

Succinct Arguments with (almost) Minimal Time and Space Overheads from Symmetric-Key Primitives

Alexander R. Block¹, Caleb Geren², and Pratik Soni²

¹University of Illinois at Chicago. alexander.r.block@gmail.com

²University of Utah. c.geren@utah.edu, psoni@cs.utah.edu

Abstract

We construct public-coin, succinct, zero-knowledge interactive arguments for **NP** relations decided by time- T , space- S random-access machines. Assuming the existence of collision-resistant hash functions, our arguments give a frontier of tradeoffs between prover time and space, and proof size. We follow the streaming PIOP+PCS framework of Block et al. (TCC 2020, CRYPTO 2021), reducing the problem of constructing time- and space-efficient arguments to time- and space-efficient streaming multilinear polynomial commitment schemes (MLPCS).

We develop two code-based streaming MLPCS inspired by Brakedown (CRYPTO 2023) and BaseFold (CRYPTO 2024), implementing each scheme when additionally given succinct access to the underlying encoding matrices. We instantiate this framework assuming row-sparse, column-sparse, and Reed-Solomon encoders, then apply a generic PIOP + MLPCS compiler to obtain our (zero-knowledge) time- and space-efficient arguments. Finally, as all our protocols are public-coin, applying Fiat-Shamir yields time- and space-efficient (zk)SNARKs from symmetric-key primitives (i.e., collision-resistant hashing).

Our Brakedown-based constructions have $O(\log T)$ rounds and $\tilde{O}(\sqrt{T})$ -size proofs and verification time, with two complementary prover profiles: (i) $\tilde{O}(S)$ space (which is optimal) and $\tilde{O}(T^{3/2})$ time, and (ii) $\tilde{O}(T)$ time (nearly optimal) and $\tilde{O}(S + \sqrt{T})$ space. Our BaseFold-based constructions achieve $\text{polylog}(T)$ proof size and verification time with $O(\log T)$ rounds. We again obtain two complementary prover profiles: (i) an encoder parameterized construction with $\tilde{O}(T + T_{\text{Enc}})$ time and $\tilde{O}(S + S_{\text{Enc}})$ space, where $T_{\text{Enc}}, S_{\text{Enc}}$ are the running time and space of a streaming log-space encoder for a suitable linear code, and (ii) an unconditional sub-quadratic instantiation $\tilde{O}(T^{3/2})$ time and $\tilde{O}(S + \sqrt{T})$ space.

1 Introduction

Zero-knowledge proofs (ZKPs) allow a computationally powerful prover to convince a computationally weak verifier of the truth of an **NP** statement while revealing nothing beyond its validity [39]. In their most succinct form, (Zero-Knowledge) Succinct Non-interactive **AR**guments of **K**nowledge ((zk)SNARKs) admit proofs that are much shorter than the underlying **NP** witness (e.g., sublinear and ideally polylogarithmic), while verification can be far cheaper than directly verifying the underlying **NP** relation given the **NP** witness. These features make (zk)SNARKs attractive in settings that demand fast, public verification, such as scaling blockchains via zkrollups [63] and verifiable delegation of computation [38]; this, in turn, has fueled substantial effort toward scaling ZKPs, with focus on optimizing proof generation time, proof size, and verification time.

Much of the work on SNARKs has focused on optimizing the prover *running time*, culminating in constructions with essentially *linear-time* (i.e., optimal) proving [20,22,24,37,40,54]. However, optimizing prover *space (memory) usage* has received comparatively less attention. A central difficulty, emphasized in prior work, is that even for the lowest-overhead SNARKs, certifying a time- T computation can require the prover to use $\Omega(T)$ space, even when the computation itself uses only space $S \ll T$ [4,12,13,42]. As observed in these works, typical computations exhibit a wide gap between time and space (e.g., $S \ll T$), so prover memory which scales with T quickly becomes the dominant practical bottleneck; e.g., a $T = 2^{40}$ time computation would need *at least* 137GB of memory. Moreover,

modern memory is hierarchical (cache, RAM, Disk), and large memory footprints translate into cache misses, yielding substantial slowdowns on real-world systems.

Complexity-Preserving SNARKs. This memory bottleneck has sparked interest in *complexity-preserving* ZKPs for **NP** relations R whose membership can be verified by a time- T , space- S random-access machine (RAM) M_R . The RAM model captures computations arising from real-world programs, and it makes the time–space gap exhibited by real-world computations (e.g., where $S \ll T$) explicit. Here, *complexity-preserving* means that the prover’s overhead is close to that of executing M_R itself: namely, prover time $\tilde{O}(T)$ and prover space $\tilde{O}(S)$.¹

Block et al. [12,13] initiated the study of *publicly verifiable* complexity-preserving arguments for RAM computations. Their constructions achieve prover time $\tilde{O}(T)$ and prover space $\tilde{O}(S)$, and are obtained in the streaming PIOP+PCS framework. However, their instantiations rely on classical group-based assumptions: [12] rely on the discrete-log assumption over prime-order groups in the random oracle model, while [13] obtains polylog verification from hidden-order groups. These public-key primitives are typically much more expensive than symmetric-key operations such as hashing, and are not post-quantum secure.

Towards Symmetric-Key, Post-Quantum Constructions. The public-key assumptions in [12,13] motivate the search for constructions from symmetric-key and post-quantum secure primitives. Bangalore et al. [4] gave the first such construction in the Ligero framework [3]. Their construction is minimal in the sense that they rely on *black-box* collision-resistant hashing and linear error-correcting codes, giving plausible post-quantum security, and can be viewed as an IOP/MPC-in-the-head protocol compiled with Merkle commitments. The prover is complexity-preserving, but the proof is not fully succinct, having length $\tilde{O}(T/S)$; consequently, when $S = \text{polylog}(T)$, the proof length is *nearly linear* in T . Moreover, they provide evidence that substantially improving over $\tilde{O}(T/S)$ proof length is difficult for approaches that encode long transcripts using codes with linear minimum distance in small space (essentially, all code-based zkSNARKs); i.e., small proofs and prover space require large prover time.

Very recently, Freitag, Paneth, and Pass [35] overcame the barrier of [4], constructing public-coin, complexity-preserving succinct arguments of knowledge for **NP** from collision-resistant hash functions assuming the existence of a suitable PCP. Their construction follows the classical PCP-plus-Merkle-tree approach of Kilian [48] and utilizes ideas from recursive proof composition [10]. Crucially, they make *non-black-box* usage of the underlying hash functions, circumventing the barriers of [4]. Thus, at the level of asymptotic feasibility, one can obtain polylog-sized proofs and verification time while obtaining a $\tilde{O}(T)$ time and $\tilde{O}(S)$ space prover. However, this approach is wildly impractical due to non-black-box usage of the underlying hash functions, and due to the cost of materializing the PCP. To contrast, Wang, Hazay, and Venkatasubramanian use the approach of Bangalore et al. to build Ligetron [44,60], which is an active ZK App Development platform that runs time- and space-efficient zkSNARKs in your browser.

The PIOP+PCS Route. The above works show that symmetric-key complexity-preserving succinct arguments are possible in principle, but they leave open what can be achieved in the polynomial-IOP plus polynomial-commitment-scheme framework. This framework underlies many of the fastest known transparent SNARKs, including FRI/STARK-style systems [6,8,9,62], Spartan/Brakedown-style systems [20,22,40,54], and HyperPlonk [24]. It also underlies the complexity-preserving constructions of [12,13], although their instantiations rely on public-key cryptographic assumptions. This motivates our central question:

What proof-size, prover-time, and prover-space tradeoffs are achievable for time- T , space- S RAM computations in the streaming PIOP+PCS framework using only symmetric-key primitives?

1.1 Our Results

We study this question by building new ZKP arguments in the streaming PIOP + PCS paradigm of Block et al. [12,13]. Our technical core is a collection of space-efficient streaming multilinear polynomial commitment schemes (MLPCSs) using code-based commitments inspired by Brakedown [40] and BaseFold [62]. Each framework gives different

¹Here and throughout the paper, $\tilde{O}(\cdot)$ hides $\text{poly}(\lambda, \log(S), \log(T))$ factors, where λ is the security parameter.

Work	Asymptotic Costs			Concrete Instantiations
	Proof Size	T_P	S_P	
[35]	$\tilde{O}(1)$	$\tilde{O}(T)$	$\tilde{O}(S)$	–
[4]	$\tilde{O}(T/S)$	$\tilde{O}(T)$	$\tilde{O}(S)$	–
This Work				
Thm 1.1	$\tilde{O}(\sqrt{T})$	$\tilde{O}(T^{1.5})$	$\tilde{O}(S)$	Shockwave (Section 3.2), $\mathcal{O}_R$ (Section 5.3), $\mathcal{O}_C$ (Section 5.6)
Thm 1.2		$\tilde{O}(T)$	$\tilde{O}(S + \sqrt{T})$	Shockwave (Section 3.3), $\mathcal{O}_R$ (Section 5.2), $\mathcal{O}_C$ (Section 5.5)
Thm 1.3	$\tilde{O}(1)$	$\tilde{O}(T_{\text{Enc}} + T)$	$\tilde{O}(S + S_{\text{Enc}})$	General Linear Code (Section 4.3)
Thm 1.4		$\tilde{O}(T^{1.5})$	$\tilde{O}(S + \sqrt{T})$	BaseFold (Section 4.4), $\mathcal{O}_R$ (Section 5.4), $\mathcal{O}_C$ (Section 5.7)

Table 1: Comparison of proof-size tradeoffs and instantiations in the context of prior work. Here, complexities are stated in terms of prover time T_P and prover space S_P . The parameters T and S denote the time and space complexities of the underlying RAM computation M_R , and $\mathcal{O}_R, \mathcal{O}_C$ denote the instantiations of our MLPCS with the row-oracle model and the column-oracle model of encoding, respectively.

tradeoffs between prover complexity and proof sizes: Brakedown-style commitments give the most efficient prover, with $\tilde{O}(\sqrt{T})$ -size proofs. BaseFold-style foldability pushes the proof size down to polylog, but at the cost of a marginally less optimal prover. Notably, the security of both systems relies only on collision-resistant hashing (i.e., Merkle commitments), and that the underlying linear code has linear minimum distance.

At a high level, we show how to implement Brakedown-style PCS and BaseFold-style PCS in the multi-pass streaming model of Block et al., developing a framework for *succinct access* to large encoding matrices along the way which enables space-efficient provers for general code-based commitments. Our approach departs from the “encode-the-transcript with (FFT-friendly) linear minimum distance codes” paradigm that drives much of the IOP/PCP literature [3,6–9,26,40,55,56]. Below, formally state our results for **NP** relations R verified by a time- T , space- S RAM M_R . We summarize our results in Table 1.

1.1.1 Square-root sized Proofs.

Our first set of results operates under the Brakedown MLPCS [40], and have square-root sized proofs.

Theorem 1.1 (Informally Stated, see Theorem 6.2). *Assume the existence of a collision-resistant hash function. Then, every NP relation that can be verified by a time- T and space- S RAM computation has a public-coin zero-knowledge argument system in which the prover, given as input the instance x and witness w , runs in time $\tilde{O}(T^{1.5})$ and uses space $\tilde{O}(S)$. The verifier runs in time $|x| \cdot \tilde{O}(\sqrt{T})$, the proof size is $\tilde{O}(\sqrt{T})$ and the number of rounds is $O(\log T)$.*

At first glance, the prover in Theorem 1.1 is space-optimal but not time-preserving. This trade-off is intentional, and possibly inherent. Bangalore et al. [4] provide evidence that when one insists on very short proofs (i.e., far below $\tilde{O}(T/S)$), the prover cannot simultaneously remain time-preserving under approaches that implicitly require encoding $\tilde{O}(T)$ -length transcripts in small space with a code that has linear minimum distance. Our first theorem should be viewed in this light: it preserves $\tilde{O}(S)$ prover space while achieving the fastest prover time compatible with our succinctness targets, up to poly-logarithmic factors. In fact, when $S = \log(T)$, our construction admits significantly smaller proofs (of size $\sqrt{T}$) than the T/S proof size in [4]. A second subtlety is that our prover space is asymptotically smaller than the proof size. This is meaningful in a natural *streaming output* setting, as shown in Figure 1, where the prover outputs the proof incrementally as it is constructed. This is analogous in spirit to the perspective of [4], which argues that a verifier can run in constant working space while processing a sublinear-length proof in a streaming fashion.

A related point is that our verifier may use more space than the prover in some regimes (e.g., $S = \log(T)$), yet still fits the intended “weak verifier” setting: its work remains *sublinear* in the computation time T . Moreover, following the

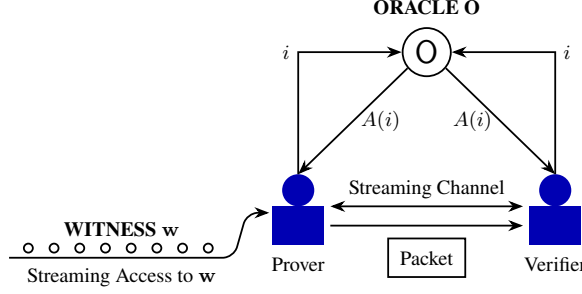


Figure 1: The communication model parties in our ZK argument/PCS setting. The prover and verifier interact using a streaming communication channel via discrete packets of information while the prover has streaming access to the witness. Both the verifier and the prover may query the oracle $\mathcal{O}$ that realizes the generator matrix in the PCS.

space-efficient verification ideas of [4], we believe our verifier can itself be implemented in a *streaming* fashion using only $\tilde{O}(1)$ working space (as we elaborate in Sections 3.2 and 5.2).

Our next theorem recovers near-linear prover time once we allow an additive $\tilde{O}(\sqrt{T})$ space term.

Theorem 1.2 (Informally Stated, see Theorem 6.4). *Assume the existence of collision-resistant hash functions. Then, every NP relation that can be verified by a time T and space S RAM computation has a public-coin zero-knowledge argument-system in which the prover, given as input the instance x and witness w , runs in time $\tilde{O}(T)$ and uses space $\tilde{O}(S + \sqrt{T})$. The verifier runs in time $|x| \cdot \tilde{O}(\sqrt{T})$, the proof size is $\tilde{O}(\sqrt{T})$ and the number of rounds is $O(\log T)$.*

Note that when $S \geq \sqrt{T}$, Theorem 1.2 is complexity-preserving and relies only on symmetric-key primitives.

Instantiations. Theorems 1.1 and 1.2 give two ZKPs with complementary prover-space profiles, which we obtain via a generic *streaming PCS-to-argument* compiler with the Brakedown MLPCS. Thus, the technical core of this work is the design of *space-efficient* code-based PCS, from which the two argument systems follow as black-box corollaries.

We introduce three different access models for the underlying linear code used in the Brakedown MLPCS, each capturing a broad and practically relevant class of codes and each requiring distinct algorithmic ideas for sublinear-space proving. First, we consider Reed–Solomon codes (Section 3): the encoding matrix is dense, but admits a succinct algebraic description (via evaluation structure), enabling on-the-fly access to the generator matrix. Here, we give both a $O(\log T)$ -space and a $\tilde{O}(\sqrt{T})$ -space committer, and each instantiation yields (via the compiler) one ZKP realizing Theorem 1.1 and one realizing Theorem 1.2. Second, we consider *row-sparse* codes (Section 5), which abstract many expander-based constructions and Brakedown-style linear-time encodable codes; again we give log-space and $\sqrt{T}$ -space committers and obtain the corresponding pair of ZKPs for Theorems 1.1 and 1.2. Third, we consider the dual setting of *column-sparse* codes and repeat the same development (Section 5).

Our motivation for treating these three regimes is twofold. On the one hand, they cover a broad spectrum of code families that appear (or are being actively developed) in succinct argument systems. On the other hand, they isolate a *small set of structural access properties*—dense-but-succinct (RS), row-succinct, and column-succinct—under which space-efficient proving is possible. As a result, advances in code design that fit into any of these regimes can be *plugged into* our framework to yield space-efficient polynomial commitments, and hence space-efficient ZKPs, with essentially no further protocol-level work.

1.1.2 Polylogarithmic-sized Proofs.

The Brakedown MLPCS constructions above achieve the fastest prover profiles at the cost of $\tilde{O}(\sqrt{T})$ proof size. To obtain polylog-sized proofs, we turn to the recent BaseFold MLPCS [62]. BaseFold shows that a MLPCS with polylog verifier costs can be built from a linear code equipped with a suitable *folding* structure. Conceptually, folding gives a FRI-like recursive proximity test [6]: instead of querying a set of square-root-size columns as in Brakedown-style commitments, the verifier checks consistency across log-many “folded” codewords; i.e., a codeword provided in round i is a random “folding” of two halves of the codeword from round $i - 1$.

However, the direction of “folding” matters in the streaming setting. Assume that codeword positions are indexed by bit strings in $\{0, 1\}^n$. Then, the folding used in BaseFold is naturally a most-significant bit (MSB) folding: for every $\mathbf{b} \in \{0, 1\}^{n-1}$, it combines the symbols at positions $0\|\vec{b}$ and $1\|\vec{b}$, i.e., it folds along the most significant bit, which folds the left half with the right half. In contrast, the streaming layout used in the PIOP+PCS framework of [12,13] gives the prover sequential access the input, and MSB folding pairs stream elements which far apart (i.e., 2^{n-1} positions away). This makes it difficult to perform the folding without either storing a large fraction of the codeword, or streaming the input multiple times and blowing up the time-complexity.

This issue is reminiscent of an observation in [12]: to make Bulletproofs [15,21] compatible with the streaming model, they use a complementary least-significant-bit folding order. Inspired by this, we introduce a notion of *LSB-foldable codes*. In an LSB folding, for every $\mathbf{b} \in \{0, 1\}^{n-1}$, the fold combines symbols at positions $\vec{b}\|0$ and $\vec{b}\|1$, i.e., adjacent locations under the streaming layout used by our prover. This folding order is compatible with the way our prover scans the witness and intermediate oracle messages.

We then construct an LSB-BaseFold MLPCS: a BaseFold-style MLPCS based on LSB-foldable codes. Intuitively, this amounts to doing a FRI-style LSB folding instead of the standard MSB folding. We prove security of this variant and show that it supports the same polylog proof-size behavior as BaseFold. We also give concrete instantiations of LSB-foldable codes: we show that Reed–Solomon codes admit the required folding structure, and we give a randomized procedure for generating suitable LSB-foldable generator matrices. See Section 4.1 for details on LSB-foldability and the LSB-BaseFold construction.

Our first polylog proof-size result isolates the encoding cost. Suppose the underlying LSB-foldable code has a streaming encoder that, given streaming access to a message to encode, outputs codeword symbols sequentially in time T_{Enc} and S_{Enc} space. Then the prover incurs only an additive $\tilde{O}(T_{\text{Enc}})$ overhead over the RAM computation time T , with no asymptotic blow-up in memory beyond the RAM program and the encoder.

Theorem 1.3 (Informally Stated, see Theorem 6.3). *Assume the existence of a collision-resistant hash function and a LSB-foldable linear code $\text{Enc}^{\text{stream}}$ which runs in time T_{Enc} , space S_{Enc} , and outputs elements of its codewords sequentially. Then, every NP relation that can be verified by a time T and space S RAM computation has a public-coin zero-knowledge argument-system in which the prover, given as input the instance x and witness w , runs in time $\tilde{O}(T + T_{\text{Enc}})$ and uses space $\tilde{O}(S + S_{\text{Enc}})$. The verifier runs in time $|x| \cdot \tilde{O}(1)$, the proof size is $\tilde{O}(1)$ and the number of rounds is $O(\log T)$.*

This theorem should not be read as claiming that known generic encoders already give an optimal prover. For example, a naive low-space streaming encoder for Reed–Solomon codes has $T_{\text{Enc}} = O(T^2)$, yielding a quadratic-time prover. The point of the theorem is modularity: the proof-system overhead, beyond the cost of streaming encoding, is nearly linear-time, and small-space. Thus, any improvement in streaming encoders for linear LSB-foldable codes with linear minimum distance immediately improves the prover. For example, an encoder with $T_{\text{Enc}} = \tilde{O}(T)$ and $S_{\text{Enc}} = \text{polylog}(T)$ would yield a symmetric-key, streaming PIOP+PCS construction matching the efficiency of [35].

Conversely, the lower-bound evidence of [4] suggests that such encoders may be precisely the hard object, and may be impossible. This lower bound identifies small-space, near-linear-time encoding using a code with linear minimum distance as the bottleneck behind the $\tilde{O}(T/S)$ proof-size barrier for transcript-encoding PIOP/PCP approaches. Moreover, a recent result due to Cook and Moshkovitz [27] shows that if the prover must compute an entire codeword of length N given streaming access to the input, then $TS = \Omega(N^2)$. Neither result is a formal lower bound against our PCS construction: our results (a) identifies the same coding bottleneck of [4] from a different perspective, and (b) may not be subject to the lower bound of [27] since we do not explicitly compute entire codewords at a time, but rather output the codeword symbols as a stream (see Section 5.3 for a more detailed discussion). Theorem 1.3 shows that in our PCS-based approach, the analogous bottleneck is the existence of low-space, near-linear-time streaming encoders for LSB-foldable codes.

Our second polylog result gives an unconditional sub-quadratic instantiation for concrete LSB-foldable codes. Rather than treating the encoder as a black-box, we exploit the recursive structure of the BaseFold’s foldable code to compute the required folded codeword symbols on the fly. This yields a $\tilde{O}(T^{3/2})$ -time prover with sublinear space.

Theorem 1.4 (Informally Stated, see Theorem 6.5). *Assume the existence of a collision-resistant hash function. Then, every NP relation that can be verified by a time T and space S RAM computation has a public-coin zero-knowledge*

argument-system in which the prover, given as input the instance x and witness w , runs in time $\tilde{O}(T^{3/2})$ and uses space $\tilde{O}(S + \sqrt{T})$. The verifier runs in time $|x| \cdot \tilde{O}(1)$, the proof size is $\tilde{O}(1)$ and the number of rounds is $O(\log T)$.

Instantiations. As with our Brakedown-style results, we instantiate this construction under several access models for the underlying generator matrix of the linear code. These instantiations show that the same streaming-access framework applies not only to square-root Brakedown-style MLPCS, but also to foldable-code MLPCS constructions with polylog proof size. In particular, in [Section 4.4](#), we show that under certain assumptions related to the sparsity of BaseFold’s code, we can achieve $\tilde{O}(T^{3/2})$ -time by leveraging the recursive structure of foldable codes and computing necessary matrix elements on-the-fly. We further instantiate LSB-BaseFold in the row- and column-oracle models; see [Section 5.4](#) and [Section 5.7](#) full for details.

Finally, since all our protocols are public-coin, we can apply the Fiat-Shamir transform [\[34\]](#) in the Random Oracle Model to obtain non-interactive proofs, yielding time- and space-efficient zkSNARKs from symmetric-key primitives in the PIOP+PCS framework.²

1.1.3 A Perspective on Our Results.

Our results should be interpreted as a frontier around two prior landmarks. Bangalore et al. [\[4\]](#) gives full complexity-preserving arguments via symmetric-key primitives, with proof size $\tilde{O}(T/S)$, along with evidence that improving the proof length is difficult for transcript-encoding approaches without sacrificing prover time or space. Our constructions respect this evidence: when trying to push below $\tilde{O}(T/S)$ -sized proofs, we pay explicitly in another resource (either time or space). On the other hand, Freitag et al. [\[35\]](#) show that polylogarithmic proofs with $\tilde{O}(T)$ -time and $\tilde{O}(S)$ -space proving is possible from collision-resistance via PCP/Merkle techniques when making non-black-box usage of the hash functions. Our encoder-parameterized construction identifies what is needed to match [\[35\]](#) in the streaming PIOP+PCS framework: a near-linear-time, low-space streaming encoder for foldable codes.

Taken together with [\[12,13\]](#), our results show that the streaming PIOP+PCS framework is robust across assumptions: it yields time- and space-efficient provers both from classical group-based assumptions and, with the tradeoffs described above, from symmetric-key code-based polynomial commitments.

1.2 Technical Overview

We follow the methodology of Block et al. [\[12,13\]](#) to construct complexity-preserving arguments in the streaming model, which reduces to constructing a time- and space-efficient *multilinear polynomial commitment scheme in the streaming model*, which we denote as a *streaming MLPCS*. The technical core of this work is the construction of a family of streaming MLPCSs via the *Brakedown-MLPCS* framework [\[40\]](#), and via the *BaseFold-MLPCS* framework.

1.2.1 Brakedown-MLPCS: Review.

Golovnev et al. [\[40\]](#) introduced Brakedown, the first linear-time and field-agnostic³ SNARK for R1CS (and thus NP). Core to Brakedown is constructing a linear-time MLPCS, a general result distilled from Bootle, Chiesa, and Groth [\[16\]](#). We refer to this as the Brakedown-MLPCS.

Let $f: \{0,1\}^n \rightarrow \mathbb{F}$ be a multivariate function and let $\tilde{f}: \mathbb{F}^n \rightarrow \mathbb{F}$ be its unique *multilinear extension*. Write $N = 2^n$ and assume that $N = t \cdot k$. The Brakedown-MLPCS allows a prover to commit to $\tilde{f}$ and open $\tilde{f}$ at any evaluation point $\mathbf{z} \in \mathbb{F}^n$ assuming the following two ingredients: (1) a $[m, k, d]$ -linear code C with generator matrix $\mathbf{A} \in \mathbb{F}^{k \times m}$ and minimum distance d (i.e., the Hamming weight of $\mathbf{x}\mathbf{A}$ for any non-zero $\mathbf{x}$ is at least d), and (2) a collision-resistant hash function family $\mathcal{H} = \{H_\sigma: \{0,1\}^* \rightarrow \{0,1\}^\lambda\}_{\sigma \in \{0,1\}^\lambda}$ for security parameter λ . Let $\mathbf{u} = (f(0), f(1), \dots, f(N-1))$, where $i \in [0, N-1] := \{0, 1, \dots, N-1\}$ is interpreted as an n -bit vector in the natural way. The Brakedown-MLPCS consists of the following 3 phases.

²A minor subtlety is that our prover operates in the *streaming-output* model, where it emits proof symbols as they are generated. This is still compatible with the Fiat-Shamir transform since the transcript hash can be computed incrementally (e.g., via Merkle-Damgård-style domain extension [\[29,53\]](#)): the prover maintains only a constant-size chaining value and updates it as each fragment a_i is produced, e.g., $h_i \leftarrow H(h_{i-1} \parallel a_i)$, and derives challenges from h_i on-the-fly.

³*Field-agnostic* requires the SNARK to operate over any finite field and assumes no structure beyond the field being “sufficiently large.”

Commitment Phase. Committing to $\tilde{f}$ proceeds as follows. (1) Arrange $\mathbf{u}$ into a $t \times k$ matrix $\mathbf{Y}$ such that row $\mathbf{Y}_{i,j} = f(i \cdot k + j) = \mathbf{u}_{i \cdot k + j}$ for $i \in [0, t-1]$ and $j \in [0, k-1]$; (2) compute encoded matrix $\hat{\mathbf{Y}} \in \mathbb{F}^{t \times m}$ where $\hat{\mathbf{Y}}_{i,*} = \mathbf{Y}_{i,*} \cdot \mathbf{A}$ for $i \in [0, t-1]$ (here, $\mathbf{Y}_{i,*}$ denotes the i^{th} row of $\mathbf{Y}$); and (3) compute and send commitment $h = \text{Merkle}(\hat{\mathbf{Y}}_{*,0}, \dots, \hat{\mathbf{Y}}_{*,m-1})$, where $\hat{\mathbf{Y}}_{*,j}$ is the j^{th} column of $\hat{\mathbf{Y}}$ and Merkle denotes computing a Merkle tree of the given input (with respect to some $H_\sigma \in \mathcal{H}$).

Testing Phase. Upon receiving the commitment h , the verifier interactively tests that each row of $\hat{\mathbf{Y}}$ is (close to) a codeword of C . This is done as follows. (1) The verifier samples and sends $\mathbf{r} \xleftarrow{\$} \mathbb{F}^t$ to the prover; (2) the prover computes and sends $\mathbf{y} = \sum_{i=0}^{t-1} \mathbf{r}_i \cdot \mathbf{Y}_{i,*} \in \mathbb{F}^k$; (3) the verifier computes $\tilde{\mathbf{Y}} = \mathbf{y} \mathbf{A} \in \mathbb{F}^m$, then samples and sends $Q \subseteq [0, m-1]$ such that $|Q| = \Theta(\lambda)$; (4) the prover opens each column $j \in Q$ as $(\hat{\mathbf{Y}}_{*,j}, \pi_j)$, where π_j is the Merkle proof for leaf $\hat{\mathbf{Y}}_{*,j}$; (5) the verifier checks that $\tilde{\mathbf{Y}}_j = \langle \mathbf{r}, \hat{\mathbf{Y}}_{*,j} \rangle$ and that π_j is valid with respect to h for all $j \in Q$, rejecting if at least one check fails.

Evaluation Phase. For multilinear polynomial $\tilde{f}$, to evaluate $\tilde{f}(\mathbf{z})$ for any $\mathbf{z} \in \mathbb{F}^n$, it is well-known that there exists $\mathbf{z}^{(0)} \in \mathbb{F}^t$ and $\mathbf{z}^{(1)} \in \mathbb{F}^k$ such that $\tilde{f}(\mathbf{z}) = \langle (\mathbf{z}^{(0)} \otimes \mathbf{z}^{(1)}), \mathbf{u} \rangle$. Thus, for the Evaluation Phase with evaluation point $\mathbf{z}$, the verifier computes $\mathbf{z}^{(0)}, \mathbf{z}^{(1)}$ as described and runs the Testing Phase again with the prover with $\mathbf{r} := \mathbf{z}^{(0)}$. Let $\mathbf{y}' \in \mathbb{F}^k$ the vector sent by the prover, claimed to be $\sum_{i=0}^{t-1} \mathbf{z}_i^{(0)} \cdot \mathbf{Y}_{i,*}$. If all the verifier's tests pass in this phase, the verifier simply outputs $\langle \mathbf{y}', \mathbf{z}^{(1)} \rangle$. Now if the prover is honest, then $\langle \mathbf{y}', \mathbf{z}^{(1)} \rangle = \langle (\mathbf{z}^{(0)} \otimes \mathbf{z}^{(1)}), \mathbf{y} \rangle$, and hence the value outputted by the verifier is equal to $\tilde{f}(\mathbf{z})$.

Time- and Space-Efficiency of Brakedown-MLPCS. The time-efficiency of the prover heavily relies on the code C . One of the main technical contributions of Golovnev et al. [40] was the design of a *linear-time encodable, constant-rate, linear-distance* code C (i.e., $d = \Theta(m)$ and $m = \Theta(k)$). With such a code in hand, the Brakedown-MLPCS prover runs in $O(N)$ time (in terms of $\mathbb{F}$ -ops and hashes). Golovnev et al. also specify *Shockwave*, which sets C as a Reed-Solomon (RS) code. This bumps the prover time from $O(N)$ to $O(N \log(N))$ (assuming a suitable $\mathbb{F}$ to compute FFTs of size N), with the benefit of concretely smaller proofs (though asymptotically the same).

In either case, assuming a constant rate, linear minimum distance code, the proof size of Brakedown-MLPCS is $O_\lambda(t)$. For space-efficiency, the Brakedown-MLPCS prover naively requires $\Omega(N)$ space to simply store the vector $\mathbf{u}$ (the specification of the multilinear polynomial $\tilde{f}$).

1.2.2 Brakedown-MLPCS in the Streaming Model.

The space-inefficiency of Brakedown-MLPCS described above is inherent to all MLPCSs. Thus, to overcome this bottleneck and achieve space-efficient polynomial commitments, Block et al. [12] introduced the multi-pass streaming model for (ML)PCS, which gives the prover streaming access to the vector $\mathbf{u} \in \mathbb{F}^N$ described above, which allows the prover to either: (1) request the next element of the stream, or (2) (re)start the stream from the start. The prover can stream $\mathbf{u}$ as many times as they want, but only has “one-way” access: from $\mathbf{u}_i$, receive $\mathbf{u}_{i+1}$ or restart the stream and receive $\mathbf{u}_0$. In the streaming model, the prover can now access the matrix $\mathbf{Y}$ defined previously in row-major order since $\mathbf{Y}_{i,j} = \mathbf{u}_\ell$ for $i \in [0, t-1]$, $j \in [0, k-1]$ and $\ell = i \cdot k + j$.

Beyond streaming the vector $\mathbf{u}$, prior works also consider how the prover accesses the public parameters of the scheme. For Brakedown-MLPCS, the public parameters consist of the generator matrix $\mathbf{A} \in \mathbb{F}^{k \times m}$ and the hash function family $\mathcal{H}$. We assume that for any key $\sigma \in \{0, 1\}^\lambda$, the hash function $H_\sigma \in \mathcal{H}$ is describable (and thus computable) in $\text{poly}(\lambda)$ bits, which we assume the prover can store explicitly. Then, the main question is: how do we model access to $\mathbf{A}$?

Small-space Shockwave-PCS. Before tackling this question, we take a slight (but helpful) detour to the Shockwave-MLPCS, which is Brakedown-MLPCS instantiated with Reed-Solomon (RS) codes. For RS codes, to enable efficient FFT encoding, the matrix $\mathbf{A} \in \mathbb{F}^{k \times m}$ has a specific structure. Let $\omega \in \mathbb{F}$ be a m^{th} primitive root of unity. Then, set $\omega_j := \omega^j$ for $j \in [0, m-1]$, and let $\mathbf{A}_{i,j} = \omega_j^i$ for $i \in [0, k-1]$. This definition of $\mathbf{A}$ allows us to use FFTs to compute $\mathbf{x}\mathbf{A}$ in $O(m \log(k))$ time and $O(m)$ space.

This definition of $\mathbf{A}$ also gives us a *succinct* way to access every column of $\mathbf{A}$: given ω , compute ω_j and ω_j^i on-the-fly. With this view of $\mathbf{A}$, we can design two small-space implementations of Shockwave-PCS: one with $O_\lambda(\log(N))$ space

at the cost of super-linear time, and one with quasi-linear time and sublinear $O_\lambda(t)$ space. We show the following results.

Theorem 1.5 (RS-MLPCS I, informal). *Given streaming access to $\mathbf{u} \in \mathbb{F}^N$ with $N = 2^n$ and $t \cdot k = N$, the Brakedown-MLPCS can be instantiated with $[m, k, d]$ Reed-Solomon codes with the following properties:*

- constant-round, public-coin system with transparent setup;
- a $\tilde{O}(\log(N))$ -space committer running in $\tilde{O}(m \cdot N)$ time, using m passes over the evaluation stream $\mathbf{u}$;
- $O_\lambda(1)$ -sized commitment, and $\tilde{O}(t)$ evaluation proof size and verification time.

We summarize the key ideas behind [Theorem 1.5](#). First, during the commitment phase, as with the standard Brakedown-MLPCS, the prover will commit to the encoded matrix $\hat{\mathbf{Y}} = \mathbf{Y}\mathbf{A}$ column-by-column using Merkle-Damgård. That is, rather than computing the whole $\hat{\mathbf{Y}}_{*,j}$ and its hash in $O(t)$ space, the prover will compute the hash of this column via Merkle-Damgård. As column $\mathbf{A}_{*,j} = (1, \omega_j, \omega_j^2, \dots, \omega_j^{k-1})^\top$, we can compute $\mathbf{A}_{*,j}$ on-the-fly (i.e., in a streaming way) using constant space and $O(t)$ $\mathbb{F}$ -ops, allowing us to compute $\hat{\mathbf{Y}}_{0,j}, \dots, \hat{\mathbf{Y}}_{t-1,j}$ using $O(t)$ $\mathbb{F}$ operations per column $j \in [0, m-1]$. Crucially, each entry of the sequence can be computed in a constant number of field elements.

This commitment proceeds in-order from column $j = 0$ to $j = m-1$. For each column, the entire matrix $\mathbf{Y}$ must be streamed, giving our stated time complexity. Rather than storing all m leaf hashes, the prover will instead aggregate the hashes it has computed so far into nodes in the Merkle tree. For example, once the prover has hashed both $\hat{\mathbf{Y}}_{*,1}$ and $\hat{\mathbf{Y}}_{*,2}$ as h_1 and h_2 , respectively, it can compute and store $h_{1,2} = H_\sigma(h_1 \| h_2)$, continuing to propagate up the full Merkle tree as new hashes are computed. With this method, the prover stores at most $O(\log(N))$ hashes at any given time, giving our desired space complexity.

The ideas behind the space-efficiency of the commitment phase carry over to the testing/eval phase as well, with the exception that now the verifier challenge $\mathbf{r} \in \mathbb{F}^t$ must be succinctly describable. This can be done by allowing the prover to compute $\mathbf{r}$ in a streaming fashion from a short seed (e.g., a PRF), or by allowing $\mathbf{r}$ to be specified using logarithmic randomness $r_0, \dots, r_{\log(t)}$ [30]. Given this, the prover can compute the vector $\mathbf{y} = \sum_{i \in [0, t-1]} \mathbf{r}_i \cdot \mathbf{Y}_{i,*}$ entry-by-entry as $\mathbf{y}_j = \sum_{i \in [0, t-1]} \mathbf{r}_i \cdot \mathbf{Y}_{i,j}$ for $j \in [0, m-1]$, streaming $\mathbf{y}_j$ to the verifier as it is computed. This incurs m streams of the matrix $\mathbf{Y}$, again giving $O(mN)$ time. Finally, the Merkle authentication proof can be computed similarly as computing the commitment during the commitment phase, giving the specified time and space complexities. See [Section 3.2](#) for more details.

Setting $t = k = \sqrt{N}$, we obtain the following corollary.

Corollary 1.6. *Given streaming access to $\mathbf{u} \in \mathbb{F}^N$ such that $N = 2^n$ for even n , there exists a construction of the Brakedown-MLPCS using $[\Theta(\sqrt{N}), \sqrt{N}, d]$ Reed-Solomon codes with the following properties:*

- constant-round, public-coin system with transparent setup;
- a $\tilde{O}(\log(N))$ -space committer running in $\tilde{O}(N^{1.5})$ time, using $O(\sqrt{N})$ passes over $\mathbf{u}$;
- $O_\lambda(1)$ -sized commitment, and $\tilde{O}(\sqrt{N})$ evaluation proof size and verification time.

If the committer is allowed $\tilde{O}(m)$ space, we can leverage FFTs to reduce the committer time to quasi-linear.

Theorem 1.7 (RS-PCS II, informal). *Given streaming access to $\mathbf{u} \in \mathbb{F}^N$ such that $N = 2^n$ and $t \cdot k = N$, there exists a construction of the Brakedown-MLPCS using $[m, k, d]$ Reed-Solomon codes for $m = \Theta(k)$ with the following properties:*

- constant-round, public-coin interaction and transparent setup;
- a $\tilde{O}(m)$ -space committer running in $\tilde{O}(N)$ time, using $O(1)$ passes over $\mathbf{u}$;
- $O_\lambda(1)$ -sized commitment, and $O_\lambda(t)$ evaluation proof size and verification time.

The key idea behind [Theorem 1.7](#) borrows from [Theorem 1.5](#). First, since we are given $\tilde{O}(m)$ space, the prover can do the Merkle-Damgård trick from before, but for *every* column simultaneously, then perform a final Merkle tree on these hashes. Moreover, $\tilde{O}(m)$ space allows us to compute $\hat{\mathbf{Y}}_{i,*}$ directly using FFTs in $O(m \log(t))$ time and $O(m)$ space and by using a single pass over $\mathbf{Y}$ (we simply store the entire row $\mathbf{Y}_{i,*}$ since it is $O(m)$ - $\mathbb{F}$ elements). We again can apply similar ideas to the testing/eval phase, though this time we need not require $\mathbf{r}$ to be succinct, and can send $\mathbf{y}'$ directly to the verifier. See [Section 3.3](#) for details.

Again setting $t = k = \sqrt{N}$ gives us the following corollary.

Corollary 1.8. *Given streaming access to $\mathbf{u} \in \mathbb{F}^N$ such that $N = 2^n$ for even n , there exists a construction of the Brakedown-MLPCS using $[\Theta(\sqrt{N}), \sqrt{N}, d]$ Reed-Solomon codes with the following properties:*

- constant-round, public-coin interaction and transparent setup;
- a $\tilde{O}(\sqrt{N})$ -space committer running in $\tilde{O}(N)$ time, using $O(1)$ passes over $\mathbf{u}$;
- $O_\lambda(1)$ -sized commitment, and $O_\lambda(\sqrt{N})$ evaluation proof size and verification time.

Towards General Codes via Succinct Matrix Access. The Brakedown-MLPCS can be instantiated with *any* linear code, and subsequent work has explored many instantiations optimizing different efficiency metrics [\[11,50,61\]](#). However, space-efficient implementations face an immediate obstacle: even representing the encoding matrix $\mathbf{A}$ explicitly requires $\Omega(N^2)$ elements, making sublinear-space goals moot. To address this, we introduce a conceptual framework for *succinct access* to the encoding matrix.

Row-oracle Model. We model the code’s generator matrix by an oracle $\mathcal{O}_{\mathbf{A}}$ that, on input an index i , returns the i -th row of $\mathbf{A}$ in a representation that lists only its non-zero entries. This model is well-motivated for sparse codes (where each row has a constant or small number of non-zeros), and can also capture settings where rows are generated from a short seed or retrieved from external storage (e.g., Gemini’s external oracle for streaming RICS instances [\[17\]](#)). We then prove the following space-efficient PCS in the row-oracle model (tracking both time and the number of oracle queries):

Theorem 1.9 (Row-PCS I, informal). *In the row-oracle model realizing a generator matrix $\mathbf{A}$ with $t = \Theta(1)$ non-zero entries per row, there exists a Brakedown-style MLPCS with:*

- constant-round, public-coin system with transparent setup;
- a $\tilde{O}(\sqrt{N})$ -space committer running in $\tilde{O}(tN)$ time, using $\tilde{O}(1)$ passes over Y and making N oracle queries to $\mathcal{O}_{\mathbf{A}}$;
- $\tilde{O}(1)$ -sized commitment, and $\tilde{O}(\sqrt{N})$ proof size and verification time.

We note that the (sparse) row-oracle model is well-motivated due to the existence of (near) linear-time encodable expander codes [\[11,40\]](#) which have $t = \Theta(1)$ or $t = \Theta(\log(N))$ non-zero entries per row of the generator matrix.

To further reduce committer space, we introduce a “code-fragment” technique that trades time (and passes) for memory while preserving the asymptotic proof and verification costs.

Theorem 1.10 (Row-PCS II, informal). *In the row-oracle model realizing a generator matrix $\mathbf{A}$ with $t = \Theta(1)$ non-zero entries per row, there exists a Brakedown-style MLPCS with:*

- constant-round, public-coin interaction and transparent setup;
- a $\tilde{O}(t \log N)$ -space committer running in $\tilde{O}(tN^{1.5})$ time, using $\sqrt{N}$ passes over Y and making N oracle queries to $\mathcal{O}_{\mathbf{A}}$;
- $\tilde{O}(1)$ -sized commitment, and $\tilde{O}(\sqrt{N})$ evaluation proof size and verification time.

The key ideas behind the “code-fragment” approach are as follows. First, we commit to each row of the encoded matrix $\hat{\mathbf{Y}}$, rather than each column (note that the verifier will still open columns). Now, consider row i of $\hat{\mathbf{Y}}$, denoted as $\hat{\mathbf{Y}}_{i,*}$. By definition, $\hat{\mathbf{Y}}_{i,*} = \mathbf{Y}_{i,*} \mathbf{A}$. However, since we have row-oracle access to $\mathbf{A}$, computing $\hat{\mathbf{Y}}_{i,*}$ would require $O(\sqrt{N})$ calls to the oracle per entry of the row, which makes committing to $\hat{\mathbf{Y}}$ computationally expensive.

Our observation is that rather than committing to $\hat{\mathbf{Y}}_{i,*}$ directly, we instead commit to the following $j \in [\sqrt{N}]$ rows: $\mathbf{y}_{i,j} = \mathbf{Y}_{i,j} \cdot \mathbf{A}_{j,*}$. Notice now that $\sum_j \mathbf{y}_{i,j} = \hat{\mathbf{Y}}_{i,*}$. At the cost of more hashes, we can commit to $\hat{\mathbf{Y}}_{i,*}$ by committing to the linear combination of rows that realizes it. Moreover, this commitment is computed using a single pass over the j^{th} row of $\mathbf{A}$ per i , and overall uses a single pass over $\mathbf{Y}$. Note that changing the commitment in this way naturally forces us to re-imagine the testing and evaluation phases of Brakedown as well. We refer the reader to [Section 5](#) for full details.

Column-oracle Model. We also study the dual model in which the prover obtains succinct access to columns of $\mathbf{A}$. We show analogous trade-offs, and we discuss which code families naturally support row access, column access, or both. We discuss the Column-oracle Model in [Section 5](#).

1.2.3 BaseFold-PCS: Review.

Zeilberger et al. [62] introduced the BaseFold-MLPCS. This MLPCS departs from Brakedown-MLPCS and instead utilizes the FRI proximity test of Ben-Sasson et al. [6] and the sum-check protocol [52]. The key technical contribution of BaseFold is the introduction of *foldable codes*, which are recursively defined linear codes that are “compatible” with FRI.

As before, let $f: \{0,1\}^n \rightarrow \mathbb{F}$ be a function with multilinear extension $\tilde{f}$. The BaseFold-MLPCS allows a prover to commit to $\tilde{f}$ and open it at any evaluation point $\mathbf{z} \in \mathbb{F}^n$ assuming the following two ingredients: (1) a $[m_\ell, k_\ell, d_\ell]$ -foldable linear code C specified by generator matrices $\mathbf{A}_i \in \mathbb{F}^{k_i \times m_i}$ for $i \in [0, \ell]$ and diagonal matrices $\mathbf{T}_j \in \mathbb{F}^{m_j \times m_j}$ for $j \in [0, \ell - 1]$, where $k_\ell = N$, $k_i = 2k_{i-1}$, and $n_i = 2m_{i-1}$; ⁴ and (2) a collision-resistant hash function family $\mathcal{H}$ (parameterized by security parameter λ). Here, the foldable code is defined as follows: $\mathbf{A}_i = \begin{bmatrix} \mathbf{A}_{i-1} & \mathbf{A}_{i-1} \\ \mathbf{A}_{i-1} \cdot \mathbf{T}_{i-1} & \mathbf{A}_{i-1} \cdot (-\mathbf{T}_{i-1}) \end{bmatrix}$. In other words, to encode $\mathbf{x} \in \mathbb{F}^{k_\ell}$, one computes $\mathbf{x}\mathbf{A}_\ell$ directly, or performs a “divide-and-conquer” merge-sort style encoding: $\mathbf{x}\mathbf{A}_\ell = (\mathbf{y}, \mathbf{y}')$, where $\mathbf{y} = \mathbf{x}_L\mathbf{A}_{\ell-1} + \mathbf{x}_R\mathbf{A}_{\ell-1}\mathbf{T}_{\ell-1}$, $\mathbf{y}' = \mathbf{x}_L\mathbf{A}_{\ell-1} - \mathbf{x}_R\mathbf{A}_{\ell-1}\mathbf{T}_{\ell-1}$, and $\mathbf{x}_L, \mathbf{x}_R \in \mathbb{F}^{k_{\ell-1}}$ denote the left and right halves, respectively, of $\mathbf{x}$. Now, as before, set $\mathbf{u} = (f(0), \dots, f(N-1))$, where $i \in [0, N-1]$ is again naturally interpreted as an n -bit vector. Then, the BaseFold-MLPCS consists of the following two phases.

Commitment Phase. To commit to $\tilde{f}$, BaseFold-MLPCS computes $\mathbf{c} = \mathbf{u}\mathbf{A}_\ell$ (i.e., encodes $\mathbf{u}$), computes and sends $h = \text{Merkle}(\mathbf{c})$ using some $H_\sigma \in \mathcal{H}$.

Evaluation Phase. To prove that $\tilde{f}(\mathbf{z}) = \gamma$ for some $\gamma \in \mathbb{F}$, the BaseFold-MLPCS prover proves it knows a vector $\mathbf{u}$ such that (a) $h = \text{Merkle}(\mathbf{u}\mathbf{A}_\ell)$; and (b) $\langle \mathbf{u}, \text{eq}_{\mathbf{z}} \rangle = \gamma$, where $\text{eq}_{\mathbf{z}} \in \mathbb{F}^N$ is the unique vector satisfying $\tilde{f}(\mathbf{z}) = \langle \mathbf{u}, \text{eq}_{\mathbf{z}} \rangle$. Part (a) is proved using (a variant of) the FRI proximity test, while Part (b) is proved using the standard sum-check protocol. The protocol is $n = \log(N)$ rounds and operates as follows. At the start of round i of the protocol, the verifier has claim γ_i and commitment h_i , and the prover has “folded” codeword $\mathbf{c}^{(i)}$. The prover sends sum-check polynomial $q_i(X_i) = \sum_{\mathbf{b} \in \{0,1\}^{n-i}} \tilde{f}(\tilde{\alpha}_{i-1}, X_i, \mathbf{b}) \cdot \tilde{\text{eq}}_{\mathbf{z}}(\tilde{\alpha}_{i-1}, X_i, \mathbf{b})$, where $\tilde{\alpha}_{i-1} \in \mathbb{F}^{i-1}$ are the verifier challenges sent so far. The verifier checks that $q_i(0) + q_i(1) = \gamma_i$, then samples and sends $\alpha_i \xleftarrow{\$} \mathbb{F}$, and finally sets $\gamma_{i+1} = q_i(\alpha_i)$. The prover then computes and sends $h_{i+1} = \text{Merkle}(\mathbf{c}^{(i+1)})$, which is computed as follows: for $j \in [m_{\ell-i}]$, compute $f(X) = \text{interpolate}((\text{diag}(\mathbf{T}_{\ell-i})[j], \mathbf{c}^{(i)}[j]), (\text{diag}(-\mathbf{T}_{\ell-i})[j], \mathbf{c}^{(i)}[j + m_{\ell-i}]))$, and set $\mathbf{c}^{(i+1)}[j] = f(\alpha_i)$. In the final round of the protocol, the prover sends $\mathbf{c}^{(n+1)} \in \mathbb{F}^{m_0}$ and $\mathbf{u}^{(n+1)} \in \mathbb{F}^{k_0}$, and the verifier now checks two things: (1) $\mathbf{u}^{(n+1)}\mathbf{A}_0 = \mathbf{c}^{(n+1)}$, and (2) $q_n(\alpha_n) = \sum_{\mathbf{b} \in \{0,1\}^{\log k_0}} \mathbf{u}_{\mathbf{b}}^{(n+1)} \tilde{\text{eq}}_{\mathbf{z}}(\tilde{\alpha}_n, \mathbf{b})$. Finally, the prover and verifier run the FRI query phase, where the verifier checks consistency between $\mathbf{c}^{(i)}$ and $\mathbf{c}^{(i+1)}$ for all $i \in [n]$ and verifying each opened position of $\mathbf{c}^{(i)}$ is consistent with h_i (via Merkle paths). Note that in every intermediate round, the prover is proving it knows a vector $\mathbf{u}^{(i)}$ such that $\mathbf{c}^{(i)}$ is the encoding of this vector under the code $\mathbf{A}_{\ell-i}$ and $\mathbf{u}^{(i)}$ is the vector underlying the MLE $\tilde{f}(\tilde{\alpha}_{i-1}, X_i, \mathbf{b})$.

Time and Space Efficiency of BaseFold-MLPCS. Above, prover runs in time $O(N \log(N))$ due to the divide-and-conquer encoding approach, and enjoys succinct proofs of size $\text{polylog}(N)$, assuming the code $\mathbf{A}_0$ is suitably defined (e.g., a maximum-distance separable (MDS) code such as Reed-Solomon, or a random (almost) MDS code⁵). As with Brakedown-MLPCS, the space-efficiency of BaseFold-MLPCS is naively $\Omega(N)$ to simply store $\mathbf{u}$. Moreover, even if

⁴Note that $k_0 \geq 1$ but can be taken to be larger (e.g., 32 or $\log(N)$).

⁵An MDS code meets the Singleton bound: $d + k = m + 1$.

we ignore this trivial bound and try to take advantage of the recursive encoding procedure to only store $\mathbf{A}_0$ (which may require $\ll N$ space), storing the diagonal matrices $\mathbf{T}_i$ also requires $\Omega(m_i)$ space, giving $\Omega(N)$ since $m_\ell = N$.

1.2.4 BaseFold-PCS in the MLPCS Streaming Model.

As with Brakedown-MLPCS, the space-inefficiency of storing $\mathbf{u}$ is inherent to BaseFold as well. Thus, we seek to overcome this bottleneck using the multi-pass streaming model over the vector $\mathbf{u}$. As with Brakedown-MLPCS, with BaseFold-MLPCS, we need to consider how we access the matrices $\mathbf{A}_i$ and $\mathbf{T}_j$. Moreover, we need to tackle an inherent space-inefficiency of BaseFold-MLPCS present in the protocol as given.

For ease of presentation, first suppose we have streaming access to both $\mathbf{u}$ and $\mathbf{c} = \mathbf{u} \cdot \mathbf{A}_\ell$. In this case, we can clearly compute the commitment $h = \text{Merkle}(\mathbf{c})$ in $O(\log(N))$ space as follows. Initialize a stack S to be empty. Then, stream $\mathbf{c}_0$, compute $h_0 = H_\sigma(\mathbf{c}_0)$, and push $(h_0, 0)$ onto the stack. Continue with $\mathbf{c}_1$, obtaining h_1 and tuple $(h_1, 0)$. Now, since the top of stack is of the form (H, j) where $j = 0$, we pop the stack and push $(H', 1)$, where $H' = H_\sigma(h_0, h_1)$. At the end of this process, the stack will have a single entry, which we pop and output h . Throughout this process, the maximum size of the stack is $O(\log(N))$.

Now, in the Evaluation Phase, we hit a space-efficiency bottleneck. The prover must compute (a) the sum-check polynomial $q_2(X_2)$ and (b) the folded codeword $\mathbf{c}^{(2)}$.⁶ Both of these steps add space-inefficiencies. In Step (a), to compute q_2 , if we unfold the sum-check, we can see that the prover is forced to compute an underlying vector $\mathbf{u}^{(2)} = \alpha_1 \mathbf{u}_L + (1 - \alpha_1) \mathbf{u}_R$, where $\mathbf{u}_L, \mathbf{u}_R$ respectively denote the left and right halves of $\mathbf{u}$. Moreover, in Step (b), a similar issue occurs: unfolding the definition, $\mathbf{c}^{(2)} = \alpha_1 \mathbf{c}_L + (1 - \alpha_1) \mathbf{c}_R$. This is exactly the situation that Block et al. [12] had to address: the “folding” procedure is performed via a *most-significant bit* folding (MSB). Thus, to obtain a streaming-friendly, small-space protocol, we adopt the *least-significant bit* (LSB) folding strategy of Block et al., also known as the *even-odd folding*.

LSB-BaseFold-MLPCS in Logarithmic Space. Adopting the LSB folding strategy requires three changes to BaseFold-MLPCS. (1) “Reverse” the sum-check order: instead of starting with variable X_1 and working up to X_n , begin with $q_n(X_n)$ and work down to X_1 .⁷ (2) “Fold” the codeword $\mathbf{c}$ as an *even-odd* folding: $f(X) = \text{interpolate}((\text{diag}(\mathbf{T}_{\ell-i}[j], \mathbf{c}^{(i)}[2j]), (\text{diag}(-\mathbf{T}_{\ell-i}[j], \mathbf{c}^{(i)}[2j+1]))$. (3) Modify the foldable code $\mathbf{A}_\ell$ to be an *even-odd* folding, rather than a *left-right* folding.

Note that (1) does not change any properties about the sum-check protocol. Moreover, (2) also does not change any properties about the protocol, *so long as* we can maintain the invariant that the prover knows vector $\mathbf{u}^{(i)}$ such that $\mathbf{c}^{(i)}$ is an encoding of it under $\mathbf{A}_{\ell-i}$ and $\mathbf{u}^{(i)}$ is the vector representing the MLE $\tilde{f}(\mathbf{b}, X_{n-i}, \vec{\alpha}_{i-1})$. This is why we require (3): we must make the foldable code compatible with this even-odd folding. Fortunately, this is straightforward: let $\mathbf{A}_i$ be the i^{th} matrix, and let $\mathbf{A}_i^e, \mathbf{A}_i^o$ denote even and odd columns, respectively. Then, we simply let $\mathbf{A}_i^e = [\mathbf{A}_{i-1}, \mathbf{A}_{i-1} \mathbf{T}_{i-1}]^\top$ and $\mathbf{A}_i^o = [\mathbf{A}_{i-1}, \mathbf{A}_{i-1} (-\mathbf{T}_{i-1})]^\top$. Note that we are simply permuting the columns of the original “left-right” BaseFold foldable code to an even-odd structure. By definition, this makes our even-odd foldable code an *equivalent code* [57], which implies that all distance properties of even-odd folding are identical to the original BaseFold code.

Given these three ingredients, we obtain a log-space BaseFold-MLPCS implementation in the streaming model. First, notice that during every round of the protocol, by (1) we can compute the current sum-check polynomial using only a single pass over the input $\mathbf{u}$ and $O(\log(N))$ space. Now, by (2) and (3), we can perform a single pass over the stream $\mathbf{c}$ to compute h_i and using only $O(\log(N))$ space since to compute $\mathbf{c}^{(i)}[j]$, we only need to stream, in order, elements $j2^i$ to $(j+1)2^i - 1$ (then appropriately compute the folding on-the-fly). Moreover, here we only need *streaming access* to the diagonals of $\mathbf{T}_{\ell-k}$ for $k \in [i]$ to efficiently compute this folding, and only need a single pass over these diagonals to compute the entire $\mathbf{c}^{(i)}$. Finally, during the Merkle opening phase, if we view the sequence of codewords $\mathbf{c}, \mathbf{c}^{(2)}, \dots, \mathbf{c}^{(\ell)}$ as a tree where each node $\mathbf{c}^{(i)}[j]$ has children $\mathbf{c}^{(i-1)}[2j], \mathbf{c}^{(i-1)}[2j+1]$, then we can compute every node in this tree using only a single pass over $\mathbf{c}$, and therefore compute all the opening proofs simultaneously in small $\text{polylog}(N)$ space.

⁶One can compute any entry of eq_z in logarithmic time and space using techniques described by Block et al. [12,13].

⁷This is actually how [62] define their sum-check protocol, but assume that the MLEs order $\mathbf{u}$ so they are consistent with the ordering of $\mathbf{A}_\ell$. Assuming the standard ascending order, performing the sum-check from X_1 to X_n satisfies this property, or reversing $\mathbf{u}$ also satisfies this property.

This gives a total complexity of $O(N)$ hashes and $O(N \log(N))$ field operations, $O(\log(N))$ passes over both $\mathbf{u}$, $\mathbf{c}$, and $\mathbf{T}_k$ for $k \in [\ell - 1]$, space and proof size $\text{polylog}(N)$.

Streaming Codeword $\mathbf{c}$. In the above discussion, we crucially relied on having streaming access to $\mathbf{c}$ in addition to $\mathbf{u}$. Recall that this was one of the key challenges in implementing our space-efficient Brakedown-PCS. In general, we may assume that to stream $\mathbf{c}$, we require T_{Enc} time, S_{Enc} space, and t passes over the stream $\mathbf{u}$. Then, to further contextualize our results, our described BaseFold-MLPCS has the following complexities: $O(N \log(N) + T_{\text{Enc}})$ time, $\text{polylog}(N) + S_{\text{Enc}}$ space, and $O(t \cdot \log(N))$ passes over the stream $\mathbf{u}$. Making things more explicit, if we only have in-order streaming access to the columns of $\mathbf{A}_\ell$ and wish to maintain logarithmic space with $S_{\text{Enc}} = \text{polylog}(N)$, we have $T_{\text{Enc}} = O(N^2)$ and $t = O(N)$. As mentioned in [Section 1](#), intuitively we are running into the lower bounds of [\[4\]](#). In [Section 4.4](#), we show that with $S_{\text{Enc}} = O(\sqrt{N})$ space, we can improve our time complexity to $O(N \log(N) + N^{1.5})$ with $t = O(\sqrt{N})$.

BaseFold-MLPCS with Succinct Matrix Access. As with Brakedown-MLPCS, BaseFold-MLPCS can be instantiated with any suitable (i.e., large minimum distance) base code $\mathbf{A}_0$. Nonetheless, for BaseFold-MLPCS, representing the sequence of matrices $\mathbf{A}_i$ and $\mathbf{T}_i$ explicitly requires $\Omega(N^2)$ - $\mathbb{F}$ elements, again making sublinear-space goals moot. Thus, we additionally explore implementing BaseFold in or row-oracle and column-oracle models. See [Section 5](#) for details.

1.2.5 From MLPCS to Succinct Arguments for RAM.

Finally, we compile our streaming MLPCSs into public-coin arguments for RAM computations using the standard “PIOP/PCP + (ML)PCS” blueprint emphasized by several prior works [\[12,13,17,23\]](#): rather than sending large oracle messages in the clear, the prover commits to them with a MLPCS, then later proves consistency of queried evaluations. Instantiating the MLPCS with our symmetric-key-based streaming MLPCS yields plausibly post-quantum argument systems with prover time/space guarantees stated in [Theorems 1.1](#) and [1.2](#).

1.3 Other Related Work

Streaming SNARKs for RICS. A complementary line of work adapts the streaming paradigm of Block et al. [\[12,13\]](#) to proving satisfiability of *RICS* instances. Since *RICS* descriptions typically do not admit small-space representations, these works give the prover streaming access to the wire values (and, when needed, to the *RICS* constraints), and aim to reduce memory usage by carefully controlling access patterns and avoiding materializing large intermediate state. Gemini [\[17\]](#) formalizes *elastic* zkSNARKs for *RICS*, offering a spectrum of tradeoffs that includes linear-time proving at the cost of linear space. Sparrow [\[55\]](#) introduces a streamable, space-aware zkSNARK for circuits of size C , achieving $\tilde{O}(C)$ prover time and $\text{polylog}(C)$ proof size, but requiring $\tilde{O}(\sqrt{C})$ prover space and focusing on a restricted class of layered-data-parallel circuits. Scribe [\[5\]](#) achieves linear-time proving with only logarithmic *working* space, but assumes access to a linear-sized external disk during proof generation. Finally, Hobbit [\[56\]](#) obtains streamable zkSNARKs for general *RICS* via tensor codes composed with Reed–Solomon codes and achieves linear-time proving; however, it exhibits a pronounced space–succinctness tradeoff: when tuned for minimal proof size, the prover requires nearly linear space, whereas tuning for small space leads to nearly linear proof size.

Linear Minimum Distance Codes for Succinct Arguments. The idea of building SNARKs using constant-rate (i.e., $k/m = \Theta(1)$) error-correcting codes was proposed by Bootle et al. [\[15\]](#) and later refined using the technique of tensor IOPs [\[16\]](#). Subsequent work [\[18\]](#) further achieved zero-knowledge and succinct verification time. On the practical side, Golovnev et al. [\[40\]](#) proposed building a PCS using tensor IOPs; this led to Brakedown, which has been implemented. Xie et al. [\[61\]](#) improved the proof size of Brakedown from square-root to polylogarithmic using code-switching, but the resulting scheme is not field-agnostic. All these schemes use the error-correcting code of Spielman [\[59\]](#) and its generalizations and/or refinements [\[32,40\]](#) to achieve a linear-time prover, but have concretely large proofs since the concrete minimum distance of the code is small.

Zeilberger et al. [\[62\]](#) constructed BaseFold, a field-agnostic SNARK based on a different approach (that of so-called “foldable codes,” which intuitively are codes that work with the FRI protocol [\[6\]](#)); this approach yields quasi-linear-time

encodable codes with concretely large linear minimum distance, assuming the base code has large minimum distance (e.g., Reed-Solomon Codes). Block et al. [11] instantiated so-called expand-accumulate codes [19] over finite fields to achieve quasi-linear time SNARKs in the Brakedown framework, and showed that the juxtaposed expand-accumulate code has linear minimum distance. Recently, Khabbazzian [46] and Li et al. [49] prove that the exact-weight variant of the expand-accumulate code over finite fields has linear minimum distance.

In Blaze [20], the repeat-accumulate-accumulate (RAA) code [31,45], a super-fast linear-time encodable code, was shown to have linear minimum distance over binary extension fields. Abbaszadeh et al. [1] adapt the techniques of [11] to compute concrete parameters of the RAA code over any finite field, though they do not give a formal distance analysis. Recently, Huang et al. [43], Akhiani and Zhang [2], and Khabbazzian [47] prove that the RAA code asymptotically has linear minimum distance, with [2,43] further giving a concrete distance analysis.

Recently, Lightning [58] and Bolt [41] introduced sketched codes, which are essentially a simplified Brakedown code that first produce a compressed “sketch” of a message $\mathbf{x}$, then encode this sketch using a large linear distance code. Both schemes use random LDPC codes [36] to produce this sketch and achieve practically efficient linear-time encodable codes, but with poor concrete minimum distance. Finally, Li et al. [51] recently give a concrete distance analysis of the quasi-abelian code [14,33], which is almost linear-time encodable. The distance values obtained are concretely large, and using this code they instantiate Brakedown with very small proof sizes. However, it is currently unknown if this code has linear minimum distance (asymptotically).

1.4 Organization

In Section 2 we introduce all necessary notation and definitions for this work to build our argument system. We build two MLPCSs in Section 3 from the Brakedown framework. In Section 4, we construct three more MLPCS from the BaseFold framework and, building upon these, we provide a suite of polynomial commitment schemes from oracles in Section 5 which complement the schemes presented in prior sections. In Section 6 we compile our different MLPCSs into interactive arguments.

2 Preliminaries

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

We use bold upper case letters to denote matrices, e.g. $\mathbf{M} \in \mathbb{F}^{n \times m}$. For a matrix $\mathbf{M}$ of dimension $n \times m$, we let $\mathbf{M}[i, *] \in \mathbb{F}^n$ and $\mathbf{M}[:, j] \in \mathbb{F}^m$ denote the i^{th} row and j^{th} column respectively. For row vector $\mathbf{u}$ of length m and column vector $\mathbf{v}$ of length n , we let $\mathbf{u} \cdot \mathbf{M}$ and $\mathbf{M} \cdot \mathbf{v}$ denote the standard matrix-vector product. For two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{F}^n$ we denote by $\langle \mathbf{u}, \mathbf{v} \rangle$ the inner product. We denote by $\otimes$ the Kronecker product and by $\bigotimes_{i=1}^n$ the iterated Kronecker product.

Instantiating a Merkle hash tree is denoted by $\mathbf{M} \leftarrow \text{Merkle tree}$ and the algorithm for committing ℓ symbols to $\mathbf{M}$ is denoted by $h := \mathbf{M}.\text{Merkle}(s_1, \dots, s_\ell)$, where the opening of any s_i is denoted by $(s_i, \text{path}(i)) := \mathbf{M}.\text{Proof}(i)$. Similarly, for a set of leaves $Q \subset \{1, \dots, m\}$, m the number of leaves in $\mathbf{M}$, we denote the opening of all leaves $i \in Q$ by $\{s_i, \text{path}(i) | i \in Q\} := \mathbf{M}.\text{Proof}(Q)$. In the case where more than one Merkle roots are in the context of the protocol, denote C^i the i^{th} committed Merkle root, and denote by $\mathbf{M}^{C^i}$ a call to Open or Proof with respect to C^i . We use the shorthand $\text{interpolate}_\alpha((x_1, y_1), (x_2, y_2))$ to denote the univariate degree one polynomial constructed by interpolation through the points (x_1, y_1) and (x_2, y_2) evaluated on α , that is, $\text{interpolate}_\alpha((x_1, y_1), (x_2, y_2)) = y_1 \frac{\alpha - x_2}{x_1 - x_2} + y_2 \frac{\alpha - x_1}{x_2 - x_1}$.

We consider constant-rate, linear min. distance encoders $\text{Enc} : \mathbb{F}^m \rightarrow \mathbb{F}^{rm}$ (distance δ), implemented as a linear map: for $\mathbf{x} \in \mathbb{F}^m$, $\text{Enc}(\mathbf{x}) = \mathbf{x} \cdot \mathbf{A} = \mathbf{w} \in \mathbb{F}^{rm}$, where $\mathbf{A}$ is a matrix of dimension $m \times rm$. Such linear codes have been used in many recent works, such as [22,40,54,61,62].

2.1 Multilinear Polynomials

An n -variate polynomial $f : \mathbb{F}^n \rightarrow \mathbb{F}$ is multilinear if the individual degree of each variable in f is at most one.

Fact 2.1. A multilinear polynomial $f : \mathbb{F}^n \rightarrow \mathbb{F}$ is uniquely defined by its evaluations over the Boolean hypercube and for every $\zeta \in \mathbb{F}^n$ it holds that

$$f(\zeta) = \sum_{\mathbf{b} \in \{0,1\}^n} f(\mathbf{b}) \cdot \prod_{i=1}^n \chi(b_i, \zeta_i); \quad \chi(b_i, \zeta_i) := b_i \cdot \zeta_i + (1 - b_i) \cdot (1 - \zeta_i).$$

We typically denote the multilinear polynomial f by the N -sized sequence $\mathcal{Y} := (f(\mathbf{b}))_{\mathbf{b} \in \{0,1\}^n}$, and denote the evaluation of the multilinear polynomial defined by $\mathcal{Y}$ on ζ as $\text{ML}(\mathcal{Y}, \zeta) := \mathbf{q}_1 \cdot \mathcal{Y} \cdot \mathbf{q}_2$, where $\mathbf{q}_1, \mathbf{q}_2$ are vectors in $\mathbb{F}^{2^{n/2}}$ (depend on ζ) such that $f(\zeta) = \langle (\mathbf{q}_1 \otimes \mathbf{q}_2), \mathcal{Y} \rangle$.

2.2 Multilinear Polynomial Commitment Schemes

We take our definitions mostly verbatim from [11–13]. We use the following convention for any list of arguments or returned tuple $(a, b, c; d, e)$: variables listed before the semicolon are known both to the prover and the verifier whereas the ones after are known only to the prover.

Definition 2.2 (Multilinear polynomial commitment scheme). A multilinear polynomial commitment scheme over a finite field $\mathbb{F}$ is a tuple of algorithms (Setup, Com, Eval) with the following specifications:

- $pp \leftarrow \text{Setup}(1^\lambda, 1^n)$: on input security parameter λ , and number of variables n , outputs public parameters pp . In our case, pp is a matrix A corresponding to the encoding algorithm of a linear code.
- $C \leftarrow \text{Com}(pp; \mathcal{Y})$: on input pp and a description $\mathcal{Y} = (f(\mathbf{b}))_{\mathbf{b} \in \{0,1\}^n} \in \mathbb{F}^{2^n}$ of an n -variate multilinear polynomial, outputs a commitment C .
- $\text{Eval}(pp, C, \zeta, \gamma; \mathcal{Y})$: is a public-coin interactive proof system (P, V) for the relation

$$R_{\text{ml}} = \{(pp, C, \zeta, \gamma; \mathcal{Y}, d) : C = \text{Com}(pp; \mathcal{Y}) \wedge \gamma = \text{ML}(\mathcal{Y}, \zeta)\},$$

where $\zeta \in \mathbb{F}^n$ is an evaluation point, $\gamma \in \mathbb{F}$ is a claimed value, and $\text{ML}(\mathcal{Y}, \zeta)$ denotes the multilinear extension of $\mathcal{Y}$ evaluated at ζ .

We require the following properties.

Perfect correctness: For all $\lambda, n \in \mathbb{N}$, all $\mathcal{Y} \in \mathbb{F}^{2^n}$ and all $\zeta \in \mathbb{F}^n$,

$$\Pr \left[1 = \text{Eval}(pp, C, \zeta, \gamma; \mathcal{Y}) \mid \begin{array}{l} pp \leftarrow \text{Setup}(1^\lambda, p, 1^n) \\ C \leftarrow \text{Com}(pp, \mathcal{Y}) \\ \gamma = \text{ML}(\mathcal{Y}, \zeta) \end{array} \right] = 1.$$

Knowledge Soundness: For any polynomial-time adversary $\mathcal{A}$, there exists an expected polynomial-time extractor $\mathcal{E}$ that outputs the witness such that

$$\Pr \left[\begin{array}{l} \mathcal{Y} \leftarrow \mathcal{E}^{\mathcal{A}}(pp, C, \zeta, \gamma), \\ \text{Com}(pp; \mathcal{Y}) = C, \\ f'(\zeta) \neq \gamma \end{array} \mid \begin{array}{l} pp \leftarrow \text{Setup}(1^\lambda, p, 1^N) \\ (C, \gamma, \zeta) \leftarrow \mathcal{A}(pp) \\ \text{Eval}(pp, C, \zeta, \gamma; \mathcal{Y}) = 1 \end{array} \right] \leq \text{negl}(\lambda)$$

where $\mathcal{E}^{\mathcal{A}}$ denotes that $\mathcal{E}$ has oracle access to $\mathcal{A}$.

Definition 2.3 (Streamable MLPCS). A streamable multilinear polynomial commitment scheme (MLPCS) is a scheme (Setup, Com, Eval) as in Definition 2.2 that, in addition, satisfies the following efficiency properties for n -variate multilinear polynomials over $\mathbb{F}$:

1. The commitment output by Com has size $\tilde{O}(1)$ bits. Assuming multi-pass streaming access to the evaluation vector $\mathcal{Y}$, Com runs in space $O(\sqrt{2^n})$, and the number of passes over the vector $\mathcal{Y}$ is $O(\sqrt{2^n})$.
2. The communication complexity of Eval is $\tilde{O}(\sqrt{2^n})$ and the verifier in Eval runs in time $\tilde{O}(\sqrt{2^n})$. Assuming multi-pass streaming access to $\mathcal{Y}$, the prover in Eval runs in space $O(\sqrt{2^n})$.

2.3 Interactive Multilinear PCP Definition

We present the definition of an interactive (multilinear) PCP (IMPCP).

Definition 2.4 (Interactive Multilinear PCPs [13]). *An interactive multilinear PCP (IMPCP) is a protocol that consists of interaction between a prover (P) and a verifier (V). At the beginning of the protocol, given a witness w and a statement x the prover provides a message π to the verifier which is encoded by its multilinear extension. The verifier is given as input the statement x and has oracle access to π . The verifier and prover then interact in a (public-coin) interactive proof and the verifier ultimately accepts or rejects where the following hold for the interaction:*

Perfect Correctness: For all tuples $(x, w) \in \mathcal{R}$ and $\lambda \in \mathbb{N}$,

$$\Pr[P(x, w, 1^\lambda) \wedge V^\pi(x, w) = 1] = 1$$

Soundness: For all possible cheating prover strategies P^* with $(x, w) \in \mathcal{R}$,

$$\Pr[P^*(x, w, 1^\lambda) \wedge V^{\pi^*}(x, w) = 1] = \text{negl}(\lambda) < \epsilon$$

We measure the running time and space requirements for an IMPCP by the length of the unencoded message and require that the prover needs to generate this message as a stream of bits.

3 Streamable Brakedown-MLPCS from Reed-Solomon Codes

In this section, we describe our Reed-Solomon-based MLPCS in the streaming model with logarithmic-space (Section 3.2) and square-root space (Section 3.3) implementations. We begin with an overview of the Brakedown PCS in Section 3.1 for completeness.

3.1 Overview of the Brakedown-style PCS

We abstract the Brakedown framework to serve as the foundation from which we build several of our time- and space-efficient polynomial commitments in the following sections. We present Algorithm 1 which describes the behavior of each of our multilinear polynomial commitment schemes built from the Brakedown-PCS found in Section 3.2, Section 3.3, Section 5.2, Section 5.3, Section 5.5, and Section 5.6.

Overview of the Brakedown Scheme. We briefly describe the contents of Algorithm 1, previously presented in Section 1.2, for completeness. In Setup on line (1) the prover establishes the parameters for the generator matrix of the encoder. This encoder can be instantiated with any linear code, such as Reed-Solomon codes. On lines (2-6) in Com the prover encodes each of the rows of the witness $\mathcal{Y}$ and then Merkle commits to the codeword symbols produced by encoding. Note in the Brakedown framework, the prover Merkle-commits to the columns of the encoded witness, but we omit this to allow for a more general framework. In Eval, on line (9), the prover sends a random linear combination of the rows of the witness to the verifier. For the rest of the protocol, the verifier tests if randomly selected linear combinations of specific columns in the encoded witness is equal to the encoded symbol of the random linear combination sent on line (9).

Building upon the overview of the Brakedown PCS presented in Section 1.2, our first task is to demonstrate how we can modify this scheme in order to admit space-efficient multilinear polynomial commitment schemes. With this, we present the first MLPCS with Reed-Solomon codes and state our first result.

3.2 Streamable Shockwave-MLPCS in Logarithmic Space

In this section we give our streamable MLPCS with $\sqrt{N}$ proof sizes that admits a time $O(N^{1.5})$ and space $O(\log N)$ committer using the Brakedown-style MLPCS with Reed-Solomon codes, denoted as Shockwave-MLPCS. We additionally use an observation about the well known MerkleDamgård hashing technique to achieve a logarithmic space committer with time $O(N^{1.5})$. We state this more formally in Theorem 3.1.

Algorithm 1 (Setup, Com, Eval)($1^\lambda, p, 1^n, \zeta, \gamma; \mathcal{Y}$) - Brakedown Overview

Setup($1^\lambda, p, 1^n$):

1: $pp \leftarrow (\mathbf{A}, \text{Enc}(\mathbf{x}))$, where $\mathbf{A} \in \mathcal{M}_{\sqrt{2^n}, r\sqrt{2^n}, d}$, $d \in O(\log(\sqrt{2^n}))$, and $\text{Enc}(\mathbf{x}) := \mathbf{x} \cdot \mathbf{A}$

Com($pp; \mathcal{Y}$):

2: $\mathbf{M} \leftarrow$ Merkle tree
3: **for** each row $\mathcal{Y}[i, *] \in \mathcal{Y}$ **do**:
4: $\hat{\mathcal{Y}}[i, *] \leftarrow \text{Enc}(\mathcal{Y}[i, *])$
5: $C \leftarrow \text{M.Merkle}(\hat{\mathcal{Y}}[i, j])$ for all $j \in [r\sqrt{2^n}]$
6: **end for**

Eval($pp, C, \zeta, \gamma; \mathcal{Y}$):

7: $P \rightarrow V : C$
8: $P \leftarrow V$: Random string $\mathbf{r} \in \mathbb{F}^m$
9: P computes and sends

$$\mathbf{y}' := \sum_{i=1}^{\sqrt{2^n}} \mathbf{r}_i \cdot \mathcal{Y}[i, *] \in \mathbb{F}^{\sqrt{2^n}}$$

10: $P \leftarrow V : Q$ where $|Q| = k \in O(\lambda)$
11: $P \rightarrow V :$

$$\hat{\mathcal{Y}}_{i,j}, i \in \mathcal{Y}[*], j \in Q, \\ \text{M.Proof}(Q) := \{q_k, \text{path}(k) | k \in Q\}$$

12: V checks for all $k \in Q$:

$$\text{Enc}(\mathbf{y}')_k \stackrel{?}{=} \sum_{i=1}^{\sqrt{2^n}} \mathbf{r}_i \hat{\mathcal{Y}}[i, k]; \text{ and} \\ \text{M.Open}(i) \text{ for all } i \in \text{M.Proof}(Q).$$

13: Now V computes:

$$\mathbf{q}_1 \in \mathbb{F}^{\sqrt{2^n}}, \mathbf{q}_2 \in \mathbb{F}^{\sqrt{2^n}}. \text{ s.t. } f(\zeta) = \langle (\mathbf{q}_1 \otimes \mathbf{q}_2), \mathcal{Y} \rangle = \gamma$$

14: $P \leftrightarrow V$: Redo testing, replacing $\mathbf{r}$ with $\mathbf{q}_1$. Let $\mathbf{z}'$ be the value sent by the prover.
15: V finally checks if $\gamma \stackrel{?}{=} \langle \mathbf{z}', \mathbf{q}_2 \rangle$
16: V : Output 1 if all tests pass, 0 otherwise.

Theorem 3.1. Assume the existence of a collision resistant hash function. Then there exists a streamable polynomial commitment scheme (Setup, Com, Eval) for any security parameter λ and input tape of length $N = 2^n$ for multilinear polynomials with the following guarantees:

- Com outputs a commitment of size $O_\lambda(1)$; and
- the running time of Com is $O(N^{1.5})$ hashes and $\mathbb{F}$ -ops, $\sqrt{N}$ passes over $\mathcal{Y}$, and uses space $\log(N)$.
- The committer of Eval performs $O(N^{1.5})$ hashes and $\mathbb{F}$ -ops, uses $\sqrt{N}$ passes over $\mathcal{Y}$, and uses space $O(\log(N))$.
- The receiver of Eval runs in time $O(\lambda\sqrt{N})$ and the size of the proof is $O(\sqrt{N})$.

Below we provide an overview of the scheme which realizes these results and subsequently present the proof of [Theorem 3.1](#).

Overview of the Logarithmic Space Shockwave-MLPCS. We achieve the results stated in [Theorem 3.1](#) through making changes to the i) Setup, ii) Com, and iii) Eval algorithms depicted in [Algorithm 1](#).

We approach Setup from the context of the Reed-Solomon (RS) encoder’s motivation, namely, the admission of an explicit generator matrix by RS encodings. This allows us to replace the Setup step on line (1) of [Algorithm 1](#) with on-the-fly computation of the values in the Reed-Solomon matrix A .

The space efficiency of this commitment comes from the changes we make to Com and the fact that we never store more than one current hash value in the prover’s memory or more than $\log(N)$ hash values in the current Merkle tree via the stack structure of our committer. We build leaves in column-major order by iterating over the indices of a codeword and building all the individual n codewords one symbol at a time. That is, we build the first entry of each codeword, commit it to the Merkle tree, and move on, until we are done computing the last symbol of each codeword. Therefore [Algorithm 3](#) can be used in [Algorithm 1](#) on lines (2-6) to generate the commitment C used as input to $\text{Eval}(pp, C, \zeta, \gamma; \mathcal{Y})$. Lastly, we modify Eval to inherit the space-efficient properties of our modified Com.

Proof of Theorem 3.1. We begin the proof by first presenting the efficiency analysis of [Algorithm 3](#).

Com and efficiency analysis. We have:

- $\sqrt{N}$ passes over $\mathcal{Y}$. On line (3), we iterate over every row of $\mathcal{Y}$, and we do this for each index of a codeword.
- $N\sqrt{N} T_P$. On lines (2-3) we iterate over $O(N)$ entries in the encoded matrix $\hat{\mathcal{Y}}$ and for each of these entries we must perform $\sqrt{N}$ summations and field multiplications.
 - We also note here that we may retrieve arbitrary columns of the Reed-Solomon matrix A due to the generalized Reed-Solomon code’s matrix’s implicit structure. Specifically:
 - For the n^{th} primitive root of unity ω : $\omega^i = \omega \cdot \omega^{i-1}$, where the prover only must store ω and ω_j . Then P can compute $(\omega^i)^j$ for the k columns $j \in Q$.
- $\log(N)$ prover space. Logarithmic space is achieved via the techniques on lines (5) and (9-12), where we use the MerkleDamgård hashing technique to hash codeword symbols together and then commit the fully hashed column node to the top of the stack. Since we have $rm \in O(\sqrt{N})$ columns we have to store at maximum $\log(\sqrt{N}) - 1$ hashes when building our Merkle tree.
- The size of our commitment is then $O_\lambda(1)$ as it is the Merkle root of the tree we build in [Algorithm 3](#).

Eval and efficiency analysis. We also make modifications to the Eval section in [Algorithm 1](#). When running the space efficient Eval, we must ensure that the prover and verifier’s interactions do not use more than $\log(N)$ working memory. Namely, the vectors sent on lines (3), (6), and the challenge vector $\mathbf{q}_1$ could prevent this. We ensure log-space through a variety of changes to Eval, presented in [Algorithm 2](#), and show that it satisfies the efficiency guarantees in [Theorem 3.1](#).

- The computation on line (3) is performed in time $N^{1.5}$ and each field element y'_j can be computed in space $O(1)$ by maintaining m accumulation buckets.
- We see that the prover can store $R_d(Q)$ defined on line (5) by observing that for k challenge indices, the maximum number of possible nodes in the upper levels of the Merkle tree that could be used for the proofs of membership is $k \log \sqrt{N}$.
- Observe that the verifier can maintain small space ($O(\lambda)$) if we perform the following computations upon receiving columns and Merkle hashes from the prover on line (6). Since the verifier only opens the vector $\mathbf{y}'$ at $k \in O(\lambda)$ points, when receiving each symbol of $\mathbf{y}'$, the verifier scales the symbol by the $O(\lambda)$ columns it has chosen as the challenge set. It aggregates these across symbols it receives from the prover, building the exact k points that the encoding of $\mathbf{y}'$ is evaluated on with one pass of the vector $\mathbf{y}'$. It performs the computation of scaling the k columns the prover sends with a similar bucketing technique, scaling them as it receives their symbols and aggregating.

- Lastly, we still must show the product of terms $\mathbf{q}_1$ can be computed on the fly or stored in small (log) space:
 - Recall from Fact 2.1 (and subsequent discussion) that the multilinear extension of our polynomial f evaluated at a point $\zeta \in \mathbb{F}^{\log N}$ is a vector-matrix-vector computation $\mathbf{q}_1 \cdot \mathcal{Y} \cdot \mathbf{q}_2$. If we analyze the structure of $\mathbf{q}_1$ (analogously $\mathbf{q}_2$), we can see it is easily computable in small space by the prover. In particular, for any $\mathbf{b} \in \{0, 1\}^{\log N/2}$ we have $\mathbf{q}_1(\mathbf{b}) := \prod_{i=1}^{\log(N)/2} \chi(\mathbf{b}_i, \zeta_i)$, which can be computed on-the-fly in space $O(\log N)$ and $\text{polylog}(N)$ time.

Security. We note that besides the logarithmic size randomness we make only algorithmic changes which affect the streaming nature of the Shockwave protocol, whereas the prover and verifier messages remain identical between the two schemes. Therefore, the security analysis follows by [40, Section 4.1]. By the analysis of [30] the scheme remains secure in spite of our modifications on lines (3) and (7) with only a $\log N$ multiplicative factor added onto the security given by [40]. $\square$

Algorithm 2 Eval($pp, C, \zeta, \gamma; \mathcal{Y}$) - Log-Space Implementation of Eval

Input: C , a Merkle commitment M ; $\mathcal{Y} \in \mathbb{F}^{n \times m=N}$

Output: Accept or reject

- 1: $P \rightarrow V : C$
- 2: $P \leftarrow V : \boldsymbol{\rho} = (\rho_1, \dots, \rho_\ell) \xleftarrow{\$} \mathbb{F}^\ell, \ell = \log m$
- 3: P defines the coefficient vector

$$\mathbf{r}^\otimes(\boldsymbol{\rho}) := \bigotimes_{i=1}^{\ell} (1 - \rho_i, \rho_i) \in \mathbb{F}^m,$$

computes

$$\mathbf{y}' := \sum_{i=1}^m \mathbf{r}_i^\otimes(\boldsymbol{\rho}) \cdot \mathcal{Y}[i, *] \in \mathbb{F}^m,$$

and sends $P \rightarrow V : \mathbf{y}'$.

- 4: $P \leftarrow V : Q$, where $|Q| = k \in \Theta(\lambda)$
- 5: P computes the relevant set of indices in M of height h at all depths d :

$$R_d(i) = \{k_d(i), s_d(i)\}, \quad k_d(i) = \left\lfloor \frac{i}{2^{h-d}} \right\rfloor, \quad s_d(i) = k_d(i) \oplus 1.$$

Then, for every level d we have $R_d(Q) = \bigcup_{i \in Q} R_d(i)$.

- 6: $P \rightarrow V :$

$$\begin{aligned} & \hat{\mathcal{Y}}[i, j], \quad i \in [m], j \in Q, \\ & \text{M.Proof} \left(h = \text{MerkleDamgård}(\hat{\mathcal{Y}}[*, j]) \right), \quad h \in R_d(Q). \end{aligned}$$

- 7: V checks, for every $k \in Q$,

$$\text{Enc}(\mathbf{y}')_k \stackrel{?}{=} \sum_{i=1}^m \mathbf{r}_i^\otimes(\boldsymbol{\rho}) \hat{\mathcal{Y}}[i, k],$$

and M.Open(i) for all $i \in \text{M.Proof}(h \in R_d(Q))$.

- 8: Now V computes:

$$\mathbf{q}_1 \in \mathbb{F}^m, \mathbf{q}_2 \in \mathbb{F}^m, \text{ s.t. } f(\zeta) = \langle \mathbf{q}_1 \otimes \mathbf{q}_2, \mathcal{Y} \rangle.$$

- 9: $P \leftrightarrow V$: Repeat the preceding proximity-consistency test with $\mathbf{q}_1$ in place of $\mathbf{r}^\otimes(\boldsymbol{\rho})$ and with a fresh query set Q' . Let $\mathbf{z}' \in \mathbb{F}^m$ denote the vector sent by P .

- 10: V checks

$$\gamma \stackrel{?}{=} \langle \mathbf{z}', \mathbf{q}_2 \rangle.$$

- 11: V : Output 1 if all tests pass, and 0 otherwise.
-

Algorithm 3 $\text{Com}(pp; \mathcal{Y})$ - Log Space Shockwave

Input: $\mathbf{A} \in \mathcal{M}_{m,rm,d}$ (a RS code), $\mathcal{Y} \in \mathbb{F}^{n \times m}$ with $N = nm$

Output: A Merkle root hash h

Require: $n \geq m$

```
1:  $Stk \leftarrow$  empty stack
2: for  $j \in [rm]$  do
3:   for  $i \in [n]$  do
4:      $c \leftarrow \langle \mathcal{Y}[i, *], \mathbf{A}[:, j] \rangle$ 
5:      $h \leftarrow \text{Merkle}-(h, c)$ 
6:   end for
7:    $\text{node} \leftarrow (h, 0)$ 
8:   while  $Stk.\text{top}.\text{height} = \text{node}.\text{height}$  do
9:      $\text{sib} \leftarrow Stk.\text{pop}$ 
10:     $\text{node} \leftarrow (H(\text{sib}.h \parallel \text{node}.h), \text{node}.\text{height} + 1)$ 
11:  end while
12:   $Stk.\text{push}(\text{node})$ 
13: end for
14: while  $Stk.\text{size}() > 1$  do
15:    $\text{right} \leftarrow Stk.\text{pop}()$ 
16:    $\text{left} \leftarrow Stk.\text{pop}()$ 
17:    $Stk.\text{push}(H(\text{left}.h \parallel \text{right}.h), \text{right}.\text{height} + 1)$ 
18: end while
19: return  $Stk.\text{pop}()$ 
```

3.3 Streamable Shockwave-MLPCS in Square-Root Space

We present our streamable MLPCS with $\sqrt{N}$ proof sizes that admits a time $\tilde{O}(N)$ and space $O(\sqrt{N})$ committer using the Shockwave-MLPCS.

Theorem 3.2. *Assume the existence of a collision resistant hash function. Then there exists a streamable polynomial commitment scheme (Setup, Com, Eval) for any security parameter λ and input tape of length $N = 2^n$ for multilinear polynomials with the following guarantees:*

- Com outputs a commitment of size $O_\lambda(1)$; and
- the running time of Com is $O(N \log N)$ hashes and $\mathbb{F}$ -ops, 1 pass over $\mathcal{Y}$, and uses space $O(\sqrt{N})$.
- The committer of Eval performs $O(N \log N)$ hashes and $\mathbb{F}$ -ops, uses 1 pass over $\mathcal{Y}$, and uses space $O(\sqrt{N})$.
- The receiver of Eval runs in time $O(\lambda\sqrt{N})$ and the size of the proof is $O(\lambda\sqrt{N})$.

As with the previous theorem, we provide an overview of our scheme which realizes the results stated in [Theorem 3.2](#) before presenting its proof.

Overview of the FFT-Enabled Scheme. We first note that we present modifications to all three algorithms in [Algorithm 1](#). At a high level, the protocol Com described in [Algorithm 4](#) and the subsequent modifications to the Eval protocol leverage techniques which exploit the $\sqrt{N}$ sized working memory the prover may use. In particular, the prover loads entire messages $\mathcal{Y}_i \in \mathbb{F}^m$ into its memory and then performs the Reed-Solomon FFT encoding. From there, the prover computes the hash vector $\mathbf{h}$, $|\mathbf{h}| = rm$ of each of these codeword symbols using the well known MerkleDamgård hashing technique. At each subsequent encoding, we continue to apply the MerkleDamgård hash to $\mathbf{h}$. When our prover exhausts the rows of $\mathcal{Y}$, it Merkle commits via $\text{M.Commit}(\mathbf{h})$, where h_j corresponds to the j^{th} hash value corresponding to column j in $\hat{\mathcal{Y}}$. In Eval, as with all subsequent sections, we will draw upon the modifications presented in [Algorithm 2](#) to argue for our complexity's presented in [Theorem 3.2](#).

Algorithm 4 $\text{Com}(pp; \mathcal{Y})$ - (Quasi)-Linear Time Shockwave

Input: $A \in \mathcal{M}_{m,rm,d}, \mathcal{Y} \in \mathbb{F}^{n \times m=N}$

Output: A Merkle root hash h

Require: $n \geq m$

- 1: $\mathbf{h} \leftarrow$ empty list
 - 2: $\mathbf{c} \leftarrow$ empty list
 - 3: **for** $i \in [n]$ **do**
 - 4: $\mathbf{c} \leftarrow \text{FFT}(\mathcal{Y}(i, *)) \in \mathbb{F}^{rm}$
 - 5: $\mathbf{h} \leftarrow \text{Merkle-Damgård}(\mathbf{h}(i), \mathbf{c}(i)) \forall i \in [rm]$
 - 6: **end for**
 - 7: **return** $\text{M.Commit}(\mathbf{h})$
-

Proof of Theorem 3.2. We begin the proof with the efficiency analysis of Com.

Com and efficiency analysis. This algorithm admits:

- 1 pass over $\mathcal{Y}$. Since we are using the Merkle-Damgård construction to aggregate hash values, on line (5) there is no need to re-stream $\mathcal{Y}$ on a per-column basis.
- $N \log \sqrt{N}$ prover time. On lines (3-6), for $\sqrt{N}$ messages we perform an FFT operation of time $\sqrt{N} \log \sqrt{N}$.
- $3\sqrt{N}$ prover space due to the cost of holding the Merkle-Damgård vector $\mathbf{h}$, the current vector being computed via the FFT, and the last vector computed via the FFT in memory.

Eval and efficiency analysis. When using the FFT algorithm for computing Reed-Solomon codewords for commitments, prior works have typically performed this with parameters set so that $\mathcal{Y} \in (\mathbb{F}^{\log N})^{\frac{N}{\log N}}$ [4,56] in order to extract the most from the fact that in this setting $T_{Enc} = n \log n$, yielding encoding times for $\log N$ many messages that are $\approx N$ in the size of the entire input. While this is efficient, and yields significantly smaller proof sizes than a square witness shape, such schemes suffer from prohibitively large demands on the prover's memory, namely also nearly N in the size of the input due to the need for the input to the FFT to be loaded into memory. We therefore present slight modifications to Eval in Algorithm 1 consistent with changes presented in Algorithm 2 which is asymptotically faster than the PCS presented in Section 3.2 with only $3\sqrt{N}$ working prover memory:

- On line (9), $\mathbf{y}'$ is computed in the natural way in one pass over the input $\mathcal{Y}$, where the prover stores $\sqrt{N}$ aggregator buckets and $\mathbf{r}$ in memory. This incurs an additional N time cost to the prover as in Section 3.2.
- On line (11) we run the FFT algorithm again, and similar to the above technique of Algorithm 2, emit codeword symbols and proofs of membership on all relevant indices after the codeword is returned. Both of these costs on lines (9) and (11) only incur a $3\sqrt{N}$ cost to the prover's working memory.

Security. As in Section 3.2, we note that we make algorithmic changes which only affect the streaming nature of the Shockwave protocol, and the security analysis follows by Section 4.1 of [40]. This concludes the proof of Theorem 3.2. $\square$

This concludes our discussion of streamable multilinear polynomial commitment schemes from Reed-Solomon codes.

4 Streamable MLPCS from Foldable Codes with Polylog-sized Proofs

In this section, we describe MLPCSs yielding proof sizes of $\tilde{O}(1)$ from foldable codes. Section 4.1 gives an overview of the BaseFold scheme as well as a description of LSB-foldable codes, and in Section 4.2 we provide a useful framework for considering the proof sizes and verifier times inherent in our BaseFold-style constructions. In Section 4.3 we describe our streamable MLPCS based on BaseFold that admits a time $O(T_{\text{Enc}} + N \log N)$ and space $O(\log N + S_{\text{Enc}})$ committer, and in Section 4.4 we provide our streamable MLPCS from BaseFold that admits a time $O(N^{1.5})$, space $O(\sqrt{N})$ committer.

At a very high level, our results are achieved through the use of the BaseFold PCS, which admits polylogarithmic proof sizes from left/right (necessarily LSB) foldable codes, that is, codes which may be folded to create a new codeword half the size of the original. Note that this folding style is analogous to MSB-folding, which natively folds a string $\mathbf{x}$ indexed by a bit string $\mathbf{b} \in \{0, 1\}^{2t}$, $t \in \mathbb{N}$ through setting $x'_i = x_{(0||\mathbf{b}')} + x_{(1||\mathbf{b}')}$ for every $\mathbf{b}' \in \{0, 1\}^t$. This will differ in our approach, which redefines the BaseFold PCS to work with even/odd foldable codes, or LSB foldable codes.

Through the use of foldable codes we gain the ability to reduce claims about codewords (i.e. that our encoded witness $\hat{\mathcal{Y}} \in \text{Enc}(\mathbf{x})$ for some $\mathbf{x} \in \mathbb{F}^m$) to claims of a much smaller form. In particular, we move from necessitating the verifier to check a statement made over a linear combination of the rows of our witness matrix $\mathcal{Y}$ to simply requiring the verifier to check a statement of size $O(1)$ with $O(\log(N))$ additional binding checks.

4.1 Overview of LSB Folding and the Modified BaseFold PCS

In this section we introduce LSB foldable codes and give an overview of the LSB Foldable BaseFold scheme. This scheme describes the folding nature of our BaseFold variant and how it interleaves with the sumcheck protocol, as well as informs the structure of the MLPCS which we give throughout the remainder of this section.

Definition 4.1 (LSB Foldable Distributions). *Let $\mathbf{M}_0 \in \mathbb{F}^{k_0 \times rk_0}$ the generator matrix of a $[rk_0, k_0, d_0]$ linear code where $k_0, r \in \mathbb{N}$ and D_0 the distribution that outputs $\mathbf{M}_0$ with probability 1. Then for every $i > 0$ we inductively define D_i , the distribution that outputs $\mathbf{M}_i \in \mathbb{F}^{k_i \times m_i}$, $k_i = 2^i \cdot k_0$, $m_i = r \cdot k_i$ which samples generator matrices $\mathbf{M}_0, \mathbf{M}_1, \dots, \mathbf{M}_i$, in the following way:*

1. Sample $\mathbf{M}_0, \mathbf{M}_1, \dots, \mathbf{M}_{i-1} \leftarrow D_{i-1}$.
2. Sample $\mathbf{t}_{i-1}^e \leftarrow \mathbb{F}^{m_{i-1}}$ and let $\mathbf{t}_{i-1}^o = -\mathbf{t}_{i-1}^e$.
3. Define $\mathbf{M}_i =$

$$\begin{bmatrix} \mathbf{M}_{i-1}[*] & \cdots & \mathbf{M}_{i-1}[*] \\ \mathbf{M}_{i-1}[*] \mathbf{t}_{i-1}^e & \cdots & \mathbf{M}_{i-1}[*] \mathbf{t}_{i-1}^o \end{bmatrix}$$

For clarity, we present an instantiation of a LSB foldable code from Reed-Solomon codes.

An Instantiation of a LSB Foldable Code. For clarity, we briefly present an instantiation of a LSB foldable code from Reed-Solomon codes, slightly modified from a similar description found in [62]. Fix $\mathbb{F}$ a finite field of (large) order such that $\mathbb{F}$ has a multiplicative subgroup G with order $m = 2^t$, $t \in \mathbb{N}$ and let $\omega \in G$ a primitive root of unity which generates G . Additionally, let $\mathbf{M}$ a Vandermonde matrix of shape $d \times d$ over $\mathbb{F}$ for degree- n polynomials such that $M[i, j] = (\omega^j)^i = \omega^{j \cdot i}$. It is true that for all even rows of $\mathbf{M}$, $M[i, j] = M[i, j + \frac{m}{2}] = \omega^{2i'j}$, $i = 2i'$, $i' \in [n/2]$. It is also true that for odd rows $M[i, j] = \omega^{2i'j} \cdot \omega^j$. Therefore, for a column j all entries within odd rows are entries from the even row above it scaled by ω^j . With this observation in hand we may define a matrix M_{n-1} of shape $\frac{d}{2} \times \frac{d}{2}$ whose columns are exactly the first $d/2$ columns of the even rows of $\mathbf{M}$. If we define $\mathbf{t}_{n-1}^e$ as the first $d/2$ exponentiations of $\omega^0, \omega^1, \dots, \omega^{d/2}$ and $\mathbf{t}_{n-1}^o$ as the last $d/2$ exponentiations of $\omega^{d/2+1}, \dots, \omega^{d-1}$, then we see that $\mathbf{M}_{n-1}$ is exactly $\mathbf{M}_{i-1}$ as defined in Definition 4.1.

Overview of the LSB Foldable BaseFold Scheme. From this definition (and example), we give an overview of our BaseFold PCS from LSB foldable codes as depicted in [Algorithm 5](#) of [Section 4.2](#). First, on lines (3-5) the prover Merkle commits to $\hat{\mathcal{Y}}$. Note that this encoding is not interleaved as is the case in Brakedown-style MLPCS. Due to this, we will treat $\hat{\mathcal{Y}}$ as the input to our MLPCS's Eval phase in this section in order to highlight the complexities inherent in the protocol itself, rather than the complexities inherited by encoding the entire witness. This is a departure from the presentation of [Section 3](#) which treats $\mathcal{Y}$ itself as the input to Eval. After Merkle commitment, the prover initiates the LSB sum-check protocol with the verifier on line (9) for the given statement and as it progresses on lines (9-10), later additionally uses the verifier's random challenges to LSB fold $\hat{\mathcal{Y}}$ on lines (14-16) via an interpolation function, halving the size of $\hat{\mathcal{Y}}$ at each round (line 14) and satisfying $\text{Enc}_i(f_i) = \text{Fold}(\hat{\mathcal{Y}}_{i+1})$, f_i our i variate folded polynomial associated with round i of the sum-check protocol for each $i \in [\log N - 1]$. For every round of folding, the prover also Merkle commits to a new Merkle tree associated with the new folded codeword they are building. This ultimately changes the verifier's check from $\text{Enc}(m) = c$ as described in [Section 1.2](#) for Brakedown/Shockwave style schemes to a check that $\text{Enc}_0(f_0) = \hat{\mathcal{Y}}_0$, a constant space check. Note that $\hat{\mathcal{Y}}_0$ may not necessarily be a codeword with a pre-image of length 1, but will always be of some constant length at which the recursion of the encoding stops. The total proof size is polylogarithmic in N because of the interleaved sum-check protocol (expressed on lines 9-10), the codeword check performed at each layer (line 25), and the Merkle paths opened and checked by the verifier (lines 23-24) required for each codeword check.

4.2 BaseFold-MLPCS with LSB Foldable Codes

In this section, we present our LSB folding scheme for the BaseFold protocol in [Algorithm 5](#) which achieves polylogarithmic proof sizes and verifier time as well as argue its complexities, correctness, and security. In particular, we defer discussion of prover complexities to [Section 4.4](#), and [Section 4.3](#) where we explore different space regimes for the prover as per the structure of [Section 3](#). We formally state this result through the following theorem.

Theorem 4.2. *Assume the existence of a collision resistant hash function. Then there exists a streamable polynomial commitment scheme (Setup, Com, Eval) for any security parameter λ and input tape of length $N = 2^n$ for multilinear polynomials with the following guarantees:*

- Com outputs a commitment of size $O_\lambda(1)$; and
- The receiver of Eval's running time is $O(\ell \log N)$, $\ell \in \Theta(\lambda)$, uses space $O(\log N)$, and the size of the proof is $\text{polylog}(N)$.

Proof of Theorem 4.2. We begin the proof with a complexity analysis of [Algorithm 5](#). We have:

- $\log^2 N$ proof size and verifier time is evident as the verifier receives $3k \log N$ codeword symbols over the course of the $\log N$ round query phase on lines (23-24), and with each symbol there is sent a Merkle proof which is of size at most $\log N$ at the top level.
- $\log^2 N(2\ell + 2)$ verifier work is evident from the sum-check protocol which incurs an overall cost to the verifier of $2 \log^2 N$ throughout the protocol from $\log N$ evaluations of $g_i(0/1)$ on lines (9, 28), and from ℓ Merkle path openings at each of the $\log N$ levels as well as 2 computations at each level on line (25).
- It can be seen that the verifier can also be made $\log N$ space through observing that on line (25) the verifier receives only the 2 elements it needs to compute each $\hat{\mathcal{Y}}_i[k]$ at round any i of the protocol, and the sizes of the Merkle proofs it receives are at most $\log N$.

In addition to our complexity analysis, we also give a proof of correctness as well as a proof overview of security which ultimately invokes the knowledge soundness theorem of [\[62\]](#).

Correctness. Observe that an honest prover on line (15) sets $\hat{\mathcal{Y}}_i[j]$ equal to exactly the verifier's check on line (25), so all checks in the query phase (lines 22-27) hold trivially when executed by an honest prover. It remains to show that $\hat{\mathcal{Y}}_0$ is a valid codeword. Similar to the proof of Lemma 5 in [\[62\]](#), we argue equivalently that for all $i = n - 1$ to 0, $\hat{\mathcal{Y}}_i$ is a valid codeword with respect to the generator matrix $\mathbf{M}_i$.

Let $i \in [n-1]$. Then, given verifier randomness $\alpha_i \in \mathbb{F}$ and message $\mathbf{x}_i \in \mathbb{F}^{k_i}$ with $\mathbf{x}_i = \mathbf{x}_{i+1}^L + \alpha_i \mathbf{x}_{i+1}^R$, where $\mathbf{x}^L, \mathbf{x}^R$ denote the left and right halves of their respective message, denote $\text{Enc}_i(\mathbf{x})$ the (even-odd) foldable encoding of $\mathbf{x} \cdot \mathbf{M}_i \in \mathbb{F}^{m_i}$.

Now, note that by the definition of our even-odd foldable code, we have that

$$\begin{aligned} \hat{\mathcal{Y}}_{i+1}[2j-1] &= \sum_{k=1}^{m_{i+1}} x_{i+1}[k] \cdot \mathbf{M}_{i+1}[k, 2j-1] \\ &= \sum_{k=1}^{m_i} x_{i+1}^L[k] \cdot \mathbf{M}_i[k, j] + \sum_{k=1}^{m_i} x_{i+1}^R[k] \cdot \mathbf{M}_i[k, j] \cdot \mathbf{t}_i^o[j] \\ &= \sum_{k=1}^{m_i} x_{i+1}^L[k] \cdot \mathbf{M}_i[k, j] + \mathbf{t}_i^o[j] \sum_{k=1}^{m_i} x_{i+1}^R[k] \cdot \mathbf{M}_i[k, j] \\ &= \text{Enc}_i(x_{i+1}^L)[j] + \mathbf{t}_i^o[j] \text{Enc}_i(x_{i+1}^R)[j] \end{aligned}$$

and similarly the same holds for $\hat{\mathcal{Y}}_{i+1}[2j] = \text{Enc}_i(x_{i+1}^L)[j] + \mathbf{t}_i^e[j] \text{Enc}_i(x_{i+1}^R)[j]$.

By line (14) of the protocol, it is true that

$$\hat{\mathcal{Y}}_i[j] \leftarrow \hat{\mathcal{Y}}_{i+1}[2j-1] \cdot \frac{\alpha_i - \mathbf{t}_i^e[j]}{\mathbf{t}_i^o[j] - \mathbf{t}_i^e[j]} + \hat{\mathcal{Y}}_{i+1}[2j] \cdot \frac{\alpha_i - \mathbf{t}_i^o[j]}{\mathbf{t}_i^e[j] - \mathbf{t}_i^o[j]}.$$

By setting both $\hat{\mathcal{Y}}_{i+1}[2j-1]$ and $\hat{\mathcal{Y}}_{i+1}[2j]$ to their forms as defined above, we see that for all $j \in [m_i]$, $\hat{\mathcal{Y}}_i[j] = \text{Enc}_i(\mathbf{x}_{i+1}^L)[j] + \text{Enc}_i(\mathbf{x}_{i+1}^R)[j] \cdot \alpha_i = \text{Enc}_i(\mathbf{x}_{i+1}^L + \alpha_i \mathbf{x}_{i+1}^R)[j] = \text{Enc}_i(\mathbf{x}_i)[j]$.

Security. We now argue the security of our protocol. We first note that the distance of our underlying code remains unchanged (since it is a permutation of the columns of the original code's encoding matrices). Then, through demonstrating that the verifier and prover messages are fundamentally unchanged between [Algorithm 5](#) and Protocol 4 of [62], this distance preservation then allows us to argue that we meet the conditions for Theorems 3, Lemmas 6-8, and then finally Theorem 4 (knowledge soundness) of [62].

To see why the distance is unchanged, simply note that for any $i \in [\log N - 1]$, $\mathbf{M}_i$ as defined by [Definition 4.1](#) satisfies $\mathbf{M}_i = \mathbf{M}_i^* \mathbf{P}$ where $\mathbf{P}$ is some permutation matrix and $\mathbf{M}_i^*$ is the foldable generator matrix described in [62, Definition 9]. It follows that our code is an equivalent code [57] to the code associated with $\mathbf{M}_i^*$ and as such its distance remains unchanged. This allows us to meet the conditions to invoke [62, Theorem 3 and Lemma 6]. What remains to argue is that the verifier and prover messages also remain unchanged between the protocols, which will allow us to invoke Lemmas 7-8 and Theorem 4 within BaseFold.

Consider the messages sent on lines (9), (10), and (23-24) within [Algorithm 5](#):

- (9): $g_i(X_i) \leftarrow \sum_{\mathbf{b} \in \{0,1\}^{i-1}} \tilde{f}(\mathbf{b}, X_i, \alpha_{i+1}, \dots, \alpha_n) \cdot \text{eq}(\mathbf{z}; (\mathbf{b}, X_i, \alpha_{i+1}, \dots, \alpha_n))$
- (10): $g_i(0), g_i(1), V$ checks: $g_i(0) + g_i(1) \stackrel{?}{=} g_{i+1}(\alpha_i)$
- (23-24): $\hat{\mathcal{Y}}_i[k], \mathbf{M}_i^{\text{C}_i}.\text{Proof}(\{k\})$ and $\hat{\mathcal{Y}}_{i+1}[2k-1], \hat{\mathcal{Y}}_{i+1}[2k], \mathbf{M}_i^{\text{C}_{i+1}}.\text{Proof}(\{2k-1, 2k\}), \forall k \in Q$,

where for any $j \in n_i$ as defined in [Definition 4.1](#) $\hat{\mathcal{Y}}_i[j] \leftarrow \text{interpolate}_{\alpha_i}((t_i^e[j], \hat{\mathcal{Y}}[2j-1]), (t_i^o[j], \hat{\mathcal{Y}}[2j]))$. We compare these to the prover messages sent in the BaseFold protocol:

- (1): $g_n(X_n) \leftarrow \sum_{\mathbf{b} \in \{0,1\}^{n-1}} f(\mathbf{b}, X_n) \cdot \text{eq}(\mathbf{z}; (\mathbf{b}, X_n)), g_n(0), g_n(1)$
- (2d): $g_i(X_i) \leftarrow \sum_{\mathbf{b} \in \{0,1\}^{i-1}} f(\mathbf{b}, X_i, \alpha_{i+1}, \dots, \alpha_{n-1}) \dots \text{eq}(\mathbf{z}; (\mathbf{b}, X_i, \alpha_{i+1}, \dots, \alpha_{n-1}))$
- (b(i-ii)): $\hat{\mathcal{Y}}_i[j] \leftarrow \text{interpolate}_{\alpha_i}((t_i^e[j], \hat{\mathcal{Y}}[j]), (t_i^o[j], \hat{\mathcal{Y}}[j + \frac{n_{i+1}}{2}]))$

- (IOPP.Query): $\hat{\mathcal{Y}}_i[k]$, $M^{C_i}.\text{Proof}(\{k\})$ and
- $\hat{\mathcal{Y}}_{i+1}[k]$, $\hat{\mathcal{Y}}_{i+1}[k + \frac{n_{i+1}}{2}]$, $M^{C_{i+1}}.\text{Proof}(\{k, \frac{n_{i+1}}{2}\})$, $\forall k \in Q$.

Observe that the only messages which substantially change between protocols are the messages sent on lines (23-24). However, it must be the case that these messages are in fact equivalent to the messages sent in the native BaseFold protocol. To see this, note that since the only difference between the messages is where a codeword at level i is indexed to build the codeword at level $i - 1$, the codewords built between the protocols are equivalent if the code used to encode the top level codeword is equivalent to that found in [62]. Since this is the case (as they are equivalent codes), it follows that the messages are equivalent, that is, they share the same codeword space. This allows us to invoke [62, Lemmas 7 and 8]. This satisfies the conditions for Theorem 4, and soundness holds due to the soundness of the native BaseFold protocol and the equivalence of our codes. This then satisfies the property of Knowledge soundness as defined in Definition 2.2, which completes the proof. $\square$

4.3 Streamable BaseFold-MLPCS in Logarithmic Space

In this section we present our streamable MLPCS with $\tilde{O}(1)$ proof sizes that admits a time $O(T_{\text{Enc}} + N \log N)$ and space $O(\log N + S_{\text{Enc}})$ committer, where we use T_{Enc} , S_{Enc} respectively to denote the total time and space it takes to compute a codeword from a given message. More formally:

Theorem 4.3. *Assume the existence of a collision resistant hash function and further assume that we may access a stream of the encoded witness through a general encoder which runs in total time T_{Enc} , requires space S_{Enc} , and takes t passes over the witness $\mathcal{Y}$. Then there exists a streamable polynomial commitment scheme (Setup, Com, Eval) for any security parameter λ and input tape of length $N = 2^n$ for multilinear polynomials with the following guarantees:*

- Com outputs a commitment of size $O_\lambda(1)$; and
- the running time of Com is $\tilde{O}(T_{\text{Enc}})$, takes $O(N)$ hashes, t passes over $\mathcal{Y}$, and uses space $O(\log N + S_{\text{Enc}})$.
- The committer of Eval runs in time $O(N + N \log N)$, takes $O(N)$ hashes, and requires space $O(\log N)$.
- The receiver of Eval's running time is $\tilde{O}(\ell)$, $\ell \in \Theta(\lambda)$, uses space $\tilde{O}(1)$, and the size of the proof is $\tilde{O}(1)$.

Remark 4.4. We remark that the foregoing Theorem statement instantiated naively with Reed-Solomon codewords yields a MLPCS with prover time $O(N^2)$ and number of passes over the witness $O(N)$ due to the density of the Vandermonde matrix used in Reed-Solomon codes. To see why this is the case, simply observe that the Vandermonde matrix of shape $(k_d \times m_d)$ at the top level is dense, as each column consists of exponentiations of a primitive root of unity ω . Thus, during the encoding we must multiply each element in the input $y_i \in \mathcal{Y}$ with $k_d \in O(N)$ many non-zero elements, resulting in $O(N)$ field operations per input element, a $O(N^2)$ total time computation. In preceding sections, the Reed-Solomon encoding was a desirable choice as the interleaved setting allows one to encode over a domain of size only $\sqrt{N}$, whereas in the non-interleaved BaseFold setting, the Vandermonde matrices become $O(N) \times O(N)$, leading to quadratic prover times. Due to this, we leave this theorem statement and its analysis in terms of a general linear encoder, which can be used in a black-box manner. Moreover, we emphasize that the asymptotics guaranteed here, given improvements in linear codes instantiations, can be readily used towards building complexity preserving arguments with logarithmic proof sizes.

Proof of Theorem 4.3. We start the proof by first presenting the efficiency analysis of Com within Algorithm 5.

Com and efficiency analysis. We have:

- t passes over $\mathcal{Y}$, as generating all elements in $\hat{\mathcal{Y}}$ requires t passes over the witness via invoking the general linear encoder to compute all entries of $\hat{\mathcal{Y}}$ on lines (3-5).
- T_{Enc} operations over $\mathbb{F}$ originating from performing the encoding in the commitment, also on lines (3-5).
- $2N - 1$ hashes from building the Merkle tree on line (5).

- $\log(N) + S_{\text{Enc}}$ space is achieved by hashing as described in [Algorithm 3](#) in [Section 3.2](#), using the technique of storing only the exact elements needed to compute the current Merkle hash, which is at most $O(\log N)$ in a binary tree with $O(N)$ leaves. The S_{Enc} term comes from invoking the general linear encoder.

Eval and efficiency analysis Now, we present our complexity analysis for Eval. In contrast to Com, in Eval we assume streaming access to $\hat{\mathcal{Y}}$ and argue the efficiencies of Eval with respect to the machinery contained therein, and without the overhead of encoding. We have:

- $O(1)$ passes over $\hat{\mathcal{Y}}$. We pass over $\hat{\mathcal{Y}}$ once on lines (8-19) when committing to the $i \in [\log N - 1]$ many codewords we create from the interpolation on line (15), and once again when we must rebuild the codeword to answer the verifier's queries on lines (22-27).
- We perform $O(N)$ operations over $\mathbb{F}$ from performing the interpolation to generate elements in all levels of codewords $i \in [\log N - 1]$ on lines (15-17) and lines (23-24), which costs 2 field multiplications per level to generate elements further down in the codeword halving. This is a geometric series $\sum_{i=1}^{\log(N)-1} \frac{m_d}{2^i} = m_d \in O(N)$. The $N \log N$ term originates from the LSB sum-check which is interleaved with the BaseFold protocol on lines (9-10), which is known to be computable in a streaming fashion in time $N \log N$ [[12,28](#)].
- We build Merkle trees on lines (13-17) and then rebuild them on lines (22-27). During both phases of Eval, in the worst case we must construct the entire Merkle tree (i.e., the verifier's query necessitates the last element of the corresponding level's codeword). Analyzing the cost of building the Merkle tree in the commitment phase of Eval, we have $2N - 1$ hashes to build the top level Merkle tree, $2(N/2) - 1$ to build the next, and so on, which gives $4N - \log N - 3$ total hashes. In the worst case, we hash $\approx 8N$ total elements across all phases of [Algorithm 5](#).
- $\log(N)$ space is achieved primarily from the LSB folding technique. This allows us to, given streaming access to the top level codeword, generate streaming access to the next level's codeword, and so on. More formally, given $\hat{\mathcal{Y}}_i[2j]$ and $\hat{\mathcal{Y}}_i[2j + 1]$, we can directly compute the next level's $\hat{\mathcal{Y}}_{i+1}[j]$ from the equation on line (14) given that we are holding only $\mathcal{Y}_i[2j + 1]$ in memory. Moreover, it is known that sum-check is compatible with logarithmic space given the LSB folding of the witness [[12,28](#)]. This allows us to perform the codeword generation phase on lines (10-20) in logarithmic space. What remains is a brief argument as to why the query phase from lines (22-27) do not take more than logarithmic space. Note that when the prover receives the query set from the verifier, they may sort it. Similar to approaches used in previous discussion surrounding [Theorem 3.1](#), the prover then uses the foregoing techniques to “re-run” the codeword generation phase and emit Merkle paths as well as queried values of codewords as it steps through the computation.

To argue the efficiency of the receiver in Eval, we invoke [Theorem 4.2](#), which concludes the proof of [Theorem 4.3](#) and our discussion of the modified BaseFold PCS from LSB folding. $\square$

4.4 Streamable BaseFold-MLPCS in Square-Root Space

We begin our exposition of this MLPCS through the use of the BaseFold foldable code encoder, and state our result formally through the following theorem.

Theorem 4.5. *Assume the existence of a collision resistant hash function. Then there exists a streamable polynomial commitment scheme (Setup, Com, Eval) for any security parameter λ and input tape of length $N = 2^n$ for multilinear polynomials with the following guarantees:*

- Com outputs a commitment of size $O_\lambda(1)$; and
- the running time of Com is $\tilde{O}(N^{1.5})$, takes $O(N)$ hashes, $\sqrt{N}$ passes over $\mathcal{Y}$, and uses space $O(\sqrt{N})$.
- The committer of Eval runs in time $\tilde{O}(N^{1.5})$ with $O(N)$ hashes, uses $\sqrt{N}$ passes over $\mathcal{Y}$, and uses space $O(\sqrt{N})$.
- The receiver of Eval's running time is $\tilde{O}(\ell)$, $\ell \in \Theta(\lambda)$, uses space $\tilde{O}(1)$, and the size of the proof is $\tilde{O}(1)$.

Proof of Theorem 4.5. We begin with a remark on the generator matrix of the BaseFold encoder's small space representation and subsequently present the efficiency analysis for Com.

Remark 4.6. Note that we assume we have streaming access to the diagonal vector $\mathbf{t}_i^e$ (where in the case of the BaseFold code $\mathbf{t}_i^e = -\mathbf{t}_i^o$) for any level of the generator matrix $\mathbf{M}_i$ as well as succinct access to the underlying BaseFold code, that is, we can access elements of the generator matrix $\mathbf{M}_d$ associated with the BaseFold code in $\tilde{O}(1)$ time and space $O(\sqrt{N})$. To see how this is achieved, note that the recursive structure of foldable codes as defined in Definition 4.1 sets any new column $\mathbf{M}_{i+1}[:, j]$ built from $\mathbf{M}_i$ equal to the concatenation of $\mathbf{M}_i[:, \frac{j}{2}]$ with $\mathbf{M}_i[:, \frac{j}{2}] \cdot \mathbf{t}_i[\frac{j}{2}]$, that is, $\mathbf{M}_{i+1}[:, \frac{j}{2}] = (\mathbf{M}_i[:, \frac{j}{2}] || \mathbf{M}_i[:, \frac{j}{2}] \cdot \mathbf{t}_i[\frac{j}{2}])$. Therefore, starting from $\mathbf{M}_0$, the base level generator matrix, we may describe $\mathbf{M}_i[:, j]$ by composing concatenations of $((\mathbf{M}_0[:, \frac{j}{2^i}] || \mathbf{M}_0[:, \frac{j}{2^i}] \cdot \mathbf{t}_0[\frac{j}{2^i}]) || (\mathbf{M}_0[:, \frac{j}{2^i}] || \mathbf{M}_0[:, \frac{j}{2^i}] \cdot \mathbf{t}_0[\frac{j}{2^i}] \cdot \mathbf{t}_1[\frac{j}{2^{i-1}}] \cdots))$ until we reach the desired level of $\mathbf{M}_i$. Then, since each element in any level $\mathbf{M}_i$ is just a series of multiplications by some $t_k[2/j^i]$ (or $-t_k[2/j^i]$) where $k \in [\log N]$, we can compute for any column $j \in [m_i]$ each entry of $\mathbf{M}_i[:, j]$ in time $\tilde{O}(1)$ while being well within our requirement of space $O(\sqrt{N})$. Moreover, we also assume that the base level matrix $\mathbf{M}_0$ is sparse such that the top level matrix $\mathbf{M}_d$ is $\sqrt{N}$ column-sparse. To achieve this, we set the sparsity c of the bottom level matrix $\mathbf{M}_0$ to be $c = \sqrt{\frac{k_0}{2^d}}$.

Com and efficiency analysis. Given the preceding remarks, we have:

- $\sqrt{N}$ passes over $\mathcal{Y}$, as generating $\sqrt{N}$ elements in $\hat{\mathcal{Y}}$ on lines (3-5) requires a pass over the witness, yielding $\sqrt{N}$ total passes over the witness.
- $N\sqrt{N}$ operations over $\mathbb{F}$ originating from performing $\sqrt{N}$ field additions and multiplications per element generated in the codeword on lines (3-5) due to the $\sqrt{N}$ column sparsity of our generator matrix $\mathbf{M}_d$.
- $2N - 1$ hashes from building our Merkle tree on lines (3-5).
- Within Com, $\sqrt{N}$ space is trivially achieved through similar arguments to the above $\log N$ space argument for computing Merkle trees, and we maintain $\log N$ space while computing our sequential codewords of size $O(\sqrt{N})$ by maintaining $O(\log N)$ many buckets with which we can sum their respective vector-matrix multiplications within.

Eval and efficiency analysis . Now, we present our complexity analysis for Eval. As before, we assume streaming access to the top level of $\mathcal{Y}$ and show that we can similarly gain streaming access to any codeword $\hat{\mathcal{Y}}_i, i \in [\log N - 1]$ in $\sqrt{N}$ space. This allows us to answer the verifier's queries to various levels of $\hat{\mathcal{Y}}$ in $\sqrt{N}$ space. We have:

- $O(1)$ passes over $\hat{\mathcal{Y}}$ due to a similar argument as to the above proof of Theorem 4.3, where we pass over the encoded witness once during lines (8-19) in the initial encoding of the codewords at levels $i \in [\log N - 1]$, and then once again when answering the verifier's queries on lines (22-27).
- $N + N \log N$ operations over $\mathbb{F}$ originating from i) performing 2 field multiplications per level to generate elements of the codeword halvings $\hat{\mathcal{Y}}_i, i \in [\log N - 1]$ and ii) performing the LSB sum-check. Note that this is the same complexity as to the $\log N$ space regime, which is explained by the fact that our LSB folding technique is extremely space efficient, and it is simply the encoding step where we exploit $\sqrt{N}$ space.
- The analysis of the number of hashes is unchanged as to that described in Theorem 4.3.
- $\sqrt{N}$ space is achieved primarily from the LSB folding technique. As seen in the argument of Theorem 4.3, this allows us to, given streaming access to the top level codeword, generate streaming access to the next level's codeword, and so on. We only must store the space it takes to perform the sum-check and the folding of individual elements towards the bottom level codeword $\hat{\mathcal{Y}}_0$, which both fit well within our space requirement of $\sqrt{N}$ (and in fact fit within a space requirement of $O(\log N)$).

To argue the efficiency of the receiver in Eval, we invoke Theorem 4.2, which concludes the proof of Theorem 4.5. $\square$

5 General Code-Based Streaming-MLPCS in the Oracle Model

We introduce our *oracle encoder model* $\mathcal{O}$ in Section 5.1, and then present our results with this model in the row-wise oracle model: linear time and square-root space MLPCS in Section 5.2 and logarithmic-space MLPCS in Section 5.3. Logarithmic proof size techniques from the row-wise oracle model are discussed in Section 5.4. We then discuss the dual mode of the column-oracle model in Section 5.5, Section 5.6, and Section 5.7.

5.1 An Oracle Model for the Encoding Matrix

Note that if we move away from the general Reed-Solomon (or Basefold) code, we have more flexibility in our prover times without having to sacrifice space efficiency. To realize this, we revisit our idea of a general linear encoder $\text{Enc}(\mathbf{x}) : \mathbb{F}^m \rightarrow \mathbb{F}^{rm}$ with constant rate r and linear minimum distance d as used in Section 4.3 but generalized towards any suitable linear code. In particular, for $\mathbf{A} \in \mathcal{M}_{n,m,t}$, fix t such that the edge expansion of our underlying graph G informing $\mathcal{M}_{n,m,t}$ is enough to realize a linear minimum distance d , rate $r > 1$, and $t = O(1)$. We call this matrix $\mathbf{A}$ our *general encoder matrix* and model the encoding process as simply $\mathbf{x} \cdot \mathbf{A} = \mathbf{y}$. Note that many linear codes are *systematic*, meaning that the original message $\mathbf{x}$ is prepended to the first $|\mathbf{x}| = m$ symbols in $\mathbf{y}$. We define our computation $\mathbf{x} \cdot \mathbf{A} = \mathbf{y}$ to also capture this behavior. Therefore, we have that our general encoder $\text{Enc}(\mathbf{x})$ is systematic, constant-rate, has linear minimum distance, and has for each row $\mathbf{A}[i, *]$ the number of non-zeroes is exactly t .

Instantiating the Oracle. We envision two natural ways to instantiate the oracle model. (a) Following [17], one may store the code matrix on an external storage server and give the prover streaming access to the requested sparse rows/columns. This is particularly suitable for *explicit* code constructions whose generators may not admit succinct local computation, but whose sparse neighborhoods can be retrieved efficiently. (b) Following [40] and subsequent work, one may generate rows/columns pseudorandomly from a short seed (e.g., by expanding a PRG seed that determines the nonzero positions and coefficients). This instantiation is appealing when no external storage is available and the prover must generate the required parts of the code matrix locally in a space- and time-efficient manner.

We define the syntax of our oracle in the following manner. Both oracles retrieve nonzero elements and their indices from $\mathbf{A}$. We denote the oracles $\mathcal{O}_C$ and $\mathcal{O}_R$ for the column-wise and row-wise oracles, respectively, and restrict our queries to either oracle for sequential rows and columns of $\mathbf{A}$; i.e., we may request nonzero indices from either $\mathcal{O}$ in sequential order $\mathbf{A}[i, *], \mathbf{A}[i + 1, *], \dots$, or $\mathbf{A}[:, j], \mathbf{A}[:, j + 1], \dots$. Let

$$\mathcal{O}_R.\text{Query}(i) := \{(j, \mathbf{A}[i, j]) \mid \text{for all } j \in \mathbf{A}[i, *] \text{ where } \mathbf{A}[i, j] \neq 0\}, \text{ and}$$

$$\mathcal{O}_C.\text{Query}(i) := \{(i, \mathbf{A}[i, j]) \mid \text{for all } i \in \mathbf{A}[:, j] \text{ where } \mathbf{A}[i, j] \neq 0\},$$

denote queries to oracles $\mathcal{O}_R$ and $\mathcal{O}_C$, respectively. In all algorithms which uses these oracles, we analyze the number of (respective) oracle queries made.

5.2 Streamable Quasi-linear Time, Square-Root Space MLPCS in the $\mathcal{O}_R$ -model

We give our streamable multilinear PCS with $\sqrt{N}$ proof sizes that admits a time $\tilde{O}(N)$ and space $O(\sqrt{N})$ committer using the row-wise oracle encoder model.

Theorem 5.1. *Assume the existence collision resistant hash functions. Then there exists a streamable polynomial commitment scheme (Setup, Com, Eval) for any security parameter λ and input tape of length $N = 2^n$ for multilinear polynomials with the following guarantees:*

- Com outputs a commitment of size $O_\lambda(1)$; and
- Com performs Nd hashes and $\mathbb{F}$ -ops, makes 1 pass over $\mathcal{Y}$, uses space $r\sqrt{N}$, and makes N calls to $\mathcal{O}_R$.
- The committer of Eval performs $Nd + N$ hashes and $\mathbb{F}$ -ops, makes 1 pass over $\mathcal{Y}$, uses space $r\sqrt{N}$, and makes N calls to $\mathcal{O}_R$.
- The receiver of Eval runs in time $O(\lambda\sqrt{N})$ and the proof size is $O(\lambda\sqrt{N} + \lambda \log(N))$.

Algorithm 5 (Setup, Com, Eval)($1^\lambda, p, 1^n, \mathbf{z}, \gamma; \mathcal{Y}$): Succinct MLPCS in Log Space

Setup($1^\lambda, p, 1^n$):

- 1: $pp \leftarrow (\mathbf{A}, \text{Enc}(\mathbf{x}), \mathbf{t}_{n-1}^e, \mathbf{t}_{n-1}^o)$, where $\mathbf{A} \in \mathcal{M}_{k_n, m_n}$, $\mathbf{t}_{n-1}^e, \mathbf{t}_{n-1}^o \in \mathbb{F}^{m_{n-1}}$, $\mathbf{t}_i^* \in \mathbb{F}^{m_i}$ the respective left/right half of $\mathbf{t}_{i+1}^e$ for $i < n$ and $\text{Enc}(\mathbf{x}) := \mathbf{x} \cdot \mathbf{A}$

Com($pp; \mathcal{Y}$):

- 2: $\mathbf{M} \leftarrow \text{Merkle tree}$
3: **for** each $j \in [m_n]$ **do**:
4: $\hat{\mathcal{Y}}_j \leftarrow \sum_{i=1}^{2^n} \mathcal{Y}_i A_{i,j}$
5: $C_n \leftarrow \text{M.Merkle}(\hat{\mathcal{Y}}_j)$
6: **end for**

Eval($pp, C, \mathbf{z}, \gamma; \hat{\mathcal{Y}}$):

- 7: $V : C_n := C; \gamma_n := \gamma$.
8: **for** $i = n$ to 1 **do**
9: $P \rightarrow V : g_i(X_i) \leftarrow \sum_{\mathbf{b} \in \{0,1\}^{i-1}} \tilde{f}(\mathbf{b}, X_i, \alpha_{i+1}, \dots, \alpha_n) \cdot \text{eq}(\mathbf{z}; (\mathbf{b}, X_i, \alpha_{i+1}, \dots, \alpha_n))$.
10: V checks: $g_i(0) + g_i(1) \stackrel{?}{=} \gamma_i$
11: $V \rightarrow P : \alpha_i \xleftarrow{\$} \mathbb{F}$.
12: $V : \gamma_{i-1} = g_i(\alpha_i)$.
13: $P : \mathbf{M} \leftarrow \text{Merkle tree}$
14: **for** $j \in [m_i]$ **do**
15: P sets: $\hat{\mathcal{Y}}_{i-1}[j] \leftarrow \text{interpolate}_{\alpha_i}((t_i^e[j], \hat{\mathcal{Y}}_i[2j]), (t_i^o[j], \hat{\mathcal{Y}}_i[2j+1]))$
16: P commits $C_{i-1} \leftarrow \text{M.Merkle}(\hat{\mathcal{Y}}_i[j])$
17: **end for**
18: $P \rightarrow V : C_{i-1}$
19: **end for**
20: $P \rightarrow V : \hat{\mathcal{Y}}_0$
21: $V \rightarrow P : Q$ where $|Q| = \ell \in \Theta(\lambda)$ and $Q \subset [0, m_d - 1]$ is randomly sampled.
22: **for** $i = n - 1$ to 0 **do**:
23: $P \rightarrow V : \hat{\mathcal{Y}}_i[k], \text{M}^{\text{Ci}}.\text{Proof}(\{k\}), \forall k \in Q$
24: $P \rightarrow V : \hat{\mathcal{Y}}_{i+1}[2k], \hat{\mathcal{Y}}_{i+1}[2k+1], \text{M}^{\text{Ci}+1}.\text{Proof}(\{2k, 2k+1\}), \forall k \in Q$
25: V checks:
 $\hat{\mathcal{Y}}_i[k] \stackrel{?}{=} \text{interpolate}_{\alpha_i}((t_i^e[k], \hat{\mathcal{Y}}[2k]), (t_i^o[k], \hat{\mathcal{Y}}[2k+1]))$
 $\text{M}^{\text{Ci}+1}.\text{Open}(\{2k, 2k+1\}), \text{M}^{\text{Ci}}.\text{Open}(\{k\})$ for all $k \in Q$
26: $k \leftarrow \lceil k/2 \rceil \forall k \in Q$
27: **end for**
28: V checks: $\text{Enc}_0(g_1(\alpha_1))/\text{eq}(\zeta, \alpha_1, \dots, \alpha_n) \stackrel{?}{=} \hat{\mathcal{Y}}_0$.
29: V : Output 1 if all checks pass, 0 otherwise.
-

Overview of the Commitment. We note that with this generalization in hand obtaining a linear-time MLPCS similar to the MLPCS in Section 3.3 is natural by allowing a prover with $\sqrt{N}$ space and by observing that we can employ the same machinery from Algorithm 4, the commitment produced by using FFTs on Reed-Solomon codewords. At a high level, we can modify Com in Algorithm 1 in the following ways to achieve this. We change the commitment Com to, instead of calling FFT on each row i , we call $\mathcal{O}.\text{Query}(j)$ to retrieve the product terms used in the partial codeword computations for that row. Then, we aggregate these computations into full symbols of the codeword for row $\mathcal{Y}[i, *]$. At the end of building a full codeword matrix, we commit with the same technique as Algorithm 3, and employ the MerkleDamgård technique to reduce the size of the Merkle tree's leaf level to $r\sqrt{N}$.

Then, we modify Eval to reflect the efficiency guarantees stated in Theorem 5.1. Our proof follows the format of the

previous theorem statements, beginning with an algorithm for Com and subsequent analysis of Com and the modified Eval.

Proof of Theorem 5.1. We begin with an efficiency analysis of Com.

Com and efficiency analysis. We have:

- Nd prover time. During computation on line (7) we see that for every symbol in $\mathcal{Y}$, we must multiply by the exact d non-zero elements in A for that symbol's row.
- 1 pass over $\mathcal{Y}$. Since we multiply each symbol of the witness matrix by the exact elements in A that it must scale, we only have to touch each symbol in the witness matrix once, as evidenced by lines (2-6) which iterate only over the rows, columns, and non-zero generator matrix elements.
- $r\sqrt{N}$ prover space. On lines (7,10) and we commit to each product symbol produced by the "scaling" operation described above, and keep $r\sqrt{N}$ of these in memory at a time to accumulate each codeword symbol in the current row of $\hat{\mathcal{Y}}$.
- N queries to $\mathcal{O}_R$. For every symbol in our witness matrix $\mathcal{Y}$, we must query the oracle on line (5) for the corresponding indices and values of A for the row of A which must be scaled by that symbol.

Eval and efficiency analysis. Again, we refer to our abstraction presented in Section 3.1 and modify Algorithm 1 to fit the results of Theorem 5.1 in the following ways:

- On line (9), we can use much of the same changes as the analysis of Eval in Section 3.3, where we compute $\mathbf{y}'$ with one pass of $\mathcal{Y}$ and maintain an aggregator bucket array with which we store each element of $\mathbf{y}'$ in as we scale rows of $\mathcal{Y}$ by $\mathbf{r}$. As in the previous sections, this incurs a N time cost for generating $\mathbf{y}'$.
- On line (11), we simply reperform the computation of Com as in Section 3.2 and Section 3.3 and emit Merkle proofs to the verifier when hashes of $\mathbf{M}$ and indices of columns of $\hat{\mathcal{Y}}$ are relevant to the challenge indices.
- When the verifier wishes to open the evaluations and create the codeword used in the test on line (12), the verifier may instead perform a similar optimization as in Section 3.2 to maintain space $O(\lambda)$. Here, in the row access model (as opposed to the column access model that the Reed-Solomon codes invite), we have the verifier do the following. Upon receiving a symbol of $\mathbf{y}'$, the verifier scales it by the exact k values of the codeword matrix A that appear in the row corresponding to the current symbol of $\mathbf{y}'$. It aggregates these in buckets, keeping only $O(\lambda)$ buckets throughout the computation. It calculates the aggregations of the scalar product between the columns it receives and $\mathbf{r}$ in the same fashion as in Section 3.2. This technique works for any verifier with row-wise access to A via $\mathcal{O}_R$.
- As in all other schemes, the new challenge vector $\mathbf{q}_1$ is produced in the same way as the analysis of Eval in Section 3.2.

Security. This protocol admits its multilinear polynomial commitment scheme from $\mathcal{O}_R$ which realizes a constant-rate linear code with linear minimum distance. Due to the results of [40], we then retain soundness by the distance and rate of our encoder. $\square$

Algorithm 6 $\text{Com}(pp; \mathcal{Y})$ - Linear Row Oracle

Input: $\mathcal{Y} \in \mathbb{F}^{n \times m=N}$ **Output:** A Merkle root hash h **Require:** $n \geq m$

```
1:  $\mathbf{h} \leftarrow$  empty list
2: for  $i \in [n]$  do
3:    $\mathbf{c} \leftarrow$  empty list
4:   for  $j \in [m]$  do
5:      $Q \leftarrow \mathcal{O}_R.\text{Query}(j)$ 
6:     for  $k \in Q$  do
7:        $c(k) \leftarrow (\mathcal{Y}[i, j] \times A[j, k]) + c(k)$ 
8:     end for
9:   end for
10:   $\mathbf{h} \leftarrow \text{MerkleDamgård}(\mathbf{h}(j), \mathbf{c}(j)) \forall j \in [m]$ 
11: end for
12: return  $\text{M.Commit}(\mathbf{h})$ 
```

5.3 Linear-time, Logarithmic Space MLPCS in the $\mathcal{O}_R$ -model

We present our streamable MLPCS with $\sqrt{N}$ -sized proofs that admits a time $O(N)$ and space $O(\log N)$ committer using the row-wise oracle encoder model.

Theorem 5.2. *Assume the existence of a collision resistant hash function. Then there exists a streamable polynomial commitment scheme (Setup, Com, Eval) for any security parameter λ and input tape of length $N = 2^n$ for multilinear polynomials with the following guarantees:*

- *Com outputs a commitment of size $O_\lambda(1)$; and*
- *the running time of Com is Nd hashes and operations over $\mathbb{F}$, 1 pass over $\mathcal{Y}$, and uses space $\log(Nd)$.*
- *The committer of Eval runs in $Nd + N$ hashes and operations over $\mathbb{F}$, uses $\sqrt{N}$ passes over $\mathcal{Y}$, and requires space $\log(Nd)$.*
- *The receiver of Eval's running time is $O(\lambda\sqrt{N} \log \sqrt{N} + \lambda\sqrt{N})$ and the size of the proof is $O(\lambda\sqrt{N} \log \sqrt{N})$.*

We achieve these results by observing that, as depicted in Fig. 2, one can commit to the encoded matrix $\hat{\mathcal{Y}}$ not by committing to its symbols but by instead committing to the computations comprising those symbols. We provide a remark here on how we leverage this technique to avoid known lower bounds associated with encoders and give a detailed overview.

Overview of the Row-Wise Small Space Encoder Scheme. We describe the modified Com and present the full algorithm in Algorithm 7. At a high level, since the map $\text{Enc}(\mathbf{x}) : \mathbb{F}^m \rightarrow \mathbb{F}^{rm}$ simply performs matrix multiplications of the form $\mathbf{x} \cdot \mathbf{A}$, it is easiest to compute codewords in column-major order across $\hat{\mathcal{Y}}$. However, if we wish to use this mapping to create a streamable multilinear polynomial commitment scheme, we cannot designate our Merkle commitments as columns, or even symbols of the encoded input sequence $\hat{\mathcal{Y}}$. This is because each symbol $\hat{\mathcal{Y}}[i, j]$ is a linear combination of A which requires holding at least $n = \sqrt{N}$ field elements in our buffer if we wish to build $\hat{\mathcal{Y}}$ in row-major order. The natural option here then is to compute codewords in column major order, but this requires re-streaming the input tape $\mathcal{Y}$ at least $\sqrt{N}$ times. We wish to avoid this. Therefore, we simply commit to the *products within* these linear combinations. Specifically, for each message to be encoded x we commit to the exact md field multiplications performed when multiplying each element $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m$ by each corresponding non-zero in the associated row of A .

Remark 5.3. This technique may bypass the lower bound of [27] since the committer is not required to output the entire codeword given streaming access to the message. Instead, for each codeword symbol c_i , the committer must compute and commit to the linear combination of $\mathcal{Y}[i, *]$ and $\mathbf{A}[:, i]$ which realizes c_i . This essentially means that the committer never fully computes c_i , but commits all information needed to open/compute this value. Moreover, the computational work of reconstructing c_i is performed by the verifier, shortcutting the encoding process altogether. The proof of [27, Theorem 9.5] shows that minimum distance properties cannot be preserved if the encoder takes insufficient time or has insufficient space, *if the entire codeword must be written at once*. Our prover-verifier asymmetry with respect to computing the codeword is distinct from typical usages of codes in MLPCSs, and may warrant further investigation.

In Fig. 2 we outline a sketch of this commitment using the toy example where $n = m = 2$, $r = \frac{3}{2}$, and $t = 2$. Now, we examine Eval. Since A is known to both parties a priori, the prover can efficiently recalculate associated symbols of requested codewords $\hat{\mathcal{Y}}[:, j]$ in the evaluation phase and send them to the verifier in a streaming manner. Once the verifier receives the respective symbols associated to challenged columns of codewords, they combine them to create the symbols of those codewords and perform the check.

In essence, we change the granularity of what we commit to. We find that if we can commit to “finer” objects of $\hat{\mathcal{Y}}$ and recall their distribution in $\hat{\mathcal{Y}}$, then we can a streamable small space prover for a polynomial commitment scheme given row-wise access to our oracle.

Proof of Theorem 5.2. We begin with the efficiency analysis of Com.

Com and efficiency analysis. We have that:

- N queries of $\mathcal{O}_R$. On line (4) in Algorithm 7 we must query A for each symbol in the witness matrix and hold the d non-zero elements in memory for that symbol.
- 1 pass over $\mathcal{Y}$. We commit to codeword fragments in order to pass over the input tape exactly once, we never have to re-stream since we are not building our Merkle tree in column major order as in Algorithm 3.
- Nd prover time. For $\sqrt{N}$ symbols per message, and for $\sqrt{N}$ messages we perform d field multiplications on lines (2-14).
- $\log(Nd)$ prover space. The height of the stack in Algorithm 7 is logarithmic in the number of leaves of the Merkle tree.

Eval and efficiency analysis. We describe the variant of Eval found in Algorithm 1 of Section 3.2, satisfying the guarantees provided in Theorem 5.2.

- On line (9), P computes $\mathbf{y}'$ in column major order similar to the computation in Eval of Com – Space. We acknowledge that both here, and in the space efficient setting for Reed Solomon commitments, this requires us to multipass over the input sequence, but the number of field multiplications we perform stays $O(N)$.
- On line (11), we re-run the commitment, and emit Merkle proofs of membership to the verifier in the same fashion as Eval – Space. Note that in this setting, the verifier will have to perform $\sqrt{N}$ field additions in order to “retrieve” the codeword symbols from the partial commitments. However, it is important to note that field additions are far less intensive computationally than field multiplications, and these do not affect our verifier asymptotically.
- $\mathbf{q}_1$ is also computed and chosen in the same way as in previous sections.

This concludes the discussion of our efficiency analysis satisfying the claims in Theorem 5.2. □

$$\begin{aligned}
\hat{\mathcal{Y}} &= \begin{bmatrix} \mathcal{Y}(1,1) & \mathcal{Y}(1,2) \\ \mathcal{Y}(2,1) & \mathcal{Y}(2,2) \end{bmatrix} \times \begin{pmatrix} \begin{bmatrix} A(1,1) & 0 & A(1,3) \\ 0 & A(2,2) & A(2,3) \end{bmatrix} \end{pmatrix} \\
&= \begin{bmatrix} \mathcal{Y}(1,1) \times A(1,1) & 0 & \mathcal{Y}(1,1) \times A(1,3) \\ 0 & \mathcal{Y}(1,2) \times A(2,2) & \mathcal{Y}(1,2) \times A(2,3) \\ \mathcal{Y}(2,1) \times A(1,1) & 0 & \mathcal{Y}(2,1) \times A(1,3) \\ 0 & \mathcal{Y}(2,2) \times A(2,2) & \mathcal{Y}(2,2) \times A(2,3) \end{bmatrix}
\end{aligned}$$

Figure 2: We commit to field products of the form $\mathcal{Y}[i, j] \times A[j, k]$ in row major order. Visually, we sum columns of the same shades to retrieve full codeword symbols. This construction enables $O(\log N)$ -space streaming Com with only one pass, adapting expander codes ($xA = c$) to SNARKs in the random access machine model.

Security. The security analysis follows similarly to that in Section 5.2. The multilinear polynomial commitment scheme with the row-wise oracle from codeword fragments requires a brief argument for the aggregation of fragments into codeword symbols so that the security analysis in [40] follows. We provide a brief proof sketch here for this argument.

Proof Sketch. Note that a single symbol in the typical encoded matrix has $\hat{\mathcal{Y}}[i, j] = \sum_{k=1}^m \mathcal{Y}[i, k] \mathbf{A}[k, j]$. Then it is apparent that summing the first m terms of any column $j \in [rm]$ of $\hat{\mathcal{Y}}$ yields a full codeword symbol $\hat{\mathcal{Y}}[i, j], i \in [m], j \in [rm]$. Therefore, after performing these summations over all rows of our codeword fragment matrix we subsequently arrive at the full codeword matrix. Since the underlying oracle realizes a linear code that is constant-rate with linear minimum distance, these codes when checked against the verifier’s challenges (namely, in the checks against specific columns of the encoded $\mathbf{y}'$), still have that if our verifier is honest we are close to a codeword, and conversely if our verifier is dishonest, the linear combination on the right side of the equation in Algorithm 2 on line (7) is far from the valid codeword. Therefore, security holds. $\square$

5.4 Streamable MLPCS with Logarithmic Proof Sizes in the $\mathcal{O}_R$ -model

We now shift to the logarithmic space MLPCS previously explored in Section 4 to demonstrate that we may achieve similar prover efficiencies to those presented in Section 5.3 and Section 5.6 while reducing our proof size from $O(\sqrt{N})$ to just $\tilde{O}(1)\text{ML}$. We begin with a presentation of our streamable MLPCS with a time $N^{1.5}$ and space $\sqrt{T}$ prover satisfying the guarantees of the following theorem using the previously established technique of the row-wise oracle encoder model found in Section 5.1. We state this formally through the following theorem.

Theorem 5.4. *Assume the existence of a collision resistant hash function. Then there exists a streamable polynomial commitment scheme (Setup, Com, Eval) for any security parameter λ and input tape of length $N = 2^n$ for multilinear polynomials with the following guarantees:*

- Com outputs a commitment of size $O_\lambda(1)$; and
- the running time of Com is $\tilde{O}(N^{1.5})$, takes $O(N)$ hashes, $\sqrt{N}$ passes over $\mathcal{Y}$, and uses space $O(\sqrt{N})$.
- The committer of Eval runs in time $O(N \log N)$ and performs $O(N)$ hashes, uses $\sqrt{N}$ passes over $\mathcal{Y}$, and uses space $O(\sqrt{N})$.
- The receiver of Eval’s running time is $\tilde{O}(\ell)$, $\ell \in \Theta(\lambda)$, uses space $\tilde{O}(1)$, and the size of the proof is $\tilde{O}(1)$.

Algorithm 7 $\text{Com}(pp; \mathcal{Y})$ - Small Space Row Oracle

Input: $A \in \mathcal{M}_{m,rm,d}, \mathcal{Y} \in \mathbb{F}^{n \times m=N}$

Output: A Merkle root hash h

Require: $n \geq m$

```
1:  $S \leftarrow \text{init}$ 
2: for  $i \in [n]$  do
3:   for  $j \in [m]$  do
4:     for  $k \in \mathcal{O}_R.\text{Query}(j)$  do
5:        $h \leftarrow \mathcal{Y}[i, j] \times A[j, k]$ 
6:        $\text{node} \leftarrow (h, 0)$ 
7:       while  $S.\text{top}.\text{height}$  is  $\text{node}.\text{height}$  do
8:          $\text{sib} \leftarrow S.\text{pop}$ 
9:          $\text{node} \leftarrow ((\text{sib}.h \circ \text{node}.h), \text{node}.\text{height} + 1)$ 
10:      end while
11:       $S.\text{push}(\text{node})$ 
12:    end for
13:  end for
14: end for
15: while  $S.\text{size}() > 1$  do
16:    $\text{right} \leftarrow S.\text{pop}()$ 
17:    $\text{left} \leftarrow S.\text{pop}()$ 
18:    $S.\text{push}((\text{left}.h \circ \text{right}.h), \text{right}.\text{height} + 1)$ 
19: end while
20: return  $S.\text{pop}()$ 
```

We present the proof of [Theorem 5.4](#) similarly to the previous proof of [Theorem 4.5](#), but note that there are several distinct differences in complexity accounting affected by the introduction of the row-sparse encoder with c non-zero entries per row. We offer all analysis with respect to [Algorithm 5](#).

Proof of [Theorem 5.4](#). We begin with an analysis of the commitment, and then move to the analysis of the evaluation.

Com and efficiency analysis. We have:

- $\sqrt{N}$ passes over $\mathcal{Y}$, as generating $\sqrt{N}$ elements in $\hat{\mathcal{Y}}$ on lines (3-5) requires a pass over the witness.
- Nc operations over $\mathbb{F}$ originating from performing c field multiplications per symbol in the input message on line (4), as each symbol in the input message is multiplied by exactly its corresponding row in the encoder matrix which has row sparsity c .
- $2N - 1$ hashes from building our Merkle tree on line (5).
- $\sqrt{N}$ space is achieved through the same arguments presented in [Theorem 4.5](#), where we compute $\sqrt{N}$ many symbols of $\hat{\mathcal{Y}}$ in parallel with one another by storing $\sqrt{N}$ many buckets for each symbol.
- We use $N^{1.5}$ many queries to the oracle $\mathcal{O}_R$, as each row is retrieved per every N input symbols and we pass over the N input symbols $\sqrt{N}$ times.

We now move to our analysis of Eval with the row-wise oracle. It proceeds as follows:

Eval and efficiency analysis. We have that

- $O(1)$ passes over $\hat{\mathcal{Y}}$, as generating the folded codewords at levels $i \in [\log N - 1]$ on lines (8-19) takes one pass over the encoded witness, and one another pass to answer the verifier's challenges on lines (22-27).
- Time $N + N \log N$ operations over $\mathbb{F}$ originating from i) performing the folding operation on lines (14-17) and ii) performing the LSB sum-check.
- The analysis of the number of hashes is unchanged as to that described in [Theorem 4.3](#).
- $\sqrt{N}$ space is achieved primarily from the same techniques in [Theorem 4.5](#).

Invoking [Theorem 4.2](#) extends the complexity of the receiver to the stated claims in [Theorem 5.4](#). $\square$

We now briefly present the dual mode of our analysis of the oracle model of encoding and consider the column-wise oracle model.

5.5 Streamable Quasi-linear Time, Square-Root Space MLPCS in the $\mathcal{O}_C$ Model

Here, we present our MLPCS with $O(\sqrt{N})$ sized proofs which admits a $O(N \log N)$ time and $O(\sqrt{N})$ space prover from the column-wise oracle $\mathcal{O}_C$. We begin with our main result of this section:

Theorem 5.5. *Assume the existence of a collision resistant hash function. Then for security parameter λ and input sequence $\mathcal{Y}$ of length $N = 2^n$, there exists a streamable polynomial commitment scheme (Setup, Com, Eval) for any security parameter λ and input tape of length $N = 2^n$ for multilinear polynomials with the following guarantees:*

- Com outputs a commitment of size $O_\lambda(1)$; and
- the running time of Com is Nd hashes and operations over $\mathbb{F}$, 1 pass over $\mathcal{Y}$, uses space $3r\sqrt{N}$, and uses N calls to $\mathcal{O}_C$.
- The committer of Eval runs in $Nd + N$ hashes and operations over $\mathbb{F}$, uses 1 pass over $\mathcal{Y}$, requires space $3r\sqrt{N}$, and requires N calls to $\mathcal{O}_C$.
- The receiver of Eval's running time is $O(\lambda\sqrt{N})$ and the size of the proof is $O(\lambda\sqrt{N} + \lambda \log(N))$.

Overview of the Commitment. We briefly describe the overview of the commitment scheme, and note that it is very similar to the row-wise commitment scheme presented in [Section 5.2](#). We modify Com from [Algorithm 6](#) in the following ways:

- Instead of querying $\mathcal{O}_R$ for rows of $\mathbf{A}$, computing partial codewords to $\hat{\mathcal{Y}}$, and aggregating them into full codewords, we query $\mathcal{O}_C$ for columns and build entire codeword symbols at a time.
- Then, we traverse across the matrix $\mathbf{A}$ in column-major-order by querying $\mathcal{O}_C$, and build symbols sequentially in codewords.
- In this way, we create codewords similar to [Algorithm 4](#) in that we create them row-by-row and employ the MerkleDamgård hashing technique to finally commit to the Merkle tree.

We present [Algorithm 8](#) which reflects these changes, as well as discuss the efficiency of it and the modified Eval from [Algorithm 1](#).

[Theorem 5.5.](#) We now give the efficiency analysis of the linear time commitment in [Algorithm 8](#):

Algorithm 8 $\text{Com}(pp; \mathcal{Y})$ - Linear Col Oracle

Input: $\mathcal{Y} \in \mathbb{F}^{n \times m=N}$

Output: A Merkle root hash h

Require: $n \geq m$

```
1:  $\mathbf{h} \leftarrow$  empty list
2:  $\mathbf{c} \leftarrow$  empty list
3: for  $i \in [n]$  do
4:   for  $j \in [rm]$  do
5:      $c(j) \leftarrow \langle \mathcal{Y}[i, *], \mathcal{O}_C.\text{Query}(j) \rangle$ 
6:   end for
7:    $\mathbf{h} \leftarrow \text{MerkleDamgard}(\mathbf{h}(j), \mathbf{c}(j)) \forall j \in [rm]$ 
8: end for
9: return  $\text{M.Commit}(\mathbf{h})$ 
```

Com and efficiency analysis. We have:

- On lines (3-6) we query the oracle N times, and for each query we perform an inner product consisting of the number of field multiplications within the set returned by $\mathcal{O}.\text{Query}(j)$. For any given j , this is only d , so we achieve a time of Nd .
- The use of the two lists $\mathbf{c}$ and $\mathbf{h}$ on lines (5, 7) incur a space cost of $2r\sqrt{N}$.
- This space cost allows us to achieve only 1 pass over $\mathcal{Y}$, we never have to revisit rows because we perform all computations associated with that row in order.

Eval and efficiency analysis. We provide analysis which allows [Algorithm 8](#) to interact with [Algorithm 1](#) and achieve the results stated in [Theorem 5.5](#). Since the steps taken in Eval rely less on the access pattern to $\mathcal{O}$, we adopt much of the same techniques as in [Section 5.2](#).

- On line (9), we may use the same aggregator technique as in [Section 3.3](#) and [Section 5.2](#) to ensure that we only have to perform one pass over $\mathcal{Y}$ to create $\mathbf{y}'$.
- On line (11), we rerun Com and emit Merkle proofs as in [Section 3.3](#) and [Section 5.2](#), and note that this adds no asymptotic increases than Com as described above.
- Finally, the challenge vector $\mathbf{q}_1$ is computed in the natural way and the verifier upon needing to encode the random linear combination $\mathbf{y}'$ simply calls Enc related to $\mathcal{O}_C$'s underlying encoder matrix.

Security. This protocol admits its multilinear polynomial commitment scheme from $\mathcal{O}_C$ which realizes a constant-rate, linear distance linear code. Due to the results of [\[40\]](#), we then retain soundness by the distance and rate of our encoder. $\square$

5.6 Logarithmic Space MLPCS in the $\mathcal{O}_C$ Model

We present our streamable MLPCS with a time $O(N^{1.5})$ and space $O(\log N)$ prover satisfying the guarantees of the following theorem using the column-wise encoder model.

Theorem 5.6. *Assume the existence of a collision resistant hash function, then there exists a streamable polynomial commitment scheme (Setup, Com, Eval) for any security parameter λ and input tape of length $N = 2^n$ for multilinear polynomials with the following guarantees::*

- Com outputs a commitment of size $O_\lambda(1)$; and

- the running time of Com is Nd hashes and operations over $\mathbb{F}$, $\sqrt{N}$ passes over $\mathcal{Y}$, and uses space $\log(r\sqrt{N})$.
- The committer of Eval runs in $Nd + N$ hashes and operations over $\mathbb{F}$, uses $\sqrt{N}$ passes over $\mathcal{Y}$, and requires space $\log(\sqrt{N})$.
- The receiver of Eval 's running time is $O(\lambda\sqrt{N})$ and the size of the proof is $O(\lambda\sqrt{N} + \lambda \log N)$.

Overview of the Commitment. As in [Section 3.2](#), the admission of a space-efficient PCS from column-wise access to the encoder matrix A is natural as the inner product between a row of $\mathcal{Y}$, namely $\mathcal{Y}[i, *]$ and a column of A , $A[:, j]$ results in a symbol of a codeword being produced. We exploit this here as in [Algorithm 3](#) and produce codeword symbols in column major order throughout $\hat{\mathcal{Y}}$, applying the MerkleDamgård hashing technique to columns of the encoded matrix which allows us to build a Merkle tree with only $r\sqrt{N}$ leaves.

Proof of [Theorem 5.6](#). We begin by presenting the following modifications to [Algorithm 3](#) and argue its efficiency.

Com and efficiency analysis. We have:

- On line (4), we simply observe that we may store the entire d elements of the column $A[:, j]$ in memory while we perform the computation of the inner product. This immediately reduces the number of field operations we perform from $N\sqrt{N}$ to Nd .
- The MerkleDamgård hash technique on line (5) allows us to only maintain one hash element in the prover's memory despite having to compute $\sqrt{N}$ codewords per column of $\hat{\mathcal{Y}}$. This means that our prover space is bounded only by the size of the Merkle tree as we build it, which remains $\log \sqrt{N}$ as we build it due to the stack implementation.

Eval and efficiency analysis. Next, we show that we can have the prover run Eval in a way that preserves the time and space properties stated in [Theorem 5.6](#).

- On line (9) the prover computes the symbols of $\mathbf{y}'$ one at a time using $\sqrt{N}$ passes over the input.
- On line (11), we sort the challenge indices and rerun the computation of Com , emitting Merkle hashes and symbols of codewords when they are relevant to the verifier's challenges as in [Section 3.2](#).
- Since the verifier's running time and the proof size is dominated by receiving the k columns of $\hat{\mathcal{Y}}$ (due to the change in what our commitments are to - hashes - not codeword symbols), we see a reduction in proof size back to $k\sqrt{N} + k \log N$.
- Here we also follow the standard technique of computing $\mathbf{q}_1$ on-the-fly and in small space.

Security. This protocol admits its MLPCS from $\mathcal{O}_C$ which realizes a constant-rate, linear minimum distance linear code. Due to the results of [\[40\]](#), we then retain soundness by the distance and rate of our encoder. $\square$

5.7 Streamable MLPCS with Polylogarithmic Proof Sizes in the $\mathcal{O}_C$ -model

In this section we briefly present our streamable MLPCS with a time $N^{1.5}$ and space $\sqrt{N}$ prover satisfying the guarantees of the following theorem using the previously established technique of the column-wise oracle encoder model found in [Section 5.1](#). The techniques for this model are equivalent to the aforementioned row-wise model, so we present the theorem statement and a short proof sketch.

Theorem 5.7. *Assume the existence of a collision resistant hash function. Then there exists a streamable polynomial commitment scheme (Setup, Com, Eval) for any security parameter λ and input tape of length $N = 2^n$ for multilinear polynomials with the following guarantees:*

- Com outputs a commitment of size $O_\lambda(1)$; and

Prover Space	Brakedown-Style					BaseFold-Style		
	SW $\log N$	SW $\sqrt{N}$	$\mathcal{O}_R, \mathcal{O}_C$ $\sqrt{N}$	$\mathcal{O}_R$ $\log N$	$\mathcal{O}_C$ $\log N$	Gen. Code $\log N$	BF $\sqrt{N}$	$\mathcal{O}_R, \mathcal{O}_C$ $\sqrt{N}$
Com								
$\mathcal{O}$ Queries	—	—	$\sqrt{N}$	N	N	—	—	N
$\mathcal{Y}$ Passes	$\sqrt{N}$	$O(1)$	$O(1)$	$O(1)$	$\sqrt{N}$	t	$\sqrt{N}$	$\sqrt{N}$
$\mathbb{F}$ -ops	$N^{1.5}$	$\tilde{O}(N)$	N	N	N	T_{Enc}	$N^{1.5}$	N
S_P	$\log N$	$\sqrt{N}$	$\sqrt{N}$	$\log N$	$\log N$	$\log N + S_{\text{Enc}}$	$\sqrt{N}$	$\sqrt{N}$
Testing/Eval								
$\mathcal{O}$ Queries	—	—	$\sqrt{N}$	N	N	—	—	$N^{1.5}$
$\mathcal{Y}$ Passes	$\sqrt{N}$	$\sqrt{N}$	$\sqrt{N}$	$\sqrt{N}$	$\sqrt{N}$	—	—	—
$\mathbb{F}$ -ops	$N^{1.5}$	$\tilde{O}(N)$	N	N	N	$\tilde{O}(N)$	$\tilde{O}(N)$	$\tilde{O}(N)$
S_P	$\log N$	$\sqrt{N}$	$\sqrt{N}$	$\log N$	$\log N$	$\log N$	$\sqrt{N}$	$\sqrt{N}$
Total Committer Time (across Com and Testing/Eval)								
$\mathcal{O}$ Queries	—	—	$\sqrt{N}$	N	N	—	—	N
$\mathcal{Y}$ Passes	$\sqrt{N}$	$O(1)$	$O(1)$	$\sqrt{N}$	$\sqrt{N}$	t	$\sqrt{N}$	$\sqrt{N}$
$\mathbb{F}$ -ops	$N^{1.5}$	$\tilde{O}(N)$	N	N	N	$T_{\text{Enc}} + \tilde{O}(N)$	$N^{1.5}$	$\tilde{O}(N)$
S_P	$\log N$	$\sqrt{N}$	$\sqrt{N}$	$\log N$	$\log N$	$\log N + S_{\text{Enc}}$	$\sqrt{N}$	$\sqrt{N}$

Table 2: Overview of our efficiency metrics achieved by our PCS. Columns titled with SW are for the Shockwave code, columns titled with BF are for the BaseFold code. Except where included for clarity, complexities are intentionally stated without their asymptotic notation for readability. Also, note that for the **Testing/Eval** phases in the BaseFold-style results, we intentionally do not remark on the number of passes over $\mathcal{Y}$ consistent with the results in [Section 4.3](#), [Section 4.4](#), [Section 5.4](#), and [Section 5.7](#).

- the running time of Com is $\tilde{O}(N^{1.5})$, takes $O(N)$ hashes, $\sqrt{N}$ passes over $\mathcal{Y}$, and uses space $O(\sqrt{N})$.
- The committer of Eval runs in time $O(N \log N)$, takes $O(N)$ hashes, uses $\sqrt{N}$ passes over $\mathcal{Y}$, and requires space $O(\sqrt{N})$.
- The receiver of Eval's running time is $\tilde{O}(\ell)$, $\ell \in \Theta(\lambda)$, uses space $\tilde{O}(1)$, and the size of the proof is $\tilde{O}(1)$.

One can extend the techniques used to prove [Theorem 5.4](#) towards the proof of [Theorem 5.7](#) by noting that the number of non zeroes throughout the entire encoding matrix is unchanged, each element $\mathcal{Y}[i, j]$ is only multiplied by at most c elements during the encoding process, and therefore we have that [Theorem 5.7](#) holds trivially. It is worth noting here that the number of queries to the oracle $\mathcal{O}_C$ is unchanged from $O(N^{1.5})$. To see why this is the case despite our technique for the proof of [Theorem 5.4](#) fundamentally relying upon accessing entries of the rows of the generator matrix, note that we store the first $\sqrt{N}$ elements in random access memory. When we query the first column of the generator matrix, we can perform the first $\sqrt{N}$ summations towards the first codeword symbol, and thus do not have to query the oracle for the column when subsequently accessing the next $\sqrt{N} - 1$ rows. This results in only needing to query the oracle, in one pass over the witness, $\sqrt{N}$ times per unique codeword symbol we generate and $\sqrt{N}$ many times over the rows we sum over while generating the exact $\sqrt{N}$ symbols. This gives $N^{1.5}$ queries to $\mathcal{O}_C$ since we perform $\sqrt{N}$ passes over the witness.

To summarize our results, we present [Table 2](#), which details the asymptotic complexities of each of our MLPCS.

6 From Commitments to Succinct Zero-Knowledge Arguments

We show how to take the streamable multilinear polynomial commitment schemes described previously and construct two time- and space-efficient succinct arguments in [Section 6.2](#) and [Section 6.3](#).

We will proceed by compiling a polynomial interactive oracle proof (PIOP) with the presented polynomial commitment. To do so, we employ a more restrictive PIOP called an interactive probabilistically checkable proof (IPCP), where the prover will send a long message and the prover and verifier interact as in a public-coin interactive proof. In particular, we show how to make this PIOP streamable in order to obtain time- and space-efficient (post-quantum) arguments.

High-level overview. We adapt the time- and space-efficient IPCP from [\[12,13\]](#). In this IPCP, the verifier sends the entirety of the multilinear extension of the input and the prover responds with an oracle that is the multilinear extension of the gate values of our RAM-to-circuit reduction. We compile this IPCP into a time- and space-efficient argument by instead simply sending our commitment to the multilinear extension rather than the extension itself.

In total, we show that there exists a IPCP for RAM programs from [\[12\]](#). We then use the results of [\[13\]](#) to show that this IPCP can be rendered a streamable IMPCP, that is, a IPCP which has its message encoded by the multilinear extension over $\mathbb{F}$. We then take again the result of [\[13\]](#) to show that given our streamable PCS and IMPCP we have a time- and space-efficient argument system, recalling that [Section 2.3](#) gives the definition of an interactive multilinear PCP.

6.1 Arguments from IMPCPs and Polynomial Commitments

There is a well known way to compile a PIOP into an interactive argument using a polynomial commitment scheme [\[13,23,25\]](#). We then follow the technique of [\[13\]](#) and show that since the IMPCP and polynomial commitments are streamable, then the resulting argument scheme has a time- and space-efficient prover.

Theorem 6.1. *Let $\mathbb{F}$ a finite field and suppose the time T and space S RAM program $\mathcal{R}$ has an IMPCP over $\mathbb{F}$ and that there exists a polynomial commitment scheme also over $\mathbb{F}$ where:*

- *IMPCP Parameters:* round complexity r_{IMPCP} , query complexity q_{IMPCP} , prover running time T_{IMPCP} , prover space S_{IMPCP} and verifier time V_{IMPCP} ,
- *Commitment Parameters:* k_{Com} bound the number of stream passes in Com and Eval; T_{Com} and S_{Com} bound the committer's time and space in Com and Eval; r_{Com} is the round complexity; V_{Com} bounds the verifier's time in Eval; c_{com} bounds the commitment length and the communication in Eval; Q_A bounds the number of committer oracle queries across Com and Eval; and T_A, S_A bound the time and space per code-oracle query.

Then, $\mathcal{R}$ has a public coin zero-knowledge argument system with the following properties:

- *Round complexity:* $1 + r_{\text{IMPCP}} + r_{\text{Com}}$
- *Communication complexity:* $c_{\text{IMPCP}} + (q_{\text{IMPCP}} + 1) \cdot c_{\text{Com}}$
- *Prover runtime:* $(q_{\text{IMPCP}} + 1) \cdot (k_{\text{Com}} \cdot T_{\text{IMPCP}} + T_{\text{Com}} + Q_A T_A)$
- *Prover space usage:* $S_{\text{IMPCP}} + q_{\text{IMPCP}} \cdot (S_{\text{Com}} + S + S_A)$
- *Verifier runtime:* $V_{\text{IMPCP}} + q_{\text{IMPCP}} \cdot V_{\text{Com}}$

[Theorem 6.1](#) follows immediately from [\[13\]](#) other than the bound on the oracle queries and oracle time. We give a brief proof sketch for completeness.

Proof Sketch. The parties emulate the IMPCP and the prover commits to the polynomial via a polynomial commitment. When the verifier queries the prover, the prover responds with values and proofs of consistency. Observe that since the prover queries the oracle Q_A times during the entire commitment, we simply add $Q_A T_A$ as an additive factor into the quantity multiplied by the query complexity term of the IMPCP. $\square$

6.2 Our Logarithmic Space Arguments for RAM

We invoke [Theorem 6.1](#) in order to gain public-coin interactive arguments via the compilation of the streamable multilinear polynomial commitment schemes stated in [Theorem 3.1](#), [Theorem 5.2](#), and [Theorem 5.6](#). We also gain a public-coin interactive argument via the compilation of the streamable polynomial commitment scheme stated in [Theorem 4.3](#).

Theorem 6.2. *Assume the existence of a collision resistant hash function. Then there exists a public-coin interactive zero-knowledge argument system for any $\mathbf{NP}$ relation verifiable by a time T and space S random access machine with the following complexities: the protocol has perfect completeness and negligible soundness error, has $O(\log T)$ rounds with communication complexity $\tilde{O}(\sqrt{T})$. The prover runs in time $\tilde{O}(T^{1.5})$ and space $\tilde{O}(S)$ and the verifier runs in time $\tilde{O}(\sqrt{T})$.*

Proof of Theorem 6.2. Observe that as shown in [Table 2](#) for any of our Brakedown-style logarithmic space polynomial commitment schemes we achieve prover run time $O(N\sqrt{N})$. By the RAM to circuit reduction of [12] we see that a circuit of time T may be encoded in time $N = \tilde{O}(T)$ and space $\tilde{O}(S)$, and as such a pass over our input requires us a time cost of $\tilde{O}(T)$. Then, from [Theorem 3.1](#) we get a total time of $\tilde{O}(T^{1.5})$ because the prover performs N operations in step with a pass over the input, which it does $\sqrt{N}$ times. Then, through [Theorems 5.2](#) and [5.6](#), we get $\tilde{O}(T^{1.5})$ despite our $O(N)$ $\mathbb{F}$ -ops due to the fact that we are only forced to multi-pass over the input when we answer the verifier's challenge vector and construct the random linear combination of rows $\mathbf{y}'$ which needs $\sqrt{N}$ passes over the input and $\sqrt{N}$ $\mathbb{F}$ -ops, yielding $\tilde{O}(T^{1.5})$ prover time. Finally, we see space $\tilde{O}(S)$ due to the fact that $S_{\text{Com}} \geq \log T$, V_{Com} is $\sqrt{N} \in \tilde{O}(\sqrt{N})$, the proof size is also dominated by $c_{\text{Com}} = \sqrt{N} \in \tilde{O}(\sqrt{N})$, and we inherit the number of rounds from the underlying IMPCP. $\square$

Theorem 6.3. *Assume the existence of a collision resistant hash function and a foldable linear encoder $\text{Enc}^{\text{Stream}}$ which runs in time T_{Enc} , takes t passes over the input, and requires space S_{Enc} while outputting codeword elements sequentially. Then there exists a public-coin interactive zero-knowledge argument system for any $\mathbf{NP}$ relation verifiable by a time T and space S random access machine with the following complexities: the protocol has perfect completeness and negligible soundness error, has $O(\log T)$ rounds with communication complexity $\tilde{O}(1)$. The prover runs in time $\tilde{O}(T_{\text{Enc}} + T)$ and space $\tilde{O}(S + S_{\text{Enc}})$ and the verifier runs in time $\tilde{O}(1)$.*

Proof of Theorem 6.3. As shown in [Table 2](#), our BaseFold style logarithmic space polynomial commitment scheme achieves a total prover run time of $T_{\text{Enc}} + N \log N$. By a similar argument to the above and through invoking the RAM to circuit reduction of [12] with the results of [Theorem 4.3](#) we admit a total time of $\tilde{O}(N^{1.5})$ as the prover computes T_{Enc} many field operations over t passes of the input vector. Space $\tilde{O}(S + S_{\text{Enc}})$ is admitted similar to the above argument in that $S_{\text{Com}} \geq \log T$, V_{Com} is $\log N$, and the space of the encoder which the prover inherits gives $\tilde{O}(S + S_{\text{Enc}})$, and the proof size is dominated by $c_{\text{com}} = \log^2 N$ which admits proof size $\tilde{O}(1)$. Lastly we note we inherit the number of rounds from the underlying IMPCP. $\square$

6.3 Our Square-Root Space Arguments for RAM

We invoke [Theorem 6.1](#) in order to gain public-coin interactive arguments via the compilation of the multilinear polynomial commitment schemes stated in [Theorems 3.2](#), [5.1](#) and [5.5](#). We also do this for the polylogarithmic proof size regime and invoke [Theorem 6.1](#) again in order to gain public-coin interactive arguments via the compilation of the multilinear polynomial commitment schemes stated in [Theorems 4.5](#), [5.4](#) and [5.7](#).

Theorem 6.4. *Assume the existence of a collision resistant hash function. Then there exists a public-coin interactive zero-knowledge argument system for any $\mathbf{NP}$ relation verifiable by a time T and space S random access machine with the following complexities: the protocol has perfect completeness and negligible soundness error, has $O(\log T)$ rounds with communication complexity $\tilde{O}(\sqrt{T})$. The prover runs in time $\tilde{O}(T)$ and space $\tilde{O}(\sqrt{T} + S)$ and the verifier runs in time $\tilde{O}(\sqrt{T})$.*

Proof of Theorem 6.4. The proof follows similarly to the above proof of [Theorem 6.2](#). Here, we only focus the verifier run time, communication complexity, and number of rounds since the PCS from the stated theorems inherit all other

complexities from [Theorem 6.2](#) as shown in [Table 2](#). By same techniques of the above proof, observe that we achieve time $\tilde{O}(T)$ via [Theorems 3.2, 5.1](#) and [5.5](#) by making a single pass over the input, while only having a linear (or quasilinear) number of $\mathbb{F}$ -ops. The space argument follows then due to the fact that in this case $S_{\text{Com}} \gg S$, and we achieve space $\tilde{O}(\sqrt{T} + S)$. $\square$

Theorem 6.5. *Assume the existence of a collision resistant hash function. Then there exists a public-coin interactive zero-knowledge argument system for any **NP** relation verifiable by a time T and space S random access machine with the following complexities: the protocol has perfect completeness and negligible soundness error, has $O(\log T)$ rounds with communication complexity $\tilde{O}(1)$. The prover runs in time $\tilde{O}(T^{1.5})$ and space $\tilde{O}(\sqrt{T} + S)$ and the verifier runs in time $\tilde{O}(1)$.*

Proof of Theorem 6.5. The proof follows similarly to the above of [Theorem 6.3](#). Note that the receiver complexities in the polynomial commitment schemes we compile through invoking [Theorem 4.5](#), [Theorem 5.7](#), and [Theorem 5.4](#) are equivalent to those found in [Theorem 6.3](#), and so the values of V_{Com} and c_{com} remain unaffected. What remains is to argue the prover time and prover space. Observe that in all cases in the compiled polynomial commitment schemes, the prover space is $O(\sqrt{N})$, it follows that as in the proof of [Theorem 6.4](#) we have space $\tilde{O}(\sqrt{T} + S)$. Then, observe that in all cases of the polynomial commitment schemes the prover time is $O(N^{1.5})$ or $O(N)$ with a minimum of $\sqrt{N}$ passes over the witness which admits a time $\tilde{O}(T^{1.5})$ prover. $\square$

Acknowledgments.

AI tools were used in the creation of this document for reformatting certain tables after content was already placed within them and partially generating [Figure 1](#) (Google Gemini). The prompts for generating the tables were given as directions to flip columns and rows, as well as merge cells together. To generate the TikZ picture, a photo was provided and the agent was asked to produce appropriate markdown corresponding to the photo.

References

- [1] Abbaszadeh, K., Hafezi, H., Katz, J., Meiklejohn, S.: Single-server private outsourcing of zk-snarks. In: IEEE Symposium on Security and Privacy, SP 2026, San Francisco, CA, USA, May 18-21, 2026. pp. 3075–3090. IEEE (2026). <https://doi.org/10.1109/SP63933.2026.00194>, <https://doi.org/10.1109/SP63933.2026.00194>
- [2] Akhiani, P., Zhang, Y.: Distance of RAA codes over large finite fields (with applications in zkSNARKs and PCGs). Cryptology ePrint Archive, Paper 2026/524 (2026), <https://eprint.iacr.org/2026/524>
- [3] Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Liger: lightweight sublinear arguments without a trusted setup. Des. Codes Cryptogr. **91**(11), 3379–3424 (2023). <https://doi.org/10.1007/S10623-023-01222-8>, <https://doi.org/10.1007/s10623-023-01222-8>
- [4] Bangalore, L., Bhadauria, R., Hazay, C., Venkitasubramaniam, M.: On black-box constructions of time and space efficient sublinear arguments from symmetric-key primitives. In: Kiltz, E., Vaikuntanathan, V. (eds.) Theory of Cryptography - 20th International Conference, TCC 2022, Chicago, IL, USA, November 7-10, 2022, Proceedings, Part I. pp. 417–446. Lecture Notes in Computer Science, Springer (2022). https://doi.org/10.1007/978-3-031-22318-1_15, https://doi.org/10.1007/978-3-031-22318-1_15
- [5] Baweja, A., Mishra, P., Mopuri, T., Newatia, K., Wang, S.: Scribe: Low-memory snarks via read-write streaming. IACR Cryptol. ePrint Arch. **2024**, 1970 (2024), <https://eprint.iacr.org/2024/1970>, to appear at USENIX Security 2026.
- [6] Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast reed-solomon interactive oracle proofs of proximity. In: Chatzigiannakis, I., Kaklamanis, C., Marx, D., Sannella, D. (eds.) 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, Prague, Czech Republic, July 9-13, 2018. pp. 14:1–14:17. LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018). <https://doi.org/10.4230/LIPIcs.ICALP.2018.14>, <https://doi.org/10.4230/LIPIcs.ICALP.2018.14>
- [7] Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. IACR Cryptol. ePrint Arch. **2018**, 46 (2018), <http://eprint.iacr.org/2018/046>
- [8] Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I. pp. 103–128. Lecture Notes in Computer Science, Springer (2019). https://doi.org/10.1007/978-3-030-17653-2_4, https://doi.org/10.1007/978-3-030-17653-2_4
- [9] Ben-Sasson, E., Goldberg, L., Kopparty, S., Saraf, S.: DEEP-FRI: sampling outside the box improves soundness. In: Vidick, T. (ed.) 11th Innovations in Theoretical Computer Science Conference, ITCS 2020, Seattle, Washington, USA, January 12-14, 2020. pp. 5:1–5:32. LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020). <https://doi.org/10.4230/LIPIcs.ITCS.2020.5>, <https://doi.org/10.4230/LIPIcs.ITCS.2020.5>
- [10] Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for snarks and proof-carrying data. In: Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing. p. 111–120. STOC '13, Association for Computing Machinery, New York, NY, USA (2013). <https://doi.org/10.1145/2488608.2488623>, <https://doi.org/10.1145/2488608.2488623>
- [11] Block, A.R., Fang, Z., Katz, J., Thaler, J., Waldner, H., Zhang, Y.: Field-agnostic snarks from expand-accumulate codes. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part X. pp. 276–307. Lecture Notes in Computer Science, Springer (2024). https://doi.org/10.1007/978-3-031-68403-6_9, https://doi.org/10.1007/978-3-031-68403-6_9

- [12] Block, A.R., Holmgren, J., Rosen, A., Rothblum, R.D., Soni, P.: Public-coin zero-knowledge arguments with (almost) minimal time and space overheads. In: Pass, R., Pietrzak, K. (eds.) *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part II*. pp. 168–197. *Lecture Notes in Computer Science*, Springer (2020). https://doi.org/10.1007/978-3-030-64378-2_7, https://doi.org/10.1007/978-3-030-64378-2_7
- [13] Block, A.R., Holmgren, J., Rosen, A., Rothblum, R.D., Soni, P.: Time- and space-efficient arguments from groups of unknown order. In: Malkin, T., Peikert, C. (eds.) *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part IV*. pp. 123–152. *Lecture Notes in Computer Science*, Springer (2021). https://doi.org/10.1007/978-3-030-84259-8_5, https://doi.org/10.1007/978-3-030-84259-8_5
- [14] Bombar, M., Couteau, G., Couvreur, A., Ducros, C.: Correlated pseudorandomness from the hardness of quasi-abelian decoding. In: Handschuh, H., Lysyanskaya, A. (eds.) *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part IV*. *Lecture Notes in Computer Science*, vol. 14084, pp. 567–601. Springer (2023). https://doi.org/10.1007/978-3-031-38551-3_18, https://doi.org/10.1007/978-3-031-38551-3_18
- [15] Bootle, J., Cerulli, A., Ghadafi, E., Groth, J., Hajiabadi, M., Jakobsen, S.K.: Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In: Takagi, T., Peyrin, T. (eds.) *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part III*. *Lecture Notes in Computer Science*, vol. 10626, pp. 336–365. Springer (2017). https://doi.org/10.1007/978-3-319-70700-6_12, https://doi.org/10.1007/978-3-319-70700-6_12
- [16] Bootle, J., Chiesa, A., Groth, J.: Linear-time arguments with sublinear verification from tensor codes. In: Pass, R., Pietrzak, K. (eds.) *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part II*. *Lecture Notes in Computer Science*, vol. 12551, pp. 19–46. Springer (2020). https://doi.org/10.1007/978-3-030-64378-2_2, https://doi.org/10.1007/978-3-030-64378-2_2
- [17] Bootle, J., Chiesa, A., Hu, Y., Orrù, M.: Gemini: Elastic snarks for diverse environments. In: Dunkelman, O., Dziembowski, S. (eds.) *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part II*. pp. 427–457. *Lecture Notes in Computer Science*, Springer (2022). https://doi.org/10.1007/978-3-031-07085-3_15, https://doi.org/10.1007/978-3-031-07085-3_15
- [18] Bootle, J., Chiesa, A., Liu, S.: Zero-knowledge iops with linear-time prover and polylogarithmic-time verifier. In: Dunkelman, O., Dziembowski, S. (eds.) *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part II*. *Lecture Notes in Computer Science*, vol. 13276, pp. 275–304. Springer (2022). https://doi.org/10.1007/978-3-031-07085-3_10, https://doi.org/10.1007/978-3-031-07085-3_10
- [19] Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Resch, N., Scholl, P.: Correlated pseudorandomness from expand-accumulate codes. In: Dodis, Y., Shrimpton, T. (eds.) *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part II*. *Lecture Notes in Computer Science*, vol. 13508, pp. 603–633. Springer (2022). https://doi.org/10.1007/978-3-031-15979-4_21, https://doi.org/10.1007/978-3-031-15979-4_21
- [20] Brehm, M., Chen, B., Fisch, B., Resch, N., Rothblum, R.D., Zeilberger, H.: Blaze: Fast snarks from interleaved RAA codes. In: Fehr, S., Fouque, P. (eds.) *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part IV*. pp. 123–152. *Lecture Notes in Computer Science*, Springer (2025). https://doi.org/10.1007/978-3-031-91134-7_5, https://doi.org/10.1007/978-3-031-91134-7_5

- [21] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA. pp. 315–334. IEEE Computer Society (2018). <https://doi.org/10.1109/SP.2018.00020>, <https://doi.org/10.1109/SP.2018.00020>
- [22] Bünz, B., Fenzi, G., Rothblum, R.D., Wang, W.: Tensorswitch: Nearly optimal polynomial commitments from tensor codes. *IACR Cryptol. ePrint Arch.* **2025**, 2065 (2025), <https://eprint.iacr.org/2025/2065>
- [23] Bünz, B., Fisch, B., Szepieniec, A.: Transparent snarks from DARK compilers. In: Canteaut, A., Ishai, Y. (eds.) *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I. pp. 677–706. *Lecture Notes in Computer Science*, Springer (2020). https://doi.org/10.1007/978-3-030-45721-1_24, https://doi.org/10.1007/978-3-030-45721-1_24
- [24] Chen, B., Bünz, B., Boneh, D., Zhang, Z.: Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In: Hazay, C., Stam, M. (eds.) *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Lyon, France, April 23-27, 2023, Proceedings, Part II. pp. 499–530. *Lecture Notes in Computer Science*, Springer (2023). https://doi.org/10.1007/978-3-031-30617-4_17, https://doi.org/10.1007/978-3-031-30617-4_17
- [25] Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, P., Ward, N.P.: Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In: Canteaut, A., Ishai, Y. (eds.) *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I. pp. 738–768. *Lecture Notes in Computer Science*, Springer (2020). https://doi.org/10.1007/978-3-030-45721-1_26, https://doi.org/10.1007/978-3-030-45721-1_26
- [26] Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. In: Canteaut, A., Ishai, Y. (eds.) *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I. pp. 769–793. *Lecture Notes in Computer Science*, Springer (2020). https://doi.org/10.1007/978-3-030-45721-1_27, https://doi.org/10.1007/978-3-030-45721-1_27
- [27] Cook, J., Moshkovitz, D.: Explicit time and space efficient encoders exist only with random access. In: Santhanam, R. (ed.) *39th Computational Complexity Conference, CCC 2024*, Ann Arbor, MI, USA, July 22-25, 2024. pp. 5:1–5:54. *LIPICs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2024). <https://doi.org/10.4230/LIPICs.CCC.2024.5>, <https://doi.org/10.4230/LIPICs.CCC.2024.5>
- [28] Cormode, G., Thaler, J., Yi, K.: Verifying computations with streaming interactive proofs. *Proc. VLDB Endow.* **5**(1), 25–36 (2011). <https://doi.org/10.14778/2047485.2047488>, http://www.vldb.org/pvldb/vol5/p025_grahamcormode_vldb2012.pdf
- [29] Damgård, I.: A design principle for hash functions. In: Brassard, G. (ed.) *Advances in Cryptology - CRYPTO '89*, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings. pp. 416–427. *Lecture Notes in Computer Science*, Springer (1989). https://doi.org/10.1007/0-387-34805-0_39, https://doi.org/10.1007/0-387-34805-0_39
- [30] Diamond, B.E., Posen, J.: Proximity testing with logarithmic randomness. *IACR Commun. Cryptol.* **1**(1), 2 (2024). <https://doi.org/10.62056/AKSDKP10>, <https://doi.org/10.62056/AKSDKP10>
- [31] Divsalar, D., Jin, H., McEliece, R.J.: Coding theorems for “turbo-like” codes. In: *Proceedings of the 36th Annual Allerton Conference on Communication, Control, and Computing*. pp. 201–210 (1998)
- [32] Druk, E., Ishai, Y.: Linear-time encodable codes meeting the gilbert-varshamov bound and their cryptographic applications. In: Naor, M. (ed.) *Innovations in Theoretical Computer Science, ITCS'14*, Princeton, NJ, USA, January 12-14, 2014. pp. 169–182. *ACM* (2014). <https://doi.org/10.1145/2554797.2554815>, <https://doi.org/10.1145/2554797.2554815>

- [33] Fan, Y., Lin, L.: Thresholds of random quasi-abelian codes. *IEEE Trans. Inf. Theory* **61**(1), 82–90 (2015). <https://doi.org/10.1109/TIT.2014.2368138>, <https://doi.org/10.1109/TIT.2014.2368138>
- [34] Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) *Advances in Cryptology - CRYPTO '86*, Santa Barbara, California, USA, 1986, Proceedings. pp. 186–194. *Lecture Notes in Computer Science*, Springer (1986). https://doi.org/10.1007/3-540-47721-7_12, https://doi.org/10.1007/3-540-47721-7_12
- [35] Freitag, C., Paneth, O., Pass, R.: Public-coin, complexity-preserving, succinct arguments of knowledge for NP from collision-resistance. In: Joye, M., Leander, G. (eds.) *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Zurich, Switzerland, May 26–30, 2024, Proceedings, Part IV. pp. 112–141. *Lecture Notes in Computer Science*, Springer (2024). https://doi.org/10.1007/978-3-031-58737-5_5, https://doi.org/10.1007/978-3-031-58737-5_5
- [36] Gallager, R.G.: Low-density parity-check codes. *IRE Trans. Inf. Theory* **8**(1), 21–28 (1962). <https://doi.org/10.1109/TIT.1962.1057683>, <https://doi.org/10.1109/TIT.1962.1057683>
- [37] Ganesh, C., Patranabis, S., Singh, N.: Samaritan: Linear-time prover SNARK from new multilinear polynomial commitments. In: Hanaoka, G., Yang, B. (eds.) *Advances in Cryptology - ASIACRYPT 2025 - 31st International Conference on the Theory and Application of Cryptology and Information Security*, Melbourne, VIC, Australia, December 8–12, 2025, Proceedings, Part V. pp. 456–489. *Lecture Notes in Computer Science*, Springer (2025). https://doi.org/10.1007/978-981-95-5116-3_15, https://doi.org/10.1007/978-981-95-5116-3_15
- [38] Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Rabin, T. (ed.) *Advances in Cryptology - CRYPTO 2010*, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15–19, 2010. Proceedings. pp. 465–482. *Lecture Notes in Computer Science*, Springer (2010). https://doi.org/10.1007/978-3-642-14623-7_25, https://doi.org/10.1007/978-3-642-14623-7_25
- [39] Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. *SIAM J. Comput.* **18**(1), 186–208 (1989). <https://doi.org/10.1137/0218012>, <https://doi.org/10.1137/0218012>
- [40] Golovnev, A., Lee, J., Setty, S.T.V., Thaler, J., Wahby, R.S.: Brakedown: Linear-time and field-agnostic snarks for R1CS. In: Handschuh, H., Lysyanskaya, A. (eds.) *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference*, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part II. pp. 193–226. *Lecture Notes in Computer Science*, Springer (2023). https://doi.org/10.1007/978-3-031-38545-2_7, https://doi.org/10.1007/978-3-031-38545-2_7
- [41] Gurkan, K., Novakovic, A., Rothblum, R.D.: Bolt: Faster snarks from sketched codes. In: Heninger, N., Rosulek, M. (eds.) *Advances in Cryptology - CRYPTO 2026 - 46th Annual International Cryptology Conference*, Santa Barbara, CA, USA, August 17–20, 2026, Proceedings, Part IX. *Lecture Notes in Computer Science*, vol. 16808, pp. 396–427. Springer (2026). https://doi.org/10.1007/978-3-032-35424-2_14, https://doi.org/10.1007/978-3-032-35424-2_14
- [42] Holmgren, J., Rothblum, R.D.: Delegating computations with (almost) minimal time and space overhead. *Cryptology ePrint Archive*, Paper 2018/861 (2018), <https://eprint.iacr.org/2018/861>
- [43] Huang, K., Guo, Y., Qu, W., Zhang, J.: Linear-time, constant-depth blind polynomial commitments from generalized RAA codes, with an end-to-end blind SNARK implementation. *Cryptology ePrint Archive*, Paper 2026/487 (2026), <https://eprint.iacr.org/2026/487>
- [44] Inc., L.: Ligetron docs (2026), <https://docs.ligetron.com/>
- [45] Kahale, N., Urbanke, R.: On the minimum distance of parallel and serially concatenated codes. In: *Proceedings of the 1998 IEEE International Symposium on Information Theory*. p. 31. IEEE, Cambridge, MA, USA (Aug 1998). <https://doi.org/10.1109/ISIT.1998.708611>

- [46] Khabbazzian, M.: Linear distance for fixed-row-weight expand-accumulate codes over arbitrary fields. Cryptology ePrint Archive, Paper 2026/1753 (2026), <https://eprint.iacr.org/2026/1753>
- [47] Khabbazzian, M.: Sharp minimum-distance lower tails for RAA codes. Cryptology ePrint Archive, Paper 2026/1075 (2026), <https://eprint.iacr.org/2026/1075>
- [48] Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: Kosaraju, S.R., Fellows, M., Wigderson, A., Ellis, J.A. (eds.) Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada. pp. 723–732. ACM (1992). <https://doi.org/10.1145/129712.129782>, <https://doi.org/10.1145/129712.129782>
- [49] Li, C., Xu, R., Li, Y., Yu, Y., Hu, Y.: EA codes approaching singleton bound (with application to field-agnostic SNARKs). Cryptology ePrint Archive, Paper 2026/1782 (2026), <https://eprint.iacr.org/2026/1782>
- [50] Li, Z., Liu, H., Xing, C., Yao, Y., Yuan, C.: More efficient SNARKs via quasi-abelian codes: Faster, smaller, and field-agnostic. Cryptology ePrint Archive, Paper 2026/939 (2026), <https://eprint.iacr.org/2026/939>
- [51] Li, Z., Liu, H., Xing, C., Yao, Y., Yuan, C.: More efficient SNARKs via quasi-abelian codes: Faster, smaller, and field-agnostic. Cryptology ePrint Archive, Paper 2026/939 (2026), <https://eprint.iacr.org/2026/939>
- [52] Lund, C., Fortnow, L., Karloff, H.J., Nisan, N.: Algebraic methods for interactive proof systems. J. ACM **39**(4), 859–868 (1992). <https://doi.org/10.1145/146585.146605>, <https://doi.org/10.1145/146585.146605>
- [53] Merkle, R.C.: One way hash functions and DES. In: Brassard, G. (ed.) Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings. pp. 428–446. Lecture Notes in Computer Science, Springer (1989). https://doi.org/10.1007/0-387-34805-0_40, https://doi.org/10.1007/0-387-34805-0_40
- [54] Nair, V., Sharma, A., Thankey, B.: Brakingbase - A linear prover, poly-logarithmic verifier, field agnostic polynomial commitment scheme. In: Hanaoka, G., Yang, B. (eds.) Advances in Cryptology - ASIACRYPT 2025 - 31st International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, VIC, Australia, December 8-12, 2025, Proceedings, Part V. pp. 427–455. Lecture Notes in Computer Science, Springer (2025). https://doi.org/10.1007/978-981-95-5116-3_14, https://doi.org/10.1007/978-981-95-5116-3_14
- [55] Pappas, C., Papadopoulos, D.: Sparrow: Space-efficient zk-snark for data-parallel circuits and applications to zero-knowledge decision trees. In: Luo, B., Liao, X., Xu, J., Kirda, E., Lie, D. (eds.) Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024. pp. 3110–3124. ACM (2024). <https://doi.org/10.1145/3658644.3690318>, <https://doi.org/10.1145/3658644.3690318>
- [56] Pappas, C., Papadopoulos, D.: Hobbit: Space-efficient zk-snark with optimal prover time. In: Bauer, L., Pellegrino, G. (eds.) 34th USENIX Security Symposium, USENIX Security 2025, Seattle, WA, USA, August 13-15, 2025. pp. 4917–4936. USENIX Association (2025), <https://www.usenix.org/conference/usenixsecurity25/presentation/pappas>
- [57] Pless, V.: Introduction to the Theory of Error-correcting Codes, vol. 48. John Wiley & Sons (2011)
- [58] Qu, W., Guo, Y., Zhang, J.: Lightning, field-agnostic super-efficient polynomial commitment scheme. Cryptology ePrint Archive, Paper 2026/258 (2026), <https://eprint.iacr.org/2026/258>
- [59] Spielman, D.A.: Linear-time encodable and decodable error-correcting codes. IEEE Trans. Inf. Theory **42**(6), 1723–1731 (1996). <https://doi.org/10.1109/18.556668>, <https://doi.org/10.1109/18.556668>
- [60] Wang, R., Hazay, C., Venkitasubramaniam, M.: Ligetron: Lightweight scalable end-to-end zero-knowledge proofs post-quantum zk-snarks on a browser. In: IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024. pp. 1760–1776. IEEE (2024). <https://doi.org/10.1109/SP54263.2024.00086>, <https://doi.org/10.1109/SP54263.2024.00086>

- [61] Xie, T., Zhang, Y., Song, D.: Orion: Zero knowledge proof with linear prover time. In: Dodis, Y., Shrimpton, T. (eds.) *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV*. pp. 299–328. *Lecture Notes in Computer Science*, Springer (2022). https://doi.org/10.1007/978-3-031-15985-5_11, https://doi.org/10.1007/978-3-031-15985-5_11
- [62] Zeilberger, H., Chen, B., Fisch, B.: Basefold: Efficient field-agnostic polynomial commitment schemes from foldable codes. In: Reyzin, L., Stebila, D. (eds.) *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part X*. *Lecture Notes in Computer Science*, vol. 14929, pp. 138–169. Springer (2024). https://doi.org/10.1007/978-3-031-68403-6_5, https://doi.org/10.1007/978-3-031-68403-6_5
- [63] ZK Rollups: Zk rollups for the crypto blockchain reduces gas fees (2026), <https://zkrollups.io/#s1>, section “Making Transactions Faster”. Accessed: 2026-02-12