming probability, the first m vectors that $\mathcal{E}$ adds to S are linearly independent. Denote these m vectors by $r_1, \ldots, r_m \in \mathbb{F}^m$. Observe that each vector r_i is a random element of T. We now explain that for each $i = 2, \ldots, m$, the probability that $r_i \in \mathrm{span}(r_1, \ldots, r_{i-1})$ is negligible. To see this, observe that since the dimension of $\mathrm{span}(r_1, \ldots, r_{i-1})$ is at most $i-1$, the span contains at most $|\mathbb{F}|^{i-1}$ vectors. Since r_i is a uniform random vector from T and $|T| \geq (\epsilon/2) \cdot \mathbb{F}^m$, the probability that $r_i \in \mathrm{span}(r_1, \ldots, r_{i-1})$ is at most $(2/\epsilon) \cdot |\mathbb{F}|^{i-1}/|\mathbb{F}|^m \leq (2/\epsilon) \cdot |\mathbb{F}|^{-1}$.

The claim then follows by a union bound over all $m-1$ vectors $r_2, \ldots, r_m$. That is, the probability that $r_1, \ldots, r_m$ are not linearly independent is at most $(m-1) \cdot (2/\epsilon) \cdot |\mathbb{F}|^{-1}$. This is negligible by our assumption that ϵ is non-negligible while $|\mathbb{F}|^{-1}$ is negligible.

□

6 Practical Linear Codes with Linear-Time Encoding

This section describes our construction of practical linear codes with linear-time encoding that we use in Brakedown's implementation of the polynomial commitment scheme from Section 5.

Let q be a prime power, and $\mathbb{F} = \mathbb{F}_q$ be the field of size q. For $p \in [0,1]$, by $H(p) = -p\log_2(p) - (1-p)\log_2(1-p)$ we denote the binary entropy function, where we adopt the convention that $0 \log 0 = 0$. For $k \leq n/2$, we'll use the bound $\sum_{0 \leq i \leq k} \binom{n}{i} \leq 2^{nH(k/n)}$. We'll also use the bounds $\binom{n}{k} \leq 2^{nH(k/n)}$ and $\binom{n}{k} \leq \left(\frac{en}{k}\right)^k$ for $0 \leq k \leq n$.

Let $\mathcal{G}_{n,m,d}$ be a distribution of bipartite graphs with n vertices in the left part and m vertices in the right part, where each vertex in the left part has d distinct uniformly random neighbors in the right part. Let $\mathcal{M}_{n,m,d}$ be a distribution of matrices $M \in \mathbb{F}^{n \times m}$, where in each row d distinct uniformly random elements are assigned uniformly random non-zero elements of $\mathbb{F}$.

We construct a systematic linear code with efficient encoding procedure. The code uses the parameters $0 < \alpha < 1, 0 < \beta < \alpha/1.28, r > (1+2\beta)/(1-\alpha) > 1, c_n, d_n \geq 3$ that will be specified later. For a constant $r > 1$, in Algorithm 1 (see also Figure 2 for a visual depiction of the encoding procedure) we give a construction of a linear map $\mathsf{Enc}_n \colon \mathbb{F}^n \to \mathbb{F}^{rn}$ such that every non-zero vector $x \in \mathbb{F}^n$ is mapped to a vector $w \in \mathbb{F}^{rn}$

of Hamming weight at least $\|w\|_0 \geq \beta n$. Moreover, the linear map Enc_n is systematic, that is, the first n coordinates of w equal x. This guarantees that the constructed map has full rank, and, that it defines a linear code of rank n, rate $1/r$, and distance $\delta = \beta n/(rn) = \beta/r$.

The map Enc_n works as follows. First we generate a random sparse matrix $A \leftarrow \mathcal{M}_{n,\alpha n,c_n}$ for

$$c_n = \left\lceil \min\left(\max(1.28\beta n,\ \beta n + 4),\ \frac{1}{\beta \log_2 \frac{\alpha}{1.28\beta}} \left(\frac{110}{n} + H(\beta) + \alpha H\left(\frac{1.28\beta}{\alpha}\right)\right)\right)\right\rceil ,$$

and compute $y = x \cdot A \in \mathbb{F}^{\alpha n}$. Recall that the parameter $\alpha < 1$, and, thus, we can recursively apply the encoding procedure Enc to y: $z = \mathsf{Enc}_{\alpha n}(y) \in \mathbb{F}^{\alpha r n}$. Finally, we generate a random sparse matrix $B \leftarrow \mathcal{M}_{\alpha r n,(r-1-r\alpha)n,d_n}$ for

$$d_n = \left\lceil \min\left(\left(2\beta + \frac{(r-1) + 110/n}{\log_2 q}\right) n,\ \frac{r\alpha H(\beta/r) + \mu H(\nu/\mu) + 110/n}{\alpha\beta \log_2 \frac{\mu}{\nu}} \right) \right\rceil ,$$

where $\mu = r - 1 - r\alpha, \nu = \beta + \alpha\beta + 0.03$, and compute $v = z \cdot B \in \mathbb{F}^{(r-1-r\alpha)n}$. The resulting codeword is the concatenation of x, z, and v:

$$w = \mathsf{Enc}(x) \coloneqq \begin{pmatrix} x \\ z \\ v \end{pmatrix} \in \mathbb{F}^{rn} .$$

It is easy to see that the constructed code is linear and systematic, and has rate $1/r$. Therefore, it remains to show that this code has distance $\delta = \beta/r$, and to estimate the running time of its encoding procedure. While our analysis of the running time of the encoding procedure is asymptotic (in that it holds for large enough values of n), our analysis of the code distance is concrete and holds for all values of n.

Algorithm 1 Encoding Algorithm $\mathsf{Enc}_n \colon \mathbb{F}^n \to \mathbb{F}^{rn}$

Input: $x \in \mathbb{F}^n$
Parameters: $\alpha, \beta, r, c_n, d_n$
Output: $w \in \mathbb{F}^{rn}$

$c_n = \left\lceil \min\left(\max(1.28\beta n,\ \beta n + 4),\ \frac{1}{\beta \log_2 \frac{\alpha}{1.28\beta}} \left(\frac{110}{n} + H(\beta) + \alpha H\left(\frac{1.28\beta}{\alpha}\right)\right)\right)\right\rceil$
$A \leftarrow \mathcal{M}_{n,\alpha n,c_n}$
$y = x \cdot A \in \mathbb{F}^{\alpha n}$
$z = \mathsf{Enc}_{\alpha n}(y) \in \mathbb{F}^{\alpha r n}$
$d_n = \left\lceil \min\left(\left(2\beta + \frac{(r-1) + 110/n}{\log_2 q}\right) n,\ \frac{r\alpha H(\beta/r) + \mu H(\nu/\mu) + 110/n}{\alpha\beta \log_2 \frac{\mu}{\nu}} \right) \right\rceil$,
where $\mu = r - 1 - r\alpha$, $\nu = \beta + \alpha\beta + 0.03$
$B \leftarrow \mathcal{M}_{\alpha r n,(r-1-r\alpha)n,d_n}$
$v = z \cdot B \in \mathbb{F}^{(r-1-r\alpha)n}$
$w = \begin{pmatrix} x \\ z \\ v \end{pmatrix} \in \mathbb{F}^{rn}$
return w

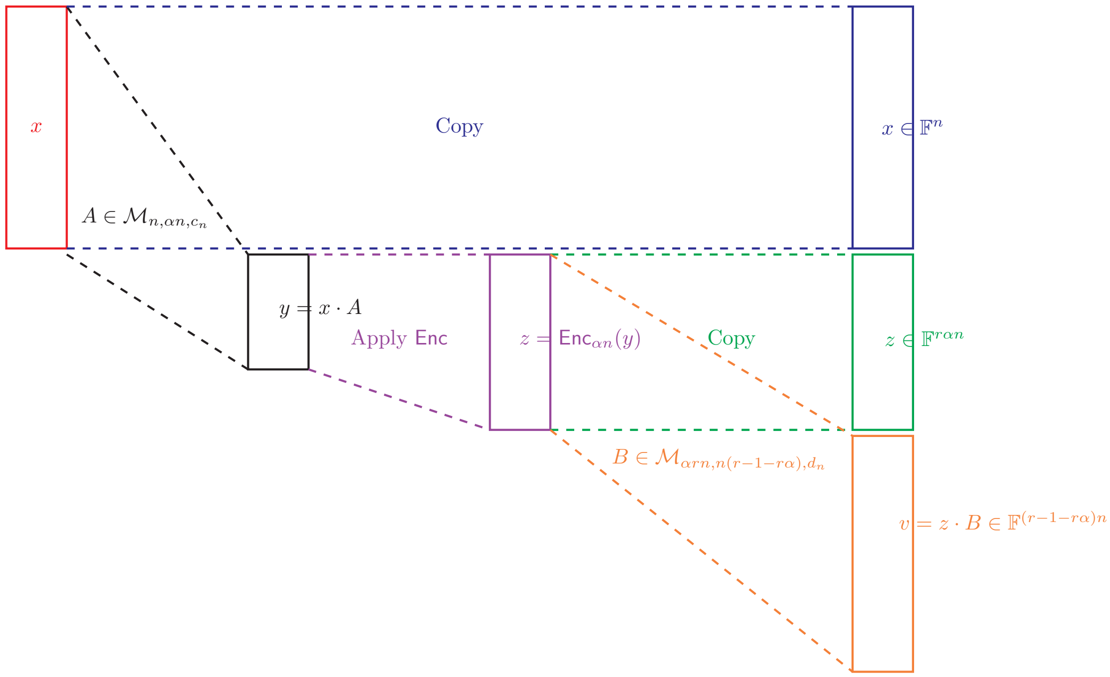

Figure 2: The encoding procedure Enc_n (with all but negligible probability) maps a non-zero vector $x \in \mathbb{F}^n$ to a vector $w = \mathsf{Enc}(x) := \begin{pmatrix} x \\ z \\ v \end{pmatrix} \in \mathbb{F}^{rn}$ of Hamming weight at least $\|w\|_0 \geq \beta n$, resulting in a linear code of rate $1/r$ and distance $\delta = \beta/r$.

Running time. We note that since the encoding procedure consists of a series of multiplications of vectors by sparse matrices, the number of field additions is strictly less than the number of field multiplications. Here we only estimate the number of field multiplications because it's a more resource-intensive operation, and it also gives an essentially tight estimate on the number of field additions. The encoding procedure for a message of length n performs a multiplication of a vector of length n by a matrix of row-sparsity c_n, a multiplication of a vector of length αrn by a matrix of row-sparsity d_n, and a recursive call for a vector of length αn. Letting $T(n)$ denote the running time of Enc_n, we have that

$$T(n) \leq nc_n + \alpha rnd_n + T(\alpha n) = n(c_n + \alpha rd_n) + T(\alpha n)\,.$$

By setting

$$c = \lim_{n\to\infty} c_n = \left\lceil \frac{H(\beta) + \alpha H\left(\frac{1.28\beta}{\alpha}\right)}{\beta \log_2 \frac{\alpha}{1.28\beta}} \right\rceil,$$

$$d = \lim_{n\to\infty} d_n = \left\lceil \frac{r\alpha H(\beta/r) + \mu H(\nu/\mu)}{\alpha\beta \log_2 \frac{\mu}{\nu}} \right\rceil,$$

we have for all large enough n,

$$T(n) \lesssim n(c + \alpha rd) + T(\alpha n) < n \cdot \frac{c + \alpha rd}{1-\alpha}\,,$$

where the last inequality uses the infinite geometric series formula.

Code distance. We show that for certain choices of the parameters α, β, r, c_n, and d_n, the following holds with all but negligible probability over the choices of random matrices A, B:

(i) for every $0 < \|x\|_0 < \beta n$, $y = x \cdot A \neq \mathbf{0}$;

(ii) for every $\alpha\beta n \leq \|z\|_0 < \beta n$, $v = z \cdot B$ has $\|v\|_0 \geq \beta n$.

Assuming that these two properties hold, we can show that Enc_n has distance $\delta = \beta/r$, that is, for every $x \neq \mathbf{0}$, $w = \mathsf{Enc}_n(x)$ satisfies $\|w\|_0 \geq \beta n$. To this end, we consider the following three cases.

1. $\|x\|_0 \geq \beta n$. In this case, $w = \begin{pmatrix} x \\ z \\ v \end{pmatrix}$ trivially satisfies $\|w\|_0 \geq \|x\|_0 \geq \beta n$.
2. $z = \mathsf{Enc}_{\alpha n}(x \cdot A)$ satisfies $\|z\|_0 \geq \beta n$. Again, since w contains z, we have $\|w\|_0 \geq \|z\|_0 \geq \beta n$.
3. $0 < \|x\|_0 < \beta n$ and $\|z\|_0 < \beta n$. In this case, by the Property (i) above, we have that $y = x \cdot A \neq \mathbf{0}$. By the code property of $\mathsf{Enc}_{\alpha n}$, we have that every non-zero vector y is mapped to $z = \mathsf{Enc}_{\alpha n}(y)$ of Hamming weight $\|z\| \geq \delta \cdot (\alpha r n) = \alpha\beta n$. Now we have that $\alpha\beta n \leq \|z\|_0 < \beta n$, and by the Property (ii) above, $\|v\|_0 \geq \beta n$, which finishes the proof.

It remains to choose the values of the parameters $\alpha, \beta, r, c_n, d_n$ such that the Property (i) and (ii) are satisfied with, say, probability at least $1 - 2^{-100}$ over the choice of matrices A and B. We remark that once the code is generated (that is, once we have generated matrices A and B for all relevant values of n), we have the guarantee that with probability $1 - 2^{-100}$, *all* non-zero messages x are mapped to vectors w of Hamming weight $\|w\| \geq \beta n$.

We bound the probability of not satisfying the Property (i) in two steps. First, for $0 < k < \beta n$, let $E^{(1)}_{n,k}$ be the event that there exists a set of k coordinates of $x \in \mathbb{F}^n$ that doesn't "expand" into $b(k) = \max(k + 4,\ 1.28k)$ coordinates of $y = x \cdot A$. Formally, let $E^{(1)}_{n,k}$ be the event that there exists a set of k rows of A that have fewer than $b(k)$ non-zero columns. Second, let $E^{(2)}_{n,k}$ denote the event that, given that every set of size k expands into a set of size at least $b(k)$ (that is, conditioned on the complement of $E^{(1)}_{n,k}$), there exists an x of Hamming weight $\|x\|_0 = k$ such that $y = x \cdot A = \mathbf{0}$. We will choose the parameters α, β, and c_n such that

$$\sum_{\substack{n \\ 0<k<\beta n}} \Pr[E^{(1)}_{n,k}] + \Pr[E^{(2)}_{n,k}] \ll 2^{-100}\,.$$

This will guarantee that with probability at least $1 - 2^{-100}$, the Property (i) is satisfied: every x with $0 < \|x\| < \beta n$ is mapped to a $y \neq \mathbf{0}$.

We use a similar strategy to bound the probability that the constructed code doesn't satisfy the Property (ii). Let $E^{(3)}_{n,k}$ be the event that there exists a set of k coordinates of $z \in \mathbb{F}^{\alpha r n}$ that doesn't expand into $b'(k) = \left(\beta + k/n + \frac{(r-1)+110/n}{\log_2 q}\right) n$ coordinates of $v = z \cdot B$: there exists a set of k rows of B that have fewer than $b'(k)$ non-zero columns. Then we define $E^{(4)}_{n,k}$ as the event that, given that all sets of size k expand into at least $b'(k)$ coordinates, there exists a $z \in \mathbb{F}^{\alpha r n}$ of Hamming weight $\|z\|_0 = k$ which is mapped to $v = B \cdot z$ of Hamming weight $\|b\|_0 < \beta n$. We will choose the parameters to guarantee that

$$\sum_{\substack{n \\ \alpha\beta n \leq k<\beta n}} \Pr[E^{(3)}_{n,k}] + \Pr[E^{(4)}_{n,k}] \ll 2^{-100}\,.$$

This will imply that with probability at least $1 - 2^{-100}$, the constructed code satisfies the Property (ii).

In Table 3 we provide several settings of the parameters and specify the corresponding provable guarantees. These are the parameter settings we use in our implementation, Brakedown. We have chosen parameters in this table to ensure that all guarantees hold for messages of length up to 2^{30}. Recall that in the polynomial commitment scheme of Section 5, the messages to be encoded have length $\Theta(\lambda \cdot N^{1/2})$ where N is the size of the polynomial to be committed. Concretely, messages of length 2^{30} are sufficient for our SNARK to support R1CS instances of size in excess of 2^{40} with the stated failure probability of 2^{-100} of the code failing to satisfy the asserted minimum distance property. Somewhat faster encoding times than those listed in Table 3 can be achieved if one is satisfied with larger failure probability, smaller messages, larger field size, etc.

Below, we also describe several conditions on the parameters sufficient for satisfying the Properties (i) and (ii) with high probability. Namely, in Claims 2, 3, 4, and 5, we give conditions sufficient for bounding the probabilities of the events $E^{(1)}_{n,k}, E^{(2)}_{n,k}, E^{(3)}_{n,k}$, and $E^{(4)}_{n,k}$, respectively. These claims may be useful for readers who wish to optimize code parameters with less stringent requirements than ours (e.g., readers who are satisfied with larger probability, or who need not support messages as large as we do, etc.).

n	q	Pr[failure]	run-time	distance	rate	α	β	r	c_n	d_n
$\le 2^{30}$	$\ge 2^{127}$	$\ll 2^{-100}$	$13.2n$	0.02	0.704	0.1195	0.0284	1.42	6	33
$\le 2^{30}$	$\ge 2^{127}$	$\ll 2^{-100}$	$14.3n$	0.03	0.68	0.138	0.0444	1.47	7	26
$\le 2^{30}$	$\ge 2^{127}$	$\ll 2^{-100}$	$15.8n$	0.04	0.65	0.178	0.061	1.521	7	22
$\le 2^{30}$	$\ge 2^{127}$	$\ll 2^{-100}$	$17.8n$	0.05	0.60	0.2	0.082	1.64	8	19
$\le 2^{30}$	$\ge 2^{127}$	$\ll 2^{-100}$	$20.5n$	0.06	0.61	0.211	0.097	1.616	9	21
$\le 2^{30}$	$\ge 2^{127}$	$\ll 2^{-100}$	$23.9n$	0.07	0.58	0.238	0.1205	1.72	10	20

Figure 3: Settings of the parameters leading to codes with small probabilities of error. n denotes the length of the message to be encoded. The specified values of c_n, d_n, and running time hold for all large enough n.

Claim 2. *Let*

$$c_n = \left\lceil \min\left(\max(1.28\beta n,\ \beta n + 4),\ \frac{1}{\beta \log_2 \frac{\alpha}{1.28\beta}} \left(\frac{110}{n} + H(\beta) + \alpha H\left(\frac{1.28\beta}{\alpha}\right)\right)\right)\right\rceil .$$

If $\beta < \alpha/1.28$, then for every n and $0 < k < \beta n$,

$$\Pr[E^{(1)}_{n,k}] \le \max\Bigg(2^{-110},\ 2^{nH(15/n)+\alpha n H(\frac{19.2}{\alpha n}) - 15 c_n \log_2(\frac{\alpha n}{19.2})},\ \max_{c_n - 3 \le k \le \min(14, \beta n)} \frac{\binom{n}{k}\binom{\alpha n}{k+3}\binom{k+3}{c_n}^k}{\binom{\alpha n}{c_n}^k}\Bigg) .$$

Proof. First we note that

$$\Pr[E^{(1)}_{n,k}] = \Pr_{G \in \mathcal{G}_{n,\alpha n, c_n}} [G \text{ contains a set of } k \text{ left vertices with fewer than } b(k) \text{ neighbors}] .$$

We will show that with all but negligible probability every set of size $k < \beta n$ will expand into a set of size at least $b(k) = \max(k+4,\ 1.28k)$. If $c_n \ge \max(1.28\beta n,\ \beta n + 4)$, then every vertex has $c_n \ge b(\beta n)$ neighbors, and, thus, every set of $k < \beta n$ vertices has at least $b(k)$ neighbors.

Now we assume that $c_n \ge \frac{1}{\beta \log_2 \frac{\alpha}{1.28\beta}} \left(\frac{110}{n} + H(\beta) + \alpha H\left(\frac{1.28\beta}{\alpha}\right)\right)$. In order to show that every set of size k expands into a set of size at least $b(k) = \max(k+4,\ 1.28k)$ with high probability, we'll first bound from above the probability that there exists a set of $k \ge 15$ vertices with at most $1.28k$ neighbors, and then we will bound the probability that there exists a set of $k \le 14$ that has fewer than $k+4$ neighbors.

The probability that a fixed set of size $15 \le k < \beta n$ has fewer than $1.28k$ neighbors is upper-bounded by

$$\frac{\binom{\alpha n}{1.28k}\binom{1.28k}{c_n}^k}{\binom{\alpha n}{c_n}^k} .$$

By the union bound over all sets of k left vertices,

$$\Pr[E^{(1)}_{n,k}] \le \frac{\binom{n}{k}\binom{\alpha n}{1.28k}\binom{1.28k}{c_n}^k}{\binom{\alpha n}{c_n}^k} \le \binom{n}{k}\binom{\alpha n}{1.28k}\left(\frac{1.28k}{\alpha n}\right)^{kc_n} \le 2^{nH(k/n)} \cdot 2^{\alpha n H(\frac{1.28k}{\alpha n})} \cdot 2^{-kc_n \log_2(\frac{\alpha n}{1.28k})}$$

$$= 2^{n\left(H(k/n) + \alpha H(\frac{1.28k}{\alpha n}) - \frac{c_n k}{n}\log_2(\frac{\alpha n}{1.28k})\right)} . \quad (8)$$

Let $f(x) = H(x) + \alpha H(1.28x/\alpha) - c_n x \log_2(\frac{\alpha}{1.28x})$ for $x \in [15/n, \beta]$. We'll show that for our choice of c_n, $f(x)$ is convex, and therefore it achieves its maximum at one of the end points of the interval $x \in [15/n, \beta]$. Indeed, $f''(x) = \frac{\log_2 e}{x}\left(c_n - \frac{1}{1-x} - \frac{1.28\alpha}{\alpha - 1.28x}\right)$. In order to show that $f(x)$ is convex, it's sufficient to show that for every $x \le \beta$, $c_n \ge \frac{1}{1-x} + \frac{1.28\alpha}{\alpha-1.28x}$. Letting $g(x) = \frac{1}{1-x} + \frac{1.28\alpha}{\alpha-1.28x}$, we have that $g'(x) = \frac{1.28^2\alpha}{(\alpha-1.28x)^2} + \frac{1}{(x-1)^2} > 0$. Thus, $g(x)$ is monotone, and it remains to verify that $c_n \ge \frac{1}{1-x} + \frac{1.28\alpha}{\alpha-1.28x}$ for $x = \beta$. For our choice of c_n, for every $\beta < \alpha/1.28$ it can be verified that

$$\begin{aligned} c_n &\ge \frac{1}{\beta \log_2 \frac{\alpha}{1.28\beta}}\left(\frac{110}{n} + H(\beta) + \alpha H\left(\frac{1.28\beta}{\alpha}\right)\right) \\ &> \frac{1}{\beta \log_2 \frac{\alpha}{1.28\beta}}\left(H(\beta) + \alpha H\left(\frac{1.28\beta}{\alpha}\right)\right) \\ &\ge \frac{1}{1-\beta} + \frac{1.28\alpha}{\alpha - 1.28\beta}\,. \end{aligned}$$

Now that we established that $f(x)$ is convex, we upper bound the probability from (8) for $k = 15$ and $k = \beta n$ as follows. When $k = \beta n$,

$$2^{n\left(H(\beta)+\alpha H(\frac{1.28\beta}{\alpha}) - \beta c_n \log_2(\frac{\alpha}{1.28\beta})\right)} \le 2^{n(-110/n)} \le 2^{-110}$$

by the choice of $c_n \ge \frac{1}{\beta \log_2 \frac{\alpha}{1.28\beta}}\left(\frac{110}{n} + H(\beta) + \alpha H\left(\frac{1.28\beta}{\alpha}\right)\right)$. For the case of $k = 15$ (assuming that $k = 15 < \beta n$), (8) gives us the following upper bound on the probability of failure

$$2^{nH(15/n) + \alpha n H(\frac{19.2}{\alpha n}) - 15 c_n \log_2(\frac{\alpha n}{19.2})}\,.$$

Finally, we show that with high probability every set of size $k \le 14$ has at least $k+4$ neighbors. For $k \le c_n - 4$, this happens with certainty as every vertex has $c_n \ge k+4$ neighbors. For a set of size $c_n - 3 \le k \le \min(14, \beta n)$, the probability of having at most $k+3$ neighbors is bounded from above by $\frac{\binom{\alpha n}{k+3}\binom{k+3}{c_n}^k}{\binom{\alpha n}{c_n}^k}$. By the union bound over all sets of size k, we obtain the following upper bound on the probability of failure:

$$\frac{\binom{n}{k}\binom{\alpha n}{k+3}\binom{k+3}{c_n}^k}{\binom{\alpha n}{c_n}^k}\,,$$

which finishes the proof. $\square$

Claim 3. *For every $n \le 2^{30}$ and every $q \ge 2^{127}$, for every n and $k \le n$,*

$$\Pr[E^{(2)}_{n,k}] \ll 2^{-100}\,.$$

Proof. Let $x \in \mathbb{F}^n$ and $\|x\|_0 = k$. Let $K \subseteq [n]$ be the set of indices of non-zero elements of x. Let T_K be a random variable denoting the number of columns of $A \leftarrow \mathcal{M}_{n,\alpha n,c_n}$ with at least one non-zero element in the rows with indices from K.

Note that since the event $E^{(2)}_{n,k}$ is conditioned on the complement of $E^{(1)}_{n,k}$, we have that $T_k \ge b(k) = \max(k+4, 1.28k)$. We have that at least T_k coordinates of Ax are non-zero linear combinations of the non-zero coordinates of x with indices from K_x. Since all the coordinates of the linear combinations are uniform random non-zero elements of $\mathbb{F}$, each such linear combination equals zero with probability at most $1/q$, and all linear combinations equal zero with probability at most $1/q^{b(k)}$. Now taking the union bound over all $< \binom{n}{k}q^k$ vectors x of Hamming weight k, we have that $\Pr[E^{(2)}_{n,k}] < \binom{n}{k}q^{k-b(k)}$.

For $k \ge 15$, this implies that

$$\Pr[E^{(2)}_{n,k}] \le \binom{n}{k} q^{-0.28k} \le \left(\frac{en}{k}\right)^k \cdot q^{-0.28k} = \left(\frac{en}{kq^{0.28}}\right)^k \le 2^{-120}$$

for every $n \le 2^{30}$ and $q \ge 2^{127}$.

For $k \le 14$, this gives us that

$$\Pr[E^{(2)}_{n,k}] \le \binom{n}{k} q^{-4} \le \left(\frac{en}{k}\right)^k \cdot q^{-4} \le \left(\frac{e2^{30}}{14}\right)^{14} \cdot 2^{-127\cdot 4} \le 2^{-120}$$

for every $n \le 2^{30}$ and $q \ge 2^{127}$. $\square$

Claim 4. *Let $\mu = r - 1 - r\alpha$, $\nu = \beta + \alpha\beta + 0.03$, and*

$$d_n = \left\lceil \min\left(\left(2\beta + \frac{(r-1)+110/n}{\log_2 q}\right) n,\ \frac{r\alpha H(\beta/r) + \mu H(\nu/\mu) + 110/n}{\alpha\beta \log_2 \frac{\mu}{\nu}}\right)\right\rceil .$$

Then for every n and $\alpha\beta n \le k < \beta n$, if

$$\begin{aligned} \frac{(r-1)+110/n}{\log_2 q} &\le 0.03\,, \\ 2\beta + 0.03 &\le r - 1 - r\alpha\,, \\ \beta &\le \alpha r\,, \end{aligned}$$

then

$$\Pr[E^{(3)}_{n,k}] \ll 2^{-100}\,.$$

Proof. We need to show that with overwhelming probability every set of k coordinates of $z \in \mathbb{F}^{\alpha r n}$ expands into at least $b'(k) = \left(\beta + k/n + \frac{(r-1)+110/n}{\log_2 q}\right) n$ coordinates of $v \in \mathbb{F}^{(1-r-r\alpha)n}$. Note that

$$\Pr[E^{(3)}_{n,k}] = \Pr_{G \in \mathcal{G}_{\alpha r n, n(r-1-r\alpha), d_n}} [G \text{ contains a set of } k \text{ left vertices with fewer than } b'(k) \text{ neighbors}]\,.$$

First, if $d_n \ge \left(2\beta + \frac{(r-1)+110/n}{\log_2 q}\right) n$, then every left vertex has at least $d_n \ge b'(\beta n)$ neighbors. Then trivially every set of $k < \beta n$ left vertices has at least $b'(k)$ neighbors.

Now we assume that $d_n \ge \frac{r\alpha H(\beta/r)+\mu H(\nu/\mu)+110/n}{\alpha\beta \log_2 \frac{\mu}{\nu}}$. Let $k = \gamma n$ for $\alpha\beta \le \gamma < \beta$. A fixed set of size γn expands into a set of size less than $b'(k) = \left(\beta + k/n + \frac{(r-1)+110/n}{\log_2 q}\right) n \le (\beta + \gamma + 0.03)n$ with probability at most

$$\begin{aligned} \binom{(r-1-r\alpha)n}{b'(k)} \frac{\binom{b'(k)}{d_n}^{\gamma n}}{\binom{(r-1-r\alpha)n}{d_n}^{\gamma n}} &\le \binom{(r-1-r\alpha)n}{b'(k)} \left(\frac{b'(k)}{(r-1-r\alpha)n}\right)^{d_n \gamma n} \\ &\le 2^{(r-1-r\alpha)nH\left(\frac{\beta+\gamma+0.03}{r-1-r\alpha}\right)} \cdot \left(\frac{\beta+\gamma+0.03}{r-1-r\alpha}\right)^{d_n\gamma n} \\ &= 2^{(r-1-r\alpha)nH\left(\frac{\beta+\gamma+0.03}{r-1-r\alpha}\right) - d_n \gamma n \log_2 \frac{r-1-r\alpha}{\beta+\gamma+0.03}}\,, \end{aligned}$$

where in the second inequality we used $\beta + \gamma + 0.03 \le 2\beta + 0.03 \le r - 1 - r\alpha$.

Taking the union bound over all sets of size γn we have that

$$\begin{aligned} \Pr[E^{(3)}_{n,k}] &\le \binom{\alpha r n}{\gamma n} 2^{(r-1-r\alpha)nH\left(\frac{\beta+\gamma+0.03}{r-1-r\alpha}\right) - d_n\gamma n \log_2 \frac{r-1-r\alpha}{\beta+\gamma+0.03}} \\ &\le 2^{\alpha r n H(\frac{\gamma}{r\alpha}) + (r-1-r\alpha)nH(\frac{\beta+\gamma+0.03}{r-1-r\alpha}) - d_n\gamma n \log_2 \frac{r-1-r\alpha}{\beta+\gamma+0.03}}\,, \end{aligned}$$

where in the second inequality we used $\gamma < \beta \le \alpha r$. By an argument similar to the argument in Claim 2, it's sufficient to consider the case $\gamma = \alpha\beta$, in which case we have

$$\begin{aligned}
\Pr[E^{(3)}_{n,k}] &\le 2^{\alpha r n H(\frac{\beta}{r}) + (r-1-r\alpha) n H(\frac{\beta+\alpha\beta+0.03}{r-1-r\alpha}) - d_n \alpha\beta n \log_2 \frac{r-1-r\alpha}{\beta+\alpha\beta+0.03}} \\
&= 2^{\alpha r n H(\frac{\beta}{r}) + \mu n H(\nu/\mu) - d_n \alpha\beta n \log_2 \frac{\mu}{\nu}} \\
&\ll 2^{-100}
\end{aligned}$$

by the choice of $d_n \ge \frac{r\alpha H(\beta/r) + \mu H(\nu/\mu) + 110/n}{\alpha\beta \log_2 \frac{\mu}{\nu}}$. ☐

Claim 5. *For every n and $\alpha\beta n \le k \le \beta n$, if $2\beta + \frac{(r-1)+110/n}{\log_2 q} \le r - 1 - r\alpha$, then*

$$\Pr[E^{(4)}_{n,k}] \ll 2^{-100}\,.$$

Proof. Assuming that every set of size k expands into a set of size at least $b'(k) = \left(\beta + k/n + \frac{(r-1)+110/n}{\log_2 q}\right) n$, we need to show that with overwhelming probability every $z \in \mathbb{F}^{\alpha r n}$ of Hamming weight $\|z\|_0 = k$ is mapped to $v = B \cdot z \in \mathbb{F}^{(r-1-r\alpha)n}$ of Hamming weight $\|v\|_0 \ge \beta n$.

Fix a $z \in \mathbb{F}^{\alpha r n}$ of Hamming weight $\|z\|_0 = k$. Similarly to Claim 3, since the set of its non-zero coordinates expands into at least $b'(k)$ coordinates, we have that at least $b'(k)$ coordinates of v are non-zero linear combinations of the non-zero coordinates of z. Each such linear combination equals zero with probability $\le 1/q$, therefore, for a fixed z, the probability that $\|v\|_0 < \beta n$ is bounded from above by

$$\frac{\binom{b'(k)}{\ge b'(k)-\beta n}}{q^{b'(k)-\beta n}} \le \frac{\binom{(r-1-r\alpha)n}{\ge (r-1-r\alpha)n-\beta n}}{q^{b'(k)-\beta n}} \le \frac{2^{(r-1-r\alpha)n}}{q^{b'(k)-\beta n}}\,,$$

where we used $b'(k) \le \left(2\beta + \frac{(r-1)+110/n}{\log_2 q}\right) n \le (r-1-r\alpha)n$. Taking the union bound over all $z \in \mathbb{F}^{\alpha r n}$ of Hamming weight $\|z\|_0 = k$, we have that

$$\Pr[E^{(4)}_{n,k}] \le \binom{r\alpha n}{k} q^k \cdot \frac{2^{(r-1-r\alpha)n}}{q^{b'(k)-\beta n}} \le \frac{2^{r\alpha n + (r-1-r\alpha)n}}{q^{b'(k)-\beta n - k}} \le \frac{2^{(r-1)n}}{q^{\left(\frac{(r-1)+110/n}{\log_2 q}\right)n}} \ll 2^{-100}\,.$$

☐

A practical consideration: when to stop recursing. One can modify Algorithm 1 such that for all message sizes less than some parameter n_0, it simply performs naive (quadratic-time) Reed-Solomon encoding with an appropriate rate parameter (the reason we use naive quadratic-time Reed-Solomon encoding is to avoid the need to perform FFTs, and thereby avoid all restrictions on the field used). For messages of length $n \gg n_0^2$, the use of quadratic-time encoding applied just once to a message of size n_0 represents a low-order cost in the total encoding time. In our implementation, we set n_0 to roughly 30.

7 Linear-time commitments for sparse multilinear polynomials

To construct SNARKs from polynomial IOPs (§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.

A straw-man approach and a conceptual outline. The following is a natural approach to turning a commitment scheme for dense polynomials into one for sparse polynomials. To commit to a sparse ℓ-variate multilinear polynomial q with $k \ll 2^\ell$ non-zero coefficients, the committer commits to a representation of the non-zero coefficients q using a polynomial commitment for dense polynomials.

In more detail, associate each Lagrange basis polynomial with a bit-vector S in $\{0,1\}^\ell$, and let c_S be the coefficient of monomial S in q. Consider the vector $v \in \mathbb{F}^{k(1+\ell)}$ that simply lists all non-zero (coefficient, monomial) pairs, i.e., $v = \{(c_S, S) : c_S \neq 0\}$. The committer commits to the multilinear extension of the vector v using the dense polynomial commitment scheme, which takes $O(k\ell)$ time. Then when the verifier asks to evaluate the sparse polynomial q at an input r, one applies an interactive proof for the function that takes as input the vector v that describes q and outputs $q(r)$ (for example, one can apply the GKR protocol [GKR08] to the circuit that takes as input v and outputs $q(r) = \sum_S c_S \cdot \chi_S(r)$, where χ_S is the Lagrange basis polynomial corresponding to S). At the end of the interactive proof the verifier needs to evaluate the multilinear extension of v at a random point and can obtain this evaluation from the dense polynomial commitment scheme used to commit to this extension of v.

The problem with this polynomial commitment scheme is that the description of v that is used consists of $\Theta(k\ell)$ field elements, which is superlinear in the number k of non-zero coefficients of q, and hence the committer in this scheme runs in superlinear time. Spartan [Set20] identified a way to use memory-checking techniques allowing to use a description of q consisting of only $\Theta(k)$ field elements, thereby obtaining a linear-time committer. Details follow.

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 $r_x, r_y \in \mathbb{F}^{\log M}$.

$$D(r_x, r_y) = \sum_{(i,j)\in\{0,1\}^{\log M}\times\{0,1\}^{\log M}\,:\,D(i,j)\neq 0} D(i,j)\cdot\widetilde{eq}(i, r_x)\cdot\widetilde{eq}(j, r_y). \tag{9}$$

Claim 6. *Let* b *be the canonical injection from* $\{0,1\}^{\log M}$ *to* $\mathbb{F}$ *and* b^{-1} *be its inverse. 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* $r_x, r_y \in \mathbb{F}^s$*.*

$$D(r_x, r_y) = \sum_{k\in\{0,1\}^{\log N}} \mathsf{val}(k)\cdot\widetilde{eq}(\mathsf{b}^{-1}(\mathsf{row}(k)), r_x)\cdot\widetilde{eq}(\mathsf{b}^{-1}(\mathsf{col}(k)), r_y). \tag{10}$$

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

Proof. 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 (call it b) from $\{0,1\}^{\log M}$ to $\mathbb{F}$, we can view the first two entries in each of these tuples as elements of $\mathbb{F}$ (let b^{-1} denote its inverse). Furthermore, these tuples can be represented with three N-sized vectors $R, C, V \in \mathbb{F}^N$, 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 (9) and (10) 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 6. 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 the multilinear extensions of the functions mapping $k \in \{0,1\}^{\log N}$ to $\widetilde{eq}(\mathsf{b}^{-1}(\mathsf{row}(k)), r_x)$ and $\widetilde{eq}(\mathsf{b}^{-1}(\mathsf{col}(k)), r_y)$.
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.

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

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 $\{\widetilde{eq}(i, r_x) \colon i \in \{0,1\}^{\log M}\}$ and $\{\widetilde{eq}(j, r_y) \colon j \in \{0,1\}^{\log M}\}$ (note that all of these values can be computed in $O(M)$ 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.[14] $RS = \{\}$, and for an M-sized memory, WS is

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

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

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

Claim 7. *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 6, 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.[15]

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

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 7.

Claim 8. *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 $\forall k \in \{0,1\}^{\log N}$, $E_{\mathsf{rx}}(k) = \widetilde{eq}(\mathsf{b}^{-1}(\mathsf{row}(k)), r_x)$ 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 $\forall k \in \{0,1\}^{\log N}$, $E_{\mathsf{ry}}(k) = \widetilde{eq}(\mathsf{b}^{-1}(\mathsf{col}(k)), r_y)$ 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 7. □

There is no direct way to prove that the checks on sets in Claim 8 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 9 ([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 $r_z \in \mathbb{F}^{\log N}$ chosen at random by the verifier over the course of the sum-check protocol:

[15] 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.

- $\mathsf{val}(r_z) \stackrel{?}{=} v_{\mathsf{val}}$; 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. $\mathcal{V} \rightarrow \mathcal{P}$: $\tau, \gamma \in_R \mathbb{F}$.
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 8 and $\mathcal{H}$ is defined in Claim 9 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. $\mathcal{V} \rightarrow \mathcal{P}$: $\tau', \gamma' \in_R \mathbb{F}$.
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 8 and $\mathcal{H}$ is defined in Claim 9 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.[16]

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.

8 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.

[16] 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 $O(\log N)$, though it requires the prover to send an additional polynomial of size $O(N)$.

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 18 by composing the SNARK of Theorem 8 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 8 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 8. 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.

9 Evaluation

In this section, we evaluate the performance of two polynomial commitment schemes and two SNARKs based on these schemes. Specifically, we evaluate two instantiations of the polynomial commitment scheme of Section 5: Ligero-PC, which uses the Reed-Solomon code (this scheme is implicit in Ligero [AHIV17]), and Brakedown-PC, which uses the new linear-time error-correcting code described in Section 6.

We answer the following questions:

1. How do the costs of Brakedown-PC and Ligero-PC compare? (§9.2)
2. How do the costs of Brakedown and Shockwave compare with prior SNARKs? (§9.3)

9.1 Implementation

We implement Ligero-PC and Brakedown-PC in ≈3500 lines of Rust. This includes an implementation of the polynomial commitment of Section 5 that is generic over fields, error-correcting codes, and hash functions; implementations of the Reed-Solomon code and our new linear-time code; and a fast parallelized FFT.

We also integrate these polynomial commitments with the open-source implementation of Spartan [liba], yielding a SNARK library for R1CS. This took less than 100 lines of glue Rust code.

All experiments reported in this section use the BLAKE3 hash function [OANWO20]. Because Ligero-PC needs to perform FFTs, our experiments use fields of characteristic p such that $p-1$ is divisible by 2^{40}, which ensures that reasonably large FFTs are possible in the field; we choose p at random.

As discussed in Section 1.1, our SNARK implementations are not currently zero-knowledge. We leave the completion of a zero-knowledge implementation to near-term future work.

9.2 Polynomial commitment microbenchmarks

We begin the experimental evaluation by examining the costs of the polynomial commitment schemes developed in this work. We report the prover's time to commit to and open polynomials, the verifier's time to check an opening, and the communication cost, for multilinear polynomials having 13 to 29 variables (i.e., 2^{13} to 2^{29} monomials) and univariate polynomials of degree between 2^{13} to 2^{29} (the implementations use exactly the same code for the univariate and multilinear cases).

Parameters of the error-correcting codes. We instantiate Brakedown-PC with the parameters given on the third line of Table 3. For Ligero-PC, the rate of the Reed-Solomon code used can be viewed as a knob that allows one to tradeoff between prover runtime and verification costs. Roughly speaking, the smaller the rate, the slower the prover, but the smaller the proof size and verification costs. To explore this tradeoff, we test Ligero-PC with three different code rates: $38/39$, $1/2$, and $1/4$. We choose $38/39$ because it gives proof sizes roughly matching Brakedown-PC, $1/2$ because it gives smaller proofs than Brakedown-PC at roughly comparable prover cost, and $1/4$ because it gives even smaller proofs at the cost of greater prover computation.

An important subtlety regarding rate parameter $^{38}/_{39}$ is that it *cannot* be used in the Ligero-PC scheme when applied to multilinear polynomials (as required by Shockwave, our SNARK for R1CS that uses Ligero-PC; §9.3). Ligero-PC only supports multilinear polynomials when instantiated with rate ρ such that ρ^{-1} is a power of two. This is a consequence of the fact that Ligero-PC's FFT requires power-of-two–length codewords, and multilinear polynomial evaluation can only be decomposed into tensor products with power-of-two–sized tensors (see Section 5). Since ρ is the ratio of one tensor's size to codeword length, ρ must be a power of two. Thus, $\rho = 1/2$ is the highest rate Ligero-PC supports for multilinear polynomials.

Other parameters. We set parameters of the commitment schemes to obtain 128 bits of security (the one exception to this is that the randomized code generation procedure in Brakedown-PC is configured to have at most a 2^{-100} probability of failing to satisfy the requisite distance properties according to the analysis of Section 6; this is acceptable because code generation occurs publicly, once for each instance size). To achieve this, we set the number of columns opened in Brakedown-PC to 6593, in Ligero-PC-$^{38}/_{39}$ to 7054, in Ligero-PC-$1/2$ to 309, and in Ligero-PC-$1/4$ to 189.

Setup and method. Our testbed for this section is an Azure Standard F64s_v2 virtual machine (64 Intel Xeon Platinum 8272CL vCPUs, 128 GiB memory) with Ubuntu 20.10. We measure single-threaded and 64-threaded speed for committing, opening, and verifying; and we also report communication cost. For each experiment, we run the operation 10 times and report the average; in all cases, variation is negligible.

Results. Figure 4 reports the results. The dominant cost for the prover is committing to the polynomial. For large enough polynomials, Brakedown-PC's commitment computation is as fast or faster than Ligero-PC-$1/2$'s, and roughly 2–3× faster than Ligero-PC-$1/4$'s. Computing commitments in Ligero-PC-$^{38}/_{39}$ is faster than in Brakedown-PC, but Ligero-PC-$^{38}/_{39}$ does not support multilinear polynomials (see discussion above).

Ligero-PC-$1/2$ and Ligero-PC-$1/4$ have lower verification cost than the other two schemes, though this advantage shrinks as instances grow. In terms of communication complexity, Brakedown-PC and Ligero-PC-$^{38}/_{39}$ have (by design) nearly the same communication cost. Ligero-PC-$1/2$ has ≈5–15× less communication than Brakedown-PC, and Ligero-PC-$1/4$ has ≈6–21× less than Brakedown-PC. As with verification time, the proof size advantage of Ligero-PC-$1/2$ and Ligero-PC-$1/4$ over Brakedown-PC shrinks as the instance size grows, with a 5×-6× difference for large instances.

The proof size and verification time comparison can roughly be be explained as follows. The total communication cost of each commitment scheme is $\Theta(\sqrt{N \cdot \ell})$ where ℓ is the number of columns that the verifier must open. This number ℓ is about 21× bigger for Brakedown-PC than Ligero-PC-$1/2$ and about 35× bigger for Brakedown-PC than Ligero-PC-$1/4$. As N grows large compared to ℓ, Brakedown-PC's proof size approaches roughly $\sqrt{21} \approx 4.6$ times larger than Ligero-PC-$1/2$ and roughly $\sqrt{35} \approx 5.9$ times larger than Ligero-PC-$1/4$.

In all cases, hashing is a low-order cost for the provers. The same is true for multi-threaded verifiers (because checking Merkle paths is embarassingly parallel) and for single-threaded Ligero-PC-$1/4$ and Ligero-PC-$1/2$. In Ligero-PC-$^{38}/_{39}$ and Brakedown-PC, however, hashing is 30–50% of the verifier's single-threaded cost (because the prover opens many columns). Our sparse matrix library does not parallelize as well as our FFT library; optimizing this should improve multi-threaded Brakedown-PC performance relative to Ligero-PC.

9.3 Evaluation of Brakedown and Shockwave SNARKs

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

	2^{13}	2^{15}	2^{17}	2^{19}	2^{21}	2^{23}	2^{25}	2^{27}	2^{29}
1 thread									
Commit (seconds)									
Brakedown-PC	0.013	0.038	0.123	0.491	2.14	8.91	36.0	150	605
Ligero-PC-38/39	0.007	0.022	0.076	0.293	1.23	5.19	22.1	92.6	405
Ligero-PC-1/2	0.007	0.028	0.115	0.492	2.08	8.75	39.9	169	717
Ligero-PC-1/4	0.015	0.058	0.244	1.04	4.46	19.7	83.9	356	1590
Open (seconds)									
Brakedown-PC	0.022	0.028	0.048	0.113	0.271	0.935	3.21	13.0	48.6
Ligero-PC-38/39	0.019	0.027	0.050	0.102	0.291	0.911	3.25	12.3	48.1
Ligero-PC-1/2	0.002	0.005	0.014	0.051	0.214	0.799	3.11	12.4	50.0
Ligero-PC-1/4	0.002	0.004	0.014	0.052	0.209	0.793	3.13	12.5	51.6
Verify (seconds)									
Brakedown-PC	0.025	0.035	0.056	0.148	0.298	0.613	0.703	2.56	2.96
Ligero-PC-38/39	0.021	0.030	0.046	0.077	0.143	0.280	0.559	1.11	2.34
Ligero-PC-1/2	0.003	0.006	0.011	0.021	0.044	0.093	0.196	0.402	0.846
Ligero-PC-1/4	0.004	0.008	0.017	0.036	0.077	0.160	0.338	0.714	1.57
64 threads									
Commit (seconds)									
Brakedown-PC	0.012	0.018	0.024	0.074	0.234	0.682	2.24	10.7	38.8
Ligero-PC-38/39	0.014	0.015	0.034	0.093	0.254	0.470	2.08	7.53	28.2
Ligero-PC-1/2	0.009	0.022	0.062	0.174	0.367	1.04	3.21	11.5	45.5
Ligero-PC-1/4	0.015	0.034	0.095	0.280	0.646	1.74	5.73	21.6	94.6
Open (seconds)									
Brakedown-PC	0.025	0.028	0.039	0.067	0.105	0.189	0.281	0.931	2.05
Ligero-PC-38/39	0.026	0.029	0.038	0.056	0.095	0.161	0.332	0.792	2.13
Ligero-PC-1/2	0.005	0.005	0.006	0.010	0.022	0.048	0.146	0.449	1.54
Ligero-PC-1/4	0.004	0.004	0.006	0.009	0.019	0.049	0.140	0.421	1.51
Verify (seconds)									
Brakedown-PC	0.010	0.017	0.031	0.120	0.270	0.558	0.551	2.37	2.40
Ligero-PC-38/39	0.008	0.012	0.020	0.031	0.051	0.088	0.164	0.325	0.647
Ligero-PC-1/2	0.012	0.006	0.010	0.014	0.022	0.035	0.058	0.106	0.201
Ligero-PC-1/4	0.006	0.009	0.013	0.018	0.027	0.043	0.075	0.136	0.278
Communication (kiB)									
Brakedown-PC	4299	5198	6739	10010	15797	27112	49157	93767	181948
Ligero-PC-38/39	4225	5172	6799	9785	15487	26837	49270	93651	182359
Ligero-PC-1/2	279	432	727	1304	2446	4718	9250	18302	36394
Ligero-PC-1/4	203	321	551	1004	1901	3689	7256	14383	28631

Figure 4: Microbenchmark results (§9.2). Brakedown-PC is instantiated with the parameters given on the third line of Table 3; Ligero-PC-38/39, Ligero-PC-1/2, and Ligero-PC-1/4 are instantiated with Reed-Solomon rates of 38/39, 1/2, and 1/4, respectively.

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 ($N \geq 2^{13}$ for Shockwave and $N \geq 2^{18}$ for Brakedown).

Unless we specify otherwise, we run our experiments in this section 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.

9.3.1 Performance of Shockwave's and Brakedown'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.[17] 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.

Later, we provide a rough 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].

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]. Unless we specify otherwise, we use 256-bit prime fields. Hyrax uses `curve25519` and SpartanNIZK uses `ristretto255` [ris, Ham15] for a group where the discrete-log problem is hard, so R1CS instances are defined over the scalar field of these curves. For Aurora and Ligero, we use the 256-bit prime field option in `libiop`. Finally, our schemes use the scalar field of `BLS12-381`, which supports FFTs (Brakedown does not need FFTs but Shockwave does). However, we note that none of these implementations leverages the specifics of the prime field to speed up scalar arithmetic.

We first experiment with Brakedown and Shockwave and their baselines with varying R1CS instance sizes up to 2^{20} constraints defined over a 256-bit prime field. Figures 5, 6, and 7 depict respectively the prover time, the proof sizes, and the verifier time from these experiments. We find the following from these experiments.

- Brakedown's and Shockwave's provers are the faster than prior work at all instance sizes we measure. Compared to baselines that are plausibly post-quantum secure (Ligero and Aurora), Brakedown's and Shockwave's provers are over an order of magnitude faster.
- Brakedown's proof size is larger than other depicted systems except for Ligero. Still, its proofs are substantially smaller than the size of the NP-witness for instance sizes $N \geq 2^{18}$. Shockwave provides shorter proofs than Brakedown as well as prior post-quantum secure baselines (Ligero and Aurora[18]). Shockwave's proof sizes are smaller than that of the NP-witness for instance sizes $N \geq 2^{13}$.
- Despite their larger proofs, Brakedown's and Shockwave's verifiers are competitive with those of SpartanNIZK and Lakonia, and is well over an order of magnitude faster than the plausibly post-quantum secure 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]	0.1	0.2	0.4	0.8	1.6	2	4	8	17	35	69
Hyrax [WTS$^+$18]	1	1.7	2.8	5	9	18	36	61	117	244	486
Aurora* [BCR$^+$19]	0.5	0.8	1.6	3.2	6.5	13.3	27	56	116	236	485
SpartanNIZK [Set20]	0.01	0.02	0.04	0.07	0.14	0.25	0.5	0.8	1.7	3	6
Lakonia [SL20]	0.2	0.2	0.4	0.5	0.8	1	2	3	6	10	19
Brakedown ($\frac{1}{2}$-SNARK)*	0.008	0.012	0.02	0.04	0.07	0.12	0.2	0.4	0.8	1.5	3.1
Shockwave ($\frac{1}{2}$-SNARK)*	0.005	0.008	0.02	0.03	0.06	0.1	0.3	0.5	1	2	4.1

* Provides plausible post-quantum security

Figure 5: Prover time (in seconds) for varying R1CS instance sizes under different schemes.

[17] We do not compare with Ligero++ [BFH$^+$20] since its source code is not public. Broadly speaking, Ligero++ has shorter proofs than Ligero at the cost of a slower prover, so its prover will be significantly slower than both Brakedown and Shockwave.

[18] Note that Aurora has asymptotically shorter proofs than Shockwave, and hence the proof size comparison would "cross over" at larger instance sizes.

	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
Brakedown ($\frac{1}{2}$-SNARK)⋆	1,279	1,597	1,974	2,200	2,710	3,165	3,926	4,824	6,122	7,899	10,230
Shockwave ($\frac{1}{2}$-SNARK)⋆	72	95	122	160	210	284	386	523	721	990	1,384

⋆ Provides plausible post-quantum security

Figure 6: Proof sizes (KBs) for Shockwave 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
Brakedown ($\frac{1}{2}$-SNARK)⋆	16	20	28	31	49	60	101	130	231	339	660
Shockwave ($\frac{1}{2}$-SNARK)⋆	2	3	4	8	11	21	35	69	121	241	466

⋆ Provides plausible post-quantum security

Figure 7: Verifier time (in ms) under different schemes.

Performance for larger instance sizes. To demonstrate Brakedown's and Shockwave's scalability to larger instance sizes, we experiment with them 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 8, 9, and 10 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 Brakedown and Shockwave 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 8, 9, and 10).

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 $\exp(-\lambda)$.[19] However, Brakedown and Shockwave are 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 A 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. Note, moreover, that Wolverine and Mac'n'Cheese are targeted at circuits rather than R1CS (R1CS can simulate circuits with addition and multiplication gates of fan-in two, with one R1CS constraint per multiplication gate).

- Both baselines require interaction between the verifier and the prover, so unlike Brakedown and Shockwave, they do not produce proofs that any verifier can verify. In other words, both baselines do

[19]To achieve *constant* soundness error, we in fact only need $|\mathbb{F}| \gg N$ (see Lemma 1 in Section 5 and Claim 10 in Appendix A).

	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
Brakedown ($\frac{1}{2}$-SNARK)* (256-bit field)	3	6	13	25	52	103	210
Shockwave ($\frac{1}{2}$-SNARK)* (256-bit field)	4	9	17	36	72	148	295
Brakedown ($\frac{1}{2}$-SNARK)* (128-bit field)	1	2	5	10	20	40	81
Shockwave ($\frac{1}{2}$-SNARK)* (128-bit field)	2	4	8	16	32	65	136

* Provides plausible post-quantum security

Figure 8: Prover time (in seconds) for varying R1CS instance sizes under Shockwave ($\frac{1}{2}$-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
Brakedown ($\frac{1}{2}$-SNARK)* (256-bit field)	10,230	13,737	18,145	25,068	33,685	47,385	64,399
Shockwave ($\frac{1}{2}$-SNARK)* (256-bit field)	1,384	1,914	2,695	3,751	5,309	7,415	10,522
Brakedown ($\frac{1}{2}$-SNARK)* (128-bit field)	6,644	8,164	11,003	13,983	19,457	25,280	36,021
Shockwave ($\frac{1}{2}$-SNARK)* (128-bit field)	730	1,059	1,395	2,048	2,711	4,012	5,332

* Provides plausible post-quantum security

Figure 9: Proof sizes (in KBs) under Shockwave 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
Brakedown ($\frac{1}{2}$-SNARK)* (256-bit field)	0.7	1.1	2.1	3.8	8	14	29
Shockwave ($\frac{1}{2}$-SNARK)* (256-bit field)	0.5	0.9	1.7	3.5	7	14	28
Brakedown ($\frac{1}{2}$-SNARK)* (128-bit field)	0.3	0.4	0.8	1.4	3	5	10
Shockwave ($\frac{1}{2}$-SNARK)* (128-bit field)	0.2	0.3	0.6	1.2	2	5	9

* Provides plausible post-quantum security

Figure 10: Verifier time (in seconds) for varying R1CS instance sizes under different schemes.

not produce publicly-verifiable proofs. The verifier's runtime is asymptotically the same as Brakedown's and Shockwave's verifiers ($\frac{1}{2}$-SNARK variants), but Brakedown's and Shockwave's verifier appear to be concretely far cheaper.

- Proof sizes are $O_\lambda(N)$ for an N-sized instance for both baselines, whereas Brakedown's and Shockwave's proofs are $O_\lambda(\sqrt{N})$. Concretely, Brakedown's and Shockwave's proofs are orders of magnitude shorter than both baselines.
- Asymptotically, the prover in both baselines uses memory proportional to that needed to produce the witness and evaluate the circuit non-cryptographically. Brakedown and Shockwave do not have this property, though for worst-case circuits, Brakedown, Shockwave, 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 Brakedown and Shockwave.

9.3.2 Performance of Brakedown's and Shockwave'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 11, 12, 13, 14 depict respectively the prover time, the proof size, the verifier time, and the verifier's preprocessing time for varying R1CS instance sizes for our schemes and their baselines. We find the following

	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
Brakedown*	0.06	0.1	0.2	0.3	0.6	1	2	4	8	17	33
Shockwave*	0.04	0.08	0.16	0.3	0.6	1.3	2.7	5	11	21	45

* Provides plausible post-quantum security

Figure 11: 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
Brakedown*	6,020	7,758	9,258	10,773	13,018	15,164	19,269	23,551	31,173	39,261	53,846
Shockwave*	438	572	695	940	1,182	1,631	2,096	2,969	3,891	5,606	7,382

* Provides plausible post-quantum security

Figure 12: Proof sizes in KBs.

from our experimental results.

- Brakedown achieves the fastest prover at instance sizes we measure. Shockwave's prover is slower than Brakedown's, both asymptotically and concretely, but Shockwave's prover is still over an order of magnitude faster than prior plausibly post-quantum secure SNARKs (namely Fractal [COS20]).
- Brakedown and Shockwave have the largest proof sizes amongst the displayed proof systems, but for large enough R1CS instances their proof sizes are sublinear in the size of the NP-witness ($N > 2^{16}$ for Shockwave and $N \geq 2^{22}$ for Brakedown).
- Brakedown's verifier is slower than Shockwave and most other schemes, particularly Xiphos [SL20] which is specifically designed for achieving a fast verifier. However, Shockwave's verifier is competitive with prior plausibly post-quantum secure SNARKs.
- Brakedown's and Shockwave's preprocessing costs for the verifier are competitive with those of prior high-speed SNARKs such as Spartan [Set20] and Xiphos [SL20], and an order of magnitude faster than the prior post-quantum secure SNARK (Fractal).

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

For these larger-scale experiments, we use an Azure Standard F64s_v2 VM which has 64 vCPUs and 128 GB memory. Figures 15, 16, and 17 depict results from these larger-scale experiments. Our findings from these experiments are similar to results from the smaller-scale results.

10 Additional related work and comparisons

On cryptographic assumptions and detailed comparison to [BCL20]. Recall from Section 1 that Bootle, Chiesa, and Liu [BCL20] give a zero-knowledge IOP for R1CS with linear-time prover and polylogarithmic time verifier. Standard transformations can then translate this IOP into a zero-knowledge SNARG that is unconditionally secure in the random oracle model.

	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
Brakedown*	111	133	190	223	323	386	576	708	1,103	1,379	2,219
Shockwave*	14	21	25	40	50	82	106	175	223	377	481

* Provides plausible post-quantum security

Figure 13: Verifier's performance (in ms).

	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	0.1	0.2	0.3	0.6	1	2	3	7	10	20
SuperSonic [BFS20]	35	64	117	216	400	747	1,400	2,600	4,900	9,400	17,900
Fractal* [COS20]	0.3	0.6	1.2	2.5	5.4	11.5	24	51	107	227	–
Kopis [SL20]	0.6	0.7	1.3	1.5	2.7	3.4	6.2	8.6	16	24	46
Xiphos [SL20]	0.8	1	2	3	3	6	7	13	18	32	49
Brakedown*	0.04	0.06	0.12	0.2	0.4	0.7	1.4	3	6	11	22
Shockwave*	0.04	0.08	0.15	0.3	0.6	1.3	2.7	5.5	11	24	47

* Provides plausible post-quantum security

Figure 14: Verifier's preprocessing time (i.e., encoder's time) in seconds for varying R1CS instance sizes under different schemes.

Our implemented SNARKs, Shockwave and Brakedown, satisfy the same type of security property as [BCL20, BCG20], in the sense that we give an IOP (of knowledge), and this can be transformed into a SNARK that is unconditionally secure in the random oracle model. However, our other theoretical 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.

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

Additional related work. Other than [BCG20, BCL20], the following works have come closest to the goals set forth in this paper: (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. All these schemes combine variants of 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 where N is the size of the circuit,[20] 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.

[20]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.

	2^{20}	2^{21}	2^{22}	2^{23}
Spartan	44	89	175	350
Brakedown⋆	33	67	136	274
Shockwave⋆	45	91	186	390

⋆ Provides plausible post-quantum security

Figure 15: Prover time (in seconds) for varying R1CS instance sizes under Brakedown, Shockwave, and Spartan.

	2^{20}	2^{21}	2^{22}	2^{23}
Spartan	132	170	208	345
Brakedown⋆	53,895	69,595	98,182	129,298
Shockwave⋆	7,440	10,845	14,453	21,259

⋆ Provides plausible post-quantum security

Figure 16: Proof sizes (in KBs) for varying R1CS instance sizes under Brakedown, Shockwave, and Spartan.

In contrast, Spartan and Xiphos apply to arbitrary R1CS (over discrete-logarithm-friendly fields) 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.[21] 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.[22] 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 for large enough instance sizes.

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.

Beyond the above works, 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]. Moreover, as mentioned in Section 1, all prior succinct argument implementations have placed restrictions on the field used, e.g., discrete-logarithm friendliness, FFT-friendliness, or the need for one or more additive or multiplicative subgroups of a specified size.

Finally, several recent works [GMO16, CDG+17, WYKW20, BMRS20, DIO20] do achieve a linear-time prover, but they do not achieve sub-linear verification costs nor proof sizes; many of these works also do not achieve non-interactivity nor publicly-verifiable proofs.

11 Conclusion and future directions

In this work, we have given new, plausibly post-quantum arguments of knowledge for expressive NP-complete problems such as R1CS. Our arguments achieve asymptotically optimal prover time, sublinear proof size, and state of the art concrete efficiency. There are a number of avenues to improve or extend our arguments, without radical departures from our paradigm.

[21] 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 18.

[22] 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.

	2^{20}	2^{21}	2^{22}	2^{23}
Spartan	96	126	180	240
Brakedown⋆	2,226	2,765	4,506	5,695
Shockwave⋆	483	803	989	1,649

⋆ Provides plausible post-quantum security

Figure 17: Verifier's time (in ms) for varying R1CS instance sizes under Brakedown, Shockwave, and Spartan.

An obvious direction is to further improve the practicality of our linear-time encodable codes, i.e., achieving better distance properties without sacrificing encoding time. A potentially more substantive observation is that, in order to achieve a sound argument, we may not actually need the code to have large minimum distance; it may suffice for the code to have large *computational minimum distance*. By this, we mean that, while pairs of close codewords might exist, they cannot be found by any efficient algorithm. Are there more efficient codes than ours that can be shown to have large computational minimum distance under standard intractability assumptions?

The proof length of our polynomial commitment scheme would benefit slightly from improved analyses of proximity gaps for general linear codes. E.g., extending the analysis of Appendix A from Reed-Solomon codes to general linear codes would improve the proof length of Brakedown by perhaps 15%. Alternatively, such an improved analysis could be leveraged to marginally improve prover time while keeping proof size constant.

It would be interesting to implement versions of the polynomial commitment scheme described in Section 5 (and also implicit in [RR20]) with proof length $\Theta_\lambda(N^{1/3})$ or $\Theta_\lambda(N^{1/4})$, where N is the size of the polynomial to be committed. While these protocols improve the proof size's dependence on N, the hidden constants and the dependence on λ worsen significantly. In addition, the prover's runtime increases by a factor of at least the inverse of the rate of the error-correcting code used. Accordingly, it is not clear that these protocols would concretely improve over the prover-time-vs-proof-size tradeoffs already obtained in this work by adjusting the rate of code used.

A final direction is to implement the composition of Shockwave or Brakedown with another SNARK, perhaps one of very small proof length, in an effort to achieve a best-of-both-worlds SNARK with a linear-time prover and tiny proofs. Doing so naively would require expressing the Shockwave or Brakedown verifier as an R1CS instance. Preliminary estimates suggest that for realistic input sizes (say, $\leq 2^{40}$) this might result in an R1CS of size perhaps 2^{30}, dominated by the need to represent cryptographic hash operations via the R1CS. This is not out of the realm of feasibility, even for popular trusted-setup SNARKs with short proofs that are difficult to scale owing to large (superlinear) prover complexity and the need for a trusted setup [WZC+18]. But it is nonetheless larger than ideal (e.g., if one wishes to compose with existing trusted-setup SNARKs with short proofs, without running a new trusted-setup ceremony, one is limited, e.g, to instances of size roughly 2^{19} [Zca18, GMN21]). It may be possible to reduce the "R1CS-representation-size" of the Shockwave or Brakedown verifiers by reducing proof length with better error-correcting codes or as per the previous paragraph.

Lastly, on the theory side, it remains open to give a SNARK with a linear-time prover that operates even over very small fields (i.e., of size sub-exponential in the desired security parameter λ).

Acknowledgments

We thank Jonathan Bootle, Alessandro Chiesa, and Jens Groth for answering our questions on [BCG20], 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.

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.

[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.

[BCG20] Jonathan Bootle, Alessandro Chiesa, and Jens Groth. Linear-time arguments with sub-linear verification from tensor codes. In *Theory of Cryptography Conference (TCC)*, 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.

[BCKL21] Eli Ben-Sasson, Dan Carmon, Swastik Kopparty, and David Levit. Elliptic Curve Fast Fourier Transform (ECFFT) part I: Fast polynomial algorithms over all finite fields. Electronic Colloquium on Computational Complexity, Report 2021/103, 2021.

[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.

[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.

[BFH+20] Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: a new optimized sublinear iop. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 2025–2038, 2020.

[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.

[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.

[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.

[GDP73] S. I. Gelfand, R. L. Dobrushin, and M. S. Pinsker. On the complexity of coding. In *International Symposium on Information Theory (ISIT)*, pages 177–184, 1973.

[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.

[GMN21] Nicolas Gailly, Mary Maller, and Anca Nitulescu. SnarkPack: Practical SNARK aggregation. Cryptology ePrint Archive, Report 2021/529, 2021.

[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.

[Ham15] Mike Hamburg. Decaf: Eliminating cofactors through point compression. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2015.

[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.

[LSTW21] Jonathan Lee, Srinath Setty, Justin Thaler, and Riad Wahby. Linear-time and post-quantum zero-knowledge SNARKs for R1CS. Cryptology ePrint Archive, Report 2021/030, 2021.

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

[OANWO20] Jack O'Connor, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O'Hearn. BLAKE3: One function, fast everywhere. `https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf`, February 2020.

[OBW20] Alex Ozdemir, Fraser Brown, and Riad S. Wahby. Unifying compilers for SNARKs, SMT, and more. Cryptology ePrint Archive, Report 2020/1586, 2020.

[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.

[ris] The Ristretto group. `https://ristretto.group/`.

[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.

[WZC+18] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. DIZK: A distributed zero-knowledge proof system. In *Proceedings of the USENIX Security Symposium*, 2018.

[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.

[Zca18] Zcash. Zcash powers of taus ceremony attestation, 2018. `https://github.com/ZcashFoundation/powersoftau-attestations`.

[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 Tighter Analysis for the Reed-Solomon Code

By invoking a state of the art analysis in place of Claim 1, 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 3.1].

Claim 10. *(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 3. *Let* ρ *and* $\epsilon = N/|\mathbb{F}|$ *be as in Claim 10 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*

$$E \coloneqq |\{j \in [N] \colon \exists i \in [m] \text{ such that } c_{i,j} \neq \hat{u}_{i,j}\}| \leq \delta N. \tag{11}$$

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 (11) 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 10, 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 3. 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 (11). $|A \cup E| \leq |A| + |E| \leq \rho \cdot N + \delta N = (\rho + \delta)N$, 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 $1 - (\rho+\delta)^\ell$. Setting $\delta = \frac{1-\rho}{2}$, we obtain the following theorem.

Theorem 7. *Consider the polynomial commitment scheme described in Section 5 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.*

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 in the random oracle model that convinces a verifier, one can efficiently extract the IOP proof strings from the prover. Our analysis of the testing phase of the polynomial commitment scheme (Lemma 3) guarantees that each row of the prover's proof string in the commitment phase has relative Hamming distance at most $\delta = \frac{1-\rho}{2}$ from some codeword. Hence, row-by-row decoding provides the coefficients of the multilinear polynomial that the prover is bound to (decoding of the Reed-Solomon code can be done in polynomial time via, e.g., the Berlekamp-Welch algorithm).

An optimization over small fields. The probability $N/|\mathbb{F}| + (\frac{1+\rho}{2})^\ell$ appearing in Theorem 7 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 7 is easily extended to show that Theorem 7 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).

B Asymptotic Efficiency of Prior SNARKs

Figure 18 depicts the asymptotic efficiency of zkSNARKs from this work and prior works.

	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
[BCG20]	$O(N)$ $\mathbb{F}$-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 18: Asymptotic efficiency of zkSNARKs from this work and prior works (we borrow style from [Set20, SL20, BCG20]). 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. [BCG20, 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 8. 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. [BCG20, BCL20] require assuming the hardness of certain lattice problems, which are not listed in the "assumptions" column for brevity (§1).